Bond Contributions to the Wiener Index[†]

Martin Juvan and Bojan Mohar*,[‡]

Department of Mathematics, University of Ljubljana, Jadranska 19, 61111 Ljubljana, Slovenia

Received October 31, 1994[⊗]

An efficient algorithm for computing bond contributions to the Wiener index of a (molecular) graph is presented.

1. INTRODUCTION

The *Wiener index* $W(G)$ of a (molecular) graph G with vertex set $V(G) = \{1,...,n\}$ is defined as the sum of distances between all pairs of distinct vertices i,j, $1 \le i < j \le n$.[1−3] The original motivation to consider this quantity was an observation by Wiener[1] that boiling points of paraffins correlate very strongly with this quantity. In subsequent studies, Wiener extended the applications of this index to other thermodynamic properties of alkanes such as heats of formation, heats of vaporization, molar refractions, and molar volumes.[4−7] The same behavior of the Wiener index was also reported in some later studies. For example, the Wiener index has been used to explain various chemical and physical properties of molecules[8] and to correlate the structure of molecules with their biological activity.[9] The reader is also referred to review articles 10−18.

It was proposed recently that the study of (appropriately defined) contributions of particular bonds (edges) to the Wiener index might be of some chemical interest.[19−22] Bond contributions are defined as follows. Recall that

$$W(G) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} \tag{1}$$

where d_{ij} ($i,j \in V(G)$) is the distance from i to j in the graph G, i.e., the length of a shortest path in G joining i and j. The matrix $D = [d_{ij}]_{i,j=1}^{n}$ is called the *distance matrix* of the graph G. The sum (1) can also be expressed as follows. For each pair $i, j \in V(G)$, choose one of shortest paths from i to j and denote it by P_{ij}. For $e \in E(G)$ let $\nu(e)$ be the number of paths P_{ij} that contain the edge e. Then

$$W(G) = \sum_{e \in E(G)} \nu(e) \tag{2}$$

This was observed already by Wiener[1] in case of trees. The relation (2) can be used to get a fast algorithm for computing the Wiener index of trees.[23] The expression (2) is not natural since in general $\nu(e)$ depends on the choice of the paths. Alternatively, let $\mathscr{P}_{ij}$ be the set of all paths of length d_{ij} from i to j in G. Denote by c_{ij} the number of all paths in $\mathscr{P}$ and by $c_{ij}(e)$ the number of paths from $\mathscr{P}_{ij}$ that contain the edge e. If we define

$$W^e = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{c_{ij}(e)}{c_{ij}} \tag{3}$$

then again

$$W(G) = \sum_{e \in E(G)} W^e \tag{4}$$

Quantities W^e are called *bond contributions* to the Wiener index since by (4), W^e is exactly the contribution of the edge e to $W(G)$.

Lukovits[24] devised an algorithm for computing bond contributions. His algorithm has complexity $\mathcal{O}(\text{diam}(G)n^3 + mn^2)$ where n and m denote the number of vertices and edges of the graph, respectively, and diam(G) is the diameter of the graph. This algorithm is appropriate for small graphs but, unfortunately, it is too time consuming for larger graphs. In this paper we present an improved algorithm whose time complexity is $\mathcal{O}(mn)$. With this algorithm we can easily handle graphs with several hundreds of vertices.

The paper is organized as follows. In section 2 our improved algorithm is described. Examples of its output are presented in section 3. Chart 1 contains the algorithm in the form of a Pascal program. The program can also be obtained on request by e-mail from the authors.

2. THE ALGORITHM

If we define

$$W_i^e = \sum_{j=1}^{n} \frac{c_{ij}(e)}{c_{ij}}$$

then

$$W^e = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{c_{ij}(e)}{c_{ij}} = \frac{1}{2}\sum_{i=1}^{n} W_i^e$$

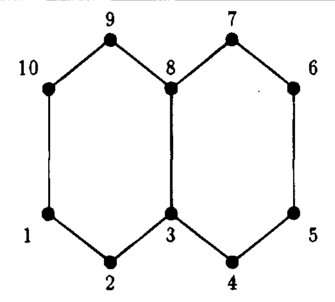
In order to efficiently compute W^e for an edge e, we compute W_i^e for every vertex i and add them together. To compute W_i^e, we fix a vertex i and first compute d_{ij} and c_{ij} for every vertex j. This is done by a breadth-first search of the given graph starting at i.[25] Initially, vertex i is put in the queue. At each step, the next vertex, call it v, is taken from the queue and all its neighbors are visited. Suppose that u is a neighbor of v. If u has not been visited before, then we set $d_{iu} = d_{iv} + 1$, and u is added at the end of the queue. Next, if $d_{iu} - d_{iv} = 1$ (and this can happen also if u was visited before), then some shortest paths from i to u pass through v.

[†] Authors' e-mail addresses: martin.juvan@uni-lj.si and bojan.mohar@uni-lj.si.

[‡] Supported in part by the Ministry of Science and Technology of Slovenia, Research Project P1-0210-101-94.

[⊗] Abstract published in *Advance ACS Abstracts,* February 1, 1995.

Table 1. Naphthalene and Its Representation

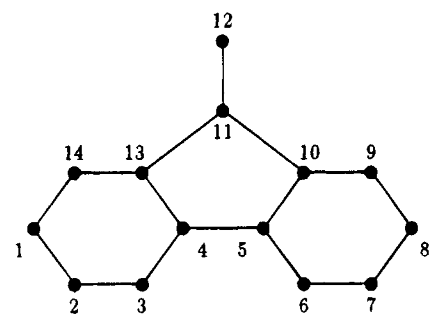


```
10
1   2   10   0
2   3   0
3   4   8    0
4   5   0
5   6   0
6   7   0
7   8   0
8   9   0
9   10  0
0
```

Table 2. Bond Contributions for Naphthalene

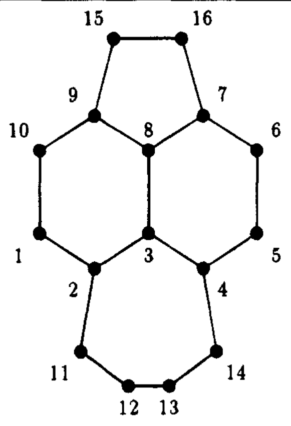
1−10	:	6.1667
1−2	:	8.5000
2−3	:	12.5000
3−8	:	12.6667
3−4	:	12.5000
4−5	:	8.5000
5−6	:	6.1667
6−7	:	8.5000
7−8	:	12.5000
8−9	:	12.5000
9−10	:	8.5000
$W = 109$		

Table 3. Bond Contributions for Dibenzfulvene



1−14	:	12.0000
1−2	:	7.0000
2−3	:	13.0000
3−4	:	21.0000
4−13	:	17.0000
4−5	:	27.0000
5−10	:	17.0000
5−6	:	21.0000
6−7	:	13.0000
7−8	:	7.0000
8−9	:	12.0000
9−10	:	20.0000
10−11	:	21.0000
11−13	:	21.0000
11−12	:	13.0000
13−14	:	20.0000
$W = 262$		

Table 4. Bond Contributions for Acepleiadylene



1−10	:	14.1667
1−2	:	18.5000
2−11	:	24.0000
2−3	:	29.5000
3−8	:	34.6667
3−4	:	29.5000
4−14	:	24.0000
4−5	:	18.5000
5−6	:	14.1667
6−7	:	15.5000
7−16	:	14.0000
7−8	:	22.5000
8−9	:	22.5000
9−15	:	14.0000
9−10	:	15.5000
11−12	:	15.0000
12−13	:	10.0000
13−14	:	15.0000
15−16	:	7.0000
$W = 358$		

The number of such paths is obviously equal to the number of c_{iv} of shortest paths from i to v, and this number has been computed in previous iterations. To make sure that c_{iu} will be determined we just add the number of these paths to the number of shortest paths from i to u obtained so far using previously considered edges. When the queue becomes empty, breadth-first search is completed. This way distances d_{ij} and the number of shortest paths c_{ij} is computed for all vertices j.

The second part of the main loop computes the numbers W_i^e. This is done by traversing vertices of the graph in the opposite order as visited by the breadth-first search. Therefore, at each step we are sure that vertices that are more distant from i than the current vertex have already been considered. At each vertex, denote it by v, we determine W_i^e for all edges that have v as their endpoint and whose other endpoint is more distant from i than v. Moreover, the sum of all these values W_i^e is stored in S_v. Suppose that we are considering an edge e with endpoints v and u such that $d_{iu} - d_{iv} = 1$. At this step we would like to compute the number W_i^e. Let P be a shortest path between i and some other vertex and suppose that P contains the edge e. Then P either has u as its endpoint, or contains (exactly) one of the edges starting at u and having the other endpoint at distance $d_{iu} + 1$ from i. The contribution of the former to W_i^e is $c_{iu}(e)/c_{iu} = c_{iv}/c_{iu}$. To calculate the contribution of the latter paths, let f be an edge of the above form. Since exactly $c_{iu}(e) = c_{iv}$ shortest paths between i and u contain e, the contribution in W_i^e of paths containing f equals $W_i^f\, c_{iv}/c_{iu}$. Summing up these contributions over all such edges f, we get $S_u c_{iv}/c_{iu}$, the number that can easily be computed from already known quantities.

Let us now estimate the time complexity of the above algorithm. Clearly, the main loop is repeated n times. In each iteration, first, a breadth-first search of the graph is performed. Since during the search each edge is considered only twice, once for each of the endpoints, and only a constant number of operations is performed in each visit, the time complexity of the search is $\mathcal{O}(m)$. The same arguments apply also when estimating the time complexity of the second part of the main loop. Therefore, the time complexity of the whole algorithm is $\mathcal{O}(mn)$.

Our algorithm also works for disconnected graphs and graphs with loops and multiple edges. It can also be adapted to work for graphs of molecules containing heteroatoms, i.e., graphs in which each edge is associated with a positive real number representing its weight. In this case the notion of a distance between vertices and derived quantities have to be redefined accordingly.

The described algorithm is presented in the form of a Pascal program in Chart 1. The program is written in Turbo Pascal. The only nonstandard Pascal features used in the program are the type **string**, which is used in the program for storing the name of the input file, and the procedure **assign**, which assigns the name of the (external) input file to a file variable. If some other Pascal compiler is used, these two features have to be modified appropriately.

Input data for the graph of the σ-electron skeleton of naphthalene (or of Z/E-decalin in case of fully-saturated hydrocarbons) are shown in Table 1. In general, the first line contains the number of vertices of the graph. The following lines contain the description of the neighbors of the vertices. Each description starts with a label of a vertex followed by labels of neighbors and ends with 0. To shorten the representation only neighbors with greater labels need

Chart 1

```
program BondContributions;
  const
    maxN = 100; {maximal number of vertices}
    maxM = 1000; {maximal number of edges}
  type
    vertices = 1..maxN;
    edges = 1..maxM;
    Pvertex = ↑vertex;
    vertex = record
      e: edges;
      u: vertices;
      next: Pvertex;
    end;
  var
    inp: text; name: string; {input file}
    n,m: integer; {number of vertices and edges}
    G: array[vertices] of Pvertex;
    WI: real; {Wiener index}
    visited: array[vertices] of boolean;
    d,c: array[vertices,vertices] of integer; {distances and number of shortest paths}
    W: array[edges] of real; {bond contributions}
    Q: array[vertices] of vertices; {queue for storing vertices during BFS}
    S: array[vertices] of real;
    i,j,k: integer;
    x: real;
    edge: Pvertex;
    v: vertices;
begin
  write('Input file: '); readln(name);
  assign(inp,name); reset(inp);
  readln(inp,n); {number of vertices}
  for i:=1 to n do G[i]:=nil;
  for i:=1 to n do for j:=1 to n do
    begin c[i,j]:=0; d[i,j]:=0; end;
  {read input data and build a graph}
  read(inp,i); m:=0;
  while i<>0 do begin
    read(inp,j);
    while j<>0 do begin
      if j>i then begin
        m:=m+1;
        new(edge); edge↑.next:=G[i]; G[i]:=edge;
        edge↑.u:=j; edge↑.e:=m;
        new(edge); edge↑.next:=G[j]; G[j]:=edge;
        edge↑.u:=i; edge↑.e:=m;
      end; {if}
      read(inp,j);
    end;
    read(inp,i);
  end; {while}
  for i:=1 to m do W[i]:=0;
  for i:=1 to n do begin {main loop}
    c[i,i]:=1;
    for j:=1 to n do visited[j]:=false;
    Q[1]:=i; j:=0; k:=1; visited[i]:=true;
    while j<k do begin {BFS}
      j:=j+1; v:=Q[j]; edge:=G[v];
      while edge<>nil do with edge↑ do begin
        if not visited[u] then begin
          d[i,u]:=d[i,v]+1; visited[u]:=true;
          k:=k+1; Q[k]:=u;
        end;
        if d[i,u]>d[i,v] then c[i,u]:=c[i,u]+c[i,v];
        edge:=next;
      end; {while edge<>nil}
    end; {BFS}
    for j:=n downto 1 do begin {second part}
      v:=Q[j]; edge:=G[v]; S[v]:=0;
      while edge<>nil do with edge↑ do begin
        if d[i,v]<d[i,u] then begin
          x:=(c[i,v]/c[i,u])*(S[u]+1);
          W[e]:=W[e]+x/2; S[v]:=S[v]+x;
        end;
        edge:=next;
      end; {while}
    end; {for j}
  end; {main loop}

  WI:=0;
  for i:=1 to n do begin
    edge:=G[i];
    while edge<>nil do with edge↑ do begin
      if u>i then begin writeln(i:3,'-',u:3,' : ',W[e]:8:4); WI:=WI+W[e]; end;
      edge:=next;
    end; {while}
  end; {for}
  writeln('W = ',WI:5:0);
end.
```

to be listed. The value 0 in place of a vertex label identifies the end of the input.

3. EXAMPLES

In this section some results obtained by our program are presented. For the graph of naphthalene (see Table 1) we computed the bond contributions shown in Table 2.

Further examples are graphs of dibenzfulvene and acepleiadylene (or of perhydro-9-methylfluorene and perhydroacepleiadylene, respectively, if we consider saturated molecules). The results are shown in Tables 3 and 4.

As expected, the results show that in all our examples the interior edges contribute more to the Wiener index then the outer ones.

REFERENCES AND NOTES

(1) Wiener, H. Structural Determination of Paraffin Boiling Points. *J. Am. Chem. Soc.* **1947**, *69*, 17−20.

(2) Hosoya, H. Topological Index. A Newly Proposed Quantity Characterizing the Topological Nature of Structural Isomers of Saturated Hydrocarbons. *Bull. Chem. Soc. Jpn.* **1971**, *44*, 2332−2339.

(3) Trinajstić, N. *Chemical Graph Theory*, 2nd revised ed.; CRC Press: Boca Raton, FL, 1992; Chapter 4.

(4) Wiener, H. Correlation of Heats of Isomerization and Differences in Heats of Vaporization of Isomers among the Paraffin Hydrocarbons. *J. Am. Chem. Soc.* **1947**, *69*, 2636−2638.

(5) Wiener, H. Influence of Interatomic Forces on Paraffin Properties. *J. Chem. Phys.* **1947**, *15*, 766.

(6) Wiener, H. Vapour Pressure Temperature Relations Among the Branched Paraffin Hydrocarbons. *J. Chem. Phys.* **1948**, *15*, 425−430.

(7) Wiener, H. Relationship of Physical Properties of Isomeric Alkanes to Molecular Structure, Surface Tension, Specific Dispersion and Critical Solution Temperature in Aniline. *J. Phys. Colloidal Chem.* **1948**, *52*, 1082−1089.

(8) *Mathematics and Computational Concepts in Chemistry*; Trinajstić, N., Ed.; Ellis Horwood: Chichester, U.K., 1986.

(9) Kier, L. B.; Hall, L. H. *Molecular Connectivity in Chemistry and Drug Research*; Academic Press: New York, 1976.

(10) Rouvray, D. H. In *Chemical Applications of Graph Theory*; Balaban, A. T., Ed.; Academic Press: London, 1976; p 175.

(11) Rouvray, D. H. Predicting Chemistry from Topology. *Sci. Am.* **1986**, *24*, 40−47.

(12) Rouvray, D. H. In *Mathematics and Computational Concepts in Chemistry*; Trinajstić, N., Ed.; Ellis Horwood: Chichester, U.K., 1986; p 295.

(13) Mihalić, Z.; Veljan, D.; Amić, D.; Nikolić, S.; Plavšić, D.; Trinajstić, N. The Distance Matrix in Chemistry. *J. Math. Chem.* **1992**, *11*, 223−258.

(14) Gutman, I.; Yeh, Y. N.; Lee, S. L.; Luo, Y. L. Some Recent Results in the Theory of the Wiener Number. *Indian J. Chem.* **1993**, *32A*, 651−661.

(15) Bošnjak, N.; Adler, N.; Perić, M.; Trinajstić, N. In *Modelling of Structure and Properties of Molecules*; Maksić, Z. B., Ed.; Ellis Horwood: Chichester, U.K., 1987; p 103.

(16) Katritsky, A. R.; Gordeeva, G. V. Traditional Topological Indices vs Electronic, Geometrical, and Combined Molecular Descriptors in QSAR/QSPR Research. *J. Chem. Inf. Comput. Sci.* **1993**, *33*, 835−857.

(17) Balaban, A. T.; Motoc, J.; Bonchev, D.; Mekenyan, O. Topological Indices for Structure−Activity Correlations. *Top. Curr. Chem.* **1983**, *114*, 21−55.

(18) Balaban, A. T.; Niculescu−Duvaz, I.; Simon, Z. *Acta Pharm. Jugosl.* **1987**, *37*, 7−36.

(19) Lukovits, I. Wiener Indices and Partition Coefficients of Unsaturated Hydrocarbons. *Quant. Struct.−Act. Relat.* **1990**, *9*, 227−231.

(20) Lukovits, I. Correlation Between Components of the Wiener Index and Partition Coefficients of Hydrocarbons. *Int. J. Quantum Chem., Quantum Biol. Symp.* **1992**, *19*, 217−223.

(21) Pisanski, T.; Žerovnik, J. Weights on Edges of Chemical Graphs Determined by Paths. *J. Chem. Inf. Comput. Sci.* **1994**, *34*, 395−397.

(22) Lukovits, I.; Gutman, I. Edge-Decomposition of the Wiener Number. *MATCH*, in press.

(23) Mohar, B.; Pisanski, T. How to Compute the Wiener Index of a Graph. *J. Math. Chem.* **1988**, *2*, 267−277.

(24) Lukovits, I. An Algorithm for Computation of Bond Contributions of the Wiener Index. preprint.

(25) Sedgewick, R. *Algorithms*; Addison-Wesley: Reading, MA, 1988.

CI9401231