Towards the computation of motif-based centrality measures over real-world networks

Sebastián Bugedo
PUC & IMFD
bugedo@uc.cl

Cristian Riveros
PUC & IMFD
cristian.riveros@uc.cl

Jorge Salas
PUC & IMFD & UoE
jusalas@uc.cl

ABSTRACT

Centrality measures have been proven to be a valuable tool for analyzing graph data. Some novel applications of centrality regarding graph databases, such as knowledge graphs or semantic web search engines, have been proposed lately. In this area, motif-based centrality measures have surged as a promising approach for understanding centrality measures in graph databases and for defining a family of new measures with desirable theoretical properties. However, these centrality measures are hard to compute in the general case (i.e., #P-hard). Consequently, until now, there are no experimental studies of such measures in real-world networks.

In this paper, we embark on computing motif-based centrality measures and understand how they behaved over real-world graphs. Towards this goal, we start by understanding whether these measures can be approximated in some way. Thus, we present new theoretical results confirming that these motif-based measures are really hard to compute and approximate. For this reason, we pursue a different approach by using advanced algorithmic techniques and parallelization to overpass the aforementioned complexity results in a practical setting. Specifically, we present an empirical fixed-parameter algorithm that runs in linear time over the graph size but exponentially in its treewidth. Moreover, we show how to parallelize the algorithm over a multi-core architecture by taking advantage of its modular structure and tree decompositions of graphs. Interestingly, we present a first implementation of this algorithm that can compute motif-based centralities over graphs with hudrends or thousands of nodes and with reasonable treewidth. Finally, we use these computations to compare such measures with other well-established centrality scores over real-world graphs. Altogether, this work constitutes the first practical study of motif-based centrality measures, settling the bases for further research and applications in graph databases.

1 INTRODUCTION

Network science considers *centrality measures* one of the critical metrics for graph data analysis. Roughly speaking, centrality measures like PageRank [42], Closeness [49], or Betweenness [24] are used for pinpointing the most influential nodes in a graph. Some examples of how network scientists use these measures are finding people who are more likely to spread a disease in the event of a pandemic [21], highlighting cancer genes in proteomic data [33], or identifying influencers in a social network [32].

With the advent of graph data management, it is natural to think that centrality measures have found their place in graph data management systems (Graph DBMS). Major Graph DBMS, like Neo4j [1] or TigerGraph [6], have already incorporated well-known measures in their data science library, where one can run centrality measures like PageRank and Harmonic, among others. Lately, new applications for centrality measures have emerged over knowledge graphs for entity linking [38] or semantic web search engines where ranking results is a core task [30]. Despite these applications, it is a long step from graph data analysis to core implementations in Graph DBMS. Nevertheless, centrality measures are currently an emergent and relevant research topic for these systems.

One way to define a node centrality is by measuring how many different *networks motifs* around the node are. Here, a network motif means any recurrent and statistically significant subgraph (e.g., triangles) which models some specific interactions inside graphs [31]. Then one could expect that the more motifs surrounding a node are, the more important (or central) such node is [36]. Several centrality measures have taken this approach, called here *motif-based centrality measures*, by using motifs like paths [54] or cliques [22] to measure the relevance of a node inside a graph. Indeed, motif-based centrality measures are of particular interest for Graph DBMS, given that one can think that motifs are no other thing but potential graph patterns over the database [9]. Under this scope, motif-based measures assign importance to a node as long as it belongs to more relevant query answers, and, hence, one can think of these measures as a reasonable approach for Graph DBMSs.

Recently, Riveros et al. presented a formal framework in [45] for studying motif-based centrality measures. Specifically, they proved that motif-based centrality measures could satisfy some desirable theoretical properties, called axioms, under specific properties over the family of motifs you choose. These results led to finding two new centrality measures that used the motifs of all possible subgraphs or trees, called All-Subgraphs and All-Trees, respectively (see Section 2 for a formal definition). Both centrality measures are of particular interest since they are the only motif-based measures that satisfy all axioms in [45] (see also [12]). Thus, they have desirable theoretical properties for graph data and rise as prominent candidates to include inside a Graph DBMS.

Despite their excellent theoretical properties, computational complexity is the primary concern when implementing motifs-based measures. It is well-known that, in general, motifs applications do not go beyond motifs of size four, even in small to medium size networks. Indeed, in [45] they showed that All-Subgraphs or All-Trees centrality measures are hard to compute in the general case (i.e., #P-hard), decreasing their potential used in practical applications and restricting them to only theoretical interest. Consequently, there is a substantial experimental gap in studies of motif-based centrality measures and how they relate to other well-known centralities from the literature. In particular, until now, there have been no implementation or experimental studies of such measures in real-world graphs.

This work aims to fill this gap by computing motif-based centrality measures, like All-Subgraphs or All-Trees, and understand

how they behaved over real-world graphs. From [45], we know this task is challenging regarding computational complexity. Then the purpose is to find a way to scale from small to medium size graphs to provide some preliminary evidence on how these centrality measures behave in practice. We expect this work to be a foundational experimental study of motif-based centrality measures that can lead to future implementations of them for Graph DBMSs.

Contributions. Our contributions are as follows.

(1) We start by better understanding how difficult it is to compute motif-based measures and whether we can approximate them somehow. First, we extend the complexity results from [45] regarding counting problems for the motif family of paths and cycles. As expected, these families are still #P-hard problems to count. Moreover, we show that these problems are even hard to approximate. Instead, for the case of All-Subgraphs and All-Trees, we extend the results from [45] in terms of computing the ranking, showing that even deciding the relative ranking between two nodes is as hard as computing the centrality itself.

(2) Given that these measures are hard to compute, we embark on direct algorithms for computing subgraph motifs centrality. The main technical result of the paper is two practical linear-time algorithms for computing the All-Subgraphs and All-Trees centralities, respectively, over graphs with bounded treewidth. Notice that [45] showed a theoretical algorithm for computing All-Subgraphs that runs in *quadratic time* over bounded treewidth graphs. Here, we improve this result by proposing two practical algorithms, one for All-Subgraph and one for All-Trees, that run in *linear time* in the graph size. Moreover, we show how to parallelize these algorithms over a multi-core architecture by exploiting its modular structure and the tree decomposition of the graph.

(3) We present the first implementation of both algorithms and their parallelized version. Using this implementation, we can compute All-Subgraphs and All-Trees centralities over more than 400 real-life graphs from the Network Repository [48]. Indeed, a naive implementation (i.e., by counting all subgraphs or all trees) cannot scale to graphs with more than 15 nodes. Here, we can rise to hundreds, even thousands of nodes using the parallelized approach. Further, we show that by using parallelism, we can outperform the sequential method in one order of magnitude, allowing us to scale to more significant graphs.

(4) We used these computations to compare commonly known centrality measures against All-Subgraphs and All-Trees. We used a diverse dataset of more than 100 graphs and compared them concerning the top-k nodes, similarity, and other statistical scores. From this comparison, we can empirically prove that over real-world graphs, All-Subgraphs and All-Trees could coincide with some standard centrality measures (e.g., Degree and PageRank), and considerably differ from others (e.g., Closeness). Still, they can also vary drastically in some graphs to all other measures, which shows that, over real-world graphs, they provide a different perspective of its centrality.

Related work. We can synthesize this work on two main objectives: to compute All-Subgraphs and All-Trees centrality in networks of medium or large size and use these computations to compare them with commonly used centrality scores. Regarding the computation of motif-based centrality measures, theoretical work usually aims to axiomatize a particular measure by the properties (axioms) it satisfies [13, 55, 61]. This direction was taken over All-Subgraphs and All-Trees in [45] with complexity analysis of computing these scores. However, it is unclear how the complexity of counting changes according to the different motifs one may consider, such as lines and cliques, and how difficult it is to compute its ranking.

Regarding our second goal, empirical studies of centrality measures have been widely done in different contexts [7, 34, 62]. Some practical studies compare centrality measures by directly analyzing the values assigned to each node. This is the case of Pearson correlation analysis [29, 37, 41]. On the other hand, people have compared centrality measures such as similarity scores [27, 37] or ranking correlations [51]. In the latter case, centrality values are not relevant but the order of relevance of the vertices in the network. Additionally, regardless of the case, just a few real-world networks or several randomly generated ones are usually used for the analyses [29, 37]. Besides including both kinds of empirical comparisons, this work will consider the most extensive possible set of networks from the Network Repository [48] where All-Subgraphs and All-Trees centrality can be computed. Hence, this work constitutes the first approach for understanding the empirical relation between All-Subgraphs, All-Trees, and commonly used centrality scores.

Outline of the paper. Section 2 gives the main definitions used in the paper. Section 3 presents the complexity analysis regarding computing motif-based centrality, and Section 4 presents the main algorithms. In Section 5, we discuss the implementation details and show an analysis of its performance. Section 6 compares motif-based centralities and other standard measures over real-world graphs. We finish with some concluding remarks in Section 7.

Because of space limitations, certain details, formal statements and proofs are deferred to the online appendix [4].

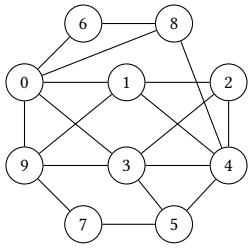
2 GRAPHS AND CENTRALITIES

Graphs. A graph (also called a network) is a pair $G = (V, E)$ where V is the set of vertices (also called nodes) and E is the set of edges (also called connections) of the form $\{u, v\}$ with $u, v \in V$ and $u \neq v$. Although we consider undirected graphs, we will usually write (u, v) instead of $\{u, v\}$ to indicate an undirected connection between u and v. We define $|G| = |V| + |E|$. Given a graph $G = (V, E)$ we use $V(G)$ and $E(G)$ for indicating the sets V and E, respectively.

We say that a graph G' is a subgraph of G, denoted $G' \subseteq G$, if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. Given two graphs G_1 and G_2, the union of G_1 and G_2, denoted by $G_1 \cup G_2$, is the union of its nodes and connections, namely, $V(G_1 \cup G_2) = V(G_1) \cup V(G_2)$ and $E(G_1 \cup G_2) = E(G_1) \cup E(G_2)$. If two graphs G and G' are isomorphic, we write $G \cong G'$.

A path in a graph G is a sequence of nodes $\pi = v_0, \ldots, v_n$ such that $(v_i, v_{i+1}) \in E(G)$ for every $i < n$ and we say that the length of π is n. Note that v_0 is the trivial path from v_0 to itself of length 0. We say that G is connected if there exists a path between any pair of vertices.

Motif-based centrality measures. We consider centrality measures based on network motifs as in [46]. A motifs family is a set of connected graphs $\mathcal{F}$ closed under isomorphism (i.e., if $G \in \mathcal{F}$ and $G \cong G'$ then $G' \in \mathcal{F}$). For example, some common motifs families used in practices are triangles, paths, cycles, etc. Given a



Rank	$C_{\mathcal{A}}$	$C_{\mathcal{T}}$	$C_{\mathcal{P}}$	$C_{\mathcal{Y}}$
1^{st}	3	0	4	4
2^{nd}	0	4	0	3
3^{rd}	4	3	3	0
4^{th}	1	9	9	9
5^{th}	9	1	1	1

Figure 1: A graph with ten nodes and the first five positions of the ranking induced by All-Subgraphs ($C_{\mathcal{A}}$), All-Trees ($C_{\mathcal{T}}$), All-Paths ($C_{\mathcal{P}}$), and All-Cycles ($C_{\mathcal{Y}}$) centralities.

motifs family $\mathcal{F}$ we define the $\mathcal{F}$-subgraph centrality, denoted by $C_{\mathcal{F}}(v, G)$, as:

$$C_{\mathcal{F}}(v, G) \;:=\; |\{S \subseteq G \mid v \in V(S) \text{ and } S \in \mathcal{F}\}|$$

for any graph G and vertex $v \in V(G)$[1]. In other words, $C_{\mathcal{F}}(v, G)$ measures the number of $\mathcal{F}$-motifs appearance containing v. The value $C_{\mathcal{F}}(v, G)$ measures how relevant v is in G and induces a ranking $<_{\mathcal{F}}$ between nodes: we write $u <_{\mathcal{F}} v$ if, and only if, $C_{\mathcal{F}}(u, G) < C_{\mathcal{F}}(v, G)$. Note that nodes with the same centrality are incomparable, and we assume they are ranked in an arbitrary but consistent order (e.g., lexicographic).

In general, we can consider any family of graph $\mathcal{F}$ that could be relevant for the user. In this paper, we restrict to these four families of graphs: we denote by $\mathcal{A}$, $\mathcal{T}$, $\mathcal{P}$, and $\mathcal{Y}$ the families of all *connected graphs*, *trees*, *paths*, and *cycles*, respectively. We denote their corresponding centrality measures by $C_{\mathcal{A}}$, $C_{\mathcal{T}}$, $C_{\mathcal{P}}$, and $C_{\mathcal{Y}}$, and call them as *All-Subgraphs*, *All-Trees*, *All-Paths*, and *All-Cycles* centrality, respectively.

Example 2.1. In Figure 1, we show a sample graph and the five most central nodes according to each of the previously defined centrality measures. This graph shows that no two rankings are similar for these measures, even though they include the same nodes in the top five. Thus, different motifs-families can induce a different node ranking.

Notice that although we took the framework of motif-based centrality measures from [46], some specific centrality measures based on graph families have been studied before. For example, subpaths centrality is the natural extension of Stress centrality [54], which considers all possible paths crossing a node instead of counting the shortest paths. Similarly, subcliques centrality was introduced in [22] by the name of Cross-Clique connectivity.

In [46], it was shown that All-Subgraphs and All-Trees satisfy most of the axioms (i.e., properties) proposed in the literature and that one would expect that a centrality measure should satisfy. In particular, they satisfy monotonicity, ranked monotonicity, and size axioms [13], among others. The main disadvantage of these measures is that they are hard to compute. Specifically, calculating the centrality value of a node is #P-complete. In the next section, we give new results regarding the hardness of computing these measures that will guide our decision to follow a more practical and pragmatic approach.

[1] In [46], the $\mathcal{F}$-subgraph centrality is defined with $\log_2(\cdot)$. To simplify the computation of the centrality measure, we omit here the log-function. This modification preserves the ranking of the centrality measure and just changes the scale.

3 MOTIFS-BASED CENTRALITIES ARE HARD

In this section we show that computing motif-based centrality measures is hard, and cannot be approximated. Formally, given a motifs family $\mathcal{F}$, we consider the following computational problem:

Problem:	$\textsc{Count}[\mathcal{F}]$
Input:	A graph G and a node $v \in V(G)$
Output:	$C_{\mathcal{F}}(v, G)$

In [46] it was shown that $\textsc{Count}[\mathcal{F}]$ is a #P-complete problem for the families $\mathcal{A}$ (All-Subgraphs) and $\mathcal{T}$ (All-Trees). #P-complete problems are a class of counting problems (i.e., the output of an instance is an integer), they are the analog of NP-complete problems for counting, and it is generally assumed to be very hard to solve them (e.g., a polynomial-time algorithm would imply a polynomial-time algorithm for any problem on the polynomial-time hierarchy, including NP-complete problems). In particular, it is generally believed that #P-complete problems cannot be solved in polynomial-time. Unfortunately, we can extend this result and show that for all motif families considered in this work, its corresponding computational problem is hard.

Theorem 3.1. *$\textsc{Count}[\mathcal{F}]$ is #P-complete when $\mathcal{F}$ is the family motifs of connected graphs $\mathcal{A}$, trees $\mathcal{T}$, paths $\mathcal{P}$, or cycles $\mathcal{Y}$.*

The negative result above forbid any polynomial-time algorithm for computing natural motif-based centrality measures. However, it does not neglect the existence of approximation algorithms for estimating $\textsc{Count}[\mathcal{F}]$, or algorithms for computing the ranking induced by $C_{\mathcal{F}}$. Next, we study the computational complexity of both approaches.

Hardness of approximation. A fully-polinomial time randomized approximation scheme (FPRAS) for $\textsc{Count}[\mathcal{F}]$ is a randomized algorithm A, that receives as input a pair (G, v) and parameters ϵ and δ, runs in polynomial-time in $|G|$, $\frac{1}{\epsilon}$, and $\log(\frac{1}{\delta})$, and outputs an estimate W of $C_{\mathcal{F}}(v, G)$ such that:

$$\mathrm{P}\big(\big|C_{\mathcal{F}}(v, G) - W\big| > \epsilon \cdot C_{\mathcal{F}}(v, G)\big) \leq \delta$$

That is, the probability that W deviates from $C_{\mathcal{F}}(v, G)$ more than ϵ (i.e., with multiplicative error) is at most δ. Of course, the more accurancy ϵ and less error probability δ, the more time is needed for the algorithm.

An FPRAS is a desirable solution for #P-complete problems, and it could be the best that we can achieve for computing $C_{\mathcal{F}}(v, G)$. Unfortunately, in the next result we show that for families $\mathcal{P}$ (All-Paths) and $\mathcal{Y}$ (All-Cycles), such solutions do not exist under certain reasonable complexity assumptions.

Theorem 3.2. *Unless NP = BPP, the problem $\textsc{Count}[\mathcal{F}]$ does not admit an FPRAS, when $\mathcal{F}$ is the family of paths $\mathcal{P}$ or cycles $\mathcal{Y}$.*

We leave open whether $\textsc{Count}[\mathcal{F}]$ admit an FPRAS when we consider the families of connected graphs $\mathcal{A}$ and trees $\mathcal{T}$. However, we believe that an FPRAS for $\mathcal{A}$ or $\mathcal{T}$ does not exist, given that the previous result provides evidence that this is the case for some simpler motifs families.

Hardness of ranking computation. Although computing or approximating the centrality $C_{\mathcal{F}}$ for a given motifs family is hard, this does not mean that generating a ranking of the nodes of a graph according to $C_{\mathcal{F}}$ is hard. Indeed, for certain graph data analysis it could be enough to decide who are the most central nodes, or top-k most central nodes in the graph (i.e., without knowing the precise value of the centrality).

To formalize the problem of rank the nodes with $C_{\mathcal{F}}$, we consider the following decision problem for a motifs family $\mathcal{F}$:

Problem:	RANK$[\mathcal{F}]$
Input:	A graph G and nodes $u, v \in V(G)$
Output:	`true` if, and only if, $u <_{\mathcal{F}} v$

The decision problem RANK$[\mathcal{F}]$ formalizes the computational problem of computing the rank for $<_{\mathcal{F}}$. Indeed, if we have an algorithm for sorting the nodes of a graph given $<_{\mathcal{F}}$, then we can solve RANK$[\mathcal{F}]$ and vice versa. Note that for solving RANK$[\mathcal{F}]$ we do not necessarily need to compute $C_{\mathcal{F}}$. Then it could be possible that deciding RANK$[\mathcal{F}]$ is tractable but computing $C_{\mathcal{F}}$ is hard. Unfortunately, as the following result shows this is not case for the motifs families of connected graphs $\mathcal{A}$ and trees $\mathcal{T}$.

THEOREM 3.3. *If $\mathcal{F}$ is the motifs family of connected graphs $\mathcal{A}$ or trees $\mathcal{T}$, then it holds that* COUNT$[\mathcal{F}] \in P^{\text{RANK}[\mathcal{F}]}$, *that is,* COUNT$[\mathcal{F}]$ *can be solved in polynomial-time with an oracle for* RANK$[\mathcal{F}]$.

Given that COUNT$[\mathcal{F}]$ is #P-complete, then by Toda's theorem [56], a polynomial-time algorithm for RANK$[\mathcal{F}]$ would imply the collapse of the polynomial-time hierarchy to P, which people widely believe that this is not the case. In other words, the previous results proves that deciding RANK$[\mathcal{F}]$ is as difficult as computing COUNT$[\mathcal{F}]$ for the families of connected graphs or trees. We leave open whether the same result holds for paths or cycles.

Theorems 3.1, 3.2, and 3.3 give strong evidence that we are doomed to fail if we want to compute directly, approximate, or even rank $C_{\mathcal{F}}$ for several reasonable motif families like connected graphs, trees, or paths. Despite these negative results, in the following, we pursue a different approach by using advanced algorithmic techniques and parallelization for computing All-Subgraphs or All-Trees centrality measures. In particular, these will allow us to compute them over networks of reasonable sizes and compare them with other centrality measures over real-world networks.

4 ALGORITHMS FOR COMPUTING MOTIF-BASED CENTRALITY MEASURES

Given the previous results, we embark on the search for direct algorithms for computing motif-based centrality measures. In [46], they showed that computing All-Subgraphs $C_{\mathcal{A}}$ for a given graph is fixed-parameter tractable and they provided a theoretical algorithm for computing $C_{\mathcal{A}}(v, G)$ in time $O(B(\text{tw}(G))^k \cdot |G|)$ for some fix k where $\text{tw}(G)$ is the treewidth of G and $B(n)$ is the n-th Bell number (i.e., the number of partitions of a set of size n). In particular, for computing $C_{\mathcal{A}}$ for all nodes, this algorithm takes time $O(B(\text{tw}(G))^k \cdot |G|^2)$. Here, we improve this algorithm by making it practical and improving it to linear-time, namely, $O(B(\text{tw}(G))^2 \cdot |G|)$ for computing $C_{\mathcal{A}}$ for all nodes. We also show how to generalize this approach for computing $C_{\mathcal{F}}$ when $\mathcal{F}$ are trees $\mathcal{T}$. To further scale this approach, we show how to parallelize the algorithm over a multi-core architecture by exploiting its modular structure and the tree decomposition of the graph.

In the following, we show the practical algorithm for computing All-Subgraphs $C_{\mathcal{A}}$ over a tree decomposition of a graph, to then show how to generalize this approach to All-Trees $C_{\mathcal{T}}$. Further, we show how to parallelize both algorithms over a tree decomposition. We start by introducing some notation.

Subpartitions and centrality. Fix a graph G and a set $D \subseteq V(G)$. We say that $P = \{A_1, \ldots, A_k\}$ is a *subpartition* of D if P forms a partition of a subset of D, namely, $\emptyset \neq A_i \subseteq D$ and $A_i \cap A_j = \emptyset$ for every $i \neq j$. We denote by $\text{Part}(D)$ the set of all subpartitions of D.

Intuitively, we will use $P = \{A_1, \ldots, A_k\}$ to count subgraphs that disjointly connect each A_i without connecting nodes excluded from D by P (i.e., nodes in $D \setminus \bigcup_i A_i$). Specifically, we generalize the definition of $C_{\mathcal{F}}$ from nodes to subpartitions of D as follows:

$$C^D_{\mathcal{F}}(P, G) = |\{ S_1 \uplus \cdots \uplus S_k \subseteq G \mid A_i \subseteq V(S_i), S_i \in \mathcal{F} \text{ and } (D \setminus \bigcup_i A_i) \cap V(S_i) = \emptyset \}|$$

where $S \uplus S'$ is the disjoint union of subgraphs $S, S' \subseteq G$ with $V(S) \cap V(S') = \emptyset$. In other words, $C^D_{\mathcal{F}}(P, G)$ counts subgraphs such that every component S_i covers some set from P and every set from P is covered in some component S_i. Further, every node $v \in D \setminus \bigcup_i A_i$ is excluded from $S_1 \uplus \cdots \uplus S_k$. Note that $C_{\mathcal{F}}(v, G)$ is a special case of $C_{\mathcal{F}}(P, G)$ if we consider $D = \{v\}$ and $P = \{\{v\}\}$.

There are several advantages of using subpartitions for computing $C_{\mathcal{A}}$ for the family of connected graphs $\mathcal{A}$. The first advantage is that we can compute the centrality of a node $v \in D$ from subpartitions of D as follows.

LEMMA 4.1. *For every graph G, set $D \subseteq V(G)$, and $v \in D$:*

$$C_{\mathcal{A}}(v, G) = \sum_{A \subseteq D : v \in A} C^D_{\mathcal{A}}(\{A\}, G)$$

where each $\{A\}$ is a subpartition of D.

Given the previous lemma, for computing $C_{\mathcal{A}}(v, G)$ we aim on computing $C^D_{\mathcal{A}}(\cdot, G)$ for some D such that $v \in D$.

The second advantage is that we can remove a node v from D and compute $C^{D'}_{\mathcal{A}}(\cdot, G)$ from $C^D_{\mathcal{A}}(\cdot, G)$ where $D' = D \setminus \{v\}$. Specifically, for computing $C^{D'}_{\mathcal{A}}(P, G)$ we can add all $C^D_{\mathcal{A}}(P', G)$ of subpartitions P' that lead to P when we remove v. Let $P - v = \{A \setminus \{v\} \mid A \in P\}$. Then we can compute $C^{D'}_{\mathcal{A}}$ from $C^D_{\mathcal{A}}$ by the following lemma.

LEMMA 4.2. *For every graph G, set $D \subseteq V(G)$, and $v \in D$:*

$$C^{D \setminus \{v\}}_{\mathcal{A}}(P, G) = \sum_{P' \in \text{Part}(D) : P = P' - v} C^D_{\mathcal{A}}(P', G)$$

As we can remove nodes from D, we can easily add nodes to D that were not initially in the graph. This fact brings us to the third advantage of using subpartitions. For a graph G and a node $v \notin V(G)$, let $G + v = (V(G) \cup \{v\}, E(G))$ be the graph of adding v to G (i.e., without connections).

LEMMA 4.3. *Let G be a graph, $D \subseteq V(G)$, and $v \notin V(G)$. For every $P \in \text{Part}(D)$, it holds that:*

$$C^{D \cup \{v\}}_{\mathcal{A}}(P, G + v) = C^D_{\mathcal{A}}(P, G)$$
$$C^{D \cup \{v\}}_{\mathcal{A}}(P \cup \{\{v\}\}, G + v) = C^D_{\mathcal{A}}(P, G)$$

In any other case, $C_{\mathcal{A}}^{D\cup\{v\}}(P', G+v) = 0$ *where* $P' \in \text{Part}(D\cup\{v\}) \setminus \text{Part}(D)$ *and* $\{v\} \notin P'$.

The last advantage of using subpartitions is that we can divide the computation of $C_{\mathcal{A}}^{D}(\cdot, G)$ when D *cuts* the graph in two disjoint subgraphs. For the next result, we need to first introduce some notation. We say that a pair of graphs (G_1, G_2) is a D-cut of G if $G = G_1 \cup G_2$, $V(G_1) \cap V(G_2) = D$, and $E(G_1) \cap E(G_2) = \emptyset$. In other words, a D-cut[2] is obtained by dividing G into two graphs G_1 and G_2 that only share the set of nodes D.

Given a D-cut (G_1, G_2) of G, we want to get $C_{\mathcal{A}}(P, G)$ by computing and combining $C_{\mathcal{A}}(P_1, G_1)$ and $C_{\mathcal{A}}(P_2, G_2)$ for all $P_1, P_2 \in \text{Part}(D)$. To combine subpartitions, the notion of *refinements* appears to be useful for this purpose. Specifically, define the following relation $\preceq$ between subpartitions: $P \preceq P'$ iff for every $A \in P$, there exists $B \in P'$ such that $A \subseteq B$. If $P \preceq P'$, we say that P *refines* P'. It is well-known that the refinement relation $\preceq$ defines a *lattice* between partitions and, also, over subpartitions (i.e., for every P and P' there exists a least upper bound with respect to $\preceq$). Then the least upper bound of P and P' with respect to $\preceq$ is always well-defined and we will denote it by $P \odot P'$. This notation leads us to the last lemma that shows how to divide the computation in the presence of a D-cut.

Lemma 4.4. *For every graph G, set $D \subseteq V(G)$, and a D-cut (G_1, G_2) of G, it holds that:*

$$C_{\mathcal{A}}^{D}(P, G) = \sum_{P_1, P_2 \in \text{Part}(D) : P = P_1 \odot P_2} C_{\mathcal{A}}^{D}(P_1, G_1) \cdot C_{\mathcal{A}}^{D}(P_2, G_2)$$

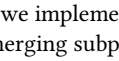
Hash tables of subpartitions. To handle subpartitions and their values $C_{\mathcal{A}}^{D}$ we use hash tables. Specifically, given a graph G and a set $D \subseteq V(G)$, a hash tables is a data structure H that maps subpartitions $P \in \text{Part}(D)$ to a number $H[P]$. We use the method $\texttt{keys}(H)$ to list all the subpartitions $P \in \text{Part}(D)$ such that $H[P] \neq 0$. We denote the empty hash table by $\emptyset$ where $\emptyset[P] = 0$ for all $P \in \text{Part}(D)$. Further, we write $H[P] \leftarrow n$ for inserting or updating P in H with the number n. By using lazy initialization, we can easily implement a hash table that starts initially empty and adds the entry for P whenever its value is non-zero. Given that H stores values only for elements in $\text{Part}(D)$, we use the method $\texttt{D}(H)$ to list the nodes in D, and write $v \in \texttt{D}(H)$ for checking if v is in D or not. By adopting the RAM computational model [8], we assume that each lookup and insertion to a hash table H takes constant-time, while the delay of iterating over the sets $\texttt{keys}(H)$ or $\texttt{D}(H)$ is constant.

Hash table operations. For dealing with subpartitions stored in hash tables, we implement Lemmas 4.2, 4.3, and 4.4 for forgetting, adding, or merging subpartitions, respectively. In Algorithm 1 we present these methods. The first method, Forget, removes a node $v \in \texttt{D}(H)$ from a hash table H. For this, starting from an empty hash table H', Forget follows Lemma 4.2 by iterating over each subpartition $P \in \texttt{keys}(H)$ (line 3), removing v from P (line 4), and adding it to H' (line 5). The second method, Add, receives a hash table H and node v, and adds v to $\texttt{D}(H)$. By Lemma 4.3, the method starts with an empty table H' and iterates over each non-zero subpartition $P \in \texttt{keys}(H)$ adding the value of $H[P]$ to $H'[P]$ and $H'[P \cup \{\{v\}\}]$ (lines 10-11). Finally, the Merge method implements the strategy of Lemma 4.4. This method receives two hash tables H_1 and H_2 storing $C_{\mathcal{A}}^{D}(\cdot, G_1)$ and $C_{\mathcal{A}}^{D}(\cdot, G_2)$, respectively, where (G_1, G_2) is a D-cut of G for some graphs G_1, G_2, and G. Then it computes $C_{\mathcal{A}}(\cdot, G)$ in the fresh hash table H' by iterating over partitions $P_1 \in \texttt{keys}(H_1)$ and $P_2 \in \texttt{keys}(H_2)$, calculating the least upper bound $P = P_1 \odot P_2$, and adding $H_1[P_1] \cdot H_2[P_2]$ to $H'[P]$. The reader can easily check that Forget, Add, and Merge correctly computes their corresponding operations over subpartitions.

In Algorithm 1 we include the last method Sync that will be useful in the main algorithm. Its purpose is to synchronize two hash tables H_1 and H_2 by making $\texttt{D}(H_1)$ equivalent to $\texttt{D}(H_2)$. For this, it copies H_1 into H' (line 22), forgets each node in $\texttt{D}(H_1)$ not included in $\texttt{D}(H_2)$ (lines 23-24), and adds each node in $\texttt{D}(H_2)$ not included in $\texttt{D}(H_1)$ (lines 25-26). It ends by returning H'. In the next subsections, we show an example of how we compute Sync and the other operations after we introduce tree decompositions.

Tree decomposition. The last and main ingredient for the main algorithm is a tree decomposition of a graph G which will help to decompose G into D-cuts. Formally, a *tree decomposition* of G is a triple $T = (V(T), E(T), \{G_x\}_{x\in V(T)})$ where $(V(T), E(T))$ is a tree, and $\{G_x\}_{x\in V(T)}$ is a set of subgraphs $G_x \subseteq G$ indexed by nodes $x \in V(T)$. Further, a tree decomposition must satisfy that (1) the union of $\{G_x\}_{x\in V(T)}$ is G (i.e., $G = \bigcup_{x\in V(T)} G_x$), (2) every pair G_x and G_y has disjoint set of connections (i.e., $E(G_x) \cap E(G_y) = \emptyset$ for each $x \neq y$), and the nodes $\{x \mid v \in V(G_x)\}$ of T forms a subtree in T for every $v \in V(G)$. The *treewidth* of a tree decomposition T is the maximum size of $V(G_x)$ for every $x \in V(T)$, formally, $\text{tw}(T) = \max_{x\in V(T)} |V(G_x)| - 1$. We will usually denote nodes of G by letters u, v, w and nodes of T by letters x, y, z. Also, we denote by G_x the subgraph of G corresponding to $x \in V(T)$.

Example 4.5. In Figure 2 (a) and (b) we provide a sample graph and one of its possible tree decompositions, respectively. Note that this tree decomposition has treewidth 2, since the biggest subgraph, G_y, has 3 nodes.

For the main algorithm, it will be convenient to introduce some additional notation. For a tree decomposition T, we will write $N_T(x)$ to denote the nodes adjacent to x in T. We say that x is a leaf of T if $|N_T(x)| = 1$. We say that T' is a *subtree* of T if $T' \subseteq T$ and T' is a tree. A subtree $T' \subseteq T$ naturally defines the subgraph $G(T') = \bigcup_{x\in V(T')} G_x$ of G, namely, the union of all subgraphs G_x. Given two adjacent nodes $x, y \in V(T)$, we denote by $T_{x,y}$ the maximum subtree of T that contains x and not y. For example, if $T =$  , then $T_{\circ,\bullet} =$ and $T_{\bullet,\circ} =$  . This notation is useful to define the subtrees hanging from a node $y \in V(T)$. Let $N_T(y) = \{x_1, \ldots, x_k\}$. Then one can easily check that $T_{x_1,y}, \ldots, T_{x_k,y}$ are all the subtrees hanging from y. We use these sequence of subtrees of y in the next lemma.

The advantage of a tree decomposition T is that we can process T in a bottom-up fashion in such a way that each step represents a D-cut of the graph. Formally, we have the following result.

Lemma 4.6. *Let $T = (V(T), E(T), \{G_x\}_{x\in V(T)})$ be a tree decomposition of a graph G. Let $y \in V(T)$ and $N_T(y) = \{x_1, \ldots, x_k\}$. Define the following sequence of graphs recursively: $G_0 := G_y$ and*

[2] In graph theory, a *cut* is a partition of $V(G)$ of a graph into two disjoint subsets. Here, we use a slightly different notion that betters fit our purpose.

Algorithm 1 Methods for managing hash tables of subpartitions where H, H_1, H_2 are hash tables and v is a vertex.

```
1:  procedure Forget(H, v)
2:    H' ← ∅
3:    for P ∈ keys(H) do
4:      P' ← {A \ {v} | A ∈ P}
5:      H'[P'] ← H'[P'] + H[P]
6:    return H'

7:  procedure Add(H, v)
8:    H' ← ∅
9:    for P ∈ keys(H) do
10:     H'[P] ← H[P]
11:     H'[P ∪ {{v}}] ← H[P]
12:   return H'

13: procedure Merge(H1, H2)
14:   H' ← ∅
15:   for P1 ∈ keys(H1) do
16:     for P2 ∈ keys(H2) do
17:       P ← P1 ⊙ P2
18:       H'[P] ← H'[P] +
19:                H1[P1] · H2[P2]
20:   return H'

21: procedure Sync(H1, H2)
22:   H' ← H1
23:   for v ∈ D(H1) \ D(H2) do
24:     H' ← Forget(H', v)
25:   for v ∈ D(H2) \ D(H1) do
26:     H' ← Add(H', v)
27:   return H'
```

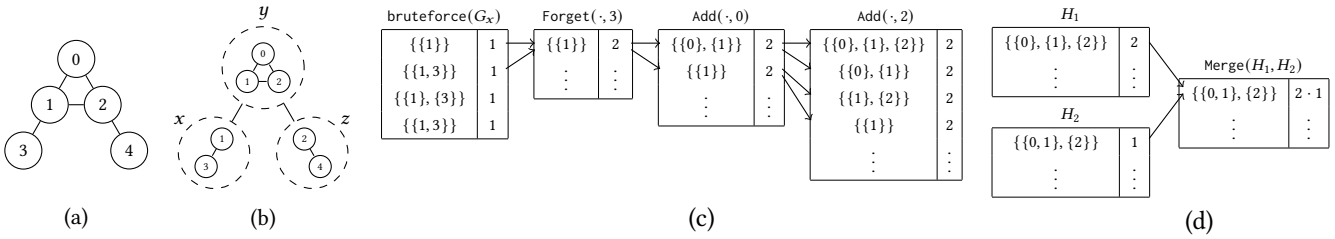


Figure 2: An example graph, a specific tree decomposition for it, and diagrams of the execution of methods from Algorithm 1.

$G_i := G(T_{x_i,y}) \cup G_{i-1}$. *For every* $i \leq k$ *and* $D = V(G_y)$, *the pair* $(G(T_{x_i,y}) \cup D, G_{i-1})$ *forms a* D*-cut of* G_i.

Lemma 4.6 is the reason behind using tree decompositions. Intuitively, given $y \in V(T)$ and $D = V(G_y)$ we can compute $C^D_{\mathcal{A}}(\cdot, G)$ by recursively computing it over $G(T_{x_1,y}), \ldots, G(T_{x_k,y})$ (i.e. probably over a different D) and then aggregate the values by applying Lemma 4.4 over each D-cut $(G(T_{x_i,y}) \cup D, G_{i-1})$. We exploit this intuition below in the main algorithm.

The main algorithm. In Algorithm 2 we show how to compute $C_{\mathcal{A}}(v, G)$ for all nodes $v \in V(G)$. The algorithm follows the usual strategy for problems on graphs of small treewidth [10, 17]. Specifically, find a tree decomposition T for G, compute a hash table of subpartitions for each node x of T by reusing the computation on the neighbors of x, and finally compute the centrality of each node v by using the hash tables. The algorithm's novelty resides in computing all such hash tables in one pass over the tree decomposition, taking linear-time over $|G|$ for computing the centrality of all nodes. In the following, we explain Algorithm 2 step-by-step to end with the analysis of its correctness and complexity.

The algorithm receives a graph G and the first step is to compute a tree decomposition $T = (V(T), E(T), \{G_x\}_{x \in V(T)})$ of G by calling the method `treedecomp`(G). Here we rely on a third-party library for computing tree decompositions of graphs with the smallest treewidth [11] (see Section 5 for implementation details). Then in lines 3-4 we compute, for every node $x \in V(T)$, the values $\{C^D_{\mathcal{A}}(P, G_x) \mid P \in \text{Part}(D)\}$ with $D = V(G_x)$ and store them in the hash table H_x. For this, we call the method `bruteforce`(G_x) that computes each value $C^D_{\mathcal{A}}(P, G_x)$ exhaustively (i.e., by brute force iterating over all possible combinations). Note that this is exponential in $|G_x| \leq \text{tw}(G)$, however, in practice this cost is small compared to the overall cost of the algorithm.

Example 4.7. For the example provided in Figure 2, the leftmost table of (c) shows the exact result of `bruteforce`(G_x) for the tree decomposition in (b). Since G_x has 2 connected nodes, there are exactly 4 subpartitions listed in this table.

The next goal is to compute $C^D_{\mathcal{A}}(\cdot, G)$ for each node y with $D = V(G_y)$ (lines 5-6). For this, we follow the strategy presented in Lemma 4.6: we decompose T into the subtrees $T_{x_1,y}, \ldots, T_{x_k,y}$ hanging from y, and compute the hash tables $H_{x_i,y}$ of $C^D_{\mathcal{A}}(\cdot, G[T_{x_i,y}])$. Then we compute the hash table storing $C^D_{\mathcal{A}}(\cdot, G)$ by iterating over the graphs $G_i := G(T_{x_i,y}) \cup G_{i-1}$ with $G_0 := G_y$. This follows a recursive strategy over each subtree $T_{x_i,y}$ and the method RecPartitions (Algorithm 2) is in charge for this task. Specifically, for each adjacent nodes y and z of T, we compute the hash table $H_{y,z}$ for the subtree $T_{y,z}$ storing $C^D_{\mathcal{A}}(\cdot, G[T_{y,z}])$ with $D = V(G_y)$. Note that $H_{y,z}$ represents $T_{y,z}$ on the node y. For reusing this notation over T and any node $x \in V(T)$, we use the hash table $H_{x,\perp}$, which stores $C^D_{\mathcal{A}}(\cdot, G)$ on $D = G_x$. With this notation, in lines 5-6 we iterate over each node $x \in V(T)$ and compute $H_{x,\perp}$ by calling RecPartitions$(x, \perp)$.

As it was already mentioned, a call to RecPartitions(y, z) follows Lemma 4.6. Fix $D = G_y$. First, this method makes $H_{y,z}$ equal to H_y, namely, $H_{y,z}$ stores the values $C^D_{\mathcal{A}}(\cdot, G_0)$ when $G_0 = G_y$. Next, for each neighbor x from y it adds the subgraph in $G(T_{x,y})$ to $H_{y,z}$. For this, let $x_i \in N_{T_{y,z}}(y)$ for the i-th iteration of the loop and let $G_i := G(T_{x_i,y}) \cup G_{i-1}$. Further, assume that $H_{y,z}$ stores the values $C^D_{\mathcal{A}}(\cdot, G_{i-1})$ before starting the i-th iteration. Then the algorithm computes $H_{x_i,y}$ recursively in lines 11-12 by calling RecPartitions(x_i, y). Here we use some dynamic programming to compute $H_{x_i,y}$ only once, by checking if $H_{x_i,y} = \emptyset$ or not. By Lemma 4.6, $(G(T_{x_i,y}) \cup D, G_{i-1})$ is a D-cut of G_i, however, it could be the case that $\mathtt{D}(H_{x_i,y}) \neq D = \mathtt{D}(H_{y,z})$. To sort this mismatch, we first call Sync$(H_{x,y}, H_{y,z})$, storing the temporary result in $H'_{x,y}$, to then call Merge$(H'_{x,y}, H_{y,z})$, satisfying the conditions by Lemma 4.6. After finishing the loop, the hash table $H_{y,z}$ contains all values $C^D_{\mathcal{A}}(\cdot, G(T_{y,z}))$.

Algorithm 2 Computation of all-subgraphs centrality $C_{\mathcal{A}}(v, G)$ for all nodes $v \in V(G)$ given a graph G.

```
1:  procedure SubgraphCentrality(G)
2:    (V(T), E(T), {G_x}) ← treedecomp(G)
3:    for x ∈ V(T) do
4:      H_x ← bruteforce(G_x)
5:    for x ∈ V(T) do
6:      RecPartitions(x, ⊥)
7:    return ComputeCentrality(G)
8:  procedure RecPartitions(y, z)
9:    H_{y,z} ← H_y
10:   for x ∈ N_{T_{y,z}}(y) do
11:     if H_{x,y} = ∅ then
12:       RecPartitions(x, y)
13:     H'_{x,y} ← Sync(H_{x,y}, H_{y,z})
14:     H_{y,z} ← Merge(H'_{x,y}, H_{y,z})
15: procedure ComputeCentrality(G)
16:   for v ∈ V(G) do
17:     let x ∈ V(T) such that v ∈ V(G_x)
18:     c_v ← 0
19:     for P ∈ keys(H_{x,⊥}) do
20:       if P = {A} and v ∈ A then
21:         c_v ← c_v + H_{x,⊥}[P]
22:   return {C_A(v, G) : c_v}_{v∈V(G)}
```

Example 4.8. For the example followed in Figure 2, a call of RecPartitions$(y, \perp)$ has its first iteration for node x in the tree decomposition. The recursive call in line 12 sets $H_{x,y}$ as the base values in the leftmost table of (c). The following steps of (c) show the call of Sync$(H_{x,y}, H_{y,z})$ and how it returns the modified hash table necessary for merging. We focus on how the values from subpartitions $\{\{1\}\}$ and $\{\{1,3\}\}$ combine in Forget$(\cdot, 3)$ and their resulting subpartition extends in Add$(\cdot, 0)$ and Add$(\cdot, 2)$. Note that only node 3 needs to be forgotten in the first step since it is the only exclusive node in x comparing to y. Also, nodes 0 and 2 need to be added in the second and third steps since they are present in y and not in x.

Now the call to Merge$(H'_{x,y}, H_{y,\perp})$ in line 12 combines the new hash table resulting in (c) with the base values from H_y, as shown in (d). Note that we focus on the resulting subpartitions $\{\{0,1\},\{2\}\}$.

The final step of Algorithm 2 is to calculate the centrality of each node in G from the hash tables $H_{x,\perp}$. Here we use the method called ComputeCentrality, which implements Lemma 4.1 for computing $C_{\mathcal{A}}(\cdot, G)$ from $C^D_{\mathcal{A}}(\cdot, G)$. The method passes over each $v \in V(G)$, find any node $x \in V(T)$ such that $v \in V(G_x)$ (line 17), and then computes $C_{\mathcal{A}}(v, G)$ from $H_{x,\perp}$ (lines 18-21). Specifically, it initialize $c_v = 0$ and, for each subpartition P with only one set A and $v \in A$, it adds the value $H_{x,\perp}(P)$. By Lemma 4.1, at the end of the loop c_v will be equal to $C_{\mathcal{A}}(v, G)$. Finally, we return the mapping $\{C_{\mathcal{A}}(v, G) : c_v\}_{v \in V(G)}$ that encodes the All-Subgraphs of G.

From the previous discussion, the correctness of Algorithm 2 for computing $C_{\mathcal{A}}$ follows. If we ignore the cost of computing the tree decomposition T, then one can check that the cost of iterating through $V(T)$ by calling methods `bruteforce` and RecPartitions takes time $O(B(\mathrm{tw}(T))^2 \cdot |V(T)|)$ where $B(n)$ is the number of partitions over n elements. The final method ComputeCentrality takes time $O(2^{\mathrm{tw}(T)} \cdot |V(G)|)$. Given that $|T| \leq |V(G)|$ and $2^{\mathrm{tw}(T)} \leq B(\mathrm{tw}(T))^2$ we get the following result.

Proposition 4.9. *If T is a tree decomposition for a graph G, then SubgraphCentrality(G) correctly computes $C_{\mathcal{A}}(\cdot, G)$ in time $O(B(\mathrm{tw}(T))^2 \cdot |V(G)|)$.*

Algorithm 2 has several advantages compared to the theoretical algorithm presented in [46]. First, the algorithm needs only a linear-time over $|G|$, compares to [46] which is quadratic. This is done by better understanding the computation of each node of the tree decomposition and reusing the partial results in RecPartitions. Second, the algorithm iterates only over non-zero subpartitions of the hash tables, which allows to skip some subpartitions in practice. Last, it helps to better understand this algorithmic approach for computing motif centrality by generalizing to other motifs, or having effective parallelization. In the following we discuss both extensions.

Extension to All-Trees. One advantage of the algorithm is that we can replicate its structure for other motifs. We present next how to extend the algorithm for computing the centrality based on trees.

To apply the strategy for other motifs family $\mathcal{F}$, one needs to reconsider the definition of $C^D_{\mathcal{F}}(\cdot, G)$, and the Lemmas 4.1 to 4.4. For the specific case of trees the definition of $C^D_{\mathcal{T}}(\cdot, G)$ and Lemmas 4.1 to 4.3 still hold. The only exception is Lemma 4.4 where the least upper bound $P_1 \odot P_2$ could not represent the merge of subpartitions P_1 and P_2 on trees. Intuitively, if two trees intersect at more than two nodes, the merge will produce cycles and the resulting motifs are not trees. For this reason, before computing the least upper bound between P_1 and P_2 we need to check that $|A_1 \cap A_2| \leq 1$ for every $A_1 \in P_1$ and $A_2 \in P_2$. We write $P_1 | P_2$ if this is the case.

Lemma 4.10. *For every graph G, set $D \subseteq V(G)$, and a D-cut (G_1, G_2) of G, it holds that:*

$$C^D_{\mathcal{T}}(P, G) = \sum_{\substack{P_1, P_2 \in \mathrm{Part}(D): \\ P = P_1 \odot P_2 \wedge P_1 | P_2}} C^D_{\mathcal{T}}(P_1, G_1) \cdot C^D_{\mathcal{T}}(P_2, G_2)$$

If we additionally check at the Merge method that $P_1 | P_2$ holds (i.e., before lines 17-19 in Algorithm 1) and update the `bruteforce` method for trees (i.e, line 4 in Algorithm 2), the main algorithm will work for computing $C_{\mathcal{T}}$. We call this extended algorithm TreesCentrality for which we get the following result.

Proposition 4.11. *If T is a tree decomposition for a graph G, then TreesCentrality(G) computes $C_{\mathcal{T}}(\cdot, G)$ in $O(B(\mathrm{tw}(T))^2 \cdot |V(G)|)$.*

Parallelization. Another advantage of the main algorithm is that we can effectively parallelize its computation by using a master-slave architecture. Specifically, the master unit processes the main thread of the algorithm, and the slave units perform some independent tasks, synchronizing only with the master. We use the master-slave architecture for computing Algorithm 2 as follows.

(1) When we compute `bruteforce`(G_x) for each node $x \in V(T)$ (lines 3-4), we assign the computation of each call to a different slave unit, running all calls in parallel given that the calculation is independent among them.
(2) For computing $H_{y,z}$ for each adjacent nodes y and z (lines 5-6) we can compute RecPartitions(y, z) for all adjacent pairs y, z non-recursively, starting from the leaves, and processing T in a bottom-up fashion. Specifically, we assign to

each slave unit the computation of all hash tables $H_{x,y}$ such that x is a leaf on T and y is the parent of x. Then we assign to a slave unit the computation of each hash table $H_{y,z}$ whenever all the hash tables $H_{x_1,y}, \ldots, H_{x_k,y}$ have been computed where $x_1, \ldots, x_k$ are all y neighbors (excluding z). Note that $H_{y,z}$ is independent of the computation of other hash tables (except its neighbors), and therefore they can be computed effectively in parallel.

(3) We parallelize the method ComputeCentrality (lines 15-22). Here, for computing the centrality $C_{\mathcal{A}}(v, G)$, we can perform the computation of each node v in a different slave unit independently.

We want to emphasize that the algorithmic approach proposed above allows for effectively parallelizing the three main phases of the algorithm. In particular, (1) and (3) admit a parallelization proportional to $V(T)$ and $V(G)$, respectively. Instead, (2) accepts a parallelization that depends on the branching of T. In the next section, we will show how this parallel version of the algorithm runs one order of magnitude faster than the serial version.

5 PERFORMANCE EXPERIMENTS

From Section 3 we conclude that calculating motifs-based centrality is problematic regarding computing or approximating the values or the ranking. Indeed, our first solution was to implement a naive approach, namely, counting all connected subgraphs directly for a node, to verify if these computational complexity results hold for real graphs. Unfortunately, our naive implementation could only scale to graphs up to 15 nodes, confirming the negative results and making this direction infeasible in practice.

This section aims to understand whether the alternative approach presented in Section 4 can break these pessimistic complexity results and scale to graphs with more than 15 nodes. For this, in this section we give the implementation aspects of our parallelized algorithmic approach, then how we organized the experimentation with real graphs data, to finish with a performance comparison between different configurations. Interestingly, we can show that our parallelized implementation can scale up to thousands of nodes over real graphs when the treewidth is bounded.

Implementation. We coded a non-parallelized implementation of Algorithm 2 and its corresponding parallelized version. We implemented both versions using Python 3.9.12 and made them publicly available in [4] under the GNU GPLv3 license. We used the *Networkx package* [3], version 2.7.1, for graph management and the computation of several classical graphs algorithms. For parallelization, we used the *Ray package* [5], version 1.11.0, which provided a simple and high-level interface to work with multiple cores in a single machine.

We obtained three different implementations of our algorithms: (1) a non-parallelized version for computing All-Subgraphs centrality for each node in a graph, (2) a parallelized version of the same algorithm, and (3) a parallelized version for computing All-Trees centrality. We only implemented the parallelized version for All-Trees since we were only interested in obtaining centrality values as fast as possible and not in developing a time performance comparison with a non-parallelized version.

For all versions, we followed the implementation of Algorithm 2 as presented in Section 4. The parallelized version has some structural differences from the non-parallelized counterpart, and we made these changes with the multi-core architecture in mind. One relevant implementation aspect was the construction of the tree decomposition of a graph. Several theoretical and practical algorithms exist for computing tree decompositions, but we decided to use the methods and heuristics provided in the Networkx library. It is important to note that these tree decompositions were not optimal in the sense of having bags whose size could be more significant than the optimal treewidth. Nevertheless, the resulting tree decompositions of Networkx were good enough for performance testing. Still, an optimal computation of tree decomposition could aid our algorithmic approach to scale to more significant networks. In the rest of this section, we will refer to the actual width of the bags obtained with the Networkx library simply as treewidth.

Graph data. For running performance experiments and comparing motifs centrality with other measures (see Section 6), we considered all graphs obtained from the *Network Repository* [48]. This website is an open and public repository of real-world graphs from different fields of study, including real-life applications and many well-known examples, with more than 6 thousand different networks [2]. Although we initially downloaded and considered all graphs, we could not study every graph by using our approach (e.g., performance constraints). Specifically, we filtered the complete set of the repository according to the following two parameters:

- *Graph size*: We noted that our approach could not scale for millions of edges based on preliminary computations done in a local environment. Further, graphs with few edges were not helpful for the comparison with other centrality measures. For this reason, we excluded all graphs with a size greater than 100MB or less than 20 nodes.
- *Connectedness*: For some centrality measures (e.g., PageRank, Closeness), the values and ranking of nodes between different connected components are incomparable. For this reason, we only considered the biggest connected component for each graph. Given that some networks in [2] had several components of comparable sizes (even of equal size), we decided to exclude those examples in which the biggest component represented less than a third of the total node size.

After these filters, we obtained 1702 potential graphs. Note that this set includes directed graphs, multigraphs, and weighted graphs. Given that we restricted this study to non-directed graphs, we removed the edge direction, weight, and multiple copies of edges of each graph whenever needed (in particular, no self-loops were considered).

Performance setup. For the performance experiments and centrality computations, we used a Supermicro Thinkmate cluster with 20 physical cores, 264 GB of RAM, using Ubuntu 18.04.5 LTS. The cluster uses the SLURM system, version 17.11.2, for scheduling. We devised two types of execution for the performance tests according to the implementations we described. First, we tested the non-parallel version of All-Subgraphs centrality, running it in a single core of the cluster. We used the same hardware to allow comparison with the parallelized version. We ran one execution for each

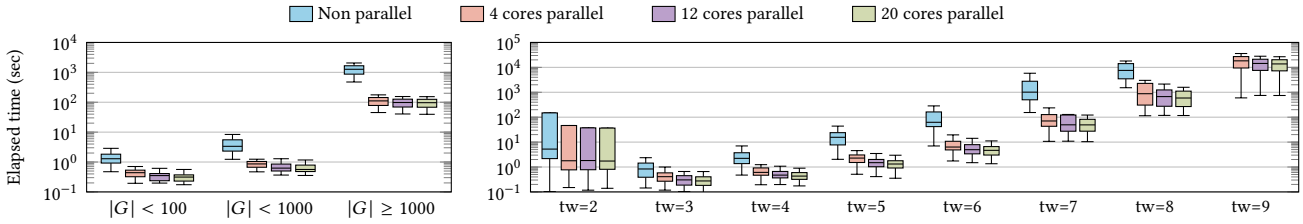


Figure 3: Execution times of the All-Subgraphs algorithms (i.e., parallel and non-parallel) for graphs of treewidth 5 by size (left) and for graphs of of any size by treewidth (right).

graph and registered its total time, as well as its centrality values to check consistency with the parallelized approach. Second, we tested the parallelized version of All-Subgraph centrality using different amounts of cores. For each graph, we ran one execution with 4, 12, and 20 cores in the same cluster node. As with the non-parallelized version, we registered the total time and centrality values to check consistency.

We ran both versions of the algorithm over the 1702 graphs of the Network Repository. Because many of those networks were too big to analyze, we defined a time threshold of 24 hours. This threshold implied that the parallelized version stopped for 413 graphs with treewidth up to 9; the rest of the graphs did not end in less than 24 hours. In the following table, we provide the number of instances (#i) that we computed for each treewidth value (tw).

tw	#i	tw	#i	tw	#i	tw	#i
2	17	4	112	6	46	8	20
3	98	5	88	7	25	9	3

Performance results. We start by analyzing how much it scales the algorithmic approach for real-world graphs of different sizes. Given that the algorithm's performance depends on the treewidth, we considered all networks of some fixed treewidth for this analysis. In Figure 3 (left), we display the refined view of the execution times for treewidth 5 instances for three size categories. Since our naive implementation of the basic recursive algorithm proposed in [46] was only able to reach graphs with at most 15 nodes, the possibility of computing graphs with hundreds or thousands of nodes shows an immediate improvement. This figure also shows that with increasing size of the graph (given by $|G| = |V| + |E|$), for fixed treewidth the performance of the parallelized version is at least one order of magnitude better than the non-parallelized version of the algorithm. Moreover, this shows that with the increasing amount of cores, there is a slight improvement in performance for the same parallel version. As described before, the tree decomposition structure impacts how much we can parallelize with multiple cores.

Next, we analyze how much it scales the algorithm when we vary the treewidth of different sizes. In Figure 3 (right), we provide execution times by treewidth of the graphs, comparing the four types of computations: non-parallel and parallel versions using 4, 12, and 20 cores in the cluster. For each treewidth category, it is clear that all parallelized versions outperform the non-parallelized approach, and this difference increases as the treewidth increase too. For treewidth 9, the non-parallel version did not complete in the 24-hour window we defined, hence the absence of the corresponding box. Therefore, the parallelized setting allows us to reach graphs with higher treewidth that we cannot compute in a reasonable time in a single core version. Finally, note that there is considerable dispersion of the values for treewidth 2 and 3. One can explain this fact by the more significant range of sizes for those categories, ranging from graphs with 20 to 5000 nodes.

We want to end this analysis by emphasizing that, without the algorithmic approach presented in Section 4, we could not scale to graphs with more than 15 nodes, as the complexity analysis of Section 3 predicted. Instead, we can scale to non-trivial real-world graphs with hundreds or thousands of nodes by exploiting the tree decomposition. As an example, we show in Figure 5 (a) the NetScience network ¨[40], where each node has a clearer shade when it has a higher ranking according to $C_{\mathcal{A}}$. This graph has 379 nodes and 914 connections. In the next section, we will use these centrality values over non-trivial graphs to understand how these motifs-based centralities behave in practice and how comparable they are with standard centrality measures.

6 BEHAVIOR OVER REAL-WORLD GRAPHS

With the computation of All-Subgraphs and All-Trees centralities over several real-world graphs, we can use this information to understand how they behave in practice, in particular, to know how similar these measures are, compared to established centrality measures. Indeed, it could be the case that, although All-Subgraphs and All-Trees are hard to compute, their values or rankings are similar to other centrality measures on real-world graphs. Thus, we could use their efficient algorithms as a good approximation for motif-based centralities. The main question is:

> *"How different are subgraph motif centralities compared to other centrality measures over real-world graphs?"*

The main goal of this section is to answer this question by analyzing and comparing All-Subgraphs and All-Trees among a set of commonly used centrality measures in practice. The centrality measures we will take into account are *Degree* [39], *PageRank* [42], *Closeness* [49], *Betweenness* [25], and *Harmonic* [47]. Although we could consider more centrality measures, one can argue that Degree, PageRank, Closeness, and Betweenness are well-established measures in the industry. Instead, Harmonic has been recently considered since it satisfies some desirable theoretical properties, similar to All-Subgraphs and All-Trees [13]. Given space restrictions, the reader can find the definitions of these measures in [4].

top-k	DG	PR	CL	BW	HM
1	12	20	26	16	9
2	6	10	11	7	5
3	6	6	5	3	2
4	4	5	5	3	1
5	3	2	3	2	1

top-k	DG	PR	CL	BW	HM	**AS**
1	10	16	25	10	9	25
2	5	8	10	5	2	9
3	5	4	5	0	1	3
4	4	3	5	0	0	3
5	3	2	3	0	0	1

top-k	DG	PR	CL	BW	HM	**TS**
1	11	20	25	13	8	58
2	4	8	10	7	3	32
3	4	6	5	3	1	20
4	1	5	5	3	0	15
5	1	2	3	2	0	10

Table 1: Each table shows the number of graphs where a measure C has a top-k discrepancy with the other measures on the table's header for $k = 1, \ldots, 5$. The first group (i.e., first table) includes Degree (DG), PageRank (PR), Closeness (CL), Betweenness (BW), Harmonic (HM). The second and third groups have the first group plus All-Subgraphs (AS) or All-Trees (TS), respectively.

We started by considering the 413 graphs from the Network Repository computed for All-Subgraphs and All-Trees (see Section 5). However, note that the Network Repository is divided into categories. When we inspected the category of the 413 graphs, we noticed that 104 were from Animal Networks, 210 were from Cheminformatics, 84 were from Miscellaneous Networks (the biggest collection in Network Repository), and the rest had less than five networks from other collections (e.g., Brain Networks, Collaboration Networks). The sample from Miscellaneous Networks looked very heterogeneous, from different contexts and shapes. Instead, we noticed that the Animals and Cheminformatics samples were very homogenous, with several graphs coming from the same source. Then, to break this bias, we randomly picked a few examples of the Animal and Cheminformatics collections to form a heterogeneous dataset of real-world graphs with the rest (i.e., from Miscellaneous Networks and others). After this filtering process, we reduced the dataset to 113 networks. Note that this collection still contains several networks considered by previous studies, like Les Miserables [35], Karate [64], NetScience [40], among others. The reader can find the complete list of 113 networks in the online appendix with their sizes, categories, and citations. It is important to note that, although our dataset is extensive and heterogeneous, it is not necessarily representative, and we should be careful in taking our conclusions. Despite this, we believe that the comparison over this dataset can shed light on answering our main question regarding motif-based centrality measures.

We divide the next analysis into three questions: (1) Are the top nodes outputted by All-Subgraphs and All-Trees different from other measures?, (2) Which centrality measures are more similar to All-Subgraphs or All-Trees?, and (3) How are the top nodes distributed inside a graph according to All-Subgraphs, All-Trees, and other measures?.

Are the top nodes different?. We aim to provide evidence that the top nodes outputted by All-Subgraphs and All-Trees are different from other measures over real-world networks. Specifically, there exist several graphs where All-Subgraphs and All-Trees choose different top-k nodes compared to other measures. Let top-k(C, G) be the set of the first k-nodes ranked by a centrality measure C over a graph G. Fix a group of centrality measures $C_1, \ldots, C_\ell$ and pick an arbitrary measure C from this group. For a fix k, we say there is a *top-k discrepancy* of C over a graph G if the set top-k(C, G) has no node in common with top-k(C_i, G) for every other measure C_i (i.e., $v \notin \bigcup_{C_i \neq C}$ top-k(C_i, G) for every $v \in$ top-k(C, G)). In particular, for $k = 1$ this means that the top-1 node found by C over G is different to all top-1 nodes found by all other measures over G. Note that top-k discrepancy means that C totally differs from the rest of the measures over an specific graph G. Then the purpose of this subsection is to identify real-world graphs where C totally differs with the other measures from the first k-nodes.

We use three different groups to verify *top-k discrepancy*: (a) all non-motif-based centralities, namely, Degree (DG), PageRank (PR), Closeness (CL), Betweenness (BW), Harmonic (HM), (b) all non-motif-based centralities plus All-Subgraphs (AS), and (c) all non-motif-based centralities plus All-Trees (TS). We use (a) as our baseline, to measure the discrepancy of the non-motif-based group before incorporating a motif-based centrality. For $k = 1, \ldots, 5$ and each centrality measure C we count the number of graphs (i.e., over the 113 graphs) where a top-k discrepancy is found. We execute this experiment over the three groups (a), (b), and (c). The results are summarized in Table 1.

Discussion. From Table 1, we can conclude that All-Subgraphs have a reasonable level of coincidence on the top nodes with other graphs. On the other side, All-Subgraphs is not "another measure", given that around 22% of its top-1 nodes are different from all other measures (2nd table). Indeed, this discrepancy is comparable to measures like PageRank or Closeness in the first group (1st table) which have the highest levels of top-1 discrepancy. Another conclusion is that there are graphs where the top-3 or even top-5 nodes are different for All-Subgraphs (2nd table). In other words, there exist real-world graphs where All-Subgraphs can give a totally different top ranking, where other measures like Betweenness or Harmonic provide no discrepancy (i.e., both mark 0 for top-5 discrepancy). Finally, the most remarkable result is All-Trees in the last table. While All-Subgraphs has similar values as the baseline (1st and 2nd table), All-Trees has a higher percentage of graphs where the most critical nodes are not in the top-k for every other centrality measure (e.g., 50% for top-1 discrepancy). In consequence, one can arguably say that other motifs, like trees, do matter in studying centrality measures. They can assign different levels of importance over real-world graphs where other measures do not.

Which centrality measures are more similar?. In contrast to the previous goal, now we want to know which measures are closest to All-Subgraphs or All-Trees. For this, we consider different statistical measures for comparing All-Subgraphs and All-Trees with the other measures.

We use fourth different statistical coefficients for comparing measures over a specific graph: Pearson, Spearman, Kendall's Tau, and similarity. First, we compute the *Pearson* coefficient [37] between

values $C_1(\cdot, G)$ and $C_2(\cdot, G)$ given centralities C_1 and C_2, and graph G. Also, we consider *Spearman* coefficient [52] and *Kendall's Tau* distance [16] of the ranking induced by C_1 and C_2 over G. In a nutshell, Pearson and Spearman represent the linear correlation over the values and rankings on a graph, respectively. Both coefficients range between -1 and 1, where 1 means correlation, -1 means inverse correlation, and 0 means no correlation. Instead, Kendall's Tau distance measure how many pairs of exchanges we should apply to transform one ranked list into another. The coefficient ranges between 0 and 1, where 0 means both measures give the same ranking. We also compute the *top-k similarity* [57] (or just similarity) of the rankings, define as $|\text{top-k}(C_1, G) \cap \text{top-k}(C_2, G)|/k$ for measures C_1 and C_2, and graph G. Given a fix k, the similarity range from 0 to 1, where 1 means that both centralities outputted the same top-k nodes (possibly in a different order). We increase k proportional to the total number of nodes, from 0% (i.e., the top-1 node) to 100% (i.e., all the graph). It is worth mentioning that Pearson, Spearman, Kendall's Tau, and similarity are standard statistical measures that people have used to compare the similitude between centrality measures in the past [18, 37, 51]. Given space restrictions, in the online appendix [4] we provide the formal definitions of these coefficients.

We compute the Pearson coefficient between All-Subgraphs and a centrality measure C over each graph in the dataset. For each C we draw a box plot with one point for each graph. Similarly, we construct the box plots for Spearman and Kendall's Tau coefficients. Instead, for top-k similarity, we want to see how it changes with k. In this case, we compute the mean value of similarity between All-Subgraphs and a centrality measure C over the dataset, and draw a dot-line for each C, varying k proportional to the total number of nodes, namely, $k = 1, 10\% \cdot |V(G)|, 20\% \cdot |V(G)|, \ldots, 100\% \cdot |V(G)|$. Finally, we repeat the All-Subgraphs' plot scheme with All-Trees centrality too. We display all these plots in Figure 4.

Discussion. The first conclusion is that all statistical coefficients agree that All-Subgraphs and All-Trees are more similar regarding values and ranking. Nevertheless, they do not necessarily give the same output. Indeed, All-Subgraphs versus All-Trees plots show several outliers, showing evidence that they can radically differ in some graphs. Despite these outliers, the mean value of all coefficients shows that both measures have a high level of coincidence.

Another conclusion is that the two non-motif-based centrality measures closer to All-Subgraphs and All-Trees are Degree and PageRank, with Degree having a slight advantage over PageRank. Interestingly, this means that the node degree highly influences the values for All-Subgraphs and All-Trees. This conclusion is somehow expected, given that the higher the degree, the more motifs are around. Again, this does not mean that these measures always coincide with Degree. As the reader can check, Figure 4 shows several outliers on the Degree plots, showing that there are graphs where All-Subgraphs or All-Trees could differ with Degree drastically. Similarly, although PageRank is closer to All-Subgraphs on average, it has higher dispersion in the box plots, with several outliers outside the whiskers. One can notice this more clearly in Pearson's box plots, where PageRank has several outliers, even with inverse correlation.

From Figure 4, one can also conclude that Closeness is the centrality measure that differs more with All-Subgraph and All-Trees. Indeed, this is the case in all plots. In particular, the k-similarity plot of All-Subgraphs shows that the line of Closeness is clearly below all other lines. An interesting fact is that Closeness differs a bit more with All-Subgraphs than with All-Trees. This provides evidence that different motif-based centrality could have different relations with other measures.

To finish, we want to highlight that all statistical coefficients show many real-world graphs where All-Subgraphs and All-Trees differ in values and rankings with other measures. Nevertheless, they also show that for many of them, they coincide. For instance, consider the Kendall's Tau box plots in Figure 4. Here, every measure has a small mean below 0.25 for Kendall's Tau distance, meaning rankings do not need many permutations to be equal to those defined by All-Subgraph and All-Trees. Moreover, for every measure, there exists a graph where they assign the same ranking as All-Subgraphs and All-Trees. Then we can conclude that, in practice, All-Subgraphs and All-Trees could coincide with the standard centrality measures. Still, they can also differ considerably, providing a different perspective of the centrality of a graph.

How are the top nodes distributed?. After computing subgraph motif centralities over several graphs, we started to note that both centralities use to distribute the most central nodes uniformly in the graphs, compared to other centrality measures. For example, in Figure 5 we display the *NetScience* network, which represents the co-authorship network among scientists working on network science [40]. Here, we show a heatmap for each centrality measure (excluding Harmonic for space reasons) over the NetScience network, where more intense-color nodes mean more central. As one can inspect, All-Subgraphs and All-Trees distribute the heat better (i.e., the top nodes), whereas other centralities do not. Given this example, the natural question is to understand how different measures distribute the top-k nodes inside a graph.

To the best of our knowledge, no previous work has studied this question regarding centrality measures. For this reason, we introduce the *k-dispersion* coefficient (or just dispersion) to measure how far are the top-k nodes distributed in a graph compared to its diameter. Formally, let $\text{dist}_G(u, v)$ be the distance between nodes u and v in a graph G (i.e., the length of a shortest path) and let $\text{diam}(G) = \max_{u,v \in V(G)} \text{dist}_G(u, v)$ be the diameter of G (i.e. the length of the longest path between any pair of nodes). Then we define the k-dispersion coefficient of C over G as:

$$\text{disp}_k(C, G) = \frac{\sum_{u,v \in \text{top-k}(C,G)} \text{dist}_G(u, v)}{\binom{k}{2} \cdot \text{diam}(G)}.$$

In other words, we sum all pair distances of top-k nodes of C over G, and normalized this sum by the number of top-k pairs times the diameter of G. Note that $\binom{k}{2} \cdot \text{diam}(G)$ is the maximum that the fraction's numerator can reach, and then $0 \leq \text{disp}_k(C, G) \leq 1$. One can easily check that $\text{disp}_k(C, G)$ is closer to 1 when the top-k are distributed further inside G.

We compute $\text{disp}_k(C, G)$ for $k = 10$ over all graphs in the dataset and for every centrality measure C. Then, we draw a box plot for each C with one point for each graph. In Figure 4 (right), we display these box plots for each centrality measure previously considered.

Discussion. In Figure 4 (right), we can distinguish between two extreme cases. On one side, Closeness is the less dispersed measure.

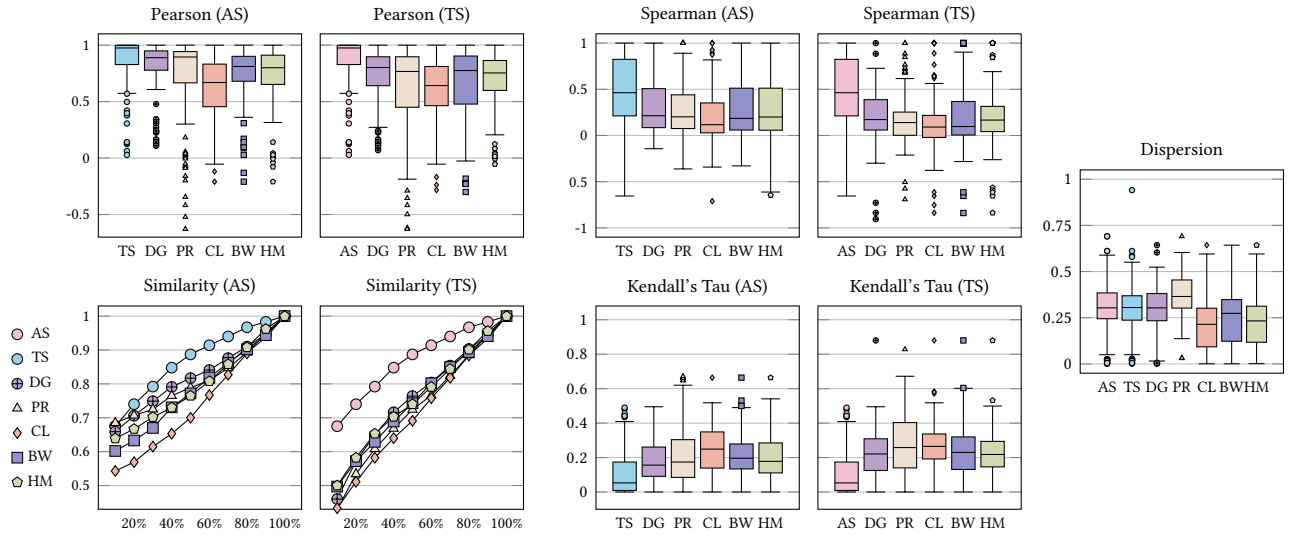


Figure 4: Box-plots for Pearson, Spearman, and Kendall's Tau coefficients, comparing All-Subgraphs (AS) or All-Trees (TS) versus Degree (DG), PageRank (PR), Closeness (CL), Betweenness (BW), and Harmonic (HM). On the left below, the k-similarity plots of All-Subgraphs and All-Trees against other measures, where k ranges from 0% (i.e., the top-1 node) to 100%. On the right middle, a box-plot for k-dispersions with $k = 10$, one box for each centrality measure.

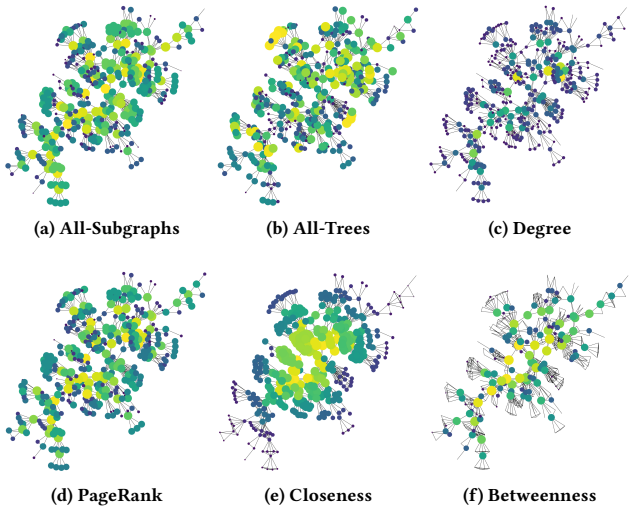


Figure 5: The NetScience graph [40] with a heatmap for each centrality measure (excluding Harmonic for space reasons), where more intense-color nodes mean more central.

This fact could be a consequence of Closeness' definition, given that it is a geometrical measure where central nodes tend to be together. Meanwhile, PageRank is the most dispersed centrality on average. Interestingly, All-Subgraphs and All-Trees are between these extremes but closer to PageRank. Another important conclusion is that Degree centrality has high dispersion (i.e., compared to the group) and similar to All-Subgraphs and All-Trees. One possible explanation of the first fact is that nodes with a higher degree are not necessarily forced to be together, which could be distributed uniformly in the graph. Furthermore, as we discussed previously, the degree tends to influence motif-based centralities, which could explain why the dispersion of All-Subgraphs, All-Trees, and Degree is similar. Finally, it is interesting to see that All-Subgraph and All-Trees have outliers where the dispersion is higher than all other measures. This last fact could probably explain our first impression regarding the distribution of both measures.

7 CONCLUSIONS

We provided a study on the complexity and a concrete implementation of algorithms for computing two motifs-based centrality measures. To the best of our knowledge, we showed the first linear time algorithm for computing All-Subgraphs and Tree-Subgraphs centrality in bounded treewidth networks. We proved that counting some families of graph motifs is a challenging problem, but also that computing the ranking is equally as hard and that it is highly unlikely that there is an approximation algorithm for solving this problem. This theoretical evidence substantiated our approach to designing an explicit algorithm based on tree decompositions.

We achieved a parallelized implementation of such algorithms and tested their performance and results over real-world networks. In doing so, we compared the performance for an increasing amount of CPU cores, revealing an objective improvement compared to the non-parallelized version. Also, since this is the first time we can generate centrality values for networks with more than 15 nodes, we compared the values and rankings obtained with these algorithms to those of well-known measures such as PageRank, Closeness, and Betweenness. We showed that in some real-world cases, All-Subgraphs and Tree-Subgraphs yield a different centrality profile than the other measures.

REFERENCES

[1] [n.d.]. Neo4j: Centrality algorithms. https://neo4j.com/docs/graph-data-science/current/algorithms/centrality/. Accessed on 2022-08-31.

[2] [n.d.]. The Network Repository Website. https://networkrepository.com/. Accessed on 2022-08-31.

[3] [n.d.]. NetworkX: Network Analysis in Python. https://networkx.org/. Accessed on 2022-08-31.

[4] [n.d.]. Online version and algorithms implementations repository. https://github.com/Motif-Based-Centralities. Accessed on 2022-08-31.

[5] [n.d.]. Ray. https://docs.ray.io/en/latest/. Accessed on 2022-08-31.

[6] [n.d.]. TigerGraph: Centrality algorithms. https://docs.tigergraph.com/graph-ml/current/centrality-algorithms/. Accessed on 2022-08-31.

[7] Taras Agryzkov, Leandro Tortosa, José F Vicent, and Richard Wilson. 2019. A centrality measure for urban networks based on the eigenvector centrality concept. *Environment and Planning B: Urban Analytics and City Science* 46, 4 (2019), 668–689.

[8] Alfred Aho, John Hopcroft, and Jeffrey Ullman. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

[9] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. 2017. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.* 50, 5 (2017), 68:1–68:40.

[10] Hans L Bodlaender. 1997. Treewidth: Algorithmic techniques and results. In *International Symposium on Mathematical Foundations of Computer Science*. Springer, 19–36.

[11] Hans L Bodlaender and Arie MCA Koster. 2010. Treewidth computations I. Upper bounds. *Information and Computation* 208, 3 (2010), 259–275.

[12] Paolo Boldi and Sebastiano Vigna. 2014. Axioms for centrality. *Internet Mathematics* 10, 3-4 (2014), 222–262.

[13] Paolo Boldi and Sebastiano Vigna. 2014. Axioms for centrality. *Internet Mathematics* 10, 3-4 (2014), 222–262.

[14] Gerald G Carter and Gerald S Wilkinson. 2013. Food sharing in vampire bats: reciprocal help predicts donations more than relatedness or harassment. *Proceedings of the Royal Society B: Biological Sciences* 280, 1753 (2013), 20122573.

[15] Caroline Casey, Isabelle Charrier, Nicolas Mathevon, and Colleen Reichmuth. 2015. Rival assessment among northern elephant seals: evidence of associative learning during male–male contests. *Royal Society open science* 2, 8 (2015), 150228.

[16] Vincent A Cicirello. 2019. Kendall tau sequence distance: Extending Kendall tau from ranks to sequences. *arXiv preprint arXiv:1905.02752* (2019).

[17] Bruno Courcelle and Joost Engelfriet. 2012. *Graph structure and monadic second-order logic: a language-theoretic approach*. Vol. 138. Cambridge University Press.

[18] Manuel Curado, Leandro Tortosa, Jose F Vicent, and Gevorg Yeghikyan. 2020. Analysis and comparison of centrality measures applied to urban networks with data. *Journal of Computational Science* 43 (2020), 101127.

[19] Stephen Davis, Babak Abbasi, Shrupa Shah, Sandra Telfer, and Mike Begon. 2015. Spatial analyses of wildlife contact networks. *Journal of the Royal Society Interface* 12, 102 (2015), 20141004.

[20] Shermin de Silva, Volker Schmid, and George Wittemyer. 2017. Fission–fusion processes weaken dominance networks of female Asian elephants in a productive habitat. *Behavioral Ecology* 28, 1 (2017), 243–252.

[21] Zoltán Dezső and Albert-László Barabási. 2002. Halting viruses in scale-free networks. *Phys. Rev. E* 65 (2002), 055103. Issue 5.

[22] Mohammad Reza Faghani and Uyen Trang Nguyen. 2013. A Study of XSS Worm Propagation and Detection Mechanisms in Online Social Networks. *IEEE Transactions on Information Forensics and Security* 8, 11 (2013), 1815–1826. https://doi.org/10.1109/TIFS.2013.2280884

[23] Mathias Franz, Jeanne Altmann, and Susan C Alberts. 2015. Knockouts of high-ranking males have limited impact on baboon social networks. *Current Zoology* 61, 1 (2015), 107–113.

[24] Linton C Freeman. [n.d.]. Centrality in Social Networks Conceptual Clarification. *Social Networks* 1, 1978/79 ([n. d.]), 215–239.

[25] Linton C Freeman. 1977. A set of measures of centrality based on betweenness. *Sociometry* (1977), 35–41.

[26] Francisco Damasceno Freitas, Joost Rommes, and Nelson Martins. 2008. Gramian-based reduction method applied to large sparse power system descriptor models. *IEEE Transactions on Power Systems* 23, 3 (2008), 1258–1270.

[27] Catarina Gouveia, Ágnes Móréh, and Ferenc Jordán. 2021. Combining centrality indices: maximizing the predictability of keystone species in food webs. *Ecological Indicators* 126 (2021), 107617.

[28] Randi H Griffin and Charles L Nunn. 2012. Community structure and the spread of infectious disease in primate social networks. *Evolutionary Ecology* 26, 4 (2012), 779–800.

[29] Xiaojia He, Natarajan Meghanathan, et al. 2016. ALTERNATIVES TO BETWEENNESS CENTRALITY: AMeasure OF CORRELATION COEFFICIENT. In *Proceedings of the Fifth Inernational Conference on Advanced Information Technologies and Application*. 1–10.

[30] Aidan Hogan, Andreas Harth, Jürgen Umbrich, Sheila Kinsella, Axel Polleres, and Stefan Decker. 2011. Searching and browsing linked data with swse: The semantic web search engine. *Journal of web semantics* 9, 4 (2011), 365–401.

[31] Paul W Holland and Samuel Leinhardt. 1974. The statistical analysis of local structure in social networks.

[32] Xinyu Huang, Dongming Chen, Dongqi Wang, and Tao Ren. 2020. Identifying Influencers in Social Networks. *Entropy* 22, 4 (2020), 450.

[33] Gábor Iván and Vince Grolmusz. 2010. When the Web meets the cell: using personalized PageRank for analyzing protein interaction networks. *Bioinformatics* 27, 3 (2010), 405–407.

[34] Ferenc Jordán, Zsófia Benedek, and János Podani. 2007. Quantifying positional importance in food webs: a comparison of centrality indices. *Ecological Modelling* 205, 1-2 (2007), 270–275.

[35] Donald E Knuth. 1993. *The Stanford GraphBase: a platform for combinatorial computing*. ACM.

[36] Dirk Koschützki, Henning Schwöbbermeyer, and Falk Schreiber. 2007. Ranking of network elements based on functional substructures. *Journal of Theoretical Biology* 248, 3 (2007), 471–479.

[37] Cong Li, Qian Li, Piet Van Mieghem, H Eugene Stanley, and Huijuan Wang. 2015. Correlation between centrality metrics and their application to the opinion model. *The European Physical Journal B* 88, 3 (2015), 1–13.

[38] Jose L Martinez-Rodriguez, Aidan Hogan, and Ivan Lopez-Arevalo. 2020. Information extraction meets the semantic web: a survey. *Semantic Web* 11, 2 (2020), 255–335.

[39] Mark Newman. 2018. *Networks*. Oxford university press.

[40] Mark EJ Newman. 2006. Finding community structure in networks using the eigenvectors of matrices. *Physical review E* 74, 3 (2006), 036104.

[41] Stuart Oldham, Ben Fulcher, Linden Parkes, Aurina Arnatkeviciūtė, Chao Suo, and Alex Fornito. 2019. Consistency and differences between centrality measures across distinct classes of networks. *PloS one* 14, 7 (2019), e0220061.

[42] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web*. Technical Report. Stanford InfoLab.

[43] Lauren E Quevillon, Ephraim M Hanks, Shweta Bansal, and David P Hughes. 2015. Social, spatial and temporal organization in a complex insect society. *Scientific reports* 5, 1 (2015), 1–11.

[44] Jennifer JH Reynolds, Ben T Hirsch, Stanley D Gehrt, and Meggan E Craft. 2015. Raccoon contact networks predict seasonal susceptibility to rabies outbreaks and limitations of vaccination. *Journal of Animal Ecology* 84, 6 (2015), 1720–1731.

[45] Cristian Riveros and Jorge Salas. 2020. A family of centrality measures for graph data based on subgraphs. In *23rd International Conference on Database Theory (ICDT 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[46] Cristian Riveros and Jorge Salas. 2020. A Family of Centrality Measures for Graph Data Based on Subgraphs. In *ICDT*, Carsten Lutz and Jean Christoph Jung (Eds.), Vol. 155. 23:1–23:18.

[47] Yannick Rochat. 2009. *Closeness centrality extended to unconnected graphs: The harmonic centrality index*. Technical Report.

[48] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. https://networkrepository.com

[49] Gert Sabidussi. 1966. The centrality index of a graph. *Psychometrika* 31, 4 (1966), 581–603.

[50] Pratha Sah, Kenneth E Nussear, Todd C Esque, Christina M Aiello, Peter J Hudson, and Shweta Bansal. 2016. Inferring social structure and its drivers from refuge use in the desert tortoise, a relatively solitary species. *Behavioral Ecology and Sociobiology* 70, 8 (2016), 1277–1289.

[51] Chengcheng Shao, Pengshuai Cui, Peng Xun, Yuxing Peng, and Xinwen Jiang. 2018. Rank correlation between centrality metrics in complex networks: an empirical study. *Open Physics* 16, 1 (2018), 1009–1023.

[52] Andrii Shekhovtsov. 2021. How strongly do rank similarity coefficients differ used in decision making problems? *Procedia Computer Science* 192 (2021), 4570–4577.

[53] Jinfang Sheng, Jinying Dai, Bin Wang, Guihua Duan, Jun Long, Junkai Zhang, Kerong Guan, Sheng Hu, Long Chen, and Wanghao Guan. 2020. Identifying influential nodes in complex networks based on global and local structure. *Physica A: Statistical Mechanics and its Applications* 541 (2020), 123262.

[54] Alfonso Shimbel. 1953. Structural parameters of communication networks. *The bulletin of mathematical biophysics* 15, 4 (1953), 501–507.

[55] Oskar Skibski, Tomasz P Michalak, and Talal Rahwan. 2018. Axiomatic characterization of game-theoretic centrality. *Journal of Artificial Intelligence Research* 62 (2018), 33–68.

[56] Seinosuke Toda. 1991. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.* 20, 5 (1991), 865–877.

[57] Stojan Trajanovski, Javier Martín-Hernández, Wynand Winterbach, and Piet Van Mieghem. 2013. Robustness envelopes of networks. *Journal of Complex Networks* 1, 1 (2013), 44–62.

[58] Leslie G Valiant. 1979. The complexity of computing the permanent. *Theoretical computer science* 8, 2 (1979), 189–201.

[59] Rene E van Dijk, Jennifer C Kaden, Araceli Argüelles-Ticó, Deborah A Dawson, Terry Burke, and Ben J Hatchwell. 2014. Cooperative investment in public goods is kin directed in communal nests of social birds. *Ecology letters* 17, 9 (2014), 1141–1148.

[60] Jose L Walteros, Alexander Veremyev, Panos M Pardalos, and Eduardo L Pasiliao. 2019. Detecting critical node structures on graphs: A mathematical programming approach. *Networks* 73, 1 (2019), 48–88.

[61] Tomasz Was and Oskar Skibski. 2018. Axiomatization of the PageRank Centrality.. In *IJCAI*. 3898–3904.

[62] Erjia Yan and Ying Ding. 2009. Applying centrality measures to impact analysis: A coauthorship network analysis. *Journal of the American Society for Information Science and Technology* 60, 10 (2009), 2107–2118.

[63] Xuemei You, Yinghong Ma, and Zhiyuan Liu. 2020. A three-stage algorithm on community detection in social networks. *Knowledge-Based Systems* 187 (2020), 104822.

[64] Wayne W Zachary. 1977. An information flow model for conflict and fission in small groups. *Journal of anthropological research* 33, 4 (1977), 452–473.

A DEFINITIONS

We introduce the definitions for every centrality measure mentioned in this paper.

Degree

The degree centrality measure is one of the simplest measures that can be found in the literature. It states that the bigger the neighborhood of a vertex, the most central it is. Formally, given a graph G and $v \in V(G)$, the *degree centrality* of v in G is

$$\textsc{Degree}(v, G) \;=\; |N_G(v)|.$$

Closeness

This is a well-known measure introduced back in 1960s [49]. It is usually called a geometrical measure since it relies on the distance inside a graph. It essentially states that the closer a vertex is to everyone in the graph the more central it is. Formally, given a graph G and a vertex $v \in V(G)$, the *closeness centrality* of v in G is defined as the ratio

$$\text{Closeness}(v, G) = \frac{1}{\sum_{u \in K_v(G)} d_G(v, u)}.$$

Let us clarify that we define the sum of distances inside a component of G since the distance between two vertices in different components of G is by definition infinite.

Harmonic

This centrality measure is commonly considered as an extension of Closeness to disconnected graphs. Given a graph G and a vertex $v \in V(G)$, the *harmonic centrality* [47] of v in G is defines as

$$\text{Harmonic}(v, G) \;=\; \begin{cases} 0 & v \text{ is isolated in } G \\ \displaystyle\sum_{u \in V(G) \setminus \{v\}} \frac{1}{d_G(v, u)} & \text{otherwise.} \end{cases}$$

Recall that, for two nodes $v, u \in V(G)$, if there is no path between them, then $d_G(v, u) = \infty$. Therefore, a vertex $u \in V(G)$ that is not connected with v in G via a path, has no impact on the harmonic centrality of v in G. Just as Closeness, Harmonic is usually called a geometrical measure since it relies on the distance inside a graph.

PageRank

This measure, introduced in [42], is used in Google's web searching engine. Given a graph G with $V(G) = \{v_1, \ldots, v_{|V(G)|}\}$, we define the adjacency matrix of G as the matrix A_G such that $A_{Gij} = 1$ if $\{v_i, v_j\} \in E(G)$. We then define $\mathbf{P}$ as the column-stochastic matrix such that:

$$\mathbf{P}_{ij} \;=\; \frac{A_{Gij}}{|N_G(i)|}.$$

In other words, $\mathbf{P}$ contains the probabilities of jumping from node i to node j during a random walk over G. Let $\mathbf{v}$ be a stochastic vector ($\mathbf{1}^T\mathbf{v} = 1$), and let $0 < \alpha < 1$ be a teleportation parameter. Then, the *$(\mathbf{v}, \alpha)$-PageRank vector* is defined as the vector $x_{\mathbf{v},\alpha}$ that solves the equation $(I - \alpha\mathbf{P})x = (1 - \alpha)\mathbf{v}$. Now, given a graph G with $V(G) = \{v_1, \ldots, v_{|V(G)|}\}$, the *PageRank centrality* of v_i in G is defined as

$$\text{PageRank}(v_i, G) = x_{0.85,\mathbf{u}}[i],$$

where $\mathbf{u} = \mathbf{1} \cdot \frac{1}{|V(G)|}$. Several variations of PageRank have been proposed in the literature. In particular, if we do not force $x_{0.85,\mathbf{u}}$ to be a probability distribution, we can ask that the sum of centrality values inside a connected component is equal to the number of nodes in that component. Therefore, this give rise to the local version of PageRank, denoted LPR. We call it local since the values in a connected component are always the same regardless of the graph.

Betweenness

For this centrality measure we take the proportion of shortest paths going through a certain vertex. Formally, given a graph G and a vertex $v \in V(G)$, the *betweenness centrality* [25] of v in G is

$$\text{Betweeness}(v, G) \;=\; \sum_{(u,w) \in (K_v(G) \setminus \{v\})^2} \frac{|S_G^v(u, w)|}{|S_G(u, w)|}.$$

B NETWORK LIST

#	Network Name	Category	Tree width	$\|V\|$	$\|E\|$	Notes
1	Chebyshev1	Miscellaneous	6	261	1542	
2	ENZYMES_g123	Cheminformatics	7	90	127	
3	ENZYMES_g146	Cheminformatics	7	39	87	
4	ENZYMES_g147	Cheminformatics	7	40	84	
5	ENZYMES_g148	Cheminformatics	7	39	80	
6	ENZYMES_g169	Cheminformatics	7	44	94	
7	ENZYMES_g438	Cheminformatics	8	48	99	
8	ENZYMES_g442	Cheminformatics	7	44	95	
9	ENZYMES_g532	Cheminformatics	8	74	120	
10	ENZYMES_g600	Cheminformatics	7	47	78	
11	GD00_a	Miscellaneous	2	83	125	
12	GD02_a	Miscellaneous	6	23	59	
13	GD02_b	Miscellaneous	8	80	232	
14	GD95_a	Miscellaneous	4	36	56	
15	GD95_b	Miscellaneous	6	70	95	
16	GD95_c	Miscellaneous	4	62	144	
17	GD96_b	Miscellaneous	3	111	193	
18	GD96_d	Miscellaneous	7	180	228	
19	GD97_b	Miscellaneous	6	46	132	
20	GD97_c	Miscellaneous	2	345	355	
21	GD98_a	Miscellaneous	3	32	43	
22	GD98_b	Miscellaneous	2	121	132	
23	GD99_c	Miscellaneous	3	105	120	
24	GlossGT	Miscellaneous	7	60	114	
25	M10PI_n	Miscellaneous	3	682	689	Build in [26]
26	M10PI_n1	Miscellaneous	3	526	533	Build in [26]
27	M20PI_n	Miscellaneous	3	1182	1198	Build in [26]
28	M40PI_n	Miscellaneous	3	2182	2198	Build in [26]
29	M40PI_n1	Miscellaneous	3	2026	2033	Build in [26]
30	M80PI_n	Miscellaneous	3	4182	4198	Build in [26]
31	M80PI_n1	Miscellaneous	3	4026	4033	Build in [26]
32	NCI1_g1677	Cheminformatics	2	52	54	
33	Ragusa16	Miscellaneous	6	24	58	
34	Ragusa18	Miscellaneous	5	23	51	
35	S20PI_n1	Miscellaneous	3	1026	1033	
36	aves-weaver-social-08	Animal Social Networks	9	28	115	Build in [59]
37	aves-weaver-social-16	Animal Social Networks	8	64	177	Build in [59]
38	bcspwr01	Miscellaneous	3	39	46	
39	bcspwr02	Miscellaneous	3	49	59	
40	bcspwr03	Miscellaneous	4	118	179	
41	bcsstk03	Miscellaneous	3	56	132	
42	bcsstk20	Miscellaneous	8	467	1295	
43	bfwa62	Miscellaneous	7	62	200	
44	bfwb62	Miscellaneous	8	35	88	
45	bn-mouse_visual-cortex_1	Brain Networks	4	29	44	
46	bn-mouse_visual-cortex_2	Brain Networks	4	193	214	
47	bwm200	Miscellaneous	2	200	298	
48	ca-CSphd	Collaboration Networks	3	1025	1043	Used in [53]
49	ca-netscience	Collaboration Networks	8	379	914	Build in [40], used in [37, 51]
50	ca-sandi_auths	Collaboration Networks	4	86	124	Used in [60]
51	cage5	Miscellaneous	8	37	98	
52	can_144	Miscellaneous	8	144	576	
53	can_24	Miscellaneous	6	24	68	
54	can_61	Miscellaneous	7	61	248	
55	can_62	Miscellaneous	3	62	78	
56	cis-n4c6-b15	Miscellaneous	5	920	959	
57	ck104	Miscellaneous	5	52	222	
58	ck400	Miscellaneous	5	200	622	

#	Network Name	Category	Tree width	$\|V\|$	$\|E\|$	Notes
59	ck656	Miscellaneous	5	328	814	
60	contiguous-usa	Miscellaneous	7	49	107	
61	curtis54	Miscellaneous	6	54	124	
62	dwt_234	Miscellaneous	6	117	162	
63	dwt_59	Miscellaneous	5	59	104	
64	dwt_66	Miscellaneous	2	66	127	
65	dwt_72	Miscellaneous	2	72	75	
66	dwt_87	Miscellaneous	8	87	227	
67	gams10am	Miscellaneous	4	171	298	
68	gams30a	Miscellaneous	4	531	938	
69	gams60am	Miscellaneous	4	1071	1898	
70	insecta-ant-trophallaxis-colony2-day7	Animal Social Networks	5	31	52	Build in [43]
71	karate	Miscellaneous	5	34	78	Build in [64], used in [37, 60, 63]
72	klein-b1	Miscellaneous	5	30	55	
73	lap_25	Miscellaneous	7	25	72	
74	laser	Miscellaneous	2	3002	4000	
75	lesmis	Miscellaneous	9	77	254	Build in [35], used in [37, 60, 63]
76	lpi_bgprtr	Miscellaneous	8	40	67	
77	lpi_woodinfe	Miscellaneous	8	89	138	
78	lung1	Miscellaneous	3	1650	4121	
79	maayan-pdzbase	Miscellaneous	6	161	209	
80	mammalia-asianelephant	Animal Social Networks	2	20	22	Build in [20]
81	mammalia-baboon-grooming-group03	Animal Social Networks	4	23	35	Build in [23]
82	mammalia-elephantseal-2010-2011	Animal Social Networks	3	46	56	Build in [15]
83	mammalia-primate-association-10	Animal Social Networks	7	21	60	Build in [28]
84	mammalia-raccoon-proximity-4	Animal Social Networks	8	24	100	Build in [44]
85	mammalia-vampire-bats-foodsharing	Animal Social Networks	8	21	72	Build in [14]
86	mammalia-voles-bhp-trapping-12	Animal Social Networks	4	52	96	Build in [19]
87	mammalia-voles-kcs-trapping-27	Animal Social Networks	9	111	300	Build in [19]
88	mhdb416	Miscellaneous	5	102	351	
89	mk10-b4	Miscellaneous	2	4475	4475	
90	n2c6-b10	Miscellaneous	3	306	329	
91	n3c4-b2	Miscellaneous	8	20	50	
92	n3c5-b1	Miscellaneous	8	45	83	
93	nos1	Miscellaneous	3	158	312	
94	nos2	Miscellaneous	3	638	1272	
95	nos4	Miscellaneous	8	100	247	
96	odepa400	Miscellaneous	2	400	402	
97	olm100	Miscellaneous	2	100	197	
98	olm500	Miscellaneous	2	500	997	
99	olm5000	Miscellaneous	2	5000	9997	
100	oscil_dcop_01	Miscellaneous	5	426	603	
101	p0033	Miscellaneous	6	48	98	
102	p0040	Miscellaneous	4	63	110	
103	pivtol	Miscellaneous	2	102	103	
104	pores_1	Miscellaneous	7	30	103	
105	problem	Miscellaneous	5	46	83	
106	rajat11	Miscellaneous	7	135	377	
107	reptilia-tortoise-network-fi-2011	Animal Social Networks	5	142	248	Build in [50]
108	rt-retweet	Retweet Networks	5	96	117	
109	scc_rt_assad	Temporal Reachability	7	28	93	
110	scc_rt_bahrain	Temporal Reachability	7	34	109	
111	tub100	Miscellaneous	2	100	148	
112	tub1000	Miscellaneous	2	1000	1498	
113	webkb-wisc	Labeled	7	251	450	

C PROOFS OF SECTION 3

C.1 Proof of theorem 3.1

All subgraphs and trees. This result has been proven in [46].

Paths. We will reduce the problem of given two nodes s, t in G, counting all the simple paths going from s to t in G. Denoted as $\textsc{Paths}(s, t, G)$. In [58] this is proven a $\# - P$ Hard problem. Now, the way we can use $\textsc{Count}[\mathcal{P}]$ to count $\textsc{Paths}(s, t, G)$ goes as follow. Given a graph G, nodes s and t in G, first we generate the graph $G' = (V(G) \cup \{v_1, v_2\}, E(G) \cup \{\{v_1, s\}, \{v_2, t\}\}))$ such that v_1, v_2 are not in G. Now, it is easy to see that

$$\textsc{Count}[\mathcal{P}](v_1, G') = \textsc{Count}[\mathcal{P}](\{v_1, v_2\}, G') + \textsc{Count}[\mathcal{P}](v_1, G' - \{v_2\}).$$

In other words, paths including v_1 can be separated in those including v_2 and those excluding v_2. However, since v_1 and v_2 are leaves in G', we have that $\textsc{Count}[\mathcal{P}](\{v_1, v_2\}, G')) = \textsc{Paths}(s, t, G)$. Which mean we have the equivalence

$$\textsc{Paths}(s, t, G) = \textsc{Count}[\mathcal{P}](v_1, G') - \textsc{Count}[\mathcal{P}](v_1, G' - \{v_2\})$$

which concludes the hardness proof. Now, this problem is in $\#P$ since we can consider a polinomial non deterministic Turing machine M that decides if v has a simple path containing it in G. It is clear that any witness for M is a simple path in G, and therefore it takes polynomial time to check if it is indeed a simple path containing v. Then, the number of acceptance executions for M corresponds to $\textsc{Count}[\mathcal{P}](c, G)$.

Cycles. Can be deduced from [58], in S-T PATHS, we just take $s = t$ and the result follows.

D RESULTS ON HARDNESS OF APPROXIMATION

Here we prove the hardness of approximating the centrality measure of paths, cycles, and cliques.

In this section we prove the hardness of approximating the value of the centrality measure when counting paths, cycles and cliques. We accomplish this by means of proving that there is no FPRAS (Fully Polynomial Randomized Approximation Scheme) for each of these three problems.

D.1 Simple cycles

D.1.1 Setting. Let $G = (V(G), E(G))$ be a connected undirected graph with $|V(G)| = n$ such that $v \in V(G)$ is a known vertex. We define by $C(G, v)$ the family of simple cycles that in G and such that they contain v in their nodes set. The subgraph $G' = (V', E')$ is a simple cycle in G if

(1) $V' = \{v_1, \ldots, v_n\}$ for $n \geq 2$
(2) $v_i \neq v_j$ for $1 \leq i < j \leq n$
(3) $E' = \{\{v_i, v_{i+1}\} \mid 1 \leq 1 \leq n - 1\} \cup \{\{v_n, v_1\}\}$ and $E' \subseteq E(G)$

Moreover, we say $G' \in C(G, v)$ if $v \in V'$.

Now, we define the counting problem we will study:

Problem:	#SimpleCycle
Input:	A graph G and a vertex $v \in V(G)$
Output:	$\lvert C(G, v)\rvert$

We will prove that #SimpleCycle does not admit an FPRAS.

D.1.2 Premise. We consider the following decision problem:

Problem:	HAM
Input:	A graph G
Output:	Yes iff G has a Hamiltonian cycle

This is a known NP-complete problem and we will show that if #SimpleCycle has an FPRAS, then HAM $\in$ BPP and therefore, NP = BPP.

D.1.3 Augmented graph. We define the augmented graph $G^r = (V^r, E^r)$ of $G = (V(G), E(G))$ as follows. Given an integer r (which value we will define later), we replace each edge $\{v_i, v_j\}$ in G by new edges and vertices as seen in Figure 1.

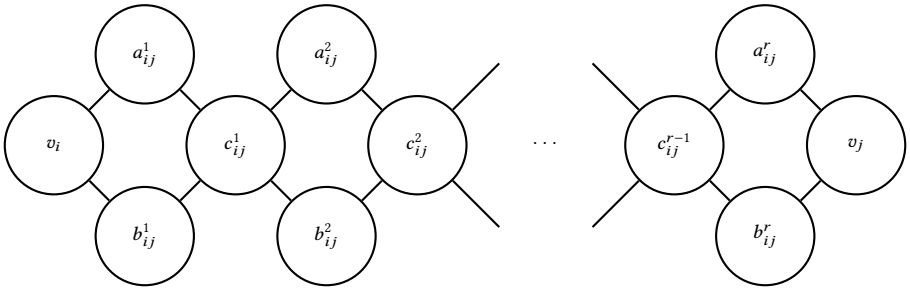


Figure 1. Augmentation of the edge $\{v_i, v_j\}$.

We formalize the set of vertices by

$$V^r = V(G) \cup \bigcup_{\substack{\{v_i,v_j\}\in E(G)\\ 1\le k\le r}} \left\{a^k_{ij}, b^k_{ij}, c^k_{ij}\right\}$$

where each $c^r_{ij} = v_j$. The set of edges is composed of three types of edges:

$$E_1 = \bigcup_{\{v_i,v_j\}\in E(G)} \left\{\{v_i, a^1_{ij}\}, \{v_i, b^1_{ij}\}\right\}$$

$$E_2 = \bigcup_{\{v_i,v_j\}\in E(G)} \left\{\{a^r_{ij}, v_j\}, \{b^r_{ij}, v_j\}\right\}$$

$$E_3 = \bigcup_{\{v_i,v_j\}\in E(G)} \left\{\{a^k_{ij}, c^k_{ij}\}, \{b^k_{ij}, c^k_{ij}\}, \{c^k_{ij}, a^{k+1}_{ij}\}, \{c^k_{ij}, b^{k+1}_{ij}\}\right\}$$

Finally, the set of edges is $E^r = E_1 \cup E_2 \cup E_3$. If we consider just the subgraph shown in Figure 1, this arrangement of edges provides 2^r different paths from v_i to v_j in this subgraph.

Since G has at most $n(n-1)$ edges (in the case of G being an n-clique), we have that G^r has at most $n + n(n-1)(3r-1)$ vertices. Therefore, the construction of G^r from G requires a polynomial amount of time according to the number of vertices.

D.1.4 Connection with HAM. Let $H \subseteq G$ be a cycle in G. By the construction of G^r, if H has length $\ell > 1$ there are $2^{\ell r}$ different cycles of length $2\ell r$ in G^r. Moreover, each cycle in G^r results from the augmentation of a unique simple cycle in G. We can now deduce that

$$G \in \text{HAM} \;\Leftrightarrow\; G^r \text{ has a simple cycle of length } 2rn \tag{1}$$

We now define two sets of cycles, given an instance G of the HAM problem.

- If $G \in$ HAM, we define

  $$\mathcal{I}^r(G) = \{C \mid C \text{ is a simple cycle in } G^r \text{ of length } 2rn\}$$

  Since G has at least one Hamiltonian cycle H, we have $|\mathcal{I}^r(G)| \geq 2^{nr}$. Furthermore, since every simple cycle $C \in \mathcal{I}^r(G)$ contains every vertex from $V(G)$, in particular they contain v and we conclude that

  $$\#\textsc{SimpleCycle}(G^r, v) \geq |\mathcal{I}^r(G)| \geq 2^{nr},\ \forall v \in V(G) \tag{2}$$

- If $G \notin$ HAM, we define

  $$\mathcal{M}^r(G) = \{C \mid C \text{ is a simple cycle in } G^r \text{ of length less than } 2rn\}$$

  Since there are at most $\binom{n(n-1)}{\ell}$ simple cycles of length ℓ in G and each one of them has $2^{\ell r}$ related cycles in G^r, an upper bound for $|\mathcal{M}^r(G)|$ is

  $$\begin{aligned}
  |\mathcal{M}^r(G)| &\leq \sum_{\ell=2}^{n-1} 2^{\ell r}\binom{n(n-1)}{\ell} \\
  &\leq \sum_{\ell=2}^{n(n-1)} 2^{\ell r}\binom{n(n-1)}{\ell} \\
  &\leq \sum_{\ell=0}^{n(n-1)} 2^{(n-1)r}\binom{n(n-1)}{\ell} \\
  &= 2^{(n-1)r}2^{n(n-1)} = 2^{(n-1)(n+r)}
  \end{aligned}$$

Since every simple cycle passing through v in G^r has a length strictly less than $2rn$,

$$\#\textsc{SimpleCycle}(G^r, v) \leq |\mathcal{M}^r(G)| \leq 2^{(n-1)(n+r)} \tag{3}$$

D.1.5 Consequence of FPRAS existence. Suppose $\mathcal{A}$ to be an FPRAS for #SimpleCycle , i.e. for every $\varepsilon \in (0,1)$, if (G^r, v) is an instance of #SimpleCycle we have

$$\mathbb{P}\left[\left|\mathcal{A}((G^r, v), \varepsilon) - \#\textsc{SimpleCycle}(G^r, v)\right| \leq \varepsilon \cdot \#\textsc{SimpleCycle}(G^r, v)\right] \geq \frac{3}{4} \tag{4}$$

With the aid of $\mathcal{A}$, we define the randomized algorithm $\mathcal{B}$ for deciding HAM as follows:

(1) Given an instance G for HAM , with n vertices, if $n \leq 1$ return **NO**.
(2) Construct the augmented graph G^r from G and select any $v \in V(G)$.
(3) Let $s \leftarrow \mathcal{A}((G^r, v), \frac{1}{2})$.
(4) If $s > (1 + \frac{1}{2})2^{(n-1)(n+r)}$ return **YES**, else return **NO**.

Since the construction of G^r from G is polynomial and $\mathcal{A}$ itself is a polynomial time algorithm, $\mathcal{B}$ is also polynomial. Now we study the probability of $\mathcal{B}$ being mistaken when classifying an instance G.

- If $G \notin$ HAM, we need to determine $\mathbb{P}[\mathcal{B}(G) \text{ returns } \textbf{YES}]$. Using (3)

$$\begin{aligned}
\mathbb{P}[\mathcal{B}(G) \text{ returns } \textbf{YES}] &= \mathbb{P}\left[\mathcal{A}((G^r, v), \tfrac{1}{2}) > (1 + \tfrac{1}{2})2^{(n-1)(n+r)}\right] \\
&\leq \mathbb{P}\left[\mathcal{A}((G^r, v), \tfrac{1}{2}) > (1 + \tfrac{1}{2}) \cdot \#\textsc{SimpleCycle}(G^r, v)\right] \\
&\leq \mathbb{P}\left[\mathcal{A}((G^r, v), \tfrac{1}{2}) > (1 + \tfrac{1}{2}) \cdot \#\textsc{SimpleCycle}(G^r, v) \vee \right. \\
&\qquad \left. \mathcal{A}((G^r, v), \tfrac{1}{2}) < (1 - \tfrac{1}{2}) \cdot \#\textsc{SimpleCycle}(G^r, v)\right] \\
&= 1 - \mathbb{P}\left[\mathcal{A}((G^r, v), \tfrac{1}{2}) \leq (1 + \tfrac{1}{2}) \cdot \#\textsc{SimpleCycle}(G^r, v) \wedge \right. \\
&\qquad \left. \mathcal{A}((G^r, v), \tfrac{1}{2}) \geq (1 - \tfrac{1}{2}) \cdot \#\textsc{SimpleCycle}(G^r, v)\right] \\
&= 1 - \mathbb{P}\left[\left|\mathcal{A}((G^r, v), \tfrac{1}{2}) - \#\textsc{SimpleCycle}(G^r, v)\right| \leq \tfrac{1}{2} \cdot \#\textsc{SimpleCycle}(G^r, v)\right]
\end{aligned}$$

  Using (4) we conclude that $\mathbb{P}[\mathcal{B}(G) \text{ returns } \textbf{YES}] \leq \frac{1}{4}$ when $G \notin$ HAM.
- If $G \in$ HAM, in order to use he same approach we need to determine a value of r such that $(1 - \frac{1}{2}) \cdot \#\textsc{SimpleCycle}(G^r, v) \geq 2^{nr} > (1 + \frac{1}{2})2^{(n-1)(n+r)}$. Taking $r = n(n-1) + 2$ satisfies this condition. For such r, we are interested in $\mathbb{P}[\mathcal{B}(G) \text{ returns } \textbf{NO}]$. Using (2) and the value of r,

$$\begin{aligned}
\mathbb{P}[\mathcal{B}(G) \text{ returns } \textbf{NO}] &= \mathbb{P}\left[\mathcal{A}((G^r, v), \tfrac{1}{2}) \leq (1 + \tfrac{1}{2})2^{(n-1)(n+r)}\right] \\
&\leq \mathbb{P}\left[\mathcal{A}((G^r, v), \tfrac{1}{2}) < (1 - \tfrac{1}{2}) \cdot \#\textsc{SimpleCycle}(G^r, v)\right] \\
&\leq \mathbb{P}\left[\mathcal{A}((G^r, v), \tfrac{1}{2}) < (1 - \tfrac{1}{2}) \cdot \#\textsc{SimpleCycle}(G^r, v) \vee \right. \\
&\qquad \left. \mathcal{A}((G^r, v), \tfrac{1}{2}) > (1 + \tfrac{1}{2}) \cdot \#\textsc{SimpleCycle}(G^r, v)\right] \\
&= 1 - \mathbb{P}\left[\mathcal{A}((G^r, v), \tfrac{1}{2}) \geq (1 - \tfrac{1}{2}) \cdot \#\textsc{SimpleCycle}(G^r, v) \wedge \right. \\
&\qquad \left. \mathcal{A}((G^r, v), \tfrac{1}{2}) \leq (1 + \tfrac{1}{2}) \cdot \#\textsc{SimpleCycle}(G^r, v)\right] \\
&= 1 - \mathbb{P}\left[\left|\mathcal{A}((G^r, v), \tfrac{1}{2}) - \#\textsc{SimpleCycle}(G^r, v)\right| \leq \tfrac{1}{2} \cdot \#\textsc{SimpleCycle}(G^r, v)\right]
\end{aligned}$$

  Using (4) we conclude that $\mathbb{P}[\mathcal{B}(G) \text{ returns } \textbf{NO}] \leq \frac{1}{4}$ when $G \in$ HAM.

Taking $r = n(n-1) + 2$, the probabilities of $\mathcal{B}$ give us that HAM $\in$ BPP, and therefore NP = RP. □

D.2 Simple paths

D.2.1 Setting. Following the same strategy from the previous counting problem, we now focus on the number of simple paths that contain a vertex in a graph. Given the undirected graph $G = (V(G), E(G))$ and a vertex $v \in V(G)$, we denote by $L(G, v)$ the family of simple paths in G that contain v in their nodes set. A subgraph $G' = (V', E')$ of G is a simple path in G if

(1) $V' = \{v_1, \ldots, v_n\}$ for $n \geq 2$
(2) $v_i \neq v_j$ for $1 \leq i < j \leq n$
(3) $E' = \{\{v_i, v_{i+1}\} \mid 1 \leq 1 \leq n-1\}$ and $E' \subseteq E(G)$

D.2.2 Premise. We define the counting problem we will study as follows

Problem:	#SimplePath
Input:	A graph G and a vertex $v \in V(G)$
Output:	$\lvert L(G,v)\rvert$

To prove that #SimplePath does not have an FPRAS, we will use the following NP-complete problem:

Problem:	HAMP
Input:	A graph G
Output:	Yes if G has a Hamiltonian path

D.2.3 Graph augmentation and bounds. Using the same construction from the previous problem, we use G^r as the augmented graph obtained from G in polynomial time with respect to the number of vertices $n = |V(G)|$. Because we take HAMP as the reference decision problem, we define our sets taking paths and not cycles into account. Consequently, we note that each path of length ℓ in G has $2^{\ell r}$ different paths associated in G^r and each path from G^r comes from at most one simple path from G (it is important to mention that a path in G^r could not have a simple path in G, e.g. the path $(\{a^1_{ij}, c^1_{ij}\}, \{c^1_{ij}, b^2_{i,j}\})$ for any valid pair i, j). The key property to connect the decision problem with the counting of simple paths is:

$$G \in \text{HAMP} \iff G^r \text{ has a simple path of length } 2rn \tag{5}$$

We define the two sets $\mathcal{I}^r(G)$ and $\mathcal{M}^r(G)$ to obtain the required bounds:

- If $G \in$ HAM, we define

  $$\mathcal{I}^r(G) = \{L \mid L \text{ is a simple path in } G^r \text{ of length } 2rn\}$$

  Since G has at least one Hamiltonian path P, we have $|\mathcal{I}^r(G)| \geq 2^{nr}$. Since any vertex from G is included in a Hamiltonian path and every augmented path obtained from it also includes every $v \in V(G)$, we have

  $$\#\text{SimplePath}(G^r, v) \geq |\mathcal{I}^r(G)| \geq 2^{nr}, \ \forall v \in V(G) \tag{6}$$

- If $G \notin$ HAM, we define

  $$\mathcal{M}^r(G) = \{P \mid P \text{ is a simple path in } G^r \text{ of length less than } 2rn\}$$

  Since there are at most $\binom{n(n-1)}{\ell}$ simple paths of length ℓ in G and each one of them has $2^{\ell r}$ related paths in G^r, we obtain the same bound for $|\mathcal{M}^r(G)| \leq 2^{(n-1)(n+r)}$ as in the previous problem.
  Since every simple path passing through v in G^r has a length strictly less than $2rn$,

  $$\#\text{SimplePath}(G^r, v) \leq |\mathcal{M}^r(G)| \leq 2^{(n-1)(n+r)} \tag{7}$$

We conclude in the same manner that that there is no FPRAS for #SimplePath . □

D.3 Cliques

D.3.1 Setting. Given the undirected graph $G = (V(G), E(G))$ and a vertex $v \in V(G)$, we are interested in the size of the family $K(G, v)$ of all cliques subgraphs of G that contain v, i.e.

Problem:	#VertexClique
Input:	A graph G and a vertex $v \in V(G)$
Output:	$\lvert K(G,v)\rvert$

We will prove that #VertexClique does not admit an FPRAS by proving first, that the following problem does not admit an FPRAS:

Problem:	#ISV
Input:	A graph G and a vertex $v \in V(G)$
Output:	$\lvert IS(G,v)\rvert$

where $IS(G, v)$ is the set of all independent sets in G that contain v.

D.3.2 Premise. We consider the following decision problems:

Problem:	GIS
Input:	A graph G and an integer $k \geq 0$
Output:	Yes if G has an independent set of size at least k

Problem:	GISV
Input:	A graph G, a vertex $v \in V(G)$ and an integer $k \geq 0$
Output:	Yes if G has an independent set of size at least k that contains v

GIS is a known NP-complete problem. We note that if an instance (G', v, k') of the GISV problem does indeed belongs to GISV, then there is a polynomial witness S that is an independent set of G', of size $|S| \geq k$ such that $v \in S$. This witness can be checked in polynomial time and therefore GISV $\in$ NP.

Let (G, k) be an instance of the GIS problem. We define a new graph $G' = (V', E')$ such that $V' = V(G) \cup \{v\}$ and $E' = E$, where $v \notin V(G)$. Also, we define $k' = k + 1$.

- If $(G, k) \in$ GIS, there is an independent set S such that $|S| \geq k$. Since there are no edges connecting v to any other vertex in G', $S' = S \cup \{v\}$ is an independent set in G' such that $|S| \geq k + 1 = k'$ and $v \in S$. Therefore, $(G', v, k') \in$ GISV.
- If $(G, k) \notin$ GIS, any independent set S in G has size $|S| < k$, thus implying that any independent set $S' = S \cup \{v\}$ in G' has size at most $k - 2 < k'$. Since there is no independent set of size at least k' in G' containing v, $(G', v, k') \notin$ GISV.

This proves that GISV is NP-hard and since GISV $\in$ NP, it is NP-complete. We will use this fact as the basis of our proof of non-existence of an FPRAS for #VertexClique.

D.3.3 Augmented graph. Given $G = (V(G), E(G))$ with $n = |V(G)|$, we define the augmented graph $G^r = (V^r, E^r)$ of G as follows. Taking $r = n + 3$, we amplify the original graph by making r copies of each vertex in $V(G)$:

$$C_i = \{u_i^1, u_i^2, \ldots, u_i^r\},\ 1 \leq i \leq n$$

The amplified set of copied vertices and edges are

$$V^r = \bigcup_{i=1}^{n} C_i$$

$$E^r = \bigcup_{\substack{\{u_i, u_j\} \in E(G) \\ i \neq j}} \{\{x, y\} \mid x \in C_i \wedge y \in C_j\}$$

We will say that a set $S' \subseteq V^r$ is generated by $S \subseteq V(G)$ if for every $u_i \in S$, $S' \cap C_i \neq \emptyset$, i.e. there is at least one copy of each vertex from S. By construction of E^r, every independent set in G^r is generated by a unique independent set in G.

D.3.4 Connection between #ISV and GISV . Following the idea of bound definition from the previous subsections, we study two cases for any instance (G, v, k) of the GISV problem. Without loss of generality, we will denote $v = u_1$. We will count #ISV for the amplified graph selecting the vertex $u_1^1 \in C_1$, i.e. #ISV(G^r, u_1^1).

- If $(G, u_1, k) \in$ GISV, there is an independent set of size at least k that contains u_1. In particular, there is an independent set S in G of size $|S| = k$ such that $u_1 \in S$. We are interested in the number of independent sets generated from S such that all of them contain u_1^1. Since there are $r - 1$ vertices from C_1 that are different from u_1^1, there are 2^{r-1} ways to choose vertices from C_1. For the other C_i sets, we have $(2^r - 1)$ ways to choose one or more vertices for each $i > 1$. Thus,

$$\#\text{ISV}(G^r, u_1^1) \geq 2^{r-1}(2^r - 1)^{k-1} \tag{8}$$

- If $(G, u_1, k) \notin$ GISV, every independent set S in G has size at most $k - 1$. In order to obtain an upper bound for #ISV(G^r, u_1^1) we define the following sets of vertices:

  $$P(\ell) = \{S \subseteq V(G) \mid u_1 \in S \wedge |S| = \ell\}$$

  being the collection of all sets containing u_1 and other $\ell - 1$ vertices. These sets are not necessarily independent. The size of this collection is obtain is simply

  $$|P(\ell)| = \binom{n-1}{\ell-1}$$

  The second set is $Q(\ell)$, being the collection of all sets of amplified vertices that contain u_1^1 and their vertices come from exactly ℓ different C_i sets. These sets are also not necessarily independent. The size of this set comes from the choice of copies from C_1 including u_1^1, which are 2^{r-1}, and the choice of at least one copy from the $\ell - 1$ remaining classes C_i, which are $(2^r - 1)$ for each class. Therefore,

  $$|Q(\ell)| = 2^{r-1}(2^r - 1)^{\ell-1}$$

Since the maximum size of an independent set in G is $k-1$, we add all the bounds for sets of size up to $k-1$ as follows:

$$\begin{aligned}
\#\mathrm{ISV}(G^r, u_1^1) &\leq \sum_{\ell=1}^{k-1} |P(\ell)| \cdot |Q(\ell)| \\
&= \sum_{\ell=1}^{k-1} \binom{n-1}{\ell-1} \cdot 2^{r-1}(2^r-1)^{\ell-1} \\
&= 2^{r-1} \sum_{\ell=1}^{k-1} \binom{n-1}{\ell-1} \cdot (2^r-1)^{\ell-1} \\
&\leq 2^{r-1} \sum_{\ell=1}^{k-1} \binom{n-1}{\ell-1} \cdot (2^r-1)^{k-2} \\
&\leq 2^{r-1} \cdot (2^r-1)^{k-2} \sum_{\ell=1}^{k-1} \left[\binom{n-1}{\ell-1} + \binom{n-1}{\ell} \right] \\
&\leq 2^{r-1} \cdot (2^r-1)^{k-2} \sum_{\ell=0}^{n} \binom{n}{\ell}
\end{aligned}$$

where the first inequality results from the over-counting of sets that are not necessarily independent. The upper bound is

$$\#\mathrm{ISV}(G^r, u_1^1) \leq 2^{r-1} \cdot (2^r-1)^{k-2} \cdot 2^n \tag{9}$$

D.3.5 Consequence of FPRAS existence. Suppose $\mathcal{A}$ to be an FPRAS for #ISV , i.e. for every $\varepsilon \in (0,1)$, if (G, v) is an instance of #ISV we have

$$\mathbb{P}\left[|\mathcal{A}((G,v),\varepsilon) - \#\mathrm{ISV}(G,v)| \leq \varepsilon \cdot \#\mathrm{ISV}(G,v) \right] \geq \frac{3}{4} \tag{10}$$

We define a polynomial algorithm $\mathcal{B}$ for deciding GISV as follows:

(1) Given an instance (G, v, k) for GISV , with n vertices.
(2) If $k = 0$ return **NO**.
(3) If $k = 1$, return **YES**.
(4) Construct the augmented graph G^r from G calling $v = u_1$ and select $u_1^1 \in V^r$ (an arbitrary copy of u_1 in the augmented graph).
(5) Let $s \leftarrow \mathcal{A}((G^r, u_1^1), \frac{1}{2})$.
(6) If $s > (1+\frac{1}{2})2^{r-1} \cdot (2^r-1)^{k-2} \cdot 2^n$ return **YES**, else return **NO**.

Since the construction of G^r from G is polynomial (nr vertices on total and at most $\binom{nr}{2}$ edges) and $\mathcal{A}$ itself is a polynomial time algorithm, $\mathcal{B}$ is also polynomial. Now we study the probability of $\mathcal{B}$ being mistaken when classifying an instance G.

– If $G \notin$ GISV, we need to determine $\mathbb{P}\left[\mathcal{B}(G) \text{ returns } \textbf{YES}\right]$. Using (9)

$$\begin{aligned}
\mathbb{P}\left[\mathcal{B}(G) \text{ returns } \textbf{YES}\right] &= \mathbb{P}\left[\mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) > (1+\tfrac{1}{2})2^{r-1} \cdot (2^r-1)^{k-2} \cdot 2^n \right] \\
&\leq \mathbb{P}\left[\mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) > (1+\tfrac{1}{2}) \cdot \#\mathrm{ISV}(G^r, u_1^1) \right] \\
&\leq \mathbb{P}\left[\mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) > (1+\tfrac{1}{2}) \cdot \#\mathrm{ISV}(G^r, u_1^1) \vee \right. \\
&\qquad \left. \mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) < (1-\tfrac{1}{2}) \cdot \#\mathrm{ISV}(G^r, u_1^1) \right] \\
&= 1 - \mathbb{P}\left[\mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) \leq (1+\tfrac{1}{2}) \cdot \#\mathrm{ISV}(G^r, u_1^1) \wedge \right. \\
&\qquad \left. \mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) \geq (1-\tfrac{1}{2}) \cdot \#\mathrm{ISV}(G^r, u_1^1) \right] \\
&= 1 - \mathbb{P}\left[\left| \mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) - \#\mathrm{ISV}(G^r, u_1^1) \right| \leq \tfrac{1}{2} \cdot \#\mathrm{ISV}(G^r, u_1^1) \right]
\end{aligned}$$

Using (10) we conclude that $\mathbb{P}\left[\mathcal{B}(G) \text{ returns } \textbf{YES}\right] \leq \frac{1}{4}$ when $G \notin$ GISV.

– If $G \in$ GISV, we note that the selected value $r = n + 3$ satisfies

$$(1+\frac{1}{2})2^{r-1} \cdot (2^r-1)^{k-2} \cdot 2^n < (1-\frac{1}{2})2^{r-1}(2^r-1)^{k-1} \leq (1-\frac{1}{2})\#\mathrm{ISV}(G^r, u_1^1)$$

For such r, we are interested in $\mathbb{P}\left[\mathcal{B}(G) \text{ returns } \mathbf{NO}\right]$. Using (8) and the value of r,

$$\begin{aligned}
\mathbb{P}\left[\mathcal{B}(G) \text{ returns } \mathbf{NO}\right] &= \mathbb{P}\left[\mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) \leq (1+\tfrac{1}{2})2^{r-1} \cdot (2^r-1)^{k-2} \cdot 2^n\right] \\
&\leq \mathbb{P}\left[\mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) < (1-\tfrac{1}{2}) \cdot \#\text{ISV}(G^r, u_1^1)\right] \\
&\leq \mathbb{P}\left[\mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) < (1-\tfrac{1}{2}) \cdot \#\text{ISV}(G^r, u_1^1) \vee \right. \\
&\qquad \left. \mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) > (1+\tfrac{1}{2}) \cdot \#\text{ISV}(G^r, u_1^1)\right] \\
&= 1 - \mathbb{P}\left[\mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) \geq (1-\tfrac{1}{2}) \cdot \#\text{ISV}(G^r, u_1^1) \wedge \right. \\
&\qquad \left. \mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) \leq (1+\tfrac{1}{2}) \cdot \#\text{ISV}(G^r, u_1^1)\right] \\
&= 1 - \mathbb{P}\left[\left|\mathcal{A}((G^r, u_1^1), \tfrac{1}{2}) - \#\text{ISV}(G^r, u_1^1)\right| \leq \tfrac{1}{2} \cdot \#\text{ISV}(G^r, u_1^1)\right]
\end{aligned}$$

Using (10) we conclude that $\mathbb{P}\left[\mathcal{B}(G) \text{ returns } \mathbf{NO}\right] \leq \frac{1}{4}$ when $G \in \text{GISV}$.
Taking $r = n+3$, the probabilities of $\mathcal{B}$ give us that GISV $\in$ BPP, and therefore NP = RP.
Knowing that #ISV does not admit an FPRAS, we now prove that #ISV $\leq^p_{par}$ #VertexClique to end the result. In fact, let $h : \Sigma^* \to \Sigma^*$ be the function such that for an instance (G, v) of the #ISV problem, with $G = (V, E)$, returns

$$h(G, v) = (\overline{G}, v)$$

where $\overline{G} = (V, V \times V \setminus E)$. We need to prove that #ISV(G, v) = #VertexClique$(h(G, v))$ to conclude that #ISV $\leq^p_{par}$ #VertexClique.

- Let $S \subseteq V$ be an independent set in G such that $v \in S$. Since all vertices in S are disconnected in G, in $\overline{G}$ they are connected with all possible edges between them, i.e. the form a clique which contains v.
- Similarly, any clique K in $\overline{G}$ which contains v corresponds to an independent set which contains v in G.

Thus, since #ISV does not admit an FPRAS, #VertexClique does not admit one either. □

E RESULTS ON HARDNESS OF DECIDING THE RANKING

E.1 Definition of the basic decision problem

In this section we study the complexity of the related decision problem of All Subgraphs Centrality. Namely, that of deciding if two nodes in G are ordered in a specific way according to the ranking provided by the centrality measure.

Problem:	Rank
Input:	A connected graph G and vertices $v, u \in V(G)$
Output:	True iff $\lvert\mathcal{A}(G, v)\rvert \leq \lvert\mathcal{A}(G, u)\rvert$

E.2 An algorithm for computing the number of subgraphs

In order to understand the complexity of Rank , we present an algorithm for computing $|\mathcal{A}(G, v)|$ for any vertex $v \in G$ with the aid of an oracle for Rank . First, we show a polynomial algorithm to construct a tree graph with good properties.

E.2.1 Building a graph with good properties. Given a binary integer $n \geq 1$ with digits $d_1 \dots d_k$, we propose an algorithm for constructing an undirected tree T such that a specific vertex is contained in exactly n connected subgraphs of T.

Let $T_u = (V, E)$ be an undirected tree such that its root u has exactly $|\mathcal{A}(T_u, u)| = n$. Let $v \notin V$ be a new vertex such that we define a new tree $T'_u = (V', E')$ with

$$V' = V \cup \{v\}, \quad E' = E \cup \{\{u, v\}\}$$

- If we consider v as the root of T' and count its subgraphs in T'_u, we have $|\mathcal{A}(T'_u, v)| = n+1$, since each one of the n subtrees including u can be extended with v and we have also the trivial subgraph $\{v\}$.
- If we take u as the root of T'_u and repeat the analysis we have $|\mathcal{A}(T'_u, u)| = 2n$, since each one of the n subgraphs of u in T_u can be extended with v in T'_u, resulting in n new subgraphs.

If we ask for n to be coded as a binary integer, we use the fact that both division by 2 and substraction by 1 can be done changing one digit of the binary representation of n. This is the base idea for the following recursive algorithm to construct a tree T such that we know a specific vertex in it with exactly n subgraphs in T.

Algorithm 3 Tree construction with known $|\mathcal{A}(T,u)|$

Require: $k > 0$ binary digits $(n)_2 = d_1 \dots d_k$ such that $d_1 = 1$
Ensure: (T,u), with a tree $T = (V,E)$ and $u \in V$ such that $|\mathcal{A}(T,u)| = n$

```
1:  procedure BuildTree(d_1 ... d_k)
2:    u ← v_{k-1}
3:    v ← v_k
4:    if k = 1 then
5:      return (({u}, ∅), u)
6:    if d_k = 1 then
7:      (V, E) ←BuildTree(d_1 ... d_{k-1} 0)
8:      return ((V ∪ {v}, E ∪ {{u, v}}), v)
9:    if d_k = 0 then
10:     (V, E) ←BuildTree(d_1 ... d_{k-1})
11:     return ((V ∪ {v}, E ∪ {{u, v}}), u)
```

Since the length of the binary representation of n reduces by 1 after at most two calls of `BuildTree`, for an initial k-digit number we expect at most $2k$ calls of the function in order to construct the complete tree. Each call adds a single vertex and at most a single edge (constant time) so `BuildTree` takes time $O(k)$.

E.3 Using an oracle for Rank

Now that we can construct a graph with known number of subgraphs for a specific vertex, we show an algorithm to compute this number for any graph (not necessarily a tree). For accomplishing this we use a binary search algorithm with access to a constant time oracle for the decision problem Rank .

The basic idea is bounding $n = |\mathcal{A}(G,v)|$ with lower and upper bounds a and b respectively. When we start calling the algorithm, for $G = (V,E)$ we have $a = 1$ and $b = 2^{|E|}$. Both numbers can be represented as binary sequences, with b consisting in a 1 followed by $|E|$ zeros. This can be constructed in polynomial time on the size of the graph.

We make recursive calls to the algorithm dividing the interval $[a,b]$ using the midpoint $c = \frac{a+b}{2}$. This binary number is easily computed from a, b in polynomial time with respect to their sizes. To decide which subinterval is selected to the next recursive call, we use the oracle for Rank on a new graph G'.

$G' = (V', E')$ is formed combining G and the result of calling $(T,u) = \texttt{BuildTree}(c)$, binding v from G with u from T according to

$$V' = V \cup V_T, \quad E' = E \cup E_T \cup \{\{v,u\}\}$$

From construction we have $|\mathcal{A}(T,u)| = c$, so if $|\mathcal{A}(G,v)| = n$, then

$$|\mathcal{A}(G',v)| = cn + n, \qquad |\mathcal{A}(G',u)| = cn + c \tag{11}$$

Calling the oracle over this graph, and using (11) we conclude

$$\begin{aligned} \textsc{Rank}(G',v,u) = \text{True} \quad &\Leftrightarrow \quad |\mathcal{A}(G',v)| \leq |\mathcal{A}(G',u)| \\ &\Leftrightarrow \quad cn + n \leq cn + c \\ &\Leftrightarrow \quad n \leq c \end{aligned}$$

In other words, in this setting the call to the oracle responds if $|\mathcal{A}(G,v)| \leq c$, giving us an algorithm to determine the exact value for n via binary search. The complete algorithm is given as follows.

Algorithm 4 Recursive algorithm for computing $|\mathcal{A}(G,v)|$ with oracle to RANK

Require: connected graph $G=(V,E)$, vertex $v \in V$ and two binary numbers m, n
Ensure: $|\mathcal{A}(G,v)|$

```
procedure AllSubgraphs(G, v, a, b)
  if a = b then
    return a
  c ← (a + b)/2
  ((V_T, E_T), u) ←BuildTree(c)
  V' ← V ∪ V_T
  E' ← E ∪ E_T ∪ {{v, u}}
  G' ← (V', E')
  if Rank(G', v, u) then
    return AllSubgraphsG, v, a, c
  else
    return AllSubgraphsG, v, c, b
```

The initial call of algorithm 3 is `AllSubgraphs`$(G, v, 1, 10\ldots0)$ where the last argument is a binary string with $|E|$ consecutive zeros (the binary representation of $2^{|E|}$, the maximum number of subgraphs in G). The total number of calls of `AllSubgraphs` is therefore polynomial with respect to $|E|$, and since each call to `BuildTree` is also polynomial on the length of c, the overall algorithm is polynomial on the size of G. We conclude that the counting problem `#AllSubgraphs` $\in \mathrm{P}^{\mathrm{RANK}}$.

E.4 Thoughts on #AllSubgraphs $\in \mathbf{P^{RANK}}$

As discussed in [**?**], if an NP-complete problem such as SAT can be used as an oracle to solve its related #P-complete counting problem #SAT, we would have $\#\mathrm{P} \leq_T^p \mathrm{P}^{\mathrm{NP}}$, so $\mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{P}^{\mathrm{P}^{\mathrm{NP}}} = \mathrm{P}^{\mathrm{NP}}$. From Toda's theorem we have that $\mathrm{PH} \subseteq \#\mathrm{P}$, resulting in a collapse of PH to its second level. This implies that if RANK is a complete problem for a specific level of the polynomial hierarchy, then PH collapses to that level. This suggests that RANK is above PH.

F MAIN ALGORITHM BASED ON TREE-DECOMPOSITION

F.1 Proof of lemma 4.1

We prove the result comparing which subgraphs are counted in each side of the equality. First, let S be a subgraph of G that is counted in $C_{\mathcal{A}}(v,G)$, that is, it is connected and contains v. Given S, we can take $A = V(S) \cap D$ as an adequate set included in the right summation from the lemma. In fact, S is counted in the term $C^D_{\mathcal{A}}(\{A\},G)$ since in $C^D_{\mathcal{A}}(\{A\},G)$ we exclude subgraphs containing nodes in $D \setminus A$ and $A \subseteq V(S)$ by construction of A.

Let us suppose there are two sets $A' \neq A''$ such that $A', A'' \subseteq D$ containing v and such that S is counted in $C^D_{\mathcal{A}}(\{A'\},G))$ and $C^D_{\mathcal{A}}(\{A''\},G)$. Since these sets are different, we can suppose there is some $u \in A'$ and $u \notin A''$. As S is counted in $C^D_{\mathcal{A}}(\{A'\},G)$, we have $u \in V(S)$. Since $u \notin A''$, $u \in D$ and $u \notin V(G - \bar{A''})$, any subgraph counted in $C^D_{\mathcal{A}}(\{A''\},G)$ does not contain u. This gives us a contradiction and therefore S is counted in a single term from the right side of the lemma. With this,

$$C_{\mathcal{A}}(v,G) \leq \sum_{A \subseteq D,\ v \in A} C^D_{\mathcal{A}}(\{A\},G).$$

Now, to prove the reverse inequality, let us take S counted in one of the terms $C^D_{\mathcal{A}}(\{A\},G)$. Since $v \in A$ and every subgraph counted in $C^D_{\mathcal{A}}(\{A\},G)$ has one single connected component. S is counted in the term $C_{\mathcal{A}}(v,G)$ from the left side, so

$$C_{\mathcal{A}}(v,G) \geq \sum_{A \subseteq D,\ v \in A} C^D_{\mathcal{A}}(\{A\},G)$$

With these two results we conclude the equality from the lemma.

F.2 Proof of lemma 4.2

In first place, set $D' = D \setminus \{v\}$. For a partition $P = \{A_1, ..., A_k\}$, we consider sets $T(P,D) = \{\, S_1 \uplus \cdots \uplus S_k \subseteq G \mid A_i \subseteq V(S_i), S_i \in \mathcal{A}$ and $(D \setminus \bigcup_i A_i) \cap V(S_i) = \emptyset \,\}$ and

$$W = \bigcup_{P' \in Part(D) : P = P' - \{v\}} T(P', D)$$

We define $f : W \to T(P,D')$ as $f(S) = S$ if v is not isolated in S and $f(S) = S - \{v\}$ if v is isolated. We will now show that f is a bijection.

First we show that it is well defined. Let fix a subgraph $S = S_1 \uplus ... \uplus S_k \in W$ and assume S is in $T(P', D)$ for some $P' = \{A_1, ..., A_{k'}\}$. We have two options: v can appear in S as an isolated vertex or it can be part of a bigger connected component. In the first case, there exists $A_i = \{v\}$ for some $i \in \{1, ..., k'\}$. Therefore, $P = \{A_1, ..., A_{i-1}, A_{i+1}, ..., A_{k'}\}$, i.e $k = k' - 1$. At the same time, we know $v \notin D'$, which means $S - \{v\} = S_1 \uplus S_2 \uplus ... \uplus S_{i-1} \uplus S_{i+1} \uplus ... \uplus S_{k'}$ is in $T(P, D')$. On the other hand, if v is not isolated in S, then every component of S covers exactly one set of P and every set of P is covered by exactly one component. It is important to notice that $P = \{A_1, ..., A_i - \{v\}, A_{i+1}, ..., A_{k'}\}$. In this case we have $k = k'$. Therefore, in particular, there exists a component S_i of S containing v which is not an isolated vertex. Meanwhile, since $v \notin D'$, then S must be in $T(P, D')$ by definition. We just proved that whenever $S \in W$, $f(S) \in T(P, D')$ and f is well defined.

To show that f is surjective, let us take $S \in T(P, D')$. If $v \in V(S)$, there is some component $S_i \in \text{ConnComp}(S)$ such that $v \in V(S_i)$. We also know that $|V(S_i)| > 1$ given that $v \notin D'$ and therefore $v \notin A_i$ for every $i \in \{1, ..., k\}$. Since $S \in T$, every component covers some $A_i \in P$, plus the fact that $P' = \{A_1, ..., A_i + \{v\}, ..., A_k\}$, we can conclude that $S \in W$. By, definition, $f(S) = S$ and therefore S is in the image of f. If $u \notin V(S)$, as $S \in T(P, D')$, every component covers some $A_i \in P$ without containing v. Therefore, if we define $S' = S \cup \{v\}$ were v is an isolated vertex, then S' is included in $T(P', D)$ where $P' = \{A_1, ..., A_k, \{v\}\}$. Therefore, S' is in W and $f(S') = S$.

Because f is clearly injective, we conclude that it is a bijection. Plus the fact that $T(P', D) \cap T(P'', D) = \emptyset$ for $P' \neq P''$, we obtain the expression from the lemma.

F.3 Proof of Lemma 4.3

We will proof each equality separately:

(1) $C_{\mathcal{A}}^{D \cup \{v\}}(P, G + v) = C_{\mathcal{A}}^{D}(P, G)$. In this case, v is not contained in any bag from P, so the count of subgraphs is the same as in the original graph.

(2) $C_{\mathcal{A}}^{D \cup \{v\}}(P \cup \{\{v\}\}, G + v) = C_{\mathcal{A}}^{D}(P, G)$. If v is in a single exclusive bag, it must be covered in its own connected component. Since there are not edges connecting it to other nodes, the component that covers $\{v\}$ must include only v. This corresponds to add this component to any subgraph counted in the original graph G.

(3) Rest of the cases. Since $\{v\} \notin P$, we know that v does not appear as a single bag and therefore, we do not count subgraphs that cover it in a single component. Also, if $v \in \cup P$, we have that v appears in some bag $A \in P$. Since v is not connected to other vertices, it cannot be contained in a single component that also covers A, so $C_{\mathcal{A}}^{D \cup \{v\}}(P, G + v) = 0$.

F.4 Proof of Lemma 4.4

In first place, we will consider sets $\mathcal{A}_D(P, G) = \{ S_1 \uplus \cdots \uplus S_k \subseteq G \mid A_i \subseteq V(S_i), S_i \in \mathcal{A}$ and $(D \setminus \bigcup_i A_i) \cap V(S_i) = \emptyset \}$. Let G, G_1, G_2 graphs as defined in the statement of the lemma. Consider as well a partition P. Let us define the set $T = \mathcal{A}_D(P, G)$ and W given by

$$W = \bigcup_{\substack{P_1, P_2 \in \mathcal{P}[D] \\ P = \sup\{P_1, P_2\}}} \{(S_1, S_2) \mid S_1 \in \mathcal{A}_D(P_1, G_1) \wedge S_2 \in \mathcal{A}_D(P_2, G_2)\}$$

We will show that there is a bijection between T and W. Let $f : W \to T$ be defined as

$$f((S_1, S_2)) = S_1 \cup S_2$$

First we show that f is well defined. Let $(S_1, S_2) \in W$, i.e. S_1 is counted in $C_{\mathcal{A}}^{D}(P_1, G_1)$ and S_2 is counted in $C_{\mathcal{A}}^{D}(P_2, G_2)$ for some pair P_1, P_2. Since P is the supremum of this pair, a set $A \in P$ can be formed by taking the union of a collection of sets from P_1 and P_2. Thus, the union of the components of S_1 and S_2 that cover these sets gives a single component that cover A in G. Since the elements of P are disjoint by definition, $S_1 \cup S_2$ covers every element of P in a different component and therefore is a subgraph counted in $C_{\mathcal{A}}^{D}(P, G)$. As $S_1 \cup S_2 \in T$, f is well defined.

Second we show that f is surjective. If $S \in T$, for $i = 1, 2$ we define

$$S_i = (V(S) \cap V(G_i),\ E(S) \cap E(G_i))$$

It is clear that we can give an expression for P_i, such that $S_i \in \mathcal{A}_D(P_i, G_i)$:

$$P_i = \bigcup_{H \in \text{ConnComp}(S_i)} D \cap V(H)$$

By construction of P_i we have that each bag in P_i is covered by a different component of S_i, so S_i is counted in $\mathcal{A}_D(P_i, G_i)$. Also, $S = S_1 \cup S_2$ since every node and edge of S is contained in one of the smaller subgraphs. Therefore, any $S \in T$ can be decomposed into a pair in W such that $S = f((S_1, S_2))$, and f is surjective.

Third we show that f is injective. Let $(S_1, S_2), (S_3, S_4)$ be different elements in W. Let us remember that G_1 and G_2 only share the nodes from D. Without loss of generality, let us suppose that $S_1 \neq S_3$ and $S_2 = S_4$.

- If there is some $u \in V(S_1)$ and $u \notin V(S_3)$, we know it is not from D, and since it is in G_1, $u \notin V(S_2)$. Then, $u \in S_1 \cup S_2$ and $u \notin S_3 \cup S_4$.
- If there is some $e \in E(S_1)$ and $e \notin E(S_3)$, we have that $e \in E(G_1)$ and $e \notin E(G_2)$. Thus, $e \notin E(S_2)$ and we conclude that $e \in S_1 \cup S_2$ and $e \notin S_3 \cup S_4$.

This proves that f is injective. We conclude that f is a bijection between W and T, and $|W| = |T|$, which proves the lemma. □

F.5 Proof of Lemma 4.6

First, it is easy to see that for any $i > 1$, then $G(T_{x_i,y}) \cup D \cup G_{i-1} = G_i$ by definition since $D \in V(G_i)$ for every $i \geq 0$. Now, we must prove that $V(G(T_{x_i,y}) \cup D) \cap V(G_{i-1}) = D$. Extending the left hand expression we have $V(G(T_{x_i,y}) \cup D) \cap V(G_{i-1}) = (V(G(T_{x_i,y}) \cap V(G_{i-1})) \cup (D \cap V(G_{i-1}))$. It is important to notice that $(V(G(T\_\{x\_i, y\})\backslash\{\}capV(G\_i - 1)) \subseteq D$ since between every two children of G_y, they can only share nodes in $V(G_y) = D$. On the other hand, we know that $D \subseteq V(G_{i-1})$ for every i. Therefore, $D \cap V(G_{i-1})) = D$ and we conclude that $(V(G(T_{x_i,y}) \cap V(G_{i-1})) \cup (D \cap V(G_{i-1})) = D$. Finally, we have that $E(G(T_{x_i,y} \cup D)) \cap E(G_{i-1}) = \emptyset$ since by definition for every pair of adjacent bags x, y then the graphs G_x and G_y do not share edges.

F.6 Proof of Lemma 4.10

This proof is really similar to the one for Lemma 4.4. In first place, we will consider sets $\mathcal{T}_D(P, G) = \{\, S_1 \uplus \cdots \uplus S_k \subseteq G \mid A_i \subseteq V(S_i), S_i \in \mathcal{T}$ and
$(D \setminus \bigcup_i A_i) \cap V(S_i) = \emptyset \,\}$. Let G, G_1, G_2 graphs as defined in the statement of the lemma. Consider as well a partition P. Let us define the set $T = \mathcal{T}_D(P, G)$ and W given by

$$W = \bigcup_{\substack{P_1,P_2 \\ P=\sup\{P_1,P_2\} \\ P_1|P_2}} \{(S_1, S_2) \mid S_1 \in \mathcal{T}_D(P_1, G_1) \wedge S_2 \in \mathcal{T}_D(P_2, G_2)\}$$

We will show that there is a bijection between T and W. Let $f : W \to T$ be defined as

$$f((S_1, S_2)) = S_1 \cup S_2$$

First we show that f is well defined. Let $(S_1, S_2) \in W$, i.e. S_1 is counted in $C^D_{\mathcal{T}}(P_1, G_1)$ and S_2 is counted in $C^D_{\mathcal{T}}(P_2, G_2)$ for some pair P_1, P_2. Since P is the supremum of this pair, a set $A \in P$ can be formed by taking the union of a collection of sets from P_1 and P_2. Thus, the union of the components of S_1 and S_2 that cover these sets gives a single component that cover A in G. Since the elements of P are disjoint by definition, $S_1 \cup S_2$ covers every element of P in a different component. Finally, because we have the restriction that $P_1|P_2$, we know for each pair of sets $A_1 \in P_1$ and $A_2 \in P_2$ then $|A_1 \cap A_2| \leq 1$. Therefore, if S_1 is a tree in $C^D_{\mathcal{T}}(P_1, G_1)$ while S_2 is in $C^D_{\mathcal{T}}(P_2, G_2)$, then there is no way $S_1 \cup S_2$ have cycles. In other words, $S_1 \cup S_2$ is a tree. In conclusion, $S_1 \cup S_2$ is a subgraph counted in $C^D_{\mathcal{T}}(P, G)$. As $S_1 \cup S_2 \in T$, f is well defined.

Second we will show that f is surjective. If $S \in T$, for $i = 1, 2$ we define

$$S_i = (V(S) \cap V(G_i),\ E(S) \cap E(G_i))$$

It is clear that we can give an expression for P_i, such that $S_i \in \mathcal{T}_D(P_i, G_i)$:

$$P_i = \bigcup_{H \in \mathrm{ConnComp}(S_i)} D \cap V(H)$$

By construction of P_i we have that each bag in P_i is covered by a different component of S_i, so S_i is counted in $\mathcal{T}_D(P_i, G_i)$. Also, $S = S_1 \cup S_2$ since every node and edge of S is contained in one of the smaller subgraphs. This fact plus that S is a tree, implies that $P_1 \mid P_2$ necessarily. Therefore, any $S \in T$ can be decomposed into a pair in W such that $S = f((S_1, S_2))$, and f is surjective.

Third we show that f is injective. Let $(S_1, S_2), (S_3, S_4)$ be different elements in W. Let us remember that G_1 and G_2 only share the nodes from D. Without loss of generality, let us suppose that $S_1 \neq S_3$ and $S_2 = S_4$.

- If there is some $u \in V(S_1)$ and $u \notin V(S_3)$, we know it is not from D, and since it is in G_1, $u \notin V(S_2)$. Then, $u \in S_1 \cup S_2$ and $u \notin S_3 \cup S_4$.
- If there is some $e \in E(S_1)$ and $e \notin E(S_3)$, we have that $e \in E(G_1)$ and $e \notin E(G_2)$. Thus, $e \notin E(S_2)$ and we conclude that $e \in S_1 \cup S_2$ and $e \notin S_3 \cup S_4$.

This proves that f is injective. We conclude that f is a bijection between W and T, and $|W| = |T|$, which proves the lemma. □

F.7 Extension for Lines

We aim to compute the values $C_{\mathcal{P}}(v, G)$ for a graph G and a node v in G.

In the same spirit as we did for the all subgraph family, we need a new structure for partitions over a cut set. It is necessary to be more precise on which kind of lines we are counting. Let D be any set of nodes. For a set of pairs $P \subseteq D^2$, we say $P = \{(A_1, B_1), (A_2, B_2), ..., (A_n, B_n)\}$ is a sub partition of D if the following holds:

(1) $\bigcup_{i=1}^{n} A_i \subseteq D$.
(2) For every $i \in \{1, ..., n\}$, $A_i \neq \emptyset$.
(3) For every $i \neq j$ then $A_i \cap A_j = \emptyset$.
(4) For every $i \in \{1, ..., n\}$, then $B_i \subseteq A_i$ and $|B_i| \leq 2$.

Now we can use sub partitions to specify which lines we are counting. Given a set of nodes D, a partition $P = \{(A_1, B_1), (A_2, B_2), ..., (A_n, B_n)\}$, a graph G and a subgraph of G, $S = L_1 \cup ... \cup L_n \cup H_{v_1}, ..., H_{v_k}$ where L_i are line graphs and H_{v_i} is the isolated vertex v_i, define the following properties:

(1) For every $i \neq j$, $L_i \cap L_j = \emptyset$.
(2) For every $i \in \{1, ..., n\}$, $A_i \subseteq V(L_i)$.
(3) $\left(D / \bigcup_{i=1}^{n} A_i\right) \cap \left(\bigcup_{i=1}^{n} V(L_i)\right) = \emptyset$. In other words, nodes not in the lines are nodes not covered by the partition.
(4) For every $i \in \{1, ..., n\}$, then $|V(L_i)| > 1$.
(5) For every B_i, then:
   (a) If $B_i = \{v\}$ then v is one end of L_i and the other end is not in A_i.
   (b) If $B_i = \{u, v\}$ then u and v are both ends of L_i.
   (c) If $B_i = \emptyset$ then the ends of L_i does not belong to A_i.
(6) For every $v \in D / \bigcup_{i=1}^{n} A_i$ there exists a unique $i \in \{1, ..., k\}$ such that $v_i = v$.

Now, given a set D, a partition P and a graph G, we define the set

$$C^D_{\mathcal{P}}(P, G) = |L_D(P, G)| = |\{S = L_1 \cup ... \cup L_n \cup H_{v_1}, ..., H_{v_k} \subseteq G \mid \text{ such that properties 1, 2, 3, 4, 5 and 6 are satisfied.}\}|.$$

Finally, for a graph G, a set of nodes $D \subseteq V(G)$ and $A \subset D$, define the amount of lines that cover A in G with respect to D as:

$$|L_D(A, G)| = \sum_{B \subseteq A : |B| \leq 2\}} |L_D(\{(A, B)\}, G)|.$$

We can introduce the main results necessary to compute the amount of lines containing a node v in G.

Theorem. Let G be a graph, v a node in G and a set of nodes $D \subseteq V(G)$ such that v is in D. Then

$$C_{\mathcal{P}}(v, G) = \sum_{A \subseteq D v \in A} C^D_{\mathcal{P}}(A, G) + 1.$$

Proof. This proof is equivalent to the one for lemma 4.1 but we don't count the isolated vertex, that is why we must add one subgraph at the end. □

In order to introduce our last result we need some notation. First, given a partition $P = \{(A_1, B_1), ..., (A_n, B_n)\}$, we define the projections $\pi_1(P) = \{A_1, ..., A_n\}$ and $\pi_2(P) = \{B_1, B_2, ..., B_n\}$. On the other hand, for two partitions P_1 and P_2 define the following graph. Define the set of edges $E_{1,2} = \{B \in \pi_2(P_1) \cup \pi_2(P_2) \mid |B| = 2\}$. Now, define the set of nodes $V_{1,2} = \bigcup \pi_2(P_1) \cup \bigcup \pi_2(P_2)$. Finally, we can define the graph $G_{1,2} = (V_{1,2}, E_{1,2})$. For two partitions P_1 and P_2 such that, first, for every $(A_1, B_1) \in P_1$ and $(A_2, B_2) \in P_2$ then $|A_1 \cap A_2| \leq 1$ and $A_1 \cap A_2 \subseteq B_1 \cap B_2$. In second place, if $G_{1,2}$ has no cycles, then we can define the operation $P_1 \star P_2 = P$ where $P = \{(A^\star_1, B^\star_1), ..., (A^\star_k, B^\star_k)\}$ satisfies the following properties:

(1) $\pi_1(P) = \sup\{\pi_1(P_1), \pi_1(P_2)\}$.
(2) For $A_i \in \pi_1(P_1)$ and $A_j \in \pi_1(P_2)$ such that $A_i \subseteq A^\star_l$ and $A_j \subseteq A^\star_l$ then $B^\star_l = \{v_1, ..., v_m\}$ such that for every $s \in \{1, ..., m\}$ then $v_s \in B_i$ and $v_s \in (D / \cup \pi_1(P_1))$ or $v_s \in B_j$ and $v_s \in (D / \cup \pi_1(P_2))$.

Theorem. Fix two graphs G_1 and G_2 such that $V(G_1) \cap V(G_2) = D \neq \emptyset$, $E(G_1) \cap E(G_2) = \emptyset$ and $G = G_1 \cup G_2$, then for a partition P,

$$C^D_{\mathcal{F}}(P, G) = \sum_{P_1, P_2 : P_1 \star P_2 = P} C^D_{\mathcal{F}}(P_1, G_1) \cdot C^D_{\mathcal{F}}(P_2, G_2).$$

Proof. In the same spirit of Lemma 4.4. In first place, we will consider sets $L_D(P, G)$ as defined before. Let G, G_1, G_2 graphs as defined in the statement of the lemma. Consider as well a partition $P = \{(A_1, B_1), ..., (A_k, B_k)\}$. Let us define the set $T = L_D(P, G)$ and W given by

$$W = \bigcup_{\substack{P_1, P_2 \\ P = \sup\{P_1, P_2\} \\ P_1 | P_2}} \{(S_1, S_2) \mid S_1 \in L_D(P_1, G_1) \ \wedge \ S_2 \in L_D(P_2, G_2)\}$$

We will show that there is a bijection between T and W. Let $f : W \to T$ be defined as

$$f((S_1, S_2)) = S_1 \cup S_2$$

First we show that f is well defined. Let $(S_1, S_2) \in W$, i.e. S_1 is counted in $C^D_{\mathcal{T}}(P_1, G_1)$ and S_2 is counted in $C^D_{\mathcal{T}}(P_2, G_2)$ for some pair P_1, P_2. Since $\pi_1(P)$ is the supremum of the pair $\pi_1(P_1), \pi_1(P_2)$, a set $A \in \pi_1(P)$ can be formed by taking the union of a collection of sets from $\pi_1(P_1)$ and $\pi_1(P_2)$. Thus, the union of the components of S_1 and S_2 that cover these sets gives a single component that cover A in G. Since the elements of $\pi_1(P)$ are disjoint by definition, $S_1 \cup S_2$ covers every element of $\pi_1(P)$ in a different component. Notice that because $G_{1,2}$ has no cycles, we can be sure that after merging S_1 and S_2 we do not generate cycles. Finally, because we have the restriction that $P_1 \star P_2$, we know for each pair of sets $A_1 \in \pi_1(P_1)$ and $A_2 \in \pi_1(P_2)$ such that $A_i \subseteq A^\star_l$ and $A_j \subseteq A^\star_l$ then $B^\star_l = \{v_1, ..., v_m\}$ such that for every $s \in \{1, ..., m\}$ then $v_s \in B_i$ and $v_s \in (D / \cup \pi_1(P_1))$ or $v_s \in B_j$ and $v_s \in (D / \cup \pi_1(P_2))$. This property ensures that whenever v_s is an isolated node in S_1 or S_2 then it is a starting node in $S_1 \cup S_2$. Therefore, if S_1 is a line in $L_D(P_1, G_1)$ while S_2 is a line in $L_D(P_2, G_2)$, then $S_1 \cup S_2$ is a line in $L_D(P, G)$

where the nodes v_s are starting nodes while the rest of the nodes in $D \setminus \cup\pi_1(P_1) \cup \pi_2(P_2)$ are isolated nodes. In conclusion, $S_1 \cup S_2$ is a subgraph counted in $C_{\mathcal{T}}^{D}(P, G)$. As $S_1 \cup S_2 \in T$, f is well defined.

Second we will show that f is surjective. If $S \in T$, for $i = 1, 2$ we define

$$S_i = (V(S) \cap V(G_i),\ E(S) \cap E(G_i))$$

It is clear that we can give an expression for P_i, such that $S_i \in \mathcal{T}_D(P_i, G_i)$:

$$\pi_1(P_i) = \bigcup_{H \in \text{ConnComp}(S_i)} D \cap V(H).$$

On the other hand, we define $\pi_2(P_i)$ as the set of all start and ending points of each line L_i in $\text{ConnComp}(S_i)$ that are not isolated nodes. By construction of P_i we have that each bag in $\pi_1(P_i)$ is covered by a different component of S_i, so S_i is counted in $\mathcal{P}_D(P_i, G_i)$. Also, $S = S_1 \cup S_2$ since every node and edge of S is contained in one of the smaller subgraphs. This fact plus that S is a line, implies that $P_1 \star P_2$ necessarily. Therefore, any $S \in T$ can be decomposed into a pair in W such that $S = f((S_1, S_2))$, and f is surjective.

Third we show that f is injective. Let $(S_1, S_2), (S_3, S_4)$ be different elements in W. Let us remember that G_1 and G_2 only share the nodes from D and no edges. Without loss of generality, let us suppose that $S_1 \neq S_3$ and $S_2 = S_4$.

- If there is some $u \in V(S_1)$ and $u \notin V(S_3)$, we know it is not from D, and since it is in G_1, $u \notin V(S_2)$. Then, $u \in S_1 \cup S_2$ and $u \notin S_3 \cup S_4$.
- If there is some $e \in E(S_1)$ and $e \notin E(S_3)$, we have that $e \in E(G_1)$ and $e \notin E(G_2)$. Thus, $e \notin E(S_2)$ and we conclude that $e \in S_1 \cup S_2$ and $e \notin S_3 \cup S_4$.

This proves that f is injective. We conclude that f is a bijection between W and T, and $|W| = |T|$, which proves the lemma. □ □

G FORMAL DEFINITION OF STATISTICAL COEFFICIENT USED IN SECTION 6

The first approach that we use to compare centrality measures is the well-known Pearson correlation coefficient, which we use to compare the centrality absolute values. Given a graph $G = (V, E)$, let $C_1(i, G)$ and $C_2(i, G)$ be the values of centrality measures C_1 and C_2 for node $i \in V$ We define the Pearson correlation coefficient as follows

$$p = \frac{\sum_i (C_1(i, G) - \bar{C_1})(C_2(i, G) - \bar{C_2})}{\sqrt{\sum_i (C_1(i, G) - \bar{C_1})^2}\ \sqrt{\sum_i (C_2(i, G) - \bar{C_2})^2}}$$

where $\bar{C_j}$ is the mean value of the centrality measure when computed on all nodes of G. This coefficient is used to compare correlation between centrality measures in [37] and [18], and as noted in [51] it has been used for comparing complex centrality measures to simpler ones.

Our second method is the Spearman correlation coefficient, used in [52] for comparing lists of rank values. We note that it corresponds to obtaining the Pearson correlation coefficient for the rank values, instead of the centrality values. Let $\sigma_j : V \to \{1, \ldots, |V|\}$ be the permutation of nodes of G given the ranking generated by centrality measure C_j. In other words, $\sigma_1(i)$ is the ranked position of node i given centrality measure C_1. With this notation, and using a similar definition as p, we have the following form for Spearman coefficient

$$s = \frac{\sum_i (\sigma_1(i) - |V|/2)(\sigma_2(i) - |V|/2)}{\sqrt{\sum_i (\sigma_1(i) - |V|/2)^2}\ \sqrt{\sum_i (\sigma_2(i) - |V|/2)^2}}.$$

The Spearman coefficient is also used to compare centrality measures in [18] and [51].

The third approach, as defined in [16], is the Kendall's Tau distance, which depends on how many pairs of exchanges we should apply to transform one ranked list into another:

$$\tau = \frac{2|D|}{n(n-1)}$$

where D is the set of discordant pairs between the two permutation σ_1 and σ_2. This distance is also used alongside with the Spearman coefficient in [18] and [51].

We also measure absolute similarity. I.e we compare rankings directly. For each graph G and pair of measures C_1 and C_2 we define similarity of C_1 and C_2 over the top-k% as the proportion of the top-k% nodes C_1 and C_2 share. We count the coincidence of elements between all-subgraph (all-trees) and each measure over every graph and for different percentage of top-k nodes. Finally we compare mean of the values described as shown in figure 4 for all-subgraphs and all-trees respectively.