University of Ljubljana
Institute of Mathematics, Physics and Mechanics
Department of Mathematics
Jadranska 19, 1000 Ljubljana, Slovenia

Preprint series, Vol. 40 (2002), 821

COMPUTING THE WEIGHTED WIENER AND SZEGED NUMBER ON WEIGHTED CACTUS GRAPHS IN LINEAR TIME

Blaž Zmazek Janez Žerovnik

ISSN 1318-4865

April 24, 2002

Ljubljana, April 24, 2002

....

Computing the weighted Wiener and Szeged number on weighted cactus graphs in linear time[⋆]

Blaž Zmazek [a,b⋆⋆] *and Janez Žerovnik*[a,b⋆⋆⋆]

[a] *Faculty of Mechanical Engineering, University of Maribor, Smetanova 17, SI-2000 Maribor, Slovenia.*

[b] *Department of Theoretical Computer Science, IMFM, Jadranska 19, SI-1111 Ljubljana, Slovenia.*

Cactus is a graph in which every edge lies on at most one cycle. Linear algorithms for computing the weighted Wiener and Szeged numbers on weighted cactus graphs are given. Graphs with weighted vertices and edges correspond to molecular graphs with heteroatoms.

⋆ Supported in part by the Ministry of Science and Technology of Slovenia under the grant J2-1015-0101.

⋆⋆ E-mail: `blaz.zmazek@uni-mb.si`.

⋆⋆⋆ E-mail: `janez.zerovnik@imfm.uni-lj.si`.

1 Introduction

A *topological index* is a numerical quantity derived in an unambigous manner from the structural graph of a molecule. These indices are graph invariants, which usually reflect molecular size and shape.

The first non-trivial topological index in chemistry was introduced by H. Wiener[1] in 1947 to study boiling points of paraffins (for historical data see, for instance Ref. 2). Since then, the Wiener number (also called the Wiener index) has been used to explain various chemical and physical properties of molecules and to correlate the structure of molecules to their biological activity.[3] The research interest in Wiener index and related indices is still considerable.[4, 5]

Wiener originally defined his index on trees and studied its use for correlations of physicochemical properties of alkanes, alcohols, amines, and other analogous compounds. In arbitrary tree, every edge is a bridge, i.e. after deletion of the edge, the graph is no more connected. The contribution to the Wiener number of an edge was taken to be the product of the numbers of vertices in the two connected components. This number also equals the number of all shortest paths in the tree which go through the edge.[6] Therefore the usual generalization of the Wiener number on arbitrary graphs is defined to be the sum of all distances in a graph. Recalling and generalizing the original definition, edge contributions to the Wiener number were studied in Refs. 7–9.

Another natural generalization was recently put forward and called the Szeged number, Sz.[10] Now the weights of edges are taken to be the product of the numbers of vertices closer to the two endpoints of the edge. For reasons to introduce the Szeged index and for basic properties of Sz see Refs. 10,11. Formulas or special algorithms for Szeged index of several families of graphs were proposed recently.[12–14]

The motivation for study of weighted graphs and their topological indices is the fact that the substitution of carbon atoms in a molecular graph with heteroatoms corresponds to weighting the vertices and edges of the graph.

The main result of this paper are linear algorithms for the weighted Wiener and the weighted Szeged number on weighted cactus graphs. Cactus graphs are generalized trees in which cycles are allowed, but any two cycles have at most one vertex in common.

2 Definitions

A *weighted graph* $G = (V, E, w, \lambda)$ is a combinatorial object consisting of an arbitrary set $V = V(G)$ of *vertices*, a set $E = E(G)$ of unordered pairs $\{x, y\} = xy$ of distinct vertices of G called *edges*, and two *weighting functions*, w and λ. $w : V(G) \mapsto \mathbb{R}$ assigns positive real numbers (weights) to vertices and $\lambda : E(G) \mapsto \mathbb{R}$ assigns positive real numbers (lengths) to edges.

A *simple path* from x to y is a finite sequence of distinct vertices $P = x_0, x_1, \ldots, x_\ell$ such that each pair x_{i-1}, x_i is connected by an edge and $x_0 = x$ and $x_\ell = y$. The *length* of the path is the sum of lengths of its edges, $l(P) = \sum_{i=1}^{\ell} \lambda(x_{i-1}x_i)$. For any pair of vertices x, y we define the *distance* $d(x, y)$ to be the minimum of lengths over all paths between x and y. If there is no such path, we set $d(x, y) = \infty$.

A graph G is *connected*, if $d(x, y) < \infty$ for any pair of vertices x, y. A vertex v is a *cut vertex* if after removing v and all edges incident to it the resulting graph is not connected. A graph without a cut vertex is called *nonseparable*. A *block* is a maximal nonseparable graph. A *cycle* is an induced subgraph which is connected and in which every vertex is of degree two. A *cactus* is a graph in which every block of three or more vertices is a cycle. Alternatively, a cactus is a connected graph in which two cycles have at most one vertex in common. For usual graph theoretical terminology not defined here see, for instance, Ref. 15.

The *weighted Wiener number* of a weighted graph G studied here is

$$W(G) = \sum_{u>v} w(u)w(v)d(u, v). \qquad (1)$$

It seems that weighted Wiener number has not been studied frequently in the literature. A definition, analogous to (1) was used in Refs. 16 and 17 for vertex weighted graphs. A different definition in which the "weights" of atoms are added to the sum of distances is used in Ref. 18. We find the definition (1) more natural from mathematical point of view and look forward to learn about possible chemical interpretations of this and other definitions. It may be worth to remark that in the definition of Ref. 18 the weighted Wiener number is simply the usual Wiener number plus the sum of weights of vertices and hence it can be computed by standard algorithms.[19]

The *weighted Szeged number* of a weighted graph G is defined as

$$Sz(G) = \sum_{e=uv} s_u(e)s_v(e)\lambda(e), \qquad (2)$$

where $s_u(e) = \sum_{x \in V, d(x,u) < d(x,v)} w(x)$ (and $s_v(e) = \sum_{x \in V, d(x,u) > d(x,v)} w(x)$).

The definitions used here are clearly generalizations of the usual definitions for (unweighted) Wiener and Szeged numbers. More precisely, if all weights of vertices are 1 and all lengths of edges are 1, then $W(G)$ and $Sz(G)$ are the usual Wiener and Szeged numbers.

Lemma 1 *For a weighted graph G, $W(G) = \sum_{e=uv} \lambda(e) \times \sum_{P^*_{a,b} \ni e} \frac{1}{n^*(a,b)} w(a)w(b)$, where $P^*_{a,b}$ is a shortest path between a and b and $n^*(a,b)$ is the number of shortest paths with endpoints a and b.*

Proof. To see this it is enough to sum up the contributions of each edge to W in two different ways. Each pair a, b of vertices contributes $w(a)w(b)d(a,b)$ to the Wiener number. This contribution can be either regarded as a contribution of the pair a, b or it can be divided to $n^*(a,b)$ path contributions which can furthermore be regarded as a sum of edge contribution along the path. Since $d(a,b)$ is the sum of edge weights along a shortest path, the contribution of edge e caused by a and b is

$$\lambda(e) \frac{1}{n^*(a,b)} w(a)w(b).$$

An edge contributes as many times as it appears on various shortest paths. Here the vertices are weighted, so one has to take into account the weights of both terminal vertices. Let us note in passing that the sum of contributions of the edge e,

$$\omega^*(e) := \sum_{P^*_{a,b} \ni e} \frac{1}{n^*(a,b)} w(a)w(b).$$

can be called the *shortest path weight distribution*, generalizing the ideas of Refs. 7–9.

Hence one can sum up the lengths of all shortest paths (as in (1)), or, equivalently, sum up the contributions of all edges. $\square$

Recall that on a tree, there is a unique shortest path between any pair of vertices. Hence $n^*(a,b) = 1$ for all a, b which implies that the definition of the weighted Wiener number and the weighted Szeged number are equivalent on trees.

Lemma 2 *For a weighted tree T, $W(T) = Sz(T)$.*

This lemma is a generalization of a well-known result for unweighted Wiener number, which was already known to Wiener.[20]

For a later reference note that on a cactus graph $n^*(a,b)$ can only have value 1 or 2. Clearly, $n^*(a,b) = 2$ exactly when a and b are opposite vertices of a cycle (i.e. $d(a,b) = d$ and a, b are vertices of a cycle of girth $2d$).

3 DFS based representation of a cactus

If in a cactus cycles are considered as (super)-vertices, there is a unique shortest path between any two vertices.

Any cactus G can always be represented as a *rooted cactus*. This means that a vertex $v_0 \in V(G)$ is distinguished and called the *root* of G. (Any vertex may be chosen for the root.) All other vertices are indexed as $v_1, v_2, \ldots, v_{n-1}$ in a DFS (depth first search) order. DFS is a standard procedure for searching a graph, see for example Ref. 21. It can easily be seen that in each cycle there is exactly one vertex v_j, which is the first vertex in the DFS order on the cycle C, containing the edge $v_i v_j$. For this reason we call the vertex v_j the *root* of the cycle C. Clearly, each cycle in a cactus has a unique root, determined as a first vertex on the cycle in the DFS order. Moreover, each vertex on a cactus G which is a successor of a vertex with higher index is the root vertex of one or more cycles. Hence, all cycles in a cactus are determined with the root and its successor.

The rooted cactus can be represented with two arrays T_i and R_i, $i = 0, 2, \ldots, n-1$, where T_i denotes the unique predecessor of vertex v_i in the rooted tree, constructed with the DFS. The element R_i denotes the root vertex of the cycle containing v_i. If v_i does not lie on a cycle then $R_i = v_i$ and if v_i is on a cycle rooted at v_j, then $R_i = v_j \neq v_i$. Note that the array R describes all cycles in the cactus.

It is clear that when traversing the graph with a DFS algorithm, one can also obtain array $ind(v_i) := |\{j \; ; \; v_i = T_j\}|$. The meaning of $ind(v_i)$ is the number of sons of v_i. Leaves of the tree have no son directing to them. Hence $ind(leave) = 0$. This can be done within the time complexity of BFS, $O(m)$.

After a DFS run, each vertex v_i can be regarded as a root of a subcactus, which we will denote by G_i.

From the discussion above we infer that after a DFS run, one can get a representation of a cactus and, furthermore, one can decide whether a given graph is a cactus or not. More formally,

Lemma 3 *Let G be a cactus graph. In a DFS run, each new vertex v_i can have at most one neighbor among the visited vertices and the neighbor can only be on the direct path from the root v_0 to v_i.*

Proof. If, for some vertex two such neighbors existed, then there will be edges on two (or more cycles). Contradiction. A back neighbor which is not on a direct path from the root v_0 to v_i was visited already and hence the vertex v_i should have been visited as its son. Contradiction. □

Hence, if G is not a cactus graph, this can be observed while traversing G in the DFS order.

Lemma 4 *Cactus graphs can be recognized in linear time.*

Proof. DFS algorithm takes $O(m)$ time. □

4 The algorithms

We will give the algorithms Sz and W for computing the indices (Szeged and Wiener number) of a rooted subcactus. Algorithms for computing Sz and W differ only in Step 3 while the Steps 1 and 2 are identical for both algorithms. Recall that by Lemma 1 the Wiener number can be computed as a sum of contributions over all edges of a graph.

With each vertex v_i of the graph G the value of V_i will be computed, for which

$$V_i = \sum_{j \in G_i} w(v_j)$$

will eventually hold. V_i will thus be the sum of weights of the subcactus G_i rooted at v_i.

Both algorithms consist of three steps. First, a representation of the given weighted cactus is found. Then we compute V_i for each vertex. In order to compute the V_i we start with edges incident to leaves and continue in reversed DFS order up to root. In the third step we compute the contributions of edges (cycles) to the invariant starting from root to leaves.

1. Find a representation of the (rooted) cactus G.
2. Traverse the cactus G in the reversed DFS order and compute the V_i.
3. Traverse the cactus G in the DFS order and compute the contributions of edges (cycles) to the invariant.

We now give more details of each step.

Step 1: Cactus recognition

Run a DFS on G. By discussion above, it is clear that we can compute T_i, R_i and $ind(v_i) := |\{j \ ; \ v_i = T_j\}|$ in time $O(m)$.

Step 2: Computation of V_i

Traverse the DFS tree in the reversed DFS order. Compute the temporal value V_i for a leave v_i and reduce the number of unvisited sons for its father by one. If the father has no more unvisited sons, put it in a FIFO (first in first out) queue. The computation of temporal V_i is the weight of v_i plus the sum of V_j of its sons:

$$V_i = w(v_i) + \textstyle\sum_{T_j = v_i} V_j.$$

Lemma 5 *After Step 2, for every vertex v_i, V_i is the sum of weights of all vertices of the subcactus rooted at v_i. More formally,*

$$V_i = \textstyle\sum_{v_j \in G_i} w(v_j).$$

Proof. It follows from the definition of V_i. □

Lemma 6 *Step 2 can be computed in $O(m)$ time.*

Proof. Clear. □

Step 3a: Computation of the edge contributions to $W(G)$

Traverse the DFS tree in the DFS order. Compute the edge contributions to the $W(G)$.

Observe that V_0 is the sum of weights of all vertices of G.

```
1. $W = 0$
2. For each vertex $v_i$ (in the DFS order) do
   – For each son $v_j$, for which $v_i v_j$ is not an edge of a cycle of $G$, (i.e., if $R_j = v_j$ and $T_j = v_i$) set
        $W_e := V_j(V_0 - V_j)\lambda(e)$
        $W := W + W_e$.
```

– For each son v_j of a root vertex v_i ($R_i = v_i$), for which v_iv_j is an edge of a cycle C_j of G, (i.e., if $R_j = v_i$ and $T_j = v_i$), compute the contributions of the cycle as follows. Mark the vertices of the cycle C in the DFS order with $u_0, \dots, u_{e-1}$. For each vertex u_i on the cycle compute L_i, the sum of weights of vertices for which the vertex u_i is the nearest vertex on the cycle C, using the values V_j and let $|C| = \sum_{i=9}^{e-1} \lambda(u_iu_{i+1})$ be the girth of the cycle.

```
• $W_C := 0$; (weighted Wiener number of the cycle $C$)
  $f = l := 0$; (loop counters - $f$ is first, $l$ is last vertex - are computed modulo $e$)
  $Cont = 0$; (contribution of the current vertices $\{u_f, u_{f+1}, \dots, u_l\}$)
  $V := 0$; (sum of weights of $\{u_f, u_{f+1}, \dots, u_l\}$)
  $d := 0$ (distance from $u_f$ to $u_l$)
• Repeat
  (compute the contributions of edges on the cycle which lies on shortest paths that enter on the cycle throught the vertex $v_f$ and continue in a DFS order)
  (a) While $d + \lambda(u_lu_{l+1}) \leq \frac{|C|}{2}$ do begin
        ∗ $l := l + 1$;
        ∗ $V := V + L_l$;
        ∗ $d := d + \lambda(u_{l-1}u_l)$;
        ∗ $Cont := Cont + L_l d$
      end(while) (Find the new last vertex $u_l$ on the cycle $C_j$ with a shortest path from $u_f$ in a DFS order.)
  (b) $W_C := W_C + L_f Cont$;
  (c) If $d = \frac{|C|}{2}$ then $W_C := W_C - \frac{1}{2}L_fL_ld$;
  (d) $Cont := Cont - V\lambda(u_fu_{f+1})$;
  (e) $V := V - L_{f+1}$;
  (f) $d := d - \lambda(u_fu_{f+1})$;
  (g) $f := f + 1$
  Until $f = 0$.
• $W := W + W_C$.
```

Step 3b: Computation of the edge contributions to $Sz(G)$

Traverse the DFS tree in the DFS order. Compute the edge contributions to the $Sz(G)$.

Observe that $V := V_0$ is the sum of weights of all vertices of G.

```
1. $Sz = 0$
2. For each vertex $v_i$ (in the DFS order) do
   – For each son $v_j$, for which $v_iv_j$ is not an edge of a cycle of $G$, (i.e., if
     $R_j = v_j$ and $T_j = v_i$) set
         $Sz_e := V_j(V - V_j)\lambda(e)$
         $Sz := Sz + Sz_e$.
   – For each son $v_j$ of a root vertex $v_i$ ($R_i = v_i$), for which $v_iv_j$ is an edge of
     a cycle $C_j$ of $G$, (i.e., if $R_j = v_i$ and $T_j = v_i$), compute the contributions
     of the cycle as follows. Mark the vertices of the cycle $C$ in the DFS order
     with $u_0, \ldots, u_{e-1}$. For each vertex $u_i$ on the cycle compute $L_i$, the sum
     of weights of vertices for which the vertex $u_i$ is the nearest vertex on the
     cycle $C$, using the values $V_j$ and let $|C| = \sum_{i=9}^{e-1} \lambda(u_iu_{i+1})$ be the girth of the
     cycle.
     • $Sz_C := 0$; (weighted Szeged number of the cycle $C$)
       $f = l := 0$; (loop counters - $f$ is first, $l$ is last vertex - are computed
       modulo $e$)
       $V_R := 0$; $V_L := L_0$; (sum of weights of vertices right (R) and left (L)
       of an edge)
       $d := \frac{\lambda(u_0u_{e-1})}{2}$;
     • Repeat
       (compute the contribution of the edge $u_{f-1}u_f$)
       (a) While $d + \lambda(u_lu_{l+1}) \leq \frac{|C|}{2}$ do begin
             * $l := l + 1$
             * $V_L := V_L + L_l$;
             * $d := d + \lambda(u_{l-1}u_l)$;
           end(while)
       (b) $V_R := V - V_L$;
       (c) If $d = \frac{|C|}{2}$ then $Sz_C := Sz_C + V_R(V_L - L_l)\lambda(u_{f-1}u_f)$
           else $Sz_C := Sz_C + V_RV_L\lambda(u_{f-1}u_f)$;
```

```
        (d) $V_L := V_L - V_f$;
        (e) $d := d - \frac{\lambda(u_f u_{f+1}) + \lambda(u_{f-1} u_f)}{2}$;
        (f) $f := f + 1$
        Until $f = 0$.
    • $Sz := Sz + Sz_C$.
```

Example 1 Compute weighted Wiener and Szeged number of path from Fig. 1.

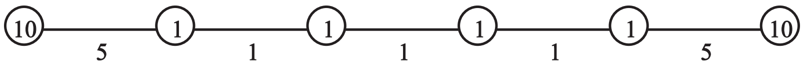

Figure 1 *Example 1.*

Traversing the path we get the following table

i	0	1	2	3	4	5
T_i	–	0	1	2	3	4
$\lambda(v_i T_i)$	0	5	1	1	1	5
V_i	24	14	13	12	11	10
$V_i(V_0 - V_i)\lambda(v_i T_i)$	0	700	143	144	143	700

Summarizing the last row of the table we get the Wiener number of the path $W = 1830$, which also coincides with the Szeged number.

Example 2 Compute weighted Wiener and Szeged number on a cycle from Fig. 2. Let the top vertex of the cycle in Fig. 2 be also the root vertex. All steps of computation of Wiener and Szeged numbers are written in the tables bellow (the entries are computed columnwise from left to right):

Step	f	0				1	2	3	4	5		6
(a)	l	1	2	3	4		5	6	0	1	2	3
	V	2	3	5	6		4	4	4	5	6	6
	d	1	2	3	5		5	5	5	4	5	5
	$Cont$	2	4	10	15		10	16	13	13	18	22
$(b),(c)$	W_C				12.5	30.5	38	60	70.5		86	**120**
(d)	$Cont$				9	5	6	8	5		12	10
(e)	V				4	3	2	3	3		4	5
(f)	d				4	3	3	3	3		4	3

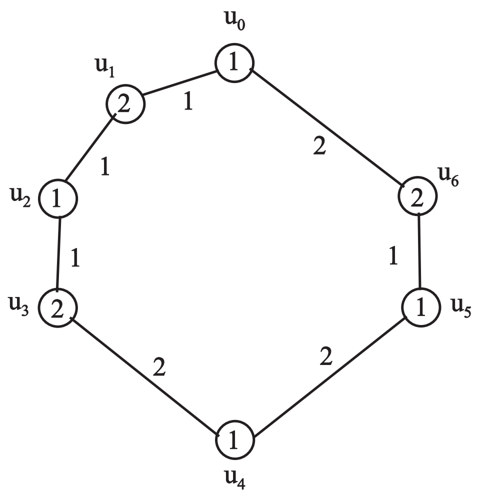

Figure 2 *Example 2.*

Step	f	0			1	2	3	4	5		6
(a)	l	1	2	3	4		5	6	0	1	2
	V_L	3	4	6	6		4	4	4	6	6
	d	2	3	4	4.5		4.5	4	4	**5**	4.5
(b)	V_R			4	4	6	6	6		4	4
(c)	Sz_C			48	72	96	120	168		200	**224**
(d)	V_L			5	4	3	2	3		5	4
(e)	d			2.5	3.5	2.5	3	2		3.5	3

In Fig. 3 we illustrate the seven steps in the computation of Wiener (Szeged) number of the cycle.

Summarizing, we read from the tables the values $W(C) = 120$ and $Sz(C) = 224$.

Lemma 7 *W_e and W_C are correctly computed. Each edge is considered exactly once.*

Proof. (sketch) After line (a) in 2. of Step 3a, the value of V is the sum of weights of vertices which are at distance less or equal to $|C|/2$ from the vertex u_f. Similarly, the value *Cont* is the weighted sum

$$Cont = \sum_{l:=1}^{f} w(u_l)d(u_f, u_l) = \sum_{l:=1}^{f}\sum_{i:=1}^{l} w(u_l)\lambda(u_{i-1}, u_i).$$

Hence

$$w(u_f)Cont = \sum_{l:=1}^{f} \sum_{i:=1}^{l} w(u_f)w(u_l)\lambda(u_{i-1}, u_i).$$

is the sum of contributions of all paths with the first vertex u_f which are oriented coherent with the orientation of the cycle defined by the labeling $u_0, u_1, \ldots$.

If there are vertices of distance exactly $|C|/2$ then there are exactly two shortest paths between them and the contribution has to be divided by 2 (hence step (c)).

Finally observe that in the repeat-until loop each shortest path shorter than $|C|/2$ is considered exactly once and that each shortest path of length $|C|/2$ is considered exactly twice. We omit the details. □

Lemma 8 *Sz_e and Sz_C are correctly computed. Each edge is considered exactly once.*

Proof. (sketch) After line (a) in 2. of Step 3b, the value of V_L is the sum of weights of vertices which are at distance less or equal to $|C|/2$ from the center of edge $u_{f-1}u_f$. If there is a vertex exactly at distance $|C|/2$ it is neither closer to u_{f-1} nor to u_f and hence this case has to be treated separately (see (c)). Clearly, in the repeat-until loop each edge is considered exactly once. We omit the details. □

Lemma 9 *Step 3a can be computed in $O(m)$ time. Step 3b can be computed in $O(m)$ time.*

Proof. (idea) Each cycle is considered only once (when its root is traversed). Furthermore, the computation on the cycle is linear in the length of the cycle, which easily follows by analysis of the repeat-until loop. □

Recalling the complexity of Steps 1 to 3 we conclude that both algorithms run in linear time.

Proposition 10 *The above algorithms compute the weighted Wiener and the weighted Szeged numbers on a weighted cactus in $O(m)$ time.*

REFERENCES

1. H.Wiener, Correlation of Heats of Isomerization, and Diferences in Heats of Vaporization of Isomers, Among the Paraffin Hydrocarbons, J.Amer.Chem.Soc **69** (1947) 2636-2638.

2. D.H.Rouvray, Should we have designs on topological indices?, (in *Chemical Applications of Topology and Graph Theory* B.B.King (ed.)), Vol. **28**, (Elsevier, Amsterdam, 1984) 159-177.
3. L.B.Kier and L.H.Hall, *Molecular Connectivity in Chemistry and Drug Research* (Academic Press, New York, 1976).
4. Discrete Mathematics **80** (1997). (Special issue on the Wiener index.)
5. MATCH **35** (1997). (Special issue on the Wiener index.)
6. H.Hosoya, Topological index. A newly proposed quantity characterizing the topological nature of structural isomers of saturated hydrocarbons, Bul. Chem. Soc. Japan 44 (1971) 2332-2339.
7. I. Lukovits, Correlation Between Components of the Wiener Index and Partition Coefficients of Hydrocarbons, Int. J. Quantum Chem.: Quantum Biol. Symp. **19** (1992) 217-223.
8. I. Lukovits, The Generalized Wiener Index for Molecules Containing Double Bonds and the Partition Coefficients, Rep.Mol.Theory **1** (1990) 127-131.
9. T.Pisanski and J.Žerovnik, Weights on Edges of Chemical Graphs Determined by Paths, J.Chem.Inf.Comput.Sci. **34** (1994) 395–397.
10. I. Gutman, Formula for the Wiener Number of trees and Its Extension to Graphs containing cycles, Graph Theory Notes New York **27** (1994) 9-15.
11. P.V. Khadikar, N.V. Deshpande, P.P.Kale, A. Dobrynin, I. Gutman, G. Dömötör, The Szeged index and an analogy to the Wiener index, J. Chem. Inf. Comput. Sci. **35** (1995) 547-550.
12. I. Gutman, S. Klavžar, An Algorithm for the Calculation of the Szeged Index of Benzenoid Hydrocarbons, J. Chem. Inf. Comput. Sci. **35** (1995) 1011-1014.
13. S.Klavžar, A.Rajapakse and I.Gutman, The Szeged and the Wiener index of graphs, Appl. Math. Lett. 9 (1996) 45-49.
14. J. Žerovnik, Computing the Szeged index, Croat. Chem. Acta 69 (1996) 837–843.
15. N.Trinajstić, Chemical Graph Theory, CRC Press, Boca Raton, FL. 1992.
16. S.Klavžar and I.Gutman, Wiener number of vertex-weighted graphs and a chemical application, Discrete Appl. Math. 80 (1997) 73-81.
17. V.Chepoi and S.Klavžar, Distances in benzenoid systems: Further developments, Discrete Math. 192 (1998) 27-39.
18. P.Senn, The computation of the distance matrix and the Wiener index for graphs of arbitrary complexity with weighted vertices and edges, Comput. Chem. 12 (1988) 219-227.
19. B.Mohar, T.Pisanski, How to Compute the Wiener Index of a Graph, J. Math. Chem. **2** (1988) 267-277.
20. H.Wiener, Structural determination of paraffin boiling points, J. Amer. Chem. Soc. **69** (1947) 17-20.
21. N.L.Biggs, Discrete Mathematics, Claredon Press, Oxford 1989.

This article was processed using the LaTeX macro package with Croatica style

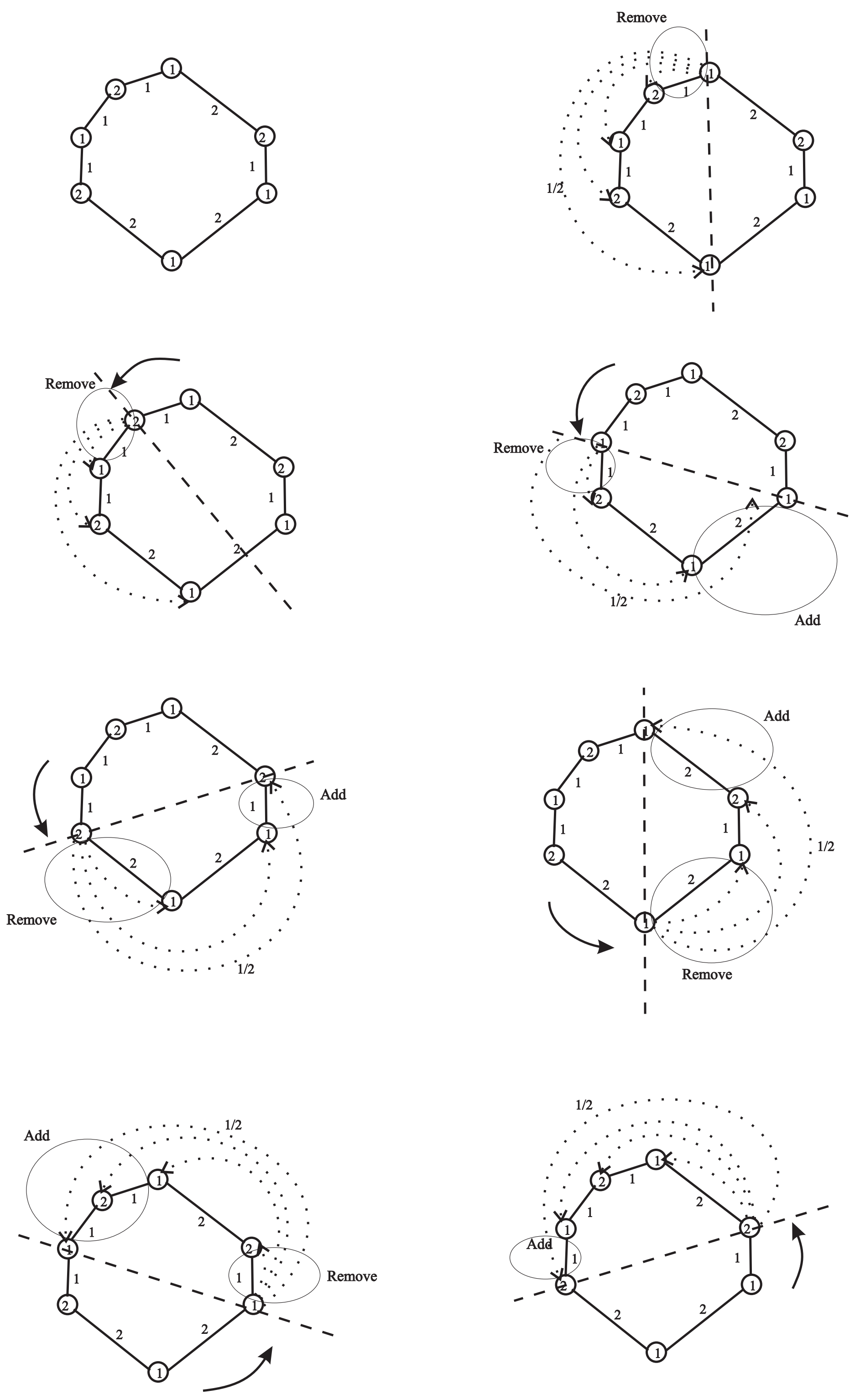

Figure 3 Seven steps in computation of Wiener and Szeged number of the cycle.