Datalogisk Institut
KØBENHAVNS UNIVERSITET

Sum of All- Pairs Shortest Path Distances in a Planar Graph in Subquadratic time

Christian Wulff - Nilsen

Technical Report no. 08-11
ISSN: 0107-8283

Dept. of Computer Science
University of Copenhagen • Universitetsparken 1
DK-2100 Copenhagen • Denmark

Sum of All-Pairs Shortest Path Distances in a Planar Graph in Subquadratic Time

Christian Wulff-Nilsen *

September 24, 2008

Abstract

We consider the problem of computing the Wiener index of a graph, defined as the sum of distances between all pairs of its vertices. It is an open problem whether the Wiener index of a planar graph can be found in subquadratic time. We solve this problem by presenting an algorithm with $O(n^2 \log\log n/\log n)$ running time and $O(n)$ space requirement where n is the number of vertices of the graph.

1 Introduction

A *molecular topological index* is a value obtained from the graph structure of a molecule such that this value (hopefully) correlates with physical and/or chemical properties of the molecule. Perhaps the most studied molecular topological index is the so called *Wiener index*, a generalization of a definition given by Wiener in 1947 [7]. The Wiener index of an unweighted graph is defined as the sum of distances between all pairs of vertices of the graph.

Computing the Wiener index of a graph can clearly be done in the amount of time it takes to compute APSP distances for the graph. For special types of graphs, faster algorithms are known. Linear time algorithms are known for cactii [8] and benzenoid systems [2] and recently Cabello and Knauer [1] gave near-linear time algorithms for graphs of bounded treewidth. More specifically, they showed that the Wiener index of an n-vertex graph of treewidth

*Department of Computer Science, University of Copenhagen, koolooz@diku.dk, http://www.diku.dk/~koolooz/

$k \geq 3$ can be found in $O(n \log^{k-1} n)$ time. All these bounds also hold for weighted graphs.

For planar graphs, the Wiener index can be found in quadratic time using the algorithm of Frederickson [3]. One of the main open problems in this context is the existence of a subquadratic time algorithm for such graphs.

In this paper, we solve this open problem by giving an $O(n^2 \log\log n / \log n)$ time and $O(n)$ space algorithm where n is the number of vertices.

The organization of the paper is as follows. In Section 2, we give various definitions and introduce some notation. We mention a result by Frederickson [3] in Section 3, a result that allows us to divide a planar graph into regions with some nice properties. In Section 4, we rely on this result to obtain our subquadratic time algorithm for computing the Wiener index of a planar graph. We show how to obtain linear space requirement in Section 5 and finally, we make some concluding remarks in Section 6.

2 Definitions and Notation

Let $G = (V, E)$ be an unweighted graph. For $u, v \in V$, we let $d_G(u, v)$ denote the length of a shortest path in G between u and v.

Given a subgraph H of G, we let V_H denote its vertex set.

Given subsets $U_1, U_2 \subseteq V$, we define

$$\sum(U_1, U_2) = \sum_{u \in U_1} \sum_{v \in U_2} d_G(u, v).$$

We omit G in the notation but this should not cause any confusion. For a vertex u and a subset U of V, we write $\sum(U, u)$ as a shorthand for $\sum(U, \{u\})$.

We let $\sum G$ denote the sum of all shortest path distances in G, i.e., $\sum G = \frac{1}{2}\sum(V, V)$ and we refer to it as the *Wiener index* of G.

A *region* of G is a subset R of vertices of G. A *boundary vertex* of R is a vertex in R which an edge of E connects to a vertex in $V \setminus R$. Vertices of R that are not boundary vertices are called *interior vertices* (of R).

3 r-division of a Planar Graph

Divide-and-conquer is an important paradigm which is the basis of efficient algorithms for all kinds of algorithmic problems. The celebrated separator

theorem of Lipton and Tarjan [5] makes the use of divide-and-conquer on planar graphs possible and has consequently lead to faster algorithms for problems related to such graphs.

The separator theorem states that, given an n-vertex planar graph $G = (V, E)$ with nonnegative vertex costs summing to no more than one, V can be partitioned into three subsets A, B, and C such that

1. no edge of E joins a vertex in A with a vertex in B,
2. neither A nor B has total cost exceeding $2/3$, and
3. C contains no more than $2\sqrt{2}\sqrt{n}$ vertices.

By applying the separator theorem recursively to a given planar graph, Frederickson [3] obtained the following result which we state as a lemma.

Lemma 1. *Given a parameter r (which may depend on n), an n-vertex planar graph can be divided into $\Theta(n/r)$ regions each of which contains at most r vertices and $O(\sqrt{r})$ boundary vertices. Furthermore, each interior vertex is contained in exactly one region. Finding such a division can be done in $O(n \log n)$ time.*

For parameter r, we refer to the division in Lemma 1 as an *r-division* (of the graph). If $R_1, \ldots, R_k \subseteq V$ are the regions obtained, we denote the r-division by the tuple $(R_1, \ldots, R_k)$.

Finding an r-division for a suitable value of r is an important part of our algorithm to compute the Wiener index of a planar graph.

4 Wiener Index of a Planar Graph

In the following, let $G = (V, E)$ be an unweighted planar graph with n vertices. In this section, we show how to compute the Wiener index $\sum G$ of G in $O(n^2 \log\log n / \log n)$ time. We will assume that G is connected since otherwise the problem is trivial.

The first step of our algorithm is to compute an r-division $(R_1, \ldots, R_k)$ of G for some parameter r which we specify later. For now, just regard r as some function of n. We will show how to compute $\sum G$ in $O(n^2/\sqrt{r} + nr^{O(\sqrt{r})})$ time. From this and from a suitable choice of r, the main result of the paper will follow.

In the following, let B be the set of boundary vertices over all regions $R_1, \ldots, R_k$. We precompute shortest path distances from each vertex in B to all vertices in V. Since $|B| = O(n/\sqrt{r})$, this can be done in $O(n^2/\sqrt{r})$ time using the linear time SSSP algorithm in [4] for each vertex in B. From these distances, we obtain values $\sum(B, V)$ and $\sum(B, B)$ in $O(n^2/\sqrt{r})$ time.

Observe that $\sum(B, V) - \frac{1}{2}\sum(B, B)$ is the sum of all shortest path distances in G between vertex pairs (u, v) for which either u or v (or both) is a boundary vertex. Since, by Lemma 1, each interior vertex belongs to exactly one region, we can thus obtain $\sum G$ as the sum

$$\sum(B, V) - \frac{1}{2}\sum(B, B) + \frac{1}{2}\sum_{i=1}^{k}\sum(R_i \setminus B, V \setminus (R_i \cup B)) + \sum(R_i \setminus B, R_i \setminus B).$$

Let R be one of the regions $R_1, \ldots, R_k$. In the following, we focus on the problem of computing $\sum(R \setminus B, V \setminus (R \cup B))$ and $\sum(R \setminus B, R \setminus B)$. If we can show that these two quantities can be computed in $O(n\sqrt{r} + r^{O(\sqrt{r})})$ time, it will follow that $\sum G$ can be computed in $O(n^2/\sqrt{r} + nr^{O(\sqrt{r})})$ time since $k = \Theta(n/r)$.

Let us start with the easy part, that of computing $\sum(R \setminus B, R \setminus B)$. We do this by computing shortest path distances in G between each pair of vertices in R. To do this efficiently, we take the subgraph of G induced by R and add to it an edge between each pair of boundary vertices of R; the length of this edge is equal to the distance in G between those two vertices (we do not add an edge between a pair of boundary vertices already connected by an edge). We then run an APSP algorithm like Floyd-Warshall on the resulting graph.

Since shortest path distances from boundary vertices of R to all vertices in G (and in particular to all boundary vertices in R) have been precomputed, it follows that we can compute shortest path distances in G between each pair of vertices in R in $O(r^3)$ time. Hence, we can compute $\sum(R \setminus B, R \setminus B)$ in $O(r^3)$ time.

Now, to compute $\sum(R \setminus B, V \setminus (R \cup B))$, let $C_1, \ldots, C_s$ be the connected components of the subgraph of G induced by R. Then

$$\sum(R \setminus B, V \setminus (R \cup B)) = \sum_{i=1}^{s}\sum(V_{C_i} \setminus B, V \setminus (R \cup B)). \tag{1}$$

Let C be one of these connected components, let $n_C = |V_C|$, and let $p_1, \ldots, p_t$ be the boundary vertices of R belonging to C. In the following, we

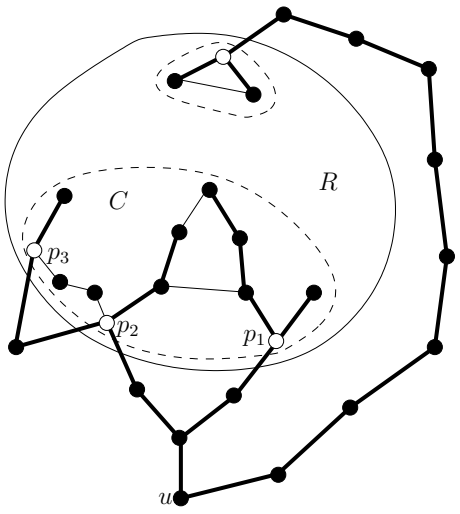

Figure 1: Graph instance for which component C has $t = 3$ boundary vertices p_1, p_2, and p_3 of R and where $(n_{1,u}, D_{1,u}) = (4, 7)$, $(n_{2,u}, D_{2,u}) = (2, 3)$, and $(n_{3,u}, D_{3,u}) = (1, 1)$. Also $\sum(V_C \setminus B, u) = \sum_{i=1}^{t} n_{i,u} d_G(u, p_i) + D_{i,u} = 34$.

show how to compute $\sum(V_C \setminus B, V \setminus (R \cup B))$ in $O(nt + n_C^{O(t)})$ time. It will then follow that the left-hand side of (1) can be computed in $O(n\sqrt{r} + r^{O(\sqrt{r})})$ time since each of the $O(\sqrt{r})$ boundary vertices of R belongs to exactly one connected component.

To compute $\sum(V_C \setminus B, V \setminus (R \cup B))$, the basic idea is the following. Given some vertex $u \in V \setminus (R \cup B)$, suppose we have precomputed, for each boundary vertex p_i, $i = 1, \ldots, t$,

1. the number $n_{i,u}$ of vertices v in $V_C \setminus B$ for which i is the smallest j such that $d_G(u, v) = d_G(u, p_j) + d_C(p_j, v)$,

2. the sum $D_{i,u}$ of distances in C from p_i to each of these vertices in $V_C \setminus B$.

Then

$$\sum(V_C \setminus B, u) = \sum_{i=1}^{t} n_{i,u} d_G(u, p_i) + D_{i,u},$$

see Figure 1.

Given these precomputations, we can thus obtain $\sum(V_C \setminus B, u)$ in $O(t)$ time and from this it follows that $\sum(V_C \setminus B, V \setminus (R \cup B))$ can be computed in $O(nt)$ time.

In order to perform the above precomputations efficiently we need the following key observation.

Lemma 2. *Let $u \in V \setminus (R \cup B)$ and let $i \in \{1, \ldots, t\}$ be given. Then $n_{i,u}$ and $D_{i,u}$ are completely determined by shortest path distances in C and values $d_G(u, p_j) - d_G(u, p_1)$ for $j = 1, \ldots, t$.*

Proof. Let $v \in V_C \setminus B$. Any path from u to v in G must contain at least one of the boundary vertices $p_1, \ldots, p_t$. Hence, the following two conditions are equivalent:

1. i is the smallest j such that $d_G(u, v) = d_G(u, p_j) + d_C(p_j, v)$,
2. $d_G(u, p_i) - d_G(u, p_j) \leq d_C(v, p_j) - d_C(v, p_i)$ holds for $1 \leq j \leq t$ with strict inequality for $1 \leq j < i$.

Since $d_G(u, p_i) - d_G(u, p_j) = (d_G(u, p_i) - d_G(u, p_1)) - (d_G(u, p_j) - d_G(u, p_1))$, the above shows that $n_{i,u}$ depends only on shortest path distances in C and values $d_G(u, p_j) - d_G(u, p_1)$, $j = 1, \ldots, t$. Clearly, this also holds for $D_{i,u}$. □

Before proceeding, let us define a map $\phi : V \setminus (R \cup B) \to \mathbb{Z}^t$ by

$$\phi(u)[j] = d_G(u, p_j) - d_G(u, p_1),$$

for $j = 1, \ldots, t$. Let p be a point in $\phi(V \setminus (R \cup B))$ and let u be a vertex in $V \setminus (R \cup B)$ such that $\phi(u) = p$. Associate with p values $n_p(i)$ and $D_p(i)$ for $i = 1, \ldots, t$, defined by

$$n_p(i) = n_{i,u},$$
$$D_p(i) = D_{i,u}.$$

By Lemma 2, this is well-defined since $n_p(i)$ and $D_p(i)$ do not depend on the choice of $u \in \phi^{-1}(\{p\})$.

The strategy now is to precompute n_p- and D_p-values for each $p \in \phi(V \setminus (R \cup B))$ and then, for each $u \in V \setminus (R \cup B)$, compute $p = \phi(u)$ and obtain, for $i = 1, \ldots, t$, $n_{u,i}$ and $D_{u,i}$ as the precomputed values $n_p(i)$ and $D_p(i)$, respectively. Function ϕ will act in a way similar to a hash function in that it maps a key (a vertex u) into a hash (the point $p = \phi(u)$) to obtain a value ($n_p(i)$ and $D_p(i)$ for $i = 1, \ldots, t$).

For this strategy to work well, we need the following lemma which shows that the number of points in $\phi(V \setminus (R \cup B))$ is small compared to $V \setminus (R \cup B)$ and hence that we only need to compute a small number of n_p- and D_p-values.

Lemma 3. $\phi(V \setminus (R \cup B)) \subseteq \{-(n_C - 1), \ldots, n_C - 1\}^t$.

Proof. Let $u \in V \setminus (R \cup B)$ and let $j \in \{1, \ldots, t\}$ be given. Since C is connected there is a simple path in C from p_1 to p_j and this path consists of at most $n_C - 1$ edges. Thus, $d_G(p_1, p_j) \leq d_C(p_1, p_j) \leq n_C - 1$ and the triangle inequality implies

$$-(n_C - 1) \leq d_G(u, p_j) - d_G(u, p_1) \leq n_C - 1.$$

This shows the lemma since $\phi(u)[j] = d_G(u, p_j) - d_G(u, p_1)$. □

We are now ready to describe how to compute $\sum(V_C \setminus B, V \setminus (R \cup B))$. We first initialize a t-dimensional table T with an entry $T[p]$ for each point $p \in \{-(n_C - 1), \ldots, n_C - 1\}^t$. Associated with $T[p]$ are two t-dimensional vectors to hold values $(n_p(1), \ldots, n_p(t))$ and $(D_p(1), \ldots, D_p(t))$. Initially, all entries of T are unmarked. The initialization step takes a total of $O(t(2n_C - 1)^t) = O(n_C^{O(t)})$ time.

For each $u \in V \setminus (R \cup B)$, we compute point $p = \phi(u)$ in $O(t)$ time (this is possible since SSSP distances in G have been precomputed for each boundary vertex). Assume first that entry $T[p]$ is unmarked. Then we mark it and compute the n_p- and D_p-values and store them in the vectors associated with $T[p]$. By Lemma 2, computing and storing these values can clearly be done in time polynomial in n_C (a weak analysis but it suffices). From these values we compute $\sum(V_C \setminus B, u)$ in $O(t)$ time.

If $T[p]$ is already marked then we do not compute n_p- and D_p-values. Instead we perform a lookup in T at entry $T[p]$ to obtain $\sum(V_C \setminus B, u)$ in $O(t)$ time.

Clearly, we compute n_p- and D_p-values at most once for each $p \in \{-(n_C - 1), \ldots, n_C - 1\}^t$. It follows that the above algorithm computes $\sum(V_C \setminus B, V \setminus (R \cup B))$ in $O(nt + n_C^{O(t)})$ time.

From the above and from (1), we get the following result.

Lemma 4. *For each region R, values $\sum(R \setminus B, V \setminus (R \cup B))$ and $\sum(R \setminus B, R \setminus B)$ can be computed in $O(n\sqrt{r} + r^{O(\sqrt{r})})$ time assuming shortest path distances from each boundary vertex of R to each vertex in G have been precomputed.*

We are now ready for the main result of this paper.

Theorem 1. *The Wiener index $\sum G$ of a planar n-vertex graph G can be computed in $O(n^2 \log\log n / \log n)$ time.*

Proof. Computing shortest path distances from each boundary vertex to each vertex in G can be done in $O(n^2/\sqrt{r})$ time.

Applying Lemma 4 to each of the $\Theta(n/r)$ regions in the r-division of G gives us $\sum G$ in $O(n^2/\sqrt{r} + nr^{c'\sqrt{r}})$ time for some constant c'.

We pick $r = c(\log n/\log\log n)^2$ where $c > 0$ is some constant (to be specified). For $n > 2^c$ we have $\log n > c$, and for $n > 4$ we have $\log\log n > 1$ (assuming base 2 logarithms). Thus, for $n > \max\{2^c, 4\}$,

$$r^{c'\sqrt{r}} < (c\log n)^{2c'\sqrt{c}\log n/\log\log n} < (\log n)^{4c'\sqrt{c}\log n/\log\log n} = n^{4c'\sqrt{c}},$$

since $(\log n)^{\log n/\log\log n} = n$. It follows that if we choose $c < (1/(4c'))^2$, we have $r^{c'\sqrt{r}} = O(n^\epsilon)$, where $\epsilon < 1$. With this choice of r, the total running time of the algorithm is

$$O(n^2/\sqrt{r} + nr^{c'\sqrt{r}}) = O(n^2\log\log n/\log n + n^{1+\epsilon}) = O(n^2\log\log n/\log n),$$

as requested. □

5 Obtaining Linear Space Requirement

In this section, we show how space requirement of the above algorithm can be improved to linear without affecting running time.

By using the algorithm of [3], we can obtain an r-division of G using $O(n)$ space. Storing the table and the associated vectors for a ϕ-function requires $O(\sqrt{r}r^{O(\sqrt{r})}) = O(r^{O(\sqrt{r})})$ space. This is $o(n)$ with the choice of r in Theorem 1. However, our algorithm keeps shortest path distances in memory requiring more than linear space. So in order to obtain linear space requirement, we need to modify the algorithm in such a way that SSSP distances in G for only a constant number of sources need to be stored in memory at any time.

To do this, consider connected component C with boundary vertices $p_1, \ldots, p_t$ and belonging to a region R and let B be the set of boundary vertices as in Section 4. We modify the algorithm such that it computes value $\sum(V_C \setminus B, V \setminus (R \cup B))$ in t iterations. With each vertex u of $V \setminus (R \cup B)$, an offset i_u in table T is kept and in each iteration, this offset is updated such that when the algorithm terminates, i_u will specify the entry in T corresponding to $\phi(u)$. The algorithm then proceeds to compute $\sum(V_C \setminus B, V \setminus (R \cup B))$ as described in Section 4 before Lemma 4.

We now describe how the algorithm iteratively computes offsets in T for vertices in $V \setminus (R \cup B)$. It will prove useful to represent table T in memory in such a way that entry $(a_1, \ldots, a_t) \in \{-(n_C - 1), \ldots, n_C - 1\}^t$ has offset

$$\sum_{j=1}^{t} (2n_C - 1)^{j-1}(a_j + n_C - 1)$$

in T (this is similar to how higher-dimensional tables are stored in the C programming language).

Initially, offset i_u is zero for each u. Also, SSSP distances with source p_1 are precomputed and stored in $O(n)$ time and space. In the jth iteration, $j = 1, \ldots, t$, SSSP distances in G with source p_j are computed and stored. The jth component of $\phi(u)[j]$ is then computed for each u and offset i_u is increased by $(2n_C + 1)^{j-1}(\phi(u)[j] + n_C - 1)$.

It follows from the way T is stored in memory that this algorithm computes offsets for the correct entries for each vertex in $V \setminus (R \cup B)$.

As for running time, observe that for any $u \in V \setminus (R \cup B)$, value $\phi(u)[j] = d_G(u, p_j) - d_G(u, p_1)$ can be computed in constant time in iteration j since SSSP distances with source p_1 and source p_j have been precomputed. Hence, we use $O(n)$ time in each iteration and so it takes a total of $O(nt)$ time to find all offsets.

Space requirement is clearly linear since we only store SSSP distances for two sources at any time and since T requires sublinear space with the choice of r in Theorem 1.

Applying the above algorithm to each connected component of R, it follows from the results of Section 4 that $\sum(R \setminus B, V \setminus (R \cup B))$ can be computed in $O(n\sqrt{r} + r^{O(\sqrt{r})})$ time and $O(n)$ space.

We can also find $\sum(R \setminus B, R \setminus B)$ within this time and space bound. This follows easily from the observation that only shortest path distances in G between vertices of R are needed (see Section 4).

Setting r as in Theorem 1, the above modifications to the algorithm of Section 4 gives us the following result.

Theorem 2. *The Wiener index $\sum G$ of a planar n-vertex graph G can be computed in $O(n^2 \log\log n / \log n)$ time and $O(n)$ space.*

6 Concluding Remarks

We solved the open problem of whether an $o(n^2)$ time algorithm exists for computing the Wiener index of an n-vertex planar graph. We did this by exhibiting an algorithm with $O(n^2 \log\log n / \log n)$ running time and $O(n)$ space requirement.

We pose the following problems: is there an $o(n^2)$ time algorithm for computing the sum of distances between all pairs of vertices in a *weighted* n-vertex planar graph? Is there a constant $c < 2$ such that the Wiener index of an n-vertex planar graph can be computed in $O(n^c)$ time?

References

[1] S. Cabello and C. Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. Manuscript, Berlin, 2007.

[2] V. Chepoi and S. Klavžar. The Wiener index and the Szeged index of benzenoid systems in linear time. J. Chem. Inf. Comput. Sci., 37:752–755, 1997.

[3] G. N. Frederickson. Fast algorithms for shortest paths in planar graphs, with applications. SIAM J. Comput., 16 (1987), pp. 1004–1022.

[4] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster Shortest-Path Algorithms for Planar Graphs. Journal of Computer and System Sciences volume 55, issue 1, August 1997, pages 3–23.

[5] R. J. Lipton and R. E. Tarjan. A Separator Theorem for Planar Graphs. STAN-CS-77-627, October 1977.

[6] B. Mohar and T. Pisanski. How to compute the Wiener index of graph. J. Math. Chem., pages 267–277, 1988.

[7] H. Wiener. Structural determination of paraffin boiling points. J. Amer. Chem. Sot., 69:17–20, 1947.

[8] B. Zmazek and J. Žerovnik. Computing the weighted Wiener and Szeged number on weighted cactus graphs in linear time. Croatica Chemica Acta, 76:137–143, 2003.