



HAL
open science

Faster Treewidth-based Approximations for Wiener Index

Giovanna K Conrado, Amir K Goharshady, Pavel Hudec, Pingjiang Li, Harshit J Motwani

► **To cite this version:**

Giovanna K Conrado, Amir K Goharshady, Pavel Hudec, Pingjiang Li, Harshit J Motwani. Faster Treewidth-based Approximations for Wiener Index. 2024. <hal-04327333v2>

HAL Id: hal-04327333

<https://hal.science/hal-04327333v2>

Preprint submitted on 12 Feb 2024 (v2), last revised 27 Apr 2024 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization

Faster Treewidth-based Approximations for Wiener Index

Giovanna Kobus Conrado ✉ 

Amir Kafshdar Goharshady ✉ 

Pavel Hudec ✉ 

Pingjiang Li ✉

Harshit Jitendra Motwani ✉ 

Department of Computer Science and Engineering

Department of Mathematics

Hong Kong University of Science and Technology (HKUST)

Clear Water Bay, Hong Kong, China

Abstract

The Wiener index of a graph G is the sum of distances between all pairs of its vertices. It is a widely used graph property in chemistry, initially introduced to examine the link between boiling points and structural properties of alkanes, which later found notable applications in drug design. Thus, computing or approximating the Wiener index of molecular graphs, i.e. graphs in which every vertex models an atom of a molecule and every edge models a bond, is of significant interest to the computational chemistry community.

In this work, we build upon the observation that molecular graphs are sparse and tree-like and focus on developing efficient algorithms parameterized by treewidth to approximate the Wiener index. We present a new randomized approximation algorithm using a combination of tree decompositions and centroid decompositions. Our algorithm approximates the Wiener index within any desired multiplicative factor $(1 + \epsilon)$ in time $O(n \cdot \log n \cdot k^3 + \sqrt{n} \cdot k/\epsilon^2)$, where n is the number of vertices of the graph and k is the treewidth. This time bound is almost-linear in n .

Finally, we provide experimental results over standard benchmark molecules from PubChem and the Protein Data Bank, showcasing the applicability and scalability of our approach on real-world chemical graphs and comparing it with previous methods.

2012 ACM Subject Classification Applied computing → Chemistry; Theory of computation → Graph algorithms analysis; Theory of computation → Data structures design and analysis


Keywords and phrases Computational Chemistry, Treewidth, Wiener Index

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

MOTIVATION. The Wiener index of a graph G is the sum of the distances between all pairs of vertices in G . Besides being a natural problem to compute, it is also a well-studied graph invariant with applications in computational chemistry and biology. Indeed, it is one of computational chemistry's oldest and most important topological indices [37].


HISTORY. In chemistry, the Wiener index was first considered by Harry Wiener in [40]. It was initially studied to establish connections between alkanes' boiling points and the underlying graphs' structural properties. This study later motivated the development of other topological indices in computational chemistry. Further development of QSAR (Quantitative Structure-Activity Relationship) and QSPR (Quantitative Structure-Property Relationship) models led to the discovery of positive correlations of even more chemical and physical properties to the Wiener index [28, 37, 38, 42]. Due to its simplicity and usefulness, the Wiener index was also studied by computer scientists and mathematicians [16, 35]. The use

 © Giovanna Kobus Conrado, Amir Kafshdar Goharshady, Pavel Hudec, Pingjiang Li, and Harshit Jitendra Motwani;

licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:16

 Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of neural networks in chemical graph theory has led to a renewed interest in topological indices and their application in molecular mining, toxicity detection, and computer-aided drug discovery. Several studies have been conducted on this topic, such as [5, 15, 17, 24, 41]. Given the significance of the Wiener index for chemists and the abundance of large molecules, it is imperative to develop faster algorithms for computing it. Indeed, there are many previous works in this direction [7, 9, 14, 21, 29].

PARAMETERIZED ALGORITHMS. Parameterized algorithms aim to tackle computationally intractable problems and identify subsets of instances that can be solved efficiently [11]. In parameterized complexity, we consider an additional parameter k along with the input size n for measuring the runtime. This is in contrast to classical complexity theory, which only considers the input size of the problem. Many parameterized algorithms focus on NP-hard problems and provide runtime bounds that depend polynomially on the size of the problem but have non-polynomial dependence on the parameter k . If we know that k is small in real-world instances, this leads to solutions that are effectively polynomial-time, i.e. they take polynomial time on all the real-world instances where this parameter is small.

FIXED-PARAMETER TRACTABLE (FPT). Given an input of size n and a parameter k , an algorithm with a running time of $O(f(k) \cdot n^c)$, for some constant c and computable function f , is called *Fixed-Parameter Tractable (FPT)* [11]. The intuition is the same as above. If the parameter k is small in all real-world instances of the problem, then the algorithm would in practice have a polynomial runtime. Crucially, the degree c of this polynomial does not depend on either k or n .

TREewidth. Treewidth is one of the most important structural parameters of graphs and has been extensively studied in combinatorics and graph theory. Intuitively speaking, it measures the tree-likeness of a graph [4]. Trees and forests have a treewidth of 1 and cliques on n vertices have treewidth $n - 1$. The main advantage of treewidth in algorithm design arises when we are designing parameterized algorithms for NP-hard problems by considering it as the parameter of the problem. Many families of commonly-studied graphs, such as trees, cacti, series-parallel graphs, outer-planar graphs, and control-flow graphs of structured programs, have bounded treewidth [2, 4, 11]. This allows efficient dynamic programming techniques using the tree decomposition of the graph [2]. See Section 2 for a formal definition.

TREewidth OF MOLECULES. Extending this idea, computational chemists and biologists have also explored the treewidth of various important classes of molecules [43, 45]. In our experimental results (Section 4), we observe that a significant majority of molecules in the PubChem repository [19] have a treewidth of at most 10. Even large proteins from the Protein Data Bank [32] are observed to have a treewidth of at most 5. Since a significant fraction of molecules have bounded treewidth, exploring and designing treewidth-based parameterized algorithms for computational problems in chemistry and biology is a natural step. In fact, the same has been done in several works in the literature [1, 7, 10, 39, 44]. We extend this line of research by presenting significantly faster treewidth-based approaches for approximating the Wiener index.

OUR CONTRIBUTION. In this paper, we introduce a novel randomized algorithm that approximates the Wiener index of a graph using its tree decomposition. The unique aspect of our algorithm is the incorporation of tree and centroid decompositions. This idea significantly enhances efficiency in answering distance queries within the graph. This is then plugged directly into an established randomized algorithm to approximate the Wiener index, obtaining the same approximation guarantees by an asymptotically faster method. Both theoretical analysis and experimental results demonstrate that our algorithm

■ **Table 1** Comparison of Different Algorithms for Computing the Wiener Index. Here, n denotes the number of vertices, k denotes the treewidth, and ϵ represents the error of approximation.

Algorithm	Time Complexity	Type	Ref.
Floyd-Warshall	$O(n^3)$	Exact	[18]
Orthogonal Range Searching	$O(n \cdot \log^{k-1} n)$	Exact Parameterized	[7]
Treewidth-based Dynamic Programming	$O(n^2 \cdot k^2)$	Exact Parameterized	[8]
BFS	$O(n^2 \cdot k)$	Exact Parameterized	[30, 46]
Classical Approximation	$O(n^{5/2}/\epsilon^2)$	Randomized Approximation	[21]
Our Algorithm	$O(n \cdot \log n \cdot k^3 + \sqrt{n} \cdot k/\epsilon^2)$	Parameterized Randomized Approximation	Sec. 3

92 outperforms current methods in calculating the Wiener index for molecular graphs, which
93 are commonly encountered in computational chemistry and biology.

94 COMPARISON WITH PREVIOUS RESULTS. Table 1 compares the runtime complexity of our
95 algorithm with previous methods. Here, n is the number of vertices in the graph, k is the
96 treewidth, and ϵ is the error in the approximation, i.e. we are reporting the runtime for a
97 $(1 + \epsilon)$ -approximation of the Wiener index. We refer to Section 4 for a detailed experimental
98 evaluation of our algorithm on datasets from PubChem [19] and the Protein Data Bank [32].

99 The most classical approach to compute the Wiener index is simply performing an all-pairs
100 shortest path computation using Floyd-Warshall and then summing up the distances. This
101 will lead to a time complexity of $O(n^3)$. In [7], the authors provided the first parameterized
102 algorithm for the Wiener index based on treewidth. Their algorithm is a divide-and-conquer
103 method based on orthogonal range searching and repeatedly finds small cuts using the tree
104 decomposition. They achieve a runtime bound of $O(n \cdot \log^{k-1} n)$. Note that this is not
105 FPT. In [8], an FPT algorithm was provided based on dynamic programming on the tree
106 decomposition. This algorithm has a quadratic dependence on n . For unweighted graphs,
107 given that a graph with treewidth k has $O(n \cdot k)$ vertices, running a BFS from each vertex
108 would lead to a total runtime of $O(n^2 \cdot k)$. Finally, [21] provides an algorithm on general
109 graphs, not using any parameters, that approximates the average pairwise distance within a
110 factor of $(1 + \epsilon)$ with a probability of at least $2/3$ by taking a random sample of the distances
111 between pairs of vertices. Note that the Wiener index is n^2 times the average distance. Thus,
112 this algorithm is directly applicable to our setting, as well. Our algorithm builds upon the
113 classical approximation of [21] and uses a tree decomposition and a centroid decomposition
114 to speed up the sampling.

115 2 Preliminaries

116 In this section, we introduce the Wiener index and define some basic concepts of parameterized
117 complexity. We refer to [11] for more details. This is followed by a short presentation of the
118 classical approximation algorithm of [21], which forms the basis of our approach.

WIENER INDEX [40]. The Wiener Index of an undirected graph $G = (V, E)$ is defined as the
all-pairs sum of distances among vertices of the graph. Formally,

$$W(G) := \sum_{u,v \in V} d(u,v).$$

119 Additionally, we define the average distance between pairs of vertices in G as $\bar{d}(G) :=$
 120 $W(G)/n^2$.

121 ► **Remark 1.** In this work we assume that our graphs are connected, unweighted, and
 122 undirected. In the context of molecular graphs, all types of covalent bonds—be they single,
 123 double, or triple—are represented as a single undirected edge in the corresponding graph.
 124 For a disconnected graph, the Wiener index is simply $+\infty$. However, in some applications,
 125 the Wiener index of a disconnected graph is defined as the sum of the Wiener indices of its
 126 connected components. In such cases, each connected component can be processed separately.

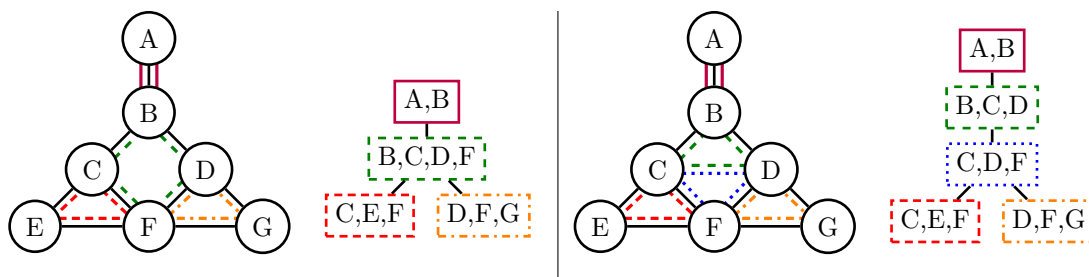
127 **TREE DECOMPOSITION (TD)** [23, 33, 34]. A *tree decomposition* of a given graph $G = (V, E_G)$
 128 is a tree $T = (\mathcal{B}, E_T)$ satisfying the following conditions:

- 129 ■ Every node $b \in \mathcal{B}$ of T , which is called a *bag*, contains a subset of vertices $V_b \subseteq V$.
- 130 ■ The bags cover the entire vertex set V of G , i.e. $\bigcup_{b \in \mathcal{B}} V_b = V$. In other words, every vertex
 131 appears in at least one bag.
- 132 ■ For every edge in the original graph G , there exists a bag that contains both endpoints
 133 of the edge. More formally, for every $e = \{u, v\} \in E_G$, there is a bag $b \in \mathcal{B}$, s.t. $u, v \in V_b$.
- 134 ■ Every vertex $v \in V$ appears in a connected subtree of T , meaning that the set $\mathcal{B}_v = \{b \in$
 135 $\mathcal{B} \mid v \in V_b\}$ forms a connected subgraph of T .

136 ► **Remark 2.** An equivalent statement of the last condition above is that for every three bags
 137 $b_1, b_2, b_3 \in \mathcal{B}$, if b_3 is on the unique path from b_1 to b_2 in T , then $V_{b_1} \cap V_{b_2} \subseteq V_{b_3}$.

138 **TREewidth** [33]. The *width* of a tree decomposition T is defined as $w(T) := \max_{b \in \mathcal{B}} |V_b| - 1$,
 139 i.e. the size of the largest bag minus one. Furthermore, the *treewidth* of the graph G , denoted
 140 as $\text{tw}(G)$, is defined as the minimum width amongst all possible tree decompositions of G .

141 Intuitively speaking, treewidth measures the structural likeness of a graph to a tree.
 142 Specifically, the smaller the treewidth of a graph, the more tree-like it appears, in the sense
 143 that a graph of treewidth k can be decomposed into small parts (bags), each of size at most
 144 $k + 1$, which are connected to each other in a tree-like manner T . Figure 1 showcases an
 145 illustration containing two distinct tree decompositions of a graph G , each having a different
 146 width. Since only forests have treewidth of 1, the tree decomposition on the right is optimal,
 147 and $\text{tw}(G) = 2$.



■ **Figure 1** A Graph G and Two Tree Decompositions of G of Width 3 (left) and 2 (right).

148 Treewidth is a parameter indicating graph sparsity, providing an upper bound on the
 149 number of edges. Specifically, in a graph with n vertices and treewidth k , the number of edges
 150 is $O(n \cdot k)$. More precisely, the number of edges is less than or equal to $n \cdot k - k \cdot (k + 1/2)$ [31].
 151 Additionally, we have the following ubiquitous lemma:

152 ► **Lemma 3** (Cut Lemma [11]). *Let $T = (\mathcal{B}, E_T)$ be a tree decomposition of $G = (V, E_G)$.
 153 Consider two vertices $u, v \in V$ and two arbitrary bags $b_u, b_v \in \mathcal{B}$ such that $u \in b_u$ and $v \in b_v$. If*

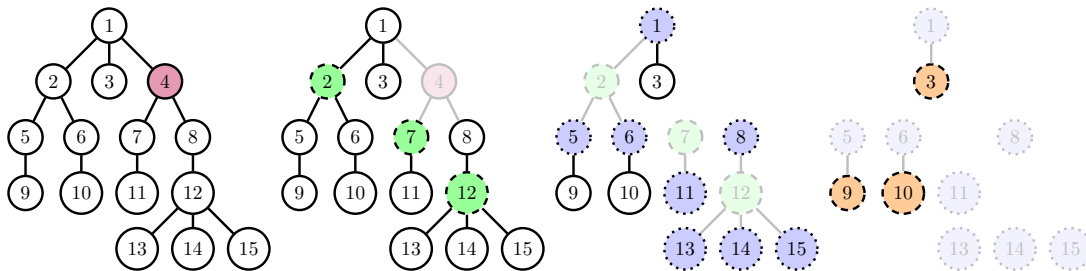
154 $b \in \mathcal{B}$ is a bag on the unique path from b_u to b_v in T , then any path from u to v in G will
 155 intersect V_b . Additionally, if $e = \{b_1, b_2\} \in E_T$ is an edge on the unique path from b_u to b_v in
 156 T , then any path from u to v in G will intersect $V_{b_1} \cap V_{b_2}$.

157 COMPUTING TREE DECOMPOSITIONS. It is well-known that the treewidth of a graph, as
 158 well as an optimal tree decomposition with $O(n)$ bags, can be computed by a linear-time
 159 FPT algorithm (parameterized by the treewidth itself) [3]. Additionally, there are many
 160 well-optimized tools for this task. Thus, in the sequel, we assume without loss of generality
 161 that an optimal tree decomposition of our graph is given as a part of the input.

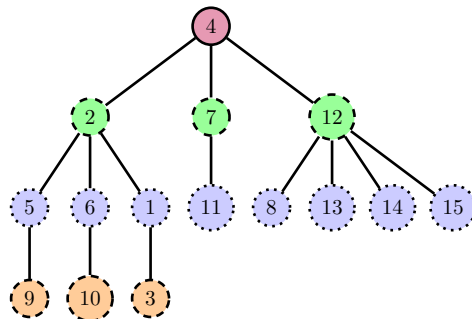
162 CENTROID [25]. Consider a tree $T = (V_T, E_T)$ with n vertices. We define a *centroid node* of
 163 T as a node whose removal breaks the tree down into several subtrees such that no resulting
 164 subtree has a size greater than $n/2$. In other words, a centroid is a $1/2$ separator of T . It is
 165 well-known that every tree has at least one centroid node, which can be obtained in linear
 166 time by dynamic programming.

167 CENTROID DECOMPOSITION (CD) [6, 12]. A *centroid decomposition* of T is another tree T'
 168 on the same set of vertices as T , recursively defined as follows:

- 169 ■ When $|V_T| = 1$, we simply have $T' = T$.
- 170 ■ For a more complex tree, we first identify a centroid node r of T , then position this node
 171 as the root of T' .
- 172 ■ Once we have selected a centroid node r and removed it from T , we end up separating
 173 the original tree into several connected subtrees. Let us denote these as T_1, T_2, \dots, T_m .
 174 For each subtree T_i , we find a centroid decomposition T'_i with a root r_i . We make each
 175 r_i a child of r .



■ **Figure 2** A Graph G and the Steps of Building its Centroid Decomposition. Each step highlights the centroid vertex of each of the current components of the graph.



■ **Figure 3** The Resulting Centroid Decomposition of G .

176 Figure 2 shows the steps of computing a centroid decomposition. Each color corresponds
 177 to a distinct layer of the centroid decomposition, with the node representing the centroid
 178 of the similarly colored dotted subtree. In this illustration, the node 4 is identified as the
 179 centroid of the initial tree. Following the removal of node 4, nodes 2, 7, and 12 are selected
 180 as the centroids of each resulting subtree. Subsequent centroids are determined in a recursive
 181 manner. The final centroid decomposition is shown in Figure 3.

182 PROPERTIES OF CDS. The height of a CD is bounded by $O(\log n)$, where n is the number
 183 of vertices in the original tree. This is because, with every new layer added to the centroid
 184 decomposition, each connected component splits into several parts, each no larger than $1/2$
 185 the size of the original component. Consequently, we can append at most $O(\log n)$ layers to
 186 the centroid decomposition. Additionally, CDs satisfy the following useful lemma:

187 ► **Lemma 4** (Proof in Appendix A). *Let $u, v \in V_T$ be two vertices of the original tree T and l be
 188 their lowest common ancestor in the centroid decomposition T' . The unique path connecting
 189 u and v in T must visit l .*

190 COMPUTING CENTROID DECOMPOSITIONS. Given a tree T with n vertices, there are a
 191 variety of algorithms in the literature that compute a centroid decomposition T' of T in
 192 $O(n)$. Examples include [6, 12].

193 LOWEST COMMON ANCESTOR QUERIES. Consider a rooted tree T with n vertices. Suppose
 194 we have q offline queries, each providing two vertices $u, v \in T$ and asking for their lowest
 195 common ancestor. The classical algorithm of Gabow and Tarjan [20] solves this problem and
 196 answers all queries in $O(n + q)$.

197 APPROXIMATION ALGORITHM OF [21]. The work [21] provides an elegant and simple
 198 approximation algorithm for the average distance $\bar{d}(G)$ between pairs of vertices. Since the
 199 Wiener index is simply $n^2 \cdot \bar{d}(G)$, the same algorithm can be reused for our problem. Given
 200 a graph G and an error bound ϵ as the input, the algorithm in [21] works as follows:

- 201 1. Uniformly select $\Theta(\sqrt{n}/\epsilon^2)$ pairs of vertices.
- 202 2. Find the distance between each selected pair of vertices.
- 203 3. Output the average of the computed distances.

204 Surprisingly, this algorithm provides a $(1 + \epsilon)$ -approximation of $\bar{d}(G)$ with probability $2/3$.

205 ► **Theorem 5** ([21], Theorem 5.1). *Given G and ϵ as input, the algorithm above outputs a
 206 $(1 + \epsilon)$ -approximation of $\bar{d}(G)$ with probability at least $2/3$.*

207 As a direct corollary, a $(1 + \epsilon)$ -approximation of the Wiener index can be computed in the
 208 same time complexity by simply multiplying the result of this algorithm by n^2 .

209 COMPLEXITY ANALYSIS. For general graphs, each distance query can take $O(n^2)$ time.
 210 Thus, the total runtime of the algorithm above is $O(n^{5/2}/\epsilon^2)$. However, if the underlying
 211 graph G is guaranteed to have small treewidth k , then it can have at most $O(n \cdot k)$ vertices.
 212 Thus, each distance query can be answered in $O(n \cdot k)$ by a BFS. This reduces the runtime
 213 to $O(n^{3/2} \cdot k/\epsilon^2)$.

214 In this work, we build upon this simple randomized algorithm and use the treewidth
 215 to obtain a faster algorithm for distance queries. This allows us to reduce the runtime
 216 dependence on n to almost-linear.

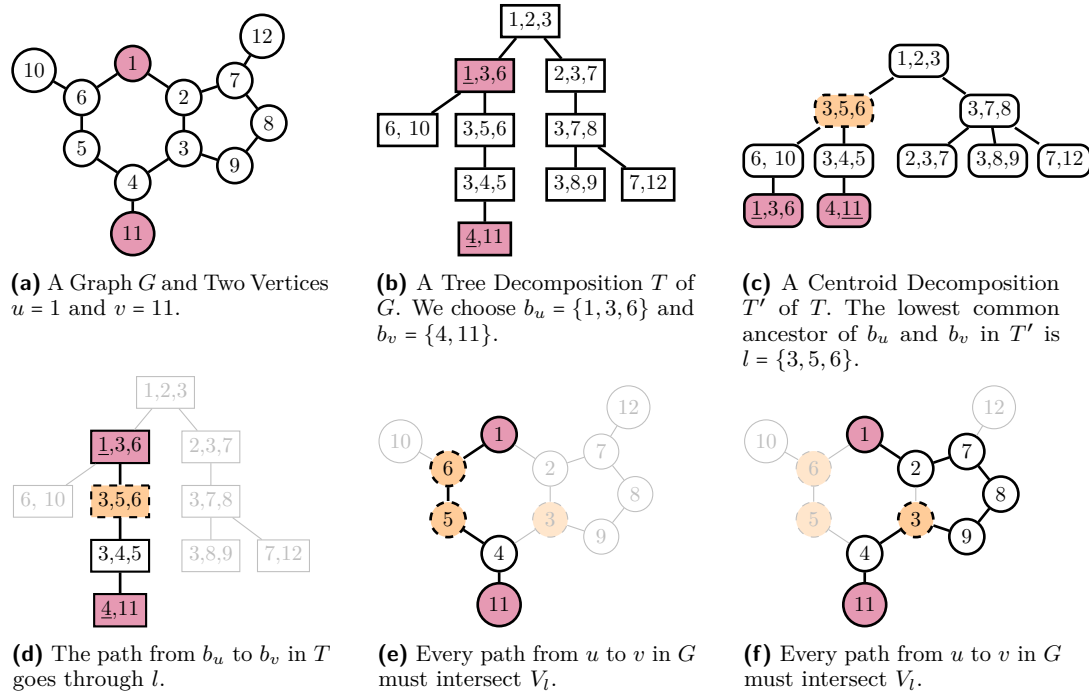
217 **3 Our Algorithm**

218 In this section, we present our treewidth-based algorithm. Our algorithm follows the same
 219 steps as the approximation algorithm of [21], except that we exploit the tree decomposition

220 to perform distance queries faster. Our main novel idea is to look not only at a tree
 221 decomposition of the underlying graph but also at a centroid decomposition of this tree
 222 decomposition. Thus, our algorithm exploits the desirable properties of both types of
 223 decomposition, as formalized by the lemma below:

224 ► **Lemma 6.** *Let $G = (V, E_G)$ be a graph, $T = (\mathcal{B}, E_T)$ a tree decomposition of G and
 225 $T' = (\mathcal{B}', E_{T'})$ a centroid decomposition of T . Consider two vertices $u, v \in V$ and arbitrary
 226 bags $b_u, b_v \in \mathcal{B}$ such that $u \in b_u$ and $v \in b_v$. Let l be the lowest common ancestor of b_u and b_v
 227 in the centroid decomposition T' . Any path that goes from u to v in G intersects V_l .*

228 **Proof.** Consider the path π_T from b_u to b_v in the tree decomposition T . By Lemma 4, we
 229 have $l \in \pi_T$. By Lemma 3, any bag in π_T intersects every path from u to v in G . This is
 230 illustrated in Figure 4. ◀



■ **Figure 4** An Illustration of Lemma 6.

Based on the lemma above, if we precompute the distances from each vertex appearing in a bag l of the centroid decomposition T' to the vertices appearing in descendants of l in T' , then we can answer distance queries in $O(k)$. In other words, to find the distance from u to v , we first find two bags b_u and b_v containing them, then compute $l = \text{lca}(b_u, b_v)$. Now, we know that every path from u to v has to go through l , thus

$$d_G(u, v) = \min_{w \in V_l} (d_G(u, w) + d_G(w, v)).$$

231 Here, d_G denotes the distance in graph G .

232 OUR ALGORITHM FOR WIENER INDEX. Based on the discussion above, given $\epsilon > 0$, a graph
 233 $G = (V, E_G)$ and a tree decomposition $T = (\mathcal{B}, E_T)$ of G with width k , our algorithm turns
 234 G into a weighted graph and takes the following steps:

- 235 ■ **Step 1 (Centroid Decomposition).** Compute a centroid decomposition T' of the tree
 236 decomposition T .
- 237 ■ **Step 2 (Local Precomputation).** For every two vertices $u, v \in V$, if there is a bag $b \in \mathcal{B}$
 238 that contains both of them, i.e. $u, v \in V_b$, then compute the distance $d_G(u, v)$ and add a
 239 direct edge with weight $d_G(u, v)$ between u and v .
- 240 ■ **Step 3 (Ancestor-Descendant Precomputation).** Let $b_1, b_2 \in \mathcal{B}$ be two bags such
 241 that b_1 is an ancestor of b_2 in the centroid decomposition T' . For every $u \in V_{b_1}$ and $v \in V_{b_2}$,
 242 compute the distance $d_G(u, v)$ and add a direct edge with weight $d_G(u, v)$ between u and
 243 v .
- 244 ■ **Step 4 (Sampling).** Uniformly select $\Theta(\sqrt{n}/\epsilon^2)$ pairs of vertices of G as in the algorithm
 245 of [21].
- 246 ■ **Step 5 (Distance Queries).** For each pair of vertices $(u, v) \in V^2$ selected in the
 247 previous step, compute $d_G(u, v)$.
- 248 ■ **Step 6 (Output).** Output the average of all the distances obtained in the previous step.
- 249 For Step 1, we can rely on previous algorithms that compute centroid decompositions,
 250 such as [6, 12]. Steps 4 and 6 are straightforward. We now provide details of Steps 2, 3, and
 251 5, followed by correctness proofs and runtime analyses.

252 **DETAILS OF STEP 2.** This step is inspired by [8]. Given the graph $G = (V, E_G)$ and its tree
 253 decomposition $T = (\mathcal{B}, E_T)$, our goal is to create shortcut edges between any pair of vertices
 254 that appear in the same bag. We provide a recursive procedure as follows:

- 255 i. Choose a leaf bag ℓ of the tree decomposition T .
- 256 ii. Perform an all-pairs shortest-path algorithm, such as Floyd-Warshall, in $G[V_\ell]$, i.e. only
 257 on the vertices and edges in ℓ . If a path of length d is found between u and v , add a
 258 direct $\{u, v\}$ edge with weight d to G .
- 259 iii. Let $T^* = T - \ell$ and $G^* = G - \{v \in V_\ell \mid \nexists b \in \mathcal{B} \ b \neq \ell \wedge v \in V_b\}$. In other words, we are
 260 removing the leaf bag ℓ from our tree decomposition and also removing any vertex that
 261 appeared only in this bag from the graph G .
- 262 iv. Run the algorithm recursively on (G^*, T^*) . This causes more shortcut edges to be added
 263 in G .
- 264 v. Repeat Step ii, i.e. perform another all-pairs shortest-path in $G[V_\ell]$ and add the resulting
 265 shortcut edges to G .

266 Figure 5 provides an example of this step.

267 ► **Lemma 7 (Proof in Appendix B).** *The procedure above runs in time $O(n \cdot k^3)$. After its
 268 execution, T is still a valid tree decomposition of G , and for every pair of vertices $u, v \in V$, if
 269 there exists a bag $b \in \mathcal{B}$ containing both of them, then there is a direct (shortcut) edge from u
 270 to v with weight $d_G(u, v)$.*

271 ► **Remark 8.** Throughout our algorithm, we always keep at most one edge, i.e. the edge with
 272 minimum weight, between every pair $\{u, v\}$ of vertices.

DETAILS OF STEP 3. In this step, we process our centroid decomposition T' in a bottom-up
 manner. For every bag $b \in \mathcal{B}$, we consider the subtree T'_b of the centroid decomposition T' ,
 consisting of b and all of its descendants in T' . Let G_b be the induced subgraph of G that
 contains all the vertices in T'_b , i.e.

$$G_b = G \left[\bigcup_{b' \in T'_b} V_{b'} \right].$$

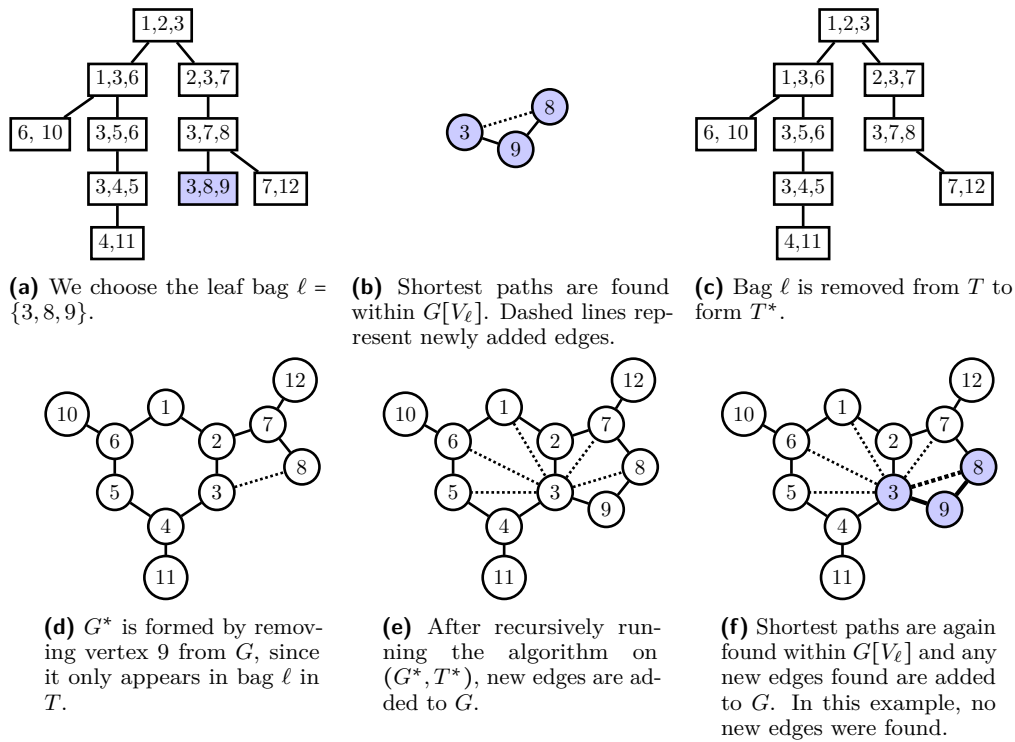


Figure 5 An Example of Step 2 on the Graph and Decomposition of Figure 4.

273 For every vertex $v \in V_b$ that appears in the bag b , our algorithm runs a shortest-path
 274 computation, such as Dijkstra’s algorithm [13], from b in the graph G_b and finds its distances
 275 to all other vertices of G_b , adding the corresponding shortcut edges. See Figure 6 for an
 276 example.

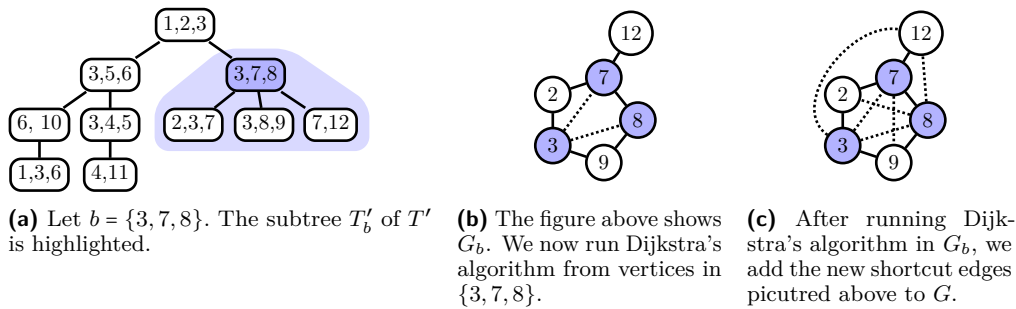


Figure 6 An Example of Step 3 on the Graph and Decompositions of Figure 4.

277 ► **Lemma 9.** *The procedure above runs in $O(n \cdot \log n \cdot k^3)$. After its execution, for every two*
 278 *bags $b_1, b_2 \in \mathcal{B}$ such that b_1 is an ancestor of b_2 in the centroid decomposition T' and every*
 279 *two vertices $u \in V_{b_1}$ and $v \in V_{b_2}$, we have a shortcut edge from u to v with weight $d_G(u, v)$.*

Proof. Let α_b and δ_b be the number of ancestors and descendants of b in T' , respectively. The graph G_b has $O(\delta_b \cdot k)$ vertices and thus $O(\delta_b \cdot k^2)$ edges. Moreover, we perform $O(k)$

23:10 Faster Treewidth-based Approximations for Wiener Index

Dijkstras* over this graph, one for each vertex in the bag b . Thus, our total runtime is

$$\sum_{b \in \mathcal{B}} O(\delta_b \cdot k^3) = \sum_{b \in \mathcal{B}} O(\alpha_b \cdot k^3) = O(n \cdot \log n \cdot k^3).$$

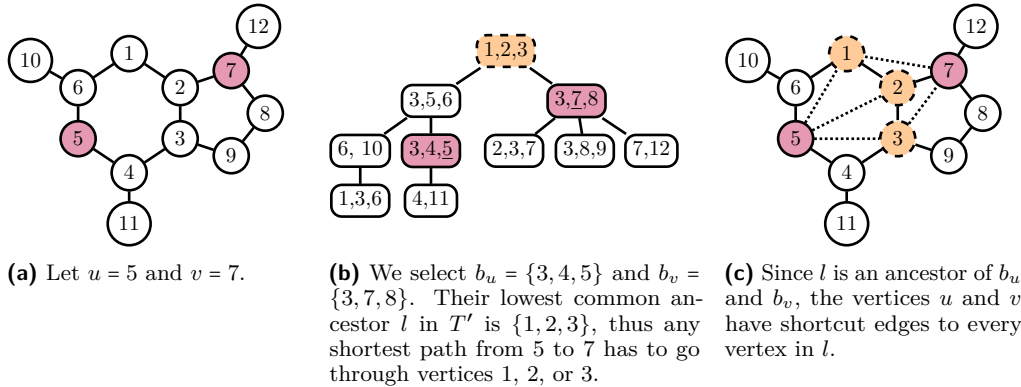
280 The latter equality is because every vertex has $O(\log n)$ ancestors.

281 For the second part, consider a shortest path π from u to v in G . Let π_T be the path
 282 from b_1 to b_2 in the tree decomposition T . By Lemma 3, π intersects the vertices of every
 283 bag b in π_T . Without loss of generality, we can assume that π stays in these bags, i.e. it only
 284 visits vertices in $\cup_{b \in \pi_T} V_b$. Note that if π leaves π_T , then it has to reenter it, but the exit and
 285 entry vertices are in the same bag and, by Lemma 7, there is already a shortcut edge between
 286 them. Additionally, since b_1 is an ancestor of b_2 in the centroid decomposition T' , there was a
 287 point in the construction of T' when b_1 was chosen as the centroid of a connected component
 288 containing b_2 . Thus, all the bags in π_T were also in the same connected component. Hence,
 289 every b is a descendant of b_1 . Therefore, the entire path π is included in G_b and the Dijkstra
 290 from u finds the shortest path to v and adds the corresponding shortcut edge. ◀

DETAILS OF STEP 5. Suppose our goal is to compute $d_G(u, v)$. We first pick two bags b_u
 and b_v such that $u \in b_u$ and $v \in b_v$. We then find the lowest common ancestor $l = \text{lca}(b_u, b_v)$.
 By Lemma 4, every path from u to v has to intersect V_l . Thus, we compute

$$d_G(u, v) = \min_{w \in V_l} (d_G(u, w) + d_G(w, v)).$$

291 Note that since l is an ancestor of both b_u and b_v , we have the distances needed on the RHS
 292 as weights of direct shortcut edges. This is illustrated in Figure 7



■ **Figure 7** An Example of Step 5 on the Graph and Decompositions of Figure 4.

293 ▶ **Lemma 10.** *The procedure above returns the correct distances in $O(n + k \cdot \sqrt{n}/\epsilon^2)$.*

294 **Proof.** Correctness is already argued above. Since the centroid decomposition T' has $O(n)$
 295 bags, preprocessing and answering offline lowest common ancestor queries takes $O(n +$
 296 $\sqrt{n}/\epsilon^2)$ [20]. For each of the \sqrt{n}/ϵ^2 queries generated in Step 4, we should compute the
 297 minimum of $O(k)$ values since $|V_l| \leq k + 1$. ◀

*Our graph is weighted at this point, but all edge weights and distances are non-negative integers less than n . Thus, Dijkstra runs in linear time on the number of vertices and edges. Intuitively, instead of keeping a priority queue of vertices in our Dijkstra, we can simply keep an array $A[n]$ of lists where $A[i]$ contains all vertices of distance i to the source.

298 Finally, the following is our main theorem in this work:

299 ► **Theorem 11.** *Given an $\epsilon > 0$, an undirected unweighted graph $G = (V, E_G)$ with n vertices
300 and a tree decomposition $T = (\mathcal{B}, E_T)$ of G with $O(n)$ bags and width k , our algorithm runs
301 in time $O(n \cdot \log n \cdot k^3 + \sqrt{n} \cdot k / \epsilon^2)$ and produces a $(1 + \epsilon)$ -approximation of the Wiener index
302 $W(G)$ with probability at least $2/3$.*

303 **Proof.** Correctness of the approximation ratio and success probability follows from Theorem 5
304 since our algorithm is the same as [21] except for how we answer distance queries. Step 1 takes
305 $O(n)$ using well-known algorithms such as [6, 12]. Step 2 takes $O(n \cdot k^3)$ based on Lemma 7.
306 Step 3 takes $O(n \cdot \log n \cdot k^3)$ as shown in Lemma 9. Step 4 simply takes $O(\sqrt{n}/\epsilon^2)$ samples
307 from the uniform distribution and Step 5 takes $O(n + k \cdot \sqrt{n}/\epsilon^2)$ time as per Lemma 10.
308 Finally, Step 6 takes $O(\sqrt{n}/\epsilon^2)$ time. Summing these up leads to the desired asymptotic
309 time complexity. ◀

310 4 Experimental Results

311 In this section, we present our experimental results, comparing the runtimes of our algorithm
312 with previous approaches. We implemented the main algorithms in C++ and provided
313 the same inputs, i.e. graph G , tree decomposition T and $\epsilon = 0.1$ to all of them. To obtain
314 this input, we first used psmiles [26], RDKit [27] and NetworkX [22] for preprocessing
315 molecular data and turning them into graphs. We also used FlowCutter [36] to compute
316 tree decompositions. All our experiments were conducted on an Intel Core i5 (2.3 GHz,
317 Quad-core) Machine with 8 GB of RAM running MacOS. We enforced a time limit of 1000
318 seconds per instance.

319 **BENCHMARKS.** We used the following datasets for our experiments: (i) PubChem [19] and
320 (ii) Protein Data Bank (PDB). Specifically, we report results on 100 randomly-selected protein
321 molecules from the PDB database and 1,311,229 molecules from PubChem.

322 **PDB.** The Protein Data Bank (PDB) [32] is an extensive repository of three-dimensional
323 structural data for large biological molecules, including proteins, DNA and RNA. We randomly
324 selected 100 protein molecules from this database. Table 2 shows some statistics about these
325 molecules. We observed that even the large molecules in this dataset have small treewidth.

■ **Table 2** Statistics of the PDB Benchmarks

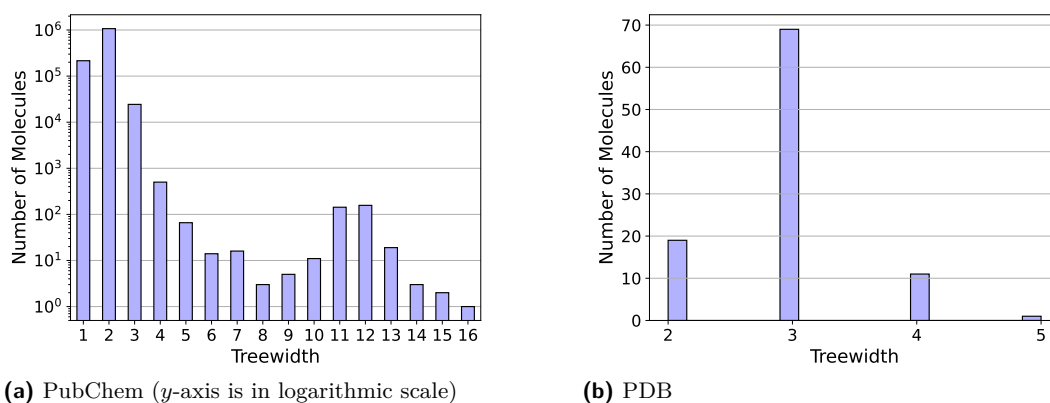
	Minimum	Maximum	Average
Number of Vertices	682	80652	9027
Number of Edges	743	82055	9240
Treewidth	2	5	2.95

326 **PUBCHEM.** PubChem [19] is an open chemistry database of the National Institutes of Health
327 (NIH). It includes information on chemical structures, identifiers, chemical and physical
328 properties, and biological activities of small molecules. As benchmarks, we took the following
329 datasets from PubChem: Common Chemistry CAS, Nature Catalysis, Wikipedia, Nature
330 Communications, Wiley, Springer Nature, Nature Chemistry, Nature Portfolio Journals,
331 Springer Materials, Drug and Medication, Nature Synthesis, Nature Chemical Biology,
332 KEGG, DrugBank. Collectively, these datasets contained 1,311,229 molecules at the time of
333 writing. See Table 3 for the statistics over this set of benchmarks.

■ **Table 3** Statistics of the PubChem Benchmarks

Metric	Minimum	Maximum	Average
Number of Vertices	2	568	21
Number of Edges	1	643	22
Treewidth	1	16	1.8

334 TREEWIDTH OF THE MOLECULES. As mentioned in Tables 2 and 3, we observed that the
 335 chemical compounds in both benchmark suites exhibit bounded treewidth. Figure 8 provides
 336 a histogram for each benchmark suite. Notably, the vast majority of PubChem compounds
 337 have a treewidth of less than 10, with very few molecules having treewidths of up to 16. In
 338 addition, the large protein molecules in the PDB dataset also have bounded treewidths of at
 339 most 5.

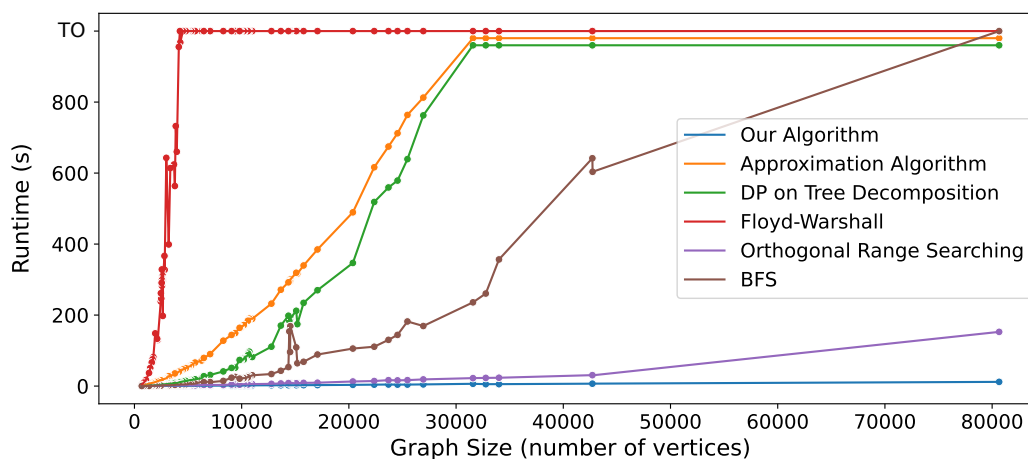


■ **Figure 8** Treewidth Distribution in Our Benchmarks

340 RESULTS. Figure 9 compares the performance of our algorithm and previous methods over the
 341 PDB dataset, whereas Table 4 provides the same comparison for PubChem. Our approach's
 342 better asymptotic complexity leads to significant gains in efficiency when considering the
 343 large graphs in PDB. However, no benefit is observed over the PubChem molecules, since
 344 they are all small and every algorithm can handle them in under 1 ms.

■ **Table 4** Runtime Comparison of the Algorithms of Table 1 over PubChem Benchmarks. All times are in milliseconds.

Algorithm	Maximum	Minimum	Average
Our Algorithm	1.425	0.187	0.296454
Approximation Algorithm	1.283	0.203	0.297342
DP on Tree Decomposition	1.256	0.192	0.296152
Floyd-Warshall	2.261	0.199	0.288638
Orthogonal Range Searching	1.121	0.199	0.292404
BFS	1.097	0.205	0.290523



■ **Figure 9** Runtime Comparison of the Algorithms of Table 1 over PDB Benchmarks. Each dot corresponds to one benchmark molecule.

345 5 Conclusion

346 In this work, we considered the problem of computing the Wiener index of a graph with
 347 n vertices and treewidth k . We provided a novel algorithm using a combination of tree
 348 decompositions and centroid decompositions, which achieves an almost-linear FPT runtime
 349 of $O(n \cdot \log n \cdot k^3 + \sqrt{n} \cdot k/\epsilon^2)$ and outputs a $(1 + \epsilon)$ -approximation of the Wiener index with
 350 probability at least $2/3$. To the best of our knowledge, this is the first sub-quadratic time
 351 FPT algorithm for this problem. We also showed that many real-world molecular graphs
 352 have small treewidth and thus our algorithm is applicable in practice.

353 6 Acknowledgments

354 The research was partially supported by the Hong Kong Research Grants Council ECS
 355 Project Number 26208122.

356 — References —

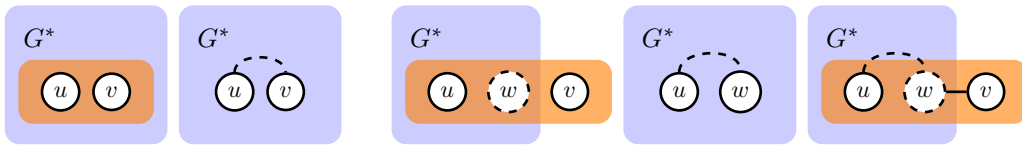
- 357 1 Tatsuya Akutsu and Hiroshi Nagamochi. Comparison and enumeration of chemical graphs.
358 *Computational and structural biotechnology journal*, 5(6):e201302004, 2013.
- 359 2 Hans L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *ICALP*,
360 volume 317, pages 105–118, 1988.
- 361 3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth.
362 *SIAM J. Comput.*, 25(6):1305–1317, 1996.
- 363 4 Hans L Bodlaender et al. A tourist guide through treewidth. 1992.
- 364 5 Danail Bonchev. *Chemical graph theory: introduction and fundamentals*, volume 1. CRC Press,
365 1991.
- 366 6 Gerth Stølting Brodal, Rolf Fagerberg, Christian N. S. Pedersen, and Anna Östlin. The
367 complexity of constructing evolutionary trees using experiments. In *ICALP*, volume 2076,
368 pages 140–151, 2001.
- 369 7 Sergio Cabello and Christian Knauer. Algorithms for graphs of bounded treewidth via
370 orthogonal range searching. *Computational Geometry*, 42(9):815–824, 2009.
- 371 8 Shiva Chaudhuri and Christos D Zaroliagis. Shortest paths in digraphs of small treewidth.
372 part i: Sequential algorithms. *Algorithmica*, 27:212–226, 2000.
- 373 9 Victor Chepoi and Sandi Klavžar. The wiener index and the szeged index of benzenoid systems
374 in linear time. *Journal of chemical information and computer sciences*, 37(4):752–755, 1997.
- 375 10 Giovanna K Conrado, Amir K Goharshady, Harshit J Motwani, and Sergei Novozhilov.
376 Parameterized algorithms for topological indices in chemistry. *arXiv preprint arXiv:2303.13279*,
377 2023.
- 378 11 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin
379 Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.
- 380 12 Davide della Giustina, Nicola Prezza, and Rossano Venturini. A new linear-time algorithm for
381 centroid decomposition. In *SPIRE*, pages 274–282, 2019.
- 382 13 E Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*,
383 1:269–271, 1959.
- 384 14 Andrey A Dobrynin, Ivan Gutman, Sandi Klavžar, and Petra Žigert. Wiener index of hexagonal
385 systems. *Acta Applicandae Mathematica*, 72:247–294, 2002.
- 386 15 Alexander G Dossetter, Edward J Griffen, and Andrew G Leach. Matched molecular pair
387 analysis in drug discovery. *Drug Discovery Today*, 18(15-16):724–731, 2013.
- 388 16 Roger C Entringer, Douglas E Jackson, and DA Snyder. Distance in graphs. *Czechoslovak*
389 *Mathematical Journal*, 26(2):283–296, 1976.
- 390 17 Ernesto Estrada and Eugenio Uriarte. Recent advances on the role of topological indices in
391 drug discovery research. *Current Medicinal Chemistry*, 8(13):1573–1588, 2001.
- 392 18 Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.
- 393 19 National Center for Biotechnology Information. Pubchem database. <https://pubchem.ncbi.nlm.nih.gov>.
- 394 20 Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of
395 disjoint set union. In *STOC*, pages 246–251, 1983.
- 396 21 Oded Goldreich and Dana Ron. Approximating average parameters of graphs. *Random*
397 *Structures & Algorithms*, 32(4):473–493, 2008.
- 398 22 Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and
399 function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos,
400 NM (United States), 2008.
- 401 23 Rudolf Halin. S-functions for graphs. *Journal of geometry*, 8:171–186, 1976.
- 402 24 Christoph Helma. *Predictive toxicology*. CRC Press, 2005.
- 403 25 Camille Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathem-*
404 *atik*, 70:185–190, 1869.
- 405 26 Peter C Kroon. pysmiles: A python library for parsing smiles strings. [https://pypi.org/
406 project/pysmiles/](https://pypi.org/project/pysmiles/).

- 408 27 Gregory Landrum. Rdkit: Open-source cheminformatics. <https://www.rdkit.org>.
- 409 28 Jerzy Leszczynski. *Handbook of computational chemistry*, volume 3. Springer Science &
410 Business Media, 2012.
- 411 29 Bojan Mohar and Tomaž Pisanski. How to compute the wiener index of a graph. *Journal of*
412 *mathematical chemistry*, 2(3):267–277, 1988.
- 413 30 Edward F Moore. The shortest path through a maze. In *Proc. of the International Symposium*
414 *on the Theory of Switching*, pages 285–292, 1959.
- 415 31 Jaroslav Nešetřil and Patrice Ossona De Mendez. Structural properties of sparse graphs. In
416 *Building Bridges: Between Mathematics and Computer Science*, pages 369–426. Springer, 2008.
- 417 32 RCSB. Protein data bank. <https://www.rcsb.org>.
- 418 33 Neil Robertson and Paul D Seymour. Graph minors. iii. planar tree-width. *Journal of*
419 *Combinatorial Theory, Series B*, 36(1):49–64, 1984.
- 420 34 Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width.
421 *Journal of algorithms*, 7(3):309–322, 1986.
- 422 35 L’ubomír Šoltés. Transmission in graphs: a bound and vertex removing. *Mathematica Slovaca*,
423 41(1):11–16, 1991.
- 424 36 Ben Strasser and KIT algorithms group. Flowcutter: Software for computing flow-based
425 balanced graph cuts. <https://github.com/kat-algo/flow-cutter-pace17>.
- 426 37 Nenad Trinajstić. *Chemical graph theory*. Routledge, 2018.
- 427 38 Stephan Wagner and Hua Wang. *Introduction to chemical graph theory*. CRC Press, 2018.
- 428 39 Pengfei Wan, Jianhua Tu, Shenggui Zhang, and Binlong Li. Computing the numbers of
429 independent sets and matchings of all sizes for graphs with bounded treewidth. *Applied*
430 *Mathematics and Computation*, 332:42–47, 2018.
- 431 40 Harry Wiener. Structural determination of paraffin boiling points. *Journal of the American*
432 *chemical society*, 69(1):17–20, 1947.
- 433 41 Jun Xu and Arnold Hagler. Chemoinformatics and drug discovery. *Molecules*, 7(8):566–600,
434 2002.
- 435 42 Ling Xue and Jurgen Bajorath. Molecular descriptors in chemoinformatics, computational
436 combinatorial chemistry, and virtual screening. *Combinatorial chemistry & high throughput*
437 *screening*, 3(5):363–372, 2000.
- 438 43 Atsuko Yamaguchi, Kiyoko F Aoki, and Hiroshi Mamitsuka. Graph complexity of chemical
439 compounds in biological pathways. *Genome Informatics*, 14:376–377, 2003.
- 440 44 Atsuko Yamaguchi, Kiyoko F Aoki, and Hiroshi Mamitsuka. Finding the maximum common
441 subgraph of a partial k-tree and a graph with a polynomially bounded number of spanning
442 trees. *Information Processing Letters*, 92(2):57–63, 2004.
- 443 45 Atsuko Yamaguchi and Kiyoko F Aoki-Kinoshita. Chemical compound complexity in biological
444 pathways. *Quantitative Graph Theory: Mathematical Foundations and Applications*, page 471,
445 2014.
- 446 46 Konrad Zuse. Der plankalkül. 1972.

447 **A Proof of Lemma 4**

448 **Proof.** We prove this lemma through induction on the size n of the tree. If n is at most 3,
 449 the lemma holds trivially. Now assume that the lemma holds for all trees with a size less
 450 than n . Let us consider a general tree of size n . In the first step, we identify a centroid
 451 node of T , denoted as c . The node c fragments T into several connected components. If
 452 any arbitrary vertices $u, v \in V_T$ exist in the same connected component T_i , then in the
 453 corresponding centroid decomposition T' , they will appear in T'_i as per the definition of
 454 centroid decomposition. According to the inductive hypothesis, their path must cross their
 455 lowest common ancestor in T'_i . In case they belong to different connected components, say
 456 T'_i and T'_j , any path originating from T'_i and terminating at T'_j must traverse the node c .
 457 This is because c separates T_i and T_j . In this scenario, their lowest common ancestor would
 458 be the root c , as the remaining nodes on the path from u to v are either in T_i or T_j and,
 459 hence, cannot serve as the common ancestor. ◀

460 **B Proof of Lemma 7**



(a) When u and v appear in G^* , a shortcut edge will be calculated during the recursive call on G^* .
 (b) If v is not in G^* , its path to u must contain a vertex w that is in the same bag ℓ as u and v and that also appears in G^* . A shortcut edge from u to w will be added during the processing of G^* and thus the path from u to v can be calculated in Step v.

■ **Figure 10** An Illustration of Lemma 7.

461 **Proof.** We run the Floyd-Warshall algorithm twice on each bag of the tree decomposition,
 462 once in Step ii and once in v. Since each bag has $k + 1$ vertices and the tree decomposition
 463 has $O(n)$ bags, the total runtime is $O(n \cdot k^3)$. The procedure above adds new shortcut edges
 464 only between pairs of vertices that were already in the same bag, thus the tree decomposition
 465 remains valid.

466 We prove the last property by induction on $|\mathcal{B}|$. If $|\mathcal{B}| = 1$, then the first Floyd-Warshall
 467 in Step ii adds all the necessary shortcut edges. Otherwise, let $u, v \in V_\ell$ be two vertices that
 468 appear in the leaf bag ℓ and let $p \in \mathcal{B}$ be the parent of ℓ in T . If there is a path between u
 469 and v that is entirely within V_ℓ , then Step ii adds a shortcut edge summarizing this path.
 470 Thus, if $u', v' \in G^*$, then $d_{G^*}(u, v) = d_G(u, v)$. Moreover, both u and v have to appear in p ,
 471 since they each appear in a connected subtree. Hence, by induction hypothesis, the recursive
 472 call in Step iv adds the required shortcut edge between u and v . Now consider the case
 473 where either u or v (or both) are not in G^* . Take a shortest path π from u to v in G . If π
 474 is entirely within V_ℓ , then Step ii adds the shortcut edge. Otherwise, we use Lemma 3 to
 475 break π down as $\pi = \pi_1 \cdot w_1 \cdots w_2 \cdot \pi_2$ where π_1 is the longest prefix of π that only contains
 476 vertices from $V_\ell \setminus V_p$ and π_2 is the longest such suffix. By Lemma 3, we have $w_1, w_2 \in V_\ell \cap V_p$.
 477 Since they are both in $V_p \subseteq V_{G^*}$, Step iv adds a shortcut edge from w_1 to w_2 . Hence, Step v
 478 adds a shortcut edge from u to v with the correct weight. Finally, if u and v are vertices
 479 that appear in the same bag $b \neq \ell$, then the recursive call on (G^*, T^*) adds a shortcut edge
 480 between them. ◀