On a Conjecture on Wiener Indices in Combinatorial Chemistry

Yih-En Andrew Ban
Dept. of Biochemistry
Duke University.

Sergei Bespamyatnikh
Dept. of Computer Science,
University of Texas at Dallas.

Nabil H. Mustafa
Dept. of Computer Science
Duke University.

{aban, nabil}@cs.duke.edu, besp @utdallas.edu

Abstract

Drugs and other chemical compounds are often modeled as polygonal shapes, where each vertex represents an atom of the molecule, and covalent bonds between atoms are represented by edges between the corresponding vertices. This polygonal shape derived from a chemical compound is often called its *molecular graph*, and can be a path, a tree, or in general a graph. An indicator defined over this molecular graph, the *Wiener index*, has been shown to be strongly correlated to various chemical properties of the compound. The Wiener index conjecture for trees states that for any integer n (except for a finite set), one can find a tree with Wiener index n. This conjecture has been open for quite some time, and many authors have presented incremental progress on this problem. In this paper, we present further progress towards proving this conjecture — through the design of efficient algorithms, we show that enumerating all possible trees to verify this conjecture (as done by all the previous approaches) is not necessary, but instead searching in a small special family of trees suffices, thus achieving the first polynomial (in n) time algorithm to verify the conjecture up to integer n. More precisely, we (i) present an infinite family of trees and prove various properties of these trees, (ii) show that a large number of integers, up to at least 10^8 are representable as Wiener indices of trees in this succinct family, (iii) provide several efficient algorithms for computing trees with given Wiener indices, (iv) implement our algorithms and experimentally show that their performance is asymptotically much better than their theoretical worst-case upper bound.

1 Introduction

Drugs and other chemical compounds are often modeled as various polygonal shapes — paths, trees, graphs etc. Each vertex in the polygonal path or tree represents an atom of the molecule, and covalent bonds between atoms are represented by edges between the corresponding vertices. This polygonal shape derived from a chemical compound is often called its *molecular graph*. As the geometry of proteins play an important role in determining the function of the protein, so can the topological properties of the molecular graphs of chemical compounds be correlated to their chemical properties. For some time now, the biochemical community has been using topological indices in an attempt to correlate a compounds molecular graph with experimentally gathered data regarding the compounds characteristics.

Usage of topological indices in biology and chemistry began in 1947 when chemist Harold Wiener developed the most widely known topological descriptor [20, 27], the Wiener index, and used it to determine physical properties of types of alkanes known as paraffins[42]. In general, topological indices are one of the oldest and most widely used descriptors in quantitative structure activity relationships: Quantitative structure activity relationships (QSAR) is a popular computational biology paradigm in modern drug design [11, 31]. It attempts to encode biological activity such as inhibition or activation into numerical measures by correlation of mass amounts of data from an initial screening of hundreds or thousands of candidate drug compounds. The data is mapped into a structural "activity space" consisting of various descriptors with the hope that these spaces capture or estimate properties of chemical compounds. Constructions of new drugs then proceed by noting the desired activity spaces and creating new compounds which occupy those regions of space.

QSAR has experienced rigorous recent investigation [20, 15, 18, 29], and has been used to help develop a wide variety of drugs in medicine with a diversity ranging from cancer [44], to infectious diseases such as AIDS [34, 37] and malaria [3, 4], to antipsychotics [40] and anti-inflammatory agents [8]. Recent techniques in QSAR include molecular field analysis [36], quantum similarity [5, 4], particle swarms [1], k-nearest neighbor principles [43, 24], neural networks [2, 41, 35], and topological indices [20, 15, 18, 29]. The power of QSAR in quickly identifying lead drug candidates gives great motivation to both developing new techniques and further improving current techniques.

Amongst the topological indices used as descriptors in QSAR, the Wiener index is by far the most popular index, as it has been shown that the Wiener index has a strong cor-

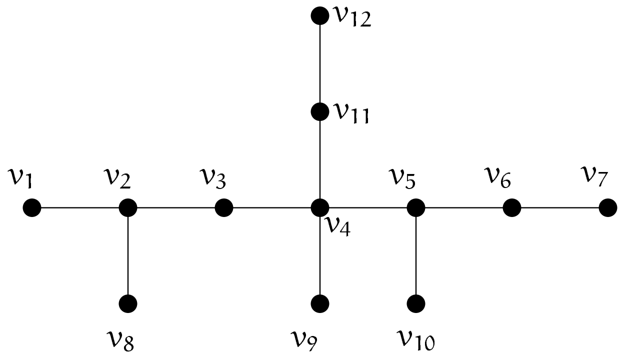

Figure 1: The molecular graph of the compound 2, 4, 5-trimethyl-4-ethyl heptane [13].

relation with the chemical properties of the compound [15]. Therefore, to construct a compound with a certain property correlated to some Wiener index, the objective becomes to build substructures in the target chemical compound giving the compound that Wiener index. This in turn leads to the following important problem: given a Wiener index, find a compound with that Wiener index.

An overwhelming majority of the chemical applications of the Wiener index deal with chemical compounds that have acyclic organic molecules (see [22, 38] for details). The molecular graphs of these compounds are trees [21]. See Figure 1 of the molecular graph of such a chemical compound. Therefore most of the prior work on Wiener indices deals with trees, relating the structure of various trees to their Wiener indices (asymptotic bounds on the Wiener indices of certain families of trees, expected Wiener indices of random trees etc.). For these reasons, we concentrate on the Wiener indices of trees as well (see Dobrynin *et al.* [13] for a recent survey).

1.1 Problem definition

DEFINITION 1.1. (**Wiener Index**) *Given an undirected graph* $G = (V, E)$*, denote by* $d(v_i, v_j)$ *the length of the shortest path between two distinct vertices* $v_i, v_j \in V$*. The Wiener index* $W(G)$ *is defined as*

$$W(G) = \sum_i \sum_{j>i} d(v_i, v_j) \tag{1.1}$$

In this paper, we will present results on the Wiener index problems where the graph G is a tree. While the case of graphs has been solved, a major conjecture in this area is whether every positive integer is the Wiener index of some tree.

CONJECTURE 1.1. (**Wiener Index Conjecture** [19, 30]) *Except for some finite set, every positive integer is the Wiener index of some tree.*

Towards this goal, we present a family of trees and design fast efficient algorithms for the Inverse Wiener problem, defined as follows [23].

PROBLEM 1.1. (**Inverse Wiener Problem,** $P_=$) *Given an integer* n*, construct a tree whose Wiener index is* n*.*

An algorithm for solving the problem $P_=$ can be used to verify the above conjecture — the algorithm for problem $P_=$ can be applied n times to verify the conjecture up to any integer n. However, there can be a faster algorithm, and so we introduce the following problem.

PROBLEM 1.2. (**Inverse Wiener Covering Problem,** $P_\leq$) *Given an integer* n*, for every* $i \leq n$ *construct a tree with Wiener index* i*.*

1.2 Previous work

Intensive work has been done on the theory of Wiener indices, both in the chemistry and biology community as well as in the mathematics community. There are broadly two types of problems that all of the previous work has addressed — *Wiener index* problem for graphs, and the *Inverse Wiener index* problem. The Wiener index problem deals with properties of the Wiener index of various substructures, i.e. the efficient computation of the index, the upper and lower bound on the index values etc., and the relation of the Wiener index to other qualities of the graph. The Inverse Wiener problem seeks to answer, given an integer n, the question about the existence of a tree T such that $W(T) = n$, and when this is the case, how to compute it efficiently.

Computing Wiener Index. The straightforward way for computing the Wiener index for any graph G is to compute distances between all pairs of vertices. This approach has at least $\Omega(n^2)$ running time since it has to check all $\binom{n}{2}$ pairs. Furthermore, for a pair (u, v), it could take $O(n)$ time to compute the distance $d(u, v)$, therefore with running time $O(n^3)$. One could use Johnson's algorithm [28] for all pairs shortest paths on sparse graphs to find $d(u, v)$ for all $u, v \in V$ in time $O(|V|^2 \log |V| + |E||V|)$ (using Fibonacci heaps), yielding the same time bound algorithm for computing Wiener index for general graphs.

For trees, although the number of edges is $O(n)$, the number of pairs of vertices is still $O(n^2)$. However, the Wiener index of a tree can be computed in $O(n)$ time, through a plethora of computational methods [10, 12, 33].

Upper and lower bounds. Given special graphs with n vertices, often closed form expressions exist for their Wiener index. See Figure 2 for some examples. It is easy to see that a path P_n of length n has Wiener index $\Theta(n^3)$, and a complete graph K_n has Wiener index $\Theta(n^2)$. Also, note that the Wiener index of every other graph or tree G must be between these two values, $W(K_n) \leq W(G) \leq W(P_n)$. In combinatorial chemistry, researchers have looked at specific polymers and alkanes, and formulated the bounds on their Wiener indices. For example, Bonchev *et al.* [7] present formulations of the Wiener index for linear polymers, star-like polymers, B-Combs etc. Similarly Rücker *et al.* [39] exhaustively enumerate cycloalkanes, computing their Wiener indices, and tabulating their variation and actual numerical bounds on the Wiener index.

There has also been some work on looking at asymptotic values of the Wiener index for random trees. Entringer *et al.* [14] show that for several classes of random trees (e.g. binary trees) with n vertices, $W(T) = \Theta(n^{5/2})$ as n goes towards ∞.

Inverse problem. The inverse Wiener problem is easily solvable for graphs: given an integer n, there always exists a graph G such that $W(G) = n$, and it is computable in constant time. This was shown by Goldman *et al.* [19]: as mentioned before, a complete graph K_n has $W(K_n) = n(n-1)/2$ while a star graph S_n has $W(S_n) = (n-1)^2$. Observe that adding an edge to S_n decreases the Wiener index by 1, and this can be iteratively repeated till a complete graph is reached. This gives a continuous interval of Wiener indices from $n(n-1)/2$ to $(n-1)^2$. The next interval defined by S_{n+1} and K_{n+1} overlaps this interval and so forth, thus showing that any n has a graph with Wiener index n.

However, once the graph is restricted to trees, Goldman *et al.* note that the problem becomes complicated and the complexity is unknown. In fact, the complexity of this question is left unsolved in Gutman and Yeh [23], and conjectured that apart from a finite set, all integers do have corresponding trees with the required Wiener indices. Later Lepović and Gutman [30] present a exhaustive search algorithm that verifies this conjecture up to integer 1206, still leaving the Inverse Wiener conjecture open. Goldman *et al.* attempt to solve $P_=$ by defining a recurrence relation on Wiener indices of trees by using dynamic programming to construct trees from smaller subtrees. Using their exponential time algorithm, they are able to verify the above conjecture for integers up to 10000. However, the Inverse Wiener conjecture remains open.

1.3 Our results

We present progress towards proving the Wiener index conjecture by showing that searching all possible trees is not necessary — a small parameterized subset suffices. More specifically, our contributions are:

- We present a family of trees, $\mathcal{F}$, and give reasons why it is a good candidate to analyze, rather than searching over the (exponential) space of all possible trees. This contrasts with the previous work of Goldman *et al.*[19] where they addressed problem $P_=$ by exhaustively checking trees using an exponential time algorithm.
- We present $O(k)$ time algorithms for computing the Wiener index of a tree in $\mathcal{F}$, where $k \leq n$ is a tree parameter. In our algorithms, k will be a small constant.
- We conjecture that the Wiener indices of our proposed family of trees cover all the integers except a small finite set. We further present a succinct four parameter family of trees that we also conjecture can express all integers.
- We present several efficient algorithms for the problem $P_=$, using variants of Frederickson and Johnsons matrix searching algorithm, and a novel use of decomposition by sorted sequences in time $O(nk)$, where k is a decomposition parameter, and by exploiting the special structure of the Wiener polynomial (in terms of the tree parameters) for our family of trees. We present empirical results on the efficiency of this algorithm, indicating that this algorithm is *orders* of magnitude faster than that using the Frederickson and Johnson technique.
- We present several efficient algorithms for the problem $P_\leq$, and give empirical results showing that it is faster by a factor of $O(\sqrt{n})$ than the naive implementation.

Finally, we believe that the approaches given in this paper can be used as general techniques for tree construction problems in combinatorial biology and chemistry when it is necessary to deal with tree classes.

2 Family of Trees

First, we state that given a tree $T = (V, E)$, one can assign integer weights to the edges of the tree such that the sum of the edge weights is exactly the Wiener index of the tree.

LEMMA 2.1. (DOBRYNIN *et al.* [13]) *Given* $T = (V, E)$, *one can assign weights* $w(e)$ *for each* $e \in E$ *such that* $W(T) = \sum_{e \in E} w(e)$.

PROOF. Look at the set of all paths from every vertex in V to every other vertex in V. The Wiener index is simply the sum of the lengths of these paths. For an edge $e \in E$, the number of all possible paths passing through the edge e is simply the product of the sizes of the two subtrees formed by deleting the edge e. Let this number be the weight $w(e)$ of the edge e. Then it follows that

$$W(T) = \sum_i \sum_{j>i} d(v_i, v_j) = \sum_e w(e)$$

□

Previous work has tended to focus on enumerating all possible trees and computing the Wiener index of each

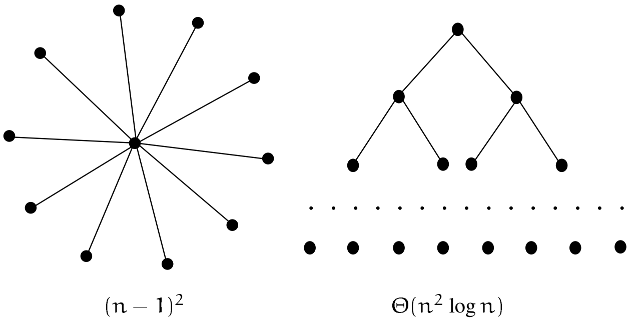

Figure 2: An illustration of various trees with n vertices, and their corresponding Wiener indices.

such tree [19]. This approach results in exponential time algorithms, and hence the previous limited computational results [19, 23, 30]. We follow a different path by defining a family of trees which allow efficient algorithms for the inverse Wiener problem. The required family of trees must satisfy the following desirable conditions:

- **Simple generation.** The required trees must be easy to generate using a set of integer parameters.
- **Low-order polynomials.** The Wiener index of a tree must be a low degree polynomial function of the input parameters. Low degree polynomials are easy to compute, and mathematically one can cover a larger set of integers using lower degree polynomials due to their higher density.

We present an infinite family of trees, satisfying the above criteria, whose structure allows for the realization of a vast number of Wiener indices. Define $T_k(r_1, \ldots, r_k)$ be a tree of *order k* where

$$V = \{s_1, \ldots, s_k\} \cup \left\{t_1^1, \ldots, t_{r_1}^1, \ldots, t_1^k, \ldots, t_{r_k}^k\right\}$$

and

$$\begin{aligned} E \;\; = \;\; & \{(s_i, s_{i+1}),\ 1 \le i \le (k-1)\} \;\cup \\ & \{(t_l^j, s_j),\ 1 \le j \le k,\ 1 \le l \le r_j\}. \end{aligned}$$

Note that $m = |V| = \sum_i r_i + k$. Tree T_k is thus defined by k parameters $r_i, i = 1 \ldots k$. Figure 3 illustrates the structure of the family of trees. An interesting question arises whether the Wiener index of $T_k(r_1, \ldots, r_k)$ can be computed in *sub-linear* time in the number of vertices m of T_k. We answer it in affirmative below. We will again compute the Wiener index of the tree family by computing the weight of each edge and summing it up, as in Lemma 2.1.

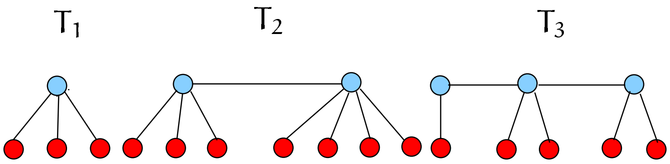

Figure 3: Examples of trees - $T_1\,(3)$, $T_2\,(3,4)$, and $T_3\,(1,2,2)$.

FACT 2.1.

$$\begin{aligned} w(T_k(r_1, \ldots, r_k)) \;\; &= \;\; \sum_{i=1}^{k-1} \left(\sum_{j=1}^{i} (r_j + i) \cdot \sum_{j=i+1}^{k} (r_j + (k-i)) \right) \\ &\quad + (m-k) \cdot (m-1) \qquad (2.2) \end{aligned}$$

PROOF. Summing up the weights of all the edges $(u, v) \in E$, denoted as $w(u, v)$, we get

$$W(T_k(r_1, \ldots, r_k)) = \sum_{i=1}^{k-1} w((s_i, s_{i+1})) + \sum_{j=1}^{k} \sum_{l=1}^{r_j} w((t_l^j, s_j))$$

It can be verified that

$$w((t_l^j, s_j)) = 1 \cdot (\sum_{i=1}^{k} r_i + (k-1)) = (m-1)$$

and

$$w((s_i, s_{i+1})) = (\sum_{j=1}^{i} r_j + i) \cdot (\sum_{j=i+1}^{k} r_j + (k-i))$$

Summing the terms yields Equation 2.2. $\square$

LEMMA 2.2. *The Wiener index of a tree of order* k *can be computed in* $O(k)$ *time.*

PROOF. Let $X_i = \sum_{j=1}^{i} r_j$, and $Y_i = \sum_{j=i+1}^{k} r_j$. Numbers X_i and Y_i can be computed in $O(k)$ time, for all $1 \le i \le k$ in an incremental fashion. Note that $\sum_{j=1}^{i}(r_j + i) = X_i + \binom{i+1}{2}$. Similarly, $\sum_{j=i+1}^{k} r_j + (k-i) = Y_i + \binom{k+1}{2} - \binom{i+1}{2}$. By Equation 2.2, $W(T_k(r_1, \ldots, r_k))$ can be computed $O(k)$ time. $\square$

We have implemented the above algorithm for the computation of Wiener indices of T_k. We now experiment to find out the set of Wiener indices not covered by our family of trees T_k. We omit details, but we verified that integers up to 10^8, except for a few numbers, can be represented as $W(T_k)$, for some k, leading us to state the following.

CONJECTURE 2.1. *Except a set S_1 (Table 1) of 56 numbers ≤ 193, all integers can be represented by $W(T_k)$.*

Actually, our experiments indicate that all numbers n, $10^3 \leq n \leq 10^8$, can be presented by $W(T_k)$, where $k \leq 5$. However, it seems that the Wiener indices of the trees T_k, $k \leq 4$, do not cover all integers except a finite set and we illustrate that by listing the set of numbers, S_2 (Table 1), less than 10^6 that are not representable. Based on the above results, we strengthen conjecture 2.1 by replacing the infinite family of trees T_k by a single family of trees for $k = 5$.

CONJECTURE 2.2. *Except a set S_3 (Table 1) of 102 numbers ≤ 557, all integers can be represented by $W(T_5)$.*

We have verified this conjecture for n up to 10^8. For $k = 5$, the Wiener index of $T_5(r_1, \ldots, r_5)$ simplifies to

$$\begin{aligned} W(T_5(\ldots)) &= r_1 \cdot (r_1 + 3r_2 + 4r_3 + 5r_4 + 6r_5 + 14) \\ &\quad + r_2 \cdot (r_2 + 3r_3 + 4r_4 + 5r_5 + 11) \\ &\quad + r_3 \cdot (r_3 + 3r_4 + 4r_5 + 10) \\ &\quad + r_4 \cdot (r_4 + 3r_5 + 11) \\ &\quad + r_5 \cdot (r_5 + 14) + 20. \end{aligned}$$

Note the symmetry between the parameters r_1 and r_5 and similarly between parameters r_2 and r_4. We exploit this symmetric property strongly in our algorithms.

We recall Lagrange's Theorem in number theory, that every integer can be represented as a sum of four squares [26]. For example, the polynomial for the Wiener index (in terms of r_i) for $T_5(r_1, \ldots, r_5)$ consists *only* of quadratic and linear terms. That gives the intuition that a properly chosen four parameter family of trees might be sufficient to represent any Wiener index. One way to remove one parameter would be to impose a constraint on the function $W(T_5)$. After some experimentation, we discovered that the constraint $r_1 = r_5$ still allows for the representation of all integers as Wiener indices, except for a certain finite set. Consider the family of trees $T_5(r_1, \ldots, r_5)$ such that $r_1 = r_5$. We denote this family of trees by $F(\cdot)$, where $F(r_1, r_2, r_3, r_4) = T_5(r_1, r_2, r_3, r_4, r_1)$. Then simplifying the Wiener formula for $W(T_5(r_1, \ldots, r_5))$ gives,

$$\begin{aligned} W(F(r_1, r_2, r_3, r_4)) &= r_1 \cdot (8r_1 + 8r_2 + 8r_3 + 8r_4 + 28) \\ &\quad + r_2 \cdot (r_2 + 3r_3 + 4r_4 + 11) \\ &\quad + r_3 \cdot (r_3 + 3r_4 + 10) \\ &\quad + r_4 \cdot (r_4 + 11) + 20. \end{aligned} \tag{2.3}$$

CONJECTURE 2.3. *Except a set S_4 (Table 1) of 181 numbers ≤ 1177, all integers can be represented by $W(F)$.*

3 Algorithm for the problem $P_=$

In this section we give a suite of algorithms for finding trees whose Wiener index is n, given the integer n as input to the algorithm. First, observe that the value of each r_i is bounded in terms of n.

CLAIM 3.1. *Given an integer n, any tree T of family $F(r_1, r_2, r_3, r_4)$ with $W(T) = n$ must have $0 \leq r_i \leq \sqrt{n}$.*

Recall that each tree in $F(V, E)$ is defined by four parameters r_1, r_2, r_3 and r_4. Given n, the objective is to find a tree $T \in F(V, E)$ such that $W(T) = n$. We call the set of all 4-tuples (r_1, r_2, r_3, r_4) the *configuration space* of $W(F(\cdot))$. To find a given integer n, we want to search this configuration space. The straightforward way of computing $F(V, E)$ is to exhaustively traverse this configuration space, i.e. iterate over all possible r_i's, and compute $W(T)$ for each 4-tuple. By Claim 3.1, the running time is $O(n^2)$.

However, on examining Equation 2.3, one finds that the equation is monotone in all parameters r_i. Therefore, a fast algorithm is as follows. Iterate over all values of r_1, r_2, and r_3. So suppose that r_1, r_2 and r_3 are some fixed constants, say c_1, c_2 and c_3 respectively. Perform the binary search over the sequence $W(F(c_1, c_2, c_3, r_4))$, $0 \leq r_4 \leq \sqrt{n}$, to find if $W(T) = n$ for some value of r_4. The running time therefore reduces to $O(n^{3/2} \log n)$.

We show how to improve the above algorithm. Suppose that $r_1 = c_1$ and $r_2 = c_2$ are fixed. Consider the matrix M of values $W(T)$ for all $0 \leq r_3, r_4 \leq \sqrt{n}$, i.e. the matrix $M_{ij} = W(F(c_1, c_2, i, j))$. Again from the equation 2.3, it follows that M is a sorted matrix, i.e. each row and column of M is nondecreasing. We therefore apply the searching technique of Frederickson and Johnson [16] for searching for a number in a sorted matrix. One can think of the previous algorithm of performing binary search over r_4 as performing binary search over each row of the matrix M, i.e. each row i corresponds to a fixed parameter value $r_3 = i$, with the value of r_4 varying across the row, and $r_1 = c_1$ and $r_2 = c_2$. Frederickson and Johnson's technique performs exponential search over M, saving an extra logarithmic factor by amortizing the search using the fact that not only the rows are non-decreasing, but the columns are non-decreasing as well, and one can interleave the binary search between rows and columns to finally get the searching time as $O(\sqrt{k})$, where k is the rank of n in M. We still need to iterate over all possible values for r_1 and r_2, giving the total searching time of our Wiener searching algorithm $O(n^{3/2})$. Furthermore, note that if we allow the use of computing square roots in constant time, then the equation 2.3, given $r_1 = c_1, r_2 = c_2, r_3 = c_3$, reduces to a quadratic equation $W(F(c_1, c_2, c_3, r_4)) = n$ which can be solved in constant time. If one of the roots is a non-negative integer, that root is the required value of r_4.

THEOREM 3.1. *Given an integer n, one can find (if there exist) integers c_1, c_2, c_3 and c_4 such that $W(F(c_1, c_2, c_3, c_4)) = n$ in time $O(n^{3/2})$.*

3.1 Decomposition using sorted sequences

We now analyze the structure of equation 2.3 more closely and use it to present an even more efficient algorithm for our problem. The first thing to note is the symmetry of the equation between r_2 and r_4, i.e. r_1 "contributes" equally to (the coefficients of) r_2 and r_4, and similarly r_3 "contributes" equally to (the coefficients of) r_2 and r_4. Therefore, instead of fixing r_1 and r_2 as before and trying to find r_3 and r_4 values more efficiently, we fix $r_1 = c_1, r_3 = c_3$. As explained above, this is crucial since r_1 and r_3 contribute symmetrically to r_2 and r_4. Then

$$\begin{aligned} W(F(c_1, r_2, c_3, r_4)) &= r_2 \cdot (8c_1 + r_2 + 3c_3 + 11) \\ &\quad + r_4 \cdot (8c_1 + r_4 + 3c_3 + 11) \\ &\quad + 4r_2r_4 + K(c_1, c_3) \\ &= (r_2 + r_4) \cdot (8c_1 + 3c_3 + 11) + \\ &\quad r_2^2 + r_4^2 + 4r_2r_4 + K(c_1, c_3) \\ &= (r_2 + r_4) \cdot (8c_1 + 3c_3 + 11) + \\ &\quad (r_2 + r_4)^2 + 2r_2r_4 + K(c_1, c_3) \end{aligned}$$

where $K(c_1, c_3) = (c_3^2 + 8c_1^2) + (8c_1c_3) + (28c_1 + 10c_3) + 20$ is a constant.

LEMMA 3.2. *Given integers* $r_2, r_4, s_2, s_4, c_1, d_1, c_3, d_3$,

$$W(F(c_1, r_2, c_3, r_4)) \geq W(F(c_1, s_2, c_3, s_4)) \implies$$
$$W(F(d_1, r_2, d_3, r_4)) \geq W(F(d_1, s_2, d_3, s_4)),$$

if $r_2 + r_4 \geq s_2 + s_4$ *and* $8(d_1 - c_1) + 3(d_3 - c_3) \geq 0$.

PROOF. Assume that $W(F(c_1, r_2, c_3, r_4)) \geq W(F(c_1, s_2, c_3, s_4))$. We will show that the increment in the Wiener index for the configuration (c_1, r_2, c_3, r_4) is larger than the increment for the configuration (c_1, s_2, c_3, s_4), i.e.

$$\begin{aligned} &W(F(d_1, r_2, d_3, r_4)) - W(F(c_1, r_2, c_3, r_4)) \geq \\ &\qquad W(F(d_1, s_2, d_3, s_4)) - W(F(c_1, s_2, c_3, s_4)) \\ &(r_2 + r_4)(8d_1 + 3d_3 + 11 - 8c_1 - 3c_3 - 11) \geq \\ &\qquad (s_2 + s_4)(8d_1 + 3d_3 + 11 - 8c_1 - 3c_3 - 11) \\ &(r_2 + r_4)(8(d_1 - c_1) + 3(d_3 - c_3)) \geq \\ &\qquad (s_2 + s_4)(8(d_1 - c_1) + 3(d_3 - c_3)) \end{aligned}$$

Since $8(d_1 - c_1) + 3(d_3 - c_3) \geq 0$, and $(r_2 + r_4) \geq (s_2 + s_4)$, the proof follows. ◻

We now use Lemma 3.2 to identify large subsets of the configuration space that can be searched to find a specific element much more efficiently.

Set $r_1 = r_3 = 0$. Define $w(\ (a, b)\ ) = W(F(0, a, 0, b))$ and $c(\ (a, b)\ ) = a + b$. Let $P = \langle\ p_i = (a_i, b_i) | 0 \leq a_i, b_i \leq \sqrt{n}\ \rangle$ be sorted by $w(\cdot, \cdot)$. Then the sequence of $|P| = n$ pairs represents the Wiener indices of all possible pairs of parameters r_2 and r_4. Let $P' = \langle p_{i_1}, \ldots, p_{i_k} \rangle$ be the longest subsequence of P such that the sequence $C(P') = \langle c(p_{i_1}), \ldots, c(p_{i_k}) \rangle$ is increasing. P' of size $\Omega(\sqrt{n})$ can be found in time $O(n \log \log n)$ [17, 25, 6]. Note that two conditions hold for any two elements $p_{i_j}, p_{i_{j'}} \in P'$ such that $i_j \leq i_{j'}$:

1. $w(p_{i_j}) \leq w(p_{i_{j'}})$ and
2. $c(p_{i_j}) \leq c(p_{i_{j'}})$.

LEMMA 3.3. *Given the subsequence* P' *described above, and an integer* n *such that* $n = W(F(r_1, a, r_3, b))$*, where the values of* r_1 *and* r_3 *are known, one can find the values of* a *and* b *in* $O(\log n)$ *time.*

PROOF. Note that in the previous algorithm, we used the binary search over one variable or used Frederickson and Johnson's technique to avoid the binary search, but we had to perform $\Omega(\sqrt{n})$ computations to find $r_2 = a$ and $r_4 = b$. Now we will show that if the subset of configuration space satisfies certain criteria, like P', then we can search in $O(\log n)$ steps.

We need to find a and b such that $W(F(r_1, a, r_3, b)) = n$ where $(a, b) \in P'$. Note that P' is increasing in both $W(F(0, a_i, 0, b_i))$ and in $C(P')$. Take any two pairs $p_{i_j} = (a_{i_j}, b_{i_j})$ and $p_{i_{j'}} = (a_{i_{j'}}, b_{i_{j'}})$ of P', $j \leq j'$. Then $a_{i_j} + b_{i_j} \leq a_{i_{j'}} + b_{i_{j'}}$ from 2. above, and for any positive r_1 and r_3, $8(r_1 - 0) + 3(r_3 - 0) \geq 0$. Thus the conditions of Lemma 3.2 are satisfied and

$$\begin{aligned} &W(F(0, a_{i_j}, 0, b_{i_j})) \leq W(F(0, a_{i_{j'}}, 0, b_{i_{j'}})) \implies \\ &\quad W(F(r_1, a_{i_j}, r_3, b_{i_j})) \leq W(F(r_1, a_{i_{j'}}, r_3, b_{i_{j'}})) \end{aligned}$$

$$w(a_{i_j}, b_{i_j}) \leq w(a_{i_{j'}}, b_{i_{j'}}) \implies \tag{3.4}$$
$$W(F(r_1, a_{i_j}, r_3, b_{i_j})) \leq W(F(r_1, a_{i_{j'}}, r_3, b_{i_{j'}}))$$

The last inequality in the displayed formula 3.4 says that the Wiener index of the 4-tuple $(r_1, a_{i_j}, r_3, b_{i_j})$ will always be less than the Wiener index of $(r_1, a_{i_{j'}}, r_3, b_{i_{j'}})$ if $j \leq j'$, *regardless* of the value of r_1 and r_3. Therefore we can do binary search since we have P' sorted by $W(F(0, a_i, 0, b_i))$ already. We only know that the order of the pairs is preserved, although the values of $w(\cdot)$ have changed (since the values of r_1 and r_3 have changed). At each step of the binary search we have to recompute $w(\cdot)$ for each pair, and proceed accordingly. ◻

Now the algorithm can be completed. Set $r_1 = r_3 = 0$, and compute the set P in time $O(n \log n)$ by sorting all tuples $(0, r_2, 0, r_4)$ by their Wiener indices. Now find the largest increasing subsequence of P in the ordered sequence $C(P)$. This subsequence satisfies the two properties, i.e. increasing with respect to $W(P)$ and $C(P)$. From Lemma 3.2, the order of this subsequence would remain unchanged with varying values of r_1 and r_3. We store this subsequence as an ordered sequence of pairs P_1. Now iteratively extract largest increasing subsequence P_i in round i till the last round k. Store these ordered sets $\mathcal{P} = \{P_1, \ldots, P_k\}$. Now, we vary the values of r_1 and r_3 from 1 to $\sqrt{n}$. Since we don't know which sequence could contain n, we have to search in all k sequences. In each sequence we do binary search as in Lemma 3.3, achieving the worst case total time $O(nk \log n)$. Note that $k \leq \sqrt{n}$ — we can always define $\mathcal{P} = \{P_1, \ldots, P_{\sqrt{n}}\}$, where $P_i = \{(i, 1), \ldots, (i, \sqrt{n})\}$. Then, from the monotonicity of the function $F(\cdot)$, each P_i is an increasing sequence in the Wiener index.

Two natural questions arise: how practical are these schemes, and is further speedup possible. In the next subsection, we empirically show that this sequence decomposition results in a practially near-linear time algorithm for our problem. Second, the $O(\log n)$ overhead of search in $\mathcal{P}$ can be removed, and we now improve the running time of the algorithm `SequenceDecomposition` to $O(nk)$. Recall that the sets $\mathcal{P} = \{P_1, \ldots, P_k\}$ were defined with $r_1 = r_3 = 0$. Lets define $\mathcal{Q}(c_1, c_3) = \{Q_1(c_1, c_3), \ldots, Q_k(c_1, c_3)\}$, where $Q_i(c_1, c_3)$ is an ordered sequence of integers, $\langle W(F(c_1, a_{i_j}, c_3, b_{i_j}))|(a_{i_j}, b_{i_j}) \in P_i\rangle$. Essentially, the previous algorithm performed a binary search over each set $Q_i(j, l)$, for all $1 \leq i \leq k$, $1 \leq j, l \leq \sqrt{n}$, thus getting total time $O(nk \log n)$. Now, instead we will do searches simultaneously over subsets of $\mathcal{Q}(c_1, c_3)$. Let $\mathcal{Q}_i(c_1) = \{Q_i(c_1, 0), \ldots, Q_i(c_1, \sqrt{n})\}$ be a subset of $\mathcal{Q}(c_1, c_3)$ of size $|\mathcal{Q}_i(c_1)| = \sqrt{n}$. Assume we can search over $\mathcal{Q}_i(c_1)$ in time $T(|\mathcal{Q}_i|)$. Then, the total running time becomes

$$\sum_{i=0}^{k} \sum_{c_1=0}^{\sqrt{n}} T(|\mathcal{Q}_i|) = \sqrt{n} \sum_{i=0}^{k} T(|\mathcal{Q}_i|)$$

Now we need to show how to search over $\mathcal{Q}_i(c_1)$ efficiently. To this end, form the matrix M where each row of M, denoted by M_j is a set, i.e. M is the matrix such that $M_j = Q_i(c_1, j)$, where i and c_1 are fixed. M is a $\sqrt{n} \times |Q_i|$ matrix.

LEMMA 3.4. *If the matrix M is associated with the set $\mathcal{Q}_i(c_1)$, M is a sorted matrix.*

The proof follows from the fact that $F(\cdot)$ is monotone in all its parameters. From Lemma 3.4, it follows that we can apply the Frederickson and Johnson technique to search over the set $\mathcal{Q}_i(c_1)$ in time $T(|\mathcal{Q}_i|) = \min\{\sqrt{n}, |Q_i|\} = O(\sqrt{n})$, similar to the description in the previous subsection. Thus, the total running time becomes $O(\sqrt{n} \cdot \sqrt{n}k) = O(nk)$.

THEOREM 3.2. *Given an integer n, one can find (if they exist) integers c_1, c_2, c_3 and c_4 such that $W(F(c_1, c_2, c_3, c_4)) = n$ in time $O(nk + n \log\log n)$, where k is a decomposition parameter.*

3.2 Experimental value of k

It is worthwhile to step back and survey what we have accomplished by the above techniques. The above algorithm can be thought of as an "output-sensitive" algorithm — it has worst case running time of $O(nk)$. According to the discussion above, the first largest increasing sequence that we extract has size at least $\sqrt{n}$. However, empirically we find that the largest increasing sequence has size much larger than $O(\sqrt{n})$ — leading to a nearly $O(n \log\log n)$ algorithm. This is verified empirically below, by comparing this algorithm, denoted as Algorithm `SequenceDecomposition` with the binary search algorithm, denoted as Algorithm `BinarySearch`.

We first run the above two algorithms for the inverse Wiener problem for Wiener indices $n = 1000, 2000, \ldots, 140,000$. For each n, we measure the the running time by counting the number of calls to $W(F(\cdot))$ of both Algorithm `BinarySearch`, and Algorithm `SequenceDecomposition`. The results are shown in Figure 4 (a). As expected, the running time for various integers n varies quite a lot (since some searches are lucky to find the number easily) but on the whole it can be easily seen that `BinarySearch` makes many more calls to $W(F(\cdot))$ than `SequenceDecomposition`. We "smooth" out the spikes in the figure to get a better understanding of the running times of the algorithms by summing the running times over discrete intervals. We run the algorithms over all the numbers in the interval $[1000 \cdot i, 1000 \cdot i + 400]$, where $i = 1, 2, \ldots, 200$. Figure 4 (b) shows the resulting graph, where the value of the curve at $x = i$ shows the sum of the running times for the integers in $[1000 \cdot i, 1000 \cdot i + 400]$. The many orders of magnitude in speed is obvious in this figure.

4 Algorithm for the problem $P_{\leq}$

The problem $P_{\leq}$ for our class of trees $F(\cdot)$ is as follows: given an integer n, for every integer $0 \leq m \leq n$, compute the configuration (a_1, a_2, a_3, a_4) such that $W(F(a_1, a_2, a_3, a_4)) = m$.

Of course, problem $P_{\leq}$can be solved by n calls to problem $P_{=}$, yielding a $O(n^2 k)$ time algorithm, and with the worst case running time of $O(n^{5/2})$. However, the problem $P_{\leq}$can be solved in $O(n^2)$ time by computing $W(F(\cdot))$ for every tuple $a = (a_1, a_2, a_3, a_4)$, where $0 \leq a_i \leq \sqrt{n}$, $i = 1, \ldots, 4$. For each tuple, we mark off the integer $W(F(\cdot))$ in an array. After all tuples have been computed, the locations in the array not marked indicate integers not representable. As before, we will measure the running time

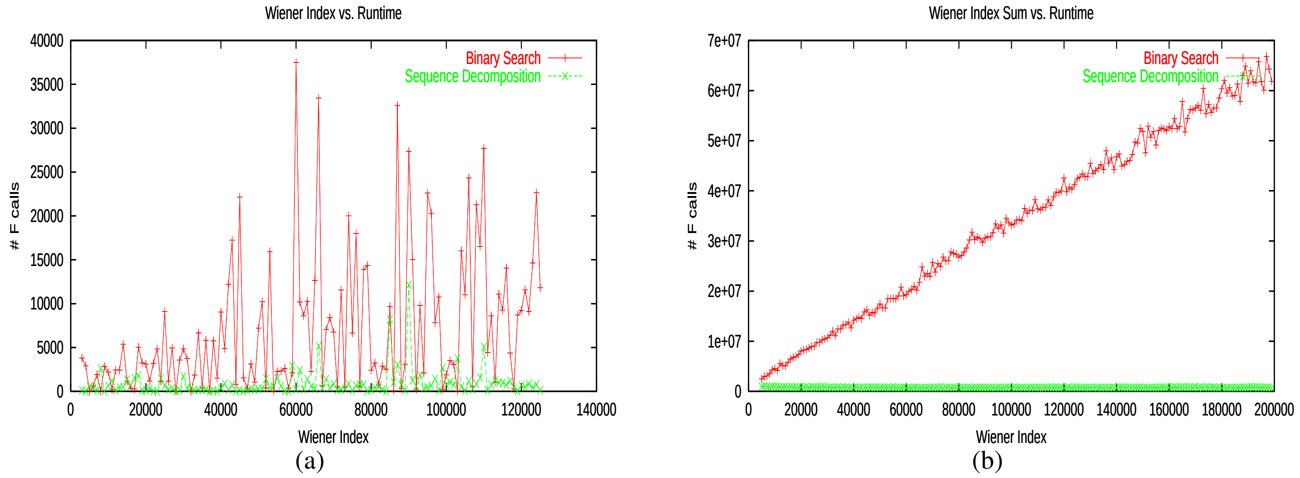

Figure 4: Figure showing the running times of the algorithms, measured by the number of calls to $F(\cdot)$

of the algorithm by the number of calls to the function. Using equation 2.3 we can bound $a_1 \le \sqrt{(n-20)/8}$, $a_i \le \sqrt{n-20}, i = 2,3,4$. Therefore the complexity of the naive algorithm is bounded by $\lfloor\sqrt{(n-20)/8}\rfloor \cdot \lfloor\sqrt{n-20}\rfloor^3$. For large n, this is approximately $n^2/(2\sqrt{2})$ and we denote it as $T_{naive}(n)$. Our goal is to design an algorithm that solves $P_{\le}$using a substantially smaller number of computations of $W(F(\cdot))$. The first idea is to make a bound for a that further restricts the search space. We will call our new algorithm as *push algorithm*. The algorithm will sequentially try to cover integers in increasing order. Let

$$s(a) = \lceil\sqrt{2}(2a_1 + a_2 + a_3 + a_4 + 7/2)\rceil$$

and let m be the smallest number whose expression $W(F(\cdot)) = m$ is not computed yet. The value of s can be bounded from below.

LEMMA 4.1. *If* $W(F(a)) = m$ *then* $s(a) > \sqrt{m}$.

PROOF. The lemma follows from the fact that $W(F(a)) < 2(2a_1 + a_2 + a_3 + a_4 + 7/2)^2$. We prove the inequality

$$\begin{aligned}
- \quad & 2(2a_1 + a_2 + a_3 + a_4)^2 - W(F(a)) \ge 0 \\
- \quad & a_1(8a_1 + 8a_2 + 8a_3 + 8a_4 + 28) \\
& + a_2(4a_2 + 4a_3 + 4a_4 + 14) \\
& + a_3(4a_3 + 4a_4 + 14) \\
& + a_4(4a_4 + 14) + 49/2 - W(F(a)) \ge 0 \\
- \quad & a_2(3a_2 + a_3 + 3) + a_3(3a_3 + a_4 + 4) \\
& + a_4(3a_4 + 3) + 9/2 > 0.
\end{aligned}$$

$\sqcup\!\!\!\sqcap$

The algorithm searches for the solution of $W(F(a)) = m, m \le n$ in increasing order of $s(a)$. Let M be the current value of $s(a)$. The algorithm enumerates all tuples a so that $s(a) = M$. Let m_0 be the smallest number not representable as $W(F())$ [1]. By Lemma 4.1, if there is solution for $W(F(a)) = m_0$ then M must be greater than $\sqrt{m_0}$. If current value of M is at most $\sqrt{m_0}$, we increase M to $\lfloor\sqrt{m_0}\rfloor$. We implemented the push algorithm and tested it for $n = 10^6$. Let $T_{push}(n)$ be the number of computations of $W(F(\cdot))$. The number of computations T_{push} is essentially quadratic, see Figure 5. However, notice that the push algorithm demonstrates a speedup factor of 42 versus T_{naive}.

The second algorithm we implemented, *sweep algorithm*, sweeps tuples a according to the increasing sum $a_1 + a_2 + a_3 + a_4$. The sweep algorithm runs faster than the push algorithm. The ratio $T_{naive}(n)/T_{sweep}(n)$ is approximately 66, as illustrated in Figure 5(b).

We implemented another algorithm that we call *jump algorithm*. The idea is to sweep tuples a lexicographically and skip tuples that do not produce new numbers. We maintain the smallest number m_0 whose representation is not yet found. Suppose that $a_1 = c_1, a_2 = c_2$ and $a_3 = c_3$ are fixed constants and the algorithm starts search for a_4. Let f denote the function $f(a_4) = W(F(c_1, c_2, c_3, a_4))$. f is monotone and the equation $f(a_4) = m_0$ is quadratic. By solving this quadratic equation, we can find the largest value a_4^* so that $f(a_4^*)$ is at most m_0. Then a_4 takes values from a_4^* to $\sqrt{n}$. The experiments show that

1. for $n < 10^5$, the sweep algorithm is faster than others, and

[1] Note that m_0 must be greater than 1177 since 1177 is not representable as $W(F())$.

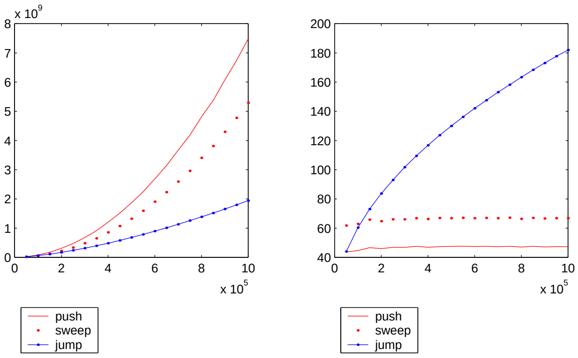

Figure 5: a) The graphs of T_{push}, T_{sweep}, T_{jump}, and b) the ratios T_{naive} versus the functions.

2. for $n \geq 10^5$, the jump algorithm is faster than others, and

3. the speedup of the jump algorithm with respect to the naive algorithm grows with n.

We did a nonlinear fit of the data using the Levenberg-Marquardt algorithm and the running time of the jump algorithm fits the polynomial equation $1.5841n^{1.51474}$. The asymptotic standard errors of the fit are 0.000655467 for the exponent and 0.0141145 for the coefficient, indicating an accurate fit.

The jump algorithm allows us to verify Conjecture 2.3 up to 10^8. The running time is 4.6 days on 360Mhz SGI MIPS R12K. We estimate that the sweep and naive algorithms would need 82.1 days and 14.1 years, respectively.

5 Conclusion

We studied the problems related to Wiener indices that have applications in computational chemistry and biology. We proposed a family of trees F such that their Wiener indices are well distributed, and we conjectured that they cover all the positive integers except a finite set of numbers. The conjecture is stronger than the conjecture by Goldman [19] made for general trees. We verified our conjecture for large numbers up to 10^8.

We designed several efficient algorithms for the problem of finding a tree whose Wiener index is a given number. The algorithm that guarantees to find a tree in F, if any, has running time $O(n^{1.5})$ and is based on sorted matrix searching technique. The algorithm exploits the properties of the Wiener function $W(F(a))$ and one can show that for general function $f(a) = \Theta(W(F(a)))$ a lower bound for solving $f(a) = n$ is actually $\Omega(n^2)$. The second algorithm is based on the longest increasing subsequences. Furthermore, we modified the algorithm using sorted matrices and improve the runtime by a factor of $O(\log n)$. The last algorithm runs 100 times faster than the first one for $n = 2 \cdot 10^5$.

We developed three algorithms for the problem of finding all Wiener indices less than a given number. The first two algorithms have essentially quadratic running time. The third algorithm runs faster and there is a strong evidence that the actual running time is sub-quadratic. This algorithm allows us to verify the main conjecture that we put forward in this paper for large integers, thereby strengthening our belief in an affirmative answer.

References

[1] Agrafiotis, D., and No, W. C. Feature selection for structure-activity correlation using binary particle swarms. *J. Med. Chem. 45* (2002), 1098–1107.

[2] Agrafiotis, D., No, W. C., and Lobanov, V. On the use of neural network ensembles in QSAR and QSPR. *J. Chem. Info. Comp. Sci.* (2002). To appear.

[3] Agrawal, V., Srivastava, R., and Khadikar, P. QSAR studies on some antimalarial sulfonamides. *Bioorganic and Medicinal Chemistry 9* (2001), 3287–3293.

[4] Besalu, E., Gallegos, A., and Carbo-Dorca, R. Topological quantum similarity indices and their use in QSAR: Application to several families of antimalarial compounds. *MATCH 44* (2001), 41–64.

[5] Besalu, E., Girones, X., Amat, L., and Carbo-Dorca, R. Molecular quantum similarity and the fundamentals of QSAR. *Accounts of Chemical Research* (2002). ACS ASAP.

[6] Bespamyatnikh, S., and Segal, M. Enumerating longest increasing subsequences and patience sorting. *Inform. Process. Lett. 76*, 1-2 (2000), 7–11.

[7] Bonchev, D., Markel, E., and Dekmezian, A. Topological analysis of long-chain branching patterns in polyolefins. *J. Chem. Inf. Comput. Sci. 41* (2001), 1274–1285.

[8] Bounaim, L., Villa, M., Smeyers, N., Senent, M., Ezzamarty, A., and Smeyers, Y. Ab initio structural study and qsar of certain ibuprofen derivatives as possible antiinflammatory agents. *Folia Chimica Theoretica Latina 28* (2000), 88–98.

[9] Buckley, F. Mean distance in line graphs. *Congr. Numer. 32* (1981), 153–162.

[10] Canfield, E. R., Robinson, R. W., and Rouvray, D. H. Determination of the wiener molecular branching index for the general tree. *J. Computational Chemistry 6* (1985), 598–609.

[11] Corwin, H., Kurup, A., Garg, R., and Gao, H. Chembioinfomatics and qsar: a review of qsar lacking positive hydrophobic terms. *Chemical Reviews 101* (2001), 619–672.

[12] Dankelmann, R. Computing the average distance of an interval graph. *Information Processing Letters 48* (1993), 311–314.

[13] Dobrynin, A. A., Entringer, R., and Gutman, I. Wiener index of trees: Theory and applications. *Acta Applicandae Mathematicae 66* (2001), 211–249.

[14] Entringer, R. C., Meir, A., Moon, J., and Székely, L. Wiener index of trees from certain families. *Australias. J. of Combinatorics 10* (1994), 211–224.

[15] Estrada, E., and Uriarte, E. Recent advances on the role of topological indices in drug discovery research. *Current Medicinal Chemistry 8* (2001), 1573–1588.

[16] Frederickson, G. N., and Johnson, D. B. Generalized selection and ranking: sorted matrices. *SIAM Journal of Computing 13*, 1 (1984), 14–30.

[17] Fredman, M. On computing the length of the longest increasing subsequence. *Discrete Math 11* (1975), 29–35.

[18] Golbraikh, A., Bonchev, D., Xiao, Y.-D., and Tropsha, A. Novel chiral topological descriptors and their applications to QSAR. In *Proc. of the European Sympos. on Quant. Structure-Activity Relationships, 13th, Duesseldorf, Germany, Aug. 27 - Sept. 1, 2000* (2001), p. 219223.

[19] Goldman, D., Istrail, S., Lancia, G., and Piccolboni, A. Algorithmic strategies in combinatorial chemistry. In *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms* (2000), pp. 275–284.

[20] Gozalbes, R., Doucet, J., and Derouin, F. Application of topological descriptors in QSAR and drug design: history and new trends. *Current Drug Targets: Infectious Disorders 2* (2002), 93–102.

[21] Gutman, I., and Polansky, O. E. *Mathematical concepts in organic chemistry*. Springer-Verlag, Berlin, 1986.

[22] Gutman, I., and Potgieter, J. J. Wiener index and intermolecular forces. *J. Serb. Checm. Soc. 62* (1997), 185–192.

[23] Gutman, I., and Yeh, Y. The sum of all distances in bipartite graphs. *Math. Slovaca 45* (1995), 327–334.

[24] Hoffman, B., Cho, S., Zheng, W., Wyrick, S., Nichols, D., Mailman, R., and Tropsha, A. A QSAR modeling of dopamine D1 agonists using comparative molecular field analysis, genetic algorithms-partial least squares, and k nearest neighbor methods. *J. Med. Chem. 42* (1999), 317–3226.

[25] Hunt, J., and Szymanski, T. A fast algorithm for computing longest common subsequences. *Communications of ACM 20* (1977), 350–353.

[26] Ireland, K. F., and Rosen, M. I. *A classical introduction to modern number theory*. Springer-Verlag, New York, 1990.

[27] Ivanciuc, O. QSAR comparative study of Wiener descriptor for weighted molecular graphs. *J. Chem. Inf. Compuc. Sci. 40* (2000), 1412–1422.

[28] Johnson, D. B. Efficient algorithms for shortest path in sparse networks. *Journal of ACM 24* (1977), 1–13.

[29] Krenkel, G., Castro, E., and Toropov, A. A. Improved molecular description based on the optimization of correlation weights of local graph invariants. *Int. J. of Mol. Sci. 2* (2001), 57–65.

[30] Lepović, M., and Gutman, I. A collective property of trees and chemical trees. *J. Chem. Inf. Comput. Sci. 38* (1998), 823–826.

[31] Martin, Y. C. 3D QSAR. current state, scope, and limitations. *Perspect. Drug Discovery Des. 12* (1998), 3–32.

[32] Merris, R. An edge version of the matrix-tree theorem and the wiener index. *Linear and Multilinear Algebra 25* (1989), 291–296.

[33] Mohar, B., and Pisanski, T. How to compute the wiener index of a graph. *Journal of Mathematical Chemistry 2* (1988), 267–277.

[34] Nair, A., Jayatilleke, P., Wang, X., Miertus, S., and Welsh, W. Computational studies on tetrahydropyrimidine-2-one HIV-1 protease inhibitors: improving three-dimensional quantitative structure-activity relationship comparative molecular field analysis models by inclusion of calculated inhibitor and receptor-based propertes. *J. Med. Chem. 45* (2002), 973–983.

[35] Ochoa, C., Chana, A., and Stud, M. Applications of neural networks in the medicinal chemistry field. *Current Medicinal Chemistry: Central Nervous System Agents 1* (2001), 247–256.

[36] Pologar, B., and Ferguson, D. QSAR and CoMFA: a perspective on the pratical application to drug discoverty. *Drug Design and Discovery 17* (2000), 4–12.

[37] Pungpo, P., Wolschann, P., and Hannongbua, S. Quantitative structure-activity relationships of HIV-1 reverse transcriptase inhibitors, using hologram QSAR. In *Proc. of the European Sympos. on Quant. Structure-Activity Relationships, 13th, Duesseldorf, Germany, Aug. 27 - Sept. 1, 2000* (2001), pp. 206–210.

[38] Rouvray, D. H. *Should we have designs on topological indices?* Elsevier, Amsterdam, 1983, pp. 159–177.

[39] Rücker, G., and Rücker, C. On topological indices, boiling points, and cycloalkanes. *J. Chem. Inf. Comput. Sci. 39* (1999), 788–802.

[40] Tehan, B., Lloyd, E., and Wong, M. Atypical antipsychotics: Modelling, QSAR and database searching. In *Proc. of the European Sympos. on Quant. Structure-Activity Relationships, 13th, Duesseldorf, Germany, Aug. 27 - Sept. 1, 2000* (2001), pp. 345–348.

[41] Tetko, I., Kovalishyn, V., Luik, A., and Livingston, D. Application of volume learning artificial neural network to calculate 3D QSAR models with enhanced predictive properties. In *Proc. of the European Sympos. on Quant. Structure-Activity Relationships, 13th, Duesseldorf, Germany, Aug. 27 - Sept. 1, 2000* (2001), pp. 229–234.

[42] Wiener, H. Structural determination of paraffin boiling points. *J. Amer. Chem. Soc. 69* (1947), 17–20.

[43] Zheng, W., and Tropsha, A. A novel variable selection QSAR approach based on the k-nearest neighbor principle. *J. Chem Inf. Comp. Sci 40* (2000), 167–177.

[44] Zhu, L., Hou, T., and Xu, X. Three-dimensional quantitative structure-activity relationship study on paullones as CDK inhibitors using CoMSIA and CoMFA. *J. Mol. Model 7* (2001), 223–230.

A Appendix: Weiner Indices

S_1	2, 3, 5, 6, 7, 8, 11, 12, 13, 14, 15, 17, 19, 21, 22, 23, 24, 26, 27, 30, 33, 34, 37, 38, 39, 41, 43, 45, 47, 51, 53, 55, 60, 61, 69, 72, 73, 77, 78, 83, 85, 87, 89, 91, 99, 101, 106, 113, 117, 129, 133, 147, 157, 159, 173, 193
S_2	1, 3, 5, 6, 7, 8, 11, 12, 13, 14, 15, 17, 19, 21, 22, 23, 24, 26, 27, 30, 33, 34, 37, 38, 39, 41, 43, 45, 47, 51, 53, 55 56, 60, 61, 69, 72, 73, 75, 77, 78, 79, 83, 85, 87, ,89, 91, 99, 101, 102, 106, 113, 117, 125 129, 131, 133, 135, 141, 143, 147, 149, 157, 159, 165, 173, 181, 193,195, 197, 199, 203, 213, 217, 219, 229, 246, 261, 277, 279, 281, 285, 293, 301, 309, 325, 333, 341, 357, 369, 373, 375, 389, 421, 453, 469, 485, 493, 501, 509, 517, 533, 549, 557, 565, 581, 617, 629, 677, 693, 725, 741, 757, 773, 781, 789, 805, 837, 869, 901, 933, 965, 997, 1029, 1045, 1061, 1093, 1101, 1109, 1173, 1189, 1237, 1317, 1381, 1413, 1445, 1477, 1493, 1509, 1541, 1573, 1605, 1637, 1669, 1701, 1733, 1845, 1861, 1893, 1909, 1973, 2085, 2101, 2149, 2277 2293, 2341, 2373, 2405, 2437, 2469, 2501, 2597, 2629, 2693, 2725, 2853, 2885, 2981, 3029, 3109, 3237, 3365, 3493, 3525, 3557, 3589, 3653, 3813, 3877, 3909, 4069, 4389, 4581, 4645, 4773, 4901, 4933, 4965, 5029, 5157, 5285, 5413, 5541, 5669, 5797, 6309, 6437, 6501, 6533, 6757, 7205, 7461, 7973, 8229, 8357, 8485, 8613, 8997, 9125, 9381, 9509, 10021, 10149, 10533, 11045, 11557, 12325, 12453, 12581, 12709, 12965, 13605, 13861, 13989, 14629, 15717, 16677, 17701, 17829, 17957, 18213, 18725, 19237, 19749, 20773, 21285, 21797, 22821, 23589, 23845, 23973, 24869, 26917, 28965, 29989, 31013, 31525, 32037, 32549, 33061, 34085, 35109, 35621, 36133, 37157, 39461, 40229, 43301, 44325, 46373, 47397, 48421, 49445, 49957, 50469, 51493, 53541, 55589, 57637, 59685, 61733, 63781, 65829, 66853, 67877, 69925, 70437, 70949, 75045, 76069, 79141, 84261, 88357, 90405, 92453, 94501, 95525, 96549, 98597, 100645, 102693, 104741, 106789, 108837, 110885, 118053, 119077, 121125, 122149, 126245, 133413, 134437, 137509, 145701, 146725 149797, 151845, 153893, 155941, 157989, 160037, 166181, 168229, 172325, 174373, 182565, 184613, 190757, 193829, 198949, 207141, 215333, 223525, 225573, 227621, 229669, 233765, 244005, 248101, 250149, 260389, 280869, 293157, 297253, 305445, 313637, 315685, 317733, 321829, 330021, 3382 416037, 418085, 432421, 461093, 477477, 510245, 526629, 534821, 543013, 551205, 575781, 583973, 600357, 608549, 641317, 649509, 674085, 706853, 739621, 788773, 796965, 805157, 813349, 829733, 870693, 887077, 895269, 936229, 985381
S_3	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 33, 34, 36, 37, 38, 39, 40, 41, 42, 43, 45, 47, 49, 51, 53, 54, 55, 57, 58, 60, 61, 64, 69, 72, 73, 77, 78, 81, 83, 85, 87, 89, 91, 93, 97, 99, 101, 106, 113, 114, 117, 129, 133, 137, 141, 143, 145, 147, 149, 157, 159, 165, 173, 189, 193, 205, 213, 217, 219, 229, 249, 265, 281, 285, 301, 309, 325, 357, 373, 389, 417, 433, 557
S_4	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 45, 47, 49, 50, 51, 52, 53, 54, 55, 57, 58, 60, 61, 64, 67, 68, 69, 70, 71, 72, 73, 74, 77, 78, 79, 81, 83, 85, 87, 89, 90, 91, 92, 93, 94, 97, 99, 101, 102, 106, 107, 110, 113, 114, 115, 117, 118, 120, 121, 124, 127, 129, 130, 131, 133, 137, 141, 142, 143, 145, 147, 149, 157, 159, 160, 165, 173, 174, 175, 177, 183, 189, 193, 194, 197, 203, 205, 208, 213, 214, 217, 219, 226, 227, 229, 235, 241, 242, 249, 257, 265, 267, 269, 270, 275, 281, 285, 288, 295, 301, 309, 311, 325, 327, 330, 335, 337, 349, 357, 373, 389, 393, 403, 405, 417, 419, 433, 435, 461, 467, 481, 484, 501, 527, 529, 533, 545, 557, 565, 575, 613, 657, 701, 729, 747, 757, 837, 857, 935, 1061, 1177

Table 1: Table presenting the various sets of integers related to the conjectures.