HOW TO COMPUTE THE WIENER INDEX OF A GRAPH*

Bojan MOHAR and Tomaž PISANSKI

Department of Mathematics, University E.K. of Ljubljana, Jadranska 19, 61111 Ljubljana, Yugoslavia

and

Department of Mathematics and Statistics☆, Simon Fraser University, Burnaby, B.C., Canada

Received 8 July 1987
(in final form 27 October 1987)

Abstract

The Wiener index of a graph G is equal to the sum of distances between all pairs of vertices of G. It is known that the Wiener index of a molecular graph correlates with certain physical and chemical properties of a molecule. In the mathematical literature, many good algorithms can be found to compute the distances in a graph, and these can easily be adapted for the calculation of the Wiener index. An algorithm that calculates the Wiener index of a tree in linear time is given. It improves an algorithm of Canfield, Robinson and Rouvray. The question remains: is there an algorithm for general graphs that would calculate the Wiener index without calculating the distance matrix? Another algorithm that calculates this index for an arbitrary graph is given.

1. Introduction

In recent research in mathematical chemistry, particular attention has been paid to so-called *topological indices*. These are invariants which can be calculated from the underlying molecular graphs and hopefully exhibit good correlations with physical and chemical properties of the corresponding molecules. The reader is referred to the overview articles by Rouvray [12–14] and Motoc and Balaban [8] for further details. For basic concepts of graph theory, the reader is referred to [4,20].

One of the oldest and widely used indices is the Wiener index. It has been used to model various properties of chemical species. Wiener [21,22] used it for

*Work supported in part by the Research Council of Slovenia, Yugoslavia.
☆This work was done while both authors were visiting the Simon Fraser University.

biparametric correlations with the boiling point and some other thermodynamic parameters. Stiel and Thodos [18] used this index to predict critical constants, Rouvray and Crafford [16] correlated it with density, viscosity, and surface tension, Rouvray and Tatong explored prediction of the ultrasonic sound velocity in alkanes and alcohols [17], Papazova et al. [11] and Bonchev et al. [3] observed a relation with chromatographic retention times.

The Wiener index can be defined for an arbitrary connected graph as follows. Without loss of generality, assume that G has vertices $1, 2, \ldots, n$. For each pair i, j of vertices, let d_{ij} denote the distance in G between i and j; i.e. the length of the shortest path between i and j. The distances d_{ij} form the so-called *distance matrix* $D(G) = [d_{ij}]$ of the graph G. The Wiener index of G is the number

$$W(G) = \sum_{i=1}^{n} \sum_{j=1}^{i} d_{ij} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} .$$

In recent papers (Bersohn [2] and Müller et al. [9]), several algorithms for calculating $W(G)$ for an arbitrary graph G are given. The computational complexity of the latter is of the order of $O(n^3 \log n)$, where n is the number of vertices of G.

In the mathematical and computer science literature on combinatorial optimization, many excellent algorithms for computing $D(G)$ can be found. These are the so-called *shortest path algorithms,* and in general solve even more complicated problems where edges are allowed to carry weights. The reader is referred to, for example, [1,7,10,19] for details.

In this note, we compare various algorithms for computing $W(G)$. In particular, the well-known BFS algorithm, adapted to compute $W(G)$ for arbitrary connected graphs, proves to be fast enough for practical purposes. Also, an algorithm that calculates the Wiener index of a tree in linear time is given. It improves an algorithm of Canfield, Robinson and Rouvray when programmed for the computer, or if used in hand-computations for small trees.

2. General graphs

One of the renowned algorithms which are mentioned in standard texts, say [1,7,10,19], is the *Floyd–Warshall* algorithm. Its input is the *adjacency matrix* $A(G) = [a_{ij}]$, where $a_{ij} = 1$ if vertices i and j are adjacent in G and $a_{ij} = 0$ otherwise. Its output is the distance matrix $D(G)$. The algorithm is very short.

2.1. FW ALGORITHM

(1) From the adjacency matrix $A(G)$ form the matrix $D(G)$ with the following properties:

```
        $d[i,j] = a[i,j]$ if $i$ is adjacent to $j$, or if $i = j$;
        $d[i,j] = n$ if $i$ is not adjacent to $j$ ($n$ is the number of vertices of $G$).

(2)     For each vertex $m$ of $G$ do
           for each vertex $i$ of $G$ different from $m$ with $d[i,m] < n$ do
              for each vertex $j$ of $G$ different from $m$ do
                 if $d[i,m] + d[m,j] < d[i,j]$ then $d[i,j] := d[i,m] + d[m,j]$.
```

The time complexity of this algorithm is $O(n^3)$. The algorithm is easy to program and is proved to be correct (see, for instance, [1,7,10]). In order to compute $W(G)$ by applying formula (1), another $O(n^2)$ additions are needed.

Below is an outline of another general algorithm for computation of $W(G)$. It is a little more involved, but its performance is better.

2.2. BFS ALGORITHM

(1) For each vertex i determine the BFS (Breadth First Search) tree $T(i)$. For each vertex j compute e_{ij} = the distance from i to j in $T(i)$.

(2) Since the distance d_{ij} is equal to the distance from i to j in $T(i)$, $e_{ij} = d_{ij}$, determine $W(G)$ by the formula

$$W(G) = \sum_{i=1}^{n} \sum_{j=1}^{i} e_{ij} .$$

For the BFS algorithm, the reader may consult [19] or the appendix to this note where the complete realization is given. The time complexity of this algorithm is $O(mn)$, where m is the number of edges of G. For each vertex, the BFS tree is computed in $O(m)$ steps. Note that chemical graphs have bounded degrees and thus $m = O(n)$. Thus, for chemical graphs the running time of this algorithm is bounded by $O(n^2)$. The reader is referred to sect. 4, where a comparison of various algorithms is given. It is shown that the BFS algorithm out-performs many other algorithms and is thus recommended for chemical applications.

There is an obvious question. In order to determine $W(G)$, do we need to compute the distance matrix $D(G)$? Or, more precisely, can we compute $W(G)$ faster than $D(G)$? Unfortunately, we do not know the answer to this question for general graphs. We do know that for certain families of graphs the answer to the latter question is *yes*. In the next section, we present a linear time algorithm for computing the Wiener index of a tree. Since $D(G)$ has $O(n^2)$ entries, computation of $W(G)$ for certain graphs can be faster than computation of $D(G)$.

3. The Wiener index of a tree

The main result of this section is an algorithm for computing $W(T)$ for an arbitrary tree T. This algorithm improves and simplifies a nice recursive algorithm of Canfield, Robinson and Rouvray [5]. The following observations are used in our algorithm.

3.1

In a tree T there is a unique shortest path between any two vertices.

3.2

Let $w(e)$ denote the number of shortest paths of T that pass through the specified edge $e \in E(T)$. Then

$$W(T) = \sum_{e \in E(G)} w(e) .$$

3.3

If T is a tree and $e \in E(T)$, let $n_1(e)$ and $n_2(e)$ denote the number of vertices in each of the two components of $T - \{e\}$, respectively. Then, $w(e) = n_1(e)n_2(e)$. Note also that for $i = 1, 2$: $n_i(e) = |V(T)| - n_{3-i}(e)$. (These results hold, more generally, in graphs that are not trees provided that e is a bridge of a graph.)

A tree T on n vertices has $n - 1$ edges. It can always be represented as a *rooted tree* (and this will be assumed henceforth). This means that a vertex $v_0 \in V(T)$ is distinguished and called the *root* of T. (Any vertex may be chosen for the root.) All other vertices are indexed as $v_1, v_2, \ldots, v_{n-1}$ in such a way that each v_i $(i \geqslant 1)$ has exactly one neighbor among $v_0, v_1, \ldots, v_{i-1}$. This unique neighbor is denoted by T_i and called the *predecessor* of v_i.

Our algorithm for computing $W(T)$ of an arbitrary tree T has the following steps.

3.4. LT ALGORITHM

(1) The (rooted) tree is input as an array T_i, $i = 1, 2, \ldots, n - 1$.

(2) For each vertex compute (in linear time) its indegree $\operatorname{ind}(v_i) = |\{j \mid v_i = T_j\}|$.

(3) Form a queue of all leaves of T, the vertices with $\operatorname{ind}(v_i) = 0$.

(4) Delete all leaves one after another and compute $w(e)$ for the corresponding deleted edge.

(5) Compute $W(T)$ using 3.2.

Each of the above steps requires at most linear time, hence the whole algorithm is linear. The appendix contains, among other things, Pascal realization of the above algorithm.

The following example shows that the above algorithm can be efficiently used in hand calculations of the Wiener index of a tree.

3.5. EXAMPLE

Let T be the tree of fig. 3.6. It is the graph of the molecule of 3,3,-dimethyl-4-isopropyloctane. The same tree was used as an illustrative worked example in [5] under the name 2,5-dimethyl-2-ethyl-4-propylhexane. T has $n = 13$ vertices. For each edge e of T, we determine the number of vertices in the smaller of the two subtrees $n_1(e)$ and then add $w(e) = n_1(e)(n - n_1(e))$ for each e from $E(T)$. Then

$$\begin{aligned} W(T) &= 6[1(13-1)] + 2[2(13-2)] + 2[3(13-3)] + 5(13-5) + 6(13-6) \\ &= 6 \times 12 + 2 \times 2 \times 11 + 2 \times 3 \times 10 + 5 \times 8 + 6 \times 7 = 258. \end{aligned}$$

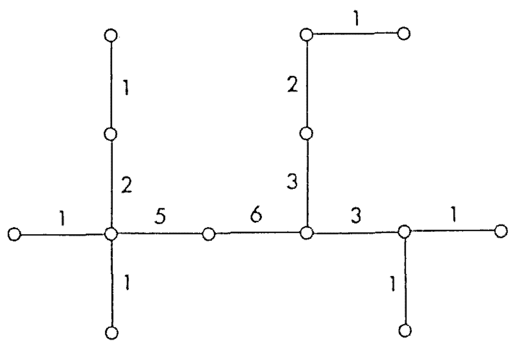

Fig. 3.6. The graph of the molecule of 3,3,-dimethyl-4-isopropyloctane. This is a tree T with $W(T) = 258$.

In passing, we mention that the LT algorithm can be extended to a linear time algorithm for computing the Wiener index of graphs with bounded block size.

4. Comparison of algorithms

Table 4.1 gives the results of practical tests performed for the following algorithms: Floyd–Warshall (FW), Matrix Multiplication (MM) algorithm of Müller et al., Breadth-First Search (BFS) algorithm as described above, and Linear Time (LT) tree algorithm.

Table 4.1

Comparison of four algorithms. Here, n is used to denote the number of vertices. The symbol T represents the fact that graphs used were trees. The symbol G denotes general graphs. The times are given in milliseconds and are averaged over various graphs. Clearly, the LT has the best performance for trees and the BFS for general graphs.

n	Tree/graph	FW	MM	BFS	LT
20	T	48	29	11	0.5
40	T	391	157	38	0.9
60	T	1265	424	84	1.3
80	T	2800	897	150	1.8
100	T	5145	1661	232	2.2
20	G	45	38	20	-
40	G	356	259	107	-
60	G	1169	810	296	-
80	G	2667	1866	536	-
100	G	5040	3785	845	-

5. The case of heteroatoms

The substitution of carbon atoms in a molecular graph with heteroatoms corresponds to weighting the edges and vertices of the graph. In this case, the BFS algorithm should be modified by applying the Dijkstra method [7,10] to compute the distances from a given vertex to all other vertices.

On the other hand, our algorithm LT carries to the weighted case with only a minor correction. Instead of $w(e) = n_1(e) \cdot n_2(e)$, one should use the weighted formula

$$w(e) = \text{weight}(e) \cdot n_1(e) \cdot n_2(e),$$

and at the end add one half of the weights of vertices to $W(T)$.

Appendix

In this appendix, all four programs that are compared in sect. 4 are listed. The following statements should be included in the main program.

```
const max = 101 ; { max number of vertices + 1 }

type  vertices = 0 .. max;
      tree     = array [vertices] of vertices;
      matrix   = array [vertices, vertices] of integer;
      queue    = array [vertices] of vertices;
```

```
function FloydWarshall (D: matrix; n: vertices): integer;

 { Calculates the Wiener index of a graph using the Floyd-Warshall
   algorithm. D is the distance matrix; at the time of calling, D
   must  be equal to the adjacency matrix of the graph;
   n  is the number of vertices }

var i, j, m: vertices;
   W: integer;

begin
 for i:=1 to n do
   for j:=1 to n do
     if (D[i,j] = 0) and (i <> j) then D[i,j] := n;
 for m:=1 to n do
   for i:=1 to n do if (i <> m) and (D[i,m] < n) then
   for j:=1 to n do if (j <> m) then
     if   D[i,m] + D[m,j] < D[i,j] then D[i,j] := D[i,m] + D[m,j];
 { D is equal to the distance matrix of the graph }
 W := 0;
 for i:=1 to n do
   for j:=1 to i-1 do W := W + D[i,j];
 FloydWarshall := W
end; { FloydWarshall }
```

```
function MM (D: matrix; n: integer): integer;

  { Matrix multiplication algorithm for computing the Wiener
    index of a graph. Used by Muller et al. }

var i, j, l, W: integer;

begin
  for i:=1 to n do for j:=1 to n do
    if (D[i,j] = 0) and (i <> j) then D[i,j] := n;
  l := 1;
  repeat
   for j:=1 to n do
    for i:=1 to n do begin
      if D[i,j] = l then begin
        for W:=1 to n do
          if D[W,i]+1 < D[W,j] then D[W,j] := D[W,i] + 1;
      end;
    end;
   l:=l+1;
  until l >= n-1;
  { compute W }
  W := 0;
  for i:=1 to n do for j:=1 to i-1 do W := W + D[i,j];
  MM := W
end; { MM }
```

```
function BFS (A: matrix; n:vertices): integer;

 { Calculate the Wiener index by using the Breadth-First Search
   starting with each of the vertices. A is the adjacency matrix
   of the graph, n is the number of vertices. }

var i, j, k: vertices;
   first, last: vertices;  { indices in the queue of open vertices }
   open: queue;  { queue of open vertices }
   found: array [vertices] of boolean;  { vertices already met }
   dist: array [vertices] of integer;   { distances from the root i }
   W: integer;  { Wiener index }

begin
 { Form the queues of neighbours within the adjacency matrix.
   The value  A[i,j] = k  will mean that the neighbour of i which
   follows  j  is the vertex  k, and A[i,j] = 0 will still represent
   non-adjacency except for A[i,i] being equal to the first
   neighbour of  i }
 for i:=1 to n do begin
   k := i;
   for j:=1 to n do if A[i,j] <> 0 then begin
     A[i,j] := k; k := j
   end;
   A[i,i] := k
 end;
 W := 0;  { The distances are added to W at the time of searching. }
 for i:=1 to n do begin  { BFS with the root i }
   { initialize }
   for j:=1 to n do found[j] := false;
   open[1] := i; last := 1; first := 1;
   dist[i] := 0; found[i] := true;
   while last >= first do begin
     j := open[first]; first := first + 1; W := W + dist[j];
     { open the neighbours of j }
     k := A[j,j];  { first neighbour of j }
     while k <> j do begin
       if not found[k] then begin
         last := last + 1; open[last] := k; found[k] := true;
         dist[k] := dist[j] + 1
       end;
       k := A[j,k]  { next neighbour of j }
     end
   end
 end; { for }
 BFS := W div 2
end; { BFS }



procedure transform (var A: matrix; var T: tree);

 { Create the rooted tree representation of a graph with the
   adjacency matrix  A. }

var v, u: vertices;
   open: queue;
   first, last: vertices;

begin
 T[1] := 0;  { vertex 1 is the root }
 open[1] := 1; last := 1; first := 1;
 while last >= first do begin
   v := open[first];
   first := first + 1;
   for u:=1 to n do if (A[u,v] <> 0) and (u <> T[v]) then begin
     last := last + 1; open[last] := u; T[u] := v
   end
 end
end; { transform }


function WItree (T: tree; n:vertices): integer;

 { Calculates the Wiener index of a tree. }
 { The tree T is represented as a rooted tree with root 1. }

var i, j, k, f: vertices;
   next: queue;  { queue of leaves }
   ind: array [vertices] of integer;  { indegrees }
   q: array [vertices] of integer;    { q[i] = n1(i,T[i]) }
   W: integer;
```

```
begin
 { compute indegrees }
 for i:=1 to n do ind[i] := 0;
 for i:=1 to n do ind[T[i]] := ind[T[i]] + 1;
 { form the queue of leaves }
 f := 0;
 for i:=1 to n do
   if ind[i] = 0 then begin next[i] := f; f := i end;
 { for each vertex determine w(i) }
 for i:=0 to n do q[i] := 0;
 for i:=1 to n do begin
   j := f; k := T[f];
   q[j] := q[j] + 1;
   q[k] := q[k] + q[j];
   f := next[f]; ind[k] := ind[k] - 1;
   if ind[k] = 0 then begin next[k] := f; f := k end
 end;
 { Calculate the Wiener index }
 W := 0;
 for i:=1 to n do   W := W + q[i] * (n - q[i]);
 WItree := W
end; { WItree }
```

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] M. Bersohn, A fast algorithm for calculation of the distance matrix of a molecule, J. Comput. Chem. 4(1983)110.

[3] D. Bonchev, O. Mekenyan, G. Protić and N. Trinajstić, J. Chromatogr. 176(1979)149.

[4] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications* (North-Holland, New York, 1976).

[5] E.R. Canfield, W.R. Robinson and D.H. Rouvray, Determination of the Wiener molecular branching index for the molecular tree, J. Comput. Chem. 6(1985)598.

[6] A. Graovac and T. Pisanski, On the Wiener index of a graph, submitted for publication.

[7] E.L. Lawler, *Combinatorial Optimization: Networks and Matroids* (Holt, Rinehart and Winston, New York, 1976).

[8] I. Motoc and A.T. Balaban, Topological indices: Intercorrelations, physical meaning, correlational ability, Rev. Roumanie Chim. 26(1981)593.

[9] W.R. Müller, K. Szymanski, J.V. Knop and N. Trinajstić, An algorithm for construction of the molecular distance matrix, J. Comput. Chem. 8(1987)170.

[10] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity* (Prentice-Hall, Englewood Cliffs, NJ, 1982).

[11] D. Papazova, M. Dimov and D. Bonchev, J. Chromatogr. 188(1980)297.

[12] D.H. Rouvray, The role of graph-theoretical invariants in chemistry, Congr. Numer 55 (1986)253.

[13] D.H. Rouvray, Should we have designs on topological indices? in: *Chemical Applications of Topology and Graph Theory*, ed. R.B. King (Elsevier, Amsterdam, 1983) pp. 159 – 177.

[14] D.H. Rouvray, Predicting chemistry from topology, Sci. Amer. 254(1986)40.

[15] D.H. Rouvray, The role of topological distance matrix in chemistry, in: *Mathematics and Computational Concepts in Chemistry*, ed. N. Trinajstić (Horwood, Chichester, UK, 1986) Ch. 25, pp. 295 – 306.
[16] D.H. Rouvray and B.C. Crafford, South Afr. J. Sci. 72(1976)74.
[17] D.H. Rouvray and W. Tatong, Z. Naturforsch. 41a(1986)1238.
[18] L.I. Stiel and G. Thodos, Amer. Inst. Chem. Eng. J. 8(1962)527.
[19] M.M. Syslo, N. Deo and J.S. Kowalik, *Discrete Optimization Algorithms* (Prentice-Hall, Englewood Cliffs, NJ, 1983).
[20] N. Trinajstić, *Chemical Graph Theory*, Vol. II (CRC Press, Florida, 1983) Ch. 4.
[21] H. Wiener, Structural determination of paraffin boiling points, J. Amer. Chem. Soc. 69 (1947)17.
[22] H. Wiener, J. Chem. Phys. 15(1947)766.