

1 Computing dense spanning trees of unweighted  
2 networks

3 Lovro Šubelj\*

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

---

6 **Abstract**

A 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. Prim's algorithm and 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.

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

---

\*Corresponding author.

Email address: [lovro.subelj@fri.uni-lj.si](mailto:lovro.subelj@fri.uni-lj.si) (Lovro Šubelj)

<sup>9</sup> **1. Introduction**

<sup>10</sup> Networks or graphs have become popular tools for analysing complex real-  
<sup>11</sup> world systems [1]. Examples include predicting the spread of contagious viruses [2],  
<sup>12</sup> the study of the interactome of species [3], understanding the structure of sci-  
<sup>13</sup> ence [4] and outreach of social connections online [5]. The sizes of today’s  
<sup>14</sup> networks are often in millions of nodes and edges, with the largest networks  
<sup>15</sup> being the WWW with more than a trillion web pages and the human brain  
<sup>16</sup> with close to a hundred billion neurons. As a result, the size of real networks  
<sup>17</sup> can make many practical applications computationally very demanding.

<sup>18</sup> Techniques to alleviate this issue include network simplification or sam-  
<sup>19</sup> pling [6, 7, 8], and revealing the so-called network backbone or skeleton [9,  
<sup>20</sup> 10, 11]. These approaches try to reduce the size of a network in a way that the  
<sup>21</sup> network still retains many of its structural properties. One of the most straight-  
<sup>22</sup> forward ways to simplify a network is to compute its spanning tree [12, 1], which  
<sup>23</sup> is a subgraph connecting all the nodes of a network with the minimum number  
<sup>24</sup> of edges. A spanning tree retains the connectivity of a network and possibly  
<sup>25</sup> other structural properties. In the case of weighted networks, one usually aims  
<sup>26</sup> to compute the minimum spanning tree, which is a subgraph connecting all the  
<sup>27</sup> nodes with the minimum overall weight of the edges. In the case of unweighted  
<sup>28</sup> networks, any spanning tree is in fact a “minimum” spanning tree.

<sup>29</sup> Prim’s algorithm and Kruskal’s algorithm are well known algorithms for  
<sup>30</sup> computing a minimum spanning tree of a weighted network [12, 1]. Although  
<sup>31</sup> developed primarily for weighted networks, the algorithms can be readily applied  
<sup>32</sup> to unweighted networks. However, the performance of these algorithms has not  
<sup>33</sup> yet been properly explored for unweighted networks. In particular, there exist  
<sup>34</sup> no theoretical guarantees of the algorithms, neither does the literature provide  
<sup>35</sup> any empirical comparison on large-scale networks. In this paper, we apply the  
<sup>36</sup> algorithms to more than a thousand real networks and compare them to differ-  
<sup>37</sup> ent priority-first search algorithms. We show that the structure of unweighted  
<sup>38</sup> networks is best preserved by an algorithm using the breadth-first search node

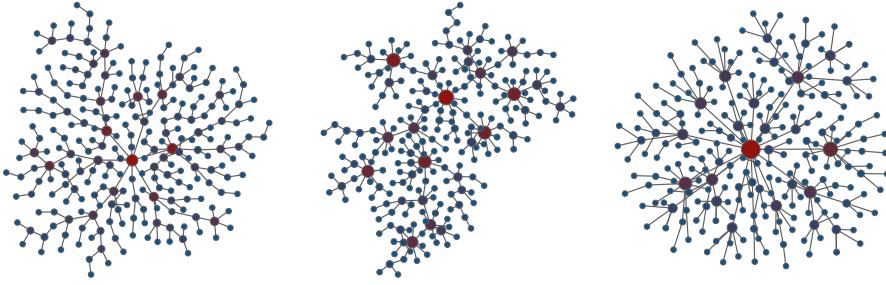


Figure 1: Wiring diagrams of spanning trees of a small random graph. The spanning trees were computed with the Prim’s algorithm (left), the Kruskal’s algorithm (middle) and the breadth-first search algorithm (right). The sizes of the nodes are proportional to their degrees, while the layouts were computed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

39 traversal. In particular, spanning trees computed with the breadth-first search  
 40 algorithm well retain the distances between the nodes of the network<sup>1</sup>, which  
 41 may be desirable in practical applications such as network visualization.

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

47 The rest of the paper is structured as follows. In the following section, we  
 48 first describe different algorithms for computing a spanning tree of a network.  
 49 Next, we analyze the structure of spanning trees of synthetic graphs and, after-  
 50 wards, the structure of spanning trees of real networks. Finally, we demonstrate  
 51 practical applications of spanning trees of networks. We conclude the paper with  
 52 suggestions for future research.

---

<sup>1</sup>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.

53    **2. Computation of spanning trees**

54    Let a network be represented by an undirected connected graph  $G = (V, E)$ ,  
55    where  $V$  denotes the set of nodes of  $G$  and  $E$  denotes the set of edges of  $G$ .  
56    (Where necessary to make explicit that these sets represent the graph  $G$ , we  
57    write  $V_G$  and  $E_G$ .) The number of nodes equals  $n = |V|$  and the number of  
58    edges equals  $m = |E|$ . We denote the average node degree as  $\langle k \rangle = 2m/n$ .  
59    Furthermore, let  $d_{ij}$  be the distance between the nodes  $i, j \in V$ , defined as the  
60    number of edges in a shortest path between the nodes  $i$  and  $j$ . Since the graph  
61    is undirected and connected,  $d_{ij} = d_{ji}$  and  $d_{ij} < \infty$ . Therefore, the average  
62    distance between the nodes equals  $\langle d \rangle = \frac{2}{n(n-1)} \sum_{i < j} d_{ij}$  and the maximum  
63    distance or diameter equals  $d_{\max} = \max_{i < j} d_{ij}$ . To measure the variability of  
64    the distances between the nodes, we also define the coefficient of variation as  
65     $c_d = \sigma_d / \langle d \rangle$ , where  $\sigma_d$  is the standard deviation of the distances.

66    Below we precisely describe different algorithms for computing a spanning  
67    tree of an undirected connected graph. In the case of a disconnected graph  
68    consisting of more than one connected component, the algorithms should be ap-  
69    plied to each of the connected components separately. Implementation of these  
70    algorithms is provided by most network analysis libraries such as *NetworkX*<sup>2</sup>.  
71    For a more extensive discussion on the differences between the algorithms, we  
72    refer the reader to classical graph theory literature [12] or network science lit-  
73    erature [17, 1].

74    *2.1. Prim's algorithm*

75    The Prim's algorithm for computing a spanning tree  $T$  of an undirected  
76    connected graph  $G$  operates as follows (see Algorithm 1). First, the algorithm  
77    selects a random seed node  $i \in V_G$  from graph  $G$  and adds it to an empty tree  $T$   
78    (lines 2, 3). The node  $i$  serves as a starting point for computing the spanning tree  
79     $T$ . Then, on each step of the algorithm (lines 4-8), a random edge  $\{i, j\} \in E_G$   
80    from graph  $G$  is selected that leads from a node  $i \in V_T$  already in the tree  $T$  to

---

<sup>2</sup><https://networkx.org>

---

**Algorithm 1** Prim's algorithm

---

**Require:** undirected graph  $G$

**Ensure:** spanning tree  $T$

```
1:  $T \leftarrow$  empty graph
2:  $i \leftarrow \text{RANDOM}(i \in V_G)$                                  $\triangleright$  Select seed node.
3: add node  $i$  to  $V_T$                                           $\triangleright$  Add selected seed node.
4: while  $\exists i \in V_G : i \notin V_T$  do                       $\triangleright$  There exists non-visited node?
5:    $\{i, j\} \leftarrow \text{RANDOM}(\{i, j\} \in E_G : i \in V_T \wedge j \notin V_T)$   $\triangleright$  Edge to non-visited node.
6:   add node  $j$  to  $V_T$                                           $\triangleright$  Add non-visited node.
7:   add edge  $\{i, j\}$  to  $E_T$                                       $\triangleright$  Add selected edge.
8: end while
9: return  $T$ 
```

---

81 a node  $j \notin V_T$  not yet in the tree  $T$  (line 5). Both node  $j$  and edge  $\{i, j\}$  are  
82 added to the tree  $T$  (lines 6, 7). Finally, when there is no further node  $i \in V_G$   
83 in graph  $G$  such that node  $i \notin V_T$  is not already in the tree  $T$ , the algorithm  
84 stops (line 4). At this point, the tree  $T$  is a spanning tree of graph  $G$  (line 9).

85 The Prim's algorithm is non-deterministic and can compute different span-  
86 ning trees. The actual spanning tree depends on random selection of the seed  
87 node (line 2 of Algorithm 1) and on random selection of the edges to expand the  
88 tree (line 5). The latter is nontrivial and most efficiently implemented by rejec-  
89 tion sampling over an array list of edges to non-visited nodes. For simplicity,  
90 we do not make these computations explicit in Algorithm 1.

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

97      *2.2. Kruskal's algorithm*

98      The Kruskal's algorithm is conceptually different from the Prim's algorithm.  
99      Instead of starting with a tree consisting of a seed node and then expanding it,  
100     the algorithm starts with a forest of trees, each consisting of a single node. The  
101     trees are then incrementally merged into larger trees by adding edges between  
102     them until only one tree remains. At this point the algorithm stops and the  
103     remaining tree is a spanning tree of a graph.

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

110     *2.3. Breadth-first search*

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

116     The breadth-first search algorithm for computing a spanning tree  $T$  of an  
117     undirected connected graph  $G$  operates as follows (see Algorithm 2). In contrast  
118     to before, we make all computations in Algorithm 2 explicit. First, the algorithm  
119     selects a random seed node  $i \in V_G$  from graph  $G$  and adds it to an empty tree  
120      $T$  (lines 3, 4). The node  $i$  is also added to an empty queue  $Q \subseteq V_T$  for further  
121     processing. Then, on each step of the algorithm (lines 5-11), a node  $i \in Q$  is  
122     removed from the beginning of the queue  $Q$  (line 6) and all edges that lead  
123     from node  $i \in V_T$  already in the tree  $T$  to nodes  $j \notin V_T$  not yet in the tree  $T$   
124     are processed (lines 7-10). All nodes  $j$  and edges  $\{i, j\}$  are added to the tree  $T$   
125     (lines 8, 9), while nodes  $j$  are also added to the queue  $Q$  for further processing.

---

**Algorithm 2** Breadth-first search

---

**Require:** undirected graph  $G$

**Ensure:** spanning tree  $T$

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

---

126 Finally, when there is no further node  $i \in Q$  in the queue  $Q$ , the algorithm stops  
127 (line 5). At this point the tree  $T$  is a spanning tree of graph  $G$  (line 12).

128 The breadth-first search algorithm is again non-deterministic, while the ac-  
129 tual spanning tree depends on random selection of the seed node (line 3 of Al-  
130 gorithm 2) and on the exact order in which the edges to expand the tree are  
131 processed (line 7). The time complexity of the algorithm using a queue of  
132 non-processed nodes is  $\mathcal{O}(m)$ , for either weighted or unweighted graphs. As an  
133 example, the right graph in Figure 1 shows a spanning tree computed with the  
134 breadth-first search algorithm.

135 *2.4. Other algorithms*

136 Other algorithms for computing a spanning tree include the Sollin's algo-  
137 rithm, which can be seen as a combination of the Prim's and Kruskal's ap-  
138 proaches, and the depth-first search node traversal. While the breadth-first  
139 search algorithm processes nodes of a graph using a level order traversal, the  
140 depth-first search algorithm uses a preorder traversal. This means that the only

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

146 **3. Spanning trees of synthetic graphs**

147 Consider an Erdős-Rényi random graph [18] with  $n$  nodes and the probability  
148 of an edge between each pair of nodes  $p = \langle k \rangle / (n - 1)$ , where  $\langle k \rangle$  is the expected  
149 node degree. A spanning tree of any connected graph with  $n$  nodes consists of  
150  $n$  nodes and  $n - 1$  edges with the average node degree  $\langle k \rangle = 2 - 2/n$ . Since it is  
151 a tree, the average clustering coefficient equals  $\langle C \rangle = 0$  [19]. We, therefore, here  
152 focus on other graph properties. In particular, we study the average distance  
153 between the nodes  $\langle d \rangle$  and the diameter  $d_{\max}$ . A theoretical estimate for the  
154 diameter  $d_{\max}$  of a random graph equals  $\log n / \log \langle k \rangle$  [1], which is  $d_{\max} =$   
155 2.40 for  $n = 250$  and  $\langle k \rangle = 10$ . Due to the sensitivity of the diameter  $d_{\max}$   
156 for relatively small  $n$  and  $\langle k \rangle$ , this turns out to be a better estimate of the  
157 average distance between the nodes  $\langle d \rangle$ . Indeed, the empirical estimates for the  
158 considered random graph are  $\langle d \rangle \approx 2.64$  and  $d_{\max} \approx 4.39$ .

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

168 The average distance  $\langle d \rangle$  and the diameter  $d_{\max}$  of a balanced tree are in  
169  $\mathcal{O}(\log n)$  for any practical definition of balance [20]. However, in the case of a

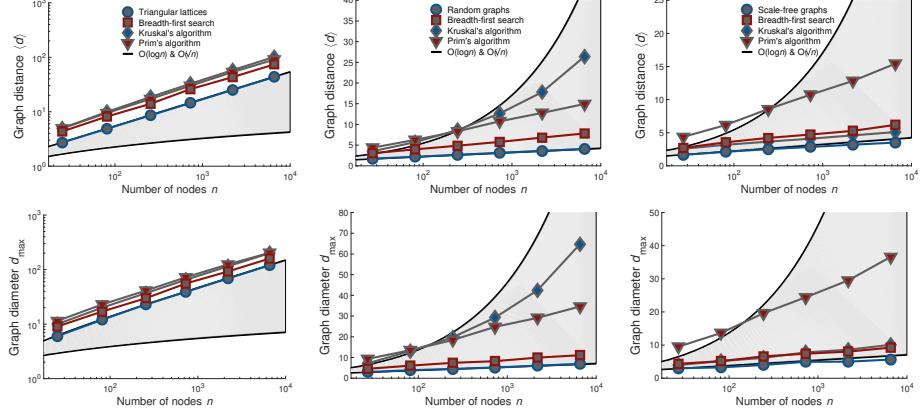


Figure 2: The average distance  $\langle d \rangle$  and the diameter  $d_{\max}$  of triangular lattices (left), random graphs (middle) and scale-free graphs (right), and their spanning trees computed with different algorithms (legend). The plots show estimates over 100 realizations, while the shaded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

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-

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  $d_{\max} = 36.6$  compared to 3.51 and 5.64 for graphs with  $n = 6561$  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

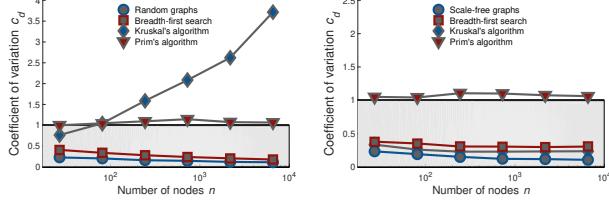


Figure 3: The coefficient of variation  $c_d$  for random (left) and scale-free graphs (right), and their spanning trees computed with different algorithms (legend). The plots show estimates over 100 realizations, while the error bars are smaller than the symbol sizes.

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 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

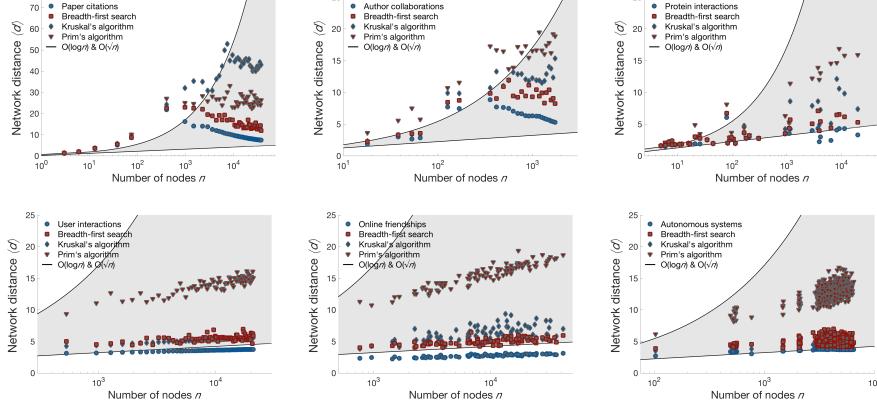


Figure 4: The average distance  $\langle d \rangle$  of real networks and their spanning trees computed with different algorithms (legend), while the shaded areas are the same as in Figure 2.

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 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 predic-

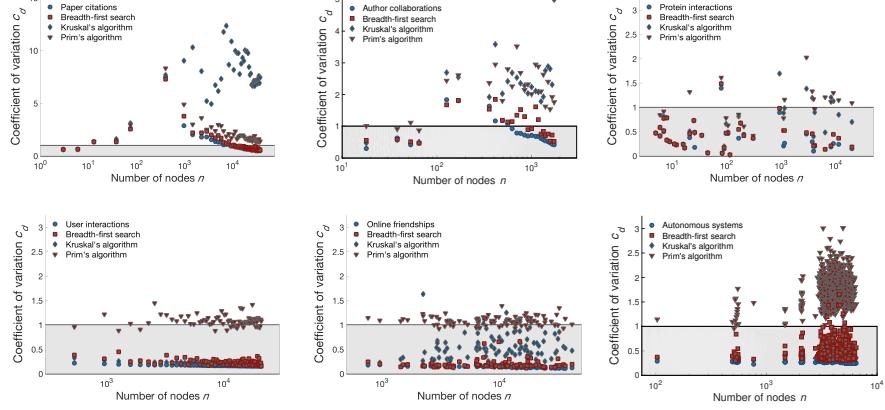


Figure 5: The coefficient of variation  $c_d$  for real networks and their spanning trees computed with different algorithms (legend), while other details are the same as in Figure 3.

tion. 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

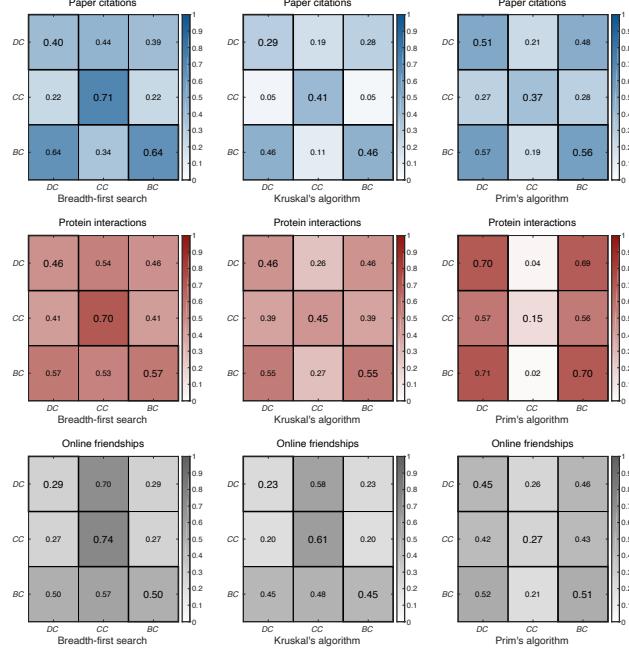


Figure 6: Pearson correlation coefficient between the measures of node centrality in real networks and their spanning trees computed with the breadth-first search algorithm (left), the Kruskal’s algorithm (middle) and the Prim’s algorithm (right). The measures include node degree centrality  $DC$ , closeness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

303 appears to be in the “center” of a network, which is often used in operations  
 304 research. The closeness centrality of a node  $i \in V$  of graph  $G(V, E)$  is defined  
 305 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 dis-  
 306 tance between the nodes  $i, j \in V$ . The time complexity of computing closeness  
 307 centrality of all nodes in a network is  $\mathcal{O}(nm)$  and no more efficient algorithm  
 308 exists [20].

309 Figure 6 shows the Pearson correlation coefficient between node closeness  
 310 centrality in real networks and their spanning trees computed with the breadth-  
 311 first search algorithm, the Kruskal’s algorithm and the Prim’s algorithm. The  
 312 coefficients for the breadth-first search algorithm are shown in the center of the  
 313 heatmaps in the left column of Figure 6. These correlations are all  $\geq 0.70$ , which

314 means strong linear correlation. On the other hand, the correlation coefficients  
315 for the spanning trees computed with the Kruskal’s algorithm in the middle  
316 column of Figure 6 and the Prim’s algorithm in the right column are  $\leq 0.61$   
317 and  $\leq 0.37$ , respectively. Therefore, consistent with the previous results, the  
318 breadth-first search algorithm best preserves the distances between the nodes  
319 of real networks, even on the level of individual nodes.

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

326 *5.2. Network visualization*

327 Besides providing computational benefits, a network simplification technique  
328 such as a spanning tree can be useful in network visualization. Any visualiza-  
329 tion with a wiring diagram or some other approach is limited in the size of a  
330 network it can represent and in the structural characteristics of a network it can  
331 reveal [41, 42]. Classical algorithms for computing a layout of a network include  
332 spring embedding or force-directed algorithms [43, 44] and algorithms that em-  
333 bed the nodes in a plane thus their Euclidean distance matches their network  
334 distance as best as possible [45]. It is, therefore, important that a spanning tree  
335 preserves the distances between the nodes of a network if it is to be used in  
336 network visualization.

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

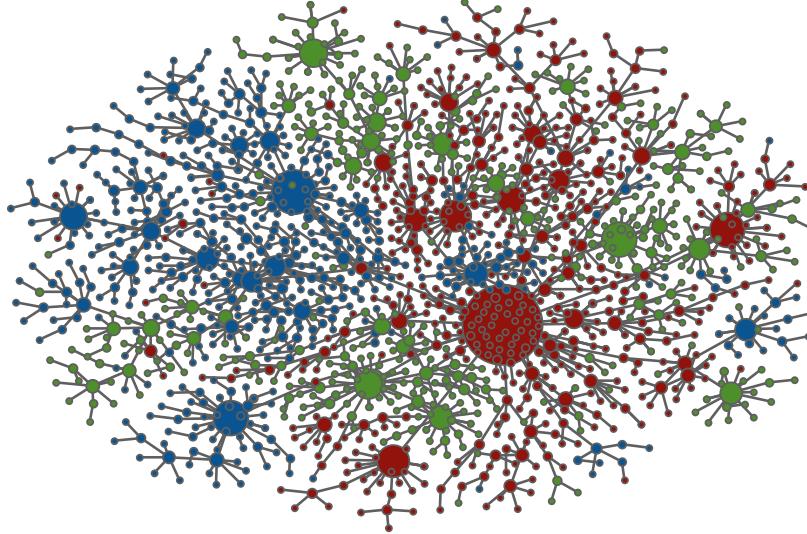


Figure 7: Spanning tree of the SICRIS author collaboration network computed with the breadth-first search algorithm. The sizes of the nodes are proportional to their degrees, while the colors represent primary author disciplines consisting of natural sciences (red), engineering (green), medical sciences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

<sup>344</sup> of the quality of a network visualization, we refrain from providing any further  
<sup>345</sup> subjective interpretation.

## <sup>346</sup> 6. Conclusions

<sup>347</sup> 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  
<sup>348</sup> known algorithms for computing a spanning tree of a weighted network are  
<sup>349</sup> the Prim’s algorithm and the Kruskal’s algorithm [12, 1]. However, when applied  
<sup>350</sup> to unweighted networks, these algorithms do not necessarily capture the  
<sup>351</sup> structure of a network such as short distances between the nodes and small  
<sup>352</sup> network diameter [19]. Prior to this work there existed no empirical evaluation  
<sup>353</sup> or comparison of the algorithms on large-scale networks. We have shown  
<sup>354</sup> that an algorithm based on the breadth-first search node traversal well retains  
<sup>355</sup>

356 the distances between the nodes in synthetic graphs and real networks. As we  
357 demonstrate, this can provide computational benefits and is also important in  
358 practical applications like network visualization. Thus, if a spanning tree of  
359 an unweighted network is supposed to retain the distances between the nodes,  
360 then the breadth-first search algorithm should be preferred to other algorithms  
361 considered here. At the same time, spanning trees that minimize the distances  
362 measured by the Wiener index [13] are known as dense spanning trees in chem-  
363 ical graph theory. Our results might also be relevant in the context of such  
364 applications.

365 In the language of the theory of computation, the breadth-first search algo-  
366 rithm computes a spanning tree with properties of a balanced tree [20]. Due to  
367 the lack of a formal definition of some sort of approximate balance that would  
368 *already* imply short distances between the nodes, our results here are merely  
369 empirical. In future research, we plan to develop such a definition that could  
370 support the results also analytically.

371 **Acknowledgments**

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

375 **Appendix A. Diameter and degree distribution**

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

380 Figure A.9 shows the node degree distribution  $p_k$  of the largest networks  
381 in Table 1 and their spanning trees computed with the breadth-first search  
382 algorithm. Under the goodness-of-fit test at  $p$ -value = 0.1 [34], a power-law  $p_k \sim$   
383  $k^{-\gamma}$  is a plausible fit of the degree distribution  $p_k$  for the protein interactions

384 and autonomous systems networks in the right column of Figure A.9. On the  
385 other hand, the degree distribution  $p_k$  of the spanning trees follows a power-  
386 law in all cases but the online friendships network. The maximum likelihood  
387 estimates of the power-law exponents  $\gamma$  are shown with solid lines in Figure A.9.

388 **References**

- 389 [1] M. E. J. Newman, Networks, 2nd Edition, Oxford University Press, Oxford,  
390 2018.
- 391 [2] M. Tizzoni, P. Bajardi, C. Poletto, J. J. Ramasco, D. Balcan, B. Gonçalves,  
392 N. Perra, V. Colizza, A. Vespignani, Real-time numerical forecast of global  
393 epidemic spreading: case study of 2009 A/H1N1pdm, BMC Medicine 10 (1)  
394 (2012) 165.
- 395 [3] M. Zitnik, R. Sosič, M. W. Feldman, J. Leskovec, Evolution of resilience  
396 in protein interactomes across the tree of life, Proceedings of the National  
397 Academy of Sciences of the United States of America 116 (10) (2019) 4426–  
398 4433.
- 399 [4] S. Fortunato, C. T. Bergstrom, K. Börner, J. A. Evans, D. Helbing,  
400 S. Milojević, A. M. Petersen, F. Radicchi, R. Sinatra, B. Uzzi, A. Vespig-  
401 nani, L. Waltman, D. Wang, A.-L. Barabási, Science of science, Science  
402 359 (6379) (2018) eaao0185.
- 403 [5] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, S. Vigna, Four degrees of  
404 separation, in: Proceedings of the ACM International Conference on Web  
405 Science, Evanston, IL, USA, 2012, pp. 45–54.
- 406 [6] J. Leskovec, C. Faloutsos, Sampling from large graphs, in: Proceedings of  
407 the ACM SIGKDD International Conference on Knowledge Discovery and  
408 Data Mining, Philadelphia, PA, USA, 2006, pp. 631–636.

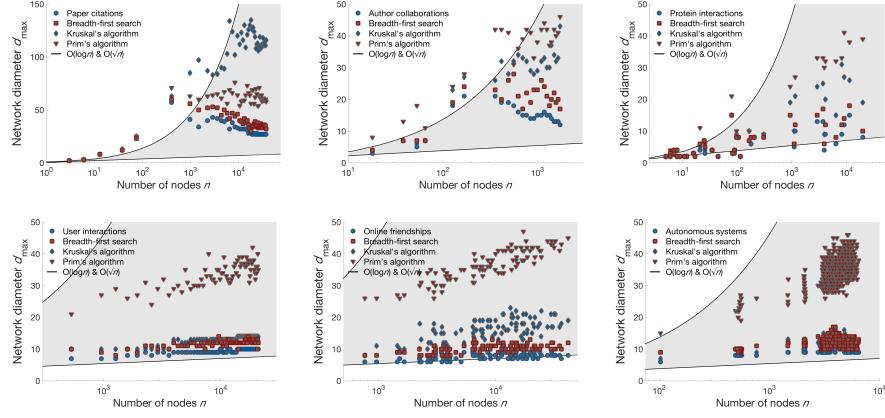


Figure A.8: The diameter  $d_{\max}$  of real networks and their spanning trees computed with different algorithms (legend), while the shaded areas are the same as in Figure 4.

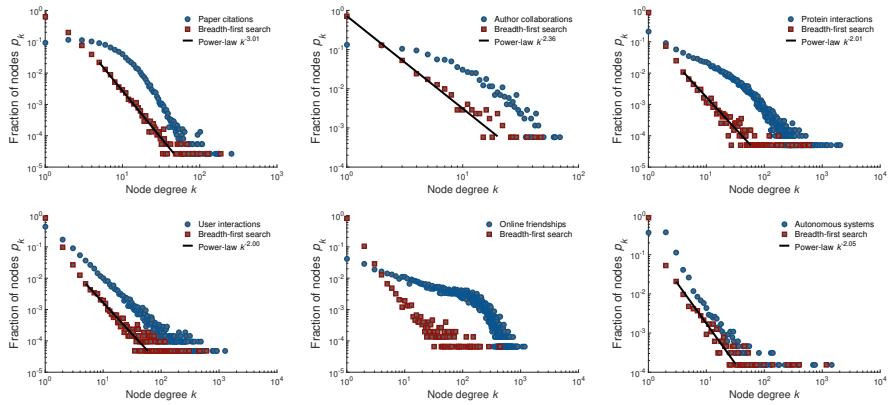


Figure A.9: Node degree distribution  $p_k$  of real networks and their spanning trees computed with the breadth-first search algorithm. The power-law distributions are maximum likelihood estimates at  $p$ -value = 0.1 [34].

- 409 [7] M. Hamann, G. Lindner, H. Meyerhenke, C. L. Staudt, D. Wagner,  
410 Structure-preserving sparsification methods for social networks, Social Net-  
411 work Analysis and Mining 6 (1) (2016) 22.
- 412 [8] N. Blagus, L. Šubelj, M. Bajec, Empirical comparison of network sam-  
413 pling: How to choose the most appropriate method?, Physica A: Statistical  
414 Mechanics and its Applications 477 (2017) 136–148.
- 415 [9] D. Grady, C. Thiemann, D. Brockmann, Robust classification of salient  
416 links in complex networks, Nature Communications 3 (2012) 864.
- 417 [10] M. Coscia, F. Neffke, Network backboning with noisy data, in: Proceedings  
418 of the IEEE International Conference on Data Engineering, San Diego, CA,  
419 USA, 2017, pp. 425–436.
- 420 [11] L. Šubelj, Convex skeletons of complex networks, Journal of the Royal  
421 Society Interface 15 (145) (2018) 20180422.
- 422 [12] B. Bollobás, Modern Graph Theory, Springer, Heidelberg, 1998.
- 423 [13] H. Wiener, Structural determination of paraffin boiling points, Journal of  
424 the American Chemical Society 69 (1) (1947) 17–20.
- 425 [14] M. Ozen, H. Wang, K. Wnag, D. Yalman, An edge-swap heuristic for find-  
426 ing dense spanning trees, Theory and Applications of Graphs 3 (1) (2016)  
427 1.
- 428 [15] M. Ozen, G. Lesaja, H. Wang, Globally optimal dense and sparse span-  
429 ning trees, and their applications, Statistics, Optimization & Information  
430 Computing 8 (2) (2020) 328–345.
- 431 [16] A. T. Adai, S. V. Date, S. Wieland, E. M. Marcotte, LGL: Creating a map  
432 of protein function with an algorithm for visualizing very large biological  
433 networks, Journal of Molecular Biology 340 (1) (2004) 179–190.
- 434 [17] A.-L. Barabási, Network Science, Cambridge University Press, Cambridge,  
435 2016.

- 436 [18] P. Erdős, A. Rényi, On random graphs I, *Publicationes Mathematicae De-*  
437 *brecen* 6 (1959) 290–297.
- 438 [19] D. J. Watts, S. H. Strogatz, Collective dynamics of 'small-world' networks,  
439 *Nature* 393 (6684) (1998) 440–442.
- 440 [20] D. E. Knuth, *The Art of Computer Programming*, Addison-Wesley Profes-
- 441 *sional*, Amsterdam, 2011.
- 442 [21] A. Rényi, G. Szekeres, On the height of trees, *Journal of the Australian*  
443 *Mathematical Society* 7 (4) (1967) 497–507.
- 444 [22] A. Meir, J. W. Moon, The distance between points in random trees, *Journal*  
445 *of Combinatorial Theory* 8 (1) (1970) 99–103.
- 446 [23] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, *Sci-*  
447 *ence* 286 (5439) (1999) 509–512.
- 448 [24] R. Cohen, S. Havlin, Scale-free networks are ultrasmall, *Physical Review*  
449 *Letters* 90 (5) (2003) 058701.
- 450 [25] American Physical Society, APS Dataset, <http://journals.aps.org/datasets> (2015).
- 452 [26] Institute of Information Science, SICRIS Database, <http://www.sicris.si> (2010).
- 454 [27] C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, M. Ty-  
455 *ers*, BioGRID: A general repository for interaction datasets, *Nucleic Acids*  
456 *Research* 34 (1) (2006) 535–539.
- 457 [28] Biological General Repository for Interaction Datasets, BioGRID  
458 Database, <http://thebiogrid.org> (2016).
- 459 [29] A. Paranjape, A. R. Benson, J. Leskovec, Motifs in temporal networks, in:  
460 Proceedings of the ACM International Conference on Web Search and Data  
461 Mining, Cambridge, UK, 2017, pp. 601–610.

- 462 [30] J. Leskovec, A. Krevl, SNAP Datasets, <http://snap.stanford.edu/data>  
463 (2014).
- 464 [31] A. L. Traud, P. J. Mucha, M. A. Porter, Social structure of Facebook  
465 networks, *Physica A: Statistical Mechanics and its Applications* 391 (16)  
466 (2012) 4165–4180.
- 467 [32] R. A. Rossi, N. K. Ahmed, Network data repository, [http://](http://networkrepository.com)  
468 [networkrepository.com](http://networkrepository.com) (2015).
- 469 [33] J. Leskovec, J. Kleinberg, C. Faloutsos, Graphs over time: Densification  
470 laws, shrinking diameters and possible explanations, in: Proceedings of  
471 the ACM SIGKDD International Conference on Knowledge Discovery and  
472 Data Mining, Chicago, IL, USA, 2005, pp. 177–187.
- 473 [34] A. Clauset, C. R. Shalizi, M. E. J. Newman, Power-law distributions in  
474 empirical data, *SIAM Review* 51 (4) (2009) 661–703.
- 475 [35] A. D. Broido, A. Clauset, Scale-free networks are rare, *Nature Communications* 10 (1) (2019) 1017.
- 477 [36] S. Fortunato, D. Hric, Community detection in networks: A user guide,  
478 *Physics Reports* 659 (2016) 1–44.
- 479 [37] D. Schoch, U. Brandes, Re-conceptualizing centrality in social networks,  
480 *European Journal of Applied Mathematics* 27 (6) (2016) 971–985.
- 481 [38] L. Freeman, A set of measures of centrality based on betweenness, *Sociometry*  
482 40 (1) (1977) 35–41.
- 483 [39] L. C. Freeman, Centrality in social networks: Conceptual clarification, *Social Networks* 1 (3) (1979) 215–239.
- 485 [40] U. Brandes, A faster algorithm for betweenness centrality, *Journal of Mathematical Sociology* 25 (2) (2001) 163–177.

- 487 [41] K.-L. Ma, C. W. Muelder, Large-scale graph visualization and analytics,  
488 Computer 46 (7) (2013) 39–46.
- 489 [42] H. Gibson, J. Faith, P. Vickers, A survey of two-dimensional graph layout  
490 techniques for information visualisation, Information Visualization 12 (3-4)  
491 (2013) 324–357.
- 492 [43] P. Eades, A heuristic for graph drawing, Congressus Numerantium 42  
493 (1984) 149–160.
- 494 [44] T. M. J. Fruchterman, E. M. Reingold, Graph drawing by force-directed  
495 placement, Software: Practice and Experience 21 (11) (1991) 1129–1164.
- 496 [45] T. Kamada, S. Kawai, An algorithm for drawing general undirected graphs,  
497 Information Processing Letters 31 (1) (1989) 7–15.

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]