Viswanathan K. Iyer

Wiener index of graphs
Wiener index of graphs
Some Graph-Theoretic and Computational Aspects
Preface

This book is intended to be a motivational exposure to Wiener index of graphs as viewed from a modest mathematical perspective. The book can be understood with foundations in graph theory, combinatorics and algorithms. Each chapter is on one theme and each stands independently. The book can be recommended as a supplementary reading for graduate students. The recent mathematical literature abounds with many interesting results pertaining to Wiener index and its variants. Many of these results are from combinatorial and computational viewpoints. It is natural to reason that efforts to consolidate these should provide a better insight and value for practical applications of the associated concepts and may also provide a unified picture of the results related to distance in graphs. Many readers may like to point out that precise significance and interpretation of Wiener index and other topological indices require a more in-depth analysis than what this book can offer. However the role of Wiener index from a chemical outlook appears to be much more specialized and it is considered to be a current inter-disciplinary area of research. Many researchers may also feel that the book could have included a longer survey with more pointers to available literature – that is at present beyond the scope of this book!

The main chapters contained in this book have appeared in print in modified forms and as parts of the Ph.D. dissertation submitted by me in the year 2006 at the National Institute of Technology, Tiruchirapalli. I remain grateful to professors Dr. R. Balakrishnan and Dr. N. Sridharan for their clarifications of many notions and S. Prasanna for working with me;
I thank them for allowing me to use their ideas without which this work would not have materialized. Thanks are also due to Dr. (Mrs.) H. Thiyagarajan for her support in many ways. For the demanding proof-reading work I would like to thank P. V. Subba Reddy. Finally, I would also like to thank the publishers of this book, in particular H. Olsen, for agreeing to review this work and resolve copyright issues through their good offices within a short span of time. Thanks are also due to A. Bernhardt for guiding me through to make the corrections in the previous print version.

_Tiruchirapalli_  
_K. Viswanathan Iyer_  
_21st March 2012_
Contents

1 Wiener index of a graph .......................................................... 1
   1.1 Introduction ...................................................................... 1
   1.1.1 Significance of Wiener index ........................................ 2
   1.2 A quick literature survey .................................................. 6
   1.3 Techniques for computing Wiener index ............................. 12
      1.3.1 \( G \) characterized by parameters ............................... 13
      1.3.2 Algorithmic determination of \( W(G) \) ......................... 18
   1.4 A decomposition algorithm ............................................... 20
   1.5 Chapter summaries .......................................................... 24

2 \( W(G) \) of two chemical trees ............................................... 26
   2.1 Introduction ...................................................................... 26
   2.2 Wiener index of complete binary trees .............................. 27
   2.3 Wiener index of \( A_{2k+1} \) and \( A_{2k} \) ............................... 29

3 Wiener index of odd graphs ................................................... 35
   3.1 Odd graphs ...................................................................... 35
   3.2 Wiener index of \( O_k \) ........................................................ 36
CONTENTS

8 Future directions 85

8.1 Introduction ................................. 85
8.2 Wiener index and Laplacian eigenvalues .......... 86
8.3 Other graph theoretical descriptors ............. 87
8.4 Inverse problem on Wiener index ............... 88
Chapter 1

Wiener index of a graph

1.1 Introduction

Let $G = (V(G), E(G))$ be a simple connected undirected graph. For subsequent discussions we will always consider such graphs only. The Wiener index or Wiener number $W(G)$ of $G$ is defined as

$$W(G) = \frac{1}{2} \sum_{u \in V(G)} \sum_{v \in V(G)} d_G(u, v). \quad (1.1)$$

Here, $d_G(u, v)$ (or simply $d(u, v)$ when no confusion arises) denotes the distance between the vertices $u$ and $v$ in $G$ (for standard definitions and notations in graph theory see [BR00], [GY06], [Har69] or [Wes96]). Given a vertex $u \in V(G)$, we define $d^+(u, G)$ as

$$d^+(u, G) = \sum_{x \in V(G)} d(u, x). \quad (1.2)$$
CHAPTER 1. WIENER INDEX OF A GRAPH

It then follows that \( W(G) = \frac{1}{2} \sum_{u \in V(G)} d^+(u, G) \). If \( G \cong T \) a tree then (1.1) can be shown to be equivalent to the expression for \( W(T) \) given by

\[
W(T) = \sum_{e \in E(T)} n_1(e)n_2(e) \tag{1.3}
\]

where, if \( e = xy \ (x, y \in V(T)) \) then \( n_1(e) = |S_1| \) and \( n_2(e) = |S_2| \) where

\[
S_1 = \{ w | w \in V(T) \text{ and } d_T(w, x) < d_T(w, y) \} \quad \text{and} \quad S_2 = \{ w | w \in V(T) \text{ and } d_T(w, y) < d_T(w, x) \}.
\]

In other words \( n_1(e)n_2(e) = p(e) \), say, is the product of the orders of the two components of \( T \setminus xy \). \( p(e) \) is also referred to as the path number of the edge \( e \). The graph parameter \( W(G) \) is also known by other names in the literature such as transmission, total status and sum of all distances (see for example [JT03]). A quantity related to \( W(G) \) is the mean distance or the average distance \( \mu(G) \) defined by \( \mu(G) = \frac{W(G)}{|V(G)|^2} \).

1.1.1 Significance of Wiener index

From the contemporary literature one can conclude that the primary importance of Wiener index is in chemical graph theory. One can also see its relevance in the study of distance in graphs and the possible associated applications. We mention some of these in the following paragraphs.

(A) Applications in Chemistry:

Within the framework of valence bond theory, chemical graph theory (see [Bal85], [GP86], [Ran03], [Tri92] for example) views organic compounds or equivalently their molecular structures as graphs, often referred to as molecular graphs wherein atoms are represented by vertices and covalent chemical bonds by edges. Double and triple bonds,
1.1. INTRODUCTION

if any, are approximated by a single edge in forming the molecular graph. In many studies it is also assumed that, the molecular graphs are hydrogen-suppressed; that is, there will be no vertices corresponding to $H$-atoms and no edges corresponding to bonds between $H$-atoms to other atoms. One interesting problem in chemical graph theory is then, to capture the molecular structure as a graph-theoretic descriptor (otherwise known as topological index or descriptor) to be computed from the underlying molecular graph and relate it meaningfully to selected physico-chemical and pharmacological properties of the corresponding chemical compound. This idea is the main theme in the modern studies on Quantitative Structure-Property Relations (QSPR) and on Quantitative Structure-Activity Relations (QSAR), a popular computational biology paradigm used in modern drug design. In this context, a property connotes a physico-chemical property (e.g., boiling point) and an activity connotes a biological or pharmacological property (e.g., toxicity of a drug). In QSAR and QSPR studies (see [DBe99], [De01]), one tries to construct a simple – usually linear – mathematical model that reproduces the experimentally available data pertaining to a class of chemical compounds, expecting that the model will correctly – in a statistical sense – predict the data or a property for new or unknown i.e., not-yet-synthesized compounds. It has been pointed out that studies based on QSPR and QSAR are often over-simplified and their theoretical bases are not yet well-developed or understood. Furthermore, many molecular properties require ad hoc combinations of several topological descriptors so that meaningful correlations to physico-chemical properties can be obtained. On
the positive side, it is claimed that \textit{QSPR} and \textit{QSAR} techniques are much less expensive than any other method of gaining information about the properties of chemical compounds. In recent times, \textit{QSAR} has become a popular computational biology paradigm in modern drug design and it has been shown to be powerful in quickly identifying lead drug candidates. Also modern technology in combinatorial chemistry uses topological indices in the screening of combinatorial libraries (see [IK02] also). For some of the important research work in Mathematical Chemistry during the period 1970 – 2005, including characterization of chemical structures, reference [Bon05] may be consulted. A recent update on the role of topological indices in drug development research is provided in [EU01].

Roughly, topological indices differentiate between molecular graphs based on their size, degree of branching and overall structure, thus suggesting a graph similarity measure among graphs with a fixed order. All topological indices based on molecular graphs fall into the following four categories as per their on their logical derivations.

- Those derived from the adjacency matrix: e.g., the total adjacency index, the Zagreb group indices, the Randic connectivity index, the Platt index.

- Those based on the topological distance matrix: e.g., the Wiener index, the polarity number, the mean square distance, the Altenburg polynomial.

- Information theoretic indices: e.g., the Shannon index, the chromatic information index, the Merrifield and Simmons index.
Among all the topological indices, Wiener index is the most popular one (e.g., see [XG06]) and it is extensively used in QSAR studies. The quantity $W(G)$ is named after the chemist Harold Wiener who seems to be the first to study the correlation between $W(G)$ and physico-chemical properties of paraffins i.e., hydrocarbons, where $G$ is taken to be the molecular graph of the corresponding chemical compound [Wie47]. Wiener’s definition corresponds to (1.3) – the definition was suggested for molecular graphs that are trees. Equation (1.1) is a natural variant suggested as a possible generalization for graphs that need not be trees. Other logical variants and modifications e.g., hyper-Wiener index, degree-distance are also known in the mathematical literature. It appears that for majority of chemical applications, the study of Wiener index is restricted to acyclic molecular graphs although mathematical studies are reported for hexagonal systems also (see [DGKv02]).

(B) **Application in networks:**

For different types of networks arising in computer architecture, the quantity $\mu(G)$ is of interest since it is a coarse measure of the average distance between any two vertices in $G$. In turn, this implies that $\mu(G)$ is a measure of the average-distance traversed by messages transmitted by processes comprising a parallel computation within the processor-interconnection network (as opposed to the network diameter) and hence the average communication delay of the network for an application. Stated differently, in running a parallel algorithm on a given architecture as specified by a graph, if we can estimate the num-
ber of messages exchanged, then we can estimate the average communication delay incurred by the algorithm for an application.

(C) **Significance in graph theory:**

The computer program *GRAFFITI* [FW86] made the following attractive conjecture: $\mu(G) \leq \alpha(G)$, where $\alpha(G)$ is the *independence number* of $G$. The weaker inequality $\mu(G) - 1 < \alpha(G)$ was proved in [FW86]. The conjecture (giving an efficiently computable lower bound for $\alpha(G)$) itself was later proved in [Chu88]. For a tree $T$, it is known that $W(L(T)) = W(T) - \binom{n}{2}$, where $L(T)$ denotes the line graph of $T$. The iterated line graph $L^k(G)$ is defined as $L^k(G) = L(L^{k-1}(G))$, where $k \geq 1$ and $L^0(G) = G$. For a tree $T$, the size of $L^k(T)$ rapidly increases as $k$ tends to infinity. Therefore, for sufficiently large $k$, we have $W(T) < W(L^k(T))$, except for paths and $K_{1,3}$. Similar results are available in the literature (see for example [Dan94], [Win90]).

### 1.2 A quick literature survey

The wide majority of previous and on-going studies related to $W(G)$ and $\mu(G)$ have focus along one or more of the following directions:

(a) **Computation of the Wiener index:** This includes the algorithmic determination of $W(G)$ or $\mu(G)$ and obtaining closed-form expressions for different graph classes of $G$.

(b) **Relationship to other graph parameters:** $W(G)$ and $\mu(G)$ bear interesting relationships to other graphical invariants. Many results are available when $G$ is restricted to a tree.


1.2. A QUICK LITERATURE SURVEY

(c) Upper/lower bounds and asymptotics: Upper/lower bounds on $W(G)$ or $\mu(G)$ are usually given in terms of order, size and other quantities pertaining to $G$. There are some results pertaining to asymptotics related to $W(G)$ when $G$ is a randomly generated tree.

(d) Characterization of extremal structures: With the use of one or more parameters of $G$ such as order and size, one can determine the extremal graphs having maximum and minimum possible $W(G)$.

(e) Inverse Wiener index problem: The inverse problem is: for a given integer $n$, does there exist (and if so, find) a graph $G$ with $W(G) = n$? If $G$ is restricted to be a tree with specified properties, interesting problems arise.

Below, we briefly outline some of the studies on Wiener index and average distance. Most of these studies can be categorized into results from a theoretical perspective and those driven by more practical considerations. The reader should consult the current literature for further study of other results and the associated inter-connections and for the more recent developments.

A relatively early work in [EJS76] proved that among all trees of a given order $n$, the Wiener index is the maximum for the path $P_n$ and it is minimum for the star $K_{1,n-1}$ (see also exercise 6.23 in [Lov93]). That is, for any tree $T_n$ (different from $P_n$ and $K_{1,n-1}$) on $n$ vertices $W(P_n) > W(T_n) > W(K_{1,n-1})$. This suggested that Wiener index may be used to order isomers according to the extent of branching.

In [Tru85], it is shown that every rational number $r \geq 1$ is the mean distance of some graph. The same result was proved independently in [Hen86].
CHAPTER 1. WIENER INDEX OF A GRAPH

In [Dan94] depending on the order of $G$ and $\alpha(G)$, a sharp upper bound for $\mu(G)$ is given. A question posed by P. Erdős asked for bounds on $\alpha(G)$ of a graph $G$ with a given $\mu(G)$. This question is answered in [Dan94]. Therein, upper and lower bounds on $\mu(G)$ are also given depending on the matching number.

The class of trees known as $r$-ary dendrimers on $n$-nodes ($n, r$ being positive integers) denoted by $D_{n,r}$ is defined recursively thus: (a) $D_{1,r}$ comprises the single node labeled 1. (b) $D_{n,r}$ has the vertex set $\{1, \ldots, n\}$. It is obtained by attaching a leaf labeled $n$, to the smallest labeled vertex of $D_{n-1,r}$, that has degree less than $r$. In [GYLC94] a closed-form expression for the Wiener index of dendrimers is derived.

The work in [EMMS94] gives exact and asymptotic formulas for the Wiener number of ordered trees, rooted labeled trees and rooted binary trees. In particular it is shown that the expectation $E(W(T_n))$ is in $O(n^{5/2})$, where $T_n$ is a random tree of order $n$.

An $n$-hexagonal chain [SLG97] is defined as a graph formed by a row of $n$-hexagonal cycles. A graph consisting of $m$ $s$-hexagonal chains (where $s$ runs from $n$ to $n - m + 1$) forming a shape of a trapezium is called an $n \times m$ hexagonal trapezium. A graph obtained by merging the base of $n$-hexagonal chains of two $n \times m$ hexagonal trapeziums forming a convex six-sided polygram is called an $n \times m$ hexagonal bitrapezium. In [SLG97] the authors obtain expressions for the Wiener index of an $n \times m$ hexagonal trapezium and an $n \times m$ hexagonal bitrapezium.

The work in [DG98] presents a unified approach to Wiener index and its various modifications and generalizations. It relates all these indices to different chemical applications.
1.2. A QUICK LITERATURE SURVEY

In [GILP00] the authors solve the inverse Wiener index problem for arbitrary graphs. For $K_n$ we have $W(K_n) = \frac{n(n-1)}{2}$ while for the star $K_{1,n-1}$ we have $W(K_{1,n-1}) = (n - 1)^2$. Adding an edge to $K_{1,n-1}$ decreases the Wiener index by 1 and this can be iteratively repeated till a complete graph is obtained. This gives a continuous interval of Wiener indices from $\frac{n(n-1)}{2}$ through $(n - 1)^2$. The next interval defined by $K_{1,n}$ and $K_{n+1}$ overlaps the previous interval, thus showing that for any integer $n$ there exists a graph $G$ with $W(G) = n$.

In the detailed survey [DEG01], the Wiener index of different classes of trees are considered, the related theory is reviewed and the relevant applications are pointed out. For example, the paper outlines the computational methods and the connections between Wiener index and the center and the centroid of a tree. Results on line graphs of trees are also given. A few open problems and conjectures are also suggested.

The work in [Nei01] analyses the Wiener index for randomly generated recursive trees in the uniform probabilistic models. The paper gives expectations, asymptotics for various variances and limit laws for Wiener index.

In [IK02], efficient equations are presented for the computation of a variety of distance-based topological indices including Wiener index of a graph from the distance invariants of its subgraphs. The authors point out the use of their method in the screening of combinatorial libraries of chemical compounds. Among other things, the authors give a general technique for obtaining the Wiener index of chemical compounds (i.e., their molecular graphs) that are assembled from a core structure with $m$ ($m \geq 1$) substitutions.
In [FHRV02] graphs with restricted degrees (equivalent to valencies in the molecular graphs) are studied. The central problem considered is as follows: what trees maximize the Wiener index among all trees of a given order $n$ and whose maximum degree is at most $\Delta$ (for $\Delta = 3$)? A related problem considered is: what trees maximize the Wiener index among all trees of a given order $n$, whose vertices are either end-vertices or of maximum degree $\Delta$ (for $\Delta = 3$)?

In [Kle03] it is shown that there are different possible ways of partitioning the Wiener index of a tree $T$ into contributions associated with different substructures of $T$.

The notion of Wiener polynomials is introduced in [SYZ03] and the properties of Wiener polynomials are studied. Using the concept of Wiener polynomials, the authors independently derive the expression for the Wiener index of $r$-ary dendrimers.

Benzenoid graphs are composed of hexagonal rings with six $C$–atoms. It is assumed that such a graph has at least two hexagonal rings. By definition, any two rings either have one common edge or have no common vertex. Also, by definition no three rings share a common vertex. Thus, each ring is adjacent to two or three other rings with the exception of terminal rings which have only a single adjacent ring. The work in [Dob03] reports a decomposition – based on vertex degrees – of Wiener index of a special class of benzenoid graphs.

In [DM04] an infinite 2-parameter family of growing chemical trees $T$ with the property $W(T) = W(L(L(T)))$ has been constructed.

The Wiener conjecture for trees states that except for some finite set (of integers), every possible integer $n$ is the Wiener index of some tree. The
1.2. A QUICK LITERATURE SURVEY

authors in [BBM04] show that enumerating all possible exponential space of trees is not required to verify the conjecture. They show that searching a small special family of trees known as caterpillars suffices (a tree is a caterpillar if the deletion of all its pendants produces a path). In particular they show that every integer $n \in [103, 108]$ is the Wiener index of some caterpillar. The authors also provide efficient algorithms with implementation details for obtaining trees with a Wiener index $n$ where $n < 108$.

A subgraph $H$ of a graph $G$ is gated if for every $x \in V(G)$ there exists a vertex $u \in V(H)$ such that $d_G(x, v) = d_G(x, u) + d_G(u, v)$, for any $v \in V(H)$. The gated amalgam of two graphs $G_1$ and $G_2$ is obtained from $G_1$ and $G_2$ by identifying their isomorphic gated sub-graphs $H_1$ and $H_2$ respectively. In [Kla05], two theorems on the Wiener index of gated-amalgams are proved. It is shown that many known results related to Wiener index of molecular graphs are corollaries of these theorems - this is demonstrated by considering gated-amalgams of trees and benzenoid systems.

In [Wag06] a class of trees called star-like trees that is connected to partition of integers, is introduced and the Wiener index of this class is obtained. The main result is that every integer greater than 470 is the Wiener index of a member of the said class. As a consequence, it proves a conjecture (which says that there is some bound $M$ such that for all $w \geq M$ there exists a tree $T$ with $W(T) = w$) of Lepović and Gutman. The work in [Wag06] also contains extremal and average results on the Wiener index of the studied class of trees.
1.3 Techniques for computing Wiener index

We begin with the following two results (see [KRG96] and [DEG01]) whose correctness follow from combinatorial arguments.

**Lemma 1.3.1:**

Given the graphs $G = (V(G), E(G))$ and $H = (V(H), E(H))$, their Cartesian product $G \times H$ is a graph defined as:

$$V(G \times H) = V(G) \times V(H) \quad \text{and} \quad E(G \times H) = \{(a, x)(b, y) | a = b \text{ and } xy \in E(H) \text{ or } ab \in E(G) \text{ and } x = y\}.$$  

The Cartesian product of $G$ and $H$ is associative and commutative. The Wiener index of $G \times H$ is given by the formula:

$$W(G \times H) = |V(H)|^2 W(G) + |V(G)|^2 W(H). \quad (1.4)$$

**Lemma 1.3.2:**

Let $T$ be a tree obtained from arbitrary trees $T_a$ and $T_b$ of orders $n_1$ and $n_2$ respectively and let $u \in V(T_a)$ and $v \in V(T_b)$. Then

(a) If $u$ and $v$ are fused together i.e., identified to be single vertex, say $u$ so that the vertex $u$ becomes a cut-vertex in $T$ – see Fig.1.1(a), then

$$W(T) = W(T_a) + W(T_b) + (n_1 - 1)d^+(u, T_b) + (n_2 - 1)d^+(u, T_a). \quad (1.5)$$

(b) If $u$ and $v$ are linked by an edge so that the edge $uv$ becomes a cut-edge
1.3. TECHNOQUES FOR COMPUTING WIENER INDEX

in T – see Fig. 1.1(b), then

\[ W(T) = W(T_a) + W(T_b) + n_1 d^+(v, T_b) + n_2 d^+(u, T_a) + n_1 n_2. \]

(1.6)

Figure 1.1: A tree \( T \): assembled from trees \( T_a \) and \( T_b \).

Remark 1.3.1: It is convenient to refer to lemma 1.3.2 as the cut-vertex/cut-edge lemma. We note that lemma 1.3.2 holds good even when \( T_a \) and \( T_b \) are connected graphs which are not necessarily trees.

1.3.1 G characterized by parameters

When a family of graphs is compactly characterized via one or more parameters, it is often possible to obtain a closed-formula for \( W(G) \) in terms of the said parameters. We first mention the example of the \( n \)-dimensional Star graphs \( S_n \) which are known to be potential candidates as networks for interconnection of processors having interesting combinatorial properties. For a fixed \( n \), the graph \( S_n \) is defined in the following way:

- The vertex set \( V(S_n) \) consists of \( n! \) vertices where each vertex is labeled by \( \pi_1, \cdots, \pi_n \), a permutation of the integers \( 1, \cdots, n \).
CHAPTER 1. WIENER INDEX OF A GRAPH

• Given a label $\pi_1, \cdots, \pi_i, \cdots, \pi_n$, let the mapping $g_i$ ($1 < i \leq n$) be defined as

$$g_i(\pi_1, \cdots, \pi_i, \cdots, \pi_n) = (\pi_i, \cdots, \pi_1, \cdots, \pi_n).$$

Stated differently, the function $g_i$ swaps the integers $\pi_1$ and $\pi_i$, leaving the other integers in place. The edge set $E(S_n)$ is defined as

$$E(S_n) = \{uv | u, v \in V(S_n) \text{ and } v = g_i(u) \text{ for some } i\}.$$

It is easy to see that $S_3$ is $C_6$. From [FA95] we infer that $\mu(S_n)$ is given by

$$\mu(S_n) = \frac{n!}{n! - 1} \left\{ n + \frac{2}{n} + H_n - 4 \right\},$$

where $H_n$ is the $n^{th}$ Harmonic number.

Aside: An interesting problem is to obtain an expression for the number $N_k$ of nodes, in $S_n$, which are at a distance $k = 1, \ldots, d = \lfloor \frac{3(n-1)}{2} \rfloor$, the diameter of $S_n$. Using a computer program, in table 1.1 we have given the values of $N_k$ for $S_4$ through $S_9$. This problem has been solved in the literature.

We now illustrate some of the more common paper-and-pencil techniques which may be useful in different cases.

(a) Direct enumeration

By direct enumeration, it follows that

$$(i) \ W(K_n) = \binom{n}{2}, \ (n \geq 2) \ \text{and} \ \ (ii) \ W(K_{m,n}) = (m+n)^2 - (m + n) - mn, \ (m, n \geq 1).$$
<table>
<thead>
<tr>
<th></th>
<th>$n=4$</th>
<th>$d=4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$n=5$</th>
<th>$d=6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>44</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$n=6$</th>
<th>$d=7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>70</td>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>170</td>
<td>170</td>
<td>170</td>
</tr>
<tr>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>169</td>
<td>169</td>
<td>169</td>
</tr>
<tr>
<td>35</td>
<td>35</td>
<td>35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$n=7$</th>
<th>$d=9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>135</td>
<td>135</td>
<td>135</td>
</tr>
<tr>
<td>460</td>
<td>460</td>
<td>460</td>
</tr>
<tr>
<td>1110</td>
<td>1110</td>
<td>1110</td>
</tr>
<tr>
<td>1689</td>
<td>1689</td>
<td>1689</td>
</tr>
<tr>
<td>1254</td>
<td>1254</td>
<td>1254</td>
</tr>
<tr>
<td>340</td>
<td>340</td>
<td>340</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$n=8$</th>
<th>$d=10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>42</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>231</td>
<td>231</td>
<td>231</td>
</tr>
<tr>
<td>1015</td>
<td>1015</td>
<td>1015</td>
</tr>
<tr>
<td>3430</td>
<td>3430</td>
<td>3430</td>
</tr>
<tr>
<td>8379</td>
<td>8379</td>
<td>8379</td>
</tr>
<tr>
<td>13083</td>
<td>13083</td>
<td>13083</td>
</tr>
<tr>
<td>10408</td>
<td>10408</td>
<td>10408</td>
</tr>
<tr>
<td>3409</td>
<td>3409</td>
<td>3409</td>
</tr>
<tr>
<td>315</td>
<td>315</td>
<td>315</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$n=9$</th>
<th>$d=12$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>56</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>364</td>
<td>364</td>
<td>364</td>
</tr>
<tr>
<td>1960</td>
<td>1960</td>
<td>1960</td>
</tr>
<tr>
<td>8540</td>
<td>8540</td>
<td>8540</td>
</tr>
<tr>
<td>28994</td>
<td>28994</td>
<td>28994</td>
</tr>
<tr>
<td>71512</td>
<td>71512</td>
<td>71512</td>
</tr>
<tr>
<td>114064</td>
<td>114064</td>
<td>114064</td>
</tr>
<tr>
<td>96116</td>
<td>96116</td>
<td>96116</td>
</tr>
<tr>
<td>36260</td>
<td>36260</td>
<td>36260</td>
</tr>
<tr>
<td>4900</td>
<td>4900</td>
<td>4900</td>
</tr>
<tr>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
</tbody>
</table>

Table 1.1: Number $N_k$ of vertices at a distance $k = 2, \ldots, d$ from a fixed vertex in $S_n$, $n = 4, \ldots, 9$. 
CHAPTER 1. WIENER INDEX OF A GRAPH

(b) **Use of recurrence relations**

In some cases it is possible to develop a recurrence relation involving $W(G)$. Following are three examples.

(i) Let $P_n$ denote a path on $n$ ($n \geq 2$) vertices. Then it follows that

$$W(P_n) = W(P_{n-1}) + \frac{1}{2} n(n - 1), \quad (n > 2) \text{, with } W(P_2) = 1.$$  

Thus, $W(P_n) = \binom{n+1}{3}$.

(ii) Let $G$ be the molecular graph of a fully methyl-substituted normal alkane i.e., $G$ is the tree $T_n$ obtained by taking the path $P_{n+2}$ and attaching two vertices to every non-pendant vertex of $P_{n+2}$. It follows that $W(T_n)$ satisfies the following recurrence:

$$W(T_n) = W(T_{n-1}) + \frac{9}{2} n^2 + \frac{27}{2} n - 3, \quad n > 1 \text{ with } W(T_1) = 16.$$  

It follows that $W(T_n) = 1 + \frac{3}{2}(n^2 + 6n + 3)$.

(iii) Let $G$ be the ladder graph $L_n$, ($n \geq 1$) i.e., $L_n$ resembles a ladder with $n + 1$ rungs. Note that $L_n \cong P_n \times P_2$, $n \geq 1$. It then follows that

$$W(L_n) = W(L_{n-1}) + 2(n + 1)^2 - 1, \quad (n > 1) \text{ and } W(L_1) = 8.$$  

Thus we get $W(L_n) = \frac{1}{3}(n + 1)(n + 3)(2n + 1)$.

(c) **Use of graph product**

The following are illustrative examples.

(i) let $P$ denote the graph of a prism obtained by the Cartesian product of $C_3$ and $K_2$ i.e., $P \cong C_3 \times K_2$. It then follows from (1.4) that $W(P) = 21$.  

1.3. TECHNIQUES FOR COMPUTING WIENER INDEX

(ii) Let $G$ be $Q(r, n)$ as defined in [VIJV96]: for a given integer $r > 1$

\[ Q(r, 1) = K_r; \ Q(r, n + 1) = Q(r, n) \times K_r, \ n \geq 1. \]

In other words, $Q(r, n) = K^n_r$. By repeated applications of (1.4) we get

\[ W(G^n) = n|V(G)|^{2(n-1)}W(G). \]

Taking $G = K_r$, it follows that

\[ W(Q(r, n)) = \frac{1}{2}n(r - 1)r^{2n-1}. \]

For the Boolean hypercube $Q_n$ (see [Hwa93]), we have $r = 2$, whence it then follows that $W(Q_n) = n2^{2n-2}$.

(iii) For $m, n \geq 2$, let $G$ be the $m \times n$ grid i.e., $G \cong P_m \times P_n$. Then by (1.4), we get

\[ W(P_m \times P_n) = \frac{1}{6}(m + n)mn(mn - 1). \]

From the expression for $W(P_m \times P_n)$ one notes that the special case for the ladder graph follows by taking $m$ to be $n + 1$ and $n$ to be 2.

(d) Use of cut-vertex/cut-edge lemma

(i) We consider the trees depicted in Fig.1.2. By applying (1.6) it follows that

\[ W(G_1) = \binom{9}{3} + \binom{7}{3} + 8.15 + 6.22 + 8.6 = 419 \quad \text{and} \]
\[ W(G_2) = \binom{10}{3} + \binom{6}{3} + 9.6 + 5.36 + 9.5 = 419. \]

(ii) In chapter 2, we compute the Wiener index of two chemical trees using the cut-vertex/cut-edge lemma.
CHAPTER 1. WIENER INDEX OF A GRAPH

Figure 1.2: Two non-isomorphic trees of order 14.

(e) Use of combinatorial techniques

(i) In [GYLC94], a closed-form expression for the Wiener index of the chemical (molecular) graph of dendrimers is obtained.

(ii) In chapter 3, we derive a closed expression for the Wiener number of the odd graphs.

1.3.2 Algorithmic determination of $W(G)$

The problem of computing $W(G)$ is interesting algorithmically – researchers have raised the question as to whether the complexity of computing $W(G)$ is less than the complexity of computing the distances between all possible vertex-pairs of $G$. From the viewpoint of Mathematical Chemistry it is meaningful to consider what may be called as chemical graphs i.e., those graphs where the degree of any vertex is not more than four. Surely the algorithmic determination of $W(G)$ is interesting when $G$ is restricted to a class of graphs characterized by a property which may be specified in terms of one or more graph parameters. A brief outline of some of the algorithmic
studies relating to $W(G)$ follows here.

(a) $G$ is an arbitrary graph

When $G = (V, E)$ is specified as an arbitrary graph, a naive approach will be to apply an all-pairs shortest path algorithm (see [CLRS01]). In the worst case, this takes a time $O(|V|^3)$. As pointed out in [BBM04], by using Johnson’s algorithm augmented with Fibonacci heaps, the time complexity can be reduced to $O(|V|^2 \log |V| + |E||V|)$. However, we point out that a modified breadth-first-search ($BFS$), keeping track of the distances to each vertex from a source, can be used once from each vertex as the source of $G$ (we call this algorithm $nBFS$). This gives a time-complexity of $O(|V|^2 + |E||V|)$ which can be as bad as $O(|V|^3)$ for dense graphs.

Let $D = (d_{ij})$, where $d_{ij}$ is the distance between the vertices labeled by $i$ and $j$ (assuming that the vertices of $G$ are labeled 1 through $|V|$).

In [Sei95] an algorithm is given to compute the entries of $D$ in time $O(M(|V| \log |V|))$, where $M(|V|)$ is the time required to multiply two $|V| \times |V|$ matrices. Therefore, in theory $W(G)$ should be computable in time $O(M(|V| \log |V|))$ since $D$ has $|V|^2$ entries.

(b) $G$ is an interval graph

An interval graph is a graph $G = (V, E)$ whose vertex set can be mapped onto a set of closed intervals, without loss of generality, on the real line such that for every pair of vertices $u, v \in V$, we have $uv \in E$ if and only if the intervals corresponding to $u$ and $v$ intersect.

In [Dan93] an algorithm is given to compute $\mu(G)$ for any interval graph of size $o(|V|^2)$ in time $O(|E|)$. This trivially implies that for such graphs $W(G)$ can be computed in time $O(|E|)$.
(c) \textbf{G is a tree}

In [Dan93] the problem of finding \( \mu(G) \) for the case when \( G \) is a tree is also considered. As a modification of the algorithm (for finding \( \mu(G) \)) for interval graphs, it is argued that \( \mu(G) \) of a tree (and hence \( W(G) \)) of order \( |V| \) can be found in time \( O(|V|) \), which is optimal. In chapter 6, we suggest a different and more general algorithm to compute \( W(G) \) in time \( O(|V|) \).

(d) \textbf{G is a benzenoid graph}

A benzenoid graph or a benzenoid system is a graph \( G \) constructed in the following manner: Let \( H \) be the hexagonal lattice (similar to the structure of a single layer of graphite lattice) and let \( Z \) be a circuit on it. Then a benzenoid graph is induced as a subgraph by the vertices and edges of \( H \) lying on \( Z \) and in the interior of \( Z \). By embedding a benzenoid graph into the Cartesian product of three trees, the algorithm described in [CK97] computes \( W(G) \) in time \( O(|V|) \), where \( |V| \) is the number of vertices in the input.

\section{1.4 A decomposition algorithm}

Let \( G = (V(G), E(G)) \) with \( |V(G)| = n \) and \( |E(G)| = m \). We first extend the cut-vertex/cut-edge lemma using the notion of a clique-separator. A clique-separator (or a clique-cutset) in \( G \) is a clique \( G[S] \), where \( S \subset V(G) \) such that \( G[V(G) \setminus S] \) has more connected components than \( G \). If \( G[S] \) is a clique-separator in \( G \), then we can decompose \( G \) into two subgraphs \( G_1 = G[V_1] \) and \( G_2 = G[V_2] \) where \( V(G) = V_1 \cup V_2 \) and \( S = V_1 \cap V_2 \). Let \( |G_1| = n_1 \), \( |G_2| = n_2 \) and \( |S| = k \). Let \( DIST \) denote
1.4. A DECOMPOSITION ALGORITHM

the sum of all distances \(d(u, v)\) where \(u \in G_1 \setminus S\) and \(v \in G_2 \setminus S\). Then it follows that

\[
W(G) = W(G_1) + W(G_2) - \binom{k}{2} + \text{DIST.} \tag{1.7}
\]

In practice, the extension involving a clique-separator is useful in determining the Wiener index of many molecular graphs whose underlying compounds consist of ring systems (i.e., containing more than one benzene ring structurally). In these cases a clique-separator with two vertices can be found. Examples of such ring systems include anthracene, phenanthracene, hexahelicene, hexaphene and the entire family of steroids and many compounds of the polyacene family. In considering molecular graphs (not necessarily acyclic), we can assume that \(m = O(n)\). We also assume that \(k\) is bounded by a constant. We can then compute \(\text{DIST}\) in (1.7) in time \(O(n_1^2)\), where we assume \(n_1 \geq n_2\). Computation of \(\text{DIST}\) can be done by the following function GetDIST which takes as input \(G_1, G_2\) and \(S\):

```plaintext
function GetDIST;
begin
  for (every \(v_i \in S\)) do
  begin
    for (every \(u \in G_1 \setminus S\)) do compute \(d(u, v_i)\);
    for (every \(v \in G_2 \setminus S\)) do compute \(d(v_i, v)\);
  end;
  \(\text{DIST} = 0\);
  for (every \(u \in G_1 \setminus S\)) do
  for (every \(v \in G_2 \setminus S\)) do
  begin
```
CHAPTER 1. WIENER INDEX OF A GRAPH

\[ d_{\text{min}}(u, v) = \infty; \]

for (every \( v_i \in S \)) do

begin

\[ d_1 = d(u, v_i) + d(v_i, v); \]

Compute \( d_j \)s of the form \( d(u, v_j) + 1 + d(v_j, v) \), \( j \neq i; \)

\[ d_2 = \min_j \{d_j\}; \]

\[ d'_{\text{min}}(u, v) = \min\{d_1, d_2\}; \]

if \( d'_{\text{min}}(u, v) < d_{\text{min}}(u, v) \) then \( d_{\text{min}}(u, v) = d'_{\text{min}}(u, v); \)

end:

\[ \text{DIST} = \text{DIST} + d_{\text{min}}(u, v); \]

end:

return \( \text{DIST}; \)

end

Given \( G \), let \text{RECUR-DECOMP} given below be the natural algorithm that computes \( W(G) \) recursively, by successively decomposing \( G \) by finding a cut-edge or a cut-vertex or a clique-separator (and applying (1.5), (1.6) or (1.7) as the case may be).

Algorithm \text{RECUR-DECOMP}:

1: If \( |V(G)| = 3 \) then return \( W(G) = 3 \) if \( G \) is \( C_3 \) or return \( W(G) = 4 \) if \( G \) is \( P_3 \).

2: If a cut-edge or a cut-vertex or a clique-separator cannot be found then apply \text{nBFS} and return \( W(G) \).

3: If a cut-edge or a cut-vertex or a clique-separator is found then decompose \( G \) into \( G_1 \) and \( G_2 \) and apply (1.5), (1.6) or (1.7) and return \( W(G) \) – in computing \( W(G_1) \) or \( W(G_2) \), call \text{RECUR-DECOMP} recursively.
In analyzing RECUR-DECOMP, the following remarks are in order. Note that we have assumed $m = O(n)$ as in trees and in uni-cyclic graphs. The quantities such as $d^+(u, G)$ can be computed by running BFS once from the vertex $u$. This takes time $O(n)$. Finding a cut-edge or a cut-vertex takes time $O(m)$ in general (see [CLRS01]). This takes time $O(n)$. Decomposition of a graph by clique-separators (see [Tar85]) can be done in time $O(mn)$ i.e., $O(n^2)$. As mentioned earlier, when $m = O(n)$, algorithm nBFS takes time $O(n^2)$.

Let $T(n)$ be the time taken by algorithm RECUR-DECOMP on the input graph $G$ of order $n$. Then it follows that

$$T(n) \leq T(n_1) + T(n_2) + cf(n). \tag{1.8}$$

Here, $c$ is a constant and $n_1$ and $n_2$ are the orders of the subgraphs $G_1$ and $G_2$. The quantity $f(n)$ captures: (i) the sum of the worst-case efforts in finding a cut-edge and a cut-vertex and a clique-separator and (ii) the time to compute $d^+(u, G_2)$ and $d^+(v, G_1)$ or the time taken by the function GetDIST. The following extreme cases of (1.8) can be identified:

**Case 1:** During each recursive call to RECUR-DECOMP, the algorithm finds a cut-edge or a cut-vertex or a clique-separator in time $O(n^2)$ and we get a situation where $n_1 = n_2 = n/2$. Then we have

$$T(n) \leq 2T\left(\frac{n}{2}\right) + K_1 n^2, \text{ where } K_1 \text{ is a constant.}$$

In this case, $T(n)$ is in $O(n^2)$ and thus algorithm RECUR-DECOMP is asymptotically similar to algorithm nBFS.

**Case 2:** In this case, we assume that the decomposition of $G$ is such that
we have \( n_1 = (n - 1) \) and \( n_2 = 1 \) (or vice versa). Then

\[
T(n) \leq T(n - 1) + T(1) + K_2n^2,
\]

where \( K_2 \) is a constant.

Here, we get \( T(n) = O(n^3) \) and algorithm \textsc{RECUR-DECOMP} is asymptotically inferior to algorithm \textsc{nBFS}.

We thus get a negative result: In the best case \textsc{RECUR-DECOMP} takes time \( O(n^2) \) whence it is asymptotically similar to \textsc{nBFS}, while in the worst case it takes time \( O(n^3) \) and is thus asymptotically inferior to \textsc{nBFS}. In other words, given \( G \), decomposing it and applying (1.5) or (1.6) or (1.7) is a bad strategy. However, if \( G \) is constructed from known subgraphs \( G_1 \) and \( G_2 \) (by giving rise to a cut-vertex or a cut-edge or a clique-separator in \( G \)), then (1.5) or (1.6) or (1.7) should be useful. It turns out that such constructions of \( G \) are apparently useful in the virtual screening of large combinatorial libraries (for example, see [IK02]).

### 1.5 Chapter summaries

We conclude this chapter by briefly outlining the summary of each of the remaining chapters. Except for the concluding chapter, these chapters are based on the work reported in [BSVI06], [BSVI07], [BSVI10], [BSVI08] and [VIP09].

In chapter 2, we consider two special cases of trees viz., the complete binary tree and a chemical tree and obtain the closed-form expressions for the Wiener index of those trees. Odd graphs are known to be interesting combinatorial objects. In chapter 3 we obtain an expression for the Wiener
1.5. CHAPTER SUMMARIES

index of odd graphs. We also provide a table of actual values of the distance distribution in some odd graphs. Chapter 4 provides details of the derivation of a sharp lower bound for $W(G)$ for an arbitrary $G$. The bound is based on the order, size and diameter of $G$. Some families of graphs attaining the bound are given. In chapter 5, we map a distributed algorithm for the all-pairs shortest path problem onto a bus-based architecture and show how to compute the Wiener index of any input graph. Given a tree $T$, in chapter 6 we give an algorithm to find a subtree of $T$ satisfying certain constraints. As a special case, we can obtain a linear-time algorithm for computing $W(T)$. It is also shown that the algorithm can be used to efficiently compute the Wiener index of all the trees that can be constructed in the cut-vertex/cut-edge lemma. Chapter 7 considers the case of graphs $G$ with more than one cut-vertex. We obtain an expression for $W(G)$ for such graphs thereby generalizing (1.5) in the cut-vertex/cut-edge lemma. In chapter 8, we outline some possible future research directions based on our study of the contemporary literature.
Chapter 2

$W(G)$ of two chemical trees

2.1 Introduction

As pointed out in chapter 1, a variety of techniques may be used to find the closed-form expressions for $W(G)$ when $G$ is characterized by one or more parameters. Many such motivating examples are summarized in [DEG01] for the case of trees. In this chapter, we derive expressions for $W(G)$ when $G$ is a complete binary tree and when $G$ is a restricted class of chemical trees (molecular graph) derived by maximum substitutions of $n$-alkyl groups on a normal alkane of a fixed diameter. The present work is of similar type to the work reported in [GYLC94] wherein an expression for the Wiener number of dendrimers was first reported.
2.2 Wiener index of complete binary trees

We begin by finding an explicit expression for the Wiener index of a non-trivial complete binary tree. A binary tree $T$ is defined on a finite set of vertices that either (a) contains no vertices or (b) is composed of three disjoint sets of vertices: a root vertex, a binary tree called the left subtree of $T$ and a binary tree called the right subtree of $T$ (see [CLRS01] for details).

We define the root $r$ to be at depth 0. The vertices connected directly to the root are at depth 1. In general, a vertex is at depth $k + 1$ if it is a child of a vertex at depth $k$. A complete binary tree of height $k$, denoted by $T_k$, is one that has vertices upto depth $k$ and has the maximum possible number of vertices at each depth. The total number $n_{T_k}$ of vertices in $T_k$ is given by

$$n_{T_k} = 2^{k+1} - 1. \quad (2.1)$$

Recalling the definition of $d^+(u, G)$ we get

$$d^+(r, T_k) = \sum_{i=0}^{k} i \cdot 2^i = 2 + (k - 1)2^{k+1}. \quad (2.2)$$

In proving the following result and the later result in this chapter, we invoke lemma 1.3.2 stated in chapter 1.

**Theorem 2.2.1:**

The Wiener index of a complete binary tree of height $k$ is given by

$$W(T_k) = (k + 4)2^{k+1} + (k - 2)2^{2(k+1)}.\quad$$

**Proof:** Let $T_k$ be the complete binary tree of height $k$ whose root is $r$. Let $u, v$ be the children of $r$ – that is, $u$ and $v$ are the roots of $T_{k-1}$. Then
$T_k$ can be formed using two subtrees $T_{k-1}$ as shown in Fig. 2.1. We first identify $T_k$ as a tree of the type as shown in Fig. 1.1(a). We take $T_a$ to be $T_{k-1} \cup \text{(edge}(ur))$ and $T_b$ to be $T_{k-1} \cup \text{(edge}(vr))$ so that $r$ is a cut-vertex of $T_k$ with $T_a \cap T_b = \{r\}$. Then by (1.5) and (2.1) and noting that $T_b \cong T_a$, we have

$$W(T_k) = 2W(T_a) + 2(2^k + 1) + \text{d}^+(u, T_a).$$  \hfill (2.3)

By invoking (1.6), it follows that

$$W(T_a) = W(T_{k-1}) + (k - 1)2^k + 1.$$  \hfill (2.4)

Also by using (2.2) it follows that

$$d^+(u, T_a) = (k - 1)2^k + 1.$$  \hfill (2.5)

Substituting (2.4) and (2.5) in (2.3) we get

$$W(T_k) = 2W(T_{k-1}) + 2^{2k+1} (k - 1) + 2^{k+1}.$$  

Solution to the above recurrence relation gives $W(T_k)$ as desired.  \hfill □

**Note 2.2.1:** In an analogous way an exercise is to show that for $r > 1$ a binomial tree $T_r$ having $2^r$ vertices has the Wiener index given by the expression $(r - 1)2^{2(r-1)} + 2^{(r-1)}$. For a definition of binomial trees and
some related properties the reader can see [CLRS01] for example.

**Remark 2.2.1:** Define the unique graph $G_{T_k}$ corresponding to $T_k$ thus: for $k \geq 1$, in building $T_k$ from $T_{k-1}$ we modify the rule of adding two child vertices to every leaf of $T_{k-1}$ – we additionally include an edge between the left child and right child. A top-down recursive argument reveals that $T_k$ is the minimum-Wiener-indexed spanning tree of $G_{T_k}$.

### 2.3 Wiener index of $A_{2k+1}$ and $A_{2k}$

We define a class of trees $A_{2k+1}$ with a parameter $k$ (where $k$ is a positive integer) that corresponds to the molecular graphs of alkanes with diameter $2k$ where every $H$-atom is substituted by normal alkyl groups having the longest possible path. In Fig.2.2(a), (b) and (c) respectively, the graphs $A_3$, $A_5$ and $A_7$ are depicted. We can define the tree $A_{2k+1}$ with diameter $2k$ as follows:

(a) $A_3$ is the tree given in Fig.2.2(a)

(b) $A_{2k+1}$ is obtained from $A_{2k-1}$ in the following manner.

Add three pendant vertices to the left-most and the right-most pendant vertices of $A_{2k-1}$; next, add a pendant vertex to every other pendant vertex of $A_{2k-1}$.

It can be seen that $A_{2k+1}$ has (a) $2k(k+1) + 1$ vertices (b) $4k$ pendant vertices, and (c) $2k-1$ vertices of degree 4. In a manner analogous to the above, we define the class of trees $A_{2k}(k \geq 1)$ with diameter $2k - 1$ starting with $A_2$ shown in Fig. 2.2(d). We note that $A_{2k}$ has (a) $2k^2$ vertices
(b) $2(2k - 1)$ pendant vertices and (c) $2k - 2$ vertices of degree 4. In this section, we obtain the Wiener index of the trees $A_{2k+1}$ and $A_{2k}$. We begin with following lemmas.

**Lemma 2.3.1:**

Let $P_n$ denote a path on $n \geq 2$ vertices. Then $W(P_n) = \binom{n+1}{3}$.

**Lemma 2.3.2:**

Let $B_n$ be the $n^{th}$ Bernoulli number (see [Knu97]) and let $r$ and $k$ be positive integers. Then the sum of the $r^{th}$ powers of the first $k$ natural numbers is given by

$$
\sum_{m=1}^{k} m^r = \sum_{j=1}^{r+1} \binom{r+1}{j} \frac{B_{r+1-j}}{(r+1)(k+1)^j}.
$$

**Proof:** For a proof see [Wil94].
2.3. WIENER INDEX OF $A_{2K+1}$ AND $A_{2K}$

Remark 2.3.1: As an example the sum $S_4(k)$ of the fourth powers of the first $k = l - 1$ natural numbers can be given as $S_4(k) = B_4l + 2B_3l^2 + 2B_2l^3 + B_1l^4 + \frac{B_0}{5}l^5$. Noting that $B_0 = \frac{1}{5}$, $B_1 = -\frac{1}{2}$, $B_2 = \frac{1}{6}$, $B_3 = 0$, $B_4 = -\frac{1}{30}$, it follows that $S_4(k) = \frac{k(k+1)(3k^2+3k-1)}{30}$.

Lemma 2.3.3:

Let $v_k$ denote the center vertex of $P_{2k+1}$ and let $w$ be a pendant of $P_{k+1}$. Then $d^+(v_k, P_{2k+1}) = k(k+1)$ and $d^+(w, P_{k+1}) = \frac{1}{2}k(k+1)$.

Proof: Follows from basic counting principles.

Lemma 2.3.4:

Let $G_3$ be the tree shown in Fig. 2.3(a). We construct the tree $G_{2k+1}$ (with parameter $k$) in the following manner. In $G_{2k-1}$, let $u_{k-1}$ denote the vertex of degree 3 on the longest path. Add an edge $u_{k-1}v_k$, connecting $G_{2k-1}$ and a path $P_{2k+1}$, where $v_k$ is the middle vertex of $P_{2k+1}$. In Fig. 2.3(b) and (c) we depict $G_5$ and $G_7$ respectively (note that the vertex $v_k$ gets identified with vertex $u_k$ of $G_{2k+1}$). Then $D_k = d^+(u_k, G_{2k+1}) = \frac{1}{2}k(k+1)(4k+5)$.

Proof: From basic counting principles we get $D_1 = 3$; and for $i = 1, \ldots, k - 1$ we have $D_i = D_{i-1} + i^2 + 2 \sum_{j=2}^{i} j$. By direct summation of these equations we get $d^+(u_k, G_{2k+1})$ as desired.

Lemma 2.3.5:

Let $G_{2k+1}$ be as defined in lemma 2.3.4. Then the Wiener index $W(G_{2k+1})$
CHAPTER 2. $W(G)$ OF TWO CHEMICAL TREES

is given by the following expression:

$$W(G_{2k+1}) = \frac{1}{6} (W_1 + W_2 + W_3 + W_4), \text{ where}$$

$$W_1 = \frac{7}{15} k(k+1)(2k+1)(3k^2 + 3k - 1)$$

$$W_2 = 6 k^2(k+1)^2$$

$$W_3 = \frac{13}{6} k(k+1)(2k+1)$$

$$W_4 = \frac{3}{2} k(k+1).$$

Proof: Let the edge $u_{k-1}v_k$ in definition of $W(G_{2k+1})$ (see lemma 2.3.4) be identified as the cut-edge of lemma 1.3.2 (case (b)). By (1.6) we can write $W(G_{2k+1})$ as

$$W(G_{2k+1}) = W(G_{2k-1}) + W(P_{2k+1}) + (2k+1) d^+(u_{k-1}, G_{2k-1})$$

$$+ k^2 d^+(v_k, P_{2k+1}) + k^2(2k+1).$$

Invoking lemmas 2.3.1, 2.3.3 and 2.3.4 and simplifying, the above equation reduces to

$$W(G_{2k+1}) = W(G_{2k-1}) + \frac{1}{6} [14k^4 + 24k^3 + 13k^2 + 3k]. \quad (2.7)$$

![Diagram](attachment:image.png)

Figure 2.3: The trees $G_3$, $G_5$ and $G_7$. 
2.3. WIENER INDEX OF $A_{2K+1}$ AND $A_{2K}$

In (2.7) we first successively replace $k$ by $k - 1, k - 2, \ldots, 2, 1$ and sum up the resulting equations. Observing that $W(G_1) = 0$ and using the result of lemma 2.3.2, subsequent simplification yields $W(G_{2k+1})$ as desired.

**Theorem 2.3.1:**

The Wiener index $W(A_{2k+1})$ of the tree $A_{2k+1}$ is given by:

$$W(A_{2k+1}) = c_1 k^5 + c_2 k^4 + c_3 k^3 + c_4 k^2 + c_5 k,$$

where

$$c_1 = \frac{34}{15}, \ c_2 = \frac{16}{3}, \ c_3 = \frac{16}{3}, \ c_4 = \frac{8}{3} \text{ and } c_5 = \frac{2}{5}.$$

**Proof:** We first identify the graph $A_{2k+1}$ as one consisting of two connected subgraphs $T_a = G_{2k+1}$ and $T_b = G_{2k-1}$ with cut-edge $uv$ where $u$ and $v$ are the vertices of degree 3 on the longest path in $G_{2k+1}$ and $G_{2k-1}$ respectively. That is, we identify $A_{2k+1}$ to be a graph of the type as shown in Fig.1.1(b). Applying lemma 1.3.2 (case (b)) and noting that $n_1 = (k + 1)^2$ and $n_2 = k^2$ we get

$$W(A_{2k+1}) = W(G_{2k+1}) + W(G_{2k-1}) + (k + 1)^2 d^+(v, G_{2k-1})$$

$$+ k^2 d^+(u, G_{2k+1}) + k^2 (k + 1)^2.$$

Using lemma 2.3.4 we get

$$W(A_{2k+1}) = W(G_{2k+1}) + W(G_{2k-1}) + (k + 1)^2 \frac{(k - 1) k (4k + 1)}{6}$$

$$+ k^2 k (k + 1) (4k + 5) 6 + k^2 (k + 1)^2.$$

We invoke lemma 2.3.5 for getting the expressions for $W(G_{2k+1})$ and $W(G_{2k-1})$ and simplify the resultant expression to get $W(A_{2k+1})$ as stated in the theorem. ■
Similar to the above proof, by taking $A_{2k}$ to be a type of the graph as in Fig.1.1(b) (where we take $T_a$ and $T_b$ to be isomorphic to $G_{2k-1}$) we derive the expression for $W(A_{2k})$ as:

$$W(A_{2k}) = c'_1 k^5 + c'_2 k^4 + c'_3 k^3 + c'_4 k^2 + c'_5 k,$$

where $c'_1 = \frac{34}{15}$, $c'_2 = \frac{-1}{3}$, $c'_3 = \frac{-4}{3}$, $c'_4 = \frac{1}{3}$, and $c'_5 = \frac{1}{15}$.

The Wiener index values of $A_2$ through $A_9$ are given in table 2.1.

<table>
<thead>
<tr>
<th>$m$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W(A_m)$</td>
<td>1</td>
<td>16</td>
<td>58</td>
<td>212</td>
<td>491</td>
<td>1152</td>
<td>2156</td>
<td>4072</td>
</tr>
</tbody>
</table>

Table 2.1: $W(A_m)$ for $m = 2, \ldots, 9$. 

Chapter 3

Wiener index of odd graphs

3.1 Odd graphs

In this chapter we obtain an expression for the Wiener index of odd graphs basing our argument on the principle of mathematical induction. We begin with the definition of an odd graph. For a positive integer \( k \geq 2 \), let \( X \) be any set of \( 2k - 1 \) elements. Then the odd graph \( O_k \) is defined by the following sets:

- The vertex set of \( O_k \) is the set of all \((k-1)\)-subsets of \( X \). Thus \( |V| = \binom{2k-1}{k-1} \). We refer to these vertices by referring to the corresponding \((k-1)\)-subsets denoted by \( A, B, \ldots \).

- To define the edges in \( O_k \) we consider any two vertices \( A \) and \( B \). An edge exists between them if \( A \cap B = \emptyset \) i.e., if \(|X \setminus (A \cup B)|\) is a singleton characterizing the edge \( AB \).

It is known that odd graphs have interesting combinatorial properties (see for example, [Big74]). It is also well-known that \( O_3 \) is the Petersen graph.
The characterization of the edges of $O_k$ aids in exploring the combinatorial properties of $O_k$.

*Aside.* Apart from possessing interesting combinatorial properties, it turns out that odd graphs are potential candidates as interconnection systems in the design of high-performance parallel computing systems. Given the current state of technology, especially the recent advances in VLSI technology, it is now possible to embed a large number of processors on a single chip connected via a communication network. Alternatively, a large number of processors could be connected to a communication network which could be typically a *backplane*. The resultant designs exploit parallelism in most of the algorithms running on them and provide enhanced fault-tolerance by using many of the available processors as redundant spares. Physically, the processors in such systems can be connected in many possible ways. The selection of a suitable interconnection network largely determines the overall performance of the system (see for example, [Hwa93] for more details on interconnection networks). The study in [GB91] analyzes odd graphs for building a communication architecture for large multiprocessor systems and shows that odd graphs possess many attractive features related to fault tolerance, resilience. It is argued that odd graphs, as interconnection networks, admit simple distributed routing algorithms both for faulty and fault-free networks.

### 3.2 Wiener index of $O_k$

It is clear that if $A$ and $B$ are two distinct vertices of $O_k$, then $|A \cap B| = i$ for some $i \in \{0, 1, \ldots, k - 2\}$. 
3.2. WIENER INDEX OF $O_K$

**Lemma 3.2.1:**

Let $k \geq 2$. Let $A_0 \in V(O_k)$ be a fixed vertex. Then for any $A \in V(O_k)$, the distance $d(A_0, A) = i$ in $O_k$ if and only if

$$|A_0 \cap A| = \begin{cases} \frac{i-1}{2} & \text{if } i \text{ is odd,} \\ k - \left(\frac{i+2}{2}\right) & \text{if } i \text{ is even.} \end{cases}$$

Hence $\text{diam}(O_k) = k - 1$.

**Proof:** If $k = 2$, $O_k = C_3$, and the result is true. So assume that $k \geq 3$. We prove the result by induction on $d(A_0, A)$. Now, $d(A_0, A) = 0$ iff $A = A_0$, and $d(A_0, A) = 1$ iff $A \cap A_0 = \emptyset$. Hence the result is true for $i = 0$ and $i = 1$.

Now consider the case when $d(A_0, A) = 2$. Then there exists $A_1 \in V(O_k)$ such that $A_0A_1A$ is a path of length 2 in $O_k$. This means that $A_0 \cap A_1 = \phi = A_1 \cap A$. Hence $A_0 \cup A \subseteq X \setminus A_1$, and so $k - 1 = |A_0| \leq |A_0 \cup A| \leq |X \setminus A_1| = (2k - 1) - (k - 1) = k$. Consequently, $k - 1 \leq |A_0 \cup A| \leq k$. But $A_0 \neq A$ and so $|A_0 \cup A| \neq k - 1$. Thus $|A_0 \cup A| = k$ and $|A_0 \cap A| = |A_0| + |A| - |A_0 \cup A| = (k - 1) + (k - 1) - k = k - 2$. Thus $d(A_0, A) = 2$ implies that

$$|A_0 \cap A| = k - 2. \quad (3.1)$$

Conversely, assume that $|A_0 \cap A| = k - 2$. Set $Y = A_0 \cap A$. Then $A_0 = Y \cup \{y_0\}$, and $A = Y \cup \{y\}$ for same $y_0, y$ in $X \setminus Y, y_0 \neq y$. Let $B = X \setminus (Y \cup \{y_0, y\})$. Then $B \in V(O_k)$ and $A \cap B = \phi = A_0 \cap B$. Hence $A_0BA$ is a path of length 2 in $O_k$, and $d(A_0, A) \leq 2$. But $A_0 \cap A \neq \phi$ and so $A$ is nonadjacent to $A_0$ in $O_k$. Thus $d(A_0, A) = 2$. Therefore

$$d(A_0, A) = 2 \text{ iff } |A_0 \cap A| = k - 2. \quad (3.2)$$
So we assume that \(d(A_0, A) = i > 2\) and that the result is true if \(d(A_0, A) \leq i - 1\). We complete the proof by considering the following two cases.

**Case 1: \(i\) is odd**

There exist vertices \(A_1, A_2, \ldots, A_{i-1}\) of \(O_k\) such that \(A_0A_1 \ldots A_{i-1}\) is a path of length \(i\) in \(O_k\). As \(d(A_0, A_{i-1}) = d(A_1, A) = i - 1\), by our induction assumption (as \(i - 1\) is even)

\[
|A_0 \cap A_{i-1}| = |A_1 \cap A| = k - \frac{i + 1}{2}.
\]

(3.3)

Since \(A_1 \cap A_0 = \emptyset\), by (3.3) we get

\[
|A_0 \cap A| \leq |A| - |A \cap A_1| = (k - 1) - \left(k - \frac{i + 1}{2}\right)
= \frac{i - 1}{2}.
\]

(3.4)

Moreover, as \(d(A_0, A) = i\), \(d(A_0, A) \not\in \{1, 3, \ldots, i - 2\}\) (by induction assumption), we get, \(|A_0 \cap A| = \frac{i-1}{2}\).

Conversely, assume that \(|A_0 \cap A| = \frac{i-1}{2}\). Set \(Y = A_0 \cap A\), and \(s = k - \frac{i+1}{2}\). Let \(A_0 = Y \cup \{a_1, \ldots, a_s\}\), and \(A = Y \cup \{b_1, \ldots, b_s\}\), where \(\{a_1, \ldots, a_s\} \cap \{b_1, \ldots, b_s\} = \emptyset\). Further assume that

\[Z = X \setminus (A_0 \cup A).\]

Now choose \(\{z_1, z_2\} \in Z\), (since \(i > 2\), \(|Z| \geq 2\)) and \(y \in Y\) and set

\[B = (Z \setminus \{z_1, z_2\}) \cup \{a_1, \ldots, a_s, y\}.
\]

Then

\[
|B| = |Z| - 1 + s
= \frac{i + 1}{2} - 1 + \left(k - \frac{i + 1}{2}\right)
= k - 1.
\]
Thus $B \in V(O_k)$, and $|A_0 \cap B| = s + 1 = k - \frac{(i-3)+2}{2}$, and $|A \cap B| = 1$.

Now as $i$ is odd, $i - 3$ is even, and by the induction hypothesis, $d(A_0, B) = i - 3$. We can take, $A = \{ y, x_1, \ldots, x_{k-2} \}$, and $B = \{ y, y_1, \ldots, y_{k-2} \}$ (where no $x_i$ is equal to any $y_j$). Let $X \setminus (A \cup B) = \{ v, w \}$. If we take $A_1 = \{ v, y_1, \ldots, y_{k-2} \}$, and $A_2 = \{ w, x_1, \ldots, x_{k-2} \}$, then $AA_1A_2B$ is a path of length 3 in $O_k$. Hence $d(A, B) \leq 3$, and therefore, $d(A_0, A) \leq d(A_0, B) + d(A, B) \leq (i - 3) + 3 = i$. By induction hypothesis, this implies that $d(A_0, A) = i$.

**Case 2: $i$ is even**

Let $d(A_0, A) = i$. So there exist vertices $A_1, \ldots, A_{i-1}$ of $O_k$ such that $A_0A_1 \ldots A_{i-1}A$ is a path of length $i$ in $O_k$. As $i$ is even, by induction assumption, $|A_0 \cap A_{i-1}| = k - \frac{i}{2}$, and $|A_0 \cap A_{i-1}| = \frac{i-2}{2}$. Therefore $|A_0 \cap A_{i-1}| + |A_0 \cap A_{i-2}| = (k - \frac{i}{2}) + \frac{i-2}{2} = k - 1 = |A_0|$. This means, since $A_{i-1} \cap A_{i-2} = \phi$, that $A_0 = (A_0 \cap A_{i-1}) \cup (A_0 \cap A_{i-2})$. Also, as $A \cap A_{i-1} = \phi$

\[
A \cap A_0 = A \cap [(A_0 \cap A_{i-1}) \cup (A_0 \cap A_{i-2})] \\
= A \cap A_0 \cap A_{i-2} \subseteq A_0 \cap A_{i-2}. \tag{3.5}
\]

Now, $A_0 \cap A \neq A_0 \cap A_{i-2}$. Otherwise, by induction hypothesis, (as $d(A_0, A_{i-2}) = i - 2$), $|A_0 \cap A_{i-2}| = k - \frac{i}{2} = |A_0 \cap A|$ and hence $d(A_0, A) = i - 2$, a contradiction. Therefore

\[
|A_0 \cap A| \leq k - \frac{i + 2}{2}. \tag{3.6}
\]

Again since $d(A_{i-2}, A) = 2$, $|A \cap A_{i-2}| = k - 2$.

Hence there exist $x \in A$ and $y \in A_{i-2}$ such that

\[
A \cap A_{i-2} = A \setminus \{ x \} = A_{i-2} \setminus \{ y \}. \tag{3.7}
\]
Chapter 3. Wiener Index of Odd Graphs

Now, from (3.7), \( A_{i-2} \setminus \{ y \} \subseteq A \), \( (A_0 \cap A_{i-2}) \setminus \{ y \} \subseteq A_0 \cap A \), and hence

\[
|A_0 \cap A| \geq |A_0 \cap A_{i-2}| - 1 = k - \frac{i}{2} - 1 = k - \left( \frac{i + 2}{2} \right). \tag{3.8}
\]

From (3.6) and (3.8), it follows that if \( d(A_0, A) = i \), \((i \text{ even})\), then \( |A_0 \cap A| = k - \frac{i+2}{2} \).

Conversely, assume that

\[
|A_0 \cap A| = k - \frac{i + 2}{2}. \tag{3.9}
\]

Let \( Y = A_0 \cap A \), and \( A_0 = Y \cup \{ \alpha_1, \ldots, \alpha_s \} \), and \( A = Y \cup \{ \beta_1, \ldots, \beta_s \} \), where \( s = i/2 \).

Let \( Z_0 = Y \cup \{ \alpha_1, \beta_2, \ldots, \beta_s \} \). Then \( Z_0 \in V(O_k) \), \( |A_0 \cap Z_0| = k - \frac{i}{2} \), and \( |A \cap Z_0| = k - 2 \).

By assumption, these imply that

\[
d(A_0, Z_0) = i - 2, \quad \text{and} \quad d(Z_0, A) = 2.
\]

Hence \( d(A_0, A) \leq i \). But by (3.9) and by induction assumption, we have \( d(A_0, A) \geq i \). Thus \( d(A_0, A) = i \).

Table 3.1 which has been constructed using Lemma 3.2.1 gives the possible values for \( d(A_0, A) \) for a fixed vertex \( A_0 \) of \( O_k \) and the corresponding numbers \( |A_0 \cap A| \).

<table>
<thead>
<tr>
<th>Table 3.1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d(A_0, A) )</td>
</tr>
<tr>
<td>(</td>
</tr>
</tbody>
</table>
3.2. WIENER INDEX OF $O_k$

Remark 3.2.1: Table 3.1 shows that the diameter of the odd graph $O_k$ is $k - 1$. Indeed, if $V(O_k) = \{v_0, \ldots, v_{2k-2}\}$, one diametrical path is given by $A_0 A_1 \ldots A_{k-1}$, where

$$A_i = \{v_j : \frac{i}{2} \leq j \leq \frac{i}{2} + k - 2 \text{ if } i \text{ is even, and } k + \frac{i-1}{2} \leq j \leq 2k + \frac{i-1}{2} - 2 \text{ if } i \text{ is odd, }$$

$(j \text{ taken modulo } 2k - 1).$

Lemma 3.2.2:

Fix $A_0 \in V(O_k)$. Let $0 \leq j \leq k - 2$. Then the number of vertices $B$ of $O_k$ such that $|A_0 \cap B| = j$ is equal to

$$\binom{k-1}{j} \binom{k}{k-1-j} = \binom{k-1}{j} \binom{k}{1+j} = \frac{k}{1+j} \binom{k-1}{j}^2.$$

Proof: The proof follows from combinatorial considerations.

Lemma 3.2.3:

If $|A_0 \cap A| = j \leq \left\lfloor \frac{k-2}{2} \right\rfloor$, then $d(A_0, A) = (2j + 1)$, and if $|A_0 \cap A| = j \geq \left\lfloor \frac{k-2}{2} \right\rfloor + 1$, then $d(A_0, A) = 2(k - 1 - j)$.

Proof: The proof should be evident from Table 3.1.

The lemmas above enable us to obtain an expression for the Wiener index $W(O_k)$ of the odd graph $O_k$.

Theorem 3.2.1:

The Wiener index $W(O_k)$ of $O_k$ is given by

$$W(O_k) = \frac{1}{2} \binom{2k-1}{k-1} \left[ \sum_{j=0}^{\left\lfloor \frac{k-2}{2} \right\rfloor} \frac{(2j+1)k}{(1+j)} \binom{k-1}{j}^2 + \sum_{j=1+\left\lfloor \frac{k-2}{2} \right\rfloor}^{k-2} \frac{2(k - 1 - j)k}{(1+j)} \binom{k-1}{j}^2 \right].$$
CHAPTER 3. WIENER INDEX OF ODD GRAPHS

Proof: By lemmas 3.2.2 and 3.2.3, the sum of the distances from a fixed vertex $A_0$ to all the vertices of $O_k$ is given by

$$D = \sum_{j=0}^{[k/2]} \frac{(2j + 1)k}{(1 + j)} \binom{k-1}{j}^2 + \sum_{j=1+\lceil k/2 \rceil}^{k-2} \frac{2(k - 1 - j)k}{(1 + j)} \binom{k-1}{j}^2.$$  

(3.10)

As expected, the expression in (3.10) is independent of the vertex $A_0$ since we know the fact that $O_k$ is vertex-transitive. Therefore, we get

$$W(O_k) = \frac{1}{2} \binom{2k-1}{k-1} D,$$

which establishes the theorem.

Remark 3.2.2: In Tables 3.2 and 3.3, we give the Wiener index and the distance distribution for $O_k$ for $k = 1, \ldots, 12$.

Note 3.2.1: Odd graphs can be generalized to Kn"{e}sser graphs defined thus: Given the natural numbers $s$ and $t$ such that $2s < t$, the Kn"{e}sser graph $K_t^{(s)}$ is defined thus: The vertex set of $K_t^{(s)}$ is the set of $s$-element subsets of $T = \{1, \ldots, t\}$; two vertices in $K_t^{(s)}$ are adjacent iff their corresponding subsets of $T$ are disjoint. An exercise then is to obtain an expression for the Wiener number of Kn"{e}sser graphs for which a solution is available in the literature.
### 3.2. WIENER INDEX OF $O_k$

<table>
<thead>
<tr>
<th>$k$</th>
<th>$W$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>75</td>
</tr>
<tr>
<td>4</td>
<td>1435</td>
</tr>
<tr>
<td>5</td>
<td>25515</td>
</tr>
<tr>
<td>6</td>
<td>436821</td>
</tr>
<tr>
<td>7</td>
<td>7339332</td>
</tr>
<tr>
<td>8</td>
<td>121782375</td>
</tr>
</tbody>
</table>

Table 3.2: $W(O_k)$ and distance distribution in $O_k$ for $k = 1, \ldots, 8$. 
**CHAPTER 3. WIENER INDEX OF ODD GRAPHS**

\[ k = 9 \quad W = 2005392675 \]

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>72</td>
<td>288</td>
<td>1008</td>
<td>2352</td>
<td>4704</td>
<td>7056</td>
<td>8826</td>
</tr>
</tbody>
</table>

\[ k = 10 \quad W = 32835436777 \]

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>90</td>
<td>405</td>
<td>1620</td>
<td>4320</td>
<td>10080</td>
<td>17640</td>
<td>26460</td>
<td>31752</td>
</tr>
</tbody>
</table>

\[ k = 11 \quad W = 535550923908 \]

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>110</td>
<td>550</td>
<td>2475</td>
<td>7425</td>
<td>19800</td>
<td>39600</td>
<td>69300</td>
<td>97020</td>
<td>116424</td>
</tr>
</tbody>
</table>

\[ k = 12 \quad W = 8707954295033 \]

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>132</td>
<td>726</td>
<td>3630</td>
<td>12100</td>
<td>36300</td>
<td>81675</td>
<td>163350</td>
<td>261360</td>
<td>365904</td>
<td>426999</td>
</tr>
</tbody>
</table>

**Table 3.3:** \( W(O_k) \) and distance distribution in \( O_k \) for \( k = 9, \ldots, 12 \).
Chapter 4

A lower bound for $W(G)$

4.1 Introduction

In this chapter, we give a lower bound for $W(G)$ in terms of the order and the diameter of a graph $G$. That the bound is sharp is illustrated by example graphs attaining the bound. We first restate the definition of Wiener index. Let $S_2(G)$ be the set of all 2-subsets derived from $V(G)$ i.e., the set of all unordered pairs of distinct vertices of $G$ so that $|S_2(G)| = n(n - 1)/2$. Then

$$W(G) = \sum_{\{u,v\} \in S_2(G)} d(u, v). \quad (4.1)$$

4.2 A sharp lower bound

Theorem 4.2.1:

Let $G$ be a graph with $d = diam(G) \geq 2$. Then
CHAPTER 4. A LOWER BOUND FOR $W(G)$

$$W(G) \geq \begin{cases} 
 n(n-1) - m + \binom{d}{3} + \frac{(n-d-1)(d-3)^2}{4}, & \text{if } n \text{ is odd.} \\
 n(n-1) - m + \binom{d}{3} + \frac{(n-d-1)(d-4)(d-4)}{4}, & \text{if } n \text{ is even.}
\end{cases}$$

(4.2)

**Proof:** Let $P = u_0u_1 \ldots u_d$ be a diameteral path of $G$, so that $d(u_0, u_d) = d = \text{diam}(G)$. We partition $S_2(G)$ into disjoint sets $X$, $Y$, and $Z$ defined as

$X = \{\{u, v\} \in S_2(G) | \text{both } u, v \in P\}$,

$Y = \{\{u, v\} \in S_2(G) | \text{none of } u \text{ and } v \text{ is in } P\}$, and

$Z = \{\{u, v\} \in S_2(G) | \text{one of } u \text{ and } v \text{ alone is in } P\}$.

It follows that

$|X| = \frac{d(d+1)}{2}$; $|Y| = \frac{(n-d-1)(n-d-2)}{2}$; $|Z| = (n-d-1)(d+1)$.

From (4.1), we have

$$W(G) = \sum_{\{u,v\} \in S_2(G)} (2 + (d(u,v) - 2))$$

$$= \sum_{\{u,v\} \in S_2(G)} 2 + \sum_{\{u,v\} \in S_2(G)} (d(u,v) - 2) \quad (4.3)$$

$$= n(n-1) + \sum_{\{u,v\} \in S_2(G)} (d(u,v) - 2) + \sum_{\{u,v\} \in S_2(G)} (d(u,v) - 2)$$

(since there are $n(n-1)/2$ elements in $|S_2(G)|$)

$$= (n(n-1) - m) + \sum_{\{u,v\} \in S_2(G)} (d(u,v) - 2)$$

$$\geq (n(n-1) - m) + \sum_{\{u,v\} \in X \cup Z} (d(u,v) - 2). \quad (4.4)$$
4.2. A SHARP LOWER BOUND

For $0 \leq k \leq (d - 1)$ in $X$, there are $(d - k)$ pairs $\{u, v\}$ with $d(u, v) = 1 + k$. Hence

$$
\sum_{\substack{\{u, v\} \in X \\ d(u, v) \geq 2}} (d(u, v) - 2) = (d - 2)1 + (d - 3)2 + \cdots + 1(d - 2)
$$

$$
= \frac{d(d - 1)(d - 2)}{6}. \quad (4.5)
$$

We next obtain a lower bound for the summation term on the right-hand side of (4.4). We begin by assuming that $d \geq 5$. Next we mark one vertex in $V(G) \setminus V(P)$, where $V(P)$ is the set of vertices in $P$. Then, by the triangle inequality for $0 \leq i \leq (d - 3)/2$ we have

$$
d(u_i, v) + d(v, u_{d-i}) \geq d(u_i, u_{d-i}) = d - 2i.
$$

Therefore, for each of the $(n - d - 1)$ choices of $v$, we have, by adding terms for which $d(u, v) = 1$ (so that $d(u, v) - 2 = -1$), we get

$$
\sum_{d(u, v) \geq 2} (d(u, v) - 2) 
\geq \sum_{i=0}^{d} (d(u_i, v) - 2) 
\geq \sum_{i=0}^{\left\lfloor \frac{d-2}{2} \right\rfloor} (d(u_i, v) + d(u_{d-i}, v) - 4) \quad (4.6)
$$

$$
\geq \sum_{i=0}^{\left\lfloor \frac{d-2}{2} \right\rfloor} (d - 2i - 4). \quad (4.7)
$$

Since each summand on the right side of (4.6) is nonnegative, so is each summand on the right side of (4.7). Hence the term on the right side of
(4.6) is
\[
W(G) \geq \begin{cases} 
(d - 4) + (d - 6) + \cdots + 5 + 3 + 1 & \text{if } d \text{ is odd, and} \\
(d - 4) + (d - 6) + \cdots + 2 & \text{if } d \text{ is even.} 
\end{cases}
\]
\[
= \begin{cases} 
\frac{(d - 3)^2}{4} & \text{if } d \text{ is odd, and} \\
\frac{(d - 2)(d - 4)}{4} & \text{if } d \text{ is even.} 
\end{cases}
\]

Thus, for each fixed \( v \in V(G) \setminus V(P) \), we have
\[
\sum_{(u, w) \in Z} (d(u, v) - 2) = \begin{cases} 
\frac{(d - 3)^2}{4} & \text{if } d \text{ is odd, and} \\
\frac{(d - 2)(d - 4)}{4} & \text{if } d \text{ is even.} 
\end{cases}
\]

In conclusion, we have for \( d \geq 5 \)
\[
W(G) \geq \begin{cases} 
n(n - 1) - m + \binom{d}{3} + (n - d - 1)\left(\frac{d - 3}{2}\right)^2, & \text{if } n \text{ is odd,} \\
n(n - 1) - m + \binom{d}{3} + (n - d - 1)\frac{(d - 2)(d - 4)}{4}, & \text{if } n \text{ is even.} 
\end{cases}
\]

We now consider the cases when \( d = 2, 3 \) and 4. If \( d = 2, d(u, v) = 1 \) or 2. Hence from (4.3) we get
\[
W(G) = n(n - 1) + \sum_{d(u, v) = 1} (1 - 2) \\
= n(n - 1) - m. 
\]

(4.8)

Now consider the case when \( d = 3 \). Then \( d(u, v) = 1, 2 \) or 3. Hence from (4.3) it follows that
\[
W(G) = n(n - 1) - m + \sum_{d(u, v) = 3} (d(u, v) - 2) \\
\geq n(n - 1) - m + 1, 
\]
as there is at least one pair with $d(u, v) = 3$. If $d = 3$, $\frac{d(d-1)(d-2)}{6} = 1$. Hence we have

$$W(G) \geq n(n-1) - m + \frac{d(d-1)(d-2)}{6}.$$  

Finally if $d = 4$ we have

$$W(G) \geq n(n-1) - m + \sum_{d(u,v)=3 \text{ or } 4} (d(u,v) - 2)$$

$$\geq n(n-1) - m + 2 + 2.$$  

(as $d(u_0, u_d) = d = 4$, $d(u_0, u_3) = 2 = d(u_1, u_4)$)

Hence $W(G) \geq n(n-1) - m + \frac{d(d-1)(d-2)}{6}$.  

(as $\frac{d(d-1)(d-2)}{6} = 4$ in this case.)

This completes the proof. 

$\blacksquare$

**Remark 4.2.1:** Let $G$ be a graph of order $n$, maximum degree $\Delta$ and diameter $d$. The Moore bound (see for example [MS05]) gives an upper bound for $n$ in terms of $\Delta$ and $d$:

$$n \leq \begin{cases} 
1 + \frac{(\Delta - 1)^d - 1}{\Delta - 2} & \text{if } \Delta > 2, \\
2d + 1 & \text{if } \Delta = 2.
\end{cases} \quad (4.9)$$

From (4.9) the lower bound on $d$ may be used in (4.2).

**Remark 4.2.2:** We observe that the lower bound given in (4.2) is sharp. The graphs $C_3 \times K_2$, path on $n$ vertices and the Petersen graph attain the bound. The lower bound is also attained for the following families of graphs $G$:

(i) $G \cong K_{1,2k+1}$, $k \geq 1$.

(ii) With $d = 2$ and even $n$, take $G$ as in Fig. 4.1.
(iii) With $d = 4$ and even $n$, take $G$ as in Fig. 4.2.

(iv) For an even $d$, take $G$ as in Fig. 4.3.

(v) For an odd $d$, take $G$ as in Fig. 4.4.

Figure 4.1: A family of graphs of diameter 2.

Figure 4.2: A family of graphs with diameter 4.
4.2. A SHARP LOWER BOUND

Figure 4.3: A graph with $d$ specified as even.

Figure 4.4: A graph with $d$ specified as odd.
Chapter 5

\(W(G)\) on a bus-architecture

5.1 Introduction

As remarked earlier, given a graph \(G = (V, E)\) the sequential approach to find \(W(G)\) is to apply the \(BFS\) procedure once from every vertex in \(V\). This approach gives a time complexity of \(O(n^2 + nm)\), where \(n = |V|\) and \(m = |E|\). Here, we outline an existing distributed algorithm for the all-pairs shortest path problem so as to compute \(W(G)\). We then present our approach which constitutes a mapping of the distributed algorithm onto a bus-topology based architecture. The approach is flexible because it assumes a common bus-based architecture independent of the connectivity of the input graph. The memory required at each processor is bounded by \(O(n)\). For an extensive treatment of distributed algorithms in general, references [Bar96], [Lyn96], and [Tel94] may be consulted.
5.2 A DISTRIBUTED APPROACH

5.2 A distributed approach

Following [Bar96], we present the outline of a distributed algorithm that computes the distances between all node pairs \( u, v \in V \) and hence \( W(G) \). For distributed algorithms for shortest paths, based on the sequential BFS and the Bellman-Ford algorithm, [Lyn96] may be consulted.

5.2.1 The distributed algorithm in brief

Given the input graph \( G = (V, E) \), we construct an architecture where each vertex in \( V \) is represented by a processor with memory and control functions and each undirected edge in \( E \) is replaced by a two-way communication line connecting the processors representing the end-vertices. We assume that each processor has a unique label and this label or identifier is known to the processor. We assume that there is an external control unit which can activate all the processors synchronously by a common clock pulse. In the assumed synchronous model, the following two points are noteworthy:

(i) All processors are driven by a global time basis, referred to as the global clock which generates time intervals of time duration \( T > 0 \).

(ii) For some duration \( \delta \geq 0 \) that is less than \( T \), the local computation takes no more than \( \delta \) time, while the time for the messages to be delivered from any one processor to a neighboring one is strictly less than \( T - \delta \).

The synchronous algorithm \( \text{SynComputeDistances} \) is run on each processor. It proceeds as follows. During the clock pulse at time \( t = 0 \), every vertex i.e., the corresponding processor sends its identifier to all of its
neighbors. During the clock pulse at time $t = 1$, every vertex has the labels of all vertices that are no farther from it than one edge. A processor then builds a set with the identifiers of all those processors that are exactly one edge away from it and sends this set to its neighbors. During clock pulse at time $t = 2$, every vertex should have received the labels of all vertices located no farther than two edges from it. Because a vertex knows precisely which vertices are zero or one edge away from it, determining the set of those vertices that are exactly two edges away is easy. In general, during clock pulse at time $t \geq 0$ a processor sends to its neighbors a set containing the labels of all those processors that are exactly $t$ edges away from it. As given in [Bar96], no more than $n$ clock pulses are required for computing the distances between all vertex pairs in $G$. The last clock pulse, is the pulse at $t = n - 1$. We note that if $d$ is the diameter of $G$, then no more than $d$ clock pulses are required. The worst-case arises only in the case of a path. The control unit computes the Wiener index from the distance information in the different processors during clock pulse at time $t = n$.

5.2.2 Remarks on algorithm SynComputeDistances

Algorithm SynComputeDistances is described in [Bar96] where a correctness argument can also be seen. During a run of this algorithm, every vertex must receive the identifier of every other vertex. It can be seen that every processor’s identifier must traverse every edge in $E$ in both directions. If we consider an identifier of a processor as a message, then the number of messages used in SynComputeDistances is $2mn$. Then algorithm SynComputeDistances has a message complexity of $O(mn)$.

A drawback of algorithm SynComputeDistances is that it relies on
an underlying hardware architecture that fully represents the connectivity
structure of the input graph \( G \). That is, for every new input instance of \( G \) the
underlying network connections changes. The algorithm suggested in the
next section overcomes this by using a bus-based architecture and mapping
the input graph \( G \) onto that architecture rather than building a new network
for every input instance of \( G \).

5.3 A bus-based approach

We now present our algorithm, which employs a bus-based architecture
([Fen81], [Gus84]) as the underlying processor interconnection network. In
the considered bus-based architecture, we assume that the processors are all
directly connected to a shared bus through which the processors can com-
municate by message-passing. We also assume that there is a control unit,
which can grant bus access to only one processor at a time and can activate
the processors by a common clock pulse. Thus only the processor that has
access to the bus transmits message at one time while all the other proces-
sors can receive the transmitted message. Since a processor cannot transmit
asynchronously, time is divided into epochs of sufficient length \( T \) – evi-
dently, \( T \) depends on the number of vertices in the input graph. Each epoch
of time duration \( T_i \) is further subdivided into time slots. Thus we can denote
by \( T_{ij} \), the time duration corresponding to \( i \)th epoch and \( j \)th time slot in it.
During this \( j \)th time slot, only processor \( j \) will be activated by the com-
mon clock, to transmit messages. This will ensure that there is no collision
on the bus between message transmissions due to two or more processors.
Given the input graph \( G \), each vertex in \( V \) is mapped in an arbitrary manner
onto one of the processors. Each processor thus represents one vertex of $G$ and this vertex’s identifier is known to the corresponding processor. Initially, a vertex’s neighboring vertices i.e., their identifiers are also loaded in the corresponding processor. The algorithm `BusBasedComputation` that follows, essentially simulates `SynComputeDistances`. The pseudo-code that should run in every processor is presented below following a notation similar to that used in [Bar96].

### 5.3.1 Algorithm `BusBasedComputation`

We begin with the terminology and notations used in the description that follows.

- $N$ is the set of all identifiers of the vertices in $V$. $|N| = n$, $n > 1$.
- $n_i$ $(1 \leq i \leq n)$ will refer to the vertices.
- $id_i$ is the unique identifier of the vertex $n_i$.
- $neighbors_i$ indicates the set of identifiers of the neighbors of the vertex $n_i$.
- $reached_i^j$ indicates whether the vertex $n_i$ has received $id_j$ at least once ($i \neq j$).
- $dist_i^j$ stores the distance of the vertex $n_j$ from the vertex $n_i$.
- $T_i$ is the duration of the $i$th-epoch.
- $T_{ij}$ is the $j$th slot in the $i$th-epoch.
- $mem_i : T_i$ is the set of values received (and stored) in a processor during the epoch $T_i$. 
5.3. A BUS-BASED APPROACH

- \textit{ReachedAll} is an interrupt raised by the processor \( n_x \) when the boolean \( reached^j_x = \text{true}, \forall n_j \in N \).

- \textit{StopAll} is an interrupt raised by the control unit at the start of an epoch. This interrupt, which is raised when all the processors have raised \textit{ReachedAll}, forces all the processors to return the stored distance information.

Algorithm BusBasedComputation:

Let this processor’s identifier be \( n_x \);

Initialization:

\[
\text{dist}^j_i = n \ \forall n_j \in N;
\]

\[
\text{reached}^j_i = \text{false} \ \forall n_j \in N;
\]

Action:

During the time slot \( T_{ij} \):

\begin{align*}
\text{if (StopAll is raised)} & \\
\quad \text{then} & \\
\qquad \text{begin} & \\
\qquad \quad \text{if } (x = j) & \\
\qquad \qquad \text{then} & \\
\qquad \qquad \quad \text{begin} & \\
\qquad \qquad \quad \quad \text{return } \frac{1}{2} \sum \text{dist}^0_x \ \forall n_y \in N; & \\
\qquad \qquad \quad \quad \text{stop processing;} & \\
\qquad \qquad \quad \text{end} & \\
\qquad \quad \text{else} & \\
\qquad \quad \quad \text{Wait for the slot } T_{ij} \text{ to get over;}
\end{align*}
end
else if \((i = 0)\)
    then
        begin
            \(\text{mem} : T_0 = n_x;\)
            \(\text{reached}_x^0 = \text{true};\)
            \(\text{dist}_x^0 = 0;\)
        end
else if \((i = 1)\)
    then
        begin
            \(\text{mem} : T_1 = \text{neig}_x;\)
            \(\text{reached}_x^1 = \text{true} \ \forall y \in \text{neig}_x;\)
            \(\text{dist}_x^1 = 1 \ \forall y \in \text{neig}_x;\)
            raise ReachedAll
            if \(\text{reached}_x^0 = \text{true} \ \forall n_y \in N;\)
        end
else begin
    if \((j = x)\) then
        begin
            Transmit \(id_x;\)
            Transmit \(id_y \ \forall y \in \text{mem} : T_{i-1};\)
        end
    else begin
        read \(id_j;\) //identifier of the sender
        if \((j \in \text{neig}_x)\) then

begin
\forall id_k \text{ that } n_j \text{ transmits }
\textbf{if } (\text{reached}_x^k = \text{false}) \text{ then }
\begin{align*}
\text{begin} \\
\quad & \text{reached}_x^k = \text{true}; \\
\quad & \text{dist}_x^k = i; \\
\quad & \text{Load } k \text{ in } \text{mem} : T_i; \\
\quad & \text{raise ReachedAll} \\
\quad & \text{if } \text{reached}_y^p = \text{true} \ \forall n_y \in N; \\
\end{align*}
\textbf{end}
\textbf{end}
\textbf{else}
\begin{align*}
\quad & \text{Wait for the slot } T_{ij} \text{ to get over}; \\
\end{align*}
\textbf{end}
\textbf{end}

When all the processors raise the \textit{ReachedAll} interrupt, the control unit raises the \textit{StopAll} interrupt halting the computations in all the processors and to return the distance information, from which the Wiener index can be computed.

\textbf{Complexity outline:}

If it is given that \(diam(G) = d\), algorithm \textit{BusBasedComputation} requires not more than \(d\) epochs for an execution. All the identifiers that are transmitted including the identifier of the transmitting processor together with the associated control characters form a message. During each slot, there is a transfer of a message on the bus. Therefore, the algorithm has a
message complexity of $O(dn)$. If the input graph $G$ is represented as an adjacency matrix to provide the connectivity information, any sequential algorithm will require $O(n^2)$ space to represent the input. For the algorithm BusBasedComputation, connectivity information from one vertex alone is required at each processor. Thus at each processor $O(n)$ space is sufficient.

5.4 Illustration

Consider the labeled graph $G$ shown below in Fig.5.1.

![Figure 5.1: An example labeled graph $G$.](image)

In the bus-based architecture model, we will now refer to the identifiers of the processors as well as the processors themselves by the respective vertex labels of $G$. We assume that nodes with labels $A, B, C, D,$ and $E$ are respectively mapped onto processors $P_1, P_2, P_3, P_4, P_5$. A simulated run of the algorithm is captured in table 5.1.

During the epoch $T_0$, the control unit loads each processor with its respective identifier. During the epoch $T_1$, at each processor, it loads the identifiers of the processor’s neighbors. At each processor, the received identi-
Table 5.1: A run of the algorithm on the graph in Fig.5.1

<table>
<thead>
<tr>
<th></th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
<th>$P_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_0$</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
</tr>
<tr>
<td>$T_1$</td>
<td>BC</td>
<td>ACD</td>
<td>ABD</td>
<td>BCE</td>
<td>D</td>
</tr>
<tr>
<td>$T_{2,1}$</td>
<td>-</td>
<td>BC</td>
<td>BC</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$T_{2,2}$</td>
<td>acD</td>
<td>-</td>
<td>acd</td>
<td>Acd</td>
<td>-</td>
</tr>
<tr>
<td>$T_{2,3}$</td>
<td>abd</td>
<td>abd</td>
<td>-</td>
<td>abd</td>
<td>-</td>
</tr>
<tr>
<td>$T_{2,4}$</td>
<td>-</td>
<td>bcE</td>
<td>bcE</td>
<td>-</td>
<td>BCe</td>
</tr>
<tr>
<td>$T_2$</td>
<td>D</td>
<td>E</td>
<td>E</td>
<td>A</td>
<td>BC</td>
</tr>
<tr>
<td>$T_{3,1}$</td>
<td>-</td>
<td>d</td>
<td>d</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$T_{3,2}$</td>
<td>E</td>
<td>-</td>
<td>e</td>
<td>e</td>
<td>-</td>
</tr>
<tr>
<td>$T_{3,3}$</td>
<td>e</td>
<td>e</td>
<td>-</td>
<td>e</td>
<td>-</td>
</tr>
<tr>
<td>$T_{3,4}$</td>
<td>-</td>
<td>a</td>
<td>a</td>
<td>-</td>
<td>A</td>
</tr>
<tr>
<td>$T_{3,5}$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>bc</td>
<td>-</td>
</tr>
<tr>
<td>$T_3$</td>
<td>E</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>A</td>
</tr>
</tbody>
</table>

fiers during subsequent epochs are also shown in table 5.1. As evident from the algorithms above, during any epoch, whenever the slot is equal to $x$, it indicates that it is process $P_x$’s turn. Then processor $P_x$ transmits its identifier first, followed by the identifiers of all the processors, that it received during the previous epoch. For the epoch $T_1$ and $T_2$, we show the details of the identifiers received by the different processors in table 5.1. A received identifier when discarded is shown in small letters. Since $\text{diam}(G) = 3$ in this case at the end of $T_3$, all the computations get completed.
5.5 Concluding remarks

We have described a non-sequential approach to computing the Wiener index of a graph on a bus-based architecture. The assumed computational model is simple: a set of identical processors communicating sequentially via a common bus. In particular we have shown a mapping of a modification of an existing distributed algorithm onto a bus-based architecture. Our approach is more general albeit at the cost of increased message complexity. It is general in the sense that a bus architecture has to be built only once for any number of input instances of graphs having an order less than a fixed bound. In the assumed model, the space complexity at each processor is $O(n)$. Our algorithm may be made more general, to deal with certain types of graphs of large order, in the following way. The input graph may be “suitably partitioned” and mapped onto the available processors. Each processor can then exchange (with the other processors) the local information on the partitions and eventually compute all the distances between vertex pairs. We remark that the idea of mapping an arbitrary graph onto a bus-based architecture appears to be general. Other distributed graph algorithms may also be similarly mapped onto a bus-based architecture to remove the dependency of hardware on the input graph structure.
Chapter 6

Finding an optimum subtree

6.1 Introduction

Finding a subtree of a given tree, with certain prescribed properties has applications in facilities location, which is concerned with the location of facilities on a network. A subtree, for example can be a core of a tree which is defined to be a path that is optimal with respect to the property of minimizing the sum of the distances from each vertex in the tree to the path. In this chapter, we consider the problem of finding a subtree, of a given tree, satisfying the certain given constraints; we present an efficient algorithmic solution to the considered problem. It follows that this general problem can be specialized to yield the Wiener index of a tree in time linear in the number of vertices. We also give an application of the solution to efficiently compute the Wiener indices of all the trees that arise in the cut-vertex/cut-edge lemma.
6.2 The Opt-SUBTREE problem

We consider an edge weighted tree \( T = (V(T), E(T)) \) with an arbitrary designated root \( r \in V(T) \). We denote the size \( |V(T)| \) of the tree \( T \) by \( n \) and the degree of a vertex \( v \in V(T) \) by \( \text{deg}_T(v) \). We also denote by \( wt(u, v) \), the weight (a real number) associated with the edge \( uv \in E(T) \).

Given two distinct vertices \( u, v \in V(T) \), \( d(u, v) \) will denote the weighted distance (sum of the weights on the edges in the path connecting \( v \) and \( u \)) between them. We define \( d(v, v) = 0 \) for all \( v \in V(T) \). For a subtree \( T' \) of \( T \) and a vertex \( v \in V(T) \), we define the distance \( d(v, T') \) as

\[
d(v, T') = \min_{x \in V(T')} \{d(v, x)\}.
\]

We also define the distance \( d(T, T') \) between the tree \( T \) and its subtree \( T' \) as

\[
d(T, T') = \sum_{v \in V(T)} d(v, T').
\]

We also refer to \( d(T, T') \) as the cost of \( T' \) (with respect to the tree \( T \)).

The weighted eccentricity \( \text{we}(v) \) of a vertex \( v \in V(T) \) is defined as

\[
\text{we}(v) = \max_{x \in V(T)} \{d(v, x)\}.
\]

In dealing with the weighted eccentricity, we can refer to a weight on an edge as the edge length. The total distance \( td(v) \) of a vertex \( v \in V(T) \) is defined as

\[
td(v) = \sum_{x \in V(T)} d(v, x).
\]

We also define a median vertex of \( T \) as any vertex \( v \in V(T) \) such that \( td(v) \) is minimum. It follows that we have given extended definitions considering
6.3. AN ALGORITHM FOR OPT-SUBTREE

the weights on the edges of $T$. Compare these with the standard definitions, for example, as given in [GY06]. For every $v \in V(T)$, we specify a degree bound $db(v)$ which is an integer such that $db(v) \leq deg_T(v)$.

We now define the problem Opt-SUBTREE as follows:

**Problem Opt-SUBTREE**: Find a subtree $T'$ of $T$ such that

$$d(T, T')$$ is the minimum and for all $v \in V(T')$ $deg_{T'}(v) \leq db(v)$.

In this chapter, we give an algorithm which solves Opt-SUBTREE. For each $v \in V(T)$, our algorithm will also find $we(v)$ and $td(v)$. It is easy to see that when we specify $db(v) = 0$ for all $v \in V(T)$, an algorithm for Opt-SUBTREE will output a median vertex of $T$. To output a path we specify $db(v) = 2$ for all $v \in V(T)$. This case may be compared with the problem of finding the $l$-core of a tree [BCL+02].

6.3 An algorithm for Opt-SUBTREE

The algorithm that we give, consists of two passes over the input tree $T$. Starting from the root $r$, each pass visits the vertices of $T$ in a depth-first search (DFS) order (see [CLRS01] for details). We begin with the following notions and notations (to be interpreted in the sense of a program variable/procedure) used in the algorithm that we describe subsequently.

6.3.1 Notations and terminology

We assume that $v \in V(T)$ and $v$ is a non-leaf.
• $\pi(v)$ will denote the parent of a vertex $v \in V(T) \setminus \{r\}$; then, the vertex $v$ will be referred to as the child of the vertex $\pi(v)$. We define $\pi(r) = \text{nil}$.

• $N(v)$ will denote the number of vertices in the subtree (of $T$) with the root as the vertex $v \in V(T)$. Thus, $N(r) = n$.

• $D(v)$ will denote a distance function at the vertex $v \in V(T)$: During the first pass, we compute $D(v)$ as $D(v) = \sum_{x \in V(S)} d(v, x)$, where $S$ is the subtree (of $T$) with vertex $v$ as the root. Initially, we set $D(u) = 0$ for all leaves $u$. In the second pass, we update $D(v)$ to $td(v)$.

• $e_1(v)$ and $e_2(v)$ will denote respectively, the maximum and the second maximum eccentricity contributions at the vertex $v$ due to the children of $v$ (in the description of the algorithm below, it will become clear as to how these quantities are computed). Initially, we set $e_1(u) = 0$ and $e_2(u) = 0$ for all leaves $u$.

• $e(v)$ will eventually denote $\text{we}(v)$ for all $v \in T$. Used for the sake of clarity, $e(v)$ will denote the weighted eccentricity of $v$ in the subtree rooted at $v$ at the end of the first pass. After the second pass, $e(v)$ will get updated to $\text{we}(v)$. It should become clear that $e(v)$ will always refer to $e_1(v)$.

• $G(v, w)$, where $v = \pi(w)$, is a gain function used in the process of building up the optimal subtree $T'$. It is the reduction in cost of the partially constructed optimal subtree, if the optimal subtree rooted at $w$ is also included.
6.3. AN ALGORITHM FOR OPT-SUBTREE

- **Gain** 
  \( \text{Gain}(v, \text{dgbd}, \text{vexclude}) \) is a function that returns a gain value at a vertex \( v \) if its degree bound is specified as \( \text{dgbd} \). It is the sum of the \( \text{dgbd} \) largest \( G(v, w) \) values, where \( vw \in E(T) \) and \( w \neq \text{vexclude} \). In other words, this function can be regarded as returning a value, by sorting, the \( G(v, w) \) values for all vertices \( w \) \( (w \neq \text{vexclude}) \) and summing up the first \( \text{dgbd} \) terms.

- **Patching** will refer to the updating of the quantities \( N(v) \), \( D(v) \), \( e_1(v) \), \( e_2(v) \) at each vertex \( v \in V(T) \) from the first pass to the second pass.

### 6.3.2 Details of the algorithm

As mentioned earlier, the algorithm visits the vertices of \( T \) in the DFS order in two passes. We first informally give the computations done in the first and the second passes of the algorithm and then we give a pseudo-code for the algorithm. Subsequently we provide justifications for these computations.

**I Pass**

Visiting of the vertices of \( T \), in this pass, is done in the “leaves-to-root” order – that is, for every vertex \( v \), \( v \) will be visited before the visit of \( \pi(v) \). During the visits, at each vertex \( v \), we compute the quantities \( N(v) \), \( D(v) \), \( e_1(v) \), \( e_2(v) \) and \( G(v, w) \), where \( w \) is adjacent to \( v \). For each vertex \( v \), we can compute these quantities once these quantities are known for all the children of \( v \). The quantities computed (recursively) at each vertex \( v \) of \( T \)
are given below.

\[ N(v) = 1 + \sum_{w} N(w), \text{ where } v = \pi(w). \]  
\[ D(v) = \sum_{w} (wt(v, w).N(w) + D(w)), \text{ where } v = \pi(w). \]  
(6.1)  
(6.2)

For each vertex \( w \), such that \( v = \pi(w) \), we obtain the set \( \{wt(v, w)+e(w)\} \).

\[ e_1(v), e_2(v) \text{ be the two largest in the set}\{wt(v, w)+e(w)\}. \]  
(6.3)

For each \( w \) such that \( v = \pi(w) \),

\[ G(v, w) = wt(v, w).N(w) + Gain(w, db(w) - 1, v). \]  
(6.4)

II Pass

In this pass, visiting of the vertices of \( T \) is done in the “root-to-leaves” order – that is, a vertex \( v \) is visited before any of its child is visited. We patch the different quantities at each vertex \( v \), so that they hold the various quantities as if \( v \) were the root of \( T \). For each vertex \( v \), the patching comprises the following computations. It should be noted that the order of these computations are significant.

\[ D(v) = D(\pi(v)) + wt(v, \pi(v)).(N(\pi(v)) - 2N(v)). \]  
(6.5)

Let \( k = \begin{cases} 
  e_1(\pi(v)), & \text{if } e_1(\pi(v)) \neq e_1(v) + wt(v, \pi(v)), \\
  e_2(\pi(v)), & \text{otherwise.}
\end{cases} \)
6.3. AN ALGORITHM FOR OPT-SUBTREE

Let $e_1(v), e_2(v)$ be the two largest in the list \( \{ e_1(v), e_2(v), wt(v, \pi(v)) + k \} \).

\begin{equation}
G(v, \pi(v)) = wt(v, \pi(v)) \{ n - N(v) \} + \text{Gain}(\pi(v), db(\pi(v)) - 1, v).
\end{equation}

\begin{equation}
N(v) = n.
\end{equation}

**Optimal degree-bounded subtree**

We first outline – omitting data structures – a procedure \textit{AddVertex} which builds a subtree $S$ of $T$. The procedure \textit{AddVertex} assumes the access to $S$ which is assumed to be declared globally.

\textbf{procedure} \textit{AddVertex}(v, deg, pr)

1: Add the vertex $v$ to $S$;

2: Find the $\text{deg}$ largest values of $G(v, w)$ excluding $pr$, where $v = \pi(w)$;

3: For each such $w$, call \textit{AddVertex}(w, db(w) - 1, v);

At the end of the II pass, we have the gain values at all the vertices of $T$. At any vertex $v \in V(T)$, the cost of the subtree rooted at $v$ is $D(v) - \text{Gain}(v, db(v), \text{nil})$. Therefore, to produce the required optimal subtree, we execute the following two steps:

1: Find a vertex $x$ in the tree $T$ such that

\[
\{ D(x) - \text{Gain}(x, db(x), \text{nil}) \} \text{ is the minimum.}
\]

2: Set $S$ to nil and call \textit{AddVertex}(x, db(x), S).
6.3.3 Code for the algorithm

To add clarity we provide a pseudo-code for our algorithm. In FindOptSubtree we assume that \( r \) is the root of the input tree \( T \) and the output is \( S \). The pseudo-codes for Compute and Patch should be obvious from the earlier descriptions.

Algorithm FindOptSubtree

\begin{verbatim}
begin
    call Compute(r,nil);
    call Patch(r, nil);
    let k be a node of minimal value for
    D(k)-Gain(k, db(k), nil);
    call BuildTree(k, db(k), nil, S);
end.
\end{verbatim}

function Compute(nd, pr)

\begin{verbatim}
begin
    D(nd) = 0; N(nd) = 1; e_1(nd) = e_2(nd) = 0;
    for (each vertex x adjacent to nd) do
       begin
           if x ≠ pr then
               begin
                   call Compute(x,nd);
                   N(nd) = N(nd) + N(x);
                   D(nd) = D(nd) + wt(nd,x).N(x) + D(x);
                   G(nd,x) = wt(nd,x).N(x) + Gain(x, db(x) - 1, nd);
               end;
       end;
\end{verbatim}
6.3. AN ALGORITHM FOR OPT-SUBTREE

Let \( L \) be a list of \( e_1(nd) \), \( e_2(nd) \), \( e(nd) + wt(x, nd) \);
Set \( e_1(nd) \) to largest in \( L \);
Set \( e_2(nd) \) to second largest in \( L \);

\[
\begin{align*}
\text{end};
\end{align*}
\]

\[
\begin{align*}
\text{end};
\end{align*}
\]

\[
\begin{align*}
e(nd) = e_1(nd);
\end{align*}
\]

\[
\begin{align*}
\text{end}.
\end{align*}
\]

function \( Patch(nd, pr) \)

\[
\begin{align*}
\text{begin}
\end{align*}
\]

\[
\begin{align*}
\text{if } \, pr \neq \text{nil} \, \text{then}
\end{align*}
\]

\[
\begin{align*}
\text{begin}
\end{align*}
\]

\[
\begin{align*}
D(nd) &= D(pr) + N(pr) - 2N(nd);
G(nd, pr) &= d(nd, pr).(n - N(nd)) + \text{Gain}(pr, db(pr) - 1, nd);
N(nd) &= n;
\text{if } \, e(pr) = e(nd) + d(nd, pr) \, \text{then} \, k = e_2(pr)
\end{align*}
\]

\[
\begin{align*}
\text{else} \, k = e_1(pr);
\end{align*}
\]

\[
\begin{align*}
\text{Let } \, L \, \text{be a list of } e_1(nd), e_2(nd), k + wt(nd, pr);\n\end{align*}
\]

\[
\begin{align*}
e_1(nd) &= \text{first largest in } L;\n\end{align*}
\]

\[
\begin{align*}
e_2(nd) &= \text{second largest in } L;\n\end{align*}
\]

\[
\begin{align*}
e(nd) &= e_1(nd);
\end{align*}
\]

\[
\begin{align*}
\text{end};
\end{align*}
\]

\[
\begin{align*}
\text{for} \, \text{(each vertex } x \text{ adjacent to } nd) \, \text{do}
\end{align*}
\]

\[
\begin{align*}
\text{if } \, x \neq pr \, \text{then} \, \text{call } Patch(x, nd);
\end{align*}
\]

\[
\begin{align*}
\text{end}.
\end{align*}
\]
function BuildTree(nd, dg, pr, T')
begin
\[ T' = T' \cup \{nd\}; \]
Let \( L_1 = \{g(x)|x, nd \text{ are adjacent, } x \neq pr; \]
and \( g(x) = gain(x, db(x) - 1, nd)\}; \]
Let \( L_2 = \text{list of first } dg \text{ largest } g(x) \text{ in } L_1; \]
for (each x such that \( g(x) \in L_2) \) do
\[
\text{call BuildTree(x,db(x)-1,nd,T');}
\end

6.3.4 Correctness and complexity

Justification of I pass:

We begin with a justification of equations (6.1) through (6.4) in I pass.

For any vertex \( v \), by the definition \( N(v) \), (6.1) is obvious. Consider a \( w \)
such that \( v = \pi(w) \). From \( v \), in order to reach all the vertices within the
subtree \( R \) rooted at \( w \), it is first required to traverse to \( w \). Hence, the
distance to any vertex in \( R \) from \( v \) is exactly \( wt(v, w) \) more than that from
\( w \). As there are \( N(w) \) vertices in \( R \), the increase in the distance (from \( v \)) is
\( wt(v, w).N(w) \). Hence \( D(w) + wt(v, w).N(w) \) is the sum of the weighted
distances from \( v \) to every vertex in \( R \). Taking the summation over all the
children of \( w \) yields (6.2).

The eccentricity of a vertex \( v \) is \( wt(v, w) \) plus the eccentricity of \( w \), since
the (weighted) farthest vertex from \( w \) in \( R \) will be farther still from \( x \) by
the edge length \( wt(x, y) \). Thus \( e_1(v) \) in (6.3) will correctly represent \( e(v) \).
Consider again, two vertices $v, w$ such that $v = \pi(w)$. $G(v, w)$ is the difference in the cost from $D(v)$ caused due to the inclusion of optimal subtree in $w$ (rooted at $w$) to be connected to $v$. The contribution (in distance) of the edge $vw$ in $D(v)$ is $wt(v, w)N(w)$. Therefore, if $w$ were included as a part of the optimal subtree, all vertices in its subtree will have to reach $w$. Hence, the edge cost is to be added to the gain. $G(v, w)$ also includes the gain with the subtree rooted at $w$, obeying the degree bounds. Since $v$ is connected to $w$, we take the degree bound on $w$ to be 1 less than $db(w)$. It follows that the associated gain by including the optimal subtree rooted at $w$ is $Gain(w, db(w) - 1, v)$.

**Justification of II pass:**

We consider vertices $x, y$ such that $y = \pi(x)$; we assume that $y$ is already patched (which implies $N(y) = n$).

Let us assume that the edge $xy$ is cut. Now the distance function, say $D'(y)$ (we will use similar notation for other quantities too) will be due to the contributions of other children of $y$ (i.e., the contribution due to $x$ is excluded). We also set $G(y, x) = 0$. We must ensure that $e(y)$ should not be reckoned by considering $x$. It is clear that $x$ could have changed $e(y)$ only if $x$ were the most eccentric child of $y$. Thus we consider the following two cases:

Case 1: $x$ was the most eccentric child of $y$ – therefore by deleting the edge $xy$, we should have $e'(y) = e_2(y)$.

Case 2: $x$ was not the most eccentric child of $y$ – by deleting the edge $xy$, nothing changes at $y$; so we have $e'(y) = e_1(y)$. 
CHAPTER 6. FINDING AN OPTIMUM SUBTREE

After deleting the edge $xy$, we have the following relations from (6.1) and (6.2):

\begin{align*}
N'(y) &= N(y) - N(x). \quad (6.9) \\
D'(y) &= D(y) - wt(x, y) \cdot N(x) - D(x). \quad (6.10)
\end{align*}

Let us now connect $y$ as a child of $x$. The new quantities, say $D''(x)$ and $e''(x)$ (we will use a similar notation for other quantities too) should be reckoned by considering $y$ also. We get the following equations:

\begin{align*}
\text{From (6.1), } N''(x) &= N'(y) + N(x). \quad (6.11) \\
\text{From (6.2), } D''(x) &= D(x) + wt(x, y) \cdot N'(y) + D'(y). \quad (6.12) \\
\text{From (6.3), } e''_1(x) \text{ and } e''_2(x) \text{ are the two largest of } \\
\{e_1(x), e_2(x), e'(y) + wt(x, y)\}. \quad (6.13) \\
\text{From (6.4), } G''(x, y) &= wt(x, y) \cdot N'(y) + Gain(y, db(y) - 1, x). \quad (6.14)
\end{align*}

By using (6.9) and (6.10) in (6.11) through (6.14), it follows that (6.5) through (6.8) are justified.

**Time complexity:**

Algorithm *FindOptSubtree* visits the vertices of the input tree $T$ two times in the DFS order (we remark that a breadth-first search implementation is also possible). The DFS process itself takes time $O(n)$ for the case of a tree. Except for the Gain function, the time for all operations done during the visit of any node can be bounded by a constant. The Gain function can be resolved if we can solve the *selection problem* [CLRS01]. It is known that in the worst-case, the selection problem can be solved in time linear in
6.4. AN APPLICATION

the number of elements in the input. Therefore, for each \( v \in V(T) \), to find the \( db(v) \) or \( db(v) - 1 \) largest gains, the time taken will be in \( O(deg_T(v)) \). Across all vertices, this time will sum up to \( O(|E(T)|) \) which is \( O(n) \). Thus the worst-case running time of Algorithm \( \text{FindOptSubtree} \) is in \( O(n) \).

To conclude, we have proved the following theorem:

**Theorem 6.3.1:**

Problem \( \text{OPT-SUBTREE} \) can be algorithmically solved in worst-case time linear in the order of the input graph.

6.4 An application

It follows that Algorithm \( \text{FindOptSubtree} \) can be easily specialized to output only \( td(v) \), for each \( v \in V(T) \), in linear time. It then follows that, for a tree \( T \), \( \mu(T) \) can be obtained in time \( O(n) \). This may be compared with the suggested algorithm (for finding \( \mu(T) \)) in [Dan93] which also runs in linear time but requiring only one \( DFS \) traversal by maintaining the distance information in a cumulative manner for an implementation. However, it can be noted that our algorithm outputs more information.

In view of the above mentioned special case of Algorithm \( \text{FindOptSubtree} \) and lemma 1.3.2, we have the following result.

**Corollary 6.4.1:**

Let \( \mathcal{F} \) denote the family of trees defined as follows:

\[
\mathcal{F} = \{ T \mid T \text{ is a tree as constructed in lemma 1.3.2} \}.
\]

The Wiener index of all the \( 2n_1n_2 \) trees in \( \mathcal{F} \) can be computed efficiently.
the required time complexity is given below – without loss of generality we assume that $n_1 \geq n_2$:

(a) Time to compute $W(T_1)$ or $W(T_2)$ is in $O(n_1)$.

(b) Time to compute all terms of the types $d^+(u,T_1)$ or $d^+(v,T_2)$ is in $O(n_1)$.

(c) Time to apply equation (1.5) or (1.6) is in $O(n_1^2)$. 
Chapter 7

\( W(G) \) for a block-cut-vertex graph

7.1 Introduction

We begin with a different version of the cut-vertex/cut-edge lemma. The lemma given below gives the formula for obtaining \( W(G) \) of a graph \( G \) that has a single cut-vertex.

Lemma 7.1.1:

Let \( G \) be a (connected) graph with a cut-vertex \( u \) such that \( G_1 \) and \( G_2 \) are two connected subgraphs of \( G \) having \( u \) as the only common vertex and \( G_1 \cup G_2 = G \). Let \( n_1 = |V(G_1)| \) and \( n_2 = |V(G_2)| \). Then

\[
W(G) = W(G_1) + W(G_2) + (n_1 - 1)d^+(v, G_2) + (n_2 - 1)d^+(u, G_1)
\]

(7.1)

where, for any vertex \( x \) of a graph \( H \), \( d^+(x, H) \) denotes the sum of the...
CHAPTER 7. W(G) FOR A BLOCK-CUT-VERTEX GRAPH

distances of the form \( d_H(x, v) \), for all \( v \in V(H) \).

Using the above lemma, the following corollary is immediate.

Corollary 7.1.1:
Let \( uv \in E(G) \) be a cut-edge in \( G \), and let \( G_1 \) and \( G_2 \) be the two components of \( G \setminus \{uv\} \) with \( n_1 = |V(G_1)| \) and \( n_2 = |V(G_2)| \). Suppose further that \( u \in V(G_1) \) and \( v \in V(G_2) \). Then

\[
W(G) = W(G_1) + W(G_2) + n_1d^+(v, G_2) + n_2d^+(u, G_1) + n_1n_2. \tag{7.2}
\]

In this chapter, we present a generalization of (7.1) by considering a graph \( G \) that has more than one cut-vertex. In particular, we obtain an expression for \( W(G) \) in terms of the Wiener indices of the blocks of \( G \) and other quantities.

### 7.2 \( G \) with more than one cut-vertex

We assume that the graph \( G \) has more than one cut-vertex. Let \( \mathcal{B} \) denote the set of all blocks and \( \mathcal{C} \) the set of all cut-vertices in \( G \). Let \( |\mathcal{B}| = k \) and \( |\mathcal{C}| = l \). From \( G \), we construct the block-cut-vertex tree \( T \) of \( G \) [Har69] as illustrated in Fig. 7.1 and 7.2. We can identify two types of vertices in \( T \):

(a) vertices that correspond to blocks in \( G \) – these appear at even levels in \( T \) (the block appearing as the root in \( T \) is labeled level 0)

(b) vertices that correspond to cut-vertices in \( G \) – these appear at odd levels in \( T \).

Two vertices in \( T \) are adjacent if one of them say, \( v_1 \) corresponds to a block in \( G \) and the other say, \( v_2 \) corresponds to a cut-vertex in \( G \) and \( v_2 \) is in \( v_1 \) in \( G \).
Let \( \mathcal{B} = \{B_1, B_2, \ldots, B_k\} \), and \( \mathcal{C} = \{v_1, v_2, \ldots, v_l\} \). Let \( B_k \) be the root of \( T \). For \( 1 \leq i < k \), let \( x_i \) be the unique vertex in \( \mathcal{C} \) such that \( x_i \in B_i \) and \( x_i \) is on the unique \( B_k-B_i \) path in \( T \). In Fig. 7.3 we list \( x_i \) and \( B_i \) for the tree \( T \) of \( G \) in Fig. 7.2. For \( 1 \leq i < k \) define

\[
B_i' = \begin{cases} 
B_k, & \text{if } i = k, \\
B_i \setminus x_i, & \text{if } 1 \leq i < k.
\end{cases}
\]

We assume \( |V(G)| = n \). Let \( |B_i| = n_i, \quad 1 \leq i \leq k, \quad |B_k'| = N_k \), so that for \( 1 \leq i < k \), \( N_i = n_i - 1 \) and \( N_k = |B_k'| = |B_k| = n_k \).

Now \( n = (n_1 + n_2 + \ldots + n_k) - (k - 1) = N_1 + N_2 + \ldots + N_k \).

Let \( P = \{(a, B)\} \) a cut-vertex belonging to block \( B \).
CHAPTER 7. W(G) FOR A BLOCK-CUT-VERTEX GRAPH

<table>
<thead>
<tr>
<th>B_i</th>
<th>B_1</th>
<th>B_2</th>
<th>B_3</th>
<th>B_4</th>
<th>B_5</th>
<th>B_6</th>
<th>B_7</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_i</td>
<td>v_1</td>
<td>v_2</td>
<td>v_3</td>
<td>v_4</td>
<td>v_5</td>
<td>v_5</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.3: B_i – x_i pairs in T of G.

If (a, B) ∈ P, then let

\[ N(a, B) = \sum_i |B'_i|, \]

where B_i is a block and in T, the B–B_i path contains a. For example, in Fig. 7.1, \( N(v_3, B_4) = \sum_{i \neq 4} |B'_i| \), and \( N(v_5, B_6) = |B'_5| + |B'_7| + |B'_8| \).

Let \( Q = \{ (a, B, b) | a \neq b \text{ and } a, b \text{ are cut-vertices both contained in the block } B \} \). Again, in Fig. 7.1 we have

\[ Q = \{ (v_1, B_3, v_3), (v_2, B_6, v_5), (v_2, B_6, v_5), (v_3, B_6, v_5), (v_4, B_7, v_5) \}. \]

For (a, B_i) ∈ P, let \( \lambda(a, B_i) \) denote the number of vertices x of G such that a is an interior vertex of every x – y path for any vertex y(\( \neq a \)) ∈ B_i.

With the above notations and meanings, we have the following expression for the Wiener index of a general graph G.

**Theorem 7.2.1:**

Let G be a simple connected undirected graph with blocks B_1, B_2, … B_k. Let \( \text{SumW}_B \) be the sum of the Wiener indexes of these blocks. Then the Wiener index W(G) of G is given by

\[ \text{SumW}_B = \sum_{(a,b) \in P} \lambda(a, B_i) d^+(a, B_i) + \sum_{(a,b) \in Q} \lambda(a, B_i) \lambda(b, B_i) d(a, b). \]
7.2. \( G \) WITH MORE THAN ONE CUT-VERTEX

**Proof:** Let \((a, B_i) \in P\) for some \(i \) where \(1 \leq i \leq k\). If \(a \neq x_i\), then the \(B_i - B_k\) path in \(T\) will not contain the cut-vertex \(a\). Then

\[
N(a, B_i) = \sum_j |B'_j| = \sum_j (n_j - 1),
\]

where the summation is over all \(j\) for which the \(B_i - B_j\) path in \(T\) contains \(a\). Note that if \(a \neq x_i\), then \(N(a, B_i)\) is the number of vertices \(x\) such that \(a\) is an interior vertex of every \(x - y\) path, for any \(y \neq a \in B_i\). If \(a = x_i\), then \(a\) lies in the \(B_i - B_k\) path in \(T\) and in this case

\[
N(a, B_i) = \sum_j |B'_j| = \sum_j N_j = 1 + \sum_j (n_j - 1), \quad \text{as } B'_i = B_k
\]

where the sum is over all \(j\) for which the \(B_i - B_j\) path in \(T\) contains \(a\). By the definition of \(\lambda(a, B_i)\), it then follows that

\[
\lambda(a, B_i) = \begin{cases} 
N(a, B_i), & \text{if } a \neq x_i \\
N(a, B_i) - 1, & \text{if } a = x_i.
\end{cases} \quad (7.3)
\]

Hence we have

\[
N(a, B_i) = \begin{cases} 
\lambda(a, B_i), & \text{if } a \neq x_i \\
\lambda(a, B_i) + 1, & \text{if } a = x_i.
\end{cases} \quad (7.4)
\]

Now consider the expression

\[
\sum_{(a, B) \in P} N(a, B)d^+(a, B'). \quad (7.5)
\]

We can split (7.5) into two terms: the first one corresponding to \(B = B_i\) and \(a \neq x_i\), for any \(i\) and the second one corresponding to \(B = B_i\) and \(a = x_i\), for some \(i\). This splitting, together with (7.4), yields

\[
\sum_{(a, B) \in P} N(a, B)d^+(a, B') = \sum_{(a, B) \in P} \lambda(a, B)d^+(a, B') + \sum_{i=1}^{k-1} d^+(x_i, B_i). \quad (7.6)
\]
It is easy to verify that whether \( a = x_i \) or not
\[
d^+(a, B'_i) = d^+(a, B_i) - d(a, x_i). \tag{7.7}
\]

From (7.6) and (7.7) we get
\[
\sum_{(a, B) \in P} N(a, B)d^+(a, B') = \sum_{(a, B) \in P} \lambda(a, B)d^+(a, B)
- \sum_{(a, B_i) \in P} \lambda(A, B_i)d(a, x_i) + \sum_{i=1}^{k-1} d^+(x_i, B_i). \tag{7.8}
\]

Next, let \((a, B, b) \in Q\) and \(B = B_i\). If \(x_i \neq a\) or \(b\) then by (7.4)
\[
N(a, B)N(b, B)d(a, b) = \lambda(a, B)\lambda(b, B)d(a, b).
\]

If one of \(a\) or \(b\) is \(x_i\), say \(b = x_i\) then
\[
N(a, B)N(b, B)d(a, b) = \lambda(a, B)\lambda(b, B)d(a, b) + \lambda(a, B_i)d(a, x_i).
\]

Therefore
\[
\sum_{(a, B, b) \in Q} N(a, B)N(b, B)d(a, b) = \sum_{(a, B, b) \in Q} \lambda(a, B)\lambda(b, B)d(a, b)
+ \sum_{(a, B_i) \in P} \lambda(a, B_i)d(a, x_i) \quad (7.9)
\]

For \(i < j\) set \(\Lambda(B'_i, B'_j) = \sum_{x \in B'_i} \sum_{y \in B'_j} d(x, y)\). It is clear that
\[
W(G) = \sum_{i=1}^{k} W(B'_i) + S, \tag{7.10}
\]

where \(S = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \Lambda(B'_i, B'_j)\). We reason that
\[
\sum_{i=1}^{k} W(B'_i) = \sum_{i=1}^{k} W(B_i) - \sum_{i=1}^{k-1} d^+(x_i, B_i) \quad (7.11)
\]

Now, in the tree \(T\), let the \(B_i - B_j\) path be
\[
B_i, a = a_0, B_{i_1}, a_1, \ldots, a_m = b, B_j
\]
7.2. \(G\) WITH MORE THAN ONE CUT-VERTEX

Then, for all \(x \in B'_i\) and \(y \in B'_j\), we have
\[
d(x, y) = d(x, a) + \sum_{s=0}^{k-1} d(a_s, a_{s+1}) + d(a_m, y).
\]
Therefore
\[
\sum_{y \in B'_j} d(x, y) = N_j d(x, a) + \sum_{s=0}^{m-1} N_j d(a_s, a_{s+1}) + d^+(a_m, B'_j)
\]
and
\[
\Lambda(B'_i, B'_j) = N_j d^+(a, B_i) + \sum_{s=0}^{m-1} N_j N_i d(a_s, a_{s+1}) + N_i d^+(a_m, B'_j).
\]
Consequently
\[
S = \sum_{i=1}^{k-1} \sum_{j=i+1}^k \Lambda(B'_i, B'_j) = \sum_{(a,b) \in P} N(a, B) d^+(a, B') + \sum_{(a,B,b) \in Q} N(a, B) N(b, B) d(a, b).
\]
Hence, from (7.10) we get \(W(G)\) as
\[
\sum_{i=1}^k W(B'_i) + \sum_{(a,B) \in P} N(a, b) d^+(a, B') + \sum_{(a,B,b) \in Q} N(a, B) N(b, B) d(a, b)
\]
(7.12)

We now use equations (7.8), (7.9), (7.11) in (7.12). This gives the desired expression for \(W(G)\) as stated in theorem 7.2.1.

**Remark 7.2.1:** We apply theorem 7.2.1 for the case considered in corollary 7.1.1. Let \(G\) be the graph in Fig.7.4 where \(B_1\) and \(B_2\) are blocks with \(n_1\) and \(n_2\) vertices respectively and \(uv\) is a cut-edge. The block-cut-vertex tree of \(G\) is shown in Fig. 7.5. We note that \(\lambda(u, B_1) = n_2, \lambda(v, B_2) = n_1, \lambda(u, B_3) = n_1 - 1, \lambda(v, B_3) = n_2 - 1\) and \(\lambda(u, B_3) \lambda(v, B_3) = (n_1 - 1)(n_2 - 1)\).
84 CHAPTER 7. $W(G)$ FOR A BLOCK-CUT-VERTEX GRAPH

Figure 7.4: A graph $G$ with a cut-edge $uv$.

Figure 7.5: Block-cut-vertex tree of $G$ in Fig.7.4.

By theorem 7.2.1, we have

$$W(G) = W(B_1) + W(B_2) + W(B_3) + n_2d^+(u, B_1) + (n_1 - 1)$$
$$+ (n_2 - 1) + n_1d^+(v, B_2) + (n_1 - 1)(n_2 - 1)$$
$$= W(B_1) + W(B_2) + n_2d^+(u, B_1) + n_1d^+(v, B_2) + n_1n_2.$$  

This agrees with (7.2) in corollary 7.1.1.
Chapter 8

Future directions

8.1 Introduction

In this concluding chapter we begin with the following observations:

- The study of mathematical and chemical significance of Wiener index and its limitations is still an ongoing area of research.

- It is possible to give definitions that modify Wiener index yielding new topological indices which may have relevance to Chemistry or Graph theory.

- The literature describes other interesting topological descriptors which are graphical invariants.

- The inverse problem for topological indices is an interesting problem (i.e., given an integer $n$ find a graph whose prescribed index has the value $n$).
Based on the above, we mention three important directions for further research which arise from the notion of Wiener index of graphs. For a few conjectures and open problems, [DEG01] may also be consulted.

8.2 Wiener index and Laplacian eigenvalues

Let $G$ be a graph whose vertices are arbitrarily labeled $v_1, \ldots, v_n$ and let $D(G)$ be the diagonal matrix whose diagonal element at row $i$ and column $i$ is $\deg_G(v_i)$. Let $A(G)$ be the adjacency matrix of $G$. Then, the Laplacian matrix $L(G)$ of $G$ is defined as

$$L(G) = D(G) - A(G).$$

The eigenvalues of $L(G)$, say $\lambda_1, \lambda_2, \ldots, \lambda_n \geq 0$ form what is known as the Laplacian spectrum of the graph $G$. When $G$ is a tree $T$, it is known that

$$W(T) = n \sum_{j=1}^{n-1} \frac{1}{\lambda_j}. \quad (8.1)$$

For a proof of the above equation and for other relevant results related to the Laplacian spectrum, see [New00].

For a general graph $G$, $W(G)$ is not equal to the right-hand-side of (8.1). For any $G$, let $\rho(u, v)$ be the resistance distance defined thus: construct an electrical network by replacing all edges by a 1 $\Omega$ resistance and measure the equivalent resistance between the points or vertices $u$ and $v$ – the value of this equivalent resistance is defined to be $\rho(u, v)$. Let $W'(G)$ be the sum of all values $\rho(u, v)$, where the summation extends over all distinct pairs $(u, v)$. It turns out (see [Mer01]) that $W'(G)$ is equal to the
right-hand-side of (8.1).

For example, consider the graph $G_1$ with

$$V(G_1) = \{v_1, v_2, v_3, v_4\} \text{ and } E(G_1) = \{v_1v_2, v_1v_3, v_2v_3, v_2v_4, v_3v_4\}.$$  

The eigenvalues of $L(G_1)$ are 4, 4, 2, 0 and the various resistance distances are as follows: $\rho(v_1, v_2) = 5/8$, $\rho(v_1, v_3) = 5/8$, $\rho(v_1, v_4) = 1/2$, $\rho(v_2, v_4) = 5/8$, $\rho(v_3, v_4) = 5/8$. It follows that $W'(G_1) = 4$ whereas $W(G_1) = 7$. For the graph $K_4$, the eigenvalues of $L(K_4)$ are 4, 4, 4, 0 and all the six resistance distances have the value $1/2$. It follows that $W'(K_4) = 3$ while $W(K_4) = 6$. Thus $W'(G)$ has been suggested as a generalization of Wiener index (from trees to non-acyclic graphs). It is therefore interesting to investigate the mathematical properties of $W'(G)$ and the algorithms for its efficient computation.

### 8.3 Other graph theoretical descriptors

In [KG97], the Wiener number of vertex-weighted (or simply weighted) graphs is considered. A weighted graph is a graph $G$ together with a weight function $w : V(G) \to N^+$ (or positive reals). Then, the weighted Weiner number $W(G, w)$ of the weighted graph $(G, w)$ is defined as

$$W(G, w) = \frac{1}{2} \sum_{u,v \in V(G)} w(u)w(v)d_G(u,v).$$  

It follows that if $w$ is a constant, say $m$, then $W(G, w) = m^2W(G)$. In [Gut94], $W(G, w)$ is studied taking $w(u) = \text{deg}_G(u)$, for all $u$. Other modifications to Wiener index (e.g., [DG98]) are also described in the literature.

A number of other molecular descriptors are important in theoretical or
mathematical or combinatorial chemistry. Similar to Wiener index, many of these descriptors are useful in rational drug design (e.g., see [Wie96]). One example descriptor is the $Z$-index $Z(G)$ or the Hosoya index which is defined to be the number of independent edge subsets of $G$. Another descriptor is the $\sigma$-index $\sigma(G)$ or the Merrifield-Simmons index which is defined to be the total number of independent (vertex) sets of $G$. For example, in [PV06] it is shown that every unicyclic graph $G$ on $n$ vertices satisfies, $L_n \leq \sigma(G) \leq 3.2^{n-3} + 1$, where $L_n = F_{n-1} + F_{n+1}$, $F_n$ being the $n^{th}$ Fibonacci number. It should be of interest to study the properties of some of these descriptors from graph theoretical and computational viewpoints for the case of molecular graphs. In this direction, a recent work can be found in [Wag07].

8.4 Inverse problem on Wiener index

Another research area originates from the Wiener index conjecture. Let

\[
\text{FORBIDDEN} = \{ 2, 3, 5, 6, 7, 8, 11, 12, 13, 14, 15, 17, 29, 21, 22, 23, 24, 26, 27, 30, 33, 34, 37, 38, 39, 41, 43, 45, 47, 51, 53, 55, 60, 61, 69, 73, 77, 78, 83, 85, 87, 89, 91, 99, 101, 106, 113, 147, 159 \}. 
\]

The conjecture was, except for the integers in the set FORBIDDEN, every positive integer is the Wiener index of some tree. The conjecture was proved to be true in [Wag06]. For other related results, see also [BBM04]. A related problem is the inverse problem mentioned above (see [LLW03]). Another related problem is to find graphs of minimal or maximal index value, given
8.4. INVERSE PROBLEM ON WIENER INDEX

the order or size of the graph (see [TW05]). We point out that the following
generic decision problem should be challenging:

Problem 8.1: Given an integer \( n \), determine if there exists a tree \( T \) satisfying a specified property \( P \) such that \( W(T) = n \).

The property \( P \), for example, can be given as a bound on the maximum
degree or a bound on the diameter of \( T \). A problem related to problem 8.1
is to find an efficient algorithm to construct a tree \( T \), if it exists.
Bibliography


BIBLIOGRAPHY


[Gut94]  I. Gutman. Selected properties of the Schultz molecular topo-


[GYLC94]  I. Gutman, Y.N. Yeh, S.N. Lee, and J.C. Chen.  Wiener num-


[Hen86]  G.R.T. Hendry.  Existence of graphs with prescribed mean dis-

[Hwa93]  K. Hwang.  *Advanced computer architecture: Parallelism, scalabil-


BIBLIOGRAPHY


Buy your books fast and straightforward online - at one of world’s fastest growing online book stores! Environmentally sound due to Print-on-Demand technologies.

Buy your books online at

www.get-morebooks.com

Kaufen Sie Ihre Bücher schnell und unkompliziert online – auf einer der am schnellsten wachsenden Buchhandelsplattformen weltweit! Dank Print-On-Demand umwelt- und ressourcenschonend produziert.

Bücher schneller online kaufen

www.morebooks.de