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

Preprint series, Vol. 46 (2008), 1039

ALGORITHMS FOR GRAPHS OF BOUNDED TREEWIDTH VIA ORTHOGONAL RANGE SEARCHING

Sergio Cabello Christian Knauer

ISSN 1318-4865

February 4, 2008

Ljubljana, February 4, 2008

Algorithms for graphs of bounded treewidth via orthogonal range searching

Sergio Cabello* Christian Knauer†

January 5, 2008

Abstract

We show that, for any fixed constant $k \geq 3$, the sum of the distances between all pairs of vertices of an abstract graphs with n vertices and treewidth at most k can be computed in $O(n \log^{k-1} n)$ time.

We also show that, for any fixed constant $k \geq 2$, the dilation of a geometric graph (i.e., a graph drawn in the plane with straight-line segments) with n vertices and treewidth at most k can be computed in $O(n \log^{k+1} n)$ expected time. Recall that the dilation (or stretch-factor) of a geometric graph is the largest ratio, taken over all pairs of vertices, between the distance measured along the graph and the Euclidean distance.

The algorithms for both problems are based on the same principle: data structures for orthogonal range searching in bounded dimension provide a compact representation of distances in abstract graphs of bounded treewidth.

1 Introduction

Let $G = (V, E)$ be a graph with n vertices and assume that each edge of E has an associated nonnegative abstract length $\ell(e)$. We can define the length of a path in G as the sum of the lengths of its edges. The shortest path distance $d_G(u, v)$ between any pair of vertices u, v is defined as the minimum length over all walks in G between u, v. We are interested in the sum over all ordered pairs of vertices of their distance, that is,

$$\Sigma(G) = \sum_{(u,v)\in V^2} d_G(u, v).$$

If the length of each edge is one, the value $\frac{1}{2}\Sigma(G)$ is known as the *Wiener index* of G, which is a generalization of the original definition given by Wiener in 1947 [23]. *Molecular topological indices* are values defined by the graph-model of a molecule with the hope that they correlate with physical and chemical properties of the molecules [22]. The Wiener index is probably the most studied molecular topological index, with over thousand publications.

From the algorithmical point of view, the main question is what classes of graphs do not require to compute all the pairwise distances to obtain the Wiener index, or more generally, the value $\Sigma(G)$. Linear time algorithms are known for trees [18], cacti [25], and benzenoid systems[1] [11, 12].

*Department of Mathematics, IMFM, and Department of Mathematics, FMF, University of Ljubljana, Slovenia. E-mail: sergio.cabello@fmf.uni-lj.si. Research supported by the Slovenian Research Agency, project J1-7218.

†Institut für Informatik, Freie Universität Berlin, Takustraße 9, D–14195 Berlin, Germany. E-mail: christian.knauer@inf.fu-berlin.de.

[1]Benzenoid systems are subgraphs of the regular hexagonal grid enclosed by a circuit.

One of the main algorithmical open problems in this context concerns the existence of subquadratic algorithms for computing the Wiener index of planar graphs.

The average distance of a graph and $\Sigma(G)$ are essentially the same object, and have been studied in other models. For abstract discrete metric spaces, given by a matrix of distances, Indyk [16] gives a sublinear $(1+\varepsilon)$-approximation algorithm based on sampling; see also Barhum *et al.* [3]. Note that this model is substantially different, since it assumes that any distance is available at constant time, which does not hold in general graphs. The well-separated pair decomposition [8] can be used to obtain deterministic $(1+\varepsilon)$-approximations to the average distance in Euclidean spaces or, more generally, in spaces of bounded doubling dimension [15]. Expanders can also be used to design deterministic $(1+\varepsilon)$-approximation algorithms for computing the average distance in Euclidean metric spaces [3].

A *geometric graph* is a graph whose vertex set is a finite set of points, and where the weight/length of each edge equals the Euclidean distance between its vertices. The *dilation* $\Delta(G)$ of a geometric graph G is the largest ratio between the distance d_G and the Euclidean distance:

$$\Delta(G) := \max_{u,v \in V(G), u \neq v} \frac{d_G(u,v)}{||u-v||}.$$

Substantial research has been done about constructing so-called geometric spanners: geometric graphs with small dilation, few edges, and other additional properties; see the monograph [20]. Here, we turn our attention to a different problem: computing the dilation of a given geometric graph.

One can trivially compute the distance between all pairs of vertices, and then compute the dilation. However, this approach neglects all the geometry of the problem, and the question of whether one actually needs to compute all distances naturally arises. Agarwal *et al.* [2] give near-linear time algorithms for computing the exact dilation of geometric paths, cycles, and trees. For computing a $(1+\varepsilon)$-approximation to the dilation, assuming that $\varepsilon > 0$ is constant, Narasimhan and Smid [19] show that the problem reduces in $O(n \log n)$ time to compute the graph distance between $O(n)$ pairs of vertices. Combining this result with the data structure of Chaudhuri and Zaroliagis [10], one obtains, for any fixed $\varepsilon > 0$, a $(1+\varepsilon)$-approximation algorithm for the dilation that runs in $O(n \log n)$ time.

Our results. We show that the sum of distances of an abstract graph and the dilation of a geometric graph can be computed in near-linear time when the graphs have bounded treewidth. More precisely:

- for any fixed constant $k \geq 3$, the sum of distances $\Sigma(G)$ can be computed in $O(n \log^{k-1} n)$ time for any abstract graph with n vertices and treewidth at most k. See Section 4.
- for any fixed constant $k \geq 2$, the dilation $\Delta(G)$ can be computed in $O(n \log^{k+1} n)$ expected time for any geometric graph with n vertices and treewidth at most k. See Section 5.

Similar results were only known for graphs of treewidth 1 (trees) and some subclasses of graphs with treewidth 2 (cycles and cacti), c.f. [2, 25]. Our algorithms are based on divide-and-conquer, using the fact that graphs with bounded treewidth have balanced small separators. Although most algorithms that use treewidth are based on dynamic programming, this approach does not seem appropriate here.

The approach in both cases is the same: distances in abstract graphs of treewidth k are closely related to orthogonal range searching in $\mathbb{R}^{k-1}$; see Section 3. This approach is implicit in [4] and

more explicitly mentioned by Shi [21]. This relation allows to use techniques from computational geometry to design efficient algorithms for problems involving distances in graphs of bounded treewidth.

2 Toolbox

2.1 Treewidth

Treewidth is a parameter measuring, in some sense, the complexity of a graph. We next give its definition; see Bodlaender [7] for an overview.

Definition 1. *A* tree decomposition *of a graph G is a pair (X, T), where $X = \{X_i \subseteq V(G) \mid i \in I\}$ is a collection of subsets of $V(G)$ (called* bags*), and a tree $T = (I, F)$ with a node set I such that:*

(i) $V(G) = \bigcup_{i \in I} X_i$;

(ii) *for every edge $uv \in E(G)$ there is some bag X_i such that $u, v \in X_i$;*

(iii) *for all $u \in V(G)$, the nodes $\{i \in I \mid u \in X_i\}$ form a connected subtree of T.*

The width *of a tree decomposition $(\{X_i \mid i \in I\}, T)$ is $\max_{i \in I} |X_i| - 1$. The* treewidth *of G is the minimum width over all tree decompositions of G.*

It is known that graphs of treewidth k have $O(kn)$ edges, an thus graphs of bounded treewidth have $O(n)$ edges. Computing the treewidth of a graph is NP-hard. However, for any fixed constant $k > 0$, we can decide in linear time if a given graph has treewidth at most k, and in such case, construct a tree decomposition of linear size and width k in linear time [6]. Our main results apply assuming that we have graphs of bounded treewidth.

Our approach will be based on divide-and-conquer. We will use the following concept, closely related to separators.

Definition 2. *Let A be a subset of vertices of the graph G. The* portals *of A are the vertices of A that have some edge incident to $V(G) \setminus A$.*

A standard result for graphs with treewidth at most k is the existence of (2/3)-separators of size $k + 1$. In our approach, the number of vertices in the separator is very relevant, and therefore we will reduce the size of the separator by making the separation more unbalanced. A sketch of the following result can be found in the notes of Biedl [5].

Lemma 3. *Let $k \geq 1$ be a constant. Given a graph G with $n > k + 1$ vertices and treewidth at most k, we can find in linear time a subset of vertices $A \subseteq V(G)$ such that:*

(i) *A has between $\frac{n}{k+1}$ and $\frac{nk}{k+1}$ vertices;*

(ii) *A has at most k portals;*

(iii) *adding edges between the portals of A does not change the treewidth.*

Proof. Consider a tree decomposition of G with width k, and transform it into another tree decomposition $(\{X_i \mid i \in I\}, T)$ whose tree T has maximum degree at most $k + 1$ and such that any two adjacent bags in T differ by at least one vertex.

Assign each vertex of G to one and only one bag where it appears, and define the weight of a bag to be the number of vertices that were assigned to it. In such vertex-weighted tree T of

maximum degree at most $k+1$ there exists always an edge ij whose removal leaves two connected components, each with weight at most $k/(k+1)$ of the total. The vertices in the bags of one of the connected components of $T - ij$ is the set A we want. Note that $S = X_i \cap X_j$ has at most k vertices and it is a superset of the portals of A. Moreover, since S is a subset of a bag, adding the edges between the portals of A does not change the treewidth of the graph.

This procedure can be implemented in linear time if k is bounded by a constant: constructing a tree decomposition of linear size takes linear time [6], making the transformation to obtain (X, T) takes linear time, and finding the edge ij of T to remove takes linear time. □

Assume that we have a graph G with treewidth k and a subset of vertices A with the properties stated in Lemma 3. Let S be the set of portals of A and let $B = (V(G) \setminus A) \cup S$. Note that the set of portals of B is a subset of S.

Consider the graph G' obtained from G by adding an edge ss' with weight $d_G(s, s')$ between each pair $s, s' \in S$. If G' has multiple edges, we only keep the edges with minimum weight. Finally, let G_A denote the subgraph of G' induced by A. From the property (iii) of A, we know that G' has treewidth k, and thus G_A has treewidth at most k. Furthermore, it is straightforward to see that $d_G(a, a') = d_{G_A}(a, a')$ for any $a, a' \in A$. Using the notation $B = (V(G) \setminus A) \cup S$, the same argument applies to G_B: it has treewidth at most k and $d_G(b, b') = d_{G_B}(b, b')$ for all $b, b' \in B$.

2.2 Orthogonal range searching

Let P be a set of points in $\mathbb{R}^d$. Assume we are given a function $w : P \to \mathbb{R}$ that assigns a weight $w(p)$ to each point $p \in P$. We extend the weight function to any subset Q of points by $w(Q) := \sum_{p \in Q} w(p)$. A *rectangle* R in $\mathbb{R}^d$ is the Cartesian product of d intervals, $R = I_1 \times \cdots \times I_d$, where each interval I_i can include both extremes, one of them, or none.

Orthogonal range searching deals with the problem of preprocessing P such that, for a query rectangle R, certain properties of $P \cap R$ can be efficiently reported. We will use the following two results.

Theorem 4 ([24]). *Let $d \geq 2$ be a constant. Given a set of n points $P \subset \mathbb{R}^d$ and a weight function $w : P \to \mathbb{R}$, there is a data structure that can be constructed in $O(n \log^{d-1} n)$ time such that, for any query rectangle R, the weight $w(P \cap R)$ can be reported in $O(\log^{d-1} n)$ time.*

Theorem 5 ([1], [13, Chapter 5]). *Let $d \geq 1$ be a constant. Given a set of n points $P \subset \mathbb{R}^d$, there is a family $\mathcal{P} = \{P_j \subseteq P \mid j \in J\}$ of* canonical *subsets of P and a data structure $DS(P)$ with the following properties:*

(i) *$DS(P)$ and $\mathcal{P}$ can be constructed in $O(n \log^d n)$ time;*

(ii) *$|J| = O(n \log^{d-1} n)$, that is, $\mathcal{P}$ consists of $O(n \log^{d-1} n)$ subsets;*

(iii) *$\sum_{j \in J} |P_j| = O(n \log^d n)$, that is, counting multiplicities $\mathcal{P}$ has $O(n \log^d n)$ points;*

(iv) *for any rectangle R, there exists a set of $O(\log^d n)$ indices $J(R) \subset J$ such that $P \cap R = \bigcup_{j \in J(R)} P_j$;*

(v) *for any query rectangle R, the data structure $DS(P)$ provides the set of indices $J(R)$ in $O(\log^d n)$ time.*

3 Distances and orthogonal range searching

Let G be a graph and let A be a subset of its vertices with k portals S, enumerated as $s_1, \ldots, s_k$. Let $B = (V(G) \setminus A) \cup S$. We use the notation $[k] = \{1, \ldots, k\}$. For any vertex $b \in B$ and any index $i \in [k]$, let $A(b, i)$ be the subset of vertices $a \in A$ such that:

(i) There exists a shortest path from b to a through s_i, that is, $d_G(a, b) = d_G(a, s_i) + d_G(s_i, b)$.

(ii) There is no shortest path from b to a through s_j for $j < i$, that is, $d_G(a, b) < d_G(a, s_j) + d_G(s_j, b)$ for all $j \in [i-1]$.

For any $b \in B$, the union of $A(b, 1), \ldots, A(b, k)$ is the whole A. We include (ii) to ensure that the sets $A(b, 1), \ldots, A(b, k)$ are pairwise disjoint, which will be relevant for not over counting in Section 4. Note that different enumerations of the portals may give completely different sets $A(b, 1), \ldots, A(b, k)$, not just a reordering.

Assume that we have a weight function $\phi : A \to \mathbb{R}$ assigning a weight to each vertex of A. We extend the weight function to any subset $A' \subseteq A$ by $\phi(A') := \sum_{a \in A'} \phi(a)$. For each $i \in [k]$, we can use orthogonal range searching to preprocess the graph G so that, for any query vertex $b \in B$, information concerning $A(b, i)$ can be reported quickly and in a compact form. Note the similarity between the following result and Theorem 4.

Theorem 6. *Let $k \geq 3$ be a constant. Assume we are given a graph G with n vertices and m edges, a subset of vertices A with k portals S enumerated $s_1, \ldots, s_k$, a weight function $\phi : A \to \mathbb{R}$, and let $B = (V(G) \setminus A) \cup S$. For any given $i \in [k]$, there is a data structure that can be constructed in $O(m + n \log^{k-2} n)$ time such that, for any query vertex $b \in B$, the weight $\phi(A(b, i))$ can be reported in $O(\log^{k-2} n)$ time.*

Proof. We first construct a shortest path tree from each of the portals $s_1, \ldots, s_k$ and store the values $d_G(s_j, v)$ for all $j \in [k]$ and $v \in V(G)$. Since we assume $k = O(1)$, we spend $O(m + n \log n)$ time for computing these shortest path trees.

For each vertex $a \in A$ we define a point $p_a \in \mathbb{R}^k$ with coordinates $p_a(j) = d_G(a, s_i) - d_G(a, s_j)$, where $p_a(j)$ denotes the j-th coordinate of point p_a. Let P be the set of points $p_a, a \in A$. Note that the i-th coordinate of the points in P is always 0, and therefore we can regard P as a set of $|A|$ points in $\mathbb{R}^{k-1}$. We define a weight function for each point $p_a \in P$ by $w(p_a) := \phi(a)$. Clearly, we have $\phi(A') = w(\{p_a \in P \mid a \in A'\})$. Finally, we preprocess the point set P with weight w into a data structure as described in Theorem 4, where $d = k - 1$. This finishes the description of the data structure. Preprocessing P takes $O(|A| \log^{k-2} |A|) = O(n \log^{k-2} n)$ time, and hence we spend $O(n \log^{k-2} n)$ time to construct the data structure.

When we receive a query $b \in B$, we proceed as follows. For $j \in [k]$ define the interval $I_j(b)$ by

$$I_j(b) = \begin{cases} (-\infty,\ d_G(s_j, b) - d_G(s_i, b)) & \text{if } j < i, \\ (-\infty,\ +\infty) & \text{if } j = i, \\ (-\infty,\ d_G(s_j, b) - d_G(s_i, b)] & \text{if } j > i. \end{cases}$$

Consider the rectangle $R(b) = I_1(b) \times \cdots \times I_k(b)$.

Any path from $a \in A$ to b has to use some portal of A, and hence the condition $d_G(a, b) = d_G(a, s_i) + d_G(s_i, b)$ can be rewritten as

$$d_G(a, s_i) + d_G(s_i, b) \leq d_G(a, s_j) + d_G(s_j, b) \quad \text{for all } j \in [k].$$

Therefore

$$\begin{aligned}
A(b,i) &= \left\{ a \in A \;\middle|\; \begin{array}{l} d_G(a,b) = d_G(a,s_i) + d_G(s_i,b) \\ d_G(a,b) < d_G(a,s_j) + d_G(s_j,b) \text{ for } j \in [i-1] \end{array} \right\} \\
&= \left\{ a \in A \;\middle|\; \begin{array}{l} d_G(a,s_i) + d_G(s_i,b) \leq d_G(a,s_j) + d_G(s_j,b) \text{ for } j \in [k] \\ d_G(a,s_i) + d_G(s_i,b) < d_G(a,s_j) + d_G(s_j,b) \text{ for } j \in [i-1] \end{array} \right\} \\
&= \left\{ a \in A \;\middle|\; \begin{array}{l} d_G(a,s_i) - d_G(a,s_j) \leq d_G(s_j,b) - d_G(s_i,b) \text{ for } j \in [k] \\ d_G(a,s_i) - d_G(a,s_j) < d_G(s_j,b) - d_G(s_i,b) \text{ for } j \in [i-1] \end{array} \right\} \\
&= \left\{ a \in A \;\middle|\; \begin{array}{l} p_a(j) \leq d_G(s_j,b) - d_G(s_i,b) \text{ for } j \in [k] \setminus [i-1] \\ p_a(j) < d_G(s_j,b) - d_G(s_i,b) \text{ for } j \in [i-1] \end{array} \right\} \\
&= \left\{ a \in A \;\middle|\; \begin{array}{l} p_a(j) \in I_j(b) \text{ for } j \in [k] \setminus [i-1] \\ p_a(j) \in I_j(b) \text{ for } j \in [i-1] \end{array} \right\} \\
&= \{ a \in A \mid p_a \in R(b) \} .
\end{aligned}$$

We conclude that $A(b,i) = \{a \in A \mid p_a \in R(b)\}$. Since

$$\phi\left(A(b,i)\right) = \phi\left(\{a \in A \mid p_a \in R(b)\}\right) = w\left(P \cap R(b)\right),$$

we can obtain $\phi(A(b,i))$ in $O(\log^{k-2} n)$ time by querying the data structure storing P for the value $w(P \cap R(b))$. □

In the previous proof we could use the data structure from Theorem 5, instead of that from Theorem 4. We next state the result in a slightly more general form, which is the one we will use in Section 5.

Theorem 7. *Let $k \geq 2$ be a constant. Assume we are given a graph G with n vertices and m edges, a subset of vertices A with k portals S enumerated $s_1, \ldots, s_k$, and let $B = (V(G) \setminus A) \cup S$. Furthermore, assume that G has been preprocessed so that any distance $d_G(v,s)$ can be obtained in constant time when $(v,s) \in V(G) \times S$. For any given $i \in [k]$ and any given $A' \subseteq A$, there is a family $\mathcal{A} = \{A_j \subseteq A \mid j \in J\}$ of canonical subsets of A and a data structure DS with the following properties:*

(i) *DS and $\mathcal{A}$ can be constructed in $O(|A'| \log^{k-1} n)$ time;*

(ii) *$|J| = O(|A'| \log^{k-2} n)$, that is, $\mathcal{A}$ consists of $O(|A'| \log^{k-2} n)$ subsets;*

(iii) *$\sum_{j \in J} |A_j| = O(|A'| \log^{k-1} n)$, that is, counting multiplicities $\mathcal{A}$ has $O(|A'| \log^{k-1} n)$ points;*

(iv) *for any $b \in B$, there exists a set of $O(\log^{k-1} n)$ indices $J(b) \subset J$ such that $A(b,i) \cap A' = \bigcup_{j \in J(b)} A_j$;*

(v) *for any query $b \in B$, the data structure DS provides $J(b)$ in $O(\log^{k-1} n)$ time.*

Proof. The proof is almost identical to the proof of Theorem 6. Each vertex $a \in A'$ is identified with a point $p_a \in \mathbb{R}^k$ whose j-th coordinate is given by $d_G(a,s_i) - d_G(a,s_j)$. For each point $a \in A'$ we can compute the coordinates of p_a in $O(k) = O(1)$ time because we assume that each distance from a portal is available in constant time. Therefore, we can construct the set of $|A'|$ points $P = \{p_a \mid a \in A'\}$ in $O(|A'|)$ time. The set P can be regarded as a point set in $\mathbb{R}^{k-1}$ because the i-th coordinate is always zero.

We apply Theorem 5 to P, where $d = k-1$, and obtain in $O(|A'|\log^{k-1} n)$ time the family $\mathcal{P} = \{P_j \subseteq P \mid j \in J\}$ and the data structure $DS(P)$. The family $\mathcal{A}$ we seek is defined from $\mathcal{P}$ by the identification $a \leftrightarrow p_a$. The set of indices J and the data structure DS we want are precisely J and $DS(P)$, as obtained from Theorem 5. The properties (i-iii) already hold. For property (iv), note that for any query b we have $J(b) = J(R(b))$, where $R(b)$ is the rectangle defined in the proof of Theorem 6. As for the remaining property (v), note that the intervals defining the rectangle $R(b)$ can be computed in $O(k) = O(1)$ time because they only involve distances from portals, and therefore we can obtain $J(b) = J(R(b))$ in $O(\log^{k-1} n)$ by querying $DS(P)$ for $J(R(b))$. □

4 Sum of distances

We are interested in computing $\Sigma(G)$ for a weighted graph G whose treewidth is bounded by a constant $k \geq 3$. Let A be a set of vertices of G obtained by Lemma 3. Like before, we enumerate the (at most) k portals S of A by $s_1, \ldots, s_k$, and set $B = (V(G) \setminus A) \cup S$. Recall the definition of G_A and G_B, given after Lemma 3. Using the notation $\Sigma(C, C') = \sum_{c \in C} \sum_{c' \in C'} d_G(c, c')$ for any $C, C' \subseteq V(G)$, we have the following relation.

Lemma 8.

$$\Sigma(G) = \Sigma(G_A) + \Sigma(G_B) + 2 \cdot \Sigma(A, B) - 2 \cdot \Sigma(S, A) - 2 \cdot \Sigma(S, B).$$

Proof. Let $V = V(G)$ and let $\sqcup$ denote the disjoint union. Since $V = A \sqcup (B \setminus S)$ we have

$$V^2 \;=\; A^2 \sqcup (A \times (B \setminus S)) \sqcup (B \setminus S)^2 \sqcup ((B \setminus S) \times A)\,,$$

and therefore

$$\Sigma(G) \;=\; \Sigma(V, V) \;=\; \Sigma(A, A) + 2 \cdot \Sigma(A, B \setminus S) + \Sigma(B \setminus S, B \setminus S). \tag{1}$$

Since $B^2 = (B \setminus S)^2 \sqcup (S \times B) \sqcup (B \times S)$ and $A \times B = (A \times (B \setminus S)) \sqcup (A \times S)$ we have

$$\Sigma(B \setminus S, B \setminus S) \;=\; \Sigma(B, B) \;-\; 2 \cdot \Sigma(S, B) \quad \text{and} \quad \Sigma(A, B \setminus S) \;=\; \Sigma(A, B) \;-\; \Sigma(A, S).$$

Substituting in (1) we obtain

$$\Sigma(G) \;=\; \Sigma(A, A) + 2 \cdot \Sigma(A, B) - 2 \cdot \Sigma(A, S) + \Sigma(B, B) - 2 \cdot \Sigma(S, B),$$

and noting that $\Sigma(A, A) = \Sigma(G_A)$, $\Sigma(B, B) = \Sigma(G_B)$ the result follows. □

Lemma 9. *We can compute $\Sigma(A, B)$ in $O(n \log^{k-2} n)$ time.*

Proof. For any $b \in B$, we know that A is the disjoint union of $A(b, 1), A(b, 2), \ldots, A(b, k)$. Therefore

$$\begin{aligned}
\Sigma(A, B) &= \sum_{b \in B} \sum_{a \in A} d_G(a, b) \\
&= \sum_{b \in B} \sum_{i=1}^{k} \sum_{a \in A(b,i)} d_G(a, b) \\
&= \sum_{b \in B} \sum_{i=1}^{k} \sum_{a \in A(b,i)} (d_G(a, s_i) + d_G(s_i, b)) \\
&= \sum_{b \in B} \sum_{i=1}^{k} \left(|A(b, i)| \cdot d_G(s_i, b) + \sum_{a \in A(b,i)} d_G(a, s_i) \right).
\end{aligned}$$

Define weight functions $\phi_0, \phi_1, \ldots \phi_k : A \to \mathbb{R}$ by $\phi_0(a) = 1$ for all $a \in A$ and by $\phi_i(a) = d_G(s_i, a)$ for $a \in A, i \in [k]$. We can then rewrite

$$\Sigma(A, B) = \sum_{b \in B} \sum_{i=1}^{k} \left[\phi_0(A(b,i)) \cdot d_G(s_i, b) + \phi_i(A(b,i))\right]. \tag{2}$$

We now use Theorem 6 several times: for each $i \in [k]$ we make a data structure $DS_0^{(i)}$ for weight ϕ_0 and a data structure $DS_1^{(i)}$ for weight ϕ_i. We construct $2k = O(1)$ data structures, and each one takes $O(n \log^{k-2} n)$ preprocessing time. Any value $\phi_0(A(b,i))$ or $\phi_i(A(b,i))$ can now be obtained in time $O(\log^{k-2} n)$ by querying the appropriate data structure. Therefore, we can get the values $\phi_0(A(b,i))$ and $\phi_i(A(b,i))$ for all $(b,i) \in B \times [k]$ in time $O(k|B| \log^{k-2} n) = O(n \log^{k-2} n)$. Finally, the values $d_G(s_i, b)$ for all $(s_i, b) \in S \times B$ can be obtained in $O(n \log n)$ time constructing a shortest path tree from each portal $s_i \in S$, and thus we can compute $\Sigma(A, B)$ using (2). □

Theorem 10. *Let $k \geq 3$ be a constant. Given a graph G with n vertices and treewidth at most k, we can compute $\Sigma(G)$ in $O(n \log^{k-1} n)$ time.*

Proof. We make an algorithm based on divide-and-conquer. If G has fewer than $k + 1 = O(1)$ vertices, we compute $\Sigma(G)$ by brute force in $O(1)$ time. Otherwise we find a set A as described in Lemma 3. The values $\Sigma(S, A)$ and $\Sigma(S, B)$ are computed using shortest path trees from each portal $s \in S$, while $\Sigma(A, B)$ is computed using Lemma 9. The graphs G_A and G_B can be constructed using shortest path trees from each $s \in S$, and we can then recursively compute $\Sigma(G_A)$ and $\Sigma(G_B)$. Finally, we use the relation from Lemma 8 to compute the value $\Sigma(G)$. This finishes the description of the algorithm.

If $T(n)$ denotes the worst-case time used by the algorithm when G has n vertices, then property (i) in Lemma 3 implies the recurrence

$$T(n) \leq \begin{cases} O(1) & \text{if } n \leq k+1. \\ O(n \log^{k-2} n) + T(|A|) + T(n - |A| + k) & \text{otherwise, where } \frac{n}{k+1} \leq |A| \leq \frac{nk}{k+1}. \end{cases}$$

Since k is a constant, this recurrence solves to $O(n \log^{k-1} n)$ by induction on n, and the result follows. □

Since graphs of treewidth 2 can be seen as graphs of treewidth 3, we obtain the following.

Corollary 11. *Given a graph G of treewidth at most 2, we can compute $\Sigma(G)$ in $O(n \log^2 n)$ time.*

The underlying reason why we get the same time bounds for $k = 2$ and $k = 3$ is that the orthogonal range queries considered in Theorem 4 have the same complexity in dimensions 1 and 2.

5 Dilation

A *weighted geometric graph* $G = (V, E)$ is a graph whose vertex set is a finite set of points in the plane, where each edge $e \in E$ has a nonnegative weight $\ell(e) \in \mathbb{R}$. When the weight of each edge $\{u, v\}$ is equal to the Euclidean distance $||u - v||$, the graph G is called a *geometric graph*.

The *dilation* between two *vertices* $x \neq y$ of G is defined as

$$\Delta_G(x, y) := \frac{d_G(x, y)}{||x - y||},$$

where $d_G(x, y)$ denotes the length (wrt. ℓ) of a shortest path in G connecting x to y. For convenience we define $\Delta_G(x, x) := 1$. The dilation between two *sets of vertices* X, Y of G is defined as

$$\Delta_G(X, Y) := \max_{(x,y)\in X\times Y} \Delta_G(x, y),$$

the dilation *of a set of vertices* X of G is defined as

$$\Delta_G(X) := \Delta_G(X, X), \text{ and}$$

the *dilation* (or *stretch factor*) of G is defined as

$$\Delta(G) := \Delta_G(V).$$

Henceforth, let G be a weighted geometric graph with n vertices and treewidth bounded by a constant $k \geq 2$. We will describe how to compute $\Delta(G)$ in $O(n \log^{k+1} n)$ expected time. As above, let A be a subset of $V(G)$ obtained by Lemma 3. We enumerate the k portals of S by $s_1, \ldots, s_k$, and set $B = (V(G) \setminus A) \cup S$. We compute a shortest path tree in G from each portal $s \in S$ in linear time [10], and store together with each vertex $v \in V$ the values $d_G(s_1, v), \ldots, d_G(s_k, v)$. With this preprocessing, any distance $d_G(v, s)$ can be recovered in constant time for any pair $(v, s) \in V \times S$. Also, the distance $d_G(a, b)$ for $(a, b) \in A \times B$ can be recovered in constant time because $d_G(a, b) = \min_i \{d_G(a, s_i) + d_G(b, s_i)\}$.

Recall the definition and properties of G_A and G_B, given after Lemma 3. Note that G_A, G_B are also weighted geometric graphs[2]. Since for any pair of vertices $a, a' \in A$ we have $d_G(a, a') = d_{G_A}(a, a')$, and similarly, $d_G(b, b') = d_{G_B}(b, b)$ for any $b, b' \in B$, we get

$$\Delta(G) = \max(\Delta(G_A), \Delta(G_B), \Delta_G(A, B)). \tag{3}$$

Consider a fixed $i \in [k]$, and recall from Section 3 the definition of $A(b, i) \subseteq A$ for any $b \in B$. For any subset $A' \subseteq A$ we define the function $\beta^{(i)}_{A'} : B \to \mathbb{R}$ as

$$\beta^{(i)}_{A'}(b) := \Delta_G(A(b, i) \cap A', b).$$

Lemma 12. *Let $\delta \geq 1$ be a given value. For any given $A' \subseteq A$ we can compute in $O(|A'| \log^k n)$ time a data structure $DB^{(i)}_{A'}$ that can decide in $O(\log^k n)$ time for any query $b \in B$ if $\beta^{(i)}_{A'}(b) \leq \delta$.*

Proof. According to Theorem 7 there is a family $\mathcal{A} = \{A_j \mid j \in J\}$ of $|J| = O(|A'| \log^{k-2} n)$ canonical subsets of A of total size $\sum_{j\in J} |A_j| = O(|A'| \log^{k-1} n)$ and a data structure DS that can be constructed in $O(|A'| \log^{k-1} n)$ time with the following properties:

- for any $b \in B$, there exists a set of $O(\log^{k-1} n)$ indices $J(b)$ such that $A(b, i) \cap A' = \bigcup_{j\in J(b)} A_j$, and
- for any query $b \in B$ the data structure DS provides the set of indices $J(b)$ in $O(\log^{k-1} n)$ time.

[2]When G is a geometric graph, it may happen that G_A, G_B are not geometric graphs, but weighted geometric graphs. This is the reason why we also need weighted geometric graphs when computing the dilation of geometric graphs.

Therefore, we can write

$$\beta^{(i)}_{A'}(b) = \max_{j \in J(b)} \Delta_G(A_j, b),$$

and thus

$$\beta^{(i)}_{A'}(b) \leq \delta \iff \forall j \in J(b) : \Delta_G(A_j, b) \leq \delta.$$

We now describe a lifting transformation that rephrases the problem of deciding for $A_j \subseteq A(b,i) \cap A'$ if $\Delta_G(A_j, b) \leq \delta$, into a point-cone incidence-problem in $\mathbb{R}^3$ (a similar approach is used in [2]).

Recall that s_i is the i-th portal of A so that $d_G(a,b) = d_G(a,s_i) + d_G(s_i,b)$ for any vertex $a \in A(b,i)$. For a vertex $v \in V(G)$, we define the *s_i-weight* of v to be $\omega(v) := d_G(v,s_i)/\delta$. We map each vertex $v = (x_v, y_v) \in \mathbb{R}^2$ of G to the point $h(v) := (x_v, y_v, \omega(v)) \in \mathbb{R}^3$. Let C denote the three-dimensional cone $\{(x,y,z) \in \mathbb{R}^3 \mid z^2 = x^2 + y^2\}$. We map each vertex $v \in V(G)$ to the cone $C(v) := C \ominus h(v) = \{(x,y,z) - h(v) \mid z^2 = x^2 + y^2\}$. Since $A_j \subseteq A(b,i)$ we have for any vertex $a \in A_j$ that

$$\begin{aligned}
\Delta_G(a,b) \leq \delta \quad &\iff \quad \frac{d_G(a,b)}{||a-b||} \leq \delta \\
&\iff \quad \frac{d_G(a,s_i) + d_G(s_i,b)}{||a-b||} \leq \delta \\
&\iff \quad \frac{d_G(s_i,b)}{\delta} \leq ||a-b|| - \frac{d_G(a,s_i)}{\delta} \\
&\iff \quad \omega(b) \leq ||a-b|| - \omega(a).
\end{aligned}$$

In other words $\Delta_G(a,b) \leq \delta$ if and only if $h(b)$ lies on or below $C(a)$. If we extend the lifting map to sets of vertices $X \subseteq V(G)$ by setting $h(X) := \{h(v) \mid v \in X\}$ and $C(X) := \{C(v) \mid v \in X\}$, we immediately get that $\Delta_G(A_j, b) \leq \delta$ if and only if $h(b)$ lies on or below $\mathcal{L}(C(A_j))$, the lower envelope of the cones $C(A_j)$.

The minimization diagram of $C(A_j)$, i.e., the projection of the lower envelope $\mathcal{L}(C(A_j))$ onto the xy-plane, is the additively weighted Voronoi diagram $\mathrm{Vor}(A_j)$ of A_j with respect to the weight function $-\omega$. If b is located in the Voronoi region of a point $a \in A_j$ we have that $h(b)$ is on or below $\mathcal{L}(C(A_j))$ if and only if $h(b)$ is on or below $C(a)$.

The data structure $DB^{(i)}_{A'}$ is constructed as follows:

1. Compute the data structure DS and the family $\mathcal{A}$ according to Theorem 7 for A'. This takes in $O(|A'| \log^{k-1} n)$ time because G has been preprocessed for distance queries, as required.

2. For each set A_j in the family $\mathcal{A} = \{A_j \mid j \in J\}$ of $|J| = O(|A'| \log^{k-2} n)$ canonical subsets of A compute the additively weighted Voronoi diagram $\mathrm{Vor}(A_j)$ of A_j with respect to the weight function $-\omega$.

   The diagram $\mathrm{Vor}(A_j)$ can be computed in $O(|A_j| \log |A_j|)$ time, c.f. [14]. Within the same time bound it can be preprocessed into a linear size data structure that supports point-location queries in $O(\log |A_j|) = O(\log n)$ time, c.f. [17]. Since $\sum_{j \in J} |A_j| = O(|A'| \log^{k-1} n)$ the total time for computing all these diagrams and their associated point-location structures is $O(|A'| \log^k n)$.

The total preprocessing time is $O(|A'| \log^k n)$.

To verify for $b \in B$ if $\beta^{(i)}_{A'}(b) \leq \delta$ we proceed as follows:

1. We query DS with b to determine the set $J(b)$ of $O(\log^{k-1} n)$ indices with $A(b,i) \cap A' = \bigcup_{j \in J(b)} A_j$ in $O(\log^{k-1} n)$ time.

2. For each $j \in J(b)$

   - perform a point-location query with b in $\mathrm{Vor}(A_j)$ in $O(\log n)$ time to determine which point $a \in A_j$ contains b in its Voronoi cell and check if $h(b)$ is on or below $C(a)$ in $O(1)$ time. If $h(b)$ lies above $C(a)$ terminate with a "no" answer.

3. Terminate with a "yes" answer.

The correctness of this approach follows from the preceding discussion. It requires $O(\log^k n)$ time per query. □

Lemma 13. *Given a value $\delta \geq 1$ and subsets $A' \subset A$, $B' \subseteq B$, we can decide in $O(|A'| \log^k n + |B'| \log^k n)$ time if $\Delta_G(A', B') \leq \delta$.*

Proof. For any $b \in B$, we know that A is the union of $A(b,1), A(b,2), \ldots, A(b,k)$. Therefore, for any subsets $A' \subseteq A$, $B' \subseteq B$ we have

$$\Delta_G(A', B') = \max_{b \in B'} \Delta_G(A', b) = \max_{b \in B'} \max_{i \in [k]} \Delta_G(A(b,i) \cap A', b) = \max_{i \in [k]} \max_{b \in B'} \beta^{(i)}_{A'}(b).$$

and thus

$$\Delta_G(A', B') \leq \delta \iff \forall i \in [k], b \in B' : \beta^{(i)}_{A'}(b) \leq \delta.$$

For each $i \in [k]$ we construct the data structure $DB^{(i)}_{A'}$ of Lemma 12 for the given δ, and then query $DB^{(i)}_{A'}$ with each $b \in B'$ to decide if $\beta^{(i)}_{A'}(b) \leq \delta$. The construction of $DB^{(i)}_{A'}$ takes $O(|A'| \log^k n)$ time and all queries together take $O(|B'| \log^k n)$ time. The result follows. □

Lemma 14. *In $O(n \log^k n)$ expected time we can compute $\Delta_G(A, B)$*

Proof. We use the technique developed by Chan [9] to obtain an algorithm that solves the optimization problem once we have an algorithm that solves the decision problem (Lemma 13). Consider the following randomized algorithm to compute $\Delta_G(A', B')$ for given input $A' \subseteq A$, $B' \subseteq B$.

Algorithm (A,B)-DILATION
Input: A', B'
Output: $\Delta_G(A', B')$

```
1.  if A' and B' have O(1) vertices
2.     then compute Δ_G(A', B') directly in O(1) time and return it
3.     else partition A' into three subsets A_1, A_2, A_3 whose cardinality differ by at most one
4.          partition B' into three subsets B_1, B_2, B_3 whose cardinality differ by at most one
5.          let (X_1, Y_1), (X_2, Y_2), ..., (X_9, Y_9) be a random permutation of (A_i, B_j), i, j ∈ [3]
6.          δ←∞
7.          for i = 1, ..., 9
8.              do use Lemma 13 to check if Δ_G(X_i, Y_i) ≤ δ
9.                 if Δ_G(X_i, Y_i) > δ
10.                   then δ←(A,B)-DILATION(X_i, Y_i)
11. return δ
```

The correctness of the algorithm follows by induction because

$$\Delta_G(A', B') = \max\{\Delta_G(A_i, B_j) \mid i, j \in [3]\} = \max\{\Delta_G(X_i, X_i) \mid i \in [9]\},$$

and we can therefore compute $\Delta_G(A, B)$ by calling (A,B)-DILATION(A, B).

For analyzing the running time, let $T(m)$ denote the worst-case expected running time of (A,B)-DILATION(A', B') when $m = \max\{|A'|, |B'|\}$. Excluding the time used in step 10, the algorithm takes $O(|A'| \log^k n + |B'| \log^k n) = O(m \log^k n)$ time. As observed by Chan [9], the expected number of times that step 10 is performed is bounded by $1+1/2+1/3+\cdots+1/9 \leq 2,83$. The expected time used in each execution of step 10 is bounded by $T(\max\{|X_i|, |Y_i|\}) = T(m/3)$, and therefore $T(m)$ satisfies the recursion $T(m) \leq O(m \log^k n) + 2,83\, T(m/3)$. This recursion solves by induction to $T(m) = O(m \log^k n)$, and thus (A,B)-DILATION(A, B) computes $\Delta_G(A, B)$ in $O(n \log^k n)$ expected time. □

Theorem 15. *Let $k \geq 2$ be a constant. Given a weighted geometric graph G with n vertices and treewidth at most k, we can compute its dilation $\Delta(G)$ in $O(n \log^{k+1} n)$ expected time.*

Proof. We proceed as in the proof of Theorem 10 with an algorithm based on divide-and-conquer. If G has fewer than $k+1 = O(1)$ vertices, we compute $\Delta(G)$ by brute force in $O(1)$ time. Otherwise Lemma 3 tells us that we can find in linear time a subset $A \subseteq V(G)$ with k portals S such that A has between $\frac{n}{k+1}$ and $\frac{nk}{k+1}$ vertices. We compute $\Delta_G(A, B)$ in $O(n \log^k n)$ expected time using Lemma 14. The graphs G_A and G_B can be constructed in $O(n)$ time once we have the shortest path trees for the portals $s \in S$, and we can then recursively compute $\Delta(G_A)$ and $\Delta(G_B)$. Finally, we use the relation (3) to compute $\Delta(G)$. Since k is a constant and $\frac{n}{k+1} \leq |A| \leq \frac{nk}{k+1}$, the recursion is balanced and the algorithm uses $O(n \log^{k+1} n)$ time. □

References

[1] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, pages 1–56. AMS, 1998.

[2] P. K. Agarwal, R. Klein, C. Knauer, S. Langerman, P. Morin, M. Sharir, and M. Soss. Computing the maximum detour and spanning ratio of 2- and 3-dimensional paths, trees, and cycles. *Discrete Comput. Geom.*, to appear.

[3] K. Barhum, O. Goldreich, and A. Shraibman. On approximating the average distance between points. In M. Charikar, K. Jansen, O. Reingold, and J. D. P. Rolim, editors, *APPROX-RANDOM*, volume 4627 of *LNCS*, pages 296–310. Springer, 2007.

[4] B. Ben-Moshe, B. K. Bhattacharya, and Q. Shi. Efficient algorithms for the weighted 2-center problem in a cactus graph. In *ISAAC 2005*, volume 3827 of *LNCS*, pages 693–703, 2005.

[5] T. Biedl. Lecture 12: Separators in partial k-trees (feb. 13, 2004). Lecture notes for the course CS 762 (Graph-Theoretic Algorithms), Winter 2004, University of Waterloo.

[6] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[7] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209:1–45, 1998.

[8] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, 1995.

[9] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete Comput. Geom.*, 22(4):547–567, 1999.

[10] S. Chaudhuri and C. D. Zaroliagis. Shortest paths in digraphs of small treewidth. Part I: Sequential algorithms. *Algorithmica*, 27:212–226, 2000.

[11] 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.

[12] V. Chepoi and S. Klavžar. Distances in benzenoid systems: further developments. *Discrete math.*, 192:27–39, 1998.

[13] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, Germany, 2nd edition edition, 2000.

[14] S. J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[15] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006.

[16] P. Indyk. Sublinear time algorithms for metric space problems. In *STOC '94*, pages 428–434, 1999.

[17] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

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

[19] G. Narasimhan and M. Smid. Approximating the stretch factor of Euclidean graphs. *SIAM J. Comput.*, 30:978–989, 2000.

[20] G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

[21] Q. Shi. Single facility location problems in partial k-trees, 2005. Poster at MITACS, Canada.

[22] N. Trinajstić. *Chemical Graph Theory*. CRC Press, 2nd edition, 1992.

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

[24] D. E. Willard. New data structures for orthogonal range queries. *SIAM J. Comput.*, 14:232–253, 1985.

[25] 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.