MEGA: MACHINE LEARNING-ENHANCED GRAPH ANALYTICS FOR COVID-19 INFODEMIC CONTROL

Ching Nam Hang[*]
cnhang3-c@my.cityu.edu.hk

Pei-Duo Yu[†]
peiduoyu@cycu.edu.tw

Lin Ling[*]
linling2-c@my.cityu.edu.hk

Chee Wei Tan[*]
cheewtan@cityu.edu.hk

[*]Department of Computer Science, City University of Hong Kong, Hong Kong
[†]Department of Applied Mathematics, Chung Yuan Christian University, Taiwan

ABSTRACT

Statistical network analysis plays a critical role in managing the coronavirus disease (COVID-19) infodemic such as addressing community detection and rumor source detection problems in social networks. As the data underlying infodemiology are fundamentally huge graphs and statistical in nature, there are computational challenges to the design of graph algorithms and algorithmic speedup. A framework that leverages cloud computing is key to designing scalable data analytics for infodemic control. This paper proposes the *MEGA* framework, which is a novel joint hierarchical clustering and parallel computing technique that can be used to process a variety of computational tasks in large graphs. Its unique feature lies in using statistical machine learning to exploit the inherent statistics of data to accelerate computation. Our MEGA framework consists of first pruning, followed by hierarchical clustering based on geodesic distance and then parallel computing, lending itself readily to parallel computing software, e.g., MapReduce or Hadoop. In particular, we illustrate how our MEGA framework computes two representative graph problems for infodemic control, namely network motif counting for community detection and network centrality computation for rumor source detection. Interesting special cases of optimal tuning in the MEGA framework are identified based on geodesic distance characterization and random graph model analysis. Finally, we evaluate its performance using cloud software implementation and real-world graph datasets to demonstrate its computational efficiency over existing state of the art.

***K*eywords** COVID-19 · Infodemic · Big graph analytics · Machine learning · Cloud computing

1 Introduction

The rapid global spread of coronavirus disease 2019 (COVID-19) leads to a proliferation of data inventory. At the same time, infodemiology (i.e., the epidemic of misinformation spread over social networks) is acknowledged by the World Health Organization (WHO) as a critical problem for COVID-19 pandemic control [1, 2]. The evolution of an infodemic can be modeled as dynamic processes on large-scale networks, and statistical network analysis is therefore playing an important role in infodemic control. Examples of such control measures are network motif enumeration for community detection [3, 4] or rumor source detection [5, 6] in online social networks. Online social networks such as Facebook that boasts of over 2.7 billion monthly active users may want to identify important communities or count user clusters efficiently. However, as data underlying infodemiology are fundamentally huge graphs and statistical in nature, there are computational challenges to the design of graph algorithms and algorithmic speedup.

Network-structured data are complex in nature, coupling both network topological structure and relational data. For instance, new relationships in online social networks can be discovered by knowing who is connected to whom and how information flows between users. Using graph theory, one can model users and relationships between users (e.g.,

friendships) as vertices and edges respectively that result in huge graphs that need to be processed. The volume of these huge graph data may reach a point that limits a direct use of standard graph algorithms. For example, a well-known algorithmic enabler for the aforementioned infodemic control measures is breadth-first search (BFS). However, running a BFS algorithm for every single vertex of a large graph becomes computationally impractical. Cloud computing can alleviate this data volume challenge to some extent [7, 8, 9]. Cloud computing software framework, e.g., MapReduce and Hadoop, decomposes computation into subproblems that are mapped to parallel computers (i.e., mappers in MapReduce) before combining the solution of the subproblems (i.e., the reducer in MapReduce).

Infodemic control (and, in fact, most data science) problems however have important statistical features that are not easily mapped to a cloud computing framework. Network topology of infodemic data are generated and then evolved statistically. This data have local and global statistical dependence that affect the problem-solving approach and its solution quality [10, 11]. For instance, the aforementioned community detection requires counting network motifs (given subgraphs) in graphs generated by a model whose statistical features can affect the speed of detecting communities. The aforementioned rumor source detection can be modeled as maximum-likelihood estimation problems whose optimal solution depends on graph algorithms whose parameters are related to the data statistics. *In general, the inherent statistics of data influences algorithmic tuning and consequentially computational performance.* Thus, it is important to understand how to exploit statistical features for algorithmic speedup without incurring significant information loss or degraded solution quality.

In this paper, we propose a novel scalable graph analytics framework leveraging large-scale cloud computing, called *MEGA*, to address some of aforementioned issues. A novelty of our MEGA framework is to use statistical machine learning for algorithmic parameter tuning, implicitly exploiting the underlying statistics in the data. MEGA contains three major steps: *pruning*, *hierarchical clustering* and *computing*. In MEGA, we first reduce irrelevant information via pruning, and then group the data with similar properties into the same cluster so as to retain the important relationships among data and minimize information loss via hierarchical clustering. We then develop theoretical results of geodesic distance bounds related to the importance of vertices that enable parallel processing in the hierarchy property for solving a statistical network problem. As a case in point, we illustrate how MEGA can be used for two representative anti-infodemic applications, namely network motif counting for community detection and network centrality computation for rumor source detection. Lastly, as MEGA is quite generic, it can be extended to accelerate other graph algorithms for infodemic control.

Overall, the contributions of the paper are as follows:

- We propose a machine learning-enhanced framework called MEGA that decomposes a large graph into smaller components that are mapped to parallel computing software for computing acceleration. We demonstrate that our MEGA framework can be tuned using statistical machine learning techniques to *learn* the statistics underlying graph data.
- Inspired by Tarjan's breadth-first search (BFS) graph decomposition technique [12, 13], the proposed hierarchical clustering step in MEGA framework builds a geodesic distance-based hierarchy of clusters that yields highly-parallelizable structure for parallel implementation in cloud computing software, e.g., MapReduce.
- We demonstrate its performance over the state of the art for two representative statistical problems of infodemic control by extensive performance evaluations using graph datasets from the public domains (e.g., Stanford SNAP website) and also synthetic random graphs generated from Barabási-Albert and Erdős-Rényi model. For random graphs, we demonstrate that optimal tuning parameters can be achieved.
- For search (BFS) graph decomposition, we also incorporate top-k ranking to reduce the number of BFSs invoked on the graph which significantly improves the performance of the MEGA framework. This provides a fast method to compute network centrality for solving statistical inference problem on large networks [5].

This paper is organized as follows. In Section 2, we generally present the MEGA framework, and outline the network motif (given subgraph) counting and network centrality computation problem for infodemic control. In Section 3, we apply MEGA to solve the network motif counting problem, and analyze its computational complexity. In Section 4, we apply MEGA to find the distance-based rumor center in a large-scale graph, and demonstrate how the new discovery of BFS scalable computation approach reduces the computational complexity of finding a rumor center in a given rumor subgraph. Performance evaluation results can be found in Section 5. We conclude the paper in Section 6.

2 Machine Learning-Enhanced Framework for Scalable Computing

In this section, we introduce the aforementioned MEGA framework and study the network motif counting and network centrality computation problem. We focus on scalable framework design for parallel computation to handle large graphs.

Table 1: Key terms and symbols

Symbol	Definition
G	The original input graph.
G'	The graph obtained after pruning G.
$V(G)$	The set of vertices of G.
$E(G)$	The set of edges of G.
N	The number of vertices in G, i.e., $\|V(G)\|$.
N'	The number of vertices in G', i.e., $\|V(G')\|$.
θ	The threshold used in pruning.
$deg(v)$	The degree of vertex v.
$dist(u, v)$	The distance between vertex u and v.
$N_G(v)$	The vertex set which contains all neighbors of vertex v in G.
$maxC$	The number of clusters in G'.
$P(G)$	The set of pruned vertices of G.
$S(v, G)$	The distance centrality of vertex v in G.
$S_{LR}(v, G')$	The lower bound on the distance centrality of vertex v in G'.
$C_{dist}(G)$	The distance center of G.
$dia(G)$	The graph diameter of G.

Note that we can always add more constraints in each step of the framework so as to specify the network computational task. Table 1 summarizes the key mathematical notations used in this paper.

Let $G = (V(G), E(G))$ be a simple graph. MEGA has three main steps:

Step 1. PRUNING
Decompose the given graph G into connected components and obtain the resultant graph G'. In this step, we set a threshold parameter θ using statistical learning theory, and then remove vertices from G until

$$\forall v \in V(G'), \theta < deg(v). \tag{1}$$

Step 2. HIERARCHICAL CLUSTERING
Cluster vertices in a connected component in a hierarchical manner using the breadth-first search algorithm (BFS) [12, 13] and utilize statistical learning approaches to find an optimal approximation to the root for BFS tree traversal.

Step 3. COMPUTING
Solve a network computational task in parallel using the hierarchy property.

The three main steps in MEGA can be parallelized using techniques like MapReduce and Hadoop when MEGA is applied to practical problems (cf. Section 3 and 4). Fig. 1 gives an overview of MEGA applied to solve the network motif counting and network centrality computation problem.

2.1 Preliminaries of Network Motif Counting

Network motif enumeration in community detection is to identify and count a given subgraph (related to subgraph isomorphism problem in computer science). This paper studies its special case when the subgraph is particularly a triangle [14]. Other subgraphs such as clique can also be counted by MEGA with different verification schemes.

Problem 1 (Counting network motif)**.** Given a simple graph $G = (V(G), E(G))$, if there exists three vertices v_i, v_j and v_k such that (v_i, v_j), (v_i, v_k) and (v_j, v_k) are all in $E(G)$, then we say v_i, v_j, v_k form a subgraph. The goal is to compute exactly the size of the set in G characterized by

$$|\{(v_i, v_j, v_k) | v_i, v_j, v_k \text{ form a subgraph where } i < j < k\}|. \tag{2}$$

Beyond the computational challenge, there are useful applications of network motif counting techniques found in anti-infodemic applications such as web spamming detection [15], community detection [16] and web recommendation

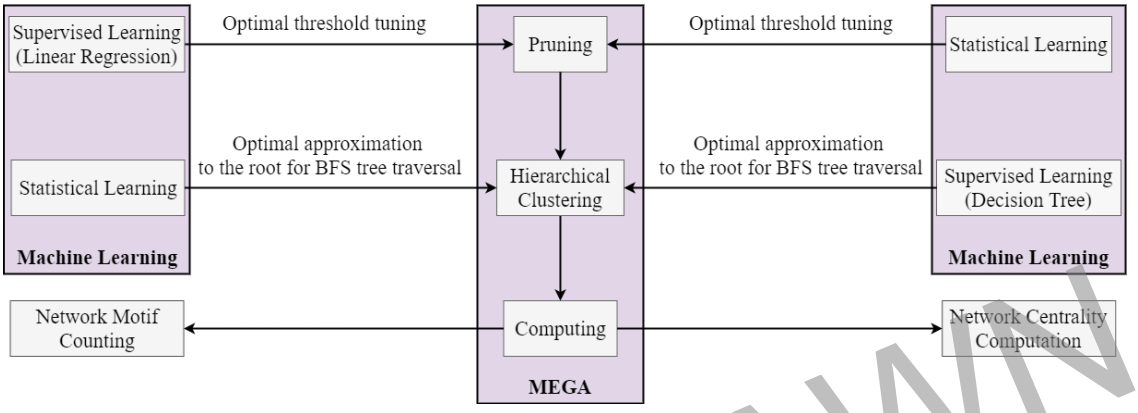


Figure 1: Overview of MEGA applied to solve the network motif counting (left) and network centrality computation problem (right). We use machine learning approaches to perform parameters tuning so as to optimize the performance of MEGA. The threshold θ in the pruning step and the root in the hierarchical clustering step are two critical parameters that can significantly affect the overall performance of MEGA. For network motif counting, we use linear regression to compute an optimal threshold θ^* and find an optimal approximation to the root based on the statistical features underlying graph data. We then use statistical data analysis to obtain θ^* for network centrality computation and apply decision tree algorithm to find an optimal root for MEGA.

[17]. Furthermore, since the given subgraph we considered in this paper is the most basic structural unit in a graph, network motif counting can be adapted to address other graph-theoretic problems such as the maximum clique problem [18] or the k-truss computation problem [19, 20] that are practically relevant to biological network analysis [21] for pandemic control.

We briefly discuss some of the recent works on subgraph counting. For a given graph G with N vertices, a naive method that exhaustively checks every group of three vertices has a computational time complexity $O(N^3)$ since there are $\binom{N}{3}$ groups altogether. This is however impractical for large graphs that often have more than hundreds of millions of vertices. Even if there is an efficient method to list down all subgraphs, this is still time consuming when the graph is dense since there are overwhelmingly many subgraphs need to be stored. There are works that use graph embedding with $O(N)$ subgraphs [22] or approximate counting techniques [23]. To address the scalability issues, some works propose parallel algorithms to compute an exact or approximate solution [24, 25]. Other works employ linear algebraic computational methods that use the adjacency matrix A of a given graph, whereupon the number of subgraphs in the graph can be obtained from the diagonal of A^3 [26, 27].

2.2 Preliminaries of Network Centrality Computation

The measure of network centrality is another classic graph theory problem. It is a fundamental concept in network analytics for the purpose of identifying critically important vertices in a graph. To compute network centrality, it often involves several BFS tree traversals [5], and each has $O(|V(G)|^2)$ or $O(|V(G)| + |E(G)|)$ time complexity [28]. Thus, reducing the number of BFSs invoked on the graph is usually an efficient way to improve the algorithmic performance. This paper aims to identify different network centers accurately and efficiently. In particular, we focus on the *rumor center* of a large graph. Note that MEGA is also able to compute other centrality measures such as betweenness centrality with different constraints.

Problem 2 (Computing network centrality for rumor source detection)**.** Assume that a rumor was spread in an online social network based on the *susceptible-infected* (SI) model. Given a rumor subgraph (all vertices in the subgraph knew the rumor), how to accurately identify the rumor culprit?

We model how a user spreads a rumor over an online social network with the SI model as follows. At the beginning of the spreading, a user $v \in V(G)$ starts to spread a rumor in a social network $G = (V(G), E(G))$ where users and relationships between users are represented by vertices and edges respectively and we assume that G is a simple connected graph. The rumor spreading follows the SI model which is simplified from the well-known *susceptible-infected-recovered* (SIR) model. In the SI model, once a vertex is "infected", it stays in this state forever. Let Nxt denote the set of vertices such that each vertex in Nxt has at least one infected neighbor. In each time slot, one vertex is uniformly chosen from Nxt to be the next infected vertex. Given a snapshot of N infected vertices, the question of interest is to find out which vertex is the rumor monger.

In [5], a network centrality called rumor centrality was proposed to solve this statistical inference problem and the problem was solved optimally when G is a infinite size regular tree. The algorithm in [5] computes the rumor centrality of a tree vertex with time complexity $O(N)$, where N is the size of the infected graph. However, it becomes computationally hard to compute the maximum-likelihood estimator of the rumor source when the infected graph is not a tree [29]. A BFS heuristic algorithm for general graph in [5] was then proposed to resolve this issue but it needs to do BFS tree traversal starting from every vertex in the infected graph. Thus, the total time complexity to find the rumor center becomes $O(N^2)$. The authors in [5] then proved that the *distance center* of the rumor subgraph was shown to be a good approximation to the rumor source. In particular, the distance center is exactly the maximum-likelihood estimator of the rumor source when the rumor subgraph is a degree regular tree [6]. Thus, we focus on utilizing MEGA to find the distance-based rumor source estimator.

Let $G = (V(G), E(G))$ be a simple connected graph. We denote the shortest distance between two vertices, say u and v, by $dist(u, v)$ and the distance centrality of vertex $v \in G$, $S(v, G)$, is defined as

$$S(v, G) = \sum_{w \in V(G)} dist(v, w). \tag{3}$$

The distance center, $C_{dist}(G)$, of G is a set of vertices such that each vertex in the set has the minimum distance centrality, i.e., we have

$$C_{dist}(G) = \{v | S(v, G) = \min_{w \in V(G)} S(w, G)\}. \tag{4}$$

The most straight-forward way (textbook algorithm) to find the distance center is to compute the distance centrality for all vertices in the graph based on BFS tree traversal starting from every vertex, and then pick the one with minimum distance centrality as the distance center. This approach is similar to solving the all-pairs shortest-path problem which has complexity of at least $O(N^3)$, where N is the number of vertices in the graph. Therefore, when the size of graphs increases to billions of vertices, computing distance centrality becomes prohibitively expensive. To overcome such computational issue, some works propose approximate algorithms instead of exact computation [30, 31]. These approximation techniques however require a high-accuracy approximation scheme. Other works enable parallel computing on all-sources BFS that costs a huge memory space [32, 33, 34].

3 Network Motif Counting

In this section, we describe how MEGA solves the network motif counting problem when the subgraph is particularly a triangle (cf. Problem 1).

3.1 Pruning

We remove vertices such that $\theta < deg(u) < N' - 1$ for all $u \in V(G')$, where G' is the pruned graph. When a vertex u is removed from G, then for each pair (a, b), where vertex a and b both belong to $N_G(u)$, we check if $(a, b) \in E(G)$. If $(a, b) \in E(G)$, we count $\{u, a, b\}$ as a subgraph. Hence, there are $\binom{deg(u)}{2}$ pairs need to be checked for each removed vertex u. When a vertex with degree $N' - 1$ is removed, there are $|E(G')| - (|V(G')| - 1)$ subgraphs counted. If $N' < 3$, then MEGA ends here. Otherwise, we go to the next step.

In general, the threshold θ is a tunable parameter and its optimal value depends on the graph topology and the problem we wish to solve. We say θ is optimal if it can largely decompose a given graph and $\theta \ll N$ (cf. Section 3.4). We provide bounds on the optimal threshold θ^* for some special cases based on their inherent structures.

Arbitrary graph The graph properties of an arbitrary graph are all arbitrary that all the settings are non-deterministic.

Lemma 1. For any arbitrary graph $G = (V(G), E(G))$, if we remove all vertices with degree less than or equal to θ to obtain the pruned graph $G' = (V(G'), E(G'))$, we have

$$|V(G')| \leq \frac{2|E(G)|}{\theta + 1}. \tag{5}$$

Proof. According to the *degree sum formula*, the sum of the degrees of all vertices in a finite graph is twice the number of edges in that graph. Since the degree of every vertex in G' must be at least $\theta + 1$, the sum of the degrees of all vertices in G' must be greater than or equal to $\theta + 1$ times the number of vertices in G'. That is,

$$(\theta + 1) \cdot |V(G')| \leq \sum_{v \in V(G')} deg(v) = 2|E(G')|.$$

Since G' is a pruned graph of G, the number of edges in G' must be less than or equal to that in G. Hence, we have

$$|V(G')| \leq \frac{2|E(G')|}{\theta+1} \leq \frac{2|E(G)|}{\theta+1}.$$

□

Barabási-Albert (BA) network The generation of a BA network starts with an initial complete network (a clique) of m_0 vertices. Subsequently, it connects new vertices to the network and each new vertex is connected to m existing vertices, where $m \leq m_0$, with a probability that is proportional to the degrees of the existing vertices. BA network is scale-free, and its degree distribution follows the power law.

Lemma 2. For any BA network G with parameters m_0 and m, where m_0 is the size of the initial complete network and m is the degree of each newly added vertex, the optimal value of the threshold θ^* to efficiently decompose G is m, and the size of the pruned graph must be at most m_0.

Proof. We denote a BA network with parameters m_0 and m as G and the initial complete network of m_0 vertices in G as G_{m_0}, where $m_0 > m$. Then, we have

$$\forall v \in G \setminus G_{m_0}, deg(v) \leq m.$$

If we set $\theta^* = m$, it is obvious that the size of the pruned graph must be equal to m_0 which is the size of G_{m_0}. If $m = m_0 - 1$, then we can completely decompose G. □

Corollary 1. For network motif counting in a BA network G with parameters m_0 and m, if the given subgraph is a triangle, then the optimal threshold θ^* is given by

$$\theta^* = m = \left\lceil \frac{|E(G)|}{|V(G)|} \right\rceil. \tag{6}$$

Erdős-Rényi (ER) random network We use two parameters (N, p) to generate a random network in ER model, where N is the number of vertices and p is the probability of each possible edge to be existed.

Lemma 3. For any ER random network $G(N, p)$, if we set $\theta^* = k \cdot deg_{avg}$, where k is a positive integer and deg_{avg} is the average degree of $G(N, p)$, then the size of the pruned graph decreases exponentially with increasing k.

Proof. Let $G(N, p)$ be an ER random network. We can leverage the fact that the distribution of the degree of any particular vertex is Poisson as the graph size N goes to infinity. Thus, we can use the Chernoff bound [35] to compute the upper bound of the size of the pruned graph. Let v be any vertex in $G(N, p)$ and $\theta^* = kNp$ be the optimal threshold for pruning, where k is a positive integer, then we have

$$\begin{aligned}
\Pr(deg(v) \geq \theta^*) &\leq \frac{(eNp)^{\theta^*} e^{-Np}}{(\theta^*)^{\theta^*}} \\
&= \left(\frac{Np}{\theta^*}\right)^{\theta^*} \cdot e^{\theta^* - Np} \\
&= \left(\frac{1}{k}\right)^{kNp} \cdot e^{(k-1)Np} \\
&< e^{-Np} \cdot \left(\frac{e}{k}\right)^{kNp}.
\end{aligned}$$

Therefore, we can have a simple upper bound for the size of the pruned graph N',

$$N' < e^{-Np} \cdot \left(\frac{e}{k}\right)^{kNp} \cdot N.$$

Note that for an ER random network $G(N, p)$, the value Np is the average degree of the graph which is a constant. Hence, the size of the pruned graph decreases exponentially as k increases. It is worth noting that the key property to prove Lemma 3 is that the probability distribution of a vertex has degree k satisfies the Poisson distribution when the size of the graph is large enough. □

Planar graph A graph is called planar if it can be drawn on the plane in such a way that any point of intersection of two distinct edges is at their endpoints.

Lemma 4. For any planar graph G, the optimal value of the threshold θ^* to completely decompose G is 5.

Proof. Let $G = (V(G), E(G))$ be a planar graph. It is known that the average degree of G is strictly less than 6. It implies that there must be at least one vertex with degree less than or equal to 5 in G. After removing such vertices from G, there is always at least one new vertex to prune since any subset of a planar graph is also a planar graph. Ultimately, G can be completely decomposed. □

3.2 Hierarchical Clustering

We leverage a hierarchical clustering algorithm proposed by [12, 13] to split vertices based on the BFS tree traversal. Note that G' may not be a connected graph even if G was. We denote the ith connected component in G' as G_i. Then, we have $V(G') = \bigcup_{i=1}^{l} V(G_i)$, where l is the number of connected components (i.e., $l = 1$ if G' is a connected graph). Let v_r^i be the root of the BFS tree traversal for the connected component G_i for $i = 1, \ldots, l$. For each G_i, vertices are in the same cluster if their distances to v_r^i are equivalent. Hence, the jth cluster of G_i, $K_{i,j}$, is defined as

$$K_{i,j} = \{v \in V(G') | dist(v, v_r^i) = j\}, \tag{7}$$

where $K_{i,0} = \{v_r^i\}$ and $K_{i,1} = N_{G'}(v_r^i)$.

We call a root an optimal root if it can maximize the computational efficiency of MEGA. In the computing step, there are two schemes to verify whether a subgraph is formed: one based on the number of edges in $K_{i,1}$ (see (8)) and one that checks every two neighboring vertices of any vertex (see (9)). In the view of algorithmic implementation, the former is way more straightforward since we do not need to loop over all the vertices in each cluster. Therefore, we choose the degree center (a vertex with maximum degree) of each connected component as an optimal root. Our intuition is that the degree center maximizes the number of vertices in $K_{i,1}$, and minimizes the average number of vertices in other clusters. Also, degree information is a local property of every vertex that do not need complicated calculations. It can potentially increase the algorithmic efficiency of MEGA.

3.3 Computing

To accelerate the counting process of the remaining subgraphs in G', we propose Lemma 5 to characterize different types of subgraphs based on the hierarchy property.

Lemma 5. For each subgraph $\{a, b, c\}$ in G', either three vertices of $\{a, b, c\}$ are in the same cluster, say $K_{i,j}$, or two of them are in $K_{i,j}$ and the remaining one is in a neighboring cluster $K_{i,j-1}$ or $K_{i,j+1}$.

Proof. First, we know that for each edge $(u, v) \in E(G')$, where $u \in K_{i_1,j_1}$ and $v \in K_{i_2,j_2}$, i_1 must be equal to i_2 and $0 \leq |j_1 - j_2| \leq 1$ based on (7). For a given subgraph $\{a, b, c\}$, since (a, b), (b, c), and (a, c) are edges, if we assume that $a \in K_{i_1,j_1}$, $b \in K_{i_2,j_2}$, and $c \in K_{i_3,j_3}$, then we have

$$\begin{aligned} i_1 = i_2 = i_3, \\ 0 \leq |j_1 - j_2| \leq 1, \\ 0 \leq |j_2 - j_3| \leq 1, \\ 0 \leq |j_1 - j_3| \leq 1. \end{aligned}$$

If $|j_1 - j_2| = |j_2 - j_3| = |j_1 - j_3| = 0$ (i.e., $j_1 = j_2 = j_3$), then all of the three vertices are in the same cluster $K_{i,j}$. Otherwise, two of them are in cluster "j" and the other one is either in cluster "$j - 1$" or cluster "$j + 1$". This implies that $\{a, b, c\}$ must be located in two neighboring clusters. □

The proof of Lemma 5 is based on the fact that for any two clusters, K_{i_1,j_1} and K_{i_2,j_2}, if $i_1 \neq i_2$ or $|j_1 - j_2| \geq 2$, then there is no edge straddling across K_{i_1,j_1} and K_{i_2,j_2}. Therefore, when counting subgraphs in $K_{i,j}$, we only need to consider the vertices in $K_{i,j}$ and its neighboring clusters. Accordingly, for each subgraph $\{v_i, v_j, v_k\}$ in G', there are only two possible structures:

Inter-cluster subgraph We call $\{v_i, v_j, v_k\}$ an inter-cluster subgraph, if v_i, v_j and v_k scatter in two neighboring clusters. In this case, one of the three vertices may be the root.

Intra-cluster subgraph If v_i, v_j and v_k are in the same cluster, then $\{v_i, v_j, v_k\}$ is regarded as an intra-cluster subgraph.

In the following, we describe how MEGA counts different structures of subgraphs in the computing step.

1. Rooted inter-cluster subgraphs: Let H_1 denote the number of rooted inter-cluster subgraphs in G'. Since each edge in $E(K_{i,1})$ forms a subgraph with the root v_r^i, we have

$$H_1 = \sum_{i=1}^{l} |E(K_{i,1})|. \tag{8}$$

2. Non-root inter-cluster subgraphs: Let H_2 denote the number of non-root inter-cluster subgraphs in G'. For each vertex $v \in K_{i,j}$, its upper neighbors, $N^{\uparrow}_{G'}(v)$, is defined by

$$N^{\uparrow}_{G'}(v) = N_{G'}(v) \bigcap K_{i,j-1},$$

and its lower neighbors, $N^{\downarrow}_{G'}(v)$, is defined by

$$N^{\downarrow}_{G'}(v) = N_{G'}(v) \bigcap K_{i,j+1}.$$

Then, for each $u^{\uparrow}_1$ and $u^{\uparrow}_2$ in $N^{\uparrow}_{G'}(v)$, if $(u^{\uparrow}_1, u^{\uparrow}_2) \in E(G)$, then $\{u^{\uparrow}_1, u^{\uparrow}_2, v\}$ forms a subgraph. Similarly, for each $w^{\downarrow}_1$ and $w^{\downarrow}_2$ in $N^{\downarrow}_{G'}(v)$, if $(w^{\downarrow}_1, w^{\downarrow}_2) \in E(G)$, then $\{w^{\downarrow}_1, w^{\downarrow}_2, v\}$ forms a subgraph. Hence, we have

$$H_2 \leq \sum_{v \in V(G') \setminus \{v_r\}} \left[\binom{|N^{\uparrow}_{G'}(v)|}{2} + \binom{|N^{\downarrow}_{G'}(v)|}{2} \right], \tag{9}$$

where v_r is the root of the BFS tree in hierarchical clustering.

3. Intra-cluster subgraphs: Let H_3 denote the number of intra-cluster subgraphs in G' and $G_{i,j} = (V(K_{i,j}), E(K_{i,j}))$ denote the induced subgraph of G'. Then, all intra-cluster subgraphs in $G_{i,j}$ can be counted in a recursive manner (i.e., we apply MEGA to count the number of subgraphs in $G_{i,j}$). Let $\mathsf{MEGA}(G)$ denote the number of subgraphs in G counted by MEGA, then we have

$$H_3 = \sum_{i,j} \mathsf{MEGA}(G_{i,j}). \tag{10}$$

Let the number of subgraphs counted in the pruning step be H_0. Then, the total number of subgraphs in the original graph G equals to $H_0 + H_1 + H_2 + H_3$. The computing step of the network motif counting problem is implemented as Algorithm 1. Note that the parameter, $maxC$, in Algorithm 1 is the total number of clusters in G_i.

Algorithm 1: Network Motif Counting

Input : Graph G', vertex sets $K_{i,j}$, number H_0
Output : sum (the number of subgraphs in G)

```
sum ← H_0
for i = 1, 2, ..., l do
    H_1 ← |E(K_{i,1})|
    for j = 1, 2, ..., maxC do
        G_{i,j} ← (V(K_{i,j}), E(K_{i,j}))
        H_3 ← MEGA(G_{i,j})
        for v ∈ K_{i,j} do
            N↑_{G'}(v) ← N_{G'}(v) ⋂ K_{i,j-1}
            N↓_{G'}(v) ← N_{G'}(v) ⋂ K_{i,j+1}
            for a, b ∈ N↑_{G'}(v) or a, b ∈ N↓_{G'}(v) do
                if (a, b) ∈ E(G) then
                    H_2 ← H_2 + 1
                end
            end
        end
    end
    sum ← sum + H_1 + H_2 + H_3
end
return sum
```

3.4 Computational Time Complexity

The computational time complexity of the pruning step depends on the number of pairs of vertices that we need to verify (whether an edge exists between them). Let $P(G)$ be the set of pruned vertices of G with threshold θ, i.e., $P(G) = \{v \in V(G) | v \notin V(G')\}$. Then, the time complexity of the pruning step is bounded above by $\binom{\theta}{2} \cdot |P(G)|$. Assume that $\theta \ll N$, where θ is the pruning threshold and N is the number of vertices in G, then the time complexity of the pruning step is bounded above by $O(N)$ since $|P(G)| \leq N$.

In hierarchical clustering, we select a vertex v_r^i as an optimal root of the BFS for G_i to obtain the distances from v_r^i to any other vertices in G_i. Hence, the time complexity of the hierarchical clustering step is $O(|V(G')| + |E(G')|) = O(|E(G')|)$ since we usually consider graphs in which $|V(G')| \leq |E(G')|$ in the network motif counting problem.

In the computing step, each vertex v has three types of neighbors: the neighbors in the upper cluster $N_{G'}^{\uparrow}(v)$, the neighbors in the lower cluster $N_{G'}^{\downarrow}(v)$, and the neighbors in the same cluster $N_{G'}^{\rightarrow}(v)$. Therefore, the number of pairs of vertices that we need to check for each vertex v is at most

$$\binom{|N_{G'}^{\uparrow}(v)|}{2} + \binom{|N_{G'}^{\downarrow}(v)|}{2} + \binom{|N_{G'}^{\rightarrow}(v)|}{2} \leq \binom{deg_{\max}}{2},$$

where $deg_{\max}$ is the maximum degree of G'. Thus, the time complexity of the computing step is $O(N' \cdot \binom{deg_{\max}}{2})$. We can conclude that the total computational time complexity of the MEGA framework for solving the network motif counting problem is $O(|E(G')| + N' \cdot \binom{deg_{\max}}{2})$.

3.5 MapReduce Software Implementation

The parallel computational structure in our MEGA framework can be naturally mapped into existing popular scalable software framework such as MapReduce. In particular, for the pruning step, a Mapper operation can count the degrees, and a single Reducer identifies vertices that do not satisfy the pruning conditions. For the hierarchical clustering step, Mapper operations can be delegated to count subgraphs in each cluster and also subgraphs that straddle clusters. For this purpose, we create a data structure to store the vertices in each cluster, and set the custom input split size to ensure that each Mapper can access a complete cluster at any one time. The total count from the Mappers can then be obtained by a single Reducer. Counting of subgraphs that straddle across clusters requires setting larger input split size so that a mapper accesses two clusters at any time. Fig. 2 shows an example of using MapReduce on MEGA to solve the network motif counting problem. For more details of the MapReduce implementation, please see: `https://github.com/MEGA-framework`.

4 Network Centrality Computation

In this section, we apply MEGA to compute the distance-based estimator and demonstrate how MEGA solves the aforementioned network inference problem (see Problem 2) efficiently but without compromising on accuracy comparing with the BFS heuristic algorithm in [5]. Note that we only consider connected graphs in this section since we are ranking vertices in the same connected component.

4.1 Pruning

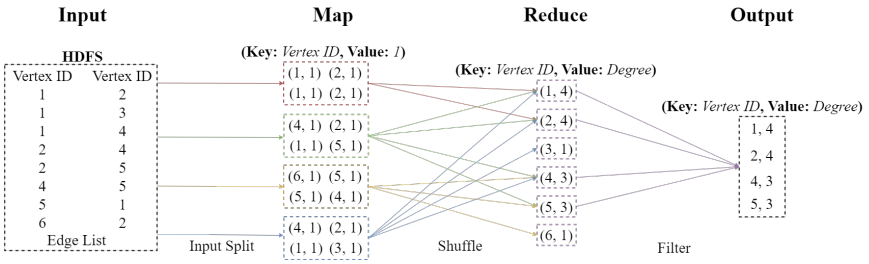
To handle less data, we first remove the trivial vertices that have less chance to be the distance center in the pruning step. Based on the statistical property of the data, we set the threshold θ to 1 such that all vertices in the pruned graph G' have degrees larger than 1. Note that if we set $\theta > 1$, G' will become fragmented, which increases the computational complexity significantly. Each vertex contains two parameters, subtree size T and sum of distances D. We use a rewriting system to update these two parameters for every vertex in the input graph G. Note that vertices receive message from a given vertex are referred to as parents p, and vertices for which a given vertex is parent are the children of that vertex $\mathsf{child}(p)$. Initially, we set the subtree size to 1 and the sum of distances to 0 for every vertex. When a vertex v with degree 1 is removed from G, it sends the message of $(T(v), D(v))$ to its parent. Subsequently, the rewriting system updates the subtree size of each parent by

$$T(p) = T(p) + \sum_{w \in \mathsf{child}(p)} T(w), \tag{11}$$

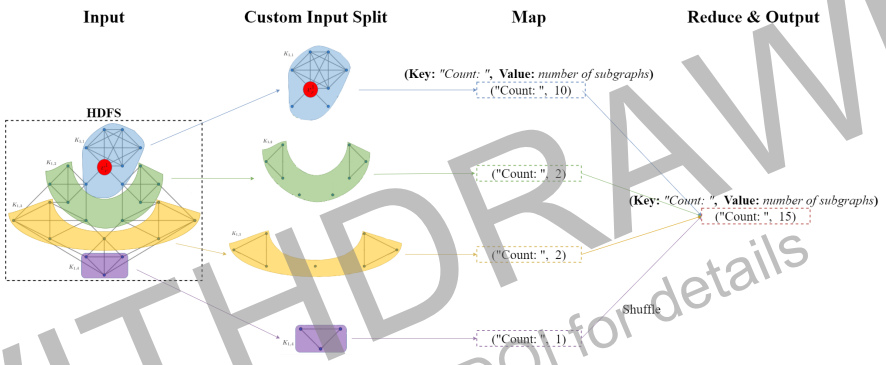
and the sum of distances by

$$D(p) = D(p) + \sum_{w \in \mathsf{child}(p)} [T(w) + D(w)]. \tag{12}$$

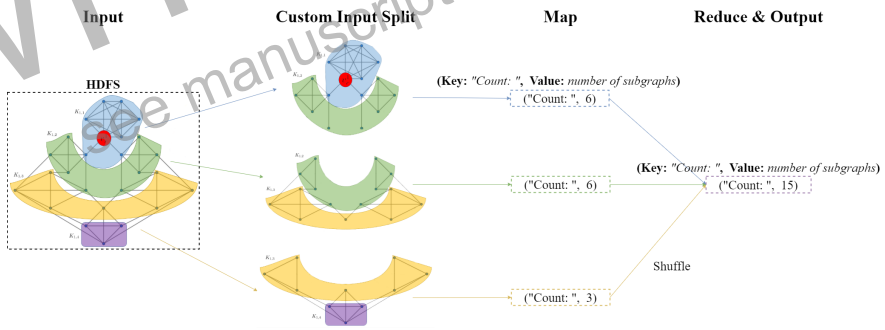
This step continues until all vertices in G' have a degree greater than 1. Note that it was proved that the rewriting system in the $\theta = 1$ pruning is equivalent to that in the *Election Algorithm* [36], and it can find the distance center in a tree within linear time complexity. In Fig. 3, we use an example to illustrate how the rewriting system of MEGA finds the distance center in a tree. However, social networks such as Facebook, Twitter have more complex structures that billions of users (vertices) can form millions of communities (cycles). Thus, finding the distance center in a tree is regarded as an ideal case.



(a) Pruning with $\theta = 2$



(b) Counting of intra-cluster subgraphs



(c) Counting of inter-cluster subgraphs

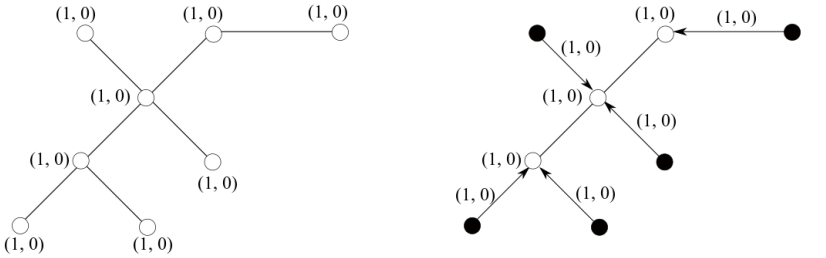
Figure 2: MapReduce implementation on MEGA for solving the network motif counting problem. The highly-parallelizable structure of MEGA allows us to adjust the custom input split size easily such that each mapper can retain sufficient information for the reducer.

Let $G' = (V(G'), E(G'))$ be the graph after pruning with N' vertices. Then, G' is a connected graph containing cycle(s) and $deg(v) > 1$ for all $v \in V(G')$. If $N' > 1$, we continue the process in the next step. Otherwise, G' is the distance center. Note that the distance center may be deleted in pruning for some graphs. For example, when G is a pseudo-tree (connected graph containing a single cycle), the remaining vertices of G after pruning are those on the cycle. To resolve this issue, we propose Theorem 1 to characterize the cases when the distance center is in G' and use Corollary 2 to find the distance center when it was deleted in pruning.

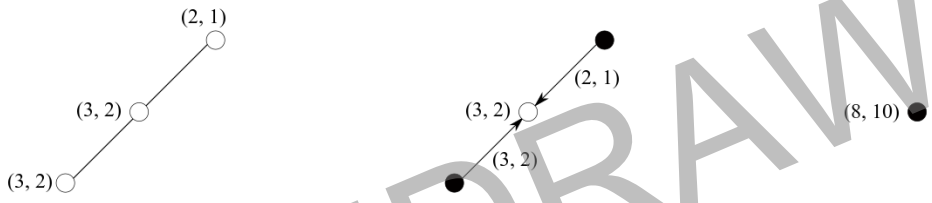
Theorem 1. Let $C_{dist}(G)$ be the distance center of G. If $N' \geq N/2$, then G' must contain $C_{dist}(G)$. If $N' < N/2$ and $C_{dist}(G) \notin G'$, then $C_{dist}(G)$ can be found in the complexity of $O(dia(G))$, where $dia(G)$ is the graph diameter of G.

Proof. First, we show that G' must contain $C_{dist}(G)$ whenever $N' \geq N/2$. We prove this part by contradiction. Assume that $N' \geq N/2$ and the distance center $v_c \in G \backslash G'$.

To prove this theorem, we define an edge (u, v) as a bridge if the removal of (u, v) disconnects G. Assume (u, v) is a bridge. Let C_u^v and C_v^u denote the connected component containing u and v respectively after removing (u, v).



(a) Initial State (b) Message Passing



(c) State 2 (d) Message Passing (e) Final State

Figure 3: Example of how the rewriting system in the $\theta = 1$ pruning finds the distance center in a tree. Initially, we set $T = 1$ and $D = 0$ for every vertex. The rewriting system then removes the leaves (in black color) and passes their messages (T, D) to the parents. We then update T and D of each parent based on (11) and (12) respectively. Pruning continues until all leaves are removed. The final state shows the distance center of this example with distance centrality equals to 10.

Lemma 6. If edge (u, v) is a bridge in G, then we have

$$S(u, G) = S(v, G) + |C_v^u| - |C_u^v|. \tag{13}$$

Proof. Let u and v be two vertices of G such that (u, v) is a bridge in G. Then, for any vertex $v_i \in C_v^u$ and $u_j \in C_u^v$, we have

$$dist(v_i, u_j) = dist(v_i, v) + 1 + dist(u_j, u).$$

Hence, the distance centrality of u in G, $S(u, G)$, can be rewritten in terms of the distance centrality of u and v in C_u^v and C_v^u respectively. We have

$$\begin{aligned} S(u, G) &= \sum_{w \in G} dist(u, w) \\ &= \sum_{w \in C_u^v} dist(u, w) + 1 + \sum_{w \in C_v^u} dist(u, w) \\ &= S(u, C_u^v) + 1 + \sum_{w \in C_v^u} [dist(v, w) + 1] \\ &= S(u, C_u^v) + 1 + S(v, C_v^u) + |C_v^u|. \end{aligned}$$

By following the same approach, we have

$$S(v, G) = S(u, C_u^v) + 1 + S(v, C_v^u) + |C_u^v|.$$

Combining two results above, we have

$$S(u, G) - S(v, G) = |C_v^u| - |C_u^v|.$$

□

Lemma 7. Let G' be the pruned graph and $v \in G \backslash G'$, then v is either a degree 1 vertex or an endpoint of a bridge.

Proof. Assume that v is not an endpoint of a bridge. Then, v must be contained in a cycle of G. The degree of v must be larger than or equal to 2 during pruning, which implies that $v \in G'$ and contradicts the assumption that $v \in G \backslash G'$. □

By Lemma 7, we conclude that v_c must be an endpoint of a bridge in G. We consider two distinct cases based on the distance between v_c and G' in the following. We define the distance from v_c to G' by

$$dist(v_c, G') = \min_{w \in G'} \{dist(v_c, w)\}.$$

Case 1: Assume that $dist(v_c, G') = 1$. Then, there is only one vertex $u \in G'$ such that $dist(v_c, u) = 1$ and (u, v_c) is a bridge by Lemma 7. Moreover, we have $S(v_c, G) = S(u, G) + |C_u^{v_c}| - |C_{v_c}^u|$ by Lemma 6. Since $|C_u^{v_c}| \geq N' \geq N/2$, we have $S(v_c, G) \geq S(u, G)$, which contradicts the assumption that v_c is the distance center.

Case 2: Assume that $dist(v_c, G') \geq 2$. From Case 1, we can deduce that for any two vertices u and v in $G \backslash G'$ and (u, v) is a bridge with $dist(u, G') < dist(v, G')$, then $S(u, G) < S(v, G)$. This implies that, for all v with $dist(v, G') \geq 2$, there is a neighbor u of v such that $S(u, G) < S(v, G)$, which is a contradiction.

Therefore, we can conclude that if $N' \geq N/2$, then G' must contain the distance center.

Now, let us consider the case that $C_{dist}(G) \notin G'$. Note that for each pruned neighbor u of $v' \in G'$, we have the fact that (v', u) is a bridge of G. Hence, we can compute $S(u, G)$ immediately by using Lemma 6. In particular, since v' has the minimum distance centrality on G', there is a unique pruned neighbor of v', say u', such that $S(u', G) < S(v', G)$. Since $C_{u'}^{v'}$ is a tree, we can apply the result of Theorem 3 in [6] to complete the rest of the proof. □

Corollary 2. If $N' < N/2$ and the unique vertex v' with the minimum distance centrality in G' is not $C_{dist}(G)$, then there is a unique path from v' to $C_{dist}(G)$. Moreover, the distance centrality of each vertex along the path is a monotonically decreasing sequence.

Let $dia(G)$ denote the graph diameter of G. Based on Corollary 2, MEGA is guaranteed to find $C_{dist}(G)$ within $O(dia(G))$ times when $C_{dist}(G)$ was deleted in pruning.

4.2 Hierarchical Clustering

In this step, we partition vertices in G' into different clusters based on the BFS tree traversal. Let v_r be the root of the BFS tree traversal for G'. Similar to (7), we use the distance between v_r and all other vertices in G' to define each cluster. Since G' must be a connected graph, we use K_i to denote the ith cluster of G'. We rewrite (7) by

$$K_i = \{v \in V(G') | dist(v, v_r) = i\}. \tag{14}$$

That is, vertex v is in cluster K_i if its distance from v_r is i. We can then find $S(v_r, G')$ by

$$S(v_r, G') = D(v_r) + \sum_{i=1}^{maxC} \left\{ \sum_{j=1}^{|K_i|} [T(v_j) \cdot i + D(v_j)] \right\}, \tag{15}$$

where $maxC$ is the number of clusters in G' and $|K_i|$ is the number of vertices in ith cluster. We can also calculate the total subtree size and the sum of distances for each cluster by $T(K_i) = \sum_{v \in K_i} T(v)$ and $D(K_i) = \sum_{v \in K_i} D(v)$ respectively.

The initial root v_r affects not only the number of clusters, but also the lower bound computation of every vertex in the next step (i.e., the overall performance of MEGA). We define a root as an optimal root if it has the highest chance to be the distance center. Note that if we can select an optimal root in this step, it can possibly minimize the number of BFSs to find the distance center in the next step. We apply decision tree algorithm to predict an ideal initial point for the first BFS in this step (cf. Fig. 4). We specify three test conditions for three attributes respectively:

Degree: Since we focus on lower bound calculation in the next step, let S be the exact distance centrality and S_{LR} be the lower bound on the distance centrality. Then, we have

$$S - S_{LR} \leq i \cdot |K_i|, \tag{16}$$

where i is the cluster level and $|K_i|$ is the number of vertices in cluster K_i. Thus, a smaller i can produce a tighter bound. As we have $S(v_r, G')$ from (15), and we know that vertices in K_1 must have a tighter bound based on (16) and

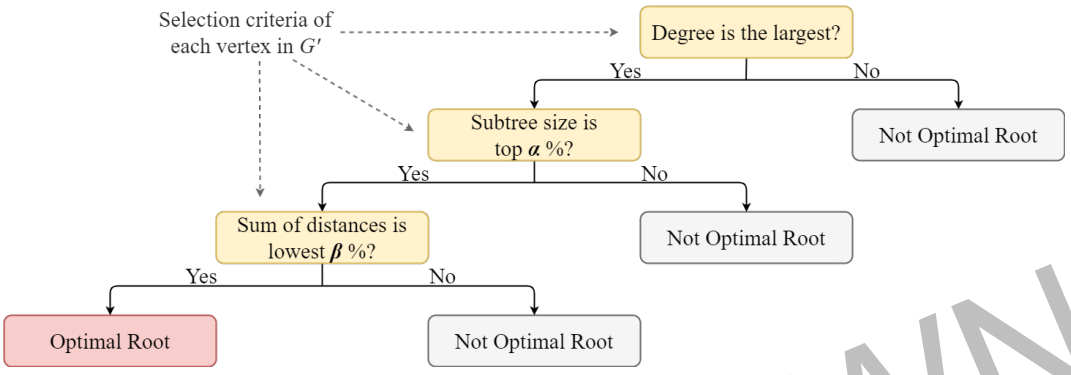


Figure 4: Schematic of decision tree for finding an optimal root in hierarchical clustering. Note that we can use a data-driven approach to obtain the hyperparameters, α and β, of the classifier.

vertices with maximum degree can maximize the number of vertices in K_1, choosing vertices with maximum degree as the root will provide more information about the degree bounds. Based on this property, our intuition is that vertices with maximum degree are more likely to be an optimal root. Indeed, our evaluation in Section 5 shows that the distance centers of 75.3% synthetic networks have maximum degree.

Subtree Size: Based on Lemma 7 and the rewriting system in pruning, we know that vertices in G' can either have a subtree size equals to 1 or be an endpoint of a bridge. If a vertex v is an endpoint of a bridge, we can then compute the exact distance centrality of v to all removed vertices along its subtree based on Lemma 6. Thus, the larger the subtree size a vertex has, the tighter the bound we can compute. Also, it is easy to deduce that vertices with a larger subtree size have a higher chance of being the center of a graph.

Sum of Distances: Based on (4), distance center has minimum distance centrality. Combining the results from the two attributes above, then vertices with a smaller sum of distances have a higher chance of being the distance center.

We use decision tree classification to estimate an optimal root based on the selection criteria above. Note that our decision tree focuses on the property of each vertex in the graph instead of the generic graph property. Thus, our approach is different from the traditional decision tree learning. Note that we can always tune the hyperparameters of the classifier using a data-driven approach to prevent overfitting and optimize the classification performance.

4.3 Computing

To avoid doing BFS starting from every vertex in G', we propose Algorithm 2 to minimize the number of BFSs. We use the idea of computing top-k closeness centrality in [37] as a basis, which is to trace the lower bound on the distance centrality of each vertex. We denote the lower bound on the distance centrality of vertex v in G' as $S_{LR}(v, G')$. Note that if $S(v, G') < S_{LR}(u, G')$ or $S(v, G') \leq S(u, G')$ for all $u \in V(G')$, then we can say v is the distance center of G'.

In this step, we first compute S_{LR} for each vertex in G' and insert it into a min-priority queue Q_s sorted in ascending order of S_{LR}. Then, we use $DeQueue$ to obtain the smallest S_{LR} in Q_s and check if $S_{LR} = S$. That is, if $S_{LR}(v, G')$ is the smallest in Q_s, then we do hierarchical clustering for G' using v as the root and compute $S(v, G')$ with (15). If $S(v, G') = S_{LR}(v, G')$, then we say v is the distance center of G'. Otherwise, we use $EnQueue$ to put $S(v, G')$ into Q_s and repeat the whole process until we get the distance center of G'. Lastly, we apply Theorem 1 to verify our result and use Corollary 2 to bring back the distance center if $C_{dist}(G) \notin G'$.

Algorithm 2: Distance Center Calculation

Input : Graph G, graph G', lower bounds S_{LR}
Output : distance center of G, $C_{dist}(G)$
$Q_s \leftarrow$EnQueue(v_r, $S(v_r, G')$)
foreach v *in* $V(G')\backslash\{v_r\}$ **do**
  $Q_s \leftarrow$EnQueue(v, $S_{LR}(v, G')$)
end
while *continue* = *true* **do**
  (v, $S_{LR}(v, G')$)$\leftarrow$ DeQueue(Q_s)
  if $S(v, G') = S_{LR}(v, G')$ **then**
    $C_{dist}(G') \leftarrow v$
    continue$\leftarrow$false
  else
    $Q_s \leftarrow$EnQueue(v, $S(v, G')$)
  end
end
if $N' \geq N/2$ **then**
  $C_{dist}(G) \leftarrow C_{dist}(G')$ (see Corollary 2)
end
return $C_{dist}(G)$

To calculate $S_{LR}(u, G')$, where $u \in K_i$, we propose Theorem 2 based on the triangle inequality to characterize the lower bound in terms of clusters, $T(K_i)$ and $D(K_i)$. Note that all information about the pruned vertices are already stored in $T(v)$ and $D(v)$ for all $v \in G'$.

Theorem 2. Let G' be a pruned graph and $u \in K_i$, we have

$$S(u, G') \geq \sum_{0 \leq j \leq maxC} [|i - j| \cdot T(K_j) + D(K_j)].$$

Proof. Let $v \in K_j$, where j is an integer such that $0 \leq j \leq maxC$. Let $\tilde{v}$ denote the set of pruned vertices such that v is the ancestor of these pruned vertices (e.g., $v_5 \in \tilde{v_4}$ in Fig. 6). First, we consider the distance from vertex u to all the vertices in $\tilde{v}$ including v itself. Without loss of generality, we assume that the common ancestor w of u and v is in level h. Then, we have

$$d(u, v) = |h - i| + |h - j| \geq |i - j|,$$

where the equality holds if $u = w$ or $v = w$. Then, for other vertices in $\tilde{v}$, we have

$$\begin{aligned} \sum_{x \in \tilde{v}} d(u, x) &= \sum_{x \in \tilde{v}} [d(u, v) + d(v, x)] \\ &= T(v) \cdot d(u, v) + D(v) \\ &\geq T(v) \cdot |i - j| + D(v). \end{aligned}$$

Hence, we have

$$\begin{aligned} S(u, G) &\geq \sum_{0 \leq j \leq maxC} \left\{ \sum_{v \in K_j} [T(v) \cdot |i - j| + D(v)] \right\} \\ &= \sum_{0 \leq j \leq maxC} \left[|i - j| \cdot \sum_{v \in K_j} T(v) + \sum_{v \in K_j} D(v) \right] \\ &= \sum_{0 \leq j \leq maxC} [|i - j| \cdot T(K_j) + D(K_j)]. \end{aligned}$$

□

The two terms, $T(K_j)$ and $D(K_j)$, in Theorem 2 were computed in hierarchical clustering, and thus the complexity of computing the lower bound of vertex u is only a linear function of $maxC$. Note that in Theorem 2, the lower bound on the distance centrality of every vertex in the same cluster must be the same. Therefore, to tighten the lower bound of each vertex, we characterize two cases when $|i - j| < 2$:

Case 1: We have $\sum_{u \in N_{G'}(v)}[T(u) + D(u)]$ as the exact distance for v since $dist(u, v) = 1$ for all $u \in N_{G'}(v)$.

Case 2: If $u \notin N_{G'}(v)$, then we have $2 \cdot T(u) + D(u)$ as the lower bound for v since $dist(u, v)$ must be at least 2.

We propose Algorithm 3 to compute the cluster-based lower bound on the distance centrality of each vertex in G' by leveraging the results in Theorem 2. Note that the initialization of the parameter, sum, in Algorithm 3 is the distance between the root v_r and the cluster K_i. Although the lower bound of each vertex is supposed to be tighter after considering Case 1 and 2, the computational cost of looping over all the vertices becomes so much higher. To address this problem, we can use MapReduce to enable parallel processing such that the lower bound of each vertex can be computed simultaneously (cf. Fig. 5).

Algorithm 3: Cluster-based Lower Bound Calculation

Input : Graph G', vertex v_r, vertex sets K
Output : Lower bound S_{LR} for every vertex in G'
for $i = 1, 2, \ldots, maxC$ **do**
  $sum \leftarrow T(v_r) \cdot i + D(v_r)$
  $K_{neighbour} \leftarrow [\,]$
  for $j = 1, 2, \ldots, maxC$ **do**
    if $|i - j| < 2$ **then**
      $K_{neighbour}$.insert(j)
    else
      $sum \leftarrow sum + T(K_j) \cdot |i - j| + D(K_j)$
    end
  end
  for $v \in K_i$ **do**
    $S_{LR}(v, G') \leftarrow sum + D(v)$
    for $u \in N_{G'}(v)$ *and* $u \neq v_r$ **do**
      $S_{LR}(v, G') \leftarrow T(u) + D(u) + S_{LR}(v, G')$
    end
    for j *in* $K_{neighbour}$ **do**
      for $u \in K_j$ *and* $u \notin N_{G'}(v)$ **do**
        $S_{LR}(v, G') \leftarrow T(u) \cdot 2 + D(u) + S_{LR}(v, G')$
      end
    end
  end
end
return S_{LR}

It is worth noting that if v is the root in the first hierarchical clustering and $S(v, G')$ is the smallest in Q_s, then we only need to do BFS once. Otherwise, in the worst case, we have to do BFS for all vertices in G'. Thus, if we can further tighten the lower bound on the distance centrality for each vertex, we can then omit the computation of the exact distance centrality of every vertex. To achieve this goal, we recompute the lower bound of every vertex in each BFS and update it if the bound is found to be tighter. That is, if the recomputed lower bound of a vertex appears to be larger than its current bound, then we update it. As we do more BFS, the lower bound should be updated to be tighter.

4.4 Illustrative Example

In the following, we use an example to illustrate the difference of finding the rumor source estimator between MEGA and the algorithm in [5]. Using the same example as in [5], we have a rumor subgraph as shown in Fig. 6(a). There are 5 infected vertices (v_1 to v_5) and we want to find the BFS heuristic rumor center of this rumor subgraph using MEGA.

In the $\theta = 1$ pruning, we remove v_5 and update its parent v_4 such that $T(v_4) = 2$ and $D(v_4) = 1$, then we obtain the pruned graph G' as shown in Fig. 6(b). We then select an optimal root (which is v_4 based on the selection criteria in Fig. 4) for the BFS tree traversal in hierarchical clustering. But in this example, we use every vertex in the rumor subgraph as the root and see, in the worst case, how many BFSs we need to perform. In Fig. 6(c), we first use v_1 as the root, then

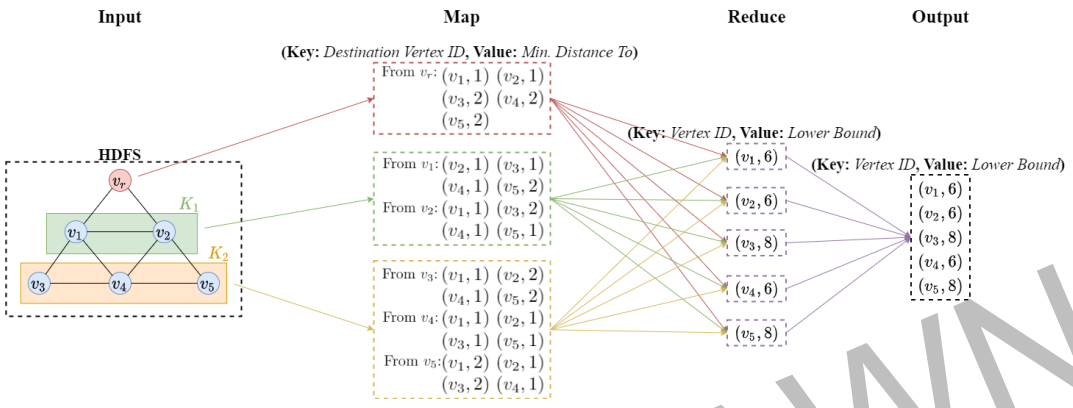


Figure 5: Example of how we use MapReduce to reduce the computational cost of MEGA. We leverage the key-value pair property of MapReduce and the hierarchy property of MEGA to calculate the lower bound on the distance centrality of each vertex based on Theorem 2.

we have

$$S(v_1, G') = 0 + 1 + 3 + 2 = 6$$
$$S_{LR}(v_2, G') = 1 + 0 + 1 + 5 = 7$$
$$S_{LR}(v_3, G') = 2 + 1 + 0 + 3 = 6$$
$$S_{LR}(v_4, G') = 1 + 2 + 1 + 1 = 5.$$

Since $S_{LR}(v_4, G')$ is the smallest, we need to check if $S(v_4, G') = S_{LR}(v_4, G')$ by using v_4 as the root for the second BFS tree traversal. As we have $S(v_4, G') = S_{LR}(v_4, G')$ and the lower bound of every vertex remains unchanged, we can say v_4 is the BFS heuristic rumor center of G'. Since $N' > N/2$, based on Theorem 1, we conclude that v_4 is the BFS heuristic rumor center of G.

Similarly, for root v_2 (Fig. 6(d)) and v_3 (Fig. 6(e)), we also need to do BFS tree traversal twice. However, for root v_4 (Fig. 6(f)), we have

$$S(v_4, G') = 1 + 2 + 1 + 1 = 5$$
$$S_{LR}(v_1, G') = 0 + 1 + 2 + 3 = 6$$
$$S_{LR}(v_2, G') = 1 + 0 + 1 + 5 = 7$$
$$S_{LR}(v_3, G') = 2 + 1 + 0 + 3 = 6.$$

Since $S(v_4, G')$ is already the smallest, we can then conclude that v_4 is the rumor center of G by using only one BFS tree. As such, in the worst case, we only need to perform BFS twice for finding the distance-based heuristic rumor center of the rumor subgraph. Meanwhile, v_4 is also identified as the rumor center using the BFS heuristic algorithm in [5] but it requires to construct BFS trees starting from every vertex in the rumor subgraph. Thus, in this example, the number of BFSs performed and the number of edges visited by MEGA are both less than that of the algorithm in [5].

5 Experimental Performance Evaluation

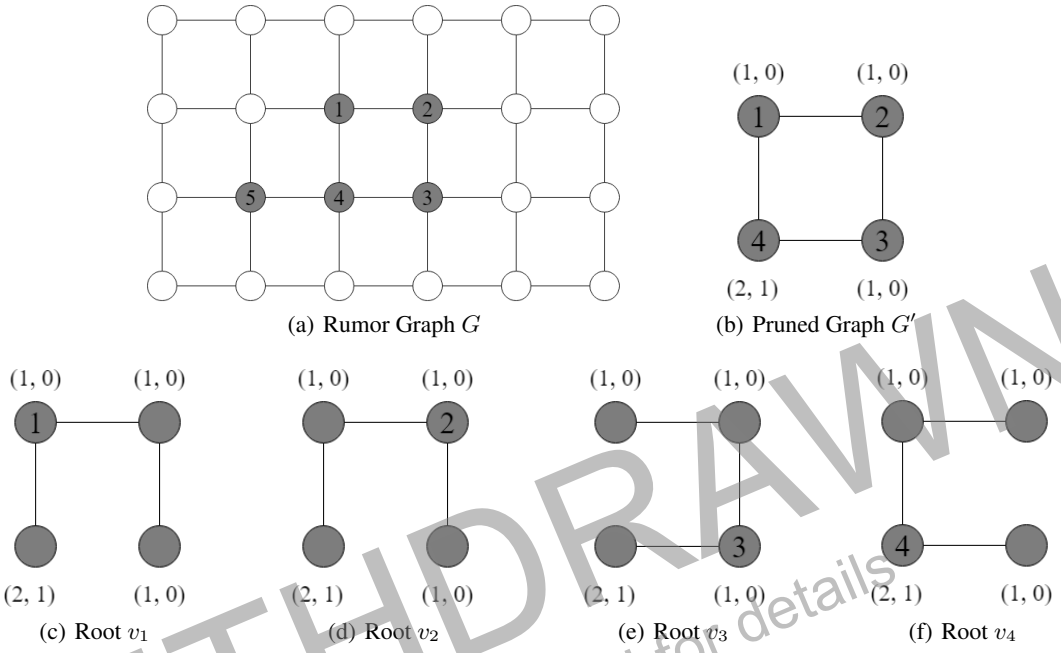
In this section, we assess the performance of MEGA on simple graphs with different degree distributions. All the experiments are conducted on a 64-bit computer running a Windows 10 system with Intel(R) Core(TM) i7-9700K CPU 3.60GHz and 64.00 GB of RAM configuration.

5.1 Network Motif Counting

We evaluate the performance of MEGA on the network motif counting problem using the graphs provided by Stanford Large Network Dataset Collection (SNAP) [38].

5.1.1 Optimal Threshold Tuning

The threshold θ is a critical parameter in the pruning step. Once θ is set, there are at most $\binom{\theta}{2} \cdot |P(G)|$ pairs of vertices need to be checked. Hence, a larger θ leads to a longer time spent for pruning. On the other hand, if more vertices



(a) Rumor Graph G (b) Pruned Graph G'

(c) Root v_1 (d) Root v_2 (e) Root v_3 (f) Root v_4

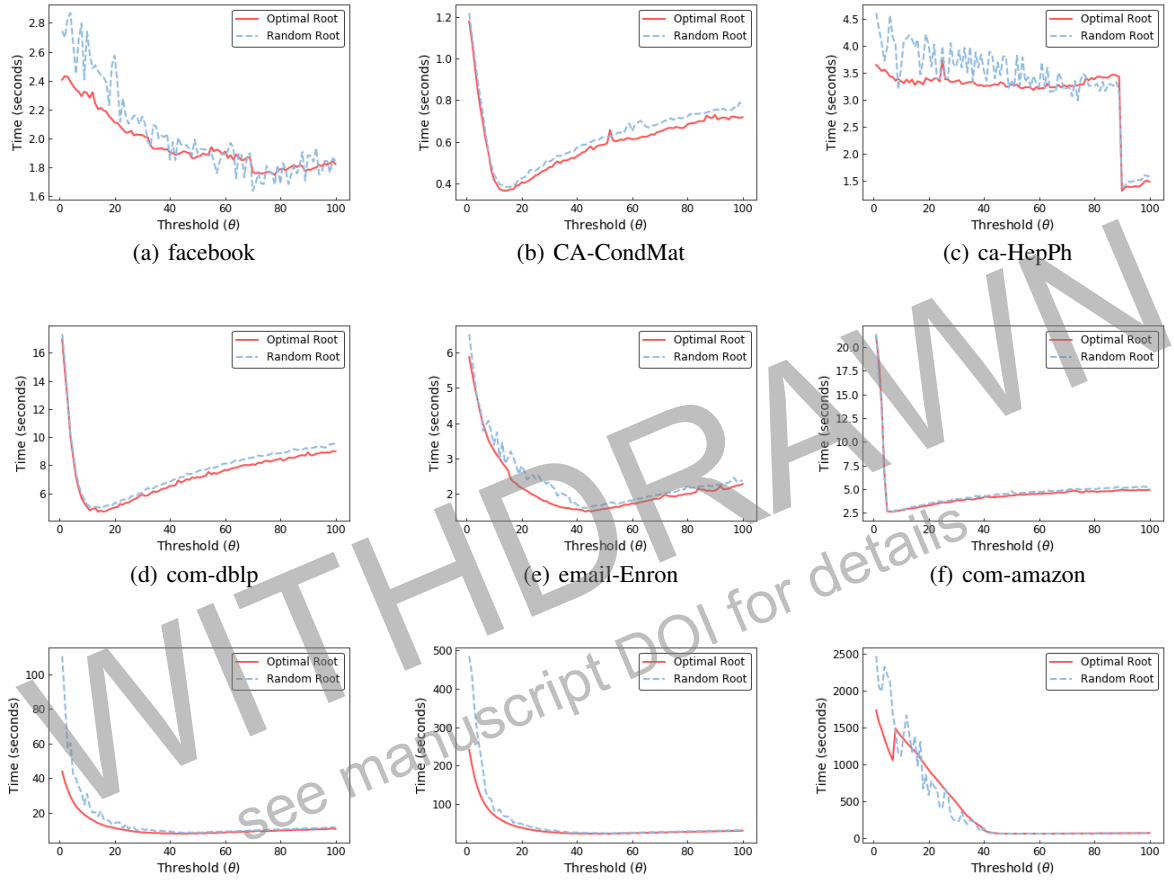
Figure 6: Example of how MEGA finds the BFS heuristic rumor center of a rumor subgraph. There are totally 5 infected vertices (in gray color). After pruning, we have a pruned graph G'. We then select v_1 to v_4 as the root in hierarchical clustering respectively and see how many BFSs we have to build for each root.

are pruned off from G, then the graph becomes more fragmented. Thus, there is a trade-off between pruning and hierarchical clustering, and it depends on the threshold θ. Fig. 7 shows how the threshold θ affects the computation time of MEGA on nine real-world networks provided by SNAP. We also compare the performance of MEGA using the optimal root and the random root in hierarchical clustering since other centrality measures such as PageRank require global information from all other vertices that must increase the computational complexity. We see that MEGA with the optimal root performs better than that with the random root. We also note that finding an optimal threshold for an arbitrary graph is not a straight-forward task unless we further exploit the inherent structure of the graph.

Based on Lemma 1, we use a data-driven (statistical) model to approximate the optimal threshold θ^* for MEGA. We generate 6,000 synthetic networks in which the optimal thresholds θ^* are computed. Each network has 1,000 vertices, and the number of edges of each network is not fixed. Note that the number of vertices of the input graph is not a critical parameter for computing θ^*. In Fig. 8, we see that the gradient of the blue line, which is a linear regression model that fits the results, is relatively small that keeps θ^* always less than 100, even the graph size starts to increase. We also note that noisy data only appear when the graph size is comparatively small. Therefore, the optimal threshold θ^* computed by this model must satisfy the condition of $\theta^* \ll N$ and balance the trade-off for large-scale networks.

5.1.2 Evaluation on Real-world Networks

We compare MEGA with the algorithm in [39], which is an award-winning work of the MIT/Amazon/IEEE Graph Challenge [14]. The algorithm in [39] assigns direction to each edge based on the degree of each vertex. It implies that, for each vertex v, there are $\binom{deg^+(v)}{2}$ pairs of vertices need to be checked, where $deg^+(v)$ is the outdegree of v. Hence, its computational time complexity is $O(|E(G)| + N \cdot \binom{deg^+_{\max}}{2})$, where $|E(G)|$ is the time complexity of the direction assignment and $deg^+_{\max}$ is the maximum outdegree. This time complexity is the same as that of MEGA (cf. Section 3.4). However, in Fig. 9, we see that MEGA with θ^* outperforms the algorithm in [39] on different real-world networks by averagely 2.97 times faster. Therefore, we conclude that if we can find an optimal threshold θ^* that leverages the structure of the graph in advance, then MEGA can complete the counting task in pruning with a computational time complexity of $O(N)$ and beat the algorithm in [39].



(a) facebook (b) CA-CondMat (c) ca-HepPh

(d) com-dblp (e) email-Enron (f) com-amazon

(g) loc-gowalla (h) com-youtube (i) as-skitter

Figure 7: The performance of MEGA on nine real-world networks with different thresholds θ ranging from 0 to 100. The y-axis is the running time in seconds, and the x-axis is the value of θ. We also compare the performance of MEGA using the optimal root (red) and the random root (blue).

5.1.3 Evaluation on Synthetic Random Networks

We compare the performance of MEGA and the algorithm in [39] on ten synthetic random networks generated by the BA model and the ER model respectively. We compute the optimal threshold θ^* for MEGA based on Lemma 2 and 3 to decompose the random networks.

For BA network, we randomly generate the two parameters (m_0, m) for the BA model such that we can obtain ten random networks, and each network contains 1 million vertices and $9,999,945$ edges. We use Corollary 1 to compute the optimal threshold θ^* of each BA network such that MEGA can optimally decompose the BA networks in pruning. In Fig. 10(a), we see that although some vertices have a large degree, MEGA can still be able to decompose the networks with θ^* and outperform the algorithm in [39]. Moreover, since we only execute the pruning step with a small threshold ($\theta^* \ll N$), the computational time complexity of MEGA is simply $O(N)$.

Similarly, we generate ten ER random networks with the same size as the BA networks. In Fig. 10(b), we observe that MEGA beats the algorithm in [39] with θ^*. Note that the probability of a given vertex has degree greater than θ^* in the ER random networks is less than 8%, which implies the random networks are largely decomposed in pruning.

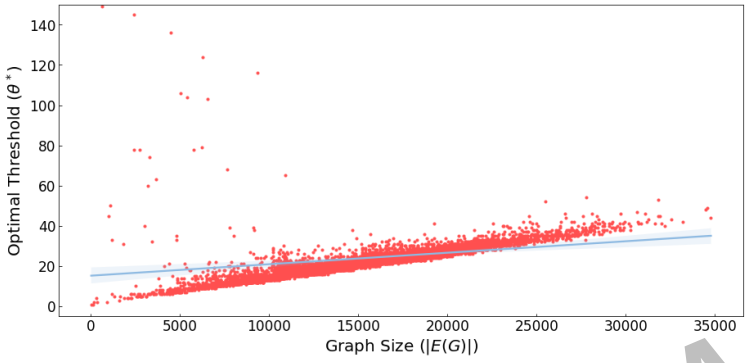


Figure 8: Optimal threshold tuning on 6,000 synthetic networks based on Lemma 1. The y-axis is the value of the optimal threshold θ^*, and the x-axis is the graph size $|E(G)|$ of each network. The blue line is a linear regression model that fits the results.

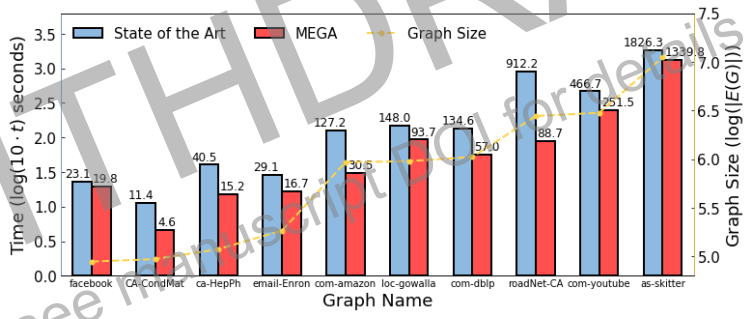
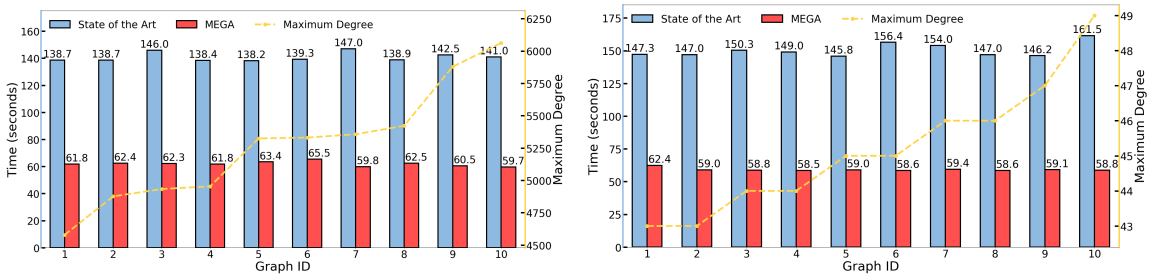


Figure 9: Comparison between MEGA and the algorithm in [39] (which we denote it as state of the art) on ten real-world networks. The primary y-axis (blue) is the running time t in seconds, and the secondary y-axis (yellow) is the graph size $|E(G)|$. A log scale is used for these two axes, and we multiply t by 10 to avoid negative values (i.e., primary = $\log(10 \cdot t)$ seconds and secondary = $\log(|E(G)|)$). The number on top of each bar is the exact running time of both algorithms.



(a) BA random networks (b) ER random networks

Figure 10: Comparison between MEGA and the algorithm in [39] (which we denote it as state of the art) on ten BA random networks (left) and ten ER random networks (right) respectively. Each random network has 1M vertices and 9.9M edges. The primary y-axis (blue) is the running time in seconds, and the secondary y-axis (yellow) is the maximum degree of each random network.

5.2 Network Centrality Computation

In this section, we assess the performance of MEGA on network centrality computation for rumor source detection.

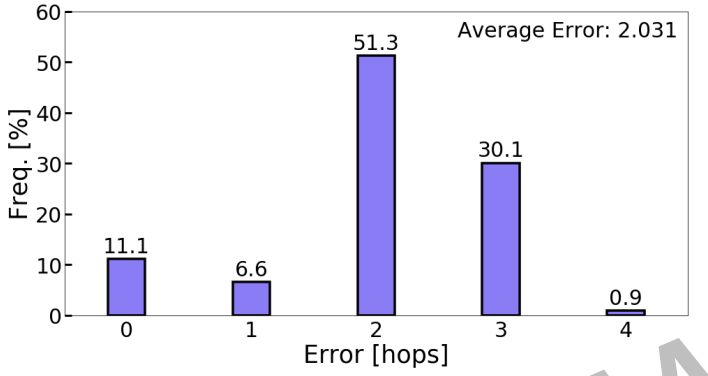


Figure 11: Performance of the decision tree classifier finding the optimal root on 1,000 synthetic networks. We define the error as the graph distance between the distance center and the predicted vertex.

5.2.1 Methodology

We consider both performance improvement and accuracy on rumor source detection. To maximize the efficiency, we use two different approaches to reduce the number of BFSs: one based on the cluster-based lower bound and one that combines the cluster-based lower bound and multi-source BFS. We first compare MEGA with the algorithm presented in [37] based on the performance improvement, and then compare the accuracy of rumor source detection with the work in [5]. Note that the work in [37] has been shown to outperform other existing algorithms for finding the exact and approximate top-k closeness centrality.

We apply MEGA on different real-world networks provided by SNAP [38] and the 10th DIMACS Implementation Challenge [40]. We use the *speedup* suggested by [37] to evaluate the performance improvement. The speedup is given by $\frac{|V(G)| \cdot |E(G)|}{|E_v(G)|}$, where $|E_v(G)|$ is the total number of edges that have been visited by an algorithm in graph G.

5.2.2 Optimal Root Estimation with Decision Tree

We apply decision tree algorithm to approximate the optimal root in hierarchical clustering. We compute the centrality measures for the 6,000 synthetic networks, and use 80% of data as the training set and the remaining as the test set. Note that the time required to estimate the optimal starting root for the real-world networks is insignificant as the best estimator is pre-trained in the preprocessing step.

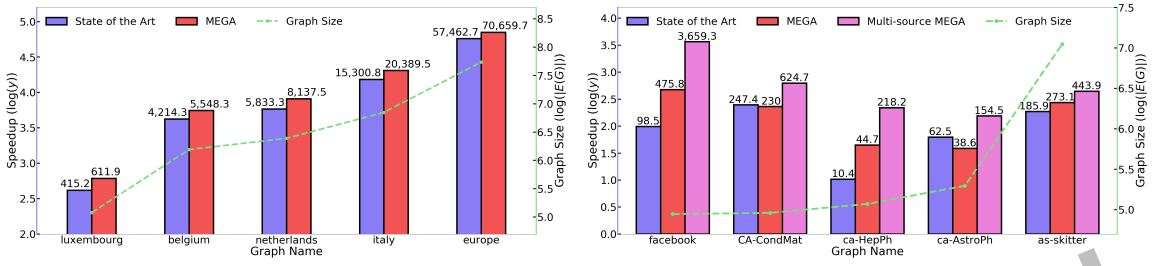
We specify the selection criteria based on the three selected attributes in Fig. 4 and fit it into our classifier. In Fig. 11, we see that the average error of our fine-tuned classifier is only 2.031 hops from the center, and the error is always within 4 hops. Note that we do not use a confusion matrix to evaluate the classifier as we are just ranking vertices with a higher chance to be the distance center.

5.2.3 Evaluation on Street Networks

The solution in [37] uses the level-based and neighborhood-based lower bound (*NBBound*) to compute the top-k closeness centrality of the input graph G directly. Since it calculates the lower bound for every vertex in G, it may take those trivial vertices (leaves) into consideration when finding the top-k closeness centrality. In Fig. 12(a), more than 15% of vertices are removed from each graph in the pruning step of MEGA. Since the number of vertices and edges we need to handle is always less than that of NBBound after pruning, the number of edges visited by our framework must be less than that of NBBound for every BFS tree traversal. As a result, the more vertices we have removed in pruning, the larger the speedup we can reach. In this case, the speedup on street networks of our MEGA framework must be better than that of NBBound.

5.2.4 Evaluation on Complex Networks

The efficacy of pruning in complex networks is relatively low since the graph diameter of such networks is usually small. To further increase the speedup of MEGA on complex networks, we utilize the *Multi-Source BFS (MS-BFS)* algorithm introduced in [41]. The idea is motivated by the observation that, when running a large number of BFSs sequentially, most of the edges are visited multiple times that would deteriorate the overall performance. The MS-BFS



(a) Street networks (b) Complex networks

Figure 12: Comparison between MEGA and the algorithm in [37] (which we denote it as state of the art) on five street networks (left) and five complex networks (right) respectively. The primary y-axis (blue) is the speedup y, and the secondary y-axis (green) is the graph size $|E(G)|$. A log scale is used for these two axes (i.e., primary = $\log y$ and secondary = $\log(|E(G)|)$). The number on top of each bar is the exact speedup of both algorithms.

algorithm addresses this problem by running BFSs from multiple sources concurrently. When a set of BFSs visits the same vertex, this vertex will only be visited once, and its information will be shared with all BFSs in the set.

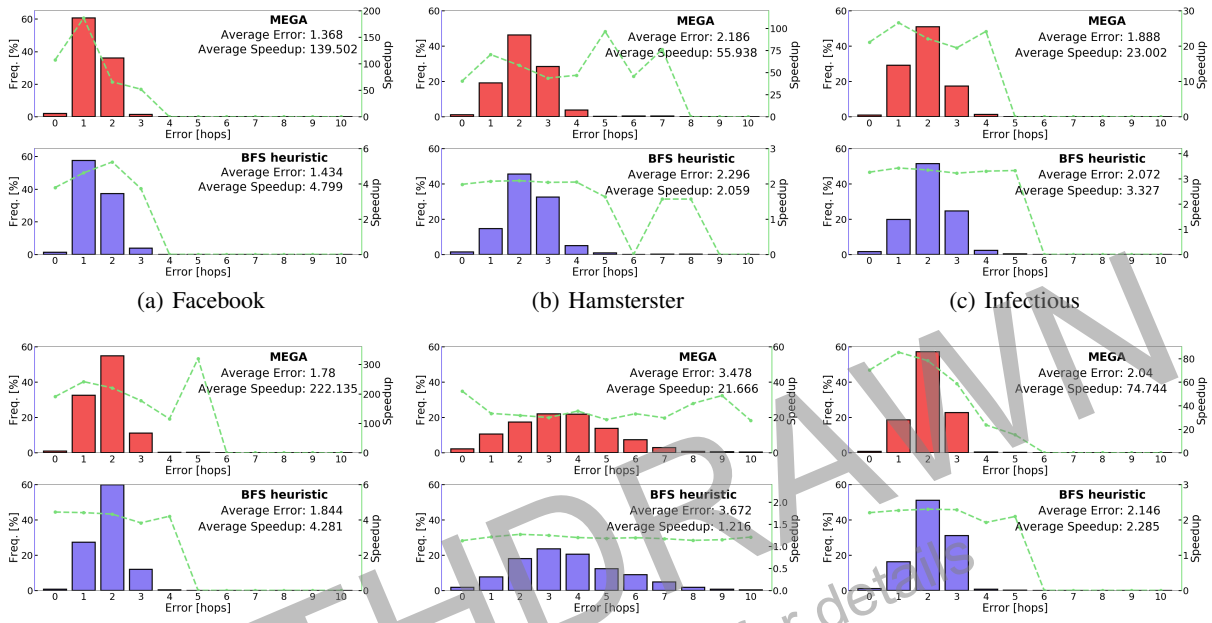
The authors of [41] also pointed out that MS-BFS is most effective in *small-world graphs*, where the graph diameter is small compared to the size. Thus, a large number of visits can be shared by multiple BFSs when applying MS-BFS algorithm on small-world graphs. Since the graph diameter of street networks is usually large, it is meaningless to apply MS-BFS on such graphs. This is confirmed by the experimental results in Fig. 12(b) that MS-BFS greatly increases the speedup of MEGA on complex networks.

5.2.5 Rumor Source Detection in Real-world Networks

We perform simulations on six important real networks obtained from KONECT [42] and compare the performance of our framework with the BFS heuristic in [5]. For each graph, we randomly select a rumor source vertex and let the rumor spread to 100 vertices. We then apply MEGA and the BFS heuristic in [5] respectively to find a source estimator. Note that the error indicates the graph distance from the source estimator to the real source vertex. We perform over 500 simulations and calculate the average error and speedup for each graph. In Fig. 13, we can observe that the speedup of our MEGA framework is significantly larger than that of the BFS heuristic in [5] and there is only a slight difference of accuracy (average error) between these two approaches.

6 Conclusion

We proposed the MEGA, a machine learning-enhanced framework for judicious pruning and hierarchical clustering of large graphs for accelerated computation. We demonstrated that machine learning techniques like regression can effectively learn the inherent statistics of graph data to derive approximately good algorithmic tuning for counting network motifs and computing network centrality, even outperforming state of the art. Interestingly, particularly for graphs with distinctive structure, e.g., when the graphs were similar to Barabási-Albert random graph model, we showed the optimal configuration to completely decompose the large graph with time complexity $O(N)$. In addition, different machine learning techniques can further reduce the computational complexity of the MEGA framework if we can train a classifier to find ideal initial point for BFS tree traversal of different graphs in hierarchical clustering. In our current work, geodesic distance is used to differentiate clusters, and it will be interesting to compare alternative graph-theoretic metrics for clustering. As future work, we will extend the MEGA framework for other computationally challenging graph problems and also study the deep learning, e.g., Graph Neural Network (GNN), for optimal parameter tuning of the hierarchical clustering as well as more efficient parallel software implementation.



(a) Facebook (b) Hamsterster (c) Infectious

(d) Twitch (e) Twitter (f) Wikipedia Elections

Figure 13: Histograms of the error for MEGA (red) and the BFS heuristic in [5] (blue) on six real networks with 100 infected vertices. The green dotted line is the average speedup for each error and the average error is the average distance (in terms of hop count) between the real source and the estimators calculated by both methods over 500 simulations.

References

[1] World Health Organization. Novel coronavirus (2019-nCoV): situation report, 13. Technical report, World Health Organization, 2020.

[2] Matteo Cinelli, Walter Quattrociocchi, Alessandro Galeazzi, Carlo Michele Valensise, Emanuele Brugnoli, Ana Lucia Schmidt, Paola Zola, Fabiana Zollo, and Antonio Scala. The COVID-19 social media infodemic. *Scientific Reports*, 10(1), Oct 2020.

[3] Paul W. Holland and Samuel Leinhardt. A method for detecting structure in sociometric data. *American Journal of Sociology*, 76(3):492–513, 1970.

[4] Z. Zhao, M. Khan, V. S. A. Kumar, and M. V. Marathe. Subgraph enumeration in large social contact networks using parallel color coding and streaming. In *ICPP*, pages 594–603, 2010.

[5] D. Shah and T. Zaman. Rumors in a network: Who's the culprit? *IEEE Transactions on Information Theory*, 57(8):5163–5181, 2011.

[6] P. Yu, C. W. Tan, and H. Fu. Averting cascading failures in networked infrastructures: Poset-constrained graph algorithms. *IEEE Journal of Selected Topics in Signal Processing*, 12(4):733–748, 2018.

[7] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, pages 135–146, 2010.

[8] Kameshwar Munagala and Abhiram Ranade. I/O-complexity of graph algorithms. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 687–694, 1999.

[9] Andrew Lumsdaine, Douglas Gregor, Bruce Hendrickson, and Jonathan Berry. Challenges in parallel graph processing. *Parallel Processing Letters*, 17:5–20, 2007.

[10] David Jensen. Statistical challenges to inductive inference in linked data. In *Seventh International Workshop on Artificial Intelligence and Statistics*, 1999.

[11] Michael Schweinberger and Mark S. Handcock. Local dependence in random graph models: characterization, properties and statistical inference. *Journal of the Royal Statistical Society*, 77:647–676, 2015.

[12] Robert Endre Tarjan. A hierarchical clustering algorithm using strong components. *Information Processing Letters*, 14(1):26 – 29, 1982.

[13] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[14] Siddharth Samsi, Vijay Gadepally, Michael Hurley, Michael Jones, Edward Kao, Sanjeev Mohindra, Paul Monticciolo, Albert Reuther, Steven Smith, William Song, Diane Staheli, and Jeremy Kepner. GraphChallenge.org: Raising the bar on graph analytic performance, 2018.

[15] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Proceedings of the 14th ACM (KDD) international conference on Knowledge discovery and data mining*, pages 16–24. ACM, 2008.

[16] LN. Wang, J. Zhang, K.-L. Tan, and A. K. Tung. On triangulationbased dense neighborhood graph discovery. *Proceedings of the VLDB Endowment*, 4:58–68, 2010.

[17] C. E. Tsourakakis, P. Drineas, E. Michelakis, I. Koutis, and C. Faloutsos. Spectral counting of triangles via element-wise sparsification and triangle-based link recommendation. *Social Network Analysis and Mining*, 1:75–81, 2011.

[18] Charalampos Tsourakakis. The k-clique densest subgraph problem. *International World Wide Web Conferences Steering Committee*, 2015.

[19] Huang Xin and et al. Querying k-truss community in large and dynamic graphs. *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014.

[20] Koryo Miura and Yasuyuki Miyazaki. Concept of the tension truss antenna. *AIAA journal*, 28(6):1098–1104, 1990.

[21] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *nature*, 393(6684):440, 1998.

[22] Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 138–148, 1990.

[23] Comandur Seshadhri, Ali Pinar, and Tamara G Kolda. Triadic measures on graphs: The power of wedge sampling. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 10–18, 2013.

[24] Charalampos E Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *2008 Eighth IEEE International Conference on Data Mining*, pages 608–617, 2008.

[25] Rasmus Pagh and Charalampos E Tsourakakis. Colorful triangle counting and a MapReduce implementation. *Information Processing Letters*, 112(7):277–281, 2012.

[26] A. Azad, A. Buluc, and J. Gilbert. Parallel triangle counting and enumeration using matrix algebra. *IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 804–811, 2015.

[27] M. M. Wolf, J. W. Berry, and D. T. Stark. A task-based linear algebra building blocks approach for scalable graph analytics. *IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, 2015.

[28] Ulrik Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177, 2001.

[29] T. Talvitie, K. Kangas, T. Niinimäki, and M. Koivisto. Counting linear extensions in practice: MCMC versus exponential Monte Carlo. In *Proc. of Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[30] David Eppstein and Joseph Wang. Fast approximation of centrality. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 228–229, 2001.

[31] U. Kang, Spiros Papadimitriou, Jimeng Sun, and Hanghang Tong. Centralities in large networks: Algorithms and observations. In *SDM*, 2011.

[32] Hang Liu, H. Howie Huang, and Yang Hu. iBFS: Concurrent breadth-first search on GPUs. In *Proceedings of the 2016 International Conference on Management of Data*, pages 403–416. ACM, 2016.

[33] B. Awerbuch and R. Gallager. A new distributed algorithm to find breadth first search trees. *IEEE Transactions on Information Theory*, 33(3):315–322, 1987.

[34] H. Gazit and J. Reif. A randomized parallel algorithm for planar graph isomorphism. In *Proceedings of the Second Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 210–219, 1990.

[35] E. Upfal and M. Mitzenmacher. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[36] Y. Mětivier and N. Saheb. Probabilistic analysis of an election algorithm in a tree. *CAAP Proceedings of the 19th International Colloquium on Trees in Algebra and Programming*, 787 of Lecture Notes in Computer Science:234–245, 1994.

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

[38] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection, 2014.

[39] Roger Pearce. Triangle counting for scale-free graphs at scale in distributed memory. In *High Performance Extreme Computing Conference (HPEC)*, pages 1–4, 2017.

[40] David Bader, Andrea Kappes, Henning Meyerhenke, Peter Sanders, Christian Schulz, and Dorothea Wagner. Benchmarking for graph clustering and partitioning. In *Encyclopedia of Social Network Analysis and Mining*, pages 73—-82. Springer, 2014.

[41] Manuel Then, Moritz Kaufmann, Fernando Chirigati, Tuan-Anh Hoang-Vu, Kien Pham, Alfons Kemper, Thomas Neumann, and Huy T Vo. The more the merrier: Efficient multi-source graph traversal. *Proceedings of the VLDB Endowment*, 8(4):449–460, 2014.

[42] J. Kunegis. Konect: The Koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web Companion*, pages 1343–1350. International World Wide Web Conferences Steering Committee, 2013.