Theory and Applications of Graphs

Volume 3 | Issue 1 Article 1

2016

An Edge-Swap Heuristic for Finding Dense Spanning Trees

Mustafa Ozen
Bogazici University, mustafa.ozen@boun.edu.tr

Hua Wang
Georgia Southern University, hwang@georgiasouthern.edu

Kai Wang
Georgia Southern University, kwang@georgiasouthern.edu

Demet Yalman
Bogazici University, demet.yalman@boun.edu.tr

An Edge-Swap Heuristic for Finding Dense Spanning Trees

Cover Page Footnote
This work was partially supported by grants from the Simons Foundation (\#245307).

Abstract

Finding spanning trees under various restrictions has been an interesting question to researchers. A "dense" tree, from a graph theoretical point of view, has small total distances between vertices and large number of substructures. In this note, the "density" of a spanning tree is conveniently measured by the weight of a tree (defined as the sum of products of adjacent vertex degrees). By utilizing established conditions and relations between trees with the minimum total distance or maximum number of subtrees, an edge-swap heuristic for generating "dense" spanning trees is presented. Computational results are presented for randomly generated graphs and specific examples from applications.

1 Introduction

Given an undirected graph G with vertex set V and edge set E, a subtree of G is a connected acyclic subgraph of G. A subtree with vertex set V is a *spanning tree* of G. Finding spanning trees (under various restrictions) of a given graph is of importance in many applications such as Information Technology and Network Design.

Many questions have been studied in this aspect, including, but not limited to, the well-known minimum-weight spanning tree problem (MSTP), spanning trees with bounded degree, with bounded number of leaves, or with bounded number of branch vertices. The goal in such studies is usually to find efficient algorithms to produce the desired subgraphs. Recently an edge-swap heuristic for generating spanning trees with minimum number of branch vertices was presented [7], where an efficient algorithm resulted from iteratively reducing the number of branch vertices from a random spanning tree by swapping tree edges with edges not currently in the tree.

A tree of given number of vertices is considered "dense" if the number of substructures (including isomorphic subgraphs) is large or the total distance between vertices is small. In applications, structures generated from such spanning trees are preferred as they have more choices of sub-networks and allow more efficient transfer of resources with minimum cost. In this note we discuss an edge-swap heuristic, inspired by similar work presented in [7], for finding dense spanning trees.

2 Preliminaries

The number of subtrees and the total distance of a tree belong to a group of graph invariants, called topological indices, that are used in the literature as effective descriptors of graph structures. For instance:

- the sum of distances between all pairs of vertices, also known as the *Wiener index* [11], is one of the most well known distance-based index in chemical graph theory;

- the number of subtrees is an example of counting-based indices introduced from pure-mathematical point of view [8] and applications in phylogeny [2].

These two indices have been extensively studied. In particular, it is well known that the star minimizes the Wiener index and maximizes the number of subtrees while the path maximizes the Wiener index and minimizes the number of subtrees. More interestingly, among tress of given degree sequence, the *greedy tree* (Definition 1) was shown to minimize the Wiener index [6, 9, 12] and maximize the number of subtrees [13], where the degree sequence is simply the nonincreasing sequence of vertex degrees.

Definition 1 (Greedy trees)**.** *Given a degree sequence, the greedy tree is achieved through the following "greedy" algorithm:*

i) Start with a single vertex $v = v_1$ as the root and give v the appropriate number of neighbors so that it has the largest degree;

ii) Label the neighbors of v as v_2, v_3, ..., assign to them the largest available degrees such that $deg(v_2) \geq deg(v_3) \geq \cdots$;

iii) Label the neighbors of v_2 (except v) as v_{21}, v_{22}, ... such that they take all the largest degrees available and that $deg(v_{21}) \geq deg(v_{22}) \geq \cdots$, then do the same for v_3, v_4, ...;

iv) Repeat (iii) for all the newly labeled vertices, always start with the neighbors of the labeled vertex with largest degree whose neighbors are not labeled yet.

For example, Fig. 1 shows a greedy tree with degree sequence

$$(4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 2, 2, 1, \ldots, 1).$$

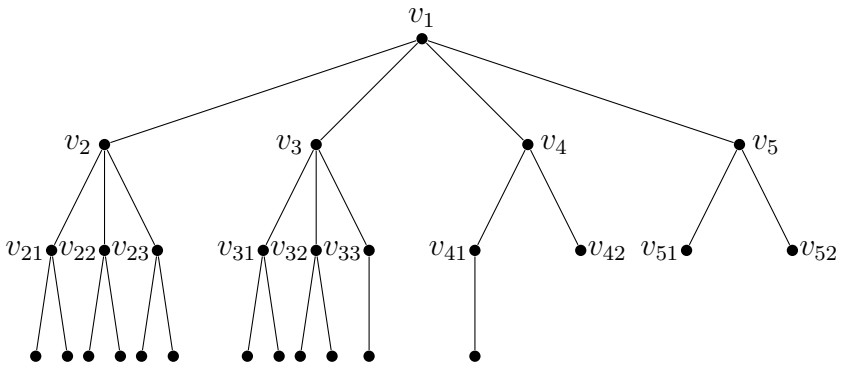

Figure 1: A greedy tree.

Interestingly, the greedy trees are also extremal with respect to many other graph indices, among which is the following special case of the *Randić index* [3], also called the *weight* of a tree [4]:

$$R(T) = \sum_{uv \in E(T)} deg(u)deg(v).$$

A comprehensive discussion of the extremal trees of given degree sequences with respect to functions defined on adjacent vertex degrees can be found in [10].

For trees of different given degree sequences, much work has been done in comparing the greedy trees (of the same order) of different degree sequence. In particular, for two

DOI: 10.20429/tag.2016.030101

nonincreasing sequences $\pi' = (d'_1, \cdots, d'_n)$ and $\pi'' = (d''_1, \cdots, d''_n)$, π'' is said to *majorize* π' if for $k = 1, \cdots, n-1$

$$\sum_{i=0}^{k} d'_i \leq \sum_{i=0}^{k} d''_i \qquad \text{and} \qquad \sum_{i=0}^{n} d'_i = \sum_{i=0}^{n} d''_i.$$

The concept of majorization has been applied to the comparison of greedy trees of different degree sequences in order to find the dense structures (with minimal total distance or maximal number of subtrees) under various constraints. See [13] for an example of such discussions. For convenience we also say that π'' is "higher in the majorization ladder" than π' if π'' majorizes π' and $\pi'' \neq \pi'$.

To find dense spanning trees, our edge-swap heuristic starts with a random spanning tree. We then continuously remove a "bad" edge and add a "good" edge in order to improve the density of the spanning tree. From the perspective of distance-based and structure-based graph indices, evaluating the corresponding index of the resulted tree at each step would be extremely time consuming.

We propose an edge-swap heuristic that is based on the above results and use $R(T)$ instead of the distance or number of subtrees as an effective measure. In every step, we consider the degrees of the end vertices of the edge to be removed or added, as well as the resulted change in $R(T)$. Such a strategy simultaneously optimizes the value of the $R(T)$ and improves the degree sequence in the ladder of majorization. The consideration of $R(T)$ results in an efficient algorithm that quickly finds a dense spanning tree, which we present in the next section. Computational results will be provided for both randomly generated graphs and specific examples from applications. We also comment on improvements of the final result with the degree sequences taken into account.

3 The edge-swap heuristic

In this section we present an edge-swap heuristic in detail. The following algorithm takes a graph $G = (V, E)$ as input and returns a dense spanning tree T as output.

Step 1.
Input $G(V, E)$ and generate a random spanning tree T for G. Let *SPARSE* be "true".

Step 2.

Step 2-1: Find the candidate edge e to be removed from T.
For each edge $e = uv \in E(T)$, let

$$f(e) = d_u d_v + \sum_{i=1}^{d_u - 1} d_{u_i} + \sum_{i=1}^{d_v - 1} d_{v_i}$$

where d_u and d_v are the degrees of the vertices u and v respectively (in T), d_{u_i} for $1 \leq i \leq d_u - 1$ (d_{v_i} for $1 \leq j \leq d_v - 1$) are the degrees of the other neighbors of u (v) in T.

Let e be an edge with the minimum $f(.)$ value.

Step 2-2: Generate the spanning forest $T' = T - e$ with two components T_u and T_v.

Step 2-3: Find the candidate edge e'' (with end vertices in T_u and T_v respectively) to be added to T.

For each edge $e' = u'v' \in E(G)$ with $u' \in T_u$ and $v' \in T_v$, let

$$g(e') = (d_{u'} + 1)(d_{v'} + 1) + \sum_{i=1}^{d_{u'}} d_{u'_i} + \sum_{i=1}^{d_{v'}} d_{v'_i}$$

where $d_{u'}$ and $d_{v'}$ are the degrees of the vertices u' and v' respectively (in T'), $d_{u'_i}$ for $1 \leq i \leq d_{u'}$ ($d_{v'_i}$ for $1 \leq j \leq d_{v'}$) are the degrees of the neighbors of u' (v') in T'.

Let e'' be such an edge with the maximum $g(.)$ value.

Step 2-4: Generate the spanning tree $T'' = T' + e''$.

Step 2-5: If $f(e) < g(e'')$, let *SPARSE* be "true". Otherwise let *SPARSE* be "false".

Step 3.

While *SPARSE* is "true", let $T = T''$ and repeat Step 2. Return T when *SPARSE* is "false".

Figure 2 and Figure 3 present a step by step illustration (left $\rightarrow$ right and top $\rightarrow$ bottom) of the algorithm, where the spanning trees in each step is shown in bold face and the removed edge in each step is shown with a dotted line.

Remark 1. *In the above algorithm, the value*

$$g(e'') - f(e) = R(T'') - R(T)$$

is the maximum possible improvement in $R(.)$ over one swap. In the case of a tie (i.e., multiple edges can serve as e or e''), we simply pick one of them. Since after each swap, the value of $R(T)$ is strictly increasing, this process terminates after finitely many steps.

4 Computational results

Of course, the heuristic proposed in the previous section does not guarantee the densest spanning tree as an output. But as experimental results show, this heuristic effectively finds a dense spanning tree within very few swaps and hence is of great practical interests. When tested on 100 randomly generated graphs, each of order 15 and containing a spanning star, the algorithm returns a star in over 60 runs. Part of this data (that is representative of the general performance) is shown in Table 1.

Note that a star on 15 vertices has total distance 196. This is attainable for all graphs considered above. As shown in Table 1, all resulting spanning trees are dense (even if it is not a star) with only one exception, the graph "D".

In the example shown in Figure 4, 7 edge-swaps resulted in the final spanning tree from the original graph on 15 vertices and 37 edges.

When applied to the US Airports data set of 332 vertices and 2126 edges [5], only 15 edge-swaps were needed to obtain the final spanning tree. In this case, total distance of the tree is reduced from 1444880 to 1421327, a reduction of 23553.

DOI: 10.20429/tag.2016.030101

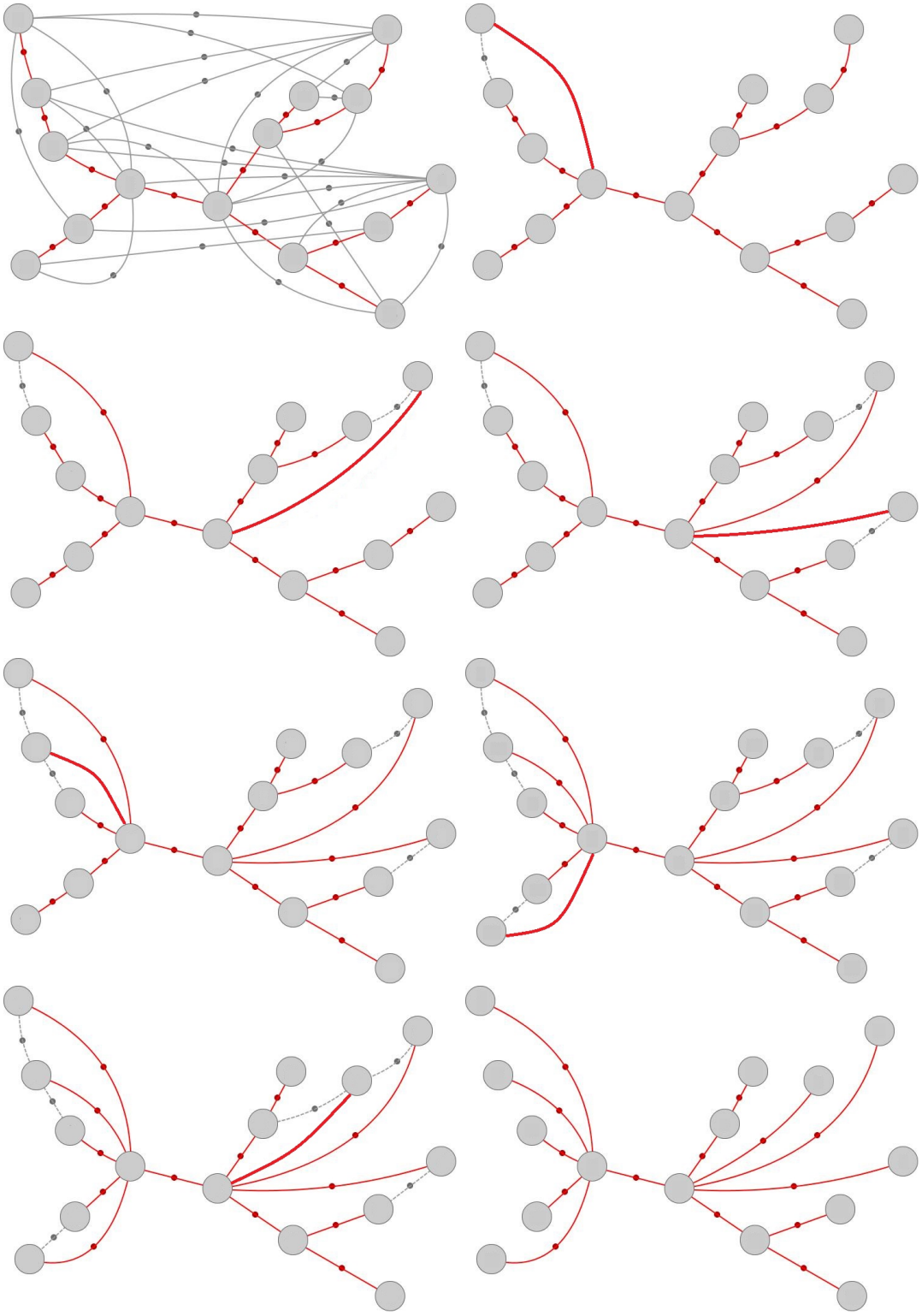

Figure 2: Step by step illustration of the algorithm

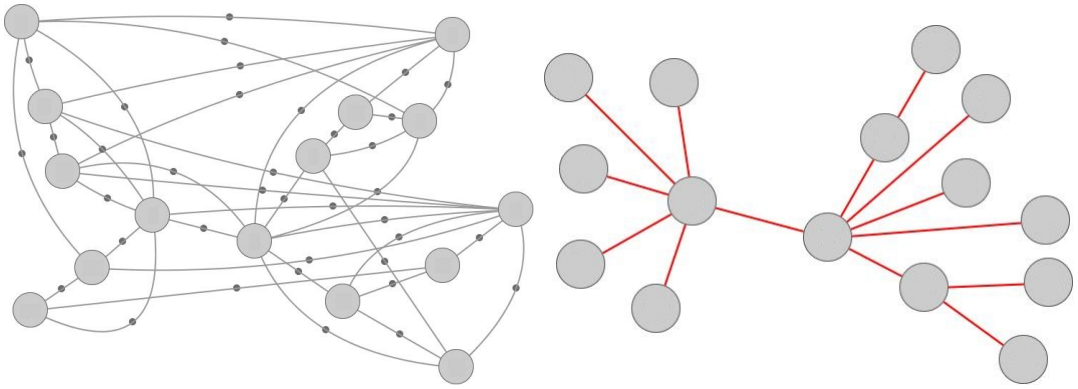

Figure 3: The original graph (on the left) and the resulted spanning tree (on the right)

Graphs	Number of swaps	Initial distance	Final distance	Returns a star
A	9	386	238	N
B	15	384	196	Y
C	10	404	196	Y
D	0	348	348	N
E	10	432	196	Y
F	10	382	196	Y
G	8	374	232	N
H	13	348	196	Y
I	16	382	196	Y
J	10	382	196	Y
Average:	10.1	382.2	219	-

Table 1: Results of ten randomly generated graphs on 15 vertices

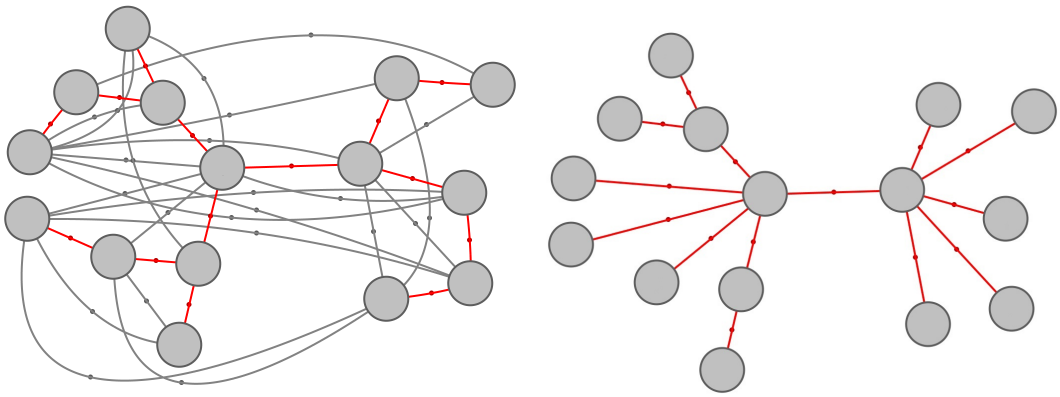

Figure 4: The original graph with first spanning tree (on the left) and the resulted spanning tree (on the right)

DOI: 10.20429/tag.2016.030101

5 Further improvements

In earlier sections we provided a simple and efficient edge-swap heuristic to find dense spanning trees. Computational results are also presented and analyzed. A simple way of improving the likelihood of achieving denser spanning trees can be obtained by replacing Step 2-5 of the algorithm with the following:

(Step 2-5)′: If $f(e) < g(e'')$ or $f(e) = g(e'')$ and the degree sequence of T'' is higher in the majorization ladder than that of T, let *SPARSE* be "true". Otherwise let *SPARSE* be "false".

In this case, after each swap, the value of $R(T)$ is strictly increasing or nondecreasing with the degree sequence moving up in the majorization ladder. Take, for instance, two of the randomly generated graphs on 15 vertices as discussed in the previous section, Figures 5 and 6 show improvements in the resulted spanning tree.

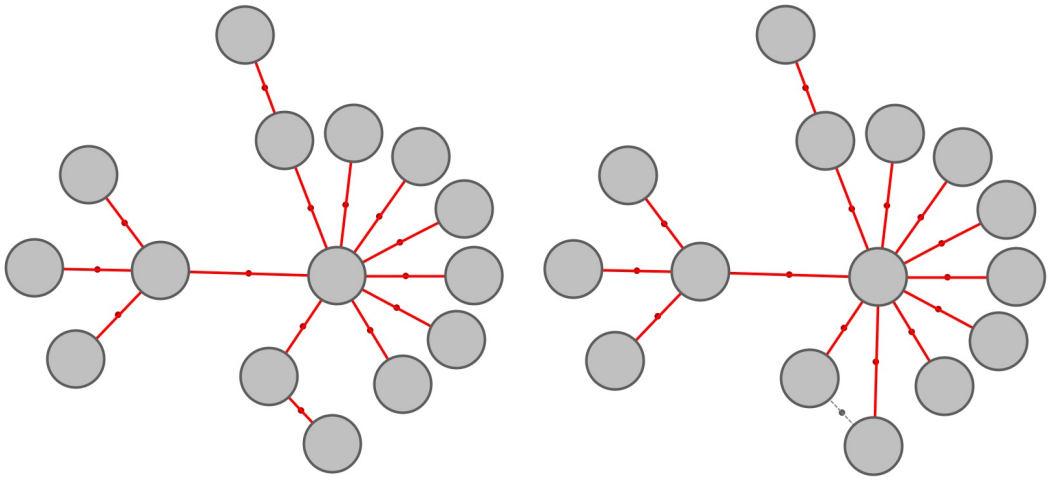

Figure 5: The resulted spanning trees from the original algorithm (left) and the modified algorithm (right)

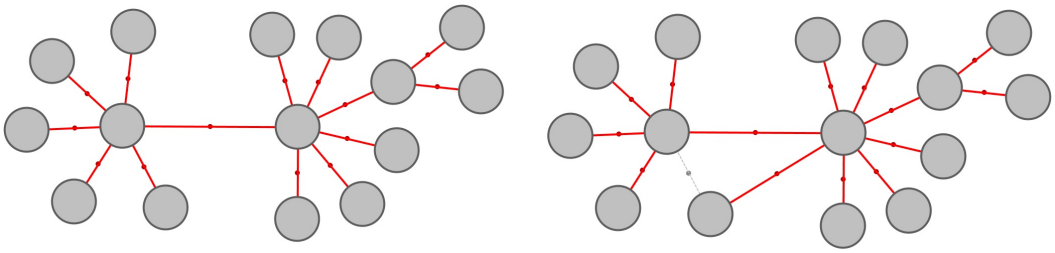

Figure 6: The resulted spanning trees from the original algorithm (left) and the modified algorithm (right)

When applying the modified algorithm to the US Airports data set, the improvement over the original algorithm is shown in Table 2.

For completeness, we also describe the pseudo-code (for the modified algorithm) in Algorithm 1.

	Number of swaps	Total distance	Decrease in distance
Algorithm	15	1421327	23553
Modified algorithm	23	1412038	32842

Table 2: Comparison of the algorithms with the US Airport data set

Data: $G = (V, E)$
Result: Updated tree T
Load data set: $G \leftarrow$ data set;
$T \leftarrow \text{MST}(G)$;
sparse $\leftarrow$ true;
while sparse is true **do**
  $(L\_remove, \min f(e)) \leftarrow$ findRemovalEdges(T);
  Select removal edge: $e \leftarrow (u, v)$;
  $T\_r \leftarrow T \setminus (u, v)$;
  $(neighbors\_u, neighbors\_v) \leftarrow \text{split}(T\_r, (u, v))$;
  $(L\_add, \max g(e)) \leftarrow$ findInsertionEdges$(G, T_r, neighbors_u, neighbors_v)$
  Select insertion edge: $e'' \leftarrow (u'', v'')$;
  $T\_a \leftarrow T\_r \cup (u'', v'')$;
  if min $f(e) <$ max $g(e'')$ **then**
    $T \leftarrow T\_a$
  else if min $f(e) ==$ max $g(e'')$ **then**
    if degree sequence of T_a majorizes degree sequences of T **then**
      T $\leftarrow T\_a$;
    else
      sparse $\leftarrow$ false;
    end
  else
    sparse $\leftarrow$ false;
  end
end

Algorithm 1: Pseudo-code for the modified edge-swap heuristic

The algorithm starts with loading data set in the format of a $n \times 3$ matrix. The first two columns represent an edge with two vertices and last column shows weights between these vertices. A minimum weighted spanning tree T is then generated through the Kruskal algorithm. Each iteration of edge-swapping includes removing a "*bad*" edge and adding a "*good*" edge which is not in the current tree. Function "$findRemovalEdges$" takes adjacency matrix of the tree T as an input and returns list of the candidate edges with the minimum value of $f(.)$. From candidate list, one of the edges, $e = (u, v)$ is chosen to be removed. After removing the edge from the adjacency matrix of the tree T, we obtain "T_r" as the

DOI: 10.20429/tag.2016.030101

updated graph. Function "*split*" is used to split the adjacency matrices of new two subtrees up and return the lists of vertices in each component. Function "$findInsertionEdges$" takes the lists, adjacency matrices of the updated graph and the original graph as inputs. After calculating $g(.)$ for each candidate edge, it returns minimum $g(.)$ and list of corresponding edges. One of the candidate edges $e'' = (u'', v'')$ is chosen to be inserted and the updated tree "T_a" is obtained. If $f(e) < g(e'')$, the new tree is denser than previous and the process continues. If $f(e) = g(e'')$, the degree sequences of the initial tree and current tree are calculated. If the degree sequence of the current tree "T_a" majorizes that of T (and the two degree sequences are not the same), then an edge-swap is made; otherwise process is terminated.

6 Complexity Analysis

For the original question of finding a spanning tree that maximizes the number of subtrees or minimizes the total distance, the complexity appears to be difficult to determine. To our best knowledge, the complexity of this problem is not yet determined. However, given that "dense" trees usually have large number of leaves and the problem of finding spanning trees with the most leaves is NP-hard [1], it is natural to guess that finding "dense" spanning trees is also hard.

On the other hand, the complexity can be easily deduced for our algorithm described in Section 3. It is easy to see that it takes $O(n)$ time to find an edge to remove in Step 2-1. The total number of edges in G is at most $\binom{n}{2}$, implying that Step 2-3 takes $O(n^2)$ time to complete. Since each iteration the original algorithm strictly increase the value of $R(.)$ and the largest possible $R(T)$ on an n-vertex tree is achieved by $R(K_{1,n-1}) = (n-1)^2$, there are at most $O(n^2)$ iterations. Thus the complexity of the original algorithm is $O(n^4)$. Note that this is just worst case analysis. In practice the algorithm usually runs much faster.

As shown in the previous section, the results can be improved by implementing a slightly different Step 2-5 as shown in the modified algorithm (Algorithm 1). Note that each degree sequence of a tree on n vertices is equivalent to (after removing 1 from each degree and only keeping non-zero entries) a non-increasing sequence of positive integers that sum up to $2n-2$. For example, a degree sequence $(5, 4, 3, 3, 3, 2, 1, \ldots, 1)$ of a tree on 16 vertices is mapped to $(4, 3, 2, 2, 2, 1)$ with $4+3+2+2+2+1 = 14$. Thus the number of all possible degree sequences of a spanning tree of a graph on n vertices is the same as the number of integer partitions of $n-2$, which is exponential. This implies that the worst case scenario of the modified algorithm could have its number of iterations being exponential. However, as shown in Table 2 for the US Airport data set and experimentation with random graphs, the modified algorithm generally terminates much faster.

References

[1] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (1979) *W. H. Freeman & Co., New York, NY, USA*

[2] B. Knudsen, Optimal multiple parsimony alignment with affine gap cost using a phylogenetic tree, *Lecture Notes in Bioinformatics* 2812, Springer Verlag, 2003, 433–446.

[3] M. Randić, On characterization of molecular branching, *J. Amer. Chem. Soc.* 97 (1975) 6609–6615.

[4] D. Rautenbach, A note on trees of maximum weight and restricted degrees, *Discrete Math.* 271 (2003) 335–342.

[5] Pajek datasets, *US Air lines*: http://vlado.fmf.uni-lj.si/pub/networks/data/

[6] N. Schmuck, S. Wagner, H. Wang, Greedy trees, caterpillars, and Wiener-type graph invariants, *MATCH Commun.Math.Comput.Chem.* 68(1) (2012), 273–292.

[7] R. Silva, D. Silva, M. Resende, G. Mateus, J. Goncalves, P. Festa, An edge-swap heuristic for generating spanning trees with minimum number of branch vertices, *Optim. Lett.* 8 (2014) 1225–1243.

[8] L.A. Szekely, H. Wang, On subtrees of trees, *Advances in Applied Mathematics* 34 (2005), 138–155.

[9] H. Wang, The extremal values of the Wiener index of a tree with given degree sequence, *Discrete Applied Mathematics,* 156 (2008), 2647–2654.

[10] H. Wang, Functions on adjacent vertex degrees of trees with given degree sequence, *Central European J. Math.* 12 (2014) 1656–1663.

[11] H. Wiener, Structural determination of paraffin boiling point, *J. Amer. Chem. Soc.* 69 (1947), 17–20.

[12] X.-D. Zhang, Q.-Y. Xiang, L.-Q. Xu, R.-Y. Pan, The Wiener index of trees with given degree sequences, *MATCH Commun.Math.Comput.Chem.*, 60 (2008), 623–644.

[13] X.-M. Zhang, X.-D. Zhang, D. Gray, H. Wang, The number of subtrees of trees with given degree sequence, *J. Graph Theory,* 73(3) (2013), 280–295.

DOI: 10.20429/tag.2016.030101