Computing well-balanced spanning trees of unweighted networks

Lovro Šubelj[1,2*]

1 University of Ljubljana, Faculty of Computer and Information Science, Ljubljana, Slovenia
2 University of Ljubljana, Faculty of Social Sciences, Ljubljana, Slovenia

* lovro.subelj@fri.uni-lj.si

Abstract

A spanning tree of a network or graph is a subgraph that connects all nodes with the least number or weight of edges. The spanning tree is one of the most straightforward techniques for network simplification and sampling, and for discovering its backbone or skeleton. Prim's algorithm and Kruskal's algorithm are well-known algorithms for computing a spanning tree of a weighted network, and are therefore also the default procedure for unweighted networks in the most popular network libraries. In this paper, we empirically study the performance of these algorithms on unweighted networks and compare them with different priority-first search algorithms. We show that the structure of a network, such as the distances between the nodes, is better preserved by a simpler algorithm based on breadth-first search. The spanning trees are also most compact and well-balanced as measured by classical graph indices. We support our findings with experiments on synthetic graphs and more than a thousand real networks, and demonstrate practical applications of the computed spanning trees. We conclude that if a spanning tree is to maintain the structure of an unweighted network, the breadth-first search algorithm should be the preferred choice.

Introduction

Networks or graphs have become a popular tool for analyzing complex real-world systems [1]. Examples include predicting the spread of contagious viruses [2], the study of the interactome of species [3], understanding the structure of science [4] and outreach of online social connections [5]. The size of today's networks is often in millions of nodes and edges, with the largest networks being the WWW with more than a trillion web pages and the human brain with nearly a hundred billion neurons. As a result, the size of real networks makes many practical applications computationally very challenging.

Techniques to alleviate this issue include network simplification or sampling [6–8] and revealing the so-called network backbone or skeleton [9–12]. These approaches try to reduce the size of a network in a way that the network still retains many of its structural properties. One of the most straightforward ways to simplify a network is to compute its spanning tree [1, 13], which is a subgraph connecting all the nodes of a network with the minimum number of edges. Spanning trees retain the connectivity of networks and possibly other structural properties, and have gained considerable interest in recent years [14–18]. In the case of weighted networks, one usually aims to compute the minimum spanning tree, which is a subgraph connecting all the nodes with the

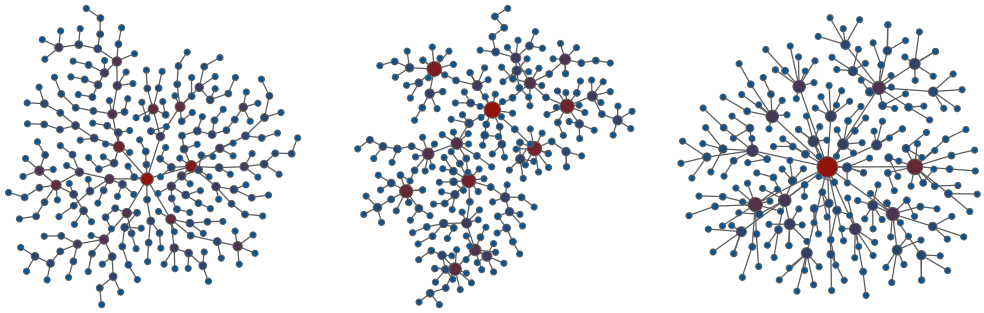

Fig 1. Wiring diagrams of spanning trees of a small random graph. The spanning trees were computed with Prim's algorithm (left), Kruskal's algorithm (middle) and the breadth-first search algorithm (right). The size of the nodes is proportional to their degree, while the layouts were computed with the Large Graph Layout algorithm [25].

minimum total weight of the edges. In the case of unweighted networks, any spanning tree is, in fact, a "minimum" spanning tree.

Prim's and Kruskal's algorithms are well-known algorithms for computing a minimum spanning tree of a weighted network [1, 13]. Although the algorithms were primarily developed for weighted networks, they can be readily applied to unweighted networks where each edge has the same weight. This is also the default procedure in the most popular network analysis libraries *NetworkX*, *igraph*, and *graph-tool* [19]. However, the performance of these algorithms has not been sufficiently studied for unweighted networks, which are much more common in practical applications due to their simplicity [20]. In particular, there exist no theoretical guarantees for the algorithms, and neither does the literature provide any empirical comparison on large-scale networks.

A fundamental result of metric embedding theory, Bourgain's theorem [21], states that any finite metric space can be embedded into a tree metric with $\mathcal{O}(\log n)$ distortion. This suggests that while spanning trees provide a reasonable approximation of network structure, they inevitably introduce some bias that must be evaluated empirically.

In this paper, we apply the algorithms to different synthetic graphs and more than a thousand real networks, and compare them to different priority-first search algorithms. We show that the structure of unweighted networks is best preserved by an algorithm using the breadth-first search node traversal. More precisely, spanning trees computed with the breadth-first search algorithm best preserve the distances between the nodes of a network. Recall that other standard network measures, such as the average node degree or clustering coefficient [22], are fixed by construction. The spanning trees are also most compact and well-balanced as measured by Wiener's index [23] and Sackin's index [24]. Such structural properties are often desirable in practical applications such as network visualization (Fig 1). Furthermore, the breadth-first search algorithm can be up to 20 times faster and uses less memory than alternatives.

The rest of the paper is structured as follows. In the following section, we first describe different algorithms for computing a spanning tree of a network, as well as different indices of tree balance and compactness. Next, we analyze the structure of spanning trees of synthetic graphs and then the structure of spanning trees of real networks. Finally, we demonstrate practical applications of spanning trees of networks.

Methods

Let a network be represented by an undirected connected graph $G = (V, E)$, where V denotes the set of nodes of G and E denotes the set of edges of G. Where necessary to make explicit that these sets represent the graph G, we write V_G and E_G. The number of nodes equals $n = |V|$ and the number of edges equals $m = |E|$. We denote the average node degree as $\langle k \rangle = 2m/n$. Furthermore, let d_{ij} be the distance between the nodes $i, j \in V$, defined as the number of edges in a shortest path between the nodes i and j. Since the graph is undirected and connected, $d_{ij} = d_{ji}$ and $d_{ij} < \infty$. Therefore, the average distance between the nodes equals $\langle d \rangle = \frac{2}{n(n-1)} \sum_{i<j} d_{ij}$ and the maximum distance or diameter equals $d_{\max} = \max_{i<j} d_{ij}$. In order to measure the variability of the distances between the nodes, we also define the coefficient of variation as $c_d = \sigma_d / \langle d \rangle$, where σ_d is the standard deviation of distances. The coefficient of variation c_d is a measure of the dispersion of a probability distribution, where the distributions with $c_d < 1$ are considered low-variance distributions, while those with $c_d > 1$ are considered high-variance distributions.

Below, we describe different algorithms for computing a spanning tree of an undirected connected graph. In the case of a disconnected graph consisting of more than one connected component, the algorithms should be applied to each of the connected components separately. Implementation of these algorithms is provided by most standard network analysis libraries. For a more extensive discussion on the differences between the algorithms, we refer the reader to classical graph theory literature [13] or network science literature [1, 26].

Prim's algorithm

Prim's algorithm for computing a spanning tree T of an undirected connected graph G operates as follows (see Algorithm 1). First, the algorithm selects a random seed node $i \in V_G$ from graph G and adds it to an empty tree T (lines 2, 3). The node i serves as a starting point for computing the spanning tree T. Then, on each step of the algorithm (lines 4-8), a random edge $\{i, j\} \in E_G$ from graph G is selected that leads from a node $i \in V_T$ already in the tree T to a node $j \notin V_T$ not yet in the tree T (line 5). Both node j and edge $\{i, j\}$ are added to the tree T (lines 6, 7). Finally, when there is no further node $i \in V_G$ in graph G such that node $i \notin V_T$ is not already in the tree T, the algorithm stops (line 4). At this point, the tree T is a spanning tree of graph G (line 9).

Algorithm 1 Prim's algorithm

Require: undirected graph G
Ensure: spanning tree T
1: $T \leftarrow$ empty graph
2: $i \leftarrow \textsc{Random}(i \in V_G)$ ▷ Random seed node.
3: add node i to V_T ▷ Add selected seed node.
4: **while** $\exists i \in V_G : i \notin V_T$ **do** ▷ There exists non-visited node?
5: $\quad \{i, j\} \leftarrow \textsc{Random}(\{i, j\} \in E_G : i \in V_T \wedge j \notin V_T)$ ▷ Edge to non-visited node.
6: $\quad$ add node j to V_T ▷ Add non-visited node.
7: $\quad$ add edge $\{i, j\}$ to E_T ▷ Add selected edge.
8: **end while**
9: **return** T

Prim's algorithm is nondeterministic and can compute different spanning trees. The actual spanning tree depends on a random selection of the seed node (line 2) and on a random selection of the edges to expand the tree (line 5). The latter is most efficiently implemented by rejection sampling over an array list of edges to non-visited nodes. For

simplicity, we do not make these computations explicit in Algorithm 1.

Assume that the graph is represented with an adjacency list. In the case of weighted graphs, the time complexity of Prim's algorithm implemented with a Fibonacci heap is $\mathcal{O}(m + n \log n)$. For unweighted graphs, which we consider in this paper, the heap can be replaced by an array list, which reduces the time complexity to $\mathcal{O}(m)$. As an example, the left graph in Fig 1 shows a spanning tree computed with Prim's algorithm.

Kruskal's algorithm

Kruskal's algorithm differs conceptually from Prim's algorithm. Instead of starting with a tree consisting of a seed node and then expanding it, the algorithm starts with a forest of trees, each consisting of a single node. The trees are then incrementally merged into larger trees by adding edges between them until only one remains. At this point the algorithm stops and the remaining tree is a spanning tree of a graph. Since the algorithm turns out inefficient for our purposes in this paper, we do not provide the exact pseudocode here.

Kruskal's algorithm is nondeterministic and the actual spanning tree depends on a random selection of the edges to merge the trees at each step. The time complexity of the algorithm using a disjoint-set data structure is $\mathcal{O}(m \log n)$, for either weighted or unweighted graphs. As an example, the middle graph in Fig 1 shows a spanning tree computed with Kruskal's algorithm.

Breadth-first search

The breadth-first search node traversal is very similar to Prim's algorithm. The main difference is in how the edges to non-visited nodes are processed. In contrast to Prim's algorithm, where only one such edge is processed on each step, the breadth-first search processes all edges from a selected node to non-visited nodes in a single step.

The breadth-first search algorithm for computing a spanning tree T of an undirected connected graph G operates as follows (see Algorithm 2). In contrast to before, we make all computations in Algorithm 2 explicit. First, the algorithm selects a random seed node $i \in V_G$ from graph G and adds it to an empty tree T (lines 3, 4). The node i is also added to an empty queue $Q \subseteq V_T$. Then, on each step of the algorithm (lines 5-11), a node $i \in Q$ is removed from the beginning of the queue Q (line 6) and all edges that lead from node $i \in V_T$ already in the tree T to nodes $j \notin V_T$ not yet in the tree T are processed (lines 7-10). All nodes j and edges $\{i, j\}$ are added to the tree T (lines 8, 9), while nodes j are also added to the queue Q for further processing. Finally, when there is no other node $i \in Q$ in the queue Q, the algorithm stops (line 5). At this point the tree T is a spanning tree of graph G (line 12).

The breadth-first search algorithm is again nondeterministic, while the actual spanning tree depends on a random selection of the seed node (line 3) and on the exact order in which the edges to expand the tree are processed (line 7). The time complexity of the algorithm using a queue of non-processed nodes is $\mathcal{O}(m)$, for either weighted or unweighted graphs. As an example, the right graph in Fig 1 shows a spanning tree computed with the breadth-first search algorithm.

Other algorithms

Other algorithms for computing a spanning tree include Sollin's algorithm, which can be seen as a combination of Prim's and Kruskal's approaches, and the depth-first search node traversal. While the breadth-first search algorithm processes nodes of a graph using a level-order traversal, the depth-first search algorithm uses a preorder traversal. This means that the only change required to the breadth-first search algorithm is to

Algorithm 2 Breadth-first search

Require: undirected graph G
Ensure: spanning tree T
1: $T \leftarrow$ empty graph
2: $Q \leftarrow$ empty queue
3: $i \leftarrow \textsc{Random}(i \in V_G)$ ▷ Random seed node.
4: add node i to V_T and Q ▷ Add selected seed node.
5: **while** $\exists i \in Q$ **do** ▷ There exists non-processed node?
6: $i \leftarrow$ remove node from Q ▷ Select first non-processed node.
7: **for** $\{i, j\} \in E_G : j \notin V_T$ **do** ▷ Edges to non-visited nodes.
8: add node j to V_T and Q ▷ Add non-visited node.
9: add edge $\{i, j\}$ to E_T ▷ Add selected edge.
10: **end for**
11: **end while**
12: **return** T

replace the queue of non-processed nodes Q with a stack (line 2 in Algorithm 2). The time complexity of the depth-first search algorithm is again $\mathcal{O}(m)$.

Sackin's index

In the language of computational theory, a balanced tree is a data structure where the time complexity of standard operations such as adding or deleting an element is $\mathcal{O}(\log n)$ [27]. Consider a rooted tree T with root $r \in V_T$ and let $\widetilde{V}_T \subseteq V_T \setminus \{r\}$ be the set of tree leaves (*i.e.*, degree-1 nodes). Then, balance implies that the distance d_{ir} between all leaf nodes $i \in \widetilde{V}_T$ and the root $r \in V_T$ is at most $\mathcal{O}(\log n)$, which further implies that the average distance between all pairs of nodes $\langle d \rangle$ and also the diameter $d_{\max}$ are in $\mathcal{O}(\log n)$, since one can always take a path through the root.

Phylogenetics literature defines various indices of tree balance [28, 29] that quantify the branching symmetry and compactness of trees. One of the most widely used is Sackin's index of imbalance [24]. Sackin's index is defined as the sum of the number of nodes between all leaves $i \in \widetilde{V}_T$ and the root $r \in V_T$, which is included in the count. This can be equivalently written as $\sum_{i \in \widetilde{V}_T} d_{ir}$. Since Sackin's index tends to increase with the number of nodes, we normalize by the minimum possible value $\widetilde{n} = |\widetilde{V}_T|$, which is reached on the star tree, and the maximum possible value $(\widetilde{n} + 2)(\widetilde{n} - 1)/2$, which is reached on the caterpillar tree. The normalized Sackin's index S [30] is then defined as

$$S = \frac{\sum_{i \in \widetilde{V}_T} d_{ir} - \widetilde{n}}{(\widetilde{n} + 2)(\widetilde{n} - 1)/2 - \widetilde{n}},$$

where smaller values correspond to more balanced trees.

For spanning trees computed with the breadth-first search and Prim's algorithms, we designate the randomly selected seed node as the root. Spanning trees computed with Kruskal's algorithm, on the other hand, do not have a naturally defined root. We, therefore, randomly select different nodes as the root and average the results.

Results

Synthetic graphs

Consider an Erdős-Rényi random graph [31] with n nodes and the probability of an edge between each pair of nodes $p = \langle k \rangle / (n - 1)$, where $\langle k \rangle$ is the expected node degree.

A spanning tree of any connected graph with n nodes consists of n nodes and $n-1$ edges. Thus, the average node degree is $\langle k \rangle = 2 - 2/n$. Since it is a tree, the average clustering coefficient is equal to $\langle C \rangle = 0$ [22]. Therefore, we focus on other graph properties here. In particular, we study the average distance between the nodes $\langle d \rangle$ and the diameter $d_{\max}$. A theoretical estimate for the diameter $d_{\max}$ of a random graph equals $\log n / \log\langle k \rangle$ [1], which is $d_{\max} = 2.40$ for $n = 250$ and $\langle k \rangle = 10$. Due to the sensitivity of the diameter $d_{\max}$ for relatively small n and $\langle k \rangle$, this turns out to be a better estimate of the average distance between the nodes $\langle d \rangle$. Indeed, the empirical estimate for the considered random graph is $\langle d \rangle \approx 2.64$ whereas $d_{\max} \approx 4.39$.

Fig 1 shows particular realizations of spanning trees of a random graph with the above parameters computed using Prim's algorithm, Kruskal's algorithm and the breadth-first search algorithm. The diameter $d_{\max}$ of the spanning trees is equal to 14, 17 and 6, respectively. While the diameters of the spanning trees computed with Prim's and Kruskal's algorithms are much higher than in the random graph, the diameter of the spanning tree computed with the breadth-first search algorithm is very close. These observations are closely related to the question of whether the computed spanning trees are balanced.

The average distance $\langle d \rangle$ and the diameter $d_{\max}$ of a balanced tree are in $\mathcal{O}(\log n)$ for any practical definition of balance [27]. However, in the case of a random tree, both values are almost certainly in $\mathcal{O}(\sqrt{n})$ [32, 33]. Since these results only talk about the scaling, they can not be directly employed to measure whether a particular spanning tree is balanced or not. Nevertheless, one can study the scaling of the average distance $\langle d \rangle$ and the diameter $d_{\max}$ of spanning trees of graphs with an increasing number of nodes n and empirically estimate whether the values scale as $\mathcal{O}(\log n)$ or worse. Note that only in the case of the former the spanning trees can possibly retain short distances between the nodes in random graphs and real small-world networks [22].

Besides Erdős-Rényi random graphs [31], we also analyse triangular lattices and Barabási-Albert scale-free graphs [34]. We vary the number of nodes n, while we keep the average node degree fixed to $\langle k \rangle = 10$.[1] Fig 2 shows the scaling of the average distance $\langle d \rangle$ and the diameter $d_{\max}$ for synthetic graphs and their spanning trees computed with different algorithms.

We first consider triangular lattices, as these results serve as a baseline for further analysis. The average distance $\langle d \rangle$ and the diameter $d_{\max}$ of any two-dimensional lattice scale as $\mathcal{O}(\sqrt{n})$ [1]. This can be observed as a straight line with slope 0.5 on double logarithmic plots in the left column of Fig 2. Notice that all spanning trees computed with different algorithms show similar scaling $\mathcal{O}(\sqrt{n})$.

Next, we consider Erdős-Rényi random graphs [31] shown in the middle column of Fig 2. The average distance $\langle d \rangle$ and the diameter $d_{\max}$ of random graphs, and also small-world networks [22], scale as $\mathcal{O}(\log n)$ [1]. This can be observed as a straight line on semi-logarithmic plots in Fig 2, whereas any upward concave function would imply a faster scaling than $\mathcal{O}(\log n)$. Notice that the spanning trees computed with the breadth-first search algorithm best preserve the distances in random graphs, while both the average distance $\langle d \rangle$ and the diameter $d_{\max}$ appear to scale as $\mathcal{O}(\log n)$, at least for a moderate number of nodes $n \leq 10^4$. In contrast, the distances between the nodes of the spanning trees computed with Kruskal's algorithm scale faster than $\mathcal{O}(\log n)$ (see also Fig 3 and the discussion alongside).

Last, we consider Barabási-Albert scale-free graphs [34] shown in the right column of Fig 2. The average distance $\langle d \rangle$ and the diameter $d_{\max}$ of scale-free graphs scale as $\mathcal{O}(\log n / \log\log n)$ [35], while such graphs are usually called ultra small-world [26]. Note that $\mathcal{O}(\log n / \log\log n)$ is indistinguishable from $\mathcal{O}(\log n)$ for $n \leq 10^4$, thus this scaling

[1] Realizations of synthetic graphs analyzed in the paper are available as Pajek files at `https://doi.org/10.5281/zenodo.15034997`.

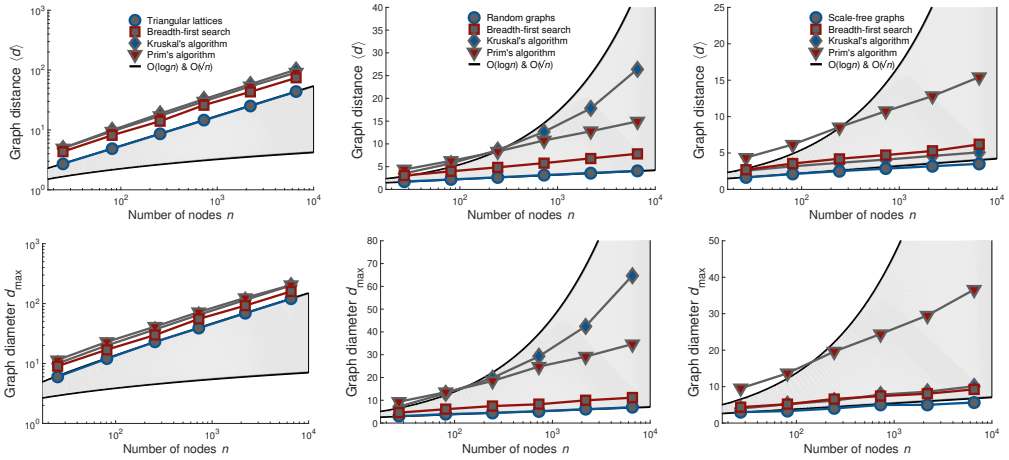

Fig 2. The average distance $\langle d \rangle$ and the diameter $d_{\max}$ of triangular lattices (left), random graphs (middle) and scale-free graphs (right), and their spanning trees computed with different algorithms. The plots show estimates over 100 realizations, where the shaded areas span between theoretical estimates for random graphs $\mathcal{O}(\log n)$ and two-dimensional lattices $\mathcal{O}(\sqrt{n})$, and are consistent between the plots.

can again be observed as a straight line on semi-logarithmic plots in Fig 2. The spanning trees computed with both the breadth-first search algorithm and Kruskal's algorithm well retain the distances between the nodes of scale-free graphs and appear to scale as $\mathcal{O}(\log n)$. On the other hand, the distances between the nodes of the spanning trees computed with Prim's algorithm can be more than five times larger than the distances in scale-free graphs (*e.g.*, $\langle d \rangle = 15.42$ and $d_{\max} = 36.60$ compared to 3.51 and 5.64 for graphs with $n = 6\,561$ nodes).

The above observations are confirmed in Fig 3, where we show the coefficient of variation of the distances between the nodes c_d. Note that the distributions of the distances between the nodes in random and scale-free graphs, and real small-world networks, are low-variance with $c_d \ll 1$ [22, 26]. As one can observe in Fig 3, all distributions of the distances in the spanning trees computed with the breadth-first search algorithm are low-variance $c_d \ll 1$. On the other hand, the distributions in the spanning trees computed with Kruskal's algorithm are high-variance $c_d \gg 1$ for random graphs with $n > 100$, while the results for Prim's algorithm are inconclusive $c_d \approx 1$.

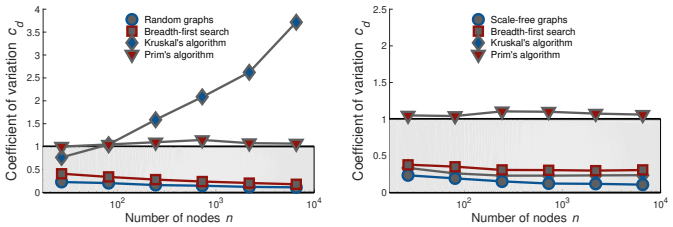

Fig 3. The coefficient of variation c_d for random graphs (left) and scale-free graphs (right), and their spanning trees computed with different algorithms. The plots show estimates over 100 realizations, while the error bars are smaller than the symbol sizes.

To summarize, if a spanning tree should retain short distances between the nodes of a graph, then the breadth-first search algorithm is the preferred choice, at least for random and scale-free graphs. In the following section, we also consider real networks.

Real networks

Table 1 shows statistics of collections of more than a thousand real networks analyzed in the paper. These represent citations between the papers published in the journal Physical Review E [36], paper collaborations between Slovenian researchers extracted from the SICRIS database [37], protein interactions of different species collected from the BioGRID repository [38, 39], interactions between the users at the stack exchange web site MathOverflow [40, 41], Facebook friendships between the students at different US universities [42, 43] and links between autonomous systems extracted by the Oregon Route Views project [41, 44]. Some collections represent temporal networks that grow through time (*e.g.*, paper citations and author collaborations), while other represent similar networks of different size (*e.g.*, protein interactions and online friendships). All networks were reduced to a simple graph of their largest connected component.[2]

Table 1. Statistics of collections of real networks.

Collection	**Networks** N	**Nodes** n	**Edges** m	**Degree** $\langle k \rangle$	**Distance** $\langle d \rangle$	**Clustering** $\langle C \rangle$
Paper citations	46	[3, 37 511]	[2, 135 260]	[1.3, 7.2]	[1.33, 21.79]	[0.00, 0.27]
Author collaborations	25	[18, 1 735]	[42, 6 710]	[4.1, 7.7]	[1.85, 8.75]	[0.46, 0.75]
Protein interactions	40	[5, 19 961]	[4, 238 886]	[1.6, 83.1]	[1.47, 6.06]	[0.00, 0.52]
User interactions	75	[2, 20 969]	[1, 86 137]	[1.0, 10.1]	[1.00, 3.80]	[0.00, 0.17]
Online friendships	97	[762, 41 536]	[16 651, 1 590 651]	[39.1, 116.2]	[2.24, 3.21]	[0.19, 0.41]
Autonomous systems	733	[103, 6 474]	[239, 12 572]	[3.4, 4.7]	[2.65, 3.98]	[0.16, 0.29]

The statistics include the number of networks in the collection N, the number of nodes n and edges m, the average node degree $\langle k \rangle$, the average distance between the nodes $\langle d \rangle$ and the average node clustering coefficient $\langle C \rangle$.

Fig 4 shows the average distance between the nodes $\langle d \rangle$ in real networks and their spanning trees, where we have used semi-logarithmic axes as in Fig 2. First, we consider the networks. As expected for small-world networks [22], the average distance $\langle d \rangle$ increases with the number of nodes n and appears to scale no faster than $\mathcal{O}(\log n)$ in all network collections but two. In the case of temporal networks representing paper citations and author collaborations in the first two plots of Fig 4, the average distance $\langle d \rangle$ actually starts to decrease when the number of nodes exceeds $n \approx 500$. This is a consequence of network densification known as a shrinking diameter [44]. Fig 5 shows also the diameter $d_{\max}$ of real networks, where the interpretation is exactly the same.

Next, we consider the spanning trees of these networks computed with different algorithms. Consistent with the results for synthetic graphs, the spanning trees computed with the breadth-first search algorithm best preserve the average distance between the nodes $\langle d \rangle$ in all network collections but two. In the case of networks representing user interactions and autonomous systems in the bottom row of Fig 4, Kruskal's algorithm performs similarly well. Furthermore, in non-temporal networks that are not subject to the densification law [44], the average distance $\langle d \rangle$ of the spanning trees computed with the breadth-first search algorithm appears to scale no faster than $\mathcal{O}(\log n)$, while in other networks, the scaling of the average distance $\langle d \rangle$ closely follows the scaling in real networks. Again, Fig 5 shows also the diameter $d_{\max}$ of spanning trees, where the interpretation is exactly the same.

[2]Real networks analyzed in the paper are available as Pajek files at `https://doi.org/10.5281/zenodo.15034997`.

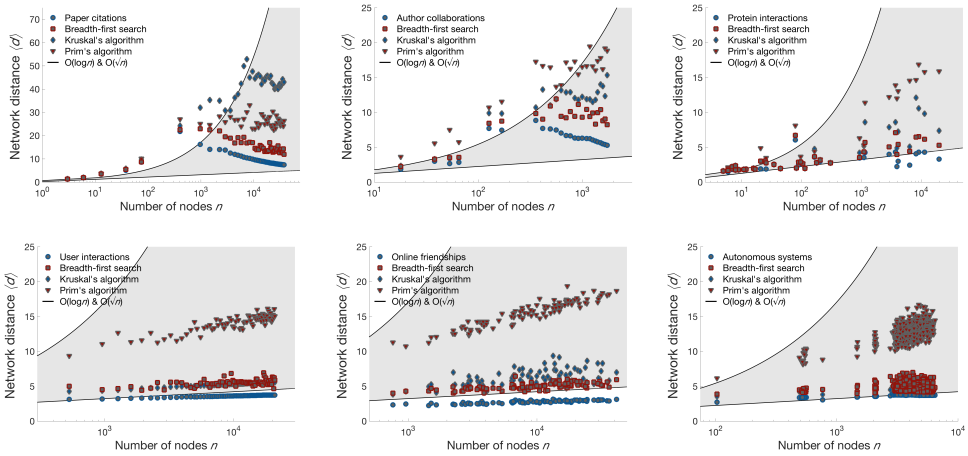

Fig 4. The average distance $\langle d \rangle$ in real networks and their spanning trees computed with different algorithms. The shaded areas are the same as in Fig 2.

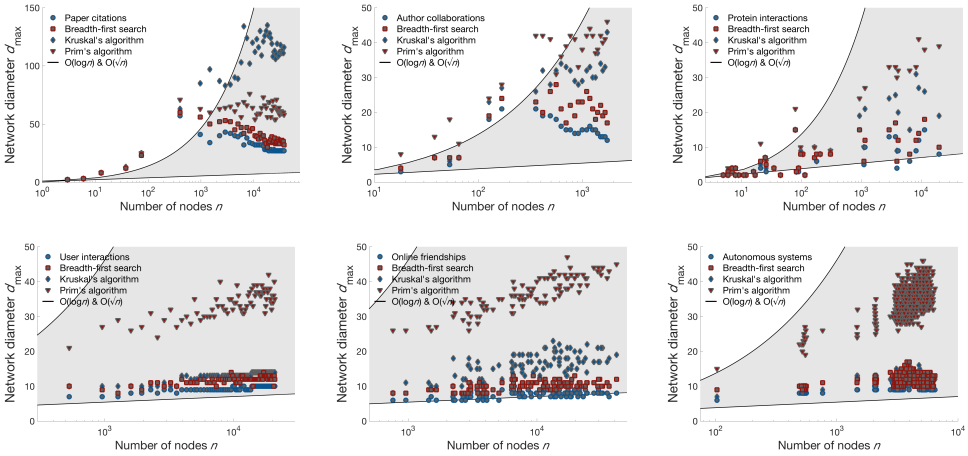

Fig 5. The diameter $d_{\max}$ of real networks and their spanning trees computed with different algorithms. The shaded areas are the same as in Fig 2.

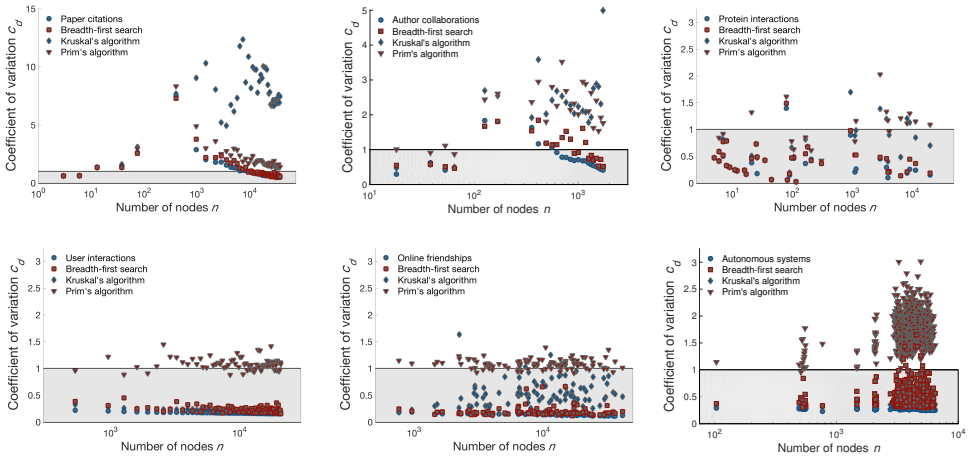

Fig 6. The coefficient of variation c_d for real networks and their spanning trees computed with different algorithms. Other details are the same as in Fig 3.

The above observations are confirmed in Fig 6, where we show the coefficient of variation of the distances between the nodes c_d. Notice that all distributions of the distances in real networks and spanning trees computed with the breadth-first search algorithm are low-variance with $c_d < 1$, as long as the networks are large enough $n \geq 10^4$. In contrast, this holds neither for Kruskal's algorithm nor Prim's algorithm where even $c_d \gg 1$ for some network collections (see first and last plot of Fig 6).

A well-balanced spanning tree would already imply short distances between the nodes $\langle d \rangle \sim \log n$ and also the diameter $d_{\max} \sim \log n$. One of the most commonly used measures of tree balance in the phylogenetics literature is Sackin's index of tree imbalance [24]. Fig 7 shows normalized Sackin's index S [30] of spanning trees computed with different algorithms. Notice that in all but a few cases, Sackin's index S of spanning trees computed with the breadth-first search algorithm is strictly smaller than that of spanning trees computed with any other algorithm, often by an order of magnitude. Therefore, the breadth-first search algorithm computes the most balanced spanning trees. Another widely used measure of tree compactness or density [45] from the chemical graph theory literature is the so-called Wiener's index [23]. Weiner's index is defined as the unnormalized distance between all pairs of nodes $\binom{n}{2}\langle d \rangle$ and thus can be readily observed in Fig 4.

It is further interesting that the node degree distribution p_k of the spanning trees computed with the breadth-first search algorithm often follows a power-law $p_k \sim k^{-\gamma}$ [46], regardless of whether the network is scale-free or not [34, 47]. Fig 8 shows the node degree distribution p_k of the largest networks in Table 1 and their spanning trees. Under the goodness-of-fit test at p-value $= 0.1$ [46], a power-law $p_k \sim k^{-\gamma}$ is a plausible fit of the degree distribution p_k for the protein interactions and autonomous systems networks in the right column of Fig 8. On the other hand, the degree distribution p_k of the spanning trees follows a power-law in all cases but the online friendships network, while the maximum likelihood estimates of the power-law exponents γ are shown with solid lines in Fig 8. Considering the impact of a power-law degree distribution on network structure and dynamics, this property may be useful in practical applications of spanning trees.

Finally, Table 2 shows the runtime and memory consumption of an efficient *Java* implementation of the breadth-first search algorithm and Prim's algorithm applied to the largest networks in Table 1. The breadth-first search algorithm is consistently faster

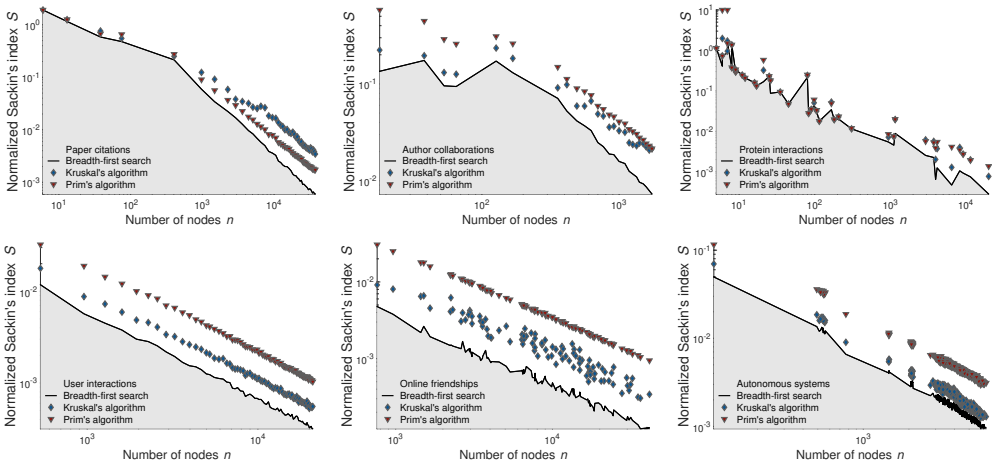

Fig 7. Normalized Sackin's index S of spanning trees computed with different algorithms. The plots show estimates over 25 realizations.

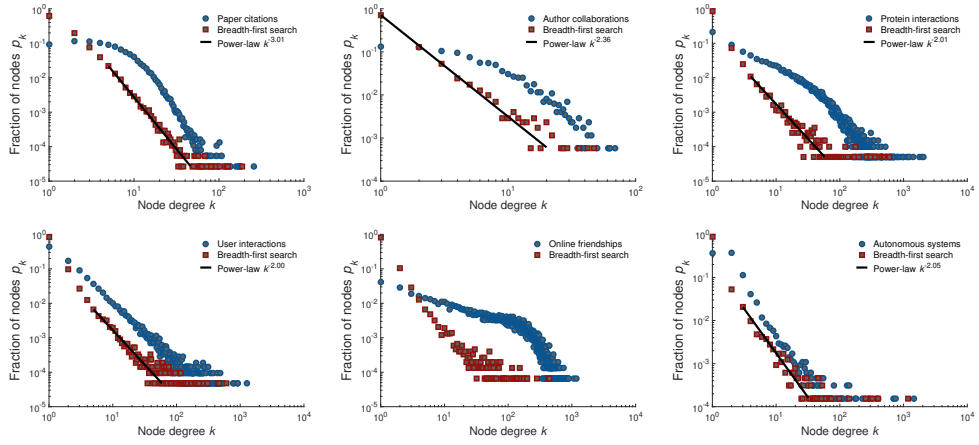

Fig 8. Node degree distribution p_k of real networks and their spanning trees computed with the breadth-first search algorithm. The power-law distributions are maximum likelihood estimates at p-value $= 0.1$ [46].

than Prim's algorithm and consumes much less memory (up to 20 times). On the other hand, Kruskal's algorithm is considerably less efficient and is therefore not included in the analysis.

Table 2. The runtime and memory consumption of algorithms for computing spanning trees of real networks.

	Wall time [ms]			**Memory** [MB]		
Network	**Breadth-first srch.**	**Prim's alg.**	**Speed-up**	**Breadth-first srch.**	**Prim's alg.**	**Gain**
Paper citations	13.33 ± 0.39	62.58 ± 0.80	$4.7\times$	14.88 ± 0.07	53.89 ± 0.07	$3.6\times$
Author collaborations	0.97 ± 0.08	2.55 ± 0.13	$2.6\times$	5.03 ± 0.00	10.05 ± 0.00	$2.0\times$
Protein interactions	8.22 ± 0.41	108.69 ± 1.16	$13.2\times$	10.23 ± 0.06	195.33 ± 0.20	$19.1\times$
User interactions	6.03 ± 0.27	78.61 ± 1.03	$13.0\times$	9.91 ± 0.03	145.95 ± 1.72	$14.7\times$
Online friendships	31.71 ± 0.67	340.36 ± 0.98	$10.7\times$	27.90 ± 0.17	386.37 ± 0.54	$13.8\times$
Autonomous systems	1.37 ± 0.13	31.16 ± 0.77	$22.7\times$	3.52 ± 0.02	68.52 ± 0.85	$19.5\times$

The values are estimates with standard errors over 100 realizations.

To summarize, we again conclude that if a spanning tree should retain the average distance between the nodes $\langle d \rangle$ and the diameter $d_{\max}$ of a real network, then the breadth-first search algorithm should be used. Whether preserving the distances is actually desired or favorable depends on a specific application, which we consider in the following section.

Applications

A spanning tree can be viewed as a technique for discovering a network backbone or skeleton [10, 11], with applications in network visualization and link prediction. Moreover, a spanning tree is one of the basic approaches for network simplification or sampling [6, 8]. Any computation that can be well approximated from a spanning tree of a network, without the need to apply an algorithm to the entire network, can provide computational benefits. In particular, many network applications require superlinear $\mathcal{O}(m \log n)$ or even quadratic algorithms $\mathcal{O}(mn)$, where n and m are the number of nodes and edges. These include community detection algorithms [48] and techniques for computing node importance or similarity [1]. In contrast, the computation of a spanning tree using the breadth-first search algorithm has a linear complexity $\mathcal{O}(m)$ and therefore does not contribute to the overall time complexity.

In this section, we consider two applications of spanning trees, where it is desired to preserve the distances between the nodes of a network.

Node importance

Computing the importance of nodes in a network is a classical application of network science with various use cases. There exist many node measures or indices [49], which are known as measures of node position or centrality in the social network analysis literature [50, 51]. The measure based on the distances between the nodes is called closeness centrality, which estimates the extent to which the node appears to be at the "center" of a network. This measure is often used in operations research and logistics. The closeness centrality of a node $i \in V$ in graph $G(V, E)$ can be defined as $\frac{1}{n-1} \sum_{j \neq i} d_{ij}^{-1}$ [51], where $n = |V|$ is the number of nodes and d_{ij} is the distance between the nodes $i, j \in V$. The time complexity of computing closeness centrality of all nodes in a network is $\mathcal{O}(nm)$ and no more efficient algorithm exists [27].

Fig 9 shows the Pearson correlation coefficient between node closeness centrality in real networks and their spanning trees computed with the breadth-first search, Kruskal's and Prim's algorithms. The coefficients for the breadth-first search algorithm

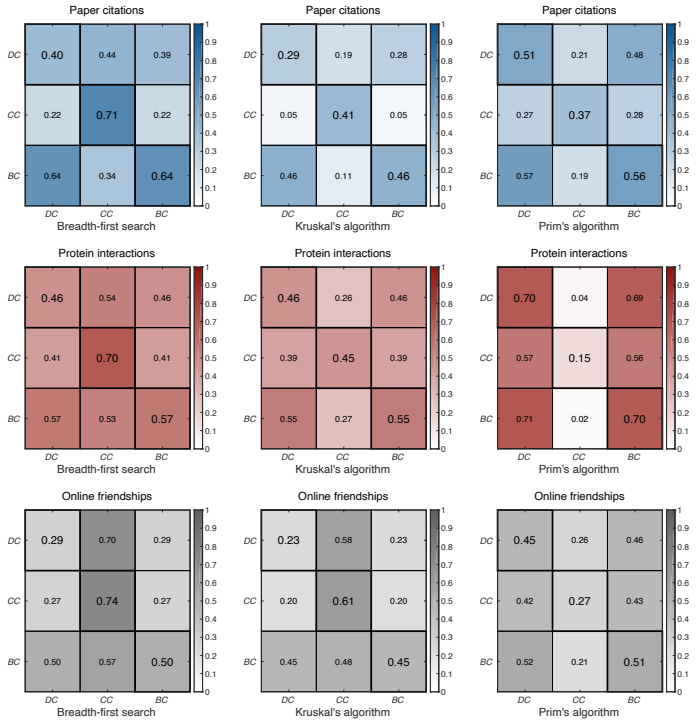

Fig 9. Pearson correlation coefficient between node centrality in real networks and their spanning trees computed with the breadth-first search (left), Kruskal's (middle) and Prim's (right) algorithms. The measures include node degree centrality DC, closeness centrality CC and betweenness centrality BC, where the values are estimates over 25 realizations.

are shown in the center of the heatmaps in the left column of Fig 9. These correlations are all ≥ 0.70, which indicates a strong linear correlation. On the other hand, the correlation coefficients for the spanning trees computed using Kruskal's algorithm in the middle column and Prim's algorithm in the right column are on average 0.49 and 0.26, respectively. Thus, in agreement with the previous results, the breadth-first search algorithm best preserves the distances between the nodes of real networks, not only on average but also at the level of individual nodes.

Merely for comparison, Fig 9 also shows the Pearson correlation coefficient for node degree centrality and betweenness centrality [50]. The latter measures the extent to which a node appears to serve as a "bridge" in a network and is defined as the proportion of all shortest paths between pairs of nodes that pass through a node. The time complexity of computing the betweenness centrality of all nodes in a network is again $\mathcal{O}(nm)$ [52].

Network visualization

In addition to the computational benefits, a network simplification technique such as a spanning tree can also be useful for network visualization. Any visualization using a wiring diagram or other approach is limited in the size of a network it can represent and in the structural properties of a network it can reveal [53, 54]. Classical algorithms for computing a layout of a network include force-directed algorithms [55, 56] and algorithms that embed the nodes in a plane so that their Euclidean distance matches

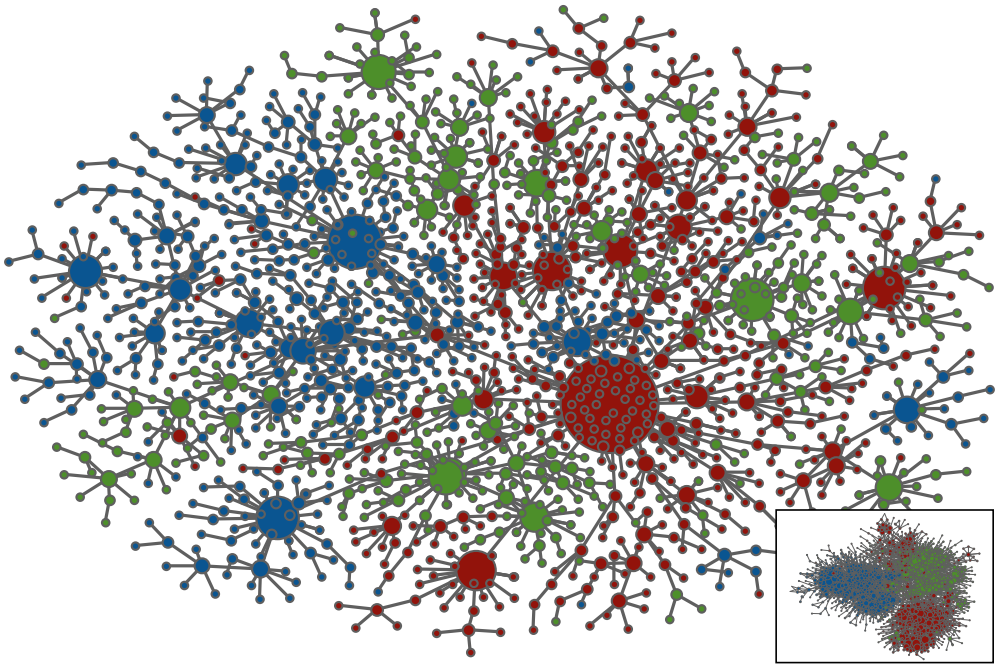

Fig 10. Spanning tree of the SICRIS author collaboration network computed with the breadth-first search algorithm. The size of the nodes is proportional to their degree, while the colors represent primary author disciplines: natural sciences (red), engineering (green), medical sciences (blue) and others (gray). The inset shows a wiring diagram of the entire network. The layouts were computed with the Large Graph Layout algorithm [25].

their network distance as closely as possible [57]. It is, therefore, important that a spanning tree preserves the distances between the nodes of a network if it is to be used with such visualization algorithms.

Fig 10 shows a wiring diagram of a spanning tree of the largest connected component of the SICRIS author collaboration network [37]. The spanning tree was computed with the breadth-first search algorithm. The wiring diagram illustrates how authors from the same discipline cluster in specific regions and how authors from different disciplines collaborate. In contrast, the visualization of the entire network is much more involved, only revealing three clusters (see inset of Fig 10), while providing very limited insight into the patterns of author collaboration [58]. Since there exists no generally accepted quantitative measure for the quality of network visualization, we refrain from further subjective interpretation.

Conclusion

A spanning tree is one of the most straightforward ways to network simplification or sampling and to reveal its backbone or skeleton [8, 10]. Well-known algorithms for computing a spanning tree of a weighted network are Prim's and Kruskal's algorithms [1, 13]. However, when applied to unweighted networks, which are much more common in practice, these algorithms do not capture the structural properties of real networks, such as short distances between the nodes and a small network diameter.

On the other hand, an algorithm based on the breadth-first search node traversal well retains the distances between the nodes in synthetic graphs and real networks. The spanning trees are also well-balanced and most compact. As we demonstrate, this can provide computational benefits of up to 20 times gain and is important for practical applications such as network visualization. Thus, if a spanning tree of an unweighted network is supposed to retain its structure, then the breadth-first search algorithm should be preferred to other algorithms.

We emphasize that the breadth-first search algorithm does not provide the correct result in weighted networks, where Prim's algorithm or Kruskal's algorithm should be used. Furthermore, a spanning tree is not the most suitable technique for simplifying a network in all applications. For instance, a convex skeleton [11] is a natural generalization of a spanning tree that also preserves local density, resulting in a high clustering coefficient and an emphasized community structure. Moreover, if one is interested in predicting the behavior of dynamical processes, then a high-salience skeleton [9] might be preferred.

Acknowledgments

The authors thank Luka Kronegger for sharing the SICRIS data. This work has been supported by the Slovenian Research Agency ARRS under the program P5-0168.

References

1. Newman MEJ. Networks. 2nd ed. Oxford: Oxford University Press; 2018.
2. Tizzoni M, Bajardi P, Poletto C, Ramasco JJ, Balcan D, Gonçalves B, et al. Real-time numerical forecast of global epidemic spreading: Case study of 2009 A/H1N1pdm. BMC Medicine. 2012;10(1):165.
3. Zitnik M, Sosič R, Feldman MW, Leskovec J. Evolution of resilience in protein interactomes across the tree of life. Proceedings of the National Academy of Sciences of the United States of America. 2019;116(10):4426–4433.
4. Fortunato S, Bergstrom CT, Börner K, Evans JA, Helbing D, Milojević S, et al. Science of science. Science. 2018;359(6379):eaao0185.
5. Backstrom L, Boldi P, Rosa M, Ugander J, Vigna S. Four degrees of separation. In: Proceedings of the ACM International Conference on Web Science. Evanston, IL, USA; 2012. p. 45–54.
6. Leskovec J, Faloutsos C. Sampling from large graphs. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Philadelphia, PA, USA; 2006. p. 631–636.
7. Hamann M, Lindner G, Meyerhenke H, Staudt CL, Wagner D. Structure-preserving sparsification methods for social networks. Social Network Analysis and Mining. 2016;6(1):22.
8. Blagus N, Šubelj L, Bajec M. Empirical comparison of network sampling: How to choose the most appropriate method? Physica A: Statistical Mechanics and its Applications. 2017;477:136–148.
9. Grady D, Thiemann C, Brockmann D. Robust classification of salient links in complex networks. Nature Communications. 2012;3:864.

10. Coscia M, Neffke F. Network backboning with noisy data. In: Proceedings of the IEEE International Conference on Data Engineering. San Diego, CA, USA; 2017. p. 425–436.

11. Šubelj L. Convex skeletons of complex networks. Journal of the Royal Society Interface. 2018;15(145):20180422.

12. Simas T, Correia RB, Rocha LM. The distance backbone of complex networks. Journal of Complex Networks. 2021;9(6):cnab021.

13. Bollobás B. Modern Graph Theory. Heidelberg: Springer; 1998.

14. Pop PC. The generalized minimum spanning tree problem: An overview of formulations, solution procedures and latest advances. European Journal of Operational Research. 2020;283(1):1–15.

15. Diggans CT, Bollt EM, Ben-Avraham D. Spanning trees of recursive scale-free graphs. Physical Review E. 2022;105:024312.

16. Henke L, Mehta D. C-RST: A parallel algorithm for random spanning trees in network analytics. Applied Network Science. 2024;9:45.

17. Dhouib S. Innovative method to solve the minimum spanning tree problem: The Dhouib-Matrix-MSTP (DM-MSTP). Results in Control and Optimization. 2024;14:100359.

18. Rezvanian A, Vahidipour SM, Jalali ZS. A spanning tree approach to social network sampling with degree constraints. Social Network Analysis and Mining. 2024;14:101.

19. Amure R, Agarwal NA. A comparative evaluation of social network analysis tools: Performance and community engagement perspectives. Social Network Analysis and Mining. 2025;15:100359.

20. Abdallah S. Generalizing unweighted network measures to capture the focus in interactions. Social Network Analysis and Mining. 2011;1:255–269.

21. Bourgain J. On Lipschitz embedding of finite metric spaces in Hilbert space. Israel Journal of Mathematics. 1985;52(1-2):46–52.

22. Watts DJ, Strogatz SH. Collective dynamics of 'small-world' networks. Nature. 1998;393(6684):440–442.

23. Wiener H. Structural determination of paraffin boiling points. Journal of the American Chemical Society. 1947;69(1):17–20.

24. Sackin MJ. "Good" and "bad" phenograms. Systematic Biology. 1972;21(2):225–226.

25. Adai AT, Date SV, Wieland S, Marcotte EM. LGL: Creating a map of protein function with an algorithm for visualizing very large biological networks. Journal of Molecular Biology. 2004;340(1):179–190.

26. Barabási AL. Network Science. Cambridge: Cambridge University Press; 2016.

27. Knuth DE. The Art of Computer Programming. Amsterdam: Addison-Wesley Professional; 2011.

28. Lemant J, Le Sueur C, Manojlović V, Noble R. Robust, universal tree balance indices. Systematic Biology. 2022;71(5):1210–1224.

29. Fischer M, Herbst L, Kersting SA, Kühn L, Wicke K. Tree balance indices: A comprehensive survey. 1st ed. Cham: Springer; 2023.

30. Shao KT, Sokal RR. Tree balance. Systematic Zoology. 1990;39(3):266–276.

31. Erdős P, Rényi A. On random graphs I. Publicationes Mathematicae Debrecen. 1959;6:290–297.

32. Rényi A, Szekeres G. On the height of trees. Journal of the Australian Mathematical Society. 1967;7(4):497–507.

33. Meir A, Moon JW. The distance between points in random trees. Journal of Combinatorial Theory. 1970;8(1):99–103.

34. Barabási AL, Albert R. Emergence of scaling in random networks. Science. 1999;286(5439):509–512.

35. Cohen R, Havlin S. Scale-free networks are ultrasmall. Physical Review Letters. 2003;90(5):058701.

36. American Physical Society. APS Dataset; 2015. `http://journals.aps.org/datasets`.

37. Institute of Information Science. SICRIS Database; 2010. `http://www.sicris.si`.

38. Stark C, Breitkreutz BJ, Reguly T, Boucher L, Breitkreutz A, Tyers M. BioGRID: A general repository for interaction datasets. Nucleic Acids Research. 2006;34(1):535–539.

39. Biological General Repository for Interaction Datasets. BioGRID Database; 2016. `http://thebiogrid.org`.

40. Paranjape A, Benson AR, Leskovec J. Motifs in temporal networks. In: Proceedings of the ACM International Conference on Web Search and Data Mining. Cambridge, UK; 2017. p. 601–610.

41. Leskovec J, Krevl A. SNAP Datasets; 2014. `http://snap.stanford.edu/data`.

42. Traud AL, Mucha PJ, Porter MA. Social structure of Facebook networks. Physica A: Statistical Mechanics and its Applications. 2012;391(16):4165–4180.

43. Rossi RA, Ahmed NK. Network Data Repository; 2015. `http://networkrepository.com`.

44. Leskovec J, Kleinberg J, Faloutsos C. Graphs over time: Densification laws, shrinking diameters and possible explanations. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Chicago, IL, USA; 2005. p. 177–187.

45. Ozen M, Lesaja G, Wang H. Globally optimal dense and sparse spanning trees, and their applications. Statistics, Optimization & Information Computing. 2020;8(2):328–345.

46. Clauset A, Shalizi CR, Newman MEJ. Power-law distributions in empirical data. SIAM Review. 2009;51(4):661–703.

47. Broido AD, Clauset A. Scale-free networks are rare. Nature Communications. 2019;10(1):1017.

48. Fortunato S, Hric D. Community detection in networks: A user guide. Physics Reports. 2016;659:1–44.

49. Schoch D, Brandes U. Re-conceptualizing centrality in social networks. European Journal of Applied Mathematics. 2016;27(6):971–985.

50. Freeman L. A set of measures of centrality based on betweenness. Sociometry. 1977;40(1):35–41.

51. Freeman LC. Centrality in social networks: Conceptual clarification. Social Networks. 1979;1(3):215–239.

52. Brandes U. A faster algorithm for betweenness centrality. Journal of Mathematical Sociology. 2001;25(2):163–177.

53. Ma KL, Muelder CW. Large-scale graph visualization and analytics. Computer. 2013;46(7):39–46.

54. Gibson H, Faith J, Vickers P. A survey of two-dimensional graph layout techniques for information visualisation. Information Visualization. 2013;12(3-4):324–357.

55. Eades P. A heuristic for graph drawing. Congressus Numerantium. 1984;42:149–160.

56. Fruchterman TMJ, Reingold EM. Graph drawing by force-directed placement. Software: Practice and Experience. 1991;21(11):1129–1164.

57. Kamada T, Kawai S. An algorithm for drawing general undirected graphs. Information Processing Letters. 1989;31(1):7–15.

58. Šubelj L, Fiala D, Ciglarič T, Kronegger L. Convexity in scientific collaboration networks. Journal of Informetrics. 2019;13(1):10–31.