Copyright © 2009 American Scientific Publishers
All rights reserved
Printed in the United States of America

Journal of
Computational and Theoretical Nanoscience
Vol. 6, 1–4, 2009

A Numerical Method for Computing the Wiener Index of One-Heptagonal Carbon Nanocone

M. A. Alipour[1,*] and A. R. Ashrafi[2]

[1] *Department of Computer Engineering, Faculty of Engineering, University of Kashan, Kashan 87317-51167, I. R. Iran*
[2] *Institute of Nanoscience and Nanotechnology, University of Kashan, Kashan 87317-51167, I. R. Iran*

The Wiener index of a graph G is defined as the sum of all distances between distinct vertices of the graph G. Here, for {x,y} $\subseteq$ G, distance between x and y, $d_G(x, y)$, is the length of a minimal path connecting x and y. In this paper, the Wiener index of the carbon nanocone $CNC_7[n]$ is computed.

Keywords: One-Heptagonal Carbon Nanocone, Wiener Index.

1. INTRODUCTION

In the past years, nanostructures involving carbon have been the focus of an intense research activity which is driven to a large extent by the quest for new materials with specific applications. One pentagonal carbon nanocones originally discovered by Ge and Sattler in 1994.[1] These are constructed from a graphene sheet by removing a 60° wedge and joining the edges produces a cone with a single pentagonal defect at the apex.

The inclusion of the heptagons in the hexagonal lattice leads to the appearance of negative curvature, Figure 1. The single sevenfold in the plain graphene lattice was theoretically studied but this situation, unfortunately, has not been observed in the experiment yet.[2]

We first describe some notations which will be kept throughout. Let G be a simple molecular graph without directed and multiple edges and without loops, the vertex and edge-sets of which are represented by $V(G)$ and $E(G)$, respectively. A topological index of a graph G is a numeric quantity related to G. The oldest topological index is the Wiener index which introduced by Harold Wiener.[3]

The most important works on the geometric structures of nanotubes, nanotori and their topological indices were done by Diudea and his co-authors.[4–9] One of the present authors (ARA) continued this program to calculate the Wiener index of some other nanostructures.[10–15] In some research papers they computed the Wiener index of some nanotubes and nanotori. We encourage the reader to consult[16–19] and references therein for background material as well as basic computational techniques. In this paper, we continue this program to compute the Wiener index of one-pentagonal carbon nanocone $CNC_7[n]$, Figure 1. Our notation is standard and mainly taken from standard books of graph theory and the books of Trinajestic.[20]

*Author to whom correspondence should be addressed.

2. MAIN RESULTS AND DISCUSSION

The adjacency matrix of a molecular graph G with n vertices is an $n \times n$ matrix $A = [a_{ij}]$ defined by: $a_{ij} = 1$, if vertices i and j are connected by an edge and, $a_{ij} = 0$, otherwise. The distance matrix $D = [d_{ij}]$ of G is another $n \times n$ matrix defined by d_{ij} is the length of a minimum path connecting vertices i and j, $i \neq j$, and zero otherwise.

In this section, a Java program is presented which is useful for computing the Wiener index of a nanocone. We apply this program to compute the adjacency and distance matrices of the molecular graph of nanocone $CNC_7[n]$, Figure 1. In Table I, we calculate the Wiener index of $CNC_7[n]$, for $1 \leq n \leq 15$. Then by curve fitting method, we will find a polynomial of degree ≤ 6, through the values of Table I. This polynomial will be the Wiener index of nanocone. Our method is general and can be applied to compute the PI and Szeged indices of nanostructures.

Curve fitting is finding a curve which has the best fit to a series of data points and possibly other constraints. DataFit is a science and engineering tool that simplifies the tasks of data plotting, regression analysis (curve fitting) and statistical analysis. This package is applied for solving our problem. The interested readers can check "http://www.oakdaleengr.com/datafit.htm" for further information on this tool. We are interested in curve fitting by polynomial functions. We search for the best polynomial to fit data of Table I. We investigated 240 different mode to find a good polynomial for curve fitting the Wiener index of $CNC_7[n]$. By these calculation, we

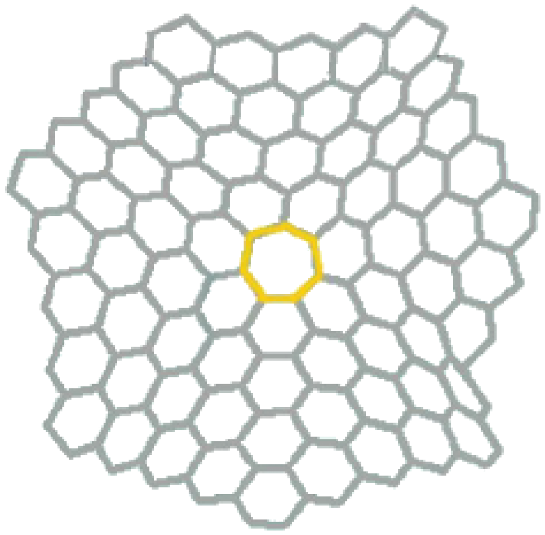

Fig. 1. The one-heptagonal carbon nanocone $CNC_7[4]$.

Table I. Values of $W(CNC_7[n])$, for $1 \le n \le 15$.

n	$W(CNC_7[n])$	n	$W(CNC_7[n])$
1	42	8	1556772
2	1477	9	2806482
3	11410	10	4754169
4	48370	11	7658266
5	148022	12	11834326
6	368879	13	17660734
7	798014	14	25584419

believe the Wiener index of this nanocone is computed as $W(CNC_7[n]) = an^5 + bn^4 + cn^3 + dn^2 + en + f$, where

$$a = 47.5999999999996, \quad b = 0.0000000000129$$

$$c = -5.83333333348213, \quad d = 0.0000000007293$$

$$e = 0.233333331850055, \quad f = 0.0000000009420$$

In the Figure 2, the standard errors of other models for curve fitting are depicted. From this figure, one can see that the first model is the best of them. So it is natural to convince $W(CNC_7[n]) = an^5 + bn^4 + cn^3 + dn^2 + en + f$, where the values $a, b, c, d, e,$ and f are determined as above.

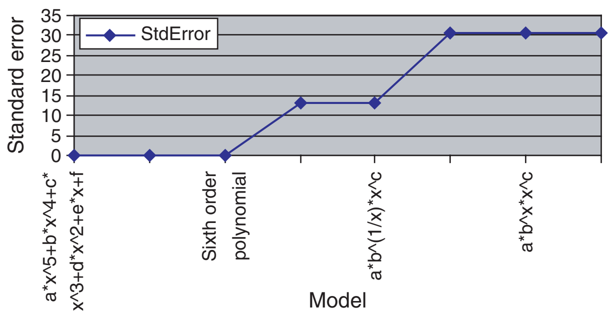

Fig. 2. Standard error in curve fitting models.

A Java Program for Computing the Wiener index of One-Heptagonal Carbon Nanocone $CNC_7[n]$

```
import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
public class Hepta {
  public Hepta()
    {for(int i= 0; i<= layers; i++)
      LayersInfo[i] = new layer();
     for(int i= 0; i< num_vertices; i++)
      vertices[i] = new node();
    }
  public int layers = 14;
  public int num_vertices = 2000;
  public int edges = 0;
  public int adj[][] = new int [num_vertices + 1]
                        ×[num_vertices + 1];
  public int dist[][] = new int [num_vertices + 1]
                        ×[num_vertices + 1];
  public node vertices[]= new node [num_vertices + 1];
  public layer LayersInfo[] = new layer [layers + 1];
public void CalculateFloyd(String S)
{
  int i, j, k;
  for (i= 0; i<= this.LayersInfo[this.layers].tail; i++)
  for (j = 0; j <= this.LayersInfo[this.layers].tail; j++)
        {if(this.adj[i][j] == 0) this.dist[i][j] = 9999;
                         else this.dist[i][j] =this.adj[i][j];
        this.dist[i][i] = 0;
        }
for (k = 1; k <= this.LayersInfo[this.layers].tail; k++)
for (i = 1; i <= this.LayersInfo[this.layers].tail; i++)
  for (j = 1; j <= this.LayersInfo[this.layers].tail; j++)
     if (this.dist[i][k] + this.dist[k][j] < this.dist[i][j])
          this.dist[i][j] = this.dist[i][k] + this.dist[k][j];
try{
 FileOutputStream o = new FileOutputStream(S);
  BufferedOutputStream bs = new
                             BufferedOutputStream(o);
  DataOutputStream ps = new DataOutputStream(bs);
  for (i = 1; i <= this.LayersInfo[this.layers].tail; i++)
   for (j = 1; j <= this.LayersInfo[this.layers].tail; j++)
        ps.writeInt(this.dist[i][j]);
  ps.close();
  o.close();}
catch(Exception e){ }
}
public void LoadFloyd(String S)
 {
  int i, j;
  try{
  FileInputStream fr=new FileInputStream(S);
  DataInputStream br=new DataInputStream(fr);
   for (i = 1; i <= this.LayersInfo[this.layers].tail; i++)
   for (j = 1; j <= this.LayersInfo[this.layers].tail; j++)
          this.dist[i][j] = br.readInt();
   br.close();
   fr.close();}
   catch (Exception e){ }
     }
void initnodes()
  {
  for (int i = 1; i <= num_vertices; i++)
     {vertices[i].degree = 0;
      vertices[i].number = i;
     }
  }
void set(int i, int j, int value)
  {
     edges++;
     adj[i][j] = value;
     adj[j][i] = value;
     vertices[i].degree++;
     vertices[j].degree++;
  }
void calculatelayers()
  {
  LayersInfo[1].number = 7;
  LayersInfo[1].head = 1;
  LayersInfo[1].tail = 7;
  for (int i = 2; i <= layers; i++)
    {
      LayersInfo[i].number
           =LayersInfo[i−1].number + 14;
      LayersInfo[i].head = LayersInfo[i−1].tail + 1;
      LayersInfo[i].tail = LayersInfo[i].head
                        + LayersInfo[i].number−1;
    }
  }
void initlayer()
  {
   calculatelayers();
   for (int i = 1; i < layers; i++)
     {
     for(int j = LayersInfo[i].head; j < LayersInfo[i].tail;
                                                    j++)
     set(j, j+1, 1);
     set(LayersInfo[i].head,LayersInfo[i].tail,1);
     }
  }
int findfirstdegree2(int l)
  {
  for(int i = LayersInfo[l].head; i <= LayersInfo[l].tail;
                                                    i++)
  if(vertices[i].degree == 2) return i;
  return −1;
  }
public void initnetwork()
   {
   initlayer();
   int temp,iterator;
   for(int i = 2; i <= layers; i++)
    {
     temp = findfirstdegree2(i − 1);
     set(temp,LayersInfo[i].head,1);
     iterator = LayersInfo[i].head;
     for(int k = 1; k <= 7; k++)
     {
     iterator = iterator + 3;
     temp = findfirstdegree2(i − 1);
     if(temp > 0)
     set(temp,iterator,1);
     for(int l = 1; l <= i − 2; l++)
          {
          iterator = iterator + 2;
          temp = findfirstdegree2(i − 1);
          if(temp > 0)
          set(temp,iterator,1);
          }
     }
    }
   }
}
```

Acknowledgment: This research was in part supported by a grant from the Center of Excellence of Algebraic Methods and Applications of Isfahan University of Technology, Isfahan, Iran.

References

1. M. Ge and K. Sattler, *Chem. Phys. Lett.* 220, 192 (**1994**).
2. D. R. Nelson and L. Peliti, *J. Phys. (Paris)* 48, 1085 (**1987**).
3. H. Wiener, *J. Am. Chem. Soc.* 69, 17 (**1947**).
4. M. V. Diudea, M. Stefu, B. Pârv, and P. E. John, *Croat. Chem. Acta* 77, 111 (**2004**).
5. M. V. Diudea, B. Parv, and E. C. Kirby, *MATCH Commun. Math. Comput. Chem.* 47, 53 (**2003**).
6. M. V. Diudea, *Bull. Chem. Soc. Japan* 75, 487 (**2002**).
7. M. V. Diudea, *MATCH Commun. Math. Comput. Chem.* 45, 109 (**2002**).
8. M. V. Diudea and P. E. John, *MATCH Commun. Math. Comput. Chem.* 44, 103 (**2001**).
9. M. V. Diudea and E. C. Kirby, *Fullerene Sci. Technol.* 9, 445 (**2001**).
10. S. Yousefi and A. R. Ashrafi, *J. Math. Chem.* 42, 1031 (**2007**).
11. A. Iranmanesh and B. Soleimani, *MATCH Commun. Math. Comput. Chem.* 57, 251 (**2007**).
12. A. R. Ashrafi and S. Yousefi, *Nanoscale Res. Lett.* 2, 202 (**2007**).
13. S. Yousefi and A. R. Ashrafi, *MATCH Commun. Math. Comput. Chem.* 56, 169 (**2006**).
14. A. R. Ashrafi and S. Yousefi, *MATCH Commun. Math. Comput. Chem.* 57, 403 (**2007**).

15. A. R. Ashrafi, B. Manoochehrian, and H. Yousefi-Azari, *Util. Math.* 71, 97 (**2006**).

16. A. A. Dobrynin, R. Entringer, and I. Gutman, *Acta Appl. Math.* 66, 211 (**2001**).

17. A. A. Dobrynin, I. Gutman, S. Klavžar, and P. Zigert, *Acta Appl. Math.* 72, 247 (**2002**).

18. A. Heydari and B. Taeri, *MATCH Commun. Math. Comput. Chem.* 57, 463 (**2007**).

19. M. Eliasi and B. Taeri, *Journal of Serb. Chem. Soc.* 73, 311 (**2008**).

20. N. Trinajstic, Chemical Graph Theory, CRC Press, Boca Raton, FL (**1992**).

Received: 23 April 2007. Accepted: 30 April 2007.