International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8 Issue-3, September 2019

Terminal Wiener Index of Balanced Trees

Sulphikar A

Abstract for A Tree T, The Terminal Wiener Index TW(T) Is Defined As Half The Sum Of All Distances Of The Form D(U, V), Where The Summation Is Over All Possible Pairs Of Pendent Vertices U,V In T. We Consider A Class Of Balanced Binary Trees Called 1-Trees And Compute Their Terminal Wiener Index Values. We Also Determine 1-Trees With Minimum And Maximum Terminal Wiener Index.

Keyword sbalanced Binary Trees, Distance In Graphs, Pendant Vertex, Terminal Wiener Index.

I. INTRODUCTION

Let $G = (V, E)$ be a connected graph on n vertices. The Wiener index $W(G)$ of G is defined as the sum of distances between all pairs of vertices of G [2, 4]. The formula

$$W(T) = \sum_{e=uv} n_u(e|T)n_v(e|T) \qquad (1)$$

given in [4, 5] holds for any tree T where the summation is over all edges - here $n_u(e|T)$ and $n_v(e|T)$ are the number of vertices lying on the two sides of the edge e=uv. For a tree T, the terminal Wiener index TW(T) of T is defined as the sum of distances between all pairs of pendent vertices of T[7] . That is

$$TW(T) = \sum_{(u,v)} d(u, v)$$

where u,v are pendent vertices in T . Let T be an n-vertex tree with k pendent vertices. Then TW(T) can be expressed as

$$TW(T) = \sum_{e} p_u(e|T)p_v(e|T) \qquad (2)$$

where $p_u(e|T)$ and $p_v(e|T)$are the number of pendent vertices in T, lying on the two sides of the edge e=uv[3]. A k-tree, k = 0, 1, 2, . . . can be defined as a rooted binary tree of height h that satisfies the following two conditions[1].

(i) every node of depth less than h - k has exactly two children where h is the height of the tree, and
(ii) a node of depth at least h - k has at most two children.

The family of all k-trees are represented by the set F_k. It is easy to see that $F_0 \subset F_1 \subset F_2$........ . Only complete binary trees are contained in the set F_0.

Terminal distance matrices were used in the mathematical modelling of proteins and genetic codes [9]. The importance of terminal Wiener index has been shown through numerous articles in both mathematics and chemistry[4, 7] as well as in other areas such as phylogeny reconstruction.

In the next section, we explain a method for computing terminal Wiener index of rooted trees. Section 3 explains computation of terminal Wiener index of 1-trees. Section 4 explains 1-trees with minimum and maximum terminal Wiener index. Section 5 gives implementation details of algorithm to compute terminal Wiener index of 1- trees.

II. COMPUTING TERMINAL WIENER INDEX OF A TREE

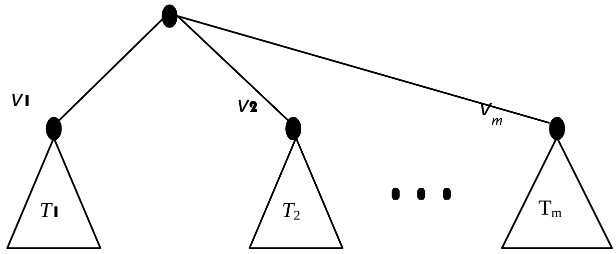


Figure 1: Computing terminal Wiener index of a tree

Let T be a tree of order n with root v_r . For $m \geq 2$, let $T_1, T_2, \ldots, T_m$ be trees

With disjoint vertex sets and pendent vertices $p_1, p_2, \ldots, p_m$. Let for

$i = 1, 2 \ldots m$, $v_i \in V(T_i)$ be the roots of T_i and $P = p_1 + p_2 + \cdots + p_m$.

Any tree T on more than two vertices can be viewed as being obtained

by joining a new vertex v_r to each of thevertices $v_1, v_2, \cdots, v_m$ as

shown in Figure 1. Then

$$TW(T) = \sum_{i=1}^{m} [TW(T_i) + d^+(v_i)(P - p_i) - p_i^2] + P^2$$

where $d^+(v_i) = \sum_{p_i \in P(T_i)} d(p_i, v_i)$

For a rooted tree T, we denote by $d^+(u)$ the sum of all distances from u to all its pendent vertices. For the tree in Fig.1, $d^+(v_r)$ can be calculated as

$$d^+(v_r) = P + \sum d^+(v_i)$$

Manuscript published on 30 September 2019
* Correspondence Author
Sulphikar A*, Department of computer science and engineering National Institute of Technology TrichyTamilnadu INDIA
E-mail:sulphis@gmail.com

where P denote the number of pendent vertices in T.
Let T be tree with a pendent vertex v.
If u is a vertex adjacent to v, then TW(T) can be calculated using the following recursive procedure:

1. **procedure** TW(T) *t*> This computes terminal Wiener index of a tree T
2. **if** |V(T)| =2 **then**
3. return 1
4. **else**
5. Choose a pendent edge uv with $degree(v)$ =1
6. **if** $degree(u) = 2$ **then**
7. return TW (T-v) + P (T-v)- 1
8. **else**
9. return $TW(T-v) + d^{+}(u) + P(T-v)$

The above recursive procedure can be used to calculate of terminal Wiener index of a rooted tree in a bottom up manner

III. A TREE TRANSFORMATION

Consider a rooted tree T with a vertex y**.** We denote by T(y) the subtree rooted at y
and its order by n(y). We define an operation that swaps two subtrees of T rooted at
x and y.

Theorem 3.1. Let T be a rooted tree with two vertices x and y of the same depth. Let z be the lowest common ancestor of both x and y. Let $x_0, x_1, \ldots, x_l$ be the path between x_0 and x_l with $x_0 = z$ and $x_l = x$ and let $y_0, y_1, \ldots, y_l$ be the path between y_0 and y_l with $y_0 = z$ and $y_l = y$. Let T' be the tree obtained by swapping subtrees T (y) and T (x) in T. Then

$$TW(T') = TW(T) - 2l\lambda^2 + 2\lambda\sum p(y_i) - p(x_i)$$

where $\lambda = p(y)-p(x)$.

Proof. It is easy to note that only those edges that lie on the path between x and y change their weights during a swap operation. The path P between x and y contains $2l$ edges. We define $f(a) = a(p - a)$. Consider the edge (x_{i-1}, x_i) of P. Its weight in T *is* $f(p(x_i))$ and
its weight in T' is $f(\lambda + p(x_i))$. For the edge (x_{i-1}, x_i), the difference in
weights is

$$\begin{aligned} f(\lambda + p(x_i)) - f(p(x_i)) &= (p(x_i) + \lambda)(p - p(x_i) - \lambda) - p(x_i)(p - p(x_i)) \\ &= \lambda(-p + 2p(x_i) - \lambda) \end{aligned}$$

Similarly the difference in weights of edge (y_{i-1}, y_i) is
$(p(y_i) - \lambda)(p - p(y_i) + \lambda) - p(y_i)(p - p(y_i)) = \lambda(-p + 2p(y_i) - \lambda)$.

Difference in weights of two edges (x_{i-1}, x_i) and (y_{i-1}, y_i) together is
$2\lambda(p(y_i)-p(y_i)-\lambda)$. Multiplying this by l we will get the result.

Corollary 3.1

If *p(y)=p(x),* then both *T* and *T′* have the same terminal Wiener Index.

If *p(y)=p(x),* then both *T* and *T′* have the same terminal Wiener Index.

Next we will derive bounds for terminal Wiener index of k-trees

Theorem 3.2. *Let T be a k-tree of order n. Then* $d^{+}(u) = n(k - 1 + log(n + 1))$ *and* $TW(T) < (n - 1)(n - 2)(k - 1 + log(n + 1))$.

Proof. Let h and h_1 denote the height of T and the smallest height of a vertex of T with at most one child. Then the vertices of height at most h_1 form a complete binary tree. The number of vertices in T, $n \geq 2^{h_1+1} - 1$ or $h_1 \leq log(n + 1) - 1$. Therefore
$d^{+}(u) \leq nh \leq n(k + h_1) \leq n(k - 1 + log(n + 1))$ since $h_1 \geq h - k$.

The terminal Wiener index of T is at most $2hp(p-1)/2 \leq p(p - 1)(k - 1 + log(n + 1)) \leq$
$(n - 1)(n - 2)(k - 1 + log(n + 1))$ since the maximum value of p is $n - 1$.

Lemma 3.1. *Let T be a* 1*-tree of order n. Let p_1 and p_2 be the number of pendent vertices in the left and right subtrees of root of T. Let T_a and T_b be two trees obtained by removing the root from T. Then*

$$TW(T) = TW(T_a) + TW(T_b) + d^{+}(u)p_2 + d^{+}(v)p_1 + 2p_1p_2$$

where u and v are the left and right children of the root of T respectively.

Lemma 3.2. *Let T_a and T_b be two trees of orders n_1 and n_2 respectively. Let $u \in V(T_a)$, $v \in V(T_b)$ and $T_a.T_b(u, v)$ denotes the new tree obtained from T_a and T_b by identifying u and v. Let the number of pendent vertices in T_a and T_b be p_1 and p_2 respectively.*

i. *If both u and v are pendent vertices, then*

$$TW(T_a.T_b(u, v)) = TW(T_a) + TW(T_b) + (p_2 - 2)d^{+}(u) + (p_1 - 2)d^{+}(v).$$

ii. *If u is nonpendent and v is pendent, then*

$$TW(T_a.T_b(u, v)) = TW(T_a) + TW(T_b) + (p_2 - 1)d^{+}(u) + (p_1 - 1)d^{+}(v).$$

iii. *If u is pendent and v is nonpendent, then*

$$TW(T_a.T_b(u, v)) = TW(T_a) + TW(T_b) + (p_2 - 1)d^{+}(u) + (p_1 - 1)d^{+}(v).$$

iv. *If both u and v are nonpendent vertices, then*

$$TW(T_a.T_b(u, v)) = TW(T_a) + TW(T_b) + p_2d^{+}(u) + p_1d^{+}(v).$$

OPEN ACCESS
International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878, Volume-8 Issue-3, September 2019

IV. TREES WITH MINIMUM AND MAXIMUM TERMINAL WIENER INDEX

1-tree of height h consists of 2^{h-1} nodes at height h- 1. The number of nodes at height h may vary between 1 and 2^h. Therefore a 1-tree of height h can be constructed from a complete binary tree of height h-1 by adding $m(1\leq m\leq 2^h)$ vertices at height h.

The operation of adding a new vertex at height h can be considered as identifying a vertex of path P_2 and a vertex at height h- 1. We use the lemma 3.2 to find 1-trees with minimum and maximum terminal Wiener index.

In this case T_a is a complete binary tree of height $h-1$ and T_b is P_2. u in lemma is a vertex of T_a at height $h-1$ and v is a vertex of P_2. Since P_2 has no nonpendent vertices, only case(i) and (ii) of lemma 3.2 is applicable in this case. Since P_2 has two pendent vertices (p_2-2 = 0), computation of $d^+(u)$ is not required in case(i). We state two theorems for finding 1-trees of height h with minimum and maximum terminal Wiener index.

Theorem 4.1. *Terminal Wiener index of 1-tree can be minimized by maximizing the number of identify operations of a pendent vertex of complete binary tree of height $h-1$ and a vertex of P_2.*

Proof. From lemma 3.2, it is easy to see that if $TW(T_a)$,$TW(T_b)$, $d^+(u)$ and $d^+(v)$ have the same value, then case(i) achieves the minimum value of terminal Wiener index. ∎

Theorem 4.2. *Terminal Wiener index can be maximized by maximizing the number of identify operations of a nonpendent vertex at height $h-1$ of a 1-tree of height h and a vertex of P_2 and by equally distributing these identifications among the left and right subtrees of root.*

Proof. From lemma 3.2, it is easy to see that if $TW(T_a)$, $TW(T_b)$, $d^+(u)$ and $d^+(v)$ have the same value, then case(ii) achieves the minimum value of terminal Wiener index. Note that case(iii) and case (iv) are not applicable. ∎

If we have only one vertex at height h, then 2^{h-1} ways to insert it. By theorem 3.1, all these trees have same terminal Wiener index. If we have 2^h vertices at height h, then also the tree is unique. The case for 2^h-1 vertices at height h is also similar. Therefore, we assume that the number of vertices at height h is between 2 and 2^h- 2. If we have only 2 vertices, then these can be identified with any two of the 2^{h-1} pendent vertices to minimize terminal Wiener index. By theorem 3.1, we can see that all such possible trees have same terminal Wiener index.

V. COMPUTATION OF TERMINAL WIENER INDEX OF 1-TREES

We use a tuple (n,tw,l,d) to denote the set of 1-trees of order n with same terminal Wiener index tw where l denotes the number of pendent vertices and d is same as $d^+(root)$. The algorithm is based on the partitioning of integers. n_1 takes values from 1 to $(n-1)/2$ and for each value of n_1, n_2 is computed as $n-n_1-1$. By using lemma 3.1 we create a new tree (n, tw, l, d) from two groups of trees (n_1, tw_1, l_1, d_1) and (n_2, tw_2, l_2, d_2). Assuming that the sets TW $(F_1(i))$, $i<n$ are already computed and stored as a sorted list, the algorithm checks whether T , the combinations of two 1-trees T_1 and T_2 corresponding to pairs (tw_1, n_1, l_1, d_1) and (tw_2, n_2, l_2, d_2) is a valid 1-tree. Heights of T_i , i = 1,2 can be computed as $h_i = log(n_i)$. If both heights are equal, then T is a valid 1-tree. If the absolute value of difference in heights is 2 or more, then the tree is invalid. Suppose that $|h_1-h_2|=1$. If $h_1<h_2$, then T is a valid 1- tree if and only if T_1 is a complete binary tree. We can compute terminal Wiener index of 1-trees efficiently by using lemma 3.1.

Since the tuples are stored as an ordered list, binary search can be used for finding tuples for n_1 and n_2. There may be many tuples with the same n value, so we will have to locate the first tuple with a given value of n. For this purpose we use a modified binary search BsearchD(list,item). The following algorithm uses procedure BsearchD(list,item) to find the first tuple with a given value of n.

```
procedure Balanced TWI(T)   ▷ This computes terminal Wiener index of a balanced tree T
Initialize a list with 3 tuples(1,0,0,0)(2,1,2,1)(3,2,2,2)
    for j = 4 to n do
        for x = 1 to int( (j − 1)/2) do
            n1 ← x
            n2 ← j − x − 1
            h1 ← int((log2(n1))
            h2 ← int (log2(n2))
            if |h1 − h2| >=2 then
                continue
            else if |h1 − h2| == 1 then
                h3 ← min(h1, h2)
                h4 ← h3 + 1
                if n1! = 2** h4-1 and n2! = 2** h4-1 then
                    continue
            mid1 ←BsearchD(list, n1)
            mid2 ←BsearchD(list, n2)
            for i = mid1 to len(list) do
                if list[i][0] == n1 then              ▷ Extract tuple for n1 from list
                    tw1 ← list[i][1]
                    l1 ← list[i][2]
                    d1 ← list[i][3]
            for m = mid2 to len(list) do
                if list[i][0] == n2 then              ▷ Extract tuple for n2 from list
                    tw2 ← list[m][1]
                    l2 ← list[m][2]
                    d2 ← list[m][3]
            Compute terminal Wiener index tw using lemma 3.1
            l ← l1 + l2
            d ← d1 + d2 + l1 + l2
            t ←  (j, tw, l, d)
            Check whether t exist in the list or not.
            If it does not exist, then insert into list.
```

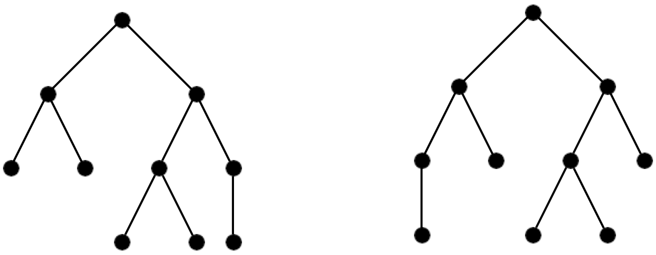
VI. IMPLEMENTATION

We implemented the above algorithm using Python 2.7.12 and computed terminal Wiener index of 1-trees having upto 100 vertices. The values of *tw* upto $n = 20$ are listed in the following table.

Table 1. (n,tw,l,d) values upto $n = 20$

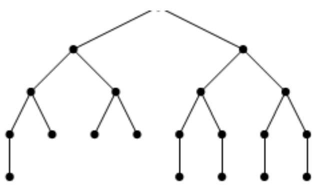
From the above table it is clear that 5 is the smallest integer that is *not* terminal Wiener index of 1-tree. The same tuple may represent two or more non-isomorphic trees [11]. As an example, consider the tuple (10,42,5,13). There exist 2 non-isomorphic 1-trees with these values as shown in the following figure.

n	tw	l	d	n	tw	l	d	n	tw	l	d	n	tw	l	d	n	tw	l	d
1	0	1	0	9	26	4	10	12	67	6	17	17	186	8	27	20	210	9	32
2	1	1	1	9	38	54	12	12	69	6	17	18	157	9	30	20	251	10	35
3	2	2	2	10	29	4	11	13	72	6	18	18	194	9	30	20	253	10	35
4	3	2	3	10	42	5	13	13	74	6	18	19	164	8	28	20	255	10	35
5	4	2	4	11	32	4	12	13	96	7	20	19	202	9	31				
5	8	3	5	11	46	5	14	14	102	7	21	19	242	10	34				
6	10	3	6	11	62	6	16	15	136	8	24	19	244	10	34				
7	20	4	8	11	64	6	16	16	143	8	25	19	246	10	34				
8	23	4	9	12	50	5	15	17	150	8	26	20	171	8	29				

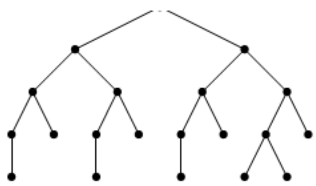


By theorem 3.2, the sizes of $TW(F_1(n_1))$ and $TW(F_1(n_2))$ are bounded by $O(n^2)$. After computing the terminal Wiener index, we will check whether the tuple already exists in the list in $O(log(n))$ time. If the tuple does not exist in list, we will insert into list. Therefore, the total computation time of BalancedTWI(T) is $O(n^5 logn)$.

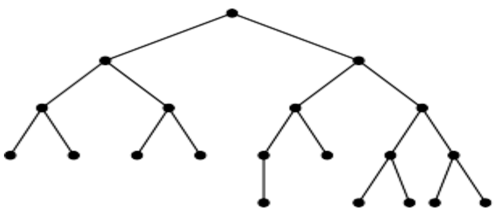
Next figure illustrates the five 1-trees generated for $n = 20$.



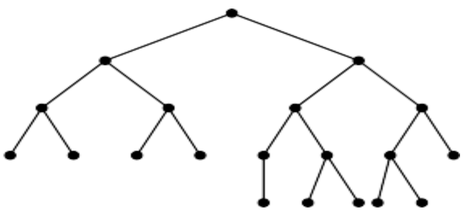
Tree for tuple (20, 171, 8, 29)



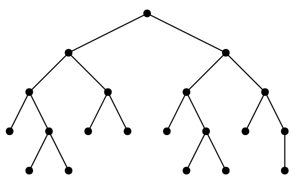
Tree for tuple (20, 210, 9, 32)



Tree for tuple (20, 251, 10, 35)



Tree for tuple (20, 253, 10, 35)



Tree for tuple (20, 255, 10, 35)

VII. CONCLUSION

In this paper we discussed about computation of terminal Wiener index of 1-trees. For a given order, 1-trees with minimum and maximum terminal Wiener index is also determined. We have implemented a method to store the terminal Wiener index of 1-trees. Future work includes computation of terminal Wiener index of k-trees for $k \geq 2$.

REFERENCES

1. S. Bereg, H. Wang, "Wiener indices of balanced binary trees," Discrete Applied Mathematics, Vol.155, pp. 457-67, Oct. 2007.
2. H. Dong, X. Guo, "Ordering trees by their Wiener indices,", MATCH Commun. Math. Comput. Chem., Vol. 56,pp.527-540,2006 .
3. Y. H. Chen, X. D. Zhang, "On Wiener and terminal Wiener indices of trees,", MATCH Commun. Math. Comput. Chem., Vol.70, pp.591-602,2013,.
4. A.A. Dobrynin, R. Entringer, I. Gutman, "Wiener index of trees: theory and applications,", Acta Appl. Math., Vol. 66, pp.211-249,2001.

A. A. Dobrynin, I. Gutman, Boris Furtula, "Equiseparable chemical trees,", Journal of serbian chemical society, Vol.68, pp.549-555,2003.

5. Gutman, Boris Furtula, D.Vukicevi´c, Biljana Arsic, "Equiseparable molecules and molecular graphs,", Indian Journal of Chemistry, Vol. 43,pp.7- 10, 2004 .
6. Gutman, B. Furtula and M. Petrovi´c, " Terminal Wiener index,", J. Math. Chem., Vol.46, pp.522-531, Oct.2009.

AUTHORS PROFILE

Sulphikar.A received his B.Tech from Kerala university and M.Tech from National Institute of Technology, Surathkal, Karnataka, India. He is Currently doing Ph.D at National institute of technology, Tiruchirappalli, Tamilnadu,India. His research areas include algorithms and graph theory.