Proceedings of the Federated Conference on Computer Science and Information Systems pp. 271–275
DOI: 10.15439/2020F4
ISSN 2300-5963 ACSIS, Vol. 21

On Finding the Optimal Tree of a Complete Weighted Graph

Seyed Soheil Hosseini, Nick Wormald, Tianhai Tian
School of Mathematics,
Monash University,
Victoria 3800, Australia
Email: {soheil.hosseini, nick.wormald, tianhai.tian}@monash.edu

***Abstract*—We want to find a tree where the path length between any two vertices on this tree is as close as possible to their corresponding distance in the complete weighted graph of vertices upon which the tree is built. We use the residual sum of squares as the optimality criterion to formulate this problem, and use the Cholesky decomposition to solve the system of linear equations to optimize weights of a given tree. We also use two metaheuristics, namely Simulated Annealing (SA) and Iterated Local Search (ILS) to optimize the tree structure. Our results suggest that SA and ILS both perform well at finding the optimal tree structure when the dispersion of distances in the complete graph is large. However, when the dispersion of distances is small, only ILS has a solid performance.**

I. Introduction

WE WANT to find an edge-weighted tree that best estimates the complete weighted graph of distances between vertices such that the discrepancy between the path length between any two vertices in the tree, and their distance in the complete graph, is minimized. To this end, we use the residual sum of squares (RSS) as it is a typical optimality criterion for these types of problems. We call the resulting tree, the residual sum of squares optimal tree (RSSOT). The underlying idea for this problem originates from three areas: stock-correlation networks, phylogenetic trees [1] and t-spanners [2] in graph theory. In the first two areas, several algorithms have been proposed to build a network based on the complete weighted graph of distances between stocks [3]–[10] or species information [11]. In the third area, the problem is similar to estimating the t-spanner tree of K_n.

We take an approach similar to some investigations in phylogenetic trees [11], but we have a different treatment of a basic improvement step used in local search heuristics. Also, in contrast to phylogenetic trees, we consider distances between all vertices of the tree, not just leaves. We investigate two metaheuristics—Simulated Annealing (SA) and Iterated Local Search (ILS)—for this problem.

In Section II, we discuss how to optimize edge weights of a given tree. In Section III, we use the aforementioned metaheuristics to optimize the tree structure—find RSSOT—and ultimately, Sections IV and V include our results and conclusion respectively.

II. Sub-problem: Tree weight optimization

For the complete weighted graph $K_n = (V, \mathbf{E}, d)$, we want to come up with a weighted spanning tree $T = (V, E, w)$ where $E \subset \mathbf{E}$ such that the path length between any two vertices on the tree best estimates the distance between them in K_n. To be precise, we want to minimize the RSS between path lengths in T and their corresponding edge distances in K_n such that

$$RSS\left(T, K_n\right) = \sum_{\substack{m,k \\ m<k}} \left(S\left(P_{m,k}\right) - d_{mk}\right)^2. \tag{1}$$

In the equation above, $P_{m,k}$ denotes the path connecting vertices v_m and v_k, and $S(P_{m,k})$ denotes the sum of edge weights on this path. For example, for the path $P_{m,k} = (e_{ma}, e_{ab}, e_{bc}, \ldots, e_{dk})$, $S(P_{m,k}) = w_{ma} + w_{ab} + w_{bc} + \ldots + w_{dk}$. Thus, equation (1) can be reformulated as

$$RSS(T, K_n) = \sum_{\substack{m,k \\ m<k}} \left(\sum_{\substack{i,j \\ e_{ij} \in P_{m,k}}} w_{ij} - d_{mk} \right)^2. \tag{2}$$

In order to find the edge weights for a given spanning tree, we take the derivative of RSS with respect to the w_{ij}'s, so that $\frac{\partial RSS}{\partial w_{ij}} = 0$. It gives us

$$\begin{aligned} \frac{\partial RSS}{\partial w_{ij}} = 2 \Bigg(& \sum_{\substack{m,k:\, e_{ij} \in P_{m,k} \\ m<k}} \left(w_{ij} + \sum_{\substack{e_{rs} \in P_{m,k} \\ e_{rs} \neq e_{ij}}} w_{rs} \right) \\ & - \sum_{\substack{m,k:\, e_{ij} \in P_{m,k} \\ m<k}} d_{mk} \Bigg) = 0 \qquad \forall e_{ij} \in E. \end{aligned} \tag{3}$$

The equation above can be written as

$$\begin{aligned} \frac{\partial RSS}{\partial w_{ij}} = 2 \Bigg(& \alpha_{ij} w_{ij} + \sum_{e_{rs} \neq e_{ij}} \beta_{rsij} w_{rs} \\ & - \sum_{\substack{m,k:\, e_{ij} \in P_{m,k} \\ m<k}} d_{mk} \Bigg) = 0 \qquad \forall e_{ij} \in E \end{aligned} \tag{4}$$

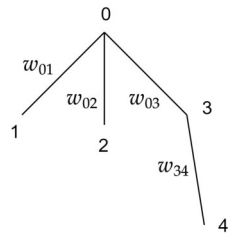


Fig. 1: An example of what matrix A and vector $\mathbf{d}$ look on this tree

where α_{ij} denotes the number of paths that edge e_{ij} is on, and β_{rsij} denotes the number of paths on which both edges e_{ij} and e_{rs} are. The reason being each term $(.)^2$ in RSS denotes the square error between a path length in T and its corresponding edge distance in K_n. We have $\binom{n}{2}$—equal to the number of paths between each two vertices in T—of these terms. Taking the derivative with respect to a w_{ij}, we are considering only the terms $(.)^2$ that include the edge e_{ij} which correspond to the paths that include edge e_{ij}. From equations (3) and (4), we have the following $n-1$ equations

$$\frac{\partial RSS}{\partial w_{ij}} = \alpha_{ij} w_{ij} + \sum_{e_{rs} \neq e_{ij}} \beta_{rsij} w_{rs} = \sum_{\substack{m,k:\, e_{ij} \in P_{m,k} \\ m<k}} d_{mk} \qquad \forall e_{ij} \in E. \tag{5}$$

The above linear system can be expressed in matrix form as $A\mathbf{w} = \mathbf{d}$, where the entries of matrix A are as follows. a_{ij} denotes the number of paths including the edge corresponding to the *i-th* entry of the vector $\mathbf{w}$ where $i = j$, and where $i \neq j$, it denotes the number of paths including the edges corresponding to the *i-th* and the *j-th* entries of the vector $\mathbf{w}$. Let us go through the following example to make it more clear.

For the tree in Fig. 1, the system of equations is as below.

$$\underbrace{\begin{bmatrix} 4 & 1 & 2 & 1 \\ 1 & 4 & 2 & 1 \\ 2 & 2 & 6 & 3 \\ 1 & 1 & 3 & 4 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} w_{01} \\ w_{02} \\ w_{03} \\ w_{34} \end{bmatrix}}_{\mathbf{w}} = \underbrace{\begin{bmatrix} d_{01}+d_{12}+d_{13}+d_{14} \\ d_{02}+d_{12}+d_{23}+d_{24} \\ d_{03}+d_{04}+d_{23}+d_{13}+d_{14}+d_{24} \\ d_{34}+d_{04}+d_{24}+d_{14} \end{bmatrix}}_{\mathbf{d}} \tag{6}$$

In the linear system above, the diagonal entries of A—a_{11}, a_{22}, a_{33} and a_{44}—are the number of paths passing respectively through the edges e_{01}, e_{02}, e_{03} and e_{34}. Also, for example, a_{12} is the number of paths passing through both edges e_{01} and e_{02}, and a_{34} is the number of paths passing through both edges e_{03} and e_{34}. In vector $\mathbf{d}$, in the first entry—$d_{01}+d_{12}+d_{13}+d_{14}$—the indices correspond to the beginning vertex and end vertex of the paths that the edge e_{01} is on, and d_{ij} is the distance between the vertices v_i and v_j in the complete graph.

The question is how do we count the number of paths passing through one specific edge or two specific edges in a tree effectively? Let us take one vertex of the tree as the root vertex and consider the tree directed based on that vertex where D_i denotes the descendants of vertex v_i. Also, α_{ij} and β_{ijrs} are as defined in equation (4). To answer the first part of the question—the number of paths passing through e_{ij} where $v_j \in D_i$—$\alpha_{ij} = (|D_j|+1)\,(n-(|D_j|+1))$. To answer the second part of the question—to count the number of edges passing through two edges—say, e_{ij} and e_{rs} where $v_j \in D_i$ and $v_s \in D_r$,

$$\beta_{ijrs} = \begin{cases} (|D_j|+1)(|D_s|+1) & D_j \cap D_s = \emptyset \\ (|D_j|+1)\,(n-(|D_s|+1)) & D_j \subset D_s \\ (|D_s|+1)\,(n-(|D_j|+1)) & D_s \subset D_j. \end{cases} \tag{7}$$

It can be seen that only the number of descendants of the bottom vertices of the edges e_{ij} and e_{ijrs} is factored in α_{ij} and $\beta ijrs$.

After finding all entries of A, we can find the edge weights by solving $A\mathbf{w} = \mathbf{d}$. Yet, is the matrix A necessarily invertible? In the following, we prove that not only is A invertible, but positive-definite.

Lemma 1. *A is a positive-definite matrix.*

Proof. We define the function Z on a spanning tree T as follows. For each edge e_{ij}, we assign a variable v_{ij}. Then we define $Z = \sum_{\substack{m,k \\ m<k}} \left(\sum_{\substack{i,j \\ e_{ij} \in P_{m,k}}} v_{ij} \right)^2$. We can see that the terms $(.)^2$ in Z are the same as those in RSS (equation (2)). The only difference being the variables w_{ij} are replaced with v_{ij} and the constants d_{ij} are replaced with 0. Z can be written as $Z = \mathbf{v}^\top B \mathbf{v} > 0$ where $\mathbf{v}$ is the vector of variables v_{ij}, and B is a matrix whose entries are as follows. b_{pq} is the number of terms $(.)^2$ in Z including the variable v_{ij} assigned to the *p-th* entry of vector $\mathbf{v}$ for $p = q$, and for $p \neq q$, b_{pq} is the number of terms $(.)^2$ including both variables v_{ij} and v_{rs} assigned to the *p-th* and *q-th* entries of vector $\mathbf{v}$. Since each term $(.)^2$ denotes a path in T, we can say that b_{pq} is the number of paths including the edge e_{ij} assigned the *p-th* entry of vector $\mathbf{v}$ for $p = q$, and for $p \neq q$, b_{pq} is the number of paths including both edges e_{ij} and e_{rs} assigned the *p-th* and *q-th* entries of vector $\mathbf{v}$. Thus, $B = A$, and since B is positive-definite, A is also a positive definite matrix. □

Since A is positive-definite, we can use the Cholesky decomposition of A in the form $A = LL^\top$ where L is a unique lower triangular matrix whose entries are computed by equations (8) and (9). From there, we can solve $Ly = \mathbf{d}$, and then $L^\top \mathbf{w} = y$ to find the weights. In the following, we discuss how to optimize the tree structure—find RSSOT.

$$L_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}^2} \tag{8}$$

$$L_{ij} = \frac{1}{L_{jj}} \left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} \right) \tag{9}$$

III. Problem: Tree structure optimization

So far, we discussed how we can find the edge weights for a given tree based on the distances in the complete graph. The question is, how can we find the tree with minimum RSS? In other words, how can we optimize the tree structure to find RSSOT? We can build n^{n-2} spanning trees on any n number of labeled vertices. That means for as few as 50 labeled vertices, we can have roughly as many spanning trees as the number of atoms in the known universe. Due to the large scale of the problem, we make use of two metaheuristics—in this case, Simulated Annealing (SA) and Iterated Local Search (ILS)—to approximate the optimal tree. These are two of the typical metaheuristics applied to such difficult optimization problems. Below, we explain how to make a structure change in a tree, and how to use SA and ILS to optimize the tree structure based on the structure change.

A. Tree structure change for optimization

Before discussing SA and ILS on a tree, let us explain how we make a change in the structure of a given tree in order ot accept or reject the transition between two states. Let T_t be the tree at time t and let us denote its corresponding structure by $T(V, E)$. Let us also denote the structure after change by $T'(V, E')$—the structure that we want to accept or reject. For $v_i \in V$, we denote the neighbours of v_i by $N(v_i)$. We pick one edge $e_{ij} \in E$. Then we define set C as $C = N(v_i) \cup N(v_j) \setminus \{v_i, v_j\}$. We pick $v_k \in C$ uniformly at random. If $v_k \in N(i)$, then $E' = E \cup \{e_{jk}\} \setminus \{e_{ik}\}$; otherwise, if $v_k \in N(j)$, then $E' = E \cup \{e_{ik}\} \setminus \{e_{jk}\}$. We denote the former structure change by $SC(T, e_{ij}, e_{jk}, e_{ik})$ and the latter by $SC(T, e_{ij}, e_{ik}, e_{jk})$. In $SC(T, ., ., .)$, the second, third, and forth terms are respectively the picked edge, the edge that is added to, and the edge that is removed from the tree.

The other thing we investigate before discussing SA and ILS algorithms on a tree is the change in matrix A and vector $\mathbf{d}$ following the structure change in $T(V, E)$. Should we recompute every entry of A and $\mathbf{d}$ after every structure change? Let us define A' and $\mathbf{d}'$ as the matrix and vector corresponding to $T'(V, E')$.

Lemma 2. *Suppose we have the structure change $SC(T, e_{ij}, ., .)$ resulting in tree $T'(V, E')$. All the entries of A and A' are the same except the rows and columns corresponding to e_{ij}. So are all the entries in $\mathbf{d}$ and $\mathbf{d}'$ except the entry corresponding to e_{ij}. Thus, we only need to recompute the entry in $\mathbf{d}$, and the rows and columns in A corresponding to e_{ij}, to obtain A' and $\mathbf{d}'$.*

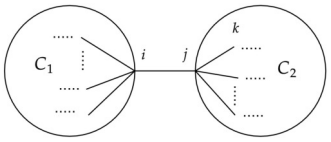
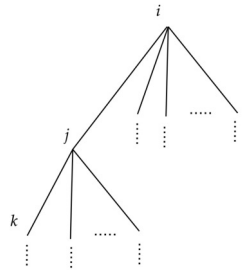
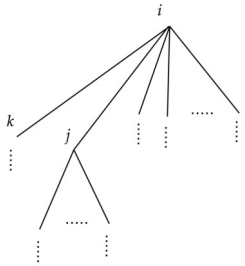


Fig. 2: Tree $T(V, E)$ before the structure change with picked edge e_{ij} connecting components C_1 and C_2, and randomly picked vertex $v_k \in C$



(a) Tree before structure change



(b) Tree after structure change

Fig. 3: Demonstration of the structure change $SC(T, e_{ij}, e_{ik}, e_{jk})$. Only v_j has a different number of descendants in T' than it has in T.

Proof. Consider the tree $T(V, E)$ in Fig. 2 on which we want to make the structure change based on the picked edge e_{ij} and $v_k \in C$—C as defined above. α and β are as defined in equation (4) for $T(V, E)$, and the equivalents of them are α' and β' for $T'(V, E')$. If $v_k \in N(v_j) \setminus \{v_i\}$, the structure change is $SC\,(T, e_{ij}, e_{ik}, e_{jk})$.

Let us look at $T(V, E)$ as a directed tree with the root vertex v_i—Fig. 3a. This tree before and after the structure change is illustrated in Fig. 3. Consider the subgraph $S = G(V, E'')$ in $T'(V, E')$—Fig. 3b—where $E'' = E' \setminus \{e_{ij}, e_{ik}\}$. It can be seen that every vertex but v_j in this subgraph has the exact same descendants in T' as they have in T. Thus, since $E'' \subset E'$ and $E'' \subset E$ and based on the calculation of α and β in Setion II, we can say that the number of paths that pass through any edge or any two edges in E'' is the same in T and T'. Similarly, regarding $e_{ik} \in E'$ and $e_{jk} \in E$, $\alpha'_{ik} = \alpha_{jk}$ and $\beta'_{ikrs} = \beta_{jkrs}$ for all $e_{rs} \in E''$. Hence, we see that e_{ij} is the only edge for which $\alpha'_{ij} \neq \alpha_{ij}$ and $\beta'_{ijrs} \neq \beta_{ijrs}$ where $e_{rs} \in E \cap E' \setminus e_{ij}$.

□

B. Simulated Annealing (SA)

As mentioned above, let us say the structure of the tree at time t is $T(V,E)$—$T_t \leftarrow T(V,E)$. Let us also denote $RSS(T', K_n)$ and $RSS(T, K_n)$ by RSS' and RSS respectively. Starting from a random initial tree structure T_0, we make the transition from $T_t \leftarrow T(V,E)$ to $T_{t+1} \leftarrow T'(V,E')$ in either of the following two cases:

1) $RSS' < RSS$
2) $P\left(\frac{RSS'-RSS}{RSS}, t\right) < \text{random}(0,1)$ if $RSS' > RSS$.

Otherwise, $T_{t+1} \leftarrow T$. In the above, random$(0,1)$ denotes a number picked uniformly at random in the interval $(0,1)$. The second case accepts the new tree structure with a worse RSS value with a certain probability. $P(RSS', RSS, t) = a_1 e^{-a_2 (\ln t)^{a_3} \frac{RSS'-RSS}{RSS'}}$, and it can be seen that the probability of accepting $RSS' > RSS$ decreases with time t. The parameters a_1, a_2 and a_3 are tuned according to how often we are willing to accept a transition with a larger RSS' than RSS, and such that accepted RSS' values roughly converge for a large t.

C. Iterated Local Search (ILS)

In ILS, we make the transition from $T_t \leftarrow T$ to $T_{t+1} \leftarrow T'$ only if $RSS' < RSS$—so far, it is a descent-only algorithm. However, in contrast to a descent-only algorithm, when we get stuck in a local minimum, we restart the algorithm—by modification of the current local minimum—to a new tree structure. Basically, ILS consists of the following two steps:

1) Modification of the current local minimum by kicking it far enough from its current basin
2) Descent to get to a new local minimum.

We want to try every possible structure change to make sure the function RSS is stuck at a local minimum. To this end, for any picked edge e_{ij}, the number of structure changes that we can make depends on $|C_i| = |N(v_i) \setminus \{v_j\}|$ and $|C_j| = |N(v_j) \setminus \{v_i\}|$. If we remove the edge e_{ij} from T, the resulting graph $G(V, E \setminus \{e_{ij}\})$ consists of two trees T_i and T_j where $v_i \in T_i$ and $v_j \in T_j$. We assume the average degree of a tree to be two; thus, we assume the degree of both v_i and v_j to be 2. With this assumption, the number of possible structure changes based on the picked edge e_{ij} is four, so for the whole tree, we estimate the number of possible structure changes at $4n$. If we try structure changes on a tree uniformly at random, the average number of times that we need to try all possible structure changes is $4nH_{4n}$—based on the well-known Coupon collector's problem—where H_{4n} is the $4n$-th harmonic number defined as $H_k = \sum_{i=1}^{k} \frac{1}{i}$. That is why we set $4nH_{4n}$ as the threshold to determine the algorithm is stuck at a local minimum.

IV. Results

We applied SA and ILS as described in Section III to evaluate the performance of these two metaheuristics in different scenarios. We evaluated whether bias towards smaller edges—picking an edge e_{ij} with a smaller weight for the tree structure change $SC(T, e_{ij}, ., .)$ with higher probability—has any advantage in SA over no bias—picking e_{ij} uniformly at random—in SA. After extensive experiments, we found that biased SA in general has a slight advantage over unbiased SA, so in the following, SA refers to biased SA.

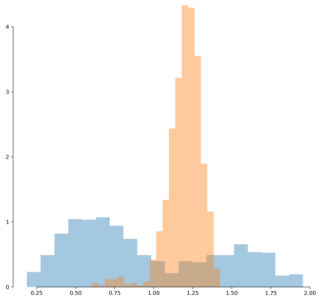


Fig. 4: Dispersion of sample of size 50 in Tables I and II

We compared the performance of SA and ILS based on running each of them ten times over the complete graph—where the distances in K_n are derived from stock-correlation data. See Tables I and II for a performance comparison of SA and ILS. In these tables, in each of the 10 runs, we ran each algorithm—SA and ILS—on trees with sizes of 20, 30, and 50 respectively for 10 minutes, two hours, and 18 hours. The values in the tables are for the minimum RSS value found in its corresponding run—according to which we evaluate the performance of the algorithm. In Table I, it can be seen that the performance of ILS is much better than that of SA. However, in Table II, we can see that there is no apparent difference between SA and ILS performance.

The reason for performance inconsistency of SA in Tables I and II seems to be the dispersion in distances of the complete weighted graph used in each of them. For example, for tree of size 50 in each table, dispersion of distances in the complete weighted graph is illustrated in Fig. 4 with a histogram. It can be seen that for distances with high dispersion, SA and ILS have a similar performance while for distances with low dispersion, ILS maintains a solid performance, but SA performance sharply decreases. We got the same result by running SA and ILS on the trees of many other complete weighted graphs of distances. It is noteworthy that for distance values with low dispersion, both the biased and unbiased SA, where the biased SA picks lightweight edges with a higher probability, have a poor performance. The reason possibly being, when distance values are close to each other, smaller distance values are not considerably smaller than the large distance values—giving no edge to biased over unbiased.

V. Conclusion

We have presented a scheme to optimize the edges weights and structure of a tree to approximate a complete weighted graph using a measure involving the path distances in the tree.

TABLE I: SA vs ILS on a complete weighted graph with low dispersion of distances. For each tree, the metaheuristic with a better performance has been highlighted.

	Tree size					
	20		30		50	
Run	SA	ILS	SA	ILS	SA	ILS
1	5.291951597	**4.84414153**	9.501749529	**8.4082242**	24.8579367	**16.8823573**
2	8.21566953	**4.84414153**	11.66835384	**8.4082242**	26.7081655	**16.8823573**
3	7.797470793	**4.84414153**	11.16650355	**7.97443788**	19.6536142	**16.8823573**
4	6.875995126	**4.84414153**	11.66835384	**7.97443788**	25.5941918	**16.8823573**
5	6.875995126	**4.84414153**	8.408224203	**7.97443788**	26.284257	**16.8823573**
6	6.019906558	**4.84414153**	12.98953771	**7.97443788**	30.2829294	**16.8823573**
7	7.016393465	**4.84414153**	10.3933439	**7.97443788**	34.8261135	**16.8823573**
8	7.016393465	**4.84414153**	11.80185307	**7.97443788**	31.0322914	**16.8823573**
9	**4.844141529**	5.2919516	13.02265027	**7.97443788**	31.1982311	**16.8823573**
10	7.016393465	**4.84414153**	10.58531443	**7.97443788**	26.265805	**16.8823573**
Average	6.697031065	**4.88892254**	11.12058844	**8.06119515**	27.6703536	**16.8823573**

TABLE II: SA vs ILS on a complete weighted graph with high dispersion of distances. For each tree, the metaheuristic with a better performance has been highlighted.

	Tree size					
	20		30		50	
Run	SA	ILS	SA	ILS	SA	ILS
1	5.75665216	5.75665216	**12.7055242**	12.8162316	32.18863674	**31.8890846**
2	5.75665216	5.75665216	**12.7055242**	12.7495148	**31.78033885**	31.8140415
3	5.75665216	5.75665216	**12.7464911**	12.8550396	**31.68615883**	31.8986455
4	5.75665216	5.75665216	**12.7055242**	12.8345686	32.0511834	**31.9636283**
5	5.75665216	5.75665216	**12.7055242**	13.0379292	**31.65783212**	31.6641679
6	5.75665216	5.75665216	**12.7055242**	12.832506	32.07947769	**31.7900896**
7	5.75665216	5.75665216	**12.7615612**	12.8750193	**31.91069847**	32.0325964
8	5.75665216	5.75665216	**12.7055242**	12.8785592	**32.00314739**	32.2380063
9	5.75665216	5.75665216	**12.7055242**	12.7207733	**31.96688445**	31.9796543
10	5.93707406	**5.75665216**	**12.7207733**	12.8472376	31.93720134	**31.792651**
Average	**5.77469435**	5.75665216	**12.7167495**	12.8447379	31.92615593	**31.9062565**

We have proposed a very efficient way of computing modifications to the tree that assist with local search metaheuristics, and evaluate the performance of two of these: SA and ILS.

REFERENCES

[1] J. Felsenstein and J. Felenstein, *Inferring phylogenies*. Sinauer associates Sunderland, MA, 2004, vol. 2.

[2] G. Narasimhan and M. Smid, *Geometric spanner networks*. Cambridge University Press, 2007.

[3] R. N. Mantegna, "Hierarchical structure in financial markets," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 11, no. 1, pp. 193–197, 1999. [Online]. Available: https://doi.org/10.1007/s100510050929

[4] M. Tumminello, T. Aste, T. Di Matteo, and R. N. Mantegna, "A tool for filtering information in complex systems," *Proceedings of the National Academy of Sciences*, vol. 102, no. 30, pp. 10 421–10 426, 2005. [Online]. Available: https://doi.org/10.1073/pnas.0500298102

[5] V. Boginski, S. Butenko, and P. M. Pardalos, "Statistical analysis of financial networks," *Computational statistics & data analysis*, vol. 48, no. 2, pp. 431–443, 2005. [Online]. Available: https://doi.org/10.1016/j.csda.2004.02.004

[6] M. Tumminello, C. Coronnello, F. Lillo, S. Micciche, and R. N. Mantegna, "Spanning trees and bootstrap reliability estimation in correlation-based networks," *International Journal of Bifurcation and Chaos*, vol. 17, no. 07, pp. 2319–2329, 2007. [Online]. Available: https://doi.org/10.1142/S0218127407018415

[7] A. Kocheturov, M. Batsyn, and P. M. Pardalos, "Dynamics of cluster structures in a financial market network," *Physica A: Statistical Mechanics and its Applications*, vol. 413, pp. 523–533, 2014. [Online]. Available: https://doi.org/10.1016/j.physa.2014.06.077

[8] J.-P. Onnela, A. Chakraborti, K. Kaski, J. Kertesz, and A. Kanto, "Asset trees and asset graphs in financial markets," *Physica Scripta*, vol. 2003, no. T106, p. 48, 2003.

[9] J. Birch, A. A. Pantelous, and K. Soramäki, "Analysis of correlation based networks representing dax 30 stock price returns," *Computational Economics*, vol. 47, no. 4, pp. 501–525, 2016. [Online]. Available: https://doi.org/10.1007/s10614-015-9481-z

[10] D. Han *et al.*, "Network analysis of the chinese stock market during the turbulence of 2015–2016 using log-returns, volumes and mutual information," *Physica A: Statistical Mechanics and its Applications*, vol. 523, pp. 1091–1109, 2019. [Online]. Available: https://doi.org/10.1016/j.physa.2019.04.128

[11] R. Desper and O. Gascuel, "Theoretical foundation of the balanced minimum evolution method of phylogenetic inference and its relationship to weighted least-squares tree fitting," *Molecular Biology and Evolution*, vol. 21, no. 3, pp. 587–598, 2004. [Online]. Available: https://doi.org/10.1093/molbev/msh049