São José dos Campos - SP
LV SBPO
LV Simpósio Brasileiro de Pesquisa Operacional
São José dos Campos, SP - 6 a 9 de novembro de 2023

Heuristics for t-admissibility with complex network approach

Carlos Santos
Universidade Federal Fluminense
Departamento de Ciência da Computação - IC - Niterói - RJ
carlosthadeu@id.uff.br

Anderson Zudio
Universidade Federal Fluminense
Departamento de Ciência da Computação - IC - Niterói - RJ
azudio@id.uff.br

Leandro Santiago
Universidade Federal Fluminense
Departamento de Ciência da Computação - IC - Niterói - RJ
leandro@ic.uff.br

Luís Cunha
Universidade Federal Fluminense
Departamento de Ciência da Computação - IC - Niterói - RJ
lfignacio@ic.uff.br

ABSTRACT

The t-admissibility is a min-max problem that concerns to determine whether a graph G contains a spanning tree T in which the distance between any two adjacent vertices of G is at most t in T. The stretch index of G, $\sigma(G)$, is the smallest t for which G is t-admissible. This problem is in P for $t \leq 2$, NP-complete for $\sigma(G) \leq t$, $t \geq 4$, and remaining open for $t = 3$. Recently, Couto et al. (2022) developed some greedy heuristics to construct a candidate solution tree, but left open how to decide between two vertices when both have equal chances of being taken in a greedy step. This criterion is important, since different branches can yield different stretch indexes. In order to answer such a question, we develop four new heuristics by exploring vertex importance of complex network. The results are compared considering Barabási-Albert, Erdös-Rényi, Watts-Strogatz, and Bipartite graphs.

KEYWORDS. **t-admissibility. Tree t-spanner. Centrality measures**

Graph Theory. Graph algorithms. Heuristics

1. Introduction

The problem of constructing spanners has been extensively studied in graph theory and has numerous applications in network design and routing. A spanner is a subgraph of a given graph G that has all vertices of G. Given a spanning tree T of a graph G, if the maximum distance in T between adjacent vertices in G is at most t, then G is called t-admissible. The integer t is the *stretch factor* of T, denoted by $\sigma(T)$, and the smallest t such that G is t-admissible is the *stretch index* of G, denoted by $\sigma(G)$ [Couto et al., 2022]. Formally, the t-ADMISSIBILITY problem can be stated as follows: given a connected graph G with edge set $E(G)$, determine whether there exists a spanning tree T of G such that for any $uv \in E(G)$, the distance $d_T(u, v)$ is at most t.

The concept of t-admissibility has applications in network design and routing algorithms, and its solution provide insights into the quality of the input graph. For several years, researchers have been exploring approximation algorithms and brute-force strategies to determine the stretch index of a given graph. In 1995, Cai e Corneil [1995] proved that the problem can be solved in linear time for $t = 2$, while it is NP-complete for $t \geq 4$. Moreover, the complexity of the problem for $t = 3$ remains an unsolved question so far. A possible way to determine the stretch index of a graph G is constructing all spanning trees of G, determining the stretch factor and obtaining the smallest factor. Despite that, building all spanning trees is inefficient, especially for dense graphs. The number of spanning trees that a complete graph has grows exponentially and is determined by Cayley's formula [West et al., 1996; Chartrand e Zhang, 2012], which states that for every positive integer n, the number of labeled spanning trees is $n^{(n-2)}$.

Researchers have attempted heuristics, sequential and parallel algorithms to the t-admissibility problem [Couto et al., 2022]. To compute the stretch index in brute force approaches, two lower bounds are considered. The first one is the *girth lower bound*, which is the length of a shortest induced cycle of G (the girth, $g(G)$) minus one, expressed as $\sigma(G) \geq g(G) - 1$. Let e be an edge of G, we denote $l(e)$ as the length of the smallest cycle minus one that contains e in G, if e does not belong to any cycle, $l(e) = 1$. Another lower bound for $\sigma(G)$ is the *smallest-e-cycle lower bound*, which is given by $max_{e \in E(G)} l(e)$ [Couto et al., 2022, p. 2]. For sequential brute-force algorithms, in the worst case its time complexity is a function of the number of spanning trees of a graph. Spanning trees obtained by parallel brute force algorithms may have repeated and unbalanced trees generated per thread. Greedy heuristics from previous works determine spanning trees of graphs by considering highest degree vertices to be chosen at each step [Couto et al., 2022]. Those approaches yield good results, but they face a crucial issue when multiple vertices have equal degrees: which one should be chosen? As one can note, considering vertices with same degree depending the vertex we have chosen, two different results can be obtained. See Figure 1 for an example.

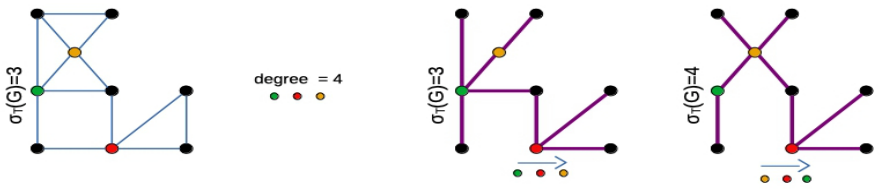


Figure 1: Graph G whose $\sigma_T(G) = 3$. Heuristic proposed by Couto et al. [2022], when starting from green vertex returns that G is 3-admissible, but starting from yellow vertex it returns that G is 4-admissible.

We propose four heuristics that utilize centrality measures in graphs. Incorporating degree, closeness and leverage centrality into our research, we improve the understanding of vertex importance in order to obtain better spanning trees.

2. Centrality measures

Before examining our proposed algorithms, it is important to first discuss the strategy utilized to improve their accuracy by the use of centrality measures. It is worth to note that in the following subsections, we use the term *"network"* to refer a graph as it is more commonly used in the context of centrality measures. Centrality measures have been studied since the early 1950s, and they enable us to identify the most central or influential vertex in a network. According to Das et al. [2018]; Chen e Evans [2022]; Joyce et al. [2010]; Scott e Carrington [2011], centrality measures are commonly used in network analysis to identify the most important vertices in a network.

There are many different types of centrality measures, where degree centrality is the most popular. *Degree centrality* simply counts the number of edges that are incident to a vertex. Vertices with higher degree centrality are perceived as more important, but it is limited and provides only local information about a vertex. Another type of centrality measure is *closeness centrality*, which is based on shortest paths between a vertex and all other vertices in the network. Vertices with high closeness centrality are those that can reach other vertices more quickly, and they are thus more efficient at transmitting information or influence through the network. The third and last type of centrality measure that interests us is *leverage centrality*. It is based on the idea that a vertex is more influential if it has connections to other highly influential vertices in the network.

2.1. Degree centrality

As previously mentioned in Section 2, degree centrality is the simplest centrality measure to calculate. It is defined as the total number of edges incident to a particular vertex, which indicates the number of connections that the vertex has. However, a high degree centrality does not necessarily imply that the vertex is more central or efficient in transmitting information within the network. Other centrality measures need to be considered, as we will discuss shortly. For degree centrality, higher values indicate greater centrality but not necessarily a position in the "middle" of the network [Golbeck, 2015]. The formula to compute the degree centrality of a vertex v using an adjacency matrix is:

$$D_C(v) = \sum_{u=1}^{n} A_{uv, u \neq v} \tag{1}$$

where v is the vertex, u is the probable neighbor of v, A_{uv} is equal to 1 if vertices u and v are connected, and 0 otherwise, n is the size of the neighbors set of vertex v. It is important to note that the degree centrality of vertices is independent of each other.

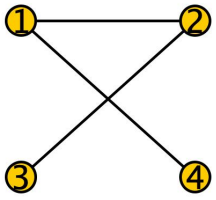


Figure 2: A network.

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \begin{aligned} D_C(1) &= 0+1+0+1 = 2 \\ D_C(2) &= 0+1+0+1 = 2 \\ D_C(3) &= 0+1+0+0 = 1 \\ D_C(4) &= 1+0+0+0 = 1 \end{aligned}$$

Table 1: Degree centrality of Figure 2.

Alternatively, one can compute the degree centrality of a vertex v by assigning the value of its degree, denoted by $d(v)$ to $D_C(v)$, i.e., $D_C(v) = d(v)$, as we can see in the Figure 2 and Table 1. Analogous to a people network, the most important individuals are those who provide the most advice to others. Additionally, individuals who receive advice from others are also important. In a people network, it is often based on the prestige or status of an individual that determines their

ability to exert immediate influence. In social network analysis, a similar concept is used to measure vertex importance. One of the big weaknesses of this measure is that it may not be able to accurately rank vertices when multiple vertices have the same degree of prestige or status. This is because it is difficult to determine which of these vertices is truly more important than the others.

Considering an undirected graph, due to direct nature by calculating the connections of each node individually, by traversing each edge once, it counts inbound and outbound node connections, making a count proportional to edge count, the time of complexity required to calculate the degree of all vertices is $\mathcal{O}(|E|)$, where $|E|$ is the set of edges, according Das et al. [2018] and Zhuge e Zhang [2009].

2.2. Closeness centrality

Closeness centrality is a metric that calculates the average length of the shortest path from a vertex v to all other vertices in a network G. It indicates how close a vertex is connected to other vertices in a network and it measures the influence that the vertex exerts over the network.

The equation used to compute closeness centrality is:

$$C_C(v) = \frac{1}{\sum\limits_{u \in V(G) \setminus v} d(u, v)} \tag{2}$$

Considering the closeness centrality, we have an efficient measure for traversing the network, the vertex with the highest closeness centrality is more efficient to reach all other vertices in the network. Since computing the closeness centrality for all vertices has the time complexity of $\mathcal{O}(|V|.(|E| + |V| \log |V|))$, Chen e Evans [2022] have proposed a mathematical formula to compute this metric approximately more efficient. Although the exact values takes more time to be determined, the order of vertices obtained by computing the closeness centrality using the proposed formula is consistent with the order obtained by traversing the network. However, using a mathematical formula to compute closeness centrality may result in a loss of accuracy, especially when multiple vertices have the same value. To address this, it is necessary to consider additional metrics to determine the relative importance of vertices in the network. The following two equations have been proposed by Chen e Evans [2022].

$$\frac{1}{Cr} = -\frac{1}{\ln(\bar{z})} \ln(k_r) + \beta \tag{3}$$

$$\beta(\bar{z}, N) = \Big(\frac{1}{(\bar{z} - 1)} + \frac{\ln(\bar{z} - 1)}{\ln(\bar{z})}\Big) + \frac{1}{\ln(\bar{z})} \ln(N) \tag{4}$$

Where N is the number of vertices, k_r is the degree of each node k_r, $\bar{z}$ is network parameter to represent the exponential growth from any root node r, C_r is the closeness.

The computation of closeness centrality involves a time complexity of $\mathcal{O}(|V| \cdot |E|)$ for all vertices in not very large graphs, using the efficient Brandes algorithm Das et al. [2018]. Alternatively, when employing the Breadth-First-Search(BFS) algorithm to traverse the graph, the time complexity for computing closeness centrality increases to $\mathcal{O}(|V|.|E| + |V|^2)$.

2.3. Leverage centrality

Previously, in undirected networks, we only used degree centrality to distinguish vertices from each other based on the number of connections they have. A vertex with more connections would be considered more important, but this approach was limited to local connections only. Leverage centrality expands upon this concept. The concept of leverage centrality was first introduced in Joyce et al. [2010]. This measure expands upon the previous notion of degree centrality. Leverage

centrality takes into account both the direct connections of the vertex and the connections of its neighbors. The formula defining leverage centrality is given by:

$$L_C(v) = \frac{1}{d(v)} \sum_{v_j \in N(v)} \frac{d(v) - d(v_j)}{d(v) + d(v_j)}, \tag{5}$$

where $N(v)$ is the neighborhood of v. In network analysis, the leverage centrality allows us to consider not only a vertex's direct connections but also its neighbors. A vertex with a positive leverage centrality value is seen as influential on its neighbors, while a negative value indicates that the vertex is being influenced by its neighbors. In other words, a vertex with high leverage centrality is one that, if removed, would have a large impact on the connectivity of the network. This measure is often used in the context of identifying critical vertices in a network, and it has applications in fields such as social network analysis, transportation planning, and epidemiology. The time complexity is $\mathcal{O}(|V|^2)$.

3. Proposed strategies

In order to compute the stretch index using the proposed heuristics, it is important to identify the most central or influential vertex in the network. This vertex must be the root vertex for the tree-building process. To identify the most central vertex, we use the centrality measures.

3.1. Global MaxDegree Heuristic (H1)

Global MaxDegree Heuristic comes in two versions, namely `v1` (sorted by degree centrality, which was proposed by Couto et al. [2022]) and `v2` (sorted by degree and closeness centrality, proposed in this work). In that strategy, we sort and traverse the vertices in descending order and insert into tree T the vertex that is not yet in T and all its valid neighbors. This process repeats until a spanning tree is found, as summarized in Algorithm 1.

Algorithm 1: Heuristic H1

Data: A graph
Result: A spanner tree
Compute degree;
Compute closeness ; `// used only H1v2`
Compute leverage ; `// used only H1v2`
List all vertices in descending order by degree and closeness (only v2) and ascending order by leverage(only v2);
while *not all vertices at list are processed* **do**
    Insert in the tree, the first vertex of list is not in the tree yet and all valid neighbors in G that are not in the tree either;
    Read next vertex;
end

3.2. Local MaxDegree Heuristic (H2)

In Local MaxDegree Heuristic, the vertices of the graph G are sorted in descending order based on their degree centrality (`v1`) or closeness centrality (`v2`). Let v be the first vertex in the sorted list. An empty tree T is created and v is added to T along with all its neighboring vertices. A list of all the leaves of T is created and the vertex v with the highest number of neighbors not in T is selected using Equation 6. This selected vertex is then added to T along with all its valid neighbors. After adding v to T, it is removed from the list L, which is then updated using Equation 7. The updated list L is then sorted and vertices are added to T as long as there exists a vertex in L that is not in T. Therefore, Local MaxDegree Heuristic follows the steps outlined above to build a tree T by iteratively adding vertices in descending order of their degree centrality or closeness centrality, and selecting vertices with the highest number of non-tree neighbors until all vertices are added to T.

$$max_{v \in L}\{f(v)\} \tag{6}$$

$$f(v) = d_G(v) - d_t(v) \tag{7}$$

The algorithm for Local MaxDegree Heuristic `v1` and `v2` are almost the same, the difference between them comes down to the tie-breaking criterion in case two or more vertices in the list L have the same degree. Local MaxDegree Heuristic `v1` uses the first vertex with the same degree in the list, while Local MaxDegree Heuristic `v2` considers the better closeness centrality to select the root vertex among them (all vertices with same degree). The algorithm for Local MaxDegree Heuristic `v2` is summarized in Algorithm 2:

Algorithm 2: Heuristic H2v2

Data: A graph
Result: A spanner tree
Compute degree;
Compute closeness;
Compute leverage;
List all vertices sorting by number of neighbors;
Select vertex with the greater number of neighbors, if one more vertex is selected, tiebreaker between them with the greater closeness and the smaller leverage;
Add to tree the selected vertex and all neighbors;
List all leaves of tree;
while *there is a vertex in leaves list that not in tree* **do**
    From the list of leaves, select the vertex that has the highest numbers of neighbors not in tree yet using equation 6, if one more vertex is selected, tiebreaker between them with higher closeness and smaller leverage is need;
    Insert in tree the selected vertex and all valid neighbors;
    Remove the select vertex from list;
    Update the list of leaves and recalculate at list the numbers of neighbors not in tree using equation 7;
end

3.3. Adaptative Global MaxDegree Heuristic (H3)

Adaptive Local MaxDegree Heuristic is a combination of Global MaxDegree Heuristic and Local MaxDegree Heuristic. In this approach, the vertices of the graph G are sorted in descending order based on their degree centrality. Unlike Local MaxDegree Heuristic (H2), all vertices are updated using Equations 6 and 7.

Adaptative Global MaxDegree Heuristic version 1 (H3v1) In the initial version (`v1`) of Adaptative Global MaxDegree Heuristic, the vertices are sorted based on their degree centrality, and the list is updated using Equation 7. However, if multiple vertices have the same degree, no tie-breaking mechanism is employed, and the first vertex from the sorted list is selected for insertion into the tree. The algorithm for Adaptative Global MaxDegree Heuristic `v1` is as follows in Algorithm 3:

Algorithm 3: Heuristic H3v1

Data: A graph
Result: A spanner tree
Initialization;
From graph, List all vertices sorting in decreasing by number of neighbors (L);
while *spanner tree is not created* **do**
    From L Select vertex with the greater number of neighbors (equation 6);
    Add to tree the selected vertex and all valid neighbors;
    For all vertices, update the list of vertices and recalculate with the number of neighbors not in tree using equation 7;
    Remove the select vertex from list;
    Resorting the list of vertices;
end

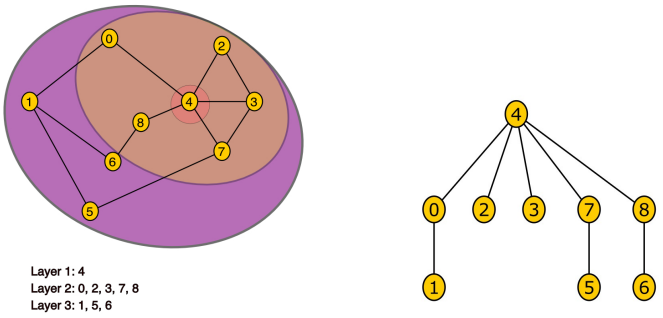
Adaptative Global MaxDegree Heuristic version 2 (H3v2) Adaptative Global MaxDegree Heuristic `v2` is a slight modification of `v1`. In this version, when vertices has same degree, we consider additional measures such as closeness centrality and leverage centrality to determine the most appropriate vertex for selection. By applying these centrality measures to the vertices with the same degree, we create a refined list and choose the vertex that is deemed most suitable, summarized in Algorithm 4.

Algorithm 4: Heuristic H3v2

Data: A graph
Result: A spanner tree
Initialization;
From graph, list all vertices(L) in decrease order by number of neighbors;
Compute closeness for all vertices;
Compute leverage for all vertices;
while *spanner tree not created* **do**
    From L select the vertex with the greater number of neighbors, if there a one more vertex selected, tiebreaker with the highest closeness and smallest leverage;
    Add to the tree, the selected vertex and all valid neighbors;
    Remove the selected vertex from list;
    Resorting the list of vertices;
    Update vertices list with the number of neighbors not in tree using equation 7;
end

3.4. Centrality Heuristic (H4)

Centrality Heuristic aims to promote horizontal growth of a tree, rather than vertical growth, by expanding from an initial vertex in layers, similar to breadth-first search (BFS). Instead of searching for vertices like BFS, Centrality Heuristic simply retrieves a list of neighbors from the vertices on the last layer and inserts them into the tree. To implement Centrality Heuristic, a graph G is first selected, and a tree T is imagined to grow in layers. The root vertex (v) is placed on layer 1, its direct neighbors are on layer 2, and the direct neighbors of those neighbors (that are not yet in T) are on layer 3, and so on, see Figure 3. This ensures that the tree grows horizontally, with vertices being added to each layer as the tree expands. In a tree T built in this manner, the maximum distance between any two vertices is given by $2l - 2$, l is denoted the number of layers where $l > 0$.



(a) Layers strategy.

(b) Centrality Heuristic solution.

Figure 3: Centrality Heuristic.

In this Section, we conducted a comprehensive evaluation of the three variations of our algorithm in order to assess their performance. The selection of an appropriate root vertex plays a crucial role in constructing a balanced tree T with a maximum distance of $2l - 2$ between any

two vertices. To establish criteria for the selection of the root vertex, we explored the concepts of complex network measures. In Section 2, we examine the theoretical foundations and concepts employed in Centrality Heuristic to effectively determine the optimal root vertex. Our objective is to establish an understanding of the complex network measures and principles that underpin the root vertex selection process. In Algorithm 5 we present Centrality Heuristic (H4) and afterwards we discuss the differences among its three versions.

Algorithm 5: Heuristic H4

```
Data: A graph
Result: A spanner tree
Initialization;
From graph, list all vertices;
Compute degree for all vertices ;        // used only H4v1
Compute closeness for all vertices;
Compute leverage for all vertices;
Select the greatest vertex to be root;
Add to the tree, the selected vertex (root) and your neighbors that
  are not in tree (the father and your children);
List all neighbors inserted in tree;
while spanner tree not created do
    Sort the list of neighbors from vertices quality;
    From the list of neighbors add to the tree the valid
      neighbors(vertices) from list and your valid neighbors (the
      children of the children);
    Clear the list of neighbors;
    Update the list of neighbors with the last neighbors added to
      the tree in this round;
end
```

Centrality Heuristic MaxDegree (H4v1) The Centrality Heuristic MaxDegree utilizes the concept of degree centrality to select the vertex root. The vertex with the highest degree is chosen as the root, but in cases where multiple vertices have the same degree, ties are broken using a combination of closeness centrality and leverage centrality. Subsequently, the neighbors are sorted based on their degree centrality, enabling a systematic analysis of the network structure.

Traveller Centrality Heuristic (H4v2r1) The Traveller Centrality Heuristic demonstrates higher accuracy compared to Centrality Heuristic MaxDegree and Algebraic Centrality Heuristic, but at the cost of slower performance. This can be attributed to the calculation of closeness centrality, which necessitates traversing all vertices to determine the shortest path between a given vertex and all others, as described in Equation 2.

Algebraic Centrality Heuristic (H4v2r3) Note, the Centrality Heuristic (H4v2r2) not was introduce because it was intentionally abandoned in favor of the improved the Algebraic Centrality Heuristic. The algorithm for Algebraic Centrality Heuristic is essentially the same as Traveller Centrality Heuristic. While the accuracy of Algebraic Centrality Heuristic is slightly lower than that of Traveller Centrality Heuristic, it exhibits higher speed due to the adoption of an approximation method proposed by [Chen e Evans, 2022]. Instead of traversing all vertices, we employ Equations 3 and 4 presented in their study to estimate closeness centrality.

4. Experiments and results

4.1. Experimental setup

In this Section, we conduct extensive testing of our algorithms implemented in C++11 on a hardware system comprising 12x Intel Core i7-8700 CPUs @3.20GHz, 32GiB RAM, and a 66GiB

HDD. The testing environment involved virtualization on a common KVM processor with 4GiB of RAM, utilizing 1 socket and 2 cores. The experiments were performed on the Linux 5.15.0-67-generic operating system, version '#74-Ubuntu SMP Wed Feb 22 14:14:39 UTC 2023', running on an x86_64 machine. During this evaluation, we executed the heuristics independently to assess their quality. The instances were run individually, ensuring accurate measurements and eliminating any potential interference or dependencies between the test cases. All the algorithms were implemented in C++11, ensuring consistency and comparability in the experimental results. The C++ code is available in the GitHub repository [Santos, 2023].

4.2. Quality Analysis

We present a quality analysis of 11 graphs ranging from 10 to 20 vertices. Additionally, we demonstrate the analysis for four graph classes (Barabási-Albert, Erdös-Rényi, Watts-Strogatz, and Bipartite graphs), using graphs with a varying number of vertices ranging from 100 to 1000. We employ a sequential algorithm [Couto et al., 2022, p. 2] to compute the stretch index for random graphs with vertices ranging from 10 to 20 vertices. Subsequently, we evaluate the quality of our proposed heuristics by computing the stretch index using these methods. The obtained results are presented in Table 2, where the brute-force algorithm outperforms the proposed heuristics are highlighted(red).

Table 2: The values in each cell represents the difference between the true value found by sequential brute-force algorithm and the value found by heuristics. [0 means a tie, in any other case the brute force result is better].

Type / n	H1v1	H1v2	H2v1	H2v2	H3v1	H3v2	H4v1	H4v2r1	H4v2r3
10	**0.0**	**0.0**	**1.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
11	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
12	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
13	**0.0**	**0.0**	**0.0**	**1.0**	**1.0**	**1.0**	**0.0**	**0.0**	**0.0**
14	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
15	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
16	**0.0**	**1.0**	**0.0**	**1.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
17	**0.0**	**0.0**	**0.0**	**0.0**	**3.0**	**3.0**	**0.0**	**1.0**	**0.0**
18	**2.0**	**1.0**	**1.0**	**1.0**	**2.0**	**1.0**	**0.0**	**0.0**	**0.0**
19	**0.0**	**0.0**	**1.0**	**0.0**	**1.0**	**1.0**	**0.0**	**0.0**	**0.0**
20	**1.0**	**1.0**	**2.0**	**1.0**	**2.0**	**2.0**	**0.0**	**0.0**	**0.0**

Next, we present Tables 3-11 that provide a quality analysis to our heuristics using four distinct graph classes: Barabási-Albert, Erdös-Rényi, Watts-Strogatz, and Bipartite. Calculating the real stretch index for these graphs using a sequential algorithm can be computationally challenging. However, we employ a lower bound computation to assess how closely our heuristics approximate the stretch index. In the header of table 2, the variable n represents the number of vertices in each graph, while $A_v(m)$ denotes the average number of edges present in graphs of that specific class. Within each cell of the graph class, the lower value indicates the average difference between the output of the heuristic and the lower bound of the smallest-e-cycle (see Section 1), whereas the upper value denotes the corresponding standard deviation in relation to lower bound.

Table 3: Analysis of H1v1 for the presented graphs classes. In each cell, the bottom(the top) values is the average (the standard deviation) between the difference of the returned value of H1v1 and the smallest-e-cycle lower bound.

Class \ n / $Av(m)$	100.0 1.9k	200.0 7.6k	300.0 17.4k	400.0 30.5k	500.0 46.9k	600.0 68.8k	700.0 93.6k	800.0 122.3k	900.0 154.8k	1000.0 191.1k
Barabasi	**0.3** 1.9	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0
Erdos	**0.70** 2.9	**0.7** 3.1	**0.67** 3.5	**0.74** 3.8	**0.64** 4.3	**1.18** 4.0	**0.87** 3.8	**0.67** 3.5	**0.74** 4.2	**0.78** 4.3
Watts	**0.8** 3.6	**0.89** 5.0	**0.63** 5.0	**0.6** 5.2	**0.74** 4.8	**0.53** 4.9	**0.79** 5.4	**1.11** 5.4	**1.04** 5.1	**0.94** 4.9
Bipartite	**1.28** 5.4	**1.66** 6.2	**1.26** 6.0	**1.07** 6.2	**1.6** 7.2	**1.28** 7.4	**1.83** 7.2	**1.74** 7.6	**1.2** 6.4	**1.61** 7.0

Table 4: Analysis of H1v2 for the presented graphs classes. In each cell, the bottom(the top) values is the average (the standard deviation) between the difference of the returned value of H1v2 and the smallest-e-cycle lower bound.

Class \ n / $Av(m)$	100.0 1.9k	200.0 7.6k	300.0 17.4k	400.0 30.5k	500.0 46.9k	600.0 68.8k	700.0 93.6k	800.0 122.3k	900.0 154.8k	1000.0 191.1k
Barabasi	**0.3** 1.9	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0
Erdos	**0.39** 2.8	**0.6** 3.2	**0.78** 3.3	**0.74** 3.8	**0.77** 4.0	**1.2** 4.4	**0.87** 3.8	**0.91** 3.6	**0.64** 4.3	**0.78** 4.3
Watts	**0.8** 3.6	**1.07** 4.8	**1.04** 5.1	**0.6** 4.8	**1.07** 5.2	**1.00** 5.3	**1.07** 5.2	**1.20** 5.5	**1.1** 5.3	**0.63** 5.0
Bipartite	**1.32** 4.8	**1.49** 5.6	**1.2** 5.6	**0.97** 6.8	**1.61** 7.0	**1.32** 6.8	**2.03** 7.2	**2.0** 8.0	**1.28** 6.6	**1.83** 7.2

Table 5: Analysis of H2v1 for the presented graphs classes. In each cell, the bottom(the top) values is the average (the standard deviation) between the difference of the returned value of H2v1 and the smallest-e-cycle lower bound.

Class \ n / $Av(m)$	100.0 1.9k	200.0 7.6k	300.0 17.4k	400.0 30.5k	500.0 46.9k	600.0 68.8k	700.0 93.6k	800.0 122.3k	900.0 154.8k	1000.0 191.1k
Barabasi	**0.45** 1.3	**0.3** 1.9	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.3** 2.1	**0.3** 2.1	**0.39** 2.2	**0.48** 2.4
Erdos	**0.0** 3.0	**0.77** 3.0	**0.5** 3.5	**0.48** 3.4	**0.87** 3.2	**0.66** 3.4	**0.45** 3.7	**0.53** 4.1	**0.39** 4.2	**0.77** 4.0
Watts	**0.6** 3.2	**0.91** 3.4	**0.53** 3.9	**0.53** 4.1	**0.53** 3.9	**0.48** 4.6	**0.92** 4.5	**0.53** 4.1	**0.45** 4.7	**0.48** 4.4
Bipartite	**0.91** 4.6	**1.07** 5.8	**1.74** 5.6	**1.2** 5.6	**1.49** 5.6	**1.49** 6.4	**0.89** 6.0	**1.40** 5.8	**1.34** 5.0	**1.56** 7.4

Table 6: Analysis of H2v2 for the presented graphs classes. In each cell, the bottom(the top) values is the average (the standard deviation) between the difference of the returned value of H2v2 and the smallest-e-cycle lower bound.

Class \ n / $Av(m)$	100.0 1.9k	200.0 7.6k	300.0 17.4k	400.0 30.5k	500.0 46.9k	600.0 68.8k	700.0 93.6k	800.0 122.3k	900.0 154.8k	1000.0 191.1k
Barabasi	**0.45** 1.3	**0.3** 1.9	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0
Erdos	**0.48** 2.4	**0.67** 2.5	**0.3** 2.9	**0.53** 2.9	**0.83** 2.9	**0.8** 3.4	**0.66** 3.4	**0.39** 3.2	**0.6** 3.2	**0.6** 3.2
Watts	**0.63** 3.0	**0.6** 3.2	**0.66** 3.6	**0.63** 3.0	**0.48** 3.6	**0.78** 3.7	**0.64** 3.7	**0.8** 3.4	**0.66** 3.4	**0.64** 3.7
Bipartite	**0.79** 4.4	**1.49** 4.4	**0.97** 4.8	**1.2** 4.4	**0.91** 4.6	**0.97** 4.8	**1.0** 5.0	**1.32** 5.2	**0.91** 4.6	**1.28** 5.4

Table 7: Analysis of H3v1 for the presented graphs classes. In each cell, the bottom(the top) values is the average (the standard deviation) between the difference of the returned value of H3v1 and the smallest-e-cycle lower bound.

Class \ n / $Av(m)$	100.0 1.9k	200.0 7.6k	300.0 17.4k	400.0 30.5k	500.0 46.9k	600.0 68.8k	700.0 93.6k	800.0 122.3k	900.0 154.8k	1000.0 191.1k
Barabasi	**0.78** 2.7	**0.45** 3.7	**0.80** 3.6	**0.48** 3.6	**0.0** 4.0	**0.6** 3.8	**0.0** 4.0	**0.0** 4.0	**0.3** 4.1	**0.0** 4.0
Erdos	**1.40** 4.2	**1.3** 4.9	**0.66** 4.4	**0.79** 5.6	**1.32** 5.2	**0.94** 4.9	**1.88** 5.8	**1.13** 4.9	**1.41** 5.3	**1.61** 5.0
Watts	**1.55** 5.7	**1.32** 5.8	**1.5** 5.5	**2.15** 6.4	**1.90** 6.3	**1.88** 7.2	**1.34** 6.0	**1.56** 6.6	**1.74** 6.4	**2.00** 6.6
Bipartite	**2.15** 5.6	**1.88** 6.2	**2.37** 6.6	**1.28** 7.4	**1.74** 7.6	**1.66** 8.2	**1.4** 7.8	**2.44** 8.2	**2.56** 9.0	**1.74** 8.4

Table 8: Analysis of H3v2 for the presented graphs classes. In each cell, the bottom(the top) values is the average (the standard deviation) between the difference of the returned value of H3v2 and the smallest-e-cycle lower bound.

Class \ n / $Av(m)$	100.0 1.9k	200.0 7.6k	300.0 17.4k	400.0 30.5k	500.0 46.9k	600.0 68.8k	700.0 93.6k	800.0 122.3k	900.0 154.8k	1000.0 191.1k
Barabasi	**0.78** 2.7	**0.5** 3.5	**0.80** 3.6	**0.45** 3.7	**0.0** 4.0	**0.6** 3.8	**0.0** 4.0	**0.0** 4.0	**0.3** 4.1	**0.0** 4.0
Erdos	**1.34** 4.3	**1.3** 5.1	**0.66** 4.4	**1.0** 5.0	**0.0** 0.0	**0.94** 5.1	**1.28** 5.4	**1.13** 4.9	**1.79** 5.7	**1.49** 5.6
Watts	**1.26** 5.3	**1.28** 6.4	**1.49** 5.4	**1.40** 6.2	**1.54** 6.0	**1.55** 7.3	**1.85** 6.5	**1.74** 6.4	**1.5** 6.5	**2.15** 6.4
Bipartite	**0.0** 0.0	**1.78** 6.0	**0.0** 0.0	**1.0** 7.0	**1.56** 7.4	**1.95** 8.4	**1.28** 8.6	**2.2** 7.4	**1.2** 8.4	**1.66** 8.2

Table 9: Analysis of H4v1 for the presented graphs classes. In each cell, the bottom(the top) values is the average (the standard deviation) between the difference of the returned value of H4v1 and the smallest-e-cycle lower bound.

Class \ n / $Av(m)$	100.0 1.9k	200.0 7.6k	300.0 17.4k	400.0 30.5k	500.0 46.9k	600.0 68.8k	700.0 93.6k	800.0 122.3k	900.0 154.8k	1000.0 191.1k
Barabasi	**0.3** 1.9	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0
Erdos	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 0.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0
Watts	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0
Bipartite	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0

Table 10: Analysis of H4v2r1 for the presented graphs classes. In each cell, the bottom(the top) values is the average (the standard deviation) between the difference of the returned value of H4v2r1 and the smallest-e-cycle lower bound.

Class \ n / $Av(m)$	100.0 1.9k	200.0 7.6k	300.0 17.4k	400.0 30.5k	500.0 46.9k	600.0 68.8k	700.0 93.6k	800.0 122.3k	900.0 154.8k	1000.0 191.1k
Barabasi	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0
Erdos	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0
Watts	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0
Bipartite	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0

Table 11: Analysis of H4v2r3 for the presented graphs classes. In each cell, the bottom(the top) values is the average (the standard deviation) between the difference of the returned value of H4v2r3 and the smallest-e-cycle lower bound.

Class \ n / $Av(m)$	100.0 1.9k	200.0 7.6k	300.0 17.4k	400.0 30.5k	500.0 46.9k	600.0 68.8k	700.0 93.6k	800.0 122.3k	900.0 154.8k	1000.0 191.1k
Barabasi	**0.3** 1.9	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0
Erdos	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0
Watts	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0
Bipartite	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0	**0.0** 2.0

Based in the information summarized in Tables 3-11, Centrality Heuristic (H4) has better results among other proposed heuristics. As we progress and evolve the heuristics, we observe a substancial enhancement in the values of the stretch index. Initially, the heuristics produced results with suboptimal quality, however, when refine and fine-tune the algorithms, incorporating more sophisticated techniques and leveraging insights from previous iterations we witness a improvement in their quality.

5. Concluding remarks

In this work, we have developed 4 new efficient heuristics for generating spanning trees to address the t-admissibility problem. The experimental results have demonstrated that our heuristics achieve high efficiency in creating spanning trees to determine the stretch index, with a combination of result quality and good performance. For future research, our aim is to further expand our approaches by incorporating additional centrality methods to determine the root vertex and its usage at each iteration. Furthermore, we seek to develop new algorithms that can efficiently compute the stretch index, enabling a comprehensive comparison with the results obtained from our heuristics. Ongoing research is being conducted to explore the implications of considering the induced cycle on the stretch index calculation in parallel algorithms. Overall, this study highlights the effectiveness and potential of our heuristics in generating spanning trees for the t-admissibility problem. The future directions outlined herein will contribute to obtain optimum tree t-spanners in generally.

References

Cai, L. e Corneil, D. G. (1995). Tree spanners. *SIAM Journal on Discrete Mathematics*, 8(3): 359–387.

Chartrand, G. e Zhang, P. (2012). *A First Course in Graph Theory*. Dover books on mathematics. Dover Publications. ISBN 9780486483689.

Chen, B. e Evans, T. S. (2022). Linking the network centrality measures closeness and degree. *Communications Physics*, 5(1).

Couto, F., Cunha, L. F. I., Juventude, D., e Santiago, L. (2022). Strategies for generating tree spanners: Algorithms, heuristics and optimal graph classes. *Information Processing Letters*, 177: 106265. ISSN 0020-0190.

Das, K., Samanta, S., e Pal, M. (2018). Study on centrality measures in social networks: a survey. *Social Network Analysis and Mining*, 8.

Golbeck, J. (2015). *Introduction to Social Media Investigation: A Hands-on Approach*. Elsevier Science. ISBN 9780128018026.

Joyce, K. E., Laurienti, P. J., Burdette, J. H., e Hayasaka, S. (2010). A new measure of centrality for brain networks. *PLOS ONE*, 5(8):1–13.

Santos, C. T. (2023). t-admissibility. `https://github.com/cthadeusantos/spanner_tree_generator`.

Scott, J. P. e Carrington, P. J. (2011). *The SAGE Handbook of Social Network Analysis*. Sage Publications Ltd. ISBN 1847873952.

West, D. B. et al. (1996). *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River.

Zhuge, H. e Zhang, J. (2009). Topological centrality and its applications.