Computing dense spanning trees of unweighted networks

Lovro Šubelj*

University of Ljubljana, Faculty of Computer and Information Science, Večna pot 113, Ljubljana, Slovenia

Abstract

Spanning tree of a network or a graph is a subgraph connecting all the nodes with the minimum number of edges. Spanning tree retains the connectivity of a network and possibly other structural properties, and is one of the simplest techniques for network simplification or sampling, and for revealing its backbone or skeleton. The Prim's algorithm and the Kruskal's algorithm are well known algorithms for computing a spanning tree of a weighted network. In this paper, we empirically study the performance of these algorithms on unweighted networks, and compare them to different priority-first search algorithms. We show that the distances between the nodes and the diameter of a network are best preserved by an algorithm based on the breadth-first search node traversal. We support our results by experiments on synthetic graphs and more than a thousand real networks, and demonstrate different practical applications of computed spanning trees. We conclude that, if a spanning tree is supposed to retain the distances between the nodes or the diameter of an unweighted network, also known as a dense spanning tree in the literature, then the breadth-first search algorithm should be the preferred choice.

Keywords: Complex networks, Dense spanning tree, Breadth-first search, Network distances, Network diameter

1. Introduction

Networks or graphs have become a popular tool for analysing complex real-world systems [1]. Examples include predicting the spread of contagious viruses [2], study of the interactome of species [3], understanding the structure of science [4] and outreach of social connections online [5]. The sizes of today's networks are often in millions of nodes and edges, with the largest networks being the WWW with more than a trillion web pages and human brain with close to a hundred of

*Corresponding author.
Email address: lovro.subelj@fri.uni-lj.si (Lovro Šubelj)

billions of neurons. In result, the size of real networks can make many practical applications computationally very demanding.

Techniques to alleviate this issue include network simplification or sampling [6, 7, 8], and revealing the so-called network backbone or skeleton [9, 10, 11]. 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 [12, 1], which is a subgraph connecting all the nodes of a network with the minimum number of edges. A spanning tree retains the connectivity of a network and possibly other structural properties. 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 minimum overall weight of the edges. In the case of unweighted networks, any spanning tree is in fact a "minimum" spanning tree.

The Prim's algorithm and the Kruskal's algorithm are well known algorithms for computing a minimum spanning tree of a weighted network [12, 1]. Although developed primarily for weighted networks, the algorithms can be readily applied to unweighted networks. However, the performance of these algorithms has not yet been properly explored for unweighted networks. In particular, there exist no theoretical guarantees of the algorithms, neither does the literature provide any empirical comparison on large-scale networks. In this paper, we apply the algorithms to 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. In particular, spanning trees computed with the breadth-first search algorithm well retain the distances between the nodes of the network[1], which may be desirable in practical applications such as network visualization.

In the chemical graph theory literature, spanning trees that minimize the distances between the nodes, as measured by the Wiener index [13], are called dense spanning trees [14, 15]. However, this literature is mainly concerned with small idealized graphs and provides no answers to how different algorithms perform on large-scale networks.

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. Next, we analyze the structure of spanning trees of synthetic graphs and, afterwards, the structure of spanning trees of real networks. Finally, we demonstrate practical applications of spanning trees of networks. We conclude the paper with suggestions for future research.

2. Computation of spanning trees

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.

[1] The breadth-first search algorithm is also a standard approach for computing the distances between one selected node and all other nodes in an undirected network.

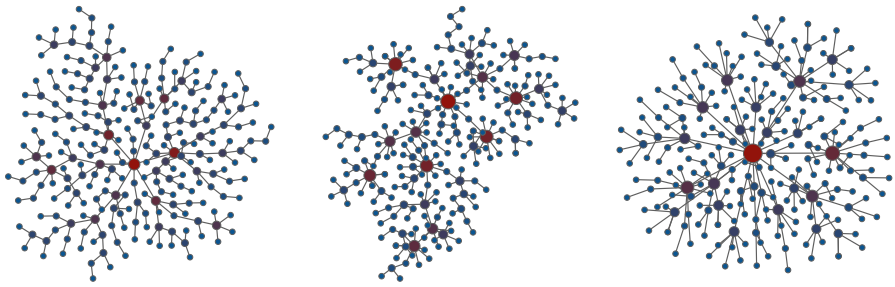

Figure 1: Wiring diagrams of spanning trees of a small random graph. The spanning trees were computed with the Prim's algorithm (left), the Kruskal's algorithm (middle) and the breadth-first search algorithm (right). The sizes of the nodes are proportional to their degrees, while the layouts were computed with the Large Graph Layout algorithm [16].

(Where necessary to make explicit that these sets represent 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 the shortest paths 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 $d_{\max} = \max_{i<j} d_{ij}$. 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 the distances.

Below we precisely 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 network analysis libraries such as *NetworkX*[2]. For a more extensive discussion on the differences between the algorithms, we refer the reader to classical graph theory literature [12] or network science literature [17, 1].

2.1. Prim's algorithm

The 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 staring 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

[2] `https://networkx.org`

Algorithm 1 Prim's algorithm

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

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).

The Prim's algorithm is non-deterministic and can compute different spanning trees. The actual spanning tree depends on random selection of the seed node (line 2 of Algorithm 1) and on random selection of the edges to expand the tree (line 5). The latter is nontrivial and 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 the Prim's algorithm implemented with a Fibonacci heap is $\mathcal{O}(m + n \log n)$. For unweighted graphs which we consider here, the heap can be replaced by a simple array list, which reduces the time complexity to $\mathcal{O}(m)$. As an example, the left graph in Figure 1 shows a spanning tree computed with the Prim's algorithm.

2.2. Kruskal's algorithm

The Kruskal's algorithm is conceptually different from the 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 tree remains. At this point the algorithm stops and the remaining tree is a spanning tree of a graph.

The Kruskal's algorithm is non-deterministic and the actual spanning tree depends on random selection of the edges to merge the trees on 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 Figure 1 shows a spanning tree computed with the Kruskal's algorithm.

2.3. Breadth-first search

The breadth-first search node traversal is very similar to the Prim's algorithm. The main difference is in how the edges to non-visited nodes are pro-

Algorithm 2 Breadth-first search

Require: undirected graph G
Ensure: spanning tree T
1: $T \leftarrow$ empty graph
2: $Q \leftarrow$ empty queue
3: $i \leftarrow \textsc{Random}(i \in V_G)$ ▷ Select 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

cessed. In contrast to the 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 $Q \subseteq V_T$ for further processing. 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 further 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 non-deterministic, while the actual spanning tree depends on random selection of the seed node (line 3 of Algorithm 2) 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 Figure 1 shows a spanning tree computed with the breadth-first search algorithm.

2.4. Other algorithms

Other algorithms for computing a spanning tree include the Sollin's algorithm, which can be seen as a combination of the 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 replace the queue of

non-processed nodes Q with a stack (line 2 in Algorithm 2). This is because the queue Q operates on a first-in first-out basis, while a stack operates on a last-in first-out basis (lines 6, 8). The time complexity of the depth-first search algorithm is again $\mathcal{O}(m)$.

3. Spanning trees of synthetic graphs

Consider an Erdős-Rényi random graph [18] 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 with the average node degree $\langle k \rangle = 2 - 2/n$. Since it is a tree, the average clustering coefficient equals $\langle C \rangle = 0$ [19]. We, therefore, here focus on other graph properties. 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 estimates for the considered random graph are $\langle d \rangle \approx 2.64$ and $d_{\max} \approx 4.39$.

Figure 1 shows particular realizations of spanning trees of a random graph with the above parameters computed with the Prim's algorithm, the Kruskal's algorithm and the breadth-first search algorithm. The diameter $d_{\max}$ of the spanning trees equals 14, 17 and 6, respectively. While the diameters of the spanning trees computed with Prim's algorithm and the Kruskal's algorithm are much higher than in the random graph, the diameter of the spanning tree computed with the breadth-first search algorithm is very close to the diameter of the random graph. These observations are closely related to the question whether the computed spanning trees are balanced [20].

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 [20]. However, in the case of a random tree, both values are almost certainly in $\mathcal{O}(\sqrt{n})$ [21, 22]. 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, and to what extent. To the best of our knowledge, no such approach exists. One can, however, 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 try to 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 also real small-world networks [19].

Besides Erdős-Rényi random graphs [18], we also analyse triangular lattices and Barabási-Albert scale-free graphs [23]. We vary the number of nodes n, while we keep the average node degree fixed to $\langle k \rangle = 10$. Figure 2 shows the scaling of the average distance $\langle d \rangle$ and the diameter $d_{\max}$ for selected synthetic graphs and their spanning trees computed with different algorithms.

We first consider triangular lattices, as these results serve as a baseline for further analyses. The average distance $\langle d \rangle$ and the diameter $d_{\max}$ of any two-

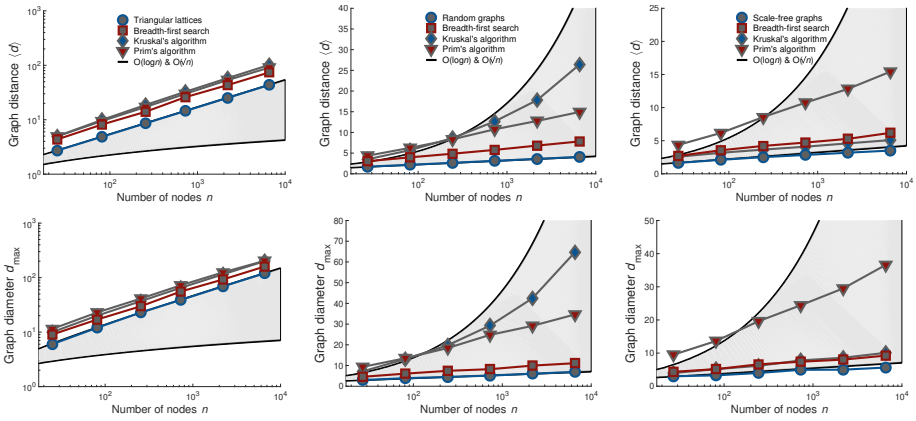

Figure 2: The average distance $\langle d \rangle$ and the diameter $d_{\max}$ of triangular lattices (left), random graphs (middle) and scale-free graphs (right), and their spanning trees computed with different algorithms (legend). The plots show estimates over 100 realizations, while the shaded areas 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.

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 Figure 2. Notice also that all the spanning trees computed with different algorithms show similar scaling $\mathcal{O}(\sqrt{n})$.

Next, we consider Erdős-Rényi random graphs [18] shown in the middle column of Figure 2. The average distance $\langle d \rangle$ and the diameter $d_{\max}$ of random graphs, and also small-world networks [19], scale as $\mathcal{O}(\log n)$ [1]. This can be observed as a straight line on semi-logarithmic plots in Figure 2, whereas any upward concave function would imply scaling faster than $\mathcal{O}(\log n)$. Notice that the spanning trees computed with the breadth-first search algorithm best retain 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 moderate number of nodes $n \leq 10^4$. In contrast, the distances between the nodes of the spanning trees computed with the Kruskal's algorithm scale faster than $\mathcal{O}(\log n)$ (see also Figure 3 and discussion below).

Last, we consider Barabási-Albert scale-free graphs [23] shown in the right column of Figure 2. The average distance $\langle d \rangle$ and the diameter $d_{\max}$ of scale-free graphs scale as $\mathcal{O}(\log n / \log\log n)$ [24], while such graphs are usually called ultra small-world [17]. Note that $\mathcal{O}(\log n / \log\log n)$ is indistinguishable from $\mathcal{O}(\log n)$ for $n \leq 10^4$, thus this scaling can again be observed as a straight line on semi-logarithmic plots in Figure 2. The spanning trees computed with both the breadth-first search algorithm and the 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 the 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 $d_{\max} = 36.6$

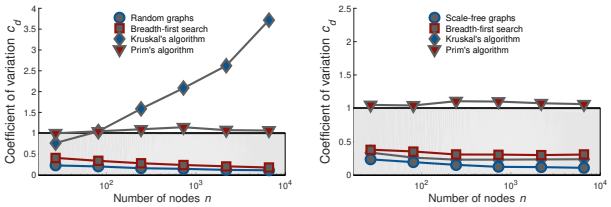

Figure 3: The coefficient of variation c_d for random (left) and scale-free graphs (right), and their spanning trees computed with different algorithms (legend). The plots show estimates over 100 realizations, while the error bars are smaller than the symbol sizes.

compared to 3.51 and 5.64 for graphs with $n = 6\,561$ nodes.

Above observations are confirmed in Figure 3, where we show the coefficient of variation of the distances between the nodes c_d. This is a standard measure of dispersion of a probability distribution, while the distributions with $c_d < 1$ are considered low-variance distributions and those with $c_d > 1$ are high-variance distributions. Note that the distributions of the distances between the nodes of random and scale-free graphs, and real small-world networks, are low-variance with $c_d \ll 1$ [19, 17]. As one can see in Figure 3, all distributions of the distances in the spanning trees computed with the breadth-first search algorithm are low-variance $c_d \ll 1$. The distributions for spanning trees computed with the Kruskal's algorithm are high-variance $c_d \gg 1$ for random graphs with $n > 100$, while the results for the Prim's algorithm are inconclusive $c_d \approx 1$.

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

4. Spanning trees of real networks

Table 1 shows statistics of collections of over a thousand real networks analyzed in the paper. These represent citations between the papers published in the Physical Review E journal [25], paper collaborations between the authors

Table 1: Statistics of collections of real networks. These are the number of networks in the collection N, and the number of nodes n and edges m and the average node degree $\langle k \rangle$.

Networks	N	n	m	$\langle k \rangle$
Paper citations	46	[3, 37 511]	[2, 135 260]	[1.3, 7.2]
Author collaborations	25	[18, 1 735]	[42, 6 710]	[4.1, 7.7]
Protein interactions	40	[5, 19 961]	[4, 238 886]	[1.6, 83.1]
User interactions	75	[2, 20 969]	[1, 86 137]	[1.0, 10.1]
Online friendships	97	[762, 41 536]	[16 651, 1 590 651]	[39.1, 116.2]
Autonomous systems	733	[103, 6 474]	[239, 12 572]	[3.4, 4.7]

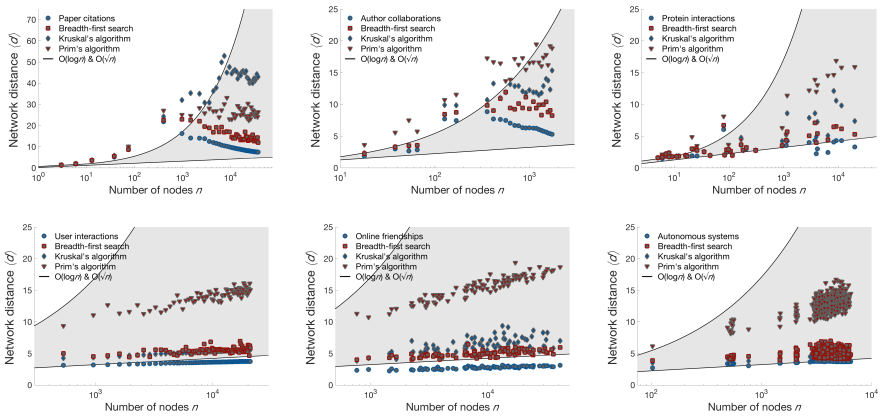

Figure 4: The average distance $\langle d \rangle$ of real networks and their spanning trees computed with different algorithms (legend), while the shaded areas are the same as in Figure 2.

extracted from the SICRIS database [26], protein interactions of different species collected from the BioGRID repository [27, 28], interactions between the users at the stack exchange web site MathOverflow [29, 30], Facebook friendships between the students at different US universities [31, 32] and links between autonomous systems extracted by the Oregon Route Views project [33, 30]. 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.

Figure 4 shows the average distance between the nodes $\langle d \rangle$ of real networks and their spanning trees, where we have used semi-logarithmic axes as in Figure 2. First, we consider the networks. As expected for small-world networks [19], 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 Figure 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 shrinking diameter [33]. Figure A.8 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 Figure 4, the Kruskal's algorithm performs similarly well. Furthermore, in non-temporal networks that are not subject to densification law [33], the average distance $\langle d \rangle$ of the spanning trees computed with the breadth-first search

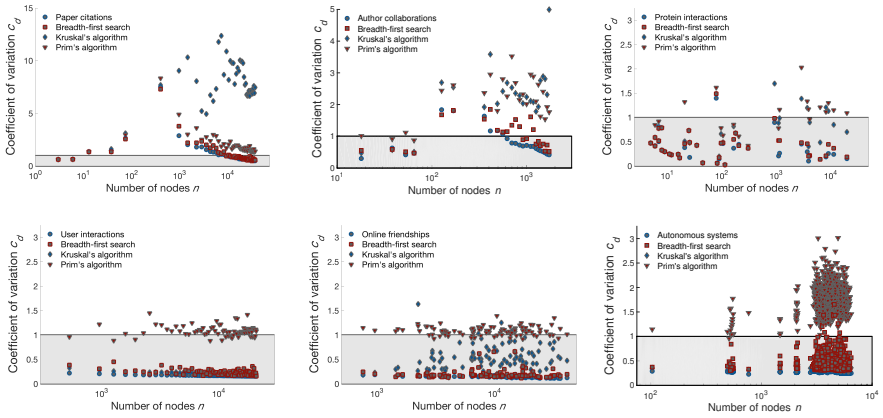

Figure 5: The coefficient of variation c_d for real networks and their spanning trees computed with different algorithms (legend), while other details are the same as in Figure 3.

algorithm, and also other algorithms, appears to scale no faster than $\mathcal{O}(\log n)$. In other networks, the scaling of the average distance $\langle d \rangle$ closely follows the scaling for real networks. Again, Figure A.8 shows also the diameter $d_{\max}$ of spanning trees, where the interpretation is exactly the same.

Above observations are confirmed in Figure 5, 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 the Kruskal's algorithm nor the Prim's algorithm where even $c_d \gg 1$ for some network collections (see first and last plot of Figure 5). 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 $p_k \sim k^{-\gamma}$ [34], regardless of whether the network is scale-free or not [23, 35] (see Figure A.9 and discussion alongside). Considering the impact of power-law degree distributions on network structure and dynamics, this property may be useful in practical applications of spanning trees.

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 network, then the breadth-first search algorithm should be used. Whether preserving the distances is actually desired or favorable depends on a particular application, which we consider in the following section.

5. Applications of spanning trees

A spanning tree can be seen as a technique for revealing a network backbone or skeleton [10, 11], with applications in network visualization and link prediction. Furthermore, a spanning is one of the simplest 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 of applying the algorithms to the entire network, can provide computational benefits. In particular, many applications on networks require super-linear $\mathcal{O}(m \log n)$ or even quadratic algorithms $\mathcal{O}(mn)$, where n and m are the number of nodes and edges. These include community detection algorithms [36] and revealing node importance or similarity [1]. In contrast, the computation of a spanning tree with the breadth-first search algorithm has linear time complexity $\mathcal{O}(m)$ and thus 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.

5.1. Node importance

Revealing the importance of nodes in a network is a classical application of network science with different use cases. There exist many node measures or indices [37], known as measures of node position or centrality in the social networks analysis literature [38, 39]. The measure based on the distances between the nodes is called closeness centrality, measuring the extent to which the node appears to be in the "center" of a network, which is often used in operations research. The closeness centrality of a node $i \in V$ of graph $G(V, E)$ is defined as $\frac{1}{n-1} \sum_{j \neq i} d_{ij}^{-1}$ [39], 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 [20].

Figure 6 shows the Pearson correlation coefficient between node closeness centrality in real networks and their spanning trees computed with the breadth-first search algorithm, the Kruskal's algorithm and the Prim's algorithm. The coefficients for the breadth-first search algorithm are shown in the center of the heatmaps in the left column of Figure 6. These correlations are all ≥ 0.70, which means strong linear correlation. On the other hand, the correlation coefficients for the spanning trees computed with the Kruskal's algorithm in the middle column of Figure 6 and the Prim's algorithm in the right column are ≤ 0.61 and ≤ 0.37, respectively. Therefore, consistent with the previous results, the breadth-first search algorithm best preserves the distances between the nodes of real networks, even on the level of individual nodes.

For comparison, Figure 6 also shows the Pearson correlation coefficient for node degree centrality and betweenness centrality [38]. The latter measures the extent to which a node appears to serve as a "bridge" in a network and is defined as the fraction of the shortest paths between all pairs of nodes that go through a node. The time complexity of computing betweenness centrality of all nodes in a network is again $\mathcal{O}(nm)$ and no more efficient algorithm exists [40].

5.2. Network visualization

Besides providing computational benefits, a network simplification technique such as a spanning tree can be useful in network visualization. Any visualization with a wiring diagram or some other approach is limited in the size of a

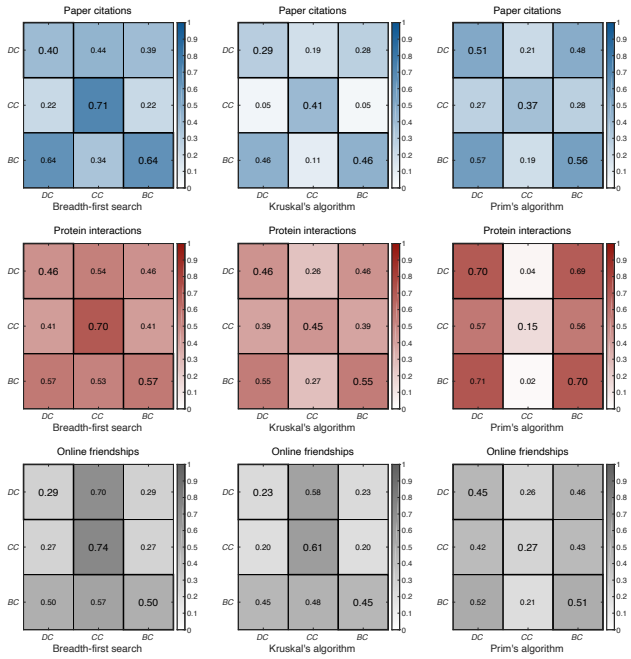

Figure 6: Pearson correlation coefficient between the measures of node centrality in real networks and their spanning trees computed with the breadth-first search algorithm (left), the Kruskal's algorithm (middle) and the Prim's algorithm (right). The measures include node degree centrality DC, closeness centrality CC and betweenness centrality BC [38], while the values are estimates over 25 realizations.

network it can represent and in the structural characteristics of a network it can reveal [41, 42]. Classical algorithms for computing a layout of a network include spring embedding or force-directed algorithms [43, 44] and algorithms that embed the nodes in a plane thus their Euclidean distance matches their network distance as best as possible [45]. It is, therefore, important that a spanning tree preserves the distances between the nodes of a network if it is to be used in network visualization.

Figure 7 shows a wiring diagram of a spanning tree of the largest connected component of the SICRIS author collaboration network [26]. The spanning tree was computed with the breadth-first search algorithm. The wiring diagram illustrates how authors from the same discipline cluster in certain regions and how authors from different disciplines collaborate. In contrast, the visualization of the entire network is much more involved [16], while it provides limited insight into the patterns of author collaboration. Since there exists no objective measure of the quality of a network visualization, we refrain from providing any further subjective interpretation.

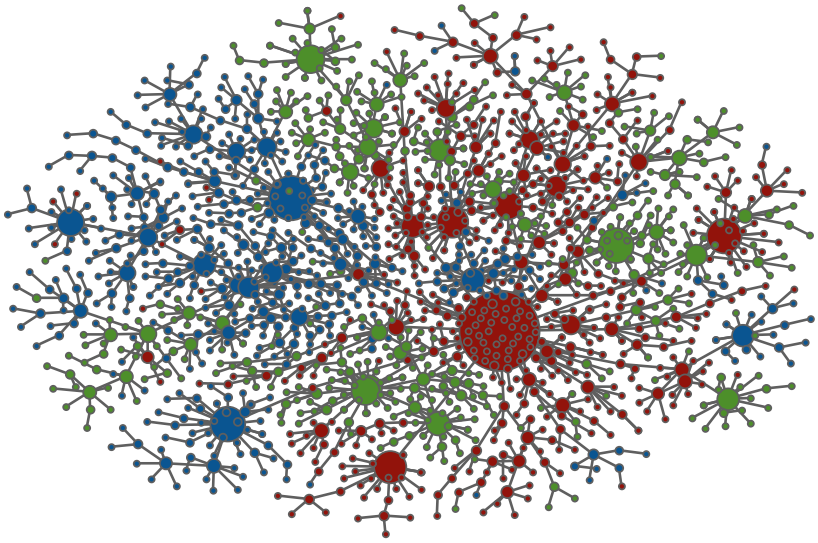

Figure 7: Spanning tree of the SICRIS author collaboration network computed with the breadth-first search algorithm. The sizes of the nodes are proportional to their degrees, while the colors represent primary author disciplines consisting of natural sciences (red), engineering (green), medical sciences (blue) and other. The layout was computed with the Large Graph Layout algorithm [16].

6. Conclusions

A spanning tree is one of the most straightforward ways to network simplification or sampling, and for revealing its backbone or skeleton [8, 10]. Well known algorithms for computing a spanning tree of a weighted network are the Prim's algorithm and the Kruskal's algorithm [12, 1]. However, when applied to unweighted networks, these algorithms do not necessarily capture the structure of a network such as short distances between the nodes and small network diameter [19]. Prior to this work there existed no empirical evaluation or comparison of the algorithms on large-scale networks. We have shown that an algorithm based on the breadth-first search node traversal well retains the distances between the nodes in synthetic graphs and real networks. As we demonstrate, this can provide computational benefits and is also important in practical applications like network visualization. Thus, if a spanning tree of an unweighted network is supposed to retain the distances between the nodes, then the breadth-first search algorithm should be preferred to other algorithms considered here. At the same time, spanning trees that minimize the distances measured by the Wiener index [13] are known as dense spanning trees in chemical graph theory. Our results might also be relevant in the context of such applications.

In the language of the theory of computation, the breadth-first search algorithm computes a spanning tree with properties of a balanced tree [20]. Due to

the lack of a formal definition of some sort of approximate balance that would *already* imply short distances between the nodes, our results here are merely empirical. In future research, we plan to develop such a definition that could support the results also analytically.

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.

Appendix A. Diameter and degree distribution

Figure A.8 shows the diameter $d_{\max}$ of real networks and their spanning trees computed with the Prim's algorithm, the Kruskal's algorithm and the breadth-first search algorithm. The interpretation of these results is the same as for the average distance $\langle d \rangle$ in Figure 4 and therefore omitted here.

Figure A.9 shows the node degree distribution p_k of the largest networks in Table 1 and their spanning trees computed with the breadth-first search algorithm. Under the goodness-of-fit test at p-value $= 0.1$ [34], 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 Figure A.9. 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. The maximum likelihood estimates of the power-law exponents γ are shown with solid lines in Figure A.9.

References

[1] M. E. J. Newman, Networks, 2nd Edition, Oxford University Press, Oxford, 2018.

[2] M. Tizzoni, P. Bajardi, C. Poletto, J. J. Ramasco, D. Balcan, B. Gonçalves, N. Perra, V. Colizza, A. Vespignani, Real-time numerical forecast of global epidemic spreading: case study of 2009 A/H1N1pdm, BMC Medicine 10 (1) (2012) 165.

[3] M. Zitnik, R. Sosič, M. W. Feldman, J. Leskovec, Evolution of resilience in protein interactomes across the tree of life, Proceedings of the National Academy of Sciences of the United States of America 116 (10) (2019) 4426–4433.

[4] S. Fortunato, C. T. Bergstrom, K. Börner, J. A. Evans, D. Helbing, S. Milojević, A. M. Petersen, F. Radicchi, R. Sinatra, B. Uzzi, A. Vespignani, L. Waltman, D. Wang, A.-L. Barabási, Science of science, Science 359 (6379) (2018) eaao0185.

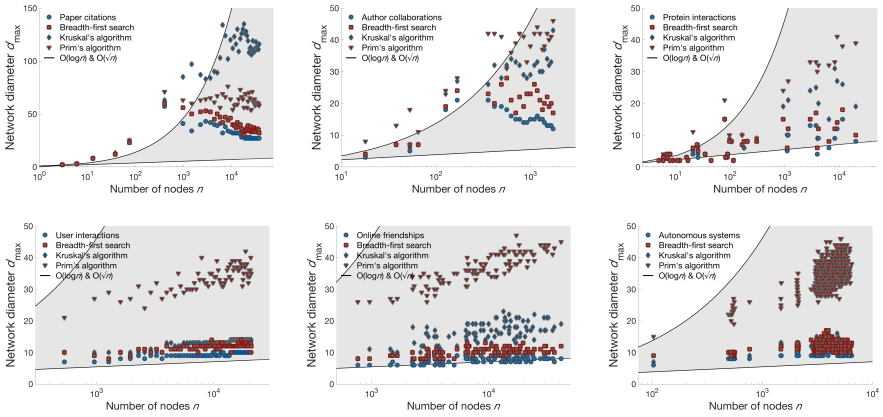

Figure A.8: The diameter $d_{\max}$ of real networks and their spanning trees computed with different algorithms (legend), while the shaded areas are the same as in Figure 4.

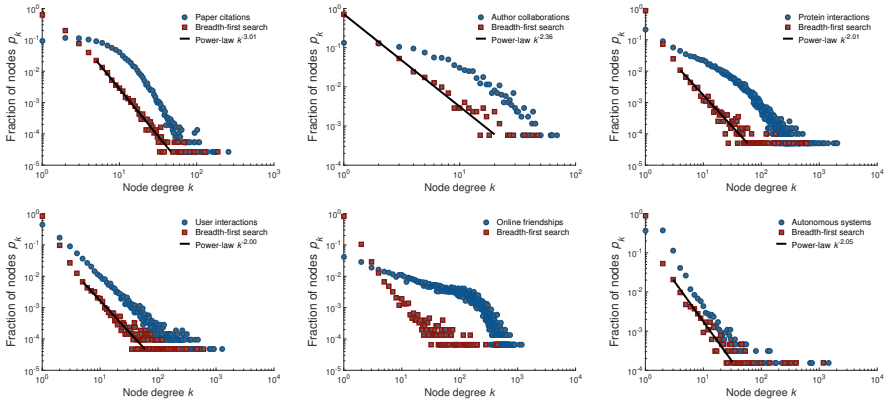

Figure A.9: Node degree distribution p_k of real networks and their spanning trees computed with the breadth-first search algorithm. The power-law distributions are maximum likelihood estimates at p-value $= 0.1$ [34].

[5] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, S. Vigna, Four degrees of separation, in: Proceedings of the ACM International Conference on Web Science, Evanston, IL, USA, 2012, pp. 45–54.

[6] J. Leskovec, C. Faloutsos, Sampling from large graphs, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, 2006, pp. 631–636.

[7] M. Hamann, G. Lindner, H. Meyerhenke, C. L. Staudt, D. Wagner, Structure-preserving sparsification methods for social networks, Social Network Analysis and Mining 6 (1) (2016) 22.

[8] N. Blagus, L. Šubelj, M. Bajec, Empirical comparison of network sampling: How to choose the most appropriate method?, Physica A: Statistical Mechanics and its Applications 477 (2017) 136–148.

[9] D. Grady, C. Thiemann, D. Brockmann, Robust classification of salient links in complex networks, Nature Communications 3 (2012) 864.

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

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

[12] B. Bollobás, Modern Graph Theory, Springer, Heidelberg, 1998.

[13] H. Wiener, Structural determination of paraffin boiling points, Journal of the American Chemical Society 69 (1) (1947) 17–20.

[14] M. Ozen, H. Wang, K. Wnag, D. Yalman, An edge-swap heuristic for finding dense spanning trees, Theory and Applications of Graphs 3 (1) (2016) 1.

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

[16] A. T. Adai, S. V. Date, S. Wieland, E. M. Marcotte, LGL: Creating a map of protein function with an algorithm for visualizing very large biological networks, Journal of Molecular Biology 340 (1) (2004) 179–190.

[17] A.-L. Barabási, Network Science, Cambridge University Press, Cambridge, 2016.

[18] P. Erdős, A. Rényi, On random graphs I, Publicationes Mathematicae Debrecen 6 (1959) 290–297.

[19] D. J. Watts, S. H. Strogatz, Collective dynamics of 'small-world' networks, Nature 393 (6684) (1998) 440–442.

[20] D. E. Knuth, The Art of Computer Programming, Addison-Wesley Professional, Amsterdam, 2011.

[21] A. Rényi, G. Szekeres, On the height of trees, Journal of the Australian Mathematical Society 7 (4) (1967) 497–507.

[22] A. Meir, J. W. Moon, The distance between points in random trees, Journal of Combinatorial Theory 8 (1) (1970) 99–103.

[23] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (5439) (1999) 509–512.

[24] R. Cohen, S. Havlin, Scale-free networks are ultrasmall, Physical Review Letters 90 (5) (2003) 058701.

[25] American Physical Society, APS Dataset, `http://journals.aps.org/datasets` (2015).

[26] Institute of Information Science, SICRIS Database, `http://www.sicris.si` (2010).

[27] C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, M. Tyers, BioGRID: A general repository for interaction datasets, Nucleic Acids Research 34 (1) (2006) 535–539.

[28] Biological General Repository for Interaction Datasets, BioGRID Database, `http://thebiogrid.org` (2016).

[29] A. Paranjape, A. R. Benson, J. Leskovec, Motifs in temporal networks, in: Proceedings of the ACM International Conference on Web Search and Data Mining, Cambridge, UK, 2017, pp. 601–610.

[30] J. Leskovec, A. Krevl, SNAP Datasets, `http://snap.stanford.edu/data` (2014).

[31] A. L. Traud, P. J. Mucha, M. A. Porter, Social structure of Facebook networks, Physica A: Statistical Mechanics and its Applications 391 (16) (2012) 4165–4180.

[32] R. A. Rossi, N. K. Ahmed, Network data repository, `http://networkrepository.com` (2015).

[33] J. Leskovec, J. Kleinberg, C. Faloutsos, 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, pp. 177–187.

[34] A. Clauset, C. R. Shalizi, M. E. J. Newman, Power-law distributions in empirical data, SIAM Review 51 (4) (2009) 661–703.

[35] A. D. Broido, A. Clauset, Scale-free networks are rare, Nature Communications 10 (1) (2019) 1017.

[36] S. Fortunato, D. Hric, Community detection in networks: A user guide, Physics Reports 659 (2016) 1–44.

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

[38] L. Freeman, A set of measures of centrality based on betweenness, Sociometry 40 (1) (1977) 35–41.

[39] L. C. Freeman, Centrality in social networks: Conceptual clarification, Social Networks 1 (3) (1979) 215–239.

[40] U. Brandes, A faster algorithm for betweenness centrality, Journal of Mathematical Sociology 25 (2) (2001) 163–177.

[41] K.-L. Ma, C. W. Muelder, Large-scale graph visualization and analytics, Computer 46 (7) (2013) 39–46.

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

[43] P. Eades, A heuristic for graph drawing, Congressus Numerantium 42 (1984) 149–160.

[44] T. M. J. Fruchterman, E. M. Reingold, Graph drawing by force-directed placement, Software: Practice and Experience 21 (11) (1991) 1129–1164.

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