Balancing graph Voronoi diagrams with one more vertex

Guillaume Ducoffe

1Software and Complex Systems Engineering, National Institute for Research and Development in Informatics, Bucharest, Romania
2Faculty of Mathematics and Computer Science, University of Bucharest, Bucharest, Romania

Abstract
Let $G = (V, E)$ be a graph with unit-length edges and nonnegative costs assigned to its vertices. Given a list of pairwise different vertices $S = (s_1, s_2, \ldots, s_p)$, the prioritized Voronoi diagram of $G$ with respect to $S$ is the partition of $G$ in $p$ subsets $V_1, V_2, \ldots, V_p$ so that, for every $i$ with $1 \leq i \leq p$, a vertex $v$ is in $V_i$ if and only if $s_i$ is a closest vertex to $v$ in $S$ and there is no closest vertex to $v$ in $S$ within the subset $\{s_1, s_2, \ldots, s_{i-1}\}$. For every $i$ with $1 \leq i \leq p$, the load of vertex $s_i$ equals the sum of the costs of all vertices in $V_i$. The load of $S$ equals the maximum load of a vertex in $S$. We study the problem of adding one more vertex $v$ at the end of $S$ in order to minimize the load. This problem occurs in the context of optimally locating a new service facility (e.g., a school or a hospital) while taking into account already existing facilities, and with the goal of minimizing the maximum congestion at a site. There is a brute-force algorithm for solving this problem in $O(nm)$ time on $n$-vertex $m$-edge graphs. We prove a matching time lower bound–up to sub-polynomial factors–for the special case where $m = n^{1+\epsilon(1)}$ and $p = 1$, assuming the so called Hitting Set Conjecture of Abboud et al. On the positive side, we present simple linear-time algorithms for this problem on cliques, paths and cycles, and almost linear-time algorithms for trees, proper interval graphs and (assuming $p$ to be a constant) bounded-treewidth graphs.

KEYWORDS
facility location problems, graph algorithms, graph Voronoi diagrams, HS-hardness

1 | INTRODUCTION

For undefined graph terminology, see [7]. All graphs considered are finite, simple (they are loopless and have no multiple edges), connected and with unit-length edges. Let $G = (V, E)$ be a graph. Throughout the paper, let $n = |V|$ and $m = |E|$. The distance between two vertices $u$ and $v$ is the minimum number of edges on a $uv$-path. We denote this distance by $d_G(u, v)$, or simply $d(u, v)$ if $G$ is clear from the context. Given a vertex $v$ and a vertex subset $S = \{s_1, s_2, \ldots, s_p\}$, let also $d(v, S) = \min_{1 \leq i \leq p} d(v, s_i)$ denote the distance between $v$ and $S$. A Voronoi diagram of $G$ with respect to $S$ is any partition $V_1, V_2, \ldots, V_p$ of $V$ such that, for every $i$ with $1 \leq i \leq p$, for every vertex $v$ with $v \in V_i$, $d(v, s_i) = d(v, S)$. This terminology originates from Computational Geometry, where the Voronoi diagram is a well-studied data structure [3]. An example of graph Voronoi diagram is given in Figure 1. We stress that in general, a Voronoi diagram is not uniquely defined, due to the possible existence of vertices $v$ such that $d(v, s_i) = d(v, s_j) = d(v, S)$, for some $i$ and $j$ with $1 \leq i < j \leq p$.

To our best knowledge, Mehlhorn was amongst the first to introduce the concept of graph Voronoi diagrams, which he used in the design of an approximation algorithm for the STEINER TREE problem [24]. Erwig further pursued the study of these objects, of which he presented some new algorithmic applications [14]. Since then, graph Voronoi diagrams have appeared as a building block of even more algorithms [9,11,17]. In [26], Okabe et al. introduced generalized network Voronoi diagrams, of which graph Voronoi diagrams are special cases. Other variations such as furthest color Voronoi diagrams were considered in [21]. We only consider graph Voronoi diagrams in what follows.
Hakimi and Labbé introduced graph Voronoi diagrams, which they called Voronoi partitions, as a location-theoretic concept [19]. We build on their previous work and terminology. Specifically, given a vertex subset \( S = \{s_1, s_2, \ldots, s_p\} \), and a Voronoi diagram \( V_1, V_2, \ldots, V_p \) of \( G \) with respect to \( S \), we call \( S \) the vertices in \( S \), and we call \( S \) the territories the subsets \( V_i \), with \( 1 \leq i \leq p \), of this partition. For concreteness, we may regard the sites in \( S \) as locations of service facilities such as schools or hospitals. With this interpretation in mind, every territory is the service area of some facility. Several optimization criteria can be considered for these territories. Given some measure \( \mu : V \times 2^V \rightarrow \mathbb{R}_{\geq 0} \) and a nonnegative integer \( p \), the \( p \)-CENTER problem is the problem of minimizing \( \max_{1 \leq i \leq p} \mu(s_i, V) \) amongst all possible graph Voronoi diagrams with respect to \( p \) sites. For instance, if \( \mu(s_i, V) = \max_{v \in V} d(v, s_i) \), then this is the classical \( p \)-CENTER problem. Throughout the paper, let some function \( \pi : V \rightarrow \mathbb{R}_{\geq 0} \) assign nonnegative costs to the vertices. We call \( p \)-BALANCE the \( \mu_x \)-VORONOI \( p \)-CENTER problem for the measure \( \mu_x \) such that \( \mu_x(s_i, V) = \sum_{v \in V} \pi(v) \). For concreteness, we may regard \( \mu_x(s_i, V) \), which is the load of site \( s_i \), as the population size in a service area. We call our objective function \( \max_{1 \leq i \leq p} \mu_x(s_i, V) \) the load of the Voronoi diagram. We stress that minimizing the load is important in the context of critical facilities, such as hospitals. For instance, during the recent COVID-19 outbreak, the relation between hospital saturation and increased risk of death was documented [25]. The \( p \)-BALANCE problem is NP-hard if \( p \) is part of the input [20]. Interestingly, the NP-hardness reduction in [20] outputs instances of the problem such that if there exists a graph Voronoi diagram with load at most some value \( q \), then there always exist \( p \) sites minimizing the load such that every vertex is at minimal distance of exactly one site. This implies that the difficulty of this problem does not come from the non-uniqueness of a Voronoi diagram.

In practice, we must account for the location of existing service facilities in order to optimally locate a new one. This incremental approach has already been considered for the \( p \)-CENTER problem, under the name of “center selection problem” [22]. However, it does not seem that it has also been studied for the \( p \)-BALANCE problem. As a starter, let us consider the following problem (which is not quite yet what we study in the paper). We are given a subset \( S \) of sites, and the goal is to compute a new site \( v \), with \( v \not\in S \), and a Voronoi diagram with respect to \( S \cup \{v\} \) such that the load is minimized. This problem is NP-hard even if \( S \) is a singleton and \( G \) is a complete graph. Indeed, in this situation we can select any vertex \( v \not\in S \) as the new site, but the difficulty consists in computing an optimal graph Voronoi diagram. The load of any such diagram must be at least \( \pi(G)/2 \), where \( \pi(G) \) is the sum of the cost of all vertices. Moreover, deciding whether there exists a solution with load \( \pi(G)/2 \) is equivalent to the well-known PARTITION problem [15]. In order to circumvent this negative result, we show that it is enough to force the Voronoi diagram to be uniquely defined. More precisely, motivated by the practical need to minimally reorganize the diagram after inserting a new site, we follow Erwig’s suggestion to prioritize the sites [14]. That is, \( S = \{s_1, s_2, \ldots, s_p\} \) is a list, and every vertex \( v \) must be assigned to its first closest site in the list. Formally, for every \( i \) with \( 1 \leq i \leq p \), we have \( v \in V_i \) if and only if \( d(v, s_i) = d(v, S) \) and \( d(v, s_j) > d(v, S) \) for every \( j \) with \( 1 \leq j < i \). We call it the prioritized Voronoi diagram, or simply the Voronoi diagram of \( G \) with respect to \( S \), and we denote it in what follows by \( \text{Vor}(G, S) \). Other conditions could be also considered. For instance, one could add every vertex \( v \) to the territories of all \( s_i \) closest sites in \( S \), which is actually what Hakimi et al. did in [20]. However, having a vertex partition is important in some practical scenarios (e.g., a student attends to only one school). Furthermore, we stress that most results in the paper do not really depend on which condition we choose (a notable exception being the hardness result of Theorem 2.8).

For every \( i \) with \( 1 \leq i \leq p \), we denote by \( \ell_x(s_i, S) \) the load of site \( s_i \) in \( \text{Vor}(G, S) \). The load of \( \text{Vor}(G, S) \) is denoted by \( L_x(S) \).

To our best knowledge, the following problem has not been considered before:

**Problem 1** (BALANCED VERTEX).

**Input:** A graph \( G = (V, E) \); A cost function \( \pi : V \rightarrow \mathbb{R}_{\geq 0} \); A list \( S = (s_1, s_2, \ldots, s_p) \) of pairwise different vertices.

**Output:** A vertex \( v \in V \setminus S \) such that \( L_x(S + v) \) is minimized, where \( S + v = (s_1, s_2, \ldots, s_p, v) \).
Closest to our work is the one-round Voronoi game on graphs [4]. In the latter two-player game, Player 1 first selects a subset $S$ of sites with some fixed cardinality. Then, Player 2 must select a $k$-set $S'$ of new sites, with $S' \cap S = \emptyset$, in order to maximize the cumulative load of all $k$ sites in $S'$ in the graph Voronoi diagram with respect to $S' + S$ (the new sites in $S'$ have higher priority than the former sites in $S$). By contrast, in our problem we aim at minimizing the load at any site, and we also account for the load of all former sites in $S$ and not just for the load of the new site. Furthermore, the new site is given the least priority.

**Our contributions**

We initiate the complexity study of the BALANCED VERTEX problem.

On general $n$-vertex $m$-edge graphs, the problem can be solved in $\mathcal{O}(nm)$ time by brute force, simply by considering each vertex $v \notin S$ (Theorem 2.5). We prove a matching time lower bound—up to sub-polynomial factors—assuming the so-called Hitting Set Conjecture (Theorem 2.8). The latter was introduced in [1] by Abboud et al. as a way to explain our lack of progress toward computing faster either the radius or the median of a graph. Since then, to our best knowledge, broader implications of the Hitting Set Conjecture, for problems beyond radius and median computations, have not been investigated. Our work makes a step in this direction. A better-studied conjecture, that is implied by the Hitting Set Conjecture and by the Strong Exponential Time Hypothesis, is the so called Orthogonal Vectors Conjecture [28]. We leave as an open problem whether we can prove a quadratic time lower bound for the BALANCED VERTEX problem assuming this conjecture.

In the second and longest part of the paper, we break the quadratic barrier for BALANCED VERTEX on several graph classes. For that, we exploit some tree-like or path-like underlying structure of these graphs. Roughly, we adapt recent tools and frameworks for fast distance computation in these graphs, proving that they can also be used in order to compute the load $\ell_A(v, S + v)$ of every potential new site $v \notin S$. In doing so, we can solve BALANCED VERTEX, but only if there are constantly many sites in $S$. We detail this approach in Section 3.3, where it is applied to bounded treewidth graphs (Theorem 3.12). However, solving BALANCED VERTEX for an arbitrary (possibly nonconstant) number of sites looks more challenging. Some simple cases are first presented in Section 3.1. For $n$-node trees\(^1\), we combine a standard but rather intricate dynamic programming approach with centroid decomposition, so as to derive an $\mathcal{O}(n \log n)$-time algorithm (Theorem 3.8). The running time can be improved to $\mathcal{O}(n \log |S|)$ (see Remark 4). Our second main contribution in this part is for proper interval graphs, for which we get an $\mathcal{O}(m + n \log n)$-time algorithm (Theorem 3.16). This result is derived from prior works on distance labeling schemes for this class of graphs [16].

Results in the paper are summarized in Table 1. Some graph classes considered are related (e.g., paths are trees, and trees and cycles have bounded treewidth). We obtain faster algorithms for paths than for general trees (for trees and cycles than for bounded-treewidth graphs, resp.).

**Organization of the paper**

Basic results about the BALANCED VERTEX problem, along with a polynomial-time algorithm for solving it and a matching conditional time lower bound, are presented in Section 2. Faster algorithms for special graph classes are presented in Section 3. We conclude the paper in Section 4 with some open questions and perspectives.

\(^1\)Throughout the paper, we only use “node” for trees and tree-based data structures.
Additional notations and terminology

Let $G = (V, E)$ be a graph. The open neighborhood of a vertex $v$ is denoted in what follows by $N_G(v) = \{ u \in V | uv \in E \}$. We denote by $N_G[v] = N_G(v) \cup \{ v \}$ its closed neighborhood. The ball of center $v$ and radius $k$ is denoted by $N^k_G[v] = \{ u \in V | d_G(u, v) \leq k \}$. For every vertices $u$ and $v$, let $I_G(u, v) = \{ w \in V | d_G(u, v) = d_G(u, w) + d_G(w, v) \}$ denote the metric interval between $u$ and $v$. Let $W_G(u, v) = \{ w \in V | d_G(u, w) < d_G(v, w) \}$ contain all vertices closer to $u$ than to $v$. In the same way, for a vertex subset $S$ (a vertex list, resp.), let $N_G(S) = \bigcup_{s \in S} N_G(s) \setminus S$ and $N_G[S] = N_G(S) \cup S$ denote its open and closed neighborhoods, respectively. The subgraph induced by a subset $S$ is denoted by $G[S]$. For every vertex subset $S$ (vertex list, resp.), let $\text{Proj}_G(v, S) = \{ s \in S | d_G(v, s) = d_G(v, S) \}$ be the metric projection of a vertex $v$ on $S$. We sometimes omit the subscript if $G$ is clear from the context. A vertex list $S = (s_1, s_2, \ldots, s_p)$ such that $s_i \neq s_j$ for every $i, j$ with $1 \leq i < j \leq p$ is called in what follows a proper vertex list. We denote the territories $V_1, V_2, \ldots, V_p$ of $\text{Vor}(G, S)$ by $\mathbb{T}(s_1, S), \ldots, \mathbb{T}(s_p, S)$. Finally, given a cost function $\pi : V \to \mathbb{R}_{\geq 0}$, the cost of a vertex subset $X$ is defined as $\pi(X) = \sum_{x \in X} \pi(x)$.

2 | THE GENERAL CASE

We here consider the \textsc{balanced vertex} problem on general graphs. In Section 2.1, we prove a few simple properties of prioritized Voronoi diagrams. We derive from the latter a quadratic-time algorithm, in the number of edges, for solving \textsc{balanced vertex}. This algorithm is presented in Section 2.2. In Section 2.3, our quadratic running time is proved to be optimal--up to sub-polynomial factors--assuming the Hitting Set Conjecture.

2.1 | Basic properties

We start the section with some easy results on the territories in a Voronoi diagram. In particular, we prove the following relation between territories and metric intervals.

**Lemma 2.1.** Let $G = (V, E)$ be a graph and let $S = (s_1, s_2, \ldots, s_p)$ be a proper vertex list. For every vertex $v \in \mathbb{T}(s_i, S)$, with $1 \leq i \leq p$, we have $I(v, S) \subseteq \mathbb{T}(s_i, S)$.

**Proof.** Suppose by contradiction the existence of some vertex $u \in I(v, S) \setminus \mathbb{T}(s_i, S)$. Let $j$, with $1 \leq j \leq p$, be such that $u \in \mathbb{T}(s_j, S)$. In particular, $d(u, S) = d(u, s_j) \leq d(u, s_i)$. Note that $d(v, s_j) \leq d(v, u) + d(u, s_j) \leq d(v, u) + d(u, s_i) = d(v, s_i) = d(v, S)$. Therefore, we must have $d(u, s_j) = d(u, s_i) = d(u, S)$ and $d(v, s_j) = d(v, s_i) = d(v, S)$. However, because $v \in \mathbb{T}(s_i, S)$, $j > i$, thus contradicting that $u \in \mathbb{T}(s_j, S)$. \hfill \square

We immediately deduce from Lemma 2.1 the following result about territories:

**Corollary 2.2.** Let $G = (V, E)$ be a graph and let $S = (s_1, s_2, \ldots, s_p)$ be a proper vertex list. For every $i$ with $1 \leq i \leq p$, $\mathbb{T}(s_i, S)$ is a connected subset.

We often use in our proofs the following structural description of the last territory in a diagram. Similar but more complicated descriptions of the other territories can be derived from this result.

**Lemma 2.3.** Let $G = (V, E)$ be a graph and let $S = (s_1, s_2, \ldots, s_p)$ be a proper vertex list. For every vertex $v \notin S$, we have that $\mathbb{T}(v, S + v) = \bigcup_{1 \leq j \leq p} (W(v, s_j) \cap \mathbb{T}(s_j, S))$.

**Proof.** Let $j$ with $1 \leq j \leq p$ be fixed. For every vertex $u \in \mathbb{T}(s_j, S)$, we have that $u \in \mathbb{T}(v, S + v)$ if and only if $d(u, v) < d(u, S) = d(u, s_j)$. The vertices that satisfy this inequality are exactly those of $W(v, s_j)$. \hfill \square

2.2 | A quadratic-time algorithm

We present our quadratic-time algorithm for solving \textsc{balanced vertex}. It is based on the following observation that for every proper vertex list, we can compute the prioritized Voronoi diagram in linear time.

**Proposition 2.4.** Let $G = (V, E)$ be a graph and let $S = (s_1, s_2, \ldots, s_p)$ be a proper vertex list. We can compute $\text{Vor}(G, S)$ in $O(n + m)$ time.

**Proof.** We run a BFS on $S$ (i.e., we consider all vertices in $S$ as being one supervertex, on which we run a BFS). In doing so, we order the vertices $v \in V \setminus S$ by nondecreasing distance to $S$. Then, we compute $r(v)$, for every vertex
v, so that:

\[ r(v) = \begin{cases} 
  v, & \text{if } v \in S \\
  \min \{r(u) \mid u \in N(v) \text{ and } d(u, S) = d(v, S) - 1\} & \text{otherwise.}
\end{cases} \]

We prove by induction on \( d(v, S) \) that \( r(v) \) is the unique site of \( S \) such that \( v \in T(r(v), S) \). This is straightforward if \( v \in S \). Thus, from now on we assume that \( d(v, S) \geq 1 \). For every \( u \in N(v) \), if \( d(u, S) < d(v, S) \), then by the induction hypothesis we have \( r(u) \in \operatorname{Proj}(v, S) \). By Lemma 2.1, the unique site \( s_i \in S \) such that \( v \in T(s_i, S) \) is contained in \( \{r(u) \mid u \in N(v) \text{ and } d(u, S) = d(v, S) - 1\} \). Therefore, \( r(v) = s_i \).

In order to solve BALANCED VERTEX, it is sufficient to compute all prioritized Voronoi diagrams with one more site. We summarize our observations as follows.

**Theorem 2.5.** We can solve BALANCED VERTEX in \( O(nm) \) time.

**Proof.** For every vertex \( v \in V \setminus S \), we compute \( \operatorname{Vor}(G, S + v) \). By Proposition 2.4, this computation takes \( O(n + m) \) time. Furthermore, we can compute \( L_v(S + v) \) in additional \( O(n) \) time if we scan each territory of \( \operatorname{Vor}(G, S + v) \) sequentially. The running time follows since there are at most \( n \) vertices \( v \) to be considered.

\[ \square \]

### 2.3 HS-hardness

Our main technical contribution in this section is a conditional time lower bound for BALANCED VERTEX, which matches up to sub-polynomial factors–the quadratic running time of the simple algorithm presented in Theorem 2.5. Our hardness result is based on the following hypothesis.

**Conjecture 1** (The Hitting Set Conjecture [1]). There is no \( \varepsilon > 0 \) such that for all \( c \geq 1 \), there is an algorithm that given two lists \( A, B \) of \( n \) subsets of a universe \( U \) of size at most \( c \log n \), can decide in \( O(n^{2+\varepsilon}) \) time if there is a set in the first list that intersects every set in the second list, that is, a “hitting set”.

In what follows, we call the triple \((A, B, U)\) a HS-instance. Two HS-instances \((A, B, U)\) and \((A', B', U')\) are called equivalent if either both contain a hitting set or none of them does.

Insofar, the Hitting Set Conjecture has been understudied. We next present reductions from arbitrary HS-instances to equivalent ones with some more structure (Lemmas 2.6 and 2.7). The running times of these reductions are roughly linear in the cumulative cardinalities of all subsets in the input, which (since there are \( 2n \) such subsets and the universe has size \( O(\log n) \)) is in \( O(n \log n) \).

**Lemma 2.6.** Every HS-instance \((A, B, U)\) can be reduced in \( O(n \log n) \) time to some equivalent HS-instance \((A', B', U')\) with the following additional property: every set in \( B' \) intersects at most half of the sets in \( B' \).

**Proof.** Let \((A_0, B_0, U_0)\) and \((A_1, B_1, U_1)\) be disjoint copies of \((A, B, U)\). Let also \( x_0, x_1 \notin U_0 \cup U_1 \). We set:

\[
A' = \{a_0 \cup U_1 \mid a_0 \in A_0\} \cup \{a_1 \cup U_0 \mid a_1 \in A_1\},
\]

\[
B' = \{b_0 \cup \{x_0\} \mid b_0 \in B_0\} \cup \{b_1 \cup \{x_1\} \mid b_1 \in B_1\},
\]

\[
U' = U_0 \cup U_1 \cup \{x_0, x_1\}.
\]

The construction of \((A', B', U')\) takes \( O(n \log n) \) time.

We can bipartition \( B' \) in two sublists \( B'_0, B'_1 \) such that, for every \( i \in \{0, 1\} \), all sets in \( B'_i \) contain \( x_i \) and are contained in \( U_i \cup \{x_i\} \). Since \( U_0 \cup \{x_0\} \) and \( U_1 \cup \{x_1\} \) are disjoint, every set of \( B' \) intersects half of the sets of \( B' \). Furthermore, if \( a \in A \) is a hitting set for \((A, B, U)\), then \( a_0, a_1 \) be its copies in \( A_0, A_1 \). By construction, \( a_0 \cup U_1 \) and \( a_1 \cup U_0 \) are hitting sets for \((A', B', U')\). Conversely, let \( a' \in A' \) be a hitting set for \((A', B', U')\). By symmetry, we may assume to have \( a' = a_0 \cup U_1 \), for some set \( a_0 \in A_0 \). Since \( x_0 \notin a' \) and, for every set \( b_0 \in B_0, b_0 \cap U_1 = \emptyset \), we must have \( a_0 \) is a hitting set for \((A_0, B_0, U_0)\). In particular, there is also a hitting set for \((A, B, U)\).

**Lemma 2.7.** Let \( \alpha, \beta \) be arbitrary integers so that \( \alpha \geq 2 \). Every HS-instance \((A, B, U)\) can be reduced in \( O(n \log n) \) time to some equivalent HS-instance \((A', B', U')\) with the following additional properties: for some \( t = O(|U|) \), the universe \( U' \) has cardinality \( \alpha \cdot t + \beta \), and every set in \( A' \) has cardinality \( t \).

**Remark 1.** We only need to apply the following reduction with \( \alpha = 2, \beta = -1 \) for proving Theorem 2.8. However, the general case with \( \alpha, \beta \) arbitrary might be useful for future works on the Hitting Set Conjecture.
**Proof.** First, we reduce \((A, B, U)\) to some equivalent instance \((A_{\text{imp}}, B, U_{\text{imp}})\) so that all sets in \(A_{\text{imp}}\) have equal cardinality. Then, we reduce \((A_{\text{imp}}, B, U_{\text{imp}})\) to our final HS-instance \((A', B, U')\).

In order to construct \((A_{\text{imp}}, B, U_{\text{imp}})\), we start computing the extreme cardinalities \(\Delta = \max\{|a| \mid a \in A\}\) and \(\delta = \min\{|a| \mid a \in A\}\) of sets in \(A\). Let \(U_{\text{dummy}} = \{u_1, u_2, \ldots, u_{\Delta-\delta}\}\) be a universe of cardinality \(\Delta - \delta\) that is disjoint from \(U\). We set \(U_{\text{imp}} = U \cup U_{\text{dummy}}\). For every set \(a \in A\), we add in \(A_{\text{imp}}\) the set \(a \cup \{u_1, u_2, \ldots, u_{\Delta-|a|}\}\).

Every set in \(A_{\text{imp}}\) has cardinality \(\Delta\). In order to construct \((A', B', U')\), there are two different cases to be considered.

- **Case \(\alpha \cdot \Delta + \beta > |U_{\text{imp}}|\).** We complete \(U_{\text{imp}}\) with \(\alpha \cdot \Delta + \beta - |U_{\text{imp}}|\) new elements in order to create \(U'\). The two lists are unchanged (i.e., we set \(A' = A_{\text{imp}}, B' = B\)).
- **Case \(\alpha \cdot \Delta + \beta < |U_{\text{imp}}|\).** Let us write \(|U_{\text{imp}}| - \alpha \cdot \Delta - \beta = q \cdot (\alpha - 1) + r\) for some unique \(q \geq 0\) and \(r \in \{0, 1, \ldots, \alpha - 2\}\). Let \(X, Y\) be disjoint universes of respective cardinalities \(\alpha - 1 - r\) and \(q + 1\), both disjoint from \(U_{\text{imp}}\). We set \(U' = U_{\text{imp}} \cup X \cup Y\). Let \(A' = \{a \cup Y\} \in A_{\text{imp}}\), \(B' = B\). Doing so, every set in \(A'\) has cardinality equal to \(t = \Delta + q + 1\), while the universe \(U'\) has cardinality:

\[
|U_{\text{imp}}| + |X| + |Y| = \alpha \cdot \Delta + \beta + (|U_{\text{imp}}| - \alpha \cdot \Delta - \beta) + (q + 1) + (\alpha - 1 - r)
= \alpha \cdot \Delta + \beta + (q \cdot (\alpha - 1) + r) + q + \alpha - r
= \alpha \cdot \Delta + \beta + q \cdot \alpha + \alpha
= \alpha \cdot (\Delta + q + 1) + \beta
= \alpha \cdot t + \beta
\]

In both cases, our construction ensures that every set in \(A'\) has cardinality \(\frac{|U'|-\beta}{\alpha}\).

We are now ready to combine both reductions in order to prove our main result in this part.

**Theorem 2.8.** *Assuming Conjecture 1, there is no \(\varepsilon > 0\) such that for all \(c \geq 1\), there is an algorithm that solves Balanced Vertex in \(\mathcal{O}(n^{2+c})\) time on the \(n\)-vertex graphs with at most \(cn \log n\) edges.*

**Proof.** Let \((A, B, U)\) be any HS-instance with the following properties:

1. every set of \(B\) intersects at most half of the sets in \(B\);
2. every set of \(A\) has cardinality \(\frac{|U|+1}{2}\).

We stress that every HS-instance can be reduced to an equivalent one with these properties by applying the reductions of Lemmas 2.6 and 2.7, in this order. Intuitively, we identify the subsets of \(A \cup B\) and the elements of \(U\) with vertices in some graph \(G\) (to be defined later). We further define a unique site \(s\), which is close to every vertex in \(A\) but distant to every vertex in \(B\). Therefore, in order to minimize the load, one should pick a new site which is closer than \(s\) to every vertex in \(B\). At first glance, one may think about picking some new site in \(U\). However, the vertices in \(U\) are closer than \(s\) to far too many vertices, thus making the load increase. The first property above ensures that no vertex of \(B\) should be picked either. Therefore, we are left picking (if any) a vertex of \(A\) that is close to every vertex of \(B\), a.k.a., a hitting set. Note that in general, the exact load should also depend on the size of \(U\) and the cardinality of the hitting set. The second property above ensures that it is not the case for our reduction.

For convenience, in what follows let \(t = \frac{|U|+1}{2}\) denote the cardinality of every set in \(A\). We may further assume that \(t > 2\) (otherwise, we can detect a hitting set in \(\mathcal{O}(n)\) time because there are \(\mathcal{O}(1)\) distinct sets in \(A\) and that \(n\) is above some large enough constant. We construct a graph \(G = (V, E)\) with \(2n + |U| + 3 = 2(n + t + 1)\) vertices, as follows:

- \(V = A \cup B \cup U \cup \{s, x, y\}\);
- \(U \cup \{x, y\}\) is a clique;
- \(N_G(s) = \{x, y\}\);
- for every set \(a \in A\), \(N_G(a) = \{y\} \cup \{u \in U | u \in a\}\);
- for every set \(b \in B\), \(N_G(b) = \{u \in U | u \in b\}\).

See Figure 2 for an illustration. Let \(S = (s)\) and let \(x : V \to \mathbb{R}_{\geq 0}\) be the all-one function (i.e., \(x(v) = 1\) for every vertex \(v\)). Given a vertex \(v \neq s\), let \(S_v = (s, v)\). By Lemma 2.3, \(\overline{T}(v, S_v) = W_G(v, s)\). We now compute the load of \(S_v\) for every vertex \(v \neq s\):

- **Case \(v = x\).** We have \(W(x, s) = U \cup \{x\} \cup B\). Then, \(L_\epsilon(S_v) = L_\epsilon(x, S_v) = n + 1 + |U| = n + 2t\).
In this section, we break the quadratic barrier for BALANCED VERTEX on various graph classes. We obtain a linear-time algorithm only for simple topologies, see Section 3.1. Nevertheless, we achieve an almost linear-time computation on trees (Section 3.2), proper interval graphs (Section 3.4) and bounded-treewidth graphs with a constant number of sites (Section 3.3).

One of the reviewers observed that for BALANCED VERTEX, we only need to consider vertices \( v \not\in S \) that reduce all territories \( T(s_i, S) \) of maximum load. This observation does help in reducing the number of vertices to be considered in practice. However, it cannot be applied if there is only one site in \( S \) (a case which is already hard, see Theorem 2.8).

3 | ALMOST LINEAR-TIME ALGORITHMS

In this section, we break the quadratic barrier for BALANCED VERTEX on various graph classes. We obtain a linear-time algorithm only for simple topologies, see Section 3.1. Nevertheless, we achieve an almost linear-time computation on trees (Section 3.2), proper interval graphs (Section 3.4) and bounded-treewidth graphs with a constant number of sites (Section 3.3).

One of the reviewers observed that for BALANCED VERTEX, we only need to consider vertices \( v \not\in S \) that reduce all territories \( T(s_i, S) \) of maximum load. This observation does help in reducing the number of vertices to be considered in practice. However, it cannot be applied if there is only one site in \( S \) (a case which is already hard, see Theorem 2.8).

3.1 | Simple cases

First, we present a complete dichotomy of the parameterized complexity of BALANCED VERTEX if the parameter is the diameter of the graph. The graphs of unit diameter are exactly the complete graphs. Given a proper vertex list \( S \) in such graph, every vertex of \( V \setminus S \) must belong to the territory of the site in \( S \) with the highest priority. We deduce from this observation a simple linear-time algorithm for BALANCED VERTEX on complete graphs.

**Lemma 3.1.** We can solve BALANCED VERTEX in \( \mathcal{O}(n) \) time if \( G \) is a clique.
Proof. Let $\Lambda = \pi(G) - \sum_{i=2}^{p} \pi(s_i)$. It suffices to output a vertex $v \notin S$ such that $\max \{\pi(v), \Lambda - \pi(v)\}$ is minimized. Indeed, since $G$ is a clique we have the following for every vertex $v \notin S$: $T(s_1, S + v) = V \setminus \{s_2, s_3, \ldots, s_p, v\} = T(s_1, S) \setminus \{v\}; T(s_i, S + v) = \{s_i\} = T(s_i, S)$ for every $i$ with $2 \leq i \leq p$; $T(v, S + v) = \{v\}$. In particular, we have $L_\varepsilon(S + v) = \max(\{\pi(s_i) | 2 \leq i \leq p\} \cup \{\pi(v), \Lambda - \pi(v)\})$.

We now generalize the simple idea behind Lemma 3.1 to the graphs of diameter two.

**Lemma 3.2.** We can solve BALANCED VERTEX in $O(n + m)$ time if $G$ has diameter two.

**Proof.** We compute $\text{Vor}(G, S)$, which by Proposition 2.4 we can do in $O(n + m)$ time. Doing so, we can compute in $O(n)$ time the values $\ell_x(s_i, S)$, for every $i$ with $1 \leq i \leq p$. We can also compute in $O(n)$ time, for every $i$ with $1 \leq i \leq p$, the value $\mu_i$ such that:

$$
\mu_i = \begin{cases} 
\max \{\ell_x(s_2, S), \ell_x(s_3, S), \ldots, \ell_x(s_p, S)\} & \text{if } i = 1 \\
\max \{\ell_x(s_j, S) | j \in \{2, 3, \ldots, p\} \setminus \{i\}\} & \text{otherwise}.
\end{cases}
$$

For every vertex $v \notin S$, let $s_i$ be the unique site of $S$ such that $v \in T(s_i, S)$. Note that we can compute $s_i$ from $\text{Vor}(G, S)$ in $O(1)$ time. We compute the value $\lambda(v)$ such that:

$$
\lambda(v) = \begin{cases} 
\max \{\pi(N(v) \setminus N[S]) + \pi(v), \ell_x(s_1, S) - \pi(N(v) \setminus N[S]) - \pi(v), \mu_1\} & \text{if } i = 1 \\
\max \{\pi(N(v) \setminus N[S]) + \pi(v), \ell_x(s_1, S) - \pi(N(v) \setminus N[S]), \ell_x(s_i, S) - \pi(v), \mu_i\} & \text{otherwise}.
\end{cases}
$$

Since we precomputed all values $\mu_1, \mu_2, \ldots, \mu_p$, we can compute the values $\lambda(v)$, for every $v \notin S$, in $O(n + m)$ time, simply by scanning the neighbourhoods of every vertex. Finally, we output a vertex $v \notin S$ such that $\lambda(v)$ is minimized. Indeed, we claim that $\lambda(v) = L_\varepsilon(S + v)$. This is because we assume the diameter of $G$ to be two, and therefore $T(s_j, S) \subseteq N[s_j]$ for every $j$ with $2 \leq j \leq p$. In particular, we have:

$$
W_G(v, s_j) \cap T(s_j, S) = \begin{cases} 
\{v\} & \text{if } v \in T(s_j, S) \\
\emptyset & \text{otherwise}.
\end{cases}
$$

In the former case, $\ell_x(s_j, S) = \ell_x(s_j, S) - \pi(v)$, while in the latter case $\ell_x(s_j, S + v) = \ell_x(s_j, S)$. Furthermore, $T(v, S + v) = \{v\} \cup (N(v) \setminus N[S])$, and so $\ell_x(v, S + v) = \pi(N(v) \setminus N[S]) + \pi(v)$. Finally,

$$
W_G(v, s_1) \cap T(s_1, S) = \begin{cases} 
\{v\} \cup (N(v) \setminus N[S]) & \text{if } v \in T(s_1, S) \\
N(v) \setminus N[S] & \text{otherwise}.
\end{cases}
$$

Altogether combined, we obtain as claimed that $\lambda(v) = L_\varepsilon(S + v)$.

**Remark 3.** If $G$ is a clique to which we remove $k$ edges, for some constant $k$, then the runtime of Lemma 3.2 can be improved to $O(kn)$ as follows. First, in order to compute $T(s_2, S), T(s_3, S), \ldots, T(s_p, S)$, we scan the neighbours of every vertex $u \in V \setminus (N[s_1] \cup S)$ and we compute the least $i$ such that $s_i, u$ are adjacent (if it exists). There are at most $k$ vertices in $V \setminus N[s_1]$. Therefore, this can be done in $O(kn)$ time. Furthermore, in additional $O(n)$ time we can also compute $\text{Vor}(G, S)$. Then, given $\text{Vor}(G, S)$, we can continue as in the proof of Lemma 3.2 and compute all values $\mu_1, \mu_2, \ldots, \mu_p$ in $O(n)$ time. In order to compute the values $\ell_x(v, S + v)$ for every $v \notin S$ (and so, complete our algorithm), as explained in the proof of Lemma 3.2, it now suffices to compute $N(v) \setminus N[S]$ for every $v \notin S$. For that, let $U = T(s_1, S) \setminus N[s_1]$ be the set of vertices of $V \setminus S$ with no neighbour in $S$. Observe that, for every $v \notin S$, $N(v) \setminus N[S] = N(v) \setminus U$. In particular, the subset $N(v) \cap U$, for every $v \notin S$, can be computed in $O(|U|) = O(k)$ time.

We cannot extend Lemmas 3.1 and 3.2 any further because the graph constructed in the proof of Theorem 2.8 has diameter three. Therefore, we can solve BALANCED VERTEX in linear time if the diameter is at most two, while we cannot solve this problem in truly subquadratic time if the diameter is at least three.

Next, we consider paths and cycles. The problem becomes local, in the following sense. Let us subdivide a path $G$ in maximal subpaths whose every end is either a site of $S$ or one of the two ends of $G$. See Figure 3 for an illustration. Doing so, we can prove that there are at most two sites of $S$ whose territory may be modified upon inserting a new site $v$, namely, those at the ends of the maximal subpath containing $v$. In particular, we can compute the load $L_\varepsilon(S + v)$, for every node $v \notin S$, by applying to every maximal subpath a classical partial sum trick for vector problems.

**Lemma 3.3.** We can solve BALANCED VERTEX in $O(n)$ time if $G$ is a path.

**Proof.** We first describe our algorithm. See Figure 3 for an illustration.
Subdivision of a path

1. We compute $\text{Vor}(G, S)$ and the loads $\lambda_x(s_j, S)$, for every $j$ with $1 \leq j \leq p$.
2. Let $C_1, C_2, \ldots, C_q$ be the connected components of $G \setminus S$. Note that we have $q \leq p + 1$. For every $i$ with $1 \leq i \leq q$, let $S_i = N(C_i)$, and let $P_i$ be the subpath induced by $N[C_i] = C_i \cup S_i$. We stress that $|S_i| = 2$, except maybe if $i \in \{1, q\}$ for which we may have $|S_i| = 1$.
3. We consider each subpath $P_i$, with $1 \leq i \leq q$, sequentially. In what follows, let $P_i = (w^i_0, w^i_1, \ldots, w^i_{t_i-1})$.
   - We compute $\lambda(i, j) = \sum_{k=0}^{j} \pi(w^i_k)$, for every $k$ with $0 \leq k < t_i$.
   - Then, there are two different cases to be considered.
   a. Case $|S_i| = 1$. Either $i = 1$, $w^i_0 \notin S_i$ or $i = q$, $w^i_{t_i-1} \notin S_q$. We observe that both subcases are symmetric up to reverting the path $G$. Therefore, we assume without loss of generality that $i = 1$, $w^i_0 \notin S_1$. In this situation, $w^i_0$ is the leftmost end of path $G$, and $S_1 = \{w^i_{t_i-1}\}$. For every $j$ with $0 \leq j < t_1 - 1$, let $j_R = \left\lfloor \frac{t_1 - j + 1}{2} \right\rfloor - 1$. For every $v \in C_1$, we compute:
      \[
      \Lambda(v) = \max\{\Lambda_1, \lambda(1, j_R), \lambda_x(w^i_0, S) - \lambda(1, j_R)\}
      \]
   b. Case $|S_i| = 2$. We first compute $\bar{j}_{\text{lim}} = \max\{|j| 0 \leq j < t_i, w^j \in \mathcal{T}(w^i_0, S)\}$. Note that if $t_i$ is even, then we always have $\bar{j}_{\text{lim}} = w^i_{t_i/2-1}$. However, if $t_i$ is odd, then $\bar{j}_{\text{lim}} \in \{w^i_{(t_i-3)/2}, w^i_{(t_i-1)/2}\}$, and the exact value depends on the respective positions of $w^i_0, w^i_{t_i-1}$ in $S$. Then, we consider each node of $C_i$ sequentially. In what follows, let $j$ with $0 < j < t_i - 1$. Let $j_L = \left\lfloor \frac{j}{2} \right\rfloor + 1$, and let $j_R = \left\lfloor \frac{t_i - j + 1}{2} \right\rfloor - 1$.
      For every $v \in C_i$, we compute:
      \[
      \Lambda(v) = \max\{\Lambda_i, \lambda(i, j_R) - \lambda(i, j_L) - 1, \lambda_x(w^i_0, S) - \lambda(i, j_{\text{lim}}) + \lambda(i, j_R) - 1, \lambda_x(w^i_{t_i-1}, S) - \lambda(i, j_{\text{lim}}) + \lambda(i, j_R)\}
      \]
4. Finally, we output a node $v \notin S$ such that $\Lambda(v)$ is minimized.

**Complexity.** By Proposition 2.4, Step 1 of the algorithm can be done in $O(n)$ time. Step 2 can be also done in $O(n)$ time. Then, for every $i$ with $1 \leq i \leq q$, in order to compute the value $\Lambda_i$, it suffices to totally order the vertices of $S$ by non-increasing load, and to search for the first such vertex that is not in $S_i$. For that, it is sufficient to only consider the max $|S_i| + 1, \ldots, p$ nodes of $S$ with maximum load. Since we have $|S_i| \leq 2$ for every $i$ with $1 \leq i \leq q$, we can compute all values $\Lambda_1, \Lambda_2, \ldots, \Lambda_q$ in $O(n)$ time, simply by keeping track of the max $3, \ldots, p$ nodes of $S$ with maximum load. Let $i$ with $1 \leq i \leq q$ be fixed. By dynamic programming, we compute in $O(t_i)$ time the values $\lambda(i, j)$ for every $j$ with $0 \leq j \leq t_i - 1$. Then, we can compute in $O(t_i)$ time the values $\Lambda(v)$, for every $v \in C_i$, simply by scanning once the path $P_i$ and performing $O(1)$ operations for each node. Since we have $\sum_{i=1}^{q} t_i = O(n)$, the total running time for Step 3 is also in $O(n)$.

**Correctness.** Let $i$ be such that $1 \leq i \leq q$, and let $v \in C_i$ be arbitrary. We prove in what follows that $\Lambda(v) = L_x(S + v)$. First we claim that $C_i \cup \emptyset \subseteq \left\{ \mathcal{T}(s, S) | s \in S_i \right\}$. Indeed, suppose by contradiction that some node $u \in C_i$ belongs to $\mathcal{T}(s', S)$, for some $s' \in S \setminus S_i$. The unique $us'$-path intersects $S_i$, which by Lemma 2.1 implies that $\mathcal{T}(s', S) \cap S_i \neq \emptyset$, a contradiction. In the same way, we have $\mathcal{T}(v, S + v) \subseteq C_i \subseteq \left\{ \mathcal{T}(x, S + v) | x \in S_i \cup \{v\} \right\}$. This implies that for every $s \in S_i$, we have $\mathcal{T}(s, S + v) \subseteq \mathcal{T}(s, S) \setminus C_i$, while for every $s' \in S \setminus S_i$, we have $\mathcal{T}(s', S) = \mathcal{T}(s', S + v)$. Hence, we obtain:
\[
L_x(S + v) = \max(\{\mathcal{E}_x(s', S + v) | s' \in S_i \} \cup \{\mathcal{E}_x(x, S + v) | x \in S_i \cup \{v\}\})
\]
\[
= \max(\left\{\Lambda_i \cup \left\{\mathcal{E}_x(x, S + v) | x \in S_i \cup \{v\}\right\}\right\})
\]
Clearly, $\mathcal{T}(v, S + v) = \{u \in C_i \mid d(u, v) < d(u, S_j)\}$. Let $j$ with $0 \leq j \leq t_1 - 1$ and $v = w_j$. There are two cases to be considered:

- **Case $|S| = 1$.** By symmetry, we may further assume that $i = 1$, $w_0 \not\in S_1$. Then, $C_1 \subseteq \mathcal{T}(w_{t_1 - 1}, S)$, and by Lemma 2.3 we have $\mathcal{T}(v, S + v) = \mathcal{W}(v, w_{t_1 - 1}) = \{w_0, w_1, \ldots, w_{t_1 - 1}\}$. In particular, $\ell_x(v, S + v) = \sum_{k=0}^{t_1} \pi(w_k) = \lambda(1, jR)$, and so $\ell_x(w_{t_1 - 1}, S + v) = \ell_x(w_{t_1 - 1}, S) - \lambda(1, jR)$.

- **Case $|S| = 2$.** In this situation, $\mathcal{T}(v, S + v) = \{w_{j_1}, w_{j_1 + 1}, \ldots, w_{j_2}\}$. Therefore, $\ell_x(v, S + v) = \sum_{k=j_1}^{j_2} \pi(w_k) = \lambda(i, jR) - \sum_{k=j_1}^{j_2-1} \pi(w_k) = \lambda(i, jR) - \lambda(i, jL - 1)$. By the definition of $j_{\text{lim}}$, we have $\mathcal{T}(w_{j_1}, S) \cap C_i = \{w_1, \ldots, w_{j_{\text{lim}}}^i\}$ while $\mathcal{T}(w_{j_1 - 1}, S) \cap C_i = \{w_{j_{\text{lim}} + 1}^i, \ldots, w_{j_2}^i\}$.

Furthermore, we claim that $j_{L} - 1 \leq j_{\text{lim}} \leq j_{R}$. Indeed, let us first assume that $j \leq j_{\text{lim}}$. In particular, $j_{L} - 1 \leq j \leq j_{\text{lim}}$. For every $k$ with $j \leq k \leq j_{\text{lim}}$, we have $d(v, w_k^i) < d(w_{k-1}^i, w_k^i)$, which follows from $d(w_0^i, w_k^i) \leq d(w_{k-1}^i, w_k^i)$ and $v$ is on the $w_0^i$-path. Therefore, $j_{\text{lim}} \leq j_{L}$. From now on, we assume that $j_{\text{lim}} < j$. In particular, $j_{\text{lim}} < j_{R}$. For every $k$ with $j_{\text{lim}} < k < j$, we have $d(v, w_k^i) < d(w_0^i, w_k^i)$, which follows from $d(w_0^i, w_k^i) \geq d(w_{k-1}^i, w_k^i)$ and $v$ is on the $w_0^i$-path. Therefore, $j_1 \leq j_{\text{lim}} + 1$.

As a result,

$$
\ell_x(w_0^i, S + v) = \ell_x(w_0^i, S) - \sum_{k=j_1}^{j_{\text{lim}}} \pi(w_k) \\
= \ell_x(w_0^i, S) - (\lambda(i, j_{\text{lim}}^i) - \lambda(i, j_L - 1))
$$

and

$$
\ell_x(w_{j_1 - 1}^i, S + v) = \ell_x(w_{j_1 - 1}^i, S) - \sum_{k=j_{\text{lim}} + 1}^{j_2} \pi(w_k) \\
= \ell_x(w_{j_1 - 1}^i, S) - (\lambda(i, j_R) - \lambda(i, j_{\text{lim}}^i))
$$

Overall, we proved that in both cases $\lambda(v) = L_x(S + v)$. Therefore, the algorithm is correct.

Up to slight adjustments (e.g., in the special case where $|S| = 1$), we can directly apply to cycles our techniques for paths. We prove it below.

**Lemma 3.4.** We can solve Balanced Vertex in $O(n)$ time if $G$ is a cycle.

**Proof.** Let us first assume that $p = |S| = 1$. Let $(v_0, v_1, \ldots, v_{n-1})$ be a cyclic ordering of the vertices, and let us assume without loss of generality that $S = (v_0)$. We compute by dynamic programming the values $\lambda(j) = \sum_{k=0}^{n} \pi(v_k)$, for every $j$ with $1 \leq j < n$. It takes $O(n)$ time. By symmetry, we only consider in what follows the vertices $v_i$ with $0 < i \leq n/2$. In this situation, by Lemma 2.3, $\mathcal{T}(v_i, S + v_i) = \mathcal{W}(v_i, v_0) = \{v_{1\lfloor n/2 \rfloor} + 1, v_{1\lfloor n/2 \rfloor} + 2, \ldots, v_{\lfloor (n-1)/2 \rfloor + 1}\}$.

Let $\ell_i = \lambda\left(\left\lfloor \frac{n-i}{2} \right\rfloor \right) - 1 - \lambda\left(\left\lfloor \frac{i}{2} \right\rfloor \right)$. Note that $L_x(S + v_i) = \max\{\ell_i, \pi(G) - \ell_i\}$. Therefore, it suffices to output a vertex $v_i$ with $1 \leq i \leq n/2$ and such that $\max\{\ell_i, \pi(G) - \ell_i\}$ is minimized. The running time is in $O(n)$.

From now on, we assume that $p = |S| > 1$. We compute $\text{Vor}(G, S)$ and the loads $\ell_x(s_j, S)$, for every $j$ with $1 \leq j \leq p$. By Proposition 2.4, this can be done in $O(n)$ time. Let $C_1, C_2, \ldots, C_q$ be the connected components of $G \setminus S$. If $q = 1$, then $p = 2$ and the two vertices of $S$ form an edge of the cycle. In this special case we remove the edge induced by $S$ and then we apply Lemma 3.3 to the resulting path. Thus from now on we assume $q > 1$. For every $i$ with $1 \leq i \leq q$, let $S_i = N(C_i)$, and let $P_i$ be the subpath induced by $N(C_i) \cap S_i$. Note that the two vertices of $S_i$ are the two ends of $P_i$. We are done applying the exact same procedure for each subpath $P_i$ as in Lemma 3.3 (Step 3 of the algorithm for paths). The running time is in $O(n)$. Correctness follows from the exact same arguments as in the proof of Lemma 3.3, and it is even simpler because the case $|S| = 1$ cannot occur.

3.2 Trees

We now address the case of general trees. Unlike for paths, the number of former sites in $S$ whose territory is modified after insertion of a new site may be arbitrarily large. See Figure 4 for an illustration. Fortunately, we can use centroid decomposition.
in order to compute, for every \( v \not\in S \), a set of \( \mathcal{O}(\log n) \) nodes that are on the paths between \( v \) and the territories of all but \( \mathcal{O}(\log n) \) former sites in \( S \). More precisely, given a tree \( T \), we can summarize our strategy as follows:

1. We compute a centroid, that is, a node \( c \) such that every connected component of \( T \setminus \{c\} \) contains at most \( n/2 \) nodes.
2. We process the unique site \( s_i \in S \) such that \( c \in T(s_i, S) \) (see Lemma 3.5);
3. We process all sites \( s_j \) whose territories are on different connected components of \( T \setminus \{c\} \) than \( v \) (see Lemmas 3.6 and 3.7);
4. Finally, we recursively apply our strategy in order to process all remaining sites whose territories are on the same connected component of \( T \setminus \{c\} \) as \( v \).

**Lemma 3.5.** Let \( s \) be an arbitrary node in a tree \( T = (V, E) \). Let \( \pi : V \rightarrow \mathbb{R}_{\geq 0} \) be any cost function. We can compute \( a(v) = \pi(W(v, s)) \), for every node \( v \in V \), in \( \mathcal{O}(n) \) time.

**Proof.** We root \( T \) at \( s \). Then, we preprocess \( T \) in \( \mathcal{O}(n) \) time so that the following holds for every node \( v \):

- We store \( d_T(v, s) \)—the level of node \( v \);
- We store \( \pi(T_c) \)—the cost of all nodes in the subtree \( T_c \) rooted at \( v \);
- For every \( i \) with \( 0 \leq i \leq d_T(v, s) \), we can compute the \( i \)-th level ancestor of \( v \) in \( \mathcal{O}(1) \) time [5].

We consider each node \( v \in V \) sequentially. Let \( d_T(v, s) = 2q + \epsilon \), with \( \epsilon \in \{0, 1\} \). In this situation, \( W(v, s) = T_x \), with \( x \) the \( (q + 1) \)-th level ancestor of \( v \). In particular, we can compute \( x \), and so, \( \pi(T_x) = a(v) \), in \( \mathcal{O}(1) \) time. Overall, the total running time is in \( \mathcal{O}(n) \).

Let \( c \) be an arbitrary cut-vertex in a tree \( T \) (in our Theorem 3.8, this node \( c \) will always be a centroid of the tree). The next two lemmas allow us to process, for all potential locations \( v \) for a new site, all former sites in \( S \) whose territory is disconnected from \( v \) in \( T \setminus \{c\} \). In particular, using Lemma 3.6 we can compute the cost of all nodes in those territories that are closer to \( v \) than to \( S \) (and so, that would end up in \( v \)'s territory). Using Lemma 3.7, we can compute the maximum load of those sites in the Voronoi diagram of \( T \) with respect to \( S + v \).

**Lemma 3.6.** Let \( c \) be an internal node in a tree \( T = (V, E) \), and let \( S \) be a proper vertex list. We can compute \( \beta(v) = \sum \{\pi(x) \mid x \in V, \ c \in I(v, x) \text{ and } d(x, v) < d(x, S)\} \), for every node \( v \in V \setminus \{c\} \), in \( \mathcal{O}(n) \) time.

**Proof.** Let us first describe our algorithm:

1. In \( \mathcal{O}(n) \) time, we compute \( d(x, S) \) and \( d(x, c) \) for every node \( x \).
2. For every \( i \) with \( 0 \leq i \leq n \), we compute \( a[i] = \sum \{\pi(x) \mid x \in V, \ d(x, c) + i < d(x, S)\} \). For that, we start computing \( b[j] = \sum \{\pi(x) \mid x \in V, \ d(x, S) = d(x, c) + j\} \), for every \( j \) with \( 0 \leq j \leq n \), simply by scanning each node \( x \) once. Since \( a[i] = \sum_{j=0} b[j] \), we are done in additional \( \mathcal{O}(n) \) time by dynamic programming.
3. We consider each connected component \( C_1, C_2, \ldots, C_q \) of \( T \setminus \{c\} \) sequentially. Let \( k \) with \( 1 \leq k \leq q \) be fixed. For every \( i \) with \( 0 \leq i \leq |C_k| \), we compute \( a_k[i] = \sum \{\pi(x) \mid x \in C_k, \ d(x, c) + i < d(x, S)\} \). This can be done in \( \mathcal{O}(|C_k|) \) time by using the exact same approach as for the prior step of the algorithm. Finally, for
every node \( v \in C_k \), we set:

\[
\beta(v) = a[d(v, c)] - a_k[d(v, c)].
\]

This can be computed in \( \mathcal{O}(1) \) time per node of \( C_k \), and therefore, in \( \mathcal{O}(|C_k|) \) time.

The total running time of the algorithm is in \( \mathcal{O}(n) \). In order to prove correctness of the algorithm, we let \( v \in V \setminus \{c\} \) be arbitrary. We have \( v \in C_k \) for some \( k \) with \( 1 \leq k \leq q \). For every node \( x \), we have \( c \in I(x, v) \) if and only if \( x \not\in C_k \). Furthermore, in this situation we have \( d(x, v) < d(x, S) \) if and only if \( d(x, c) + d(c, v) < d(x, S) \). The sum of the costs of all nodes \( x \) such that \( d(x, c) + d(c, v) < d(x, S) \) is exactly \( a[d(v, c)] \). In order to compute \( \beta(v) \) from \( a[d(v, c)] \), we need to remove from the latter the sum of the costs of all such nodes \( x \) with \( x \in C_k \), that is exactly \( a_k[d(v, c)] \).

\[
\lambda(v) = \frac{1}{\gamma(v)} = \frac{1}{\max\{\ell_x(s, S + v)\mid s \in S, \ c \not\in \mathbb{T}(s, S), \ c \in I(v, s)\}}.\]

Intuitively, \( \ell_x(s) \) would be the load of \( s \) upon insertion of a new node \( v \), with \( v \not\in C(s) \) and \( d(v, c) = j \). In particular, the following holds for every node \( v \), with \( v \neq c \) and \( d(v, c) = j \): \( \gamma(v) = \max\{\ell_x(s)\mid s \in S \setminus C(v)\} \). Therefore, in order to compute \( \gamma(v) \), for every node \( v \) with \( v \neq c \), it suffices to compute the following information for every \( j \) with \( 0 \leq j \leq n \):

- a node \( s_j \) maximizing \( \ell_x(s_j) \) within \( S' \);
- a node \( s'_j \) maximizing \( \ell_x(s'_j) \) within \( S' \setminus C(s_j) \).

For that, we first compute \( \text{Vor}(T, S) \), and the distances \( d(x, S) \) and \( d(x, c) \) for every node \( x \). By Proposition 2.4, this can be done in \( \mathcal{O}(n) \) time. We order the nodes \( x \), with \( x \neq c \), by nondecreasing values \( d(x, S) - d(x, c) \), which can also be done in \( \mathcal{O}(n) \) time by using counting sort. For every node \( s \) within \( S' \), let \( \lambda(s) = \ell_x(s) \). Intuitively, at the end of each step \( j \) of our main procedure (presented next), we must have \( \lambda(s) = \ell_x(s) \). We now proceed as follows for every \( j \) with \( 0 \leq j \leq n \):

- Case \( j = 0 \) (first step). Let \( X_0 = \{x \in V \setminus \{c\} \mid d(x, S) > d(x, c)\} \). We consider each node \( x \in X_0 \) sequentially. Let \( s \in S \) be such that \( x \in \mathbb{T}(s, S) \). If \( c \not\in \mathbb{T}(s, S) \), then we set \( \lambda(s) = \lambda(s) - \pi(x) \). Finally, we compute a vertex \( s_0 \) (\( s'_0 \), resp.) that maximizes \( \lambda(s_0) \) (\( \lambda(s'_0) \), resp.) within \( S' \) (within \( S' \setminus C(s_0) \), resp.).
- Case \( j > 0 \). Let \( X_j = \{x \in V \setminus \{c\} \mid d(x, S) - d(x, c) = j\} \). We scan each node in \( X_j \) computing along the way a subset \( S_j \) of relevant nodes in \( S' \). Initially, we set \( S_j = \emptyset \). We consider each node \( x \in X_j \) sequentially. Let \( s \in S \) be such that \( x \in \mathbb{T}(s, S) \). If \( c \not\in \mathbb{T}(s, S) \), then we add \( s \) in \( S_j \), then we set \( \lambda(s) = \lambda(s) + \pi(x) \). Finally, we compute a vertex \( s_j \) (\( s'_j \), resp.) that maximizes \( \lambda(s_j) \) (\( \lambda(s'_j) \), resp.) within \( S_j \cup \{s_{j-1}\} \) (within \( S_j \cup \{s_{j-1}, s'_{j-1}\} \setminus C(s_j) \), resp.).

Step 0 can be done in \( \mathcal{O}(n) \) time. Furthermore, since we ordered all nodes \( x \) with \( x \neq c \) by nondecreasing values \( d(x, S) - d(x, c) \), all subsets \( X_1, X_2, \ldots, X_n \) can be computed in \( \mathcal{O}(n) \) time. Given \( X_j \), for some \( j \) with \( 1 \leq j \leq n \), Step \( j \) can be done in \( \mathcal{O}(|X_j| + |S_j|) = \mathcal{O}(|X_j|) \) time, provided we can compute in \( \mathcal{O}(1) \) time, for every \( x \in X_j \), the unique \( s \in S \) such that \( x \in \mathbb{T}(s, S) \). That is indeed the case because we precomputed \( \text{Vor}(T, S) \) in \( \mathcal{O}(n) \) time. Overall, the running time is in \( \mathcal{O}(\sum_j |X_j|) = \mathcal{O}(|X_0|) = \mathcal{O}(n) \).

In order to prove correctness of the algorithm, we prove (as claimed above) that at the end of each step \( j \), we have \( \lambda(s) = \ell_x(s) \), for every \( s \in S' \). We prove it by induction on \( j \). If \( j = 0 \), then at the end of step 0 we have:

\[
\lambda(s) = \ell_x(s) = \sum \{\pi(x)\mid x \in X_0 \cap \mathbb{T}(s, S)\} = \ell_x(s) = \sum \{\pi(x)\mid x \in \mathbb{T}(s, S), \ d(x, S) > d(x, c)\} = \ell_x(s) = \sum \{\pi(x)\mid x \in \mathbb{T}(s, S), \ d(x, S) > d(x, c)\} = \ell_x(s)
\]
From now on, we assume that \( j > 0 \). By the induction hypothesis, we have \( \lambda(s) = \ell_{j-1}(s) \) before we start step \( j \). Therefore, at the end of step \( j \) we have:
\[
\lambda(s) = \ell_{j-1}(s) + \sum \{ \pi(x) \mid x \in X_j \cap T(s, S) \}
= \ell_{j-1}(s) + \sum \{ \pi(x) \mid x \in T(s, S), \ d(x, S) = d(x, c) + j \}
= \ell_{j-1}(s) - \sum \{ \pi(x) \mid x \in T(s, S), \ d(x, s) > d(x, c) + j - 1 \} + \sum \{ \pi(x) \mid x \in T(s, S), \ d(x, S) = d(x, c) + j \}
= \ell_{j-1}(s) - \sum \{ \pi(x) \mid x \in T(s, S), \ d(x, s) > d(x, c) + j \}
= \ell_j(s)
\]

Finally, we need to prove the maximality of the selected nodes \( s_j, s'_j \) at each step \( j \). We also prove it by induction on \( j \). This is straightforward if \( j = 0 \) because we consider all nodes of \( S' \) (of \( S' \setminus C(s_0) \), resp.) in order to compute \( s_0 \) (\( s'_0 \), resp.). From now on, we assume that \( j > 0 \). Suppose by contradiction the existence of some node \( s \in S' \), with \( s \neq s_j \) and \( \ell_j(s) > \ell_j(s_j) \). In particular, \( s \notin S_j \), which implies \( \ell_j(s) = \ell_j(s) \). But then, by induction on \( j \),
\[
\ell_j(s) = \ell_{j-1}(s) \leq \ell_{j-1}(s_{j-1}) \leq \ell_j(s_{j-1}) \leq \ell_j(s_j).
\]
A contradiction. We can prove similarly as above that \( s'_j \) maximizes \( \ell_j(s'_j) \) within \( S' \setminus C(s_j) \).

We are now ready to prove the main result in this subsection.

**Theorem 3.8.** We can solve \( \text{BALANCED VERTEX} \) in \( \mathcal{O}(n \log n) \) time for trees.

**Proof.** Let \( T = (V, E) \) be an \( n \)-node tree, let \( \pi \) be any nonnegative cost function and let \( S \) be a proper vertex list. We consider a more general problem, where every node \( v \notin S \) is further assigned nonnegative values \( \lambda(v), \Lambda(v) \) (initially, both values equal 0). Intuitively, we account for sites and territories in a supertree of \( T \) that we are currently considering. Let us define:
\[
\mathcal{R}(v) = \max(\{ \ell_x(v, S + v) + \lambda(v), \Lambda(v) \} \cup \{ \ell_x(s, S + v) \mid s \in S \}).
\]

Our algorithm in what follows compute all values \( \mathcal{R}(v) \), for every \( v \notin S \). We may assume \( S \neq \emptyset \) (otherwise, \( \mathcal{R}(v) = \max(\{ \lambda(v) + \pi(T), \Lambda(v) \} \) for every node \( v \)). If \( V \setminus S \) is reduced to a unique node \( v \), then we first compute \( \text{Vor}(T, S + v) \), which by Proposition 2.4 can be done in \( \mathcal{O}(n) \) time. Doing so, we can compute in \( \mathcal{O}(n) \) time the values \( \ell_x(v, S + v) \) and \( \ell_x(s, S + v) \), for every \( s \in S \), and so, \( \mathcal{R}(v) \). Thus, from now on we assume that \( |V \setminus S| > 1 \). Our algorithm goes as follows:

1. We compute \( \text{Vor}(T, S) \). By Proposition 2.4, this can be done in \( \mathcal{O}(n) \) time.
2. We compute a centroid, that is, a node \( c \) such that every connected component of \( T \setminus \{c\} \) contains at most \( n/2 \) nodes. It can be done in \( \mathcal{O}(n) \) time [18] p. 215]. If furthermore \( c \notin S \), then we compute \( R(c) \). It can be done in \( \mathcal{O}(n) \) time if we are given \( \text{Vor}(T, S + c) \). Therefore, by Proposition 2.4, we can compute \( R(c) \) in \( \mathcal{O}(n) \) time.
3. Let \( C_c \subseteq S \) be such that \( c \in T(s_c, S) \). We locally define a modified cost function \( \pi_c \) obtained from \( \pi \) by setting the cost of every node in \( V \setminus T(s_c, S) \) to 0. Then, we compute \( \alpha(v) = \sum \{ \pi(x) \mid x \in W(v, s_c) \cap T(s_c, S) \} \), for every \( v \notin S \), which can be done in \( \mathcal{O}(n) \) time if we apply Lemma 3.5 for \( \pi_c \). Note that we only use \( \pi_c \) during this step (i.e., it is not used in the remainder of the algorithm).
4. We replace \( \pi \) with some new cost function \( \pi' \) so that:
\[
\pi'(x) = \begin{cases} 
\pi(x) & \text{if } x \notin T(s_c, S) \\
0 & \text{otherwise}.
\end{cases}
\]
5. For every \( v \) with \( v \notin S \cup \{c\} \), we compute the cost of all vertices \( x \in T(v, S + v) \) such that \( v, x \) are in separate connected components of \( T \setminus \{c\} \). Equivalently, we aim at computing \( \beta(v) = \sum \{ \pi'(x) \} \mid c \in I(v, x) \text{ and } d(x, v) < d(x, S) \} \), which can be done in \( \mathcal{O}(n) \) time if we apply Lemma 3.6. For every \( v \) with \( v \notin S \cup \{c\} \), we further compute the maximum cost of a subset \( T(s, S + v) \), amongst all sites \( s \) with \( s \neq s_c \) and \( v, T(s, S) \) are in separate connected components of \( T \setminus \{c\} \). Equivalently, we aim at computing \( \gamma(v) = \max \{ \ell_{x}(s, S + v) \} \mid c \notin T(s, S), c \in I(v, s) \} \), which can be done in \( \mathcal{O}(n) \) time if we apply Lemma 3.7.
6. Let \( C_1, C_2, \ldots, C_q \) be the connected components of \( T \setminus \{c\} \). We consider each component \( C_j \) sequentially, for every \( j \) with \( 1 \leq j \leq q \). Let \( S_j = (S \setminus s_c) \cap C_j \). Let \( \pi_j \) be the restriction of \( \pi' \) to \( C_j \).
For every $v \in C_j$, we define:

$$\lambda_j(v) = \lambda(v) + \alpha(v) + \beta(v)$$

$$\Lambda_j(v) = \max \{ \Lambda(v), \ell_x(s_c, S) - \alpha(v), \gamma(v) \}$$

$$R_j(v) = \max \{ \lambda_j(v) + \ell_x(v, S_j + v), \Lambda_j(v) \} \cup \{ \ell_x(s, S_j + v) | s \in S_j \}$$

We compute $R_j(v)$ for every $v \in C_j$. For that, it suffices to apply our algorithm recursively on $C_j, \pi_j, S_j, \lambda_j, \Lambda_j$. Finally, we set $R(v) = R_j(v)$ for every $v \in C_j$.

The recursion depth is in $O(\log n)$ because at each stage the maximum number of nodes in a subtree considered is halved. Since all subtrees considered at any recursion stage are node disjoint, the running time of the stage is in $O(n)$. Hence, the total running time is in $O(n \log n)$.

In order to prove correctness of the algorithm, it suffices to prove that for every $j$ with $1 \leq j \leq q$, for every $v \in C_j$, $R_j(v) = R(v)$. For that, by Lemma 2.3:

$$\mathbb{T}(v, S + v) = \{ x \in V | d(x, v) < d(s, S) \} = \bigcup \{ W(v, s) \cap \mathbb{T}(s, S) \} | s \in S \}$$

In particular,

$$\lambda(v) + \ell_x(v, S + v) = \lambda(v) + \sum_{s \in S} \pi(W(v, s) \cap \mathbb{T}(s, S))$$

$$= \lambda(v) + \pi(W(v, s) \cap \mathbb{T}(s, S)) + \sum_{s \in S \setminus \{ s \} } \pi(W(v, s) \cap \mathbb{T}(s, S))$$

$$= \lambda(v) + \sum_{s \in S \setminus \{ s \} } \pi(W(v, s) \cap \mathbb{T}(s, S))$$

$$= \lambda(v) + \alpha(v) + \sum_{s \in S \setminus \{ s \} } \pi(W(v, s) \cap \mathbb{T}(s, S))$$

$$= \lambda(v) + \alpha(v) + \sum_{s \in S \setminus \{ s \} } \pi(W(v, s) \cap \mathbb{T}(s, S))$$

$$= \lambda(v) + \alpha(v) + \sum_{s \in S \setminus \{ s \} } \pi(W(v, s) \cap \mathbb{T}(s, S)) + \sum_{s \in S_j} \pi(W(v, s) \cap \mathbb{T}(s, S))$$

$$= \lambda(v) + \alpha(v) + \sum_{s \in S \setminus \{ s \} } \pi(W(v, s) \cap \mathbb{T}(s, S))$$

$$= \lambda(v) + \alpha(v) + \beta(v) + \sum_{s \in S_j} \pi(W(v, s) \cap \mathbb{T}(s, S_j))$$

For every $s \in S$, we have that $\mathbb{T}(s, S + v) = \mathbb{T}(s, S) \setminus W(v, s)$. Therefore,

$$\max(\{ \Lambda(v) \} \cup \{ \ell_x(s, S + v) | s \in S \}) = \max(\{ \Lambda(v), \ell_x(s_c, S + v) \} \cup \{ \ell_x(s, S + v) | s \in S \setminus \{ s_c \} \})$$

$$= \max(\{ \Lambda(v), \ell_x(s_c, S) - \alpha(v) \} \cup \{ \ell_x(s, S + v) | s \in S \setminus \{ s_c \} \})$$

$$\cup \{ \ell_x(s, S + v) | s \in S_j \})$$

$$= \max(\{ \Lambda(v), \ell_x(s_c, S) - \alpha(v), \gamma(v) \} \cup \{ \ell_x(s, S + v) | s \in S_j \})$$

By combining both equalities, we obtain as desired $R(v) = R_j(v)$.

Remark 4. In the recursive algorithm presented for trees, a site $s_c$ is discarded from $S$ at every recursion stage. Therefore, the running time of the algorithm is in $O(n|S|)$, that is linear if $|S|$ is a constant. This can be improved to $O(n \log |S|)$ as follows: at every recursion stage, we compute a node $c$ so that there are at most $|S|/2$ sites in every connected component of $T \setminus \{ c \}$. It can be done in $O(n)$ time because it is a special case of weighted centroid computation [18] [p. 215]. In doing so, the recursion depth goes down to $O(\log |S|)$.

3.3 Bounded-treewidth graphs

We generalize the results of the prior subsections to some increasing hierarchy of graph classes, namely, graphs of bounded treewidth, but only for constantly many sites. Recall that a tree decomposition of a graph $G$ is a pair $(T, \mathcal{X})$, where $T$ is a tree
and \( \lambda' : V(T) \mapsto 2^{V(G)} \) is a mapping that satisfies the following properties:

- For every vertex \( v \in V(G) \), there exists a node \( t \in V(T) \) such that \( v \in \lambda'_t \);
- For every edge \( uv \in E(G) \), there exists a node \( t \in V(T) \) such that \( u, v \in \lambda'_t \);
- For every vertex \( v \in V(G) \), the node subset \( \{ t \in V(T) \mid v \in \lambda'_t \} \) induces a connected subtree of \( T \).

The subsets \( \lambda'_t \), for every node \( t \in V(T) \), are called the bags of the tree decomposition. We define the width of a tree decomposition as the largest size of its bags minus one. The treewidth of \( G \) is the minimum width of its tree decompositions. In particular, graphs of unit treewidth are exactly the forests, and every cycle has treewidth two.

Cabello and Knauer were the first to propose a reduction from some polynomial-time solvable distance problems on bounded-treewidth graphs to orthogonal range queries [10]. We propose a novel application of their framework to the BALANCED VERTEX problem with constant number of sites in \( S \).

We only consider counting queries in what follows. In particular, a range query will always refer to a counting orthogonal range query. Let \( P \) denote a \( n \)-set of \( k \)-dimensional points in \( \mathbb{R}^k \). Let also \( f : P \mapsto \mathbb{R} \) be fixed. A box is the Cartesian product of \( k \) intervals. Finally, a range query asks to compute

\[
\sum \{ f(p) \mid p^2 \in P \cap B \}
\]

for some box \( B \). Bentley presented the \( k \)-range tree data structure in order to answer to range queries [6]. The performance analysis of the latter has been recently improved:

**Lemma 3.9** ([8]). Let \( B(n, k) = \binom{k + \log n}{k} \). Given a \( n \)-set of \( k \)-dimensional points, and any weight function \( f \), a \( k \)-range tree can be constructed in \( \Theta(k^2 B(n, k) n) \) time. Moreover, a range query can be answered in \( \Theta(2^k B(n, k)) \) time.

Throughout this subsection, we write all running times as some functions of \( B(n, k) \). We refer to Remark 5 for a more detailed analysis of \( B(n, k) \).

Roughly, our strategy consists in mimicking our previous approach for trees. Specifically, centroids in a tree decomposition of width at most \( k \) are balanced separators of size at most \( k \). We generalize Lemma 3.6, for cut-vertices in trees, to the following Lemma 3.10 for \( k \)-separators. Our generalization makes use of range queries. Intuitively, the function \( D_S \) in the statement of Lemma 3.10 below represents the distances to the sites of \( S \) in some supergraph of \( G \). Some edges in \( G \) may have a positive integer weight larger than one, that is in order to account for shortest paths in some supergraph of \( G \). Then, the distances in \( G \) depend on the edge weights.

**Lemma 3.10.** Given a graph \( G = (V, E, \lambda) \) with positive integer edge weights, and two nonnegative cost functions \( \pi \) and \( D_S \), let \( A \) and \( B \) satisfy: \( A \cup B = V, |A \cap B| \leq k \) and there is no edge between \( A \setminus B \) and \( B \setminus A \). We can compute \( \delta(a, B) = \sum \{ \pi(b) \mid b \in B \setminus A \text{ and } d(b, a) < D_S(b) \} \), for every \( a \in A \setminus B \), in \( \Theta(km + k^2 B(n, k)n) \) time.

**Proof.** For convenience, let \( X = A \cap B \). We further assume for simplicity that \( |X| = k \). Since we assume that there is no edge between \( A \setminus B \) and \( B \setminus A \), every path between a vertex of \( A \) and a vertex of \( B \) must go through \( X \). We first compute a shortest-path tree rooted at \( x \), for every \( x \in X \). Since all edge weights are positive integer, this can be done in \( \Theta(m) \) time per vertex in \( X \), using Thorup’s single-source shortest-path algorithm [27]. Let \( X = \{ x_1, x_2, \ldots, x_k \} \) be totally ordered in some arbitrary way. We proceed as follows for every \( i \) with \( 1 \leq i \leq k \).

1. For every \( b \in B \setminus X \), we create a \( k \)-dimensional point \( p_i(b) = (p_{i,1}(b), p_{i,2}(b), \ldots, p_{i,k}(b)) \), where:

\[
\begin{align*}
p_{i,1}(b) &= D_S(b) - d(b, x_i) \\
p_{i,j}(b) &= d(b, x_j) - d(b, x_i) \quad \text{for every } j \text{ with } 1 \leq j \leq k, j \neq i
\end{align*}
\]

Let \( f(p_i(b)) = \pi(b) \). We add all these points and their weights in some \( k \)-range tree. By Lemma 3.9, this can be done in \( \Theta(k^2 B(n, k)n) \) time.

2. For every \( a \in A \setminus X \), we aim at computing the cost \( \delta_j(a, B) \) of all vertices \( b \in B \setminus X \) with the following properties: \( D_S(b) > d(b, a) \), \( \lambda_i \) is on a shortest \( \lambda_k \)-path; no vertex of \( \{ x_1, x_2, \ldots, x_{i-1} \} \) is on a shortest \( \lambda_k \)-path.

We can formulate these properties as range constraints for each coordinate of \( p_i(b) \), as follows:

\[
\begin{align*}
p_{i,j}(b) &= d(b, x_j) - d(b, x_i) > d(a, x_j) - d(a, x_i), \quad \text{for every } j \text{ with } 1 \leq j < i \\
p_{i,i}(b) &= D_S(b) - d(b, x_i) > d(a, x_i) \\
p_{i,j}(b) &= d(b, x_j) - d(b, x_i) \geq d(a, x_j) - d(a, x_i), \quad \text{for every } j \text{ with } i < j \leq k
\end{align*}
\]
As a result, for a fixed \( a \), we can compute the cost of all corresponding vertices of \( B \setminus X \) with a single range query, which by Lemma 3.9 can be answered in \( O(2^k B(n,k)) \) time.

Finally, we compute \( \delta(a, B) = \sum_{i=1}^k \delta_i(a, B) \), for every \( a \in A \setminus X \).  

Given a tree decomposition \((T,X)\) of width \( O(k) \), the following result is a combination of Lemma 3.10 with a centroid decomposition of \( T \).

\[ \text{Lemma 3.11. Let } S \text{ be an arbitrary vertex subset in a graph } G = (V,E), \text{ and let } \pi \text{ be any nonnegative cost function. If } G \text{ has treewidth at most } k, \text{ then we can compute } \delta(v) = \sum \{ \pi(u) | d(u,v) < d(u,S) \}, \text{ for every } v \in V \setminus S, \text{ in } O(k^2 O(k) B(n,k) n \log n) \text{ time.} \]

**Proof.** We consider a more general case where we are also given a positive integral edge weight function \( \lambda \) (initially, we have \( \lambda(e) = 1 \) for every \( e \in E \)). We may assume that \( n > k + 1 \) (otherwise, we can compute all values \( \delta(v) \) in \( O(k^2) \) time, if we use a brute-force algorithm). We may further assume that all distances \( D_\lambda(v) = d(v,S) \), for every vertex \( v \), are known throughout the entire procedure, which can always be ensured by running a BFS on \( S \) before the algorithm starts. Let \((T,X)\) be a tree decomposition of \( G \) of width \( O(k) \). Such a tree decomposition can be computed in \( O(2^O(k) n) \) time [23]. We often use that \( m = O(kn) \).

1. We compute some bag \( X_t \), with \( t \in V(T) \), such that every connected component of \( G \setminus X_t \) contains at most \( n/2 \) vertices. Such a bag always exists and it can be computed in \( O(mn) \) time if \((T,X)\) is given [13] [Lemma 11.16]. For every \( x \in X_t \setminus S \), we can compute \( \delta(x) \) in \( O(m) = O(kn) \) time if we run a BFS on \( x \).
2. We group the connected components of \( G \setminus X_t \) in two subsets \( A \) and \( B \) so that: \( A \cup B = V \), \( A \cap B = X_t \) and \( \max \{|A|, |B| \} \leq 2n/3 \). We apply Lemma 3.10 twice in order to compute \( \delta(a, B) \) for every \( a \in A \setminus X_t \) (\( \delta(b, A) \), for every \( b \in B \setminus X_t \), resp.). This can be done in \( O(km + k^2 B(n,k)n) = O(k^2 B(n,k)n) \) time.
3. Let \( G_A \) (resp.) be obtained from \( G[A] \) (resp.) by adding edges \( xx' \) of weights \( d(x, x') \), for every two \( x, x' \in X_t \). The computation of all distances \( d(x, x') \) can be done in \( O(km) = O(k^2 n) \) time, if we run a BFS on every vertex of \( X_t \). We apply our algorithm recursively to \( G_A, G_B \) in order to compute the values \( \delta_{G_A}(a) = \sum \{ \pi(a') | a' \in A, d(a, a') < D_\lambda(a') \} \), for every \( a \in A \), and the values \( \delta_{G_B}(b) = \sum \{ \pi(b') | b' \in B, d(b, b') < D_\lambda(b') \} \), for every \( b \in B \).
4. For every \( a \in A \setminus X_t \), we set \( \delta(a) = \delta(a, B) + \delta_{G_A}(a) \). In the same way, for every \( b \in B \setminus X_t \), we set \( \delta(b) = \delta(b, A) + \delta_{G_B}(b) \).

The recursion depth is in \( O(\log n) \). Each recursive stage can be done in \( O(k^2 B(n,k)n) \) time (the running time is dominated by Step 2). Therefore, the running time of the algorithm is in \( O(k^2 O(k) B(n,k)n \log n) \).

Finally, we reduce \textsc{Balanced Vertex} with constant number of sites to Lemma 3.11.

\[ \text{Theorem 3.12. We can solve } \textsc{Balanced Vertex} \text{ in } O(k^2 O(k) B(n,k)n |S| \log n) \text{ time if } G \text{ has treewidth at most } k. \]

**Proof.** For every \( v \notin S \), our algorithm in what follows computes \( \ell_\pi(v, S + v) \) and all values \( \ell_\pi(s, S + v) \), for every \( s \in S \). Note that doing so, we can compute \( L_\pi(S + v) \), for every \( v \notin S \), in additional \( O(n|S|) \) time.

1. We compute \( \text{Vor}(G, S) \). By Proposition 2.4, this can be done in \( O(m) \) time, that is in \( O(kn) \) if \( G \) has treewidth \( k \). Doing so, we can compute the loads \( \ell_\pi(s, S) \), for every \( s \in S \), in additional \( O(n) \) time.
2. We apply Lemma 3.11 in order to compute \( \ell_\pi(v, S + v) \), for every \( v \notin S \).
3. We then consider each \( s \in S \) sequentially. We replace \( S \) by the list \( (s) \) with a unique site, and the cost function \( \pi \) by a new cost function \( \pi_x \), such that

\[
\begin{align*}
\pi_x(u) &= 0 \text{ if } u \notin T(s, S) \\
\pi_x(u) &= \pi(u) \text{ otherwise}.
\end{align*}
\]

Note that we can compute the new cost function \( \pi_x \) in \( O(n) \) time, that is because we are given \( \text{Vor}(G, S) \). We apply Lemma 3.11 in order to compute, for every \( v \notin S \), \( \pi(W(v,s) \cap T(s, S)) \). Doing so, we can compute \( \ell_\pi(s, S + v) = \ell_\pi(s, S) - \pi(W(v,s) \cap T(s, S)) \).

Overall, since we apply Lemma 3.11 \(|S| + 1 \) times, the running time is in \( O(k^2 O(k) B(n,k)n |S| \log n) \).

**Remark 5.** Bringmann et al. proved that for every \( \varepsilon > 0 \), there exists a constant \( c \) such that \( B(n, k) = 2^{ck} n^\varepsilon \) [8]. In particular, for every \( \varepsilon > 0 \), there exists a constant \( c' \) such that the running time of our algorithm for bounded
treewidth graphs is in $\mathcal{O}(2^k|S|n^{1+\epsilon})$. The exponential dependency on the treewidth is necessary, even for one site, if one assumes the Hitting Set Conjecture. Indeed, the graph constructed in the proof of Theorem 2.8 has treewidth $\mathcal{O}(\log n)$.

### 3.4 Proper interval graphs

We devote our last and arguably most technical subsection to proper interval graphs. Recall that an interval graph is the intersection graph of a family of intervals on the real line. A realization of an interval graph $G$ is a mapping of its vertices to closed intervals on the real line, so that two vertices are adjacent in $G$ if and only if their corresponding intervals are intersecting. It is a proper realization if there are no two vertices such that the interval of one is contained in the interval of the other. A proper interval graph is one admitting a proper realization. Note that a proper realization can be computed in $\mathcal{O}(n+m)$ time [12].

Our scheme only applies to proper interval graphs, not to interval graphs, and it is good to explain why it is so. Roughly, we would like to mimic the scheme for paths (see Lemma 3.3). A natural path representation for the interval graphs is to order the vertices

$$G = (V, E)$$

by nondecreasing left boundary, within some fixed realization. Then, what we would like to prove is the existence of some constant $\kappa$ (ideally, $\kappa = 1$) such that if we insert a new site $v$ at the end of $S$, then only the $\kappa$ closest sites to $v$ on the left and right of the path representation could have their respective territories modified. Unfortunately, this is impossible due to the potential existence of vertices whose interval covers an arbitrary number of other intervals. Therefore, we restrict ourselves to proper realizations, for which the latter case cannot happen. Some important properties of proper realizations are captured by the following distance formula.

**Lemma 3.13** ([16]). Let $G = (V, E)$ be a proper interval graph, and let $x_0$ be the vertex with the minimum left boundary in some proper realization of $G$. We can compute in $\mathcal{O}(n)$ time a total ordering $\sigma : V \to \{1, \ldots , n\}$ so that, for every vertices $u$ and $v$ with $d(u, x_0) \leq d(v, x_0)$:

$$d(u, v) = \begin{cases} d(v, x_0) - d(u, x_0) & \text{if } d(u, x_0) < d(v, x_0) \text{ and } \sigma(v) < \sigma(u) \\ d(v, x_0) - d(u, x_0) + 1 & \text{otherwise.} \end{cases}$$

First, we combine Lemma 3.13 with orthogonal range queries (see Section 3.3) in order to precompute the load of every new site in a prioritized Voronoi diagram. We could easily do so with three-dimensional range queries because, for every vertex $u$, we only need to memorize $\sigma(u)$ and $d(u, x_0)$, $d(u, S)$. However, we can shave some polylogarithmic factors by reducing further down to one-dimensional range queries. Indeed, a one-range tree is just a balanced binary search tree$^2$. In particular, by using a self-balanced binary search tree implementation, we can serve range queries on a dynamic collection of values. Doing so, we only need to store $\sigma(u)$, for every vertex $u$, while other relevant distance information can be deduced contextually for the vertices $u$ that are currently stored in the search tree.

**Lemma 3.14.** Let $G = (V, E)$ be a proper interval graph, let $x_0$ be the vertex with the minimum left boundary in some proper realization of $G$, and let $\sigma$ be as in Lemma 3.13. For every nonnegative cost function $\pi$, and every proper vertex list $S$, we can compute the values $\ell_\pi^*(v; S + v)$, for every $v \notin S$, in $\mathcal{O}(m + n \log n)$ time.

**Proof.** We first describe the algorithm.

1. We compute all distances $d(v, x_0), d(v, S)$, for every vertex $v$. Doing so, we partition $V$ into distance layers $X_0 = \{x_0\}, X_1 = N(x_0), X_2, \ldots , X_{\ell(x_0)}$, where $\ell(x_0) = \max_{v \in V} d(x_0, v)$ and, for every $j$ with $0 \leq j \leq \ell(x_0)$, $X_j = \{v \in V | d(x_0, v) = j\}$. This can be done in $\mathcal{O}(n + m)$ time. Let us further compute $\text{Vor}(G, S)$, which by Proposition 2.4 can also be done in $\mathcal{O}(n + m)$ time. For convenience, for every vertex $v$, we denote in what follows by $s_v$ the unique site of $S$ such that $v \in \mathcal{T}(s_v, S)$.

2. We consider each distance layer $X_0, X_1, \ldots , X_{\ell(x_0)}$ sequentially. Throughout all iterations of this phase, we maintain a counter $\Lambda^-$ and a balanced binary search tree $T^-$ such that the following invariants hold before considering any layer $X_j$:

   - $\Lambda^+ = \sum \{\pi(u) | d(x_0, u) < j, \text{ and } d(u, s_u) > j - d(x_0, u) + 1\}$.

   Initially, $\Lambda^- = 0$. Then, for every vertex $u$, we can precompute the values $d(x_0, u) + 1$ and $d(u, s_u) + d(x_0, u) - 1$, which represent the layers for which we must add and remove $\pi(u)$ from $\Lambda^-$, respectively. Note that given the distance layers $X_0, X_1, \ldots , X_{\ell(x_0)}$, and given $\text{Vor}(G, S)$, the values $d(x_0, u) + 1$ and $d(u, s_u) + d(x_0, u) - 1$ for every vertex $u$ can be computed in total $\mathcal{O}(n)$

$^2$Values are stored at the leaves, but this feature is not important in our case.
time. Furthermore, if \( d(x_0, u) = e(x_0) \) or \( d(x_0, u) + 1 \geq d(u, s_u) + d(x_0, u) - 1 \), then we never add nor remove \( \pi(u) \) from \( \Lambda^- \). If \( d(u, s_u) + d(x_0, u) - 1 > e(x_0) \), then we never remove \( \pi(u) \) from \( \Lambda^- \). Since the cost \( \pi(u) \) of every vertex \( u \) needs to be added and removed from \( \Lambda^- \) at most once, we can maintain the counter \( \Lambda^- \) throughout this whole phase in total \( \Theta(n) \) time.

- we store in \( T^- \) all vertices \( u \) such that: \( d(x_0, u) < j \), and \( d(u, s_u) = j - d(x_0, s_u) + 1 \). A vertex \( u \) is identified in \( T^- \) with the value \( \sigma(u) \). Furthermore, at each vertex \( u \) in \( T^- \) we store the sum of the costs \( \pi(u') \) of all vertices \( u' \) in its rooted subtree.

Each vertex \( u \) is inserted and deleted from \( T^- \) at most once. Standard implementations such as AVL [2] can be easily modified in order to dynamically maintain the cost of rooted subtrees after each insertion/deletion. As a result, we can maintain \( T^- \) throughout this whole phase in total \( \Theta(n \log n) \) time.

For every \( v \in X_j \), we compute the cumulative cost of all vertices \( u \in T^- \) such that \( \sigma(v) < \sigma(u) \). Since every node of \( T^- \) stores the cost of its rooted subtree, this can be done in \( \Theta(n \log n) \) time by using range searching techniques, namely: we search for the vertex \( u \) of \( T^- \) such that \( \sigma(u) > \sigma(v) \) and \( \sigma(u) \) is minimized, then we process the costs of all vertices on the path to the root and of their respective right subtrees (details left out can be found, e.g., in [6,8]). Let \( \lambda^-(v) = \Lambda^- + \sum \{ \pi(u) | u \in T^-, \sigma(v) < \sigma(u) \} \).

3. Roughly, we do the reverse of the previous phase. Specifically, we consider each distance layer \( X_{e(x_0)}, X_{e(x_0)} - 1, \ldots, X_0 \) sequentially. Throughout all iterations of this phase, we maintain a counter \( \Lambda^+ \) and a balanced binary search tree \( T^+ \) such that the following invariants hold before considering any layer \( X_j \):

- \( \Lambda^+ = \sum \{ \pi(u) | d(x_0, u) > j, \text{ and } d(u, s_u) > d(x_0, u) - j + 1 \} \).
- we store in \( T^+ \) all vertices \( u \) such that: \( d(x_0, u) > j \), and \( d(u, s_u) = d(x_0, s_u) - j + 1 \). A vertex \( u \) is identified in \( T^+ \) with the value \( \sigma(u) \). Furthermore, at each vertex \( u \) in \( T^+ \) we store the sum of the costs \( \pi(u') \) of all vertices \( u' \) in its rooted subtree.

For every \( v \in X_j \), we compute the cumulative cost of all vertices \( u \in T^+ \) such that \( \sigma(v) > \sigma(u) \). Let \( \lambda^+(v) = \Lambda^+ + \sum \{ \pi(u) | u \in T^+, \sigma(v) > \sigma(u) \} \).

The total running time is in \( \Theta(m + n \log n) \).

Let \( j \) with \( 0 \leq j \leq e(x_0) \) be fixed. We set \( \Pi(j) = \sum \{ \pi(u) | u \in X_j, \text{ } d(u, s_u) > 1 \} \). For every \( v \in X_j \setminus S \), we claim that we have

\[
\ell^+(v, S + V) = \begin{cases} 
\lambda^+(v) + \lambda^+(v) + \Pi(j) & \text{if } d(v, s_v) > 1 \\
\lambda^- + \lambda^+(v) + \Pi(j) + \pi(v) & \text{else}
\end{cases}
\]

Indeed, let \( u \in V \setminus \{ v \} \) be arbitrary. There are three different cases:

- If \( d(u, x_0) = j \), then by Lemma 3.13, \( d(u, v) = 1 \). In particular, \( u \in T(v, S + v) \) if and only if \( d(u, s_u) > 1 \). Therefore, the contribution to \( \ell^+(v, S + v) \) of all vertices \( u' \) such that \( d(u', x_0) = j \) must equal \( \Pi(j) \) if \( d(v, s_v) > 1 \), and it must equal \( \Pi(j) + \pi(v) \) if \( d(v, s_v) = 1 \).
- If \( d(u, x_0) < j \), then by Lemma 3.13, \( d(u, v) \in \{ j - d(u, x_0), j - d(u, x_0) + 1 \} \). In particular, \( u \notin T(v, S + v) \) if \( d(u, s_u) \leq j - d(u, x_0) \). Furthermore, \( u \notin T(v, S + v) \) if \( d(u, s_u) > j - d(u, x_0) + 1 \). Otherwise, \( d(u, s_u) = j - d(u, x_0) + 1 \), and so we have \( u \in T(v, S + v) \) if and only if \( d(u, v) = j - d(u, x_0) \), which by Lemma 3.13 is equivalent to \( \sigma(v) < \sigma(u) \). As a result, the contribution to \( \ell^+(v, S + v) \) of all vertices \( u' \) such that \( d(u', x_0) < j \) must equal \( \lambda^- \).
- Otherwise, \( d(u, x_0) > j \). By Lemma 3.13, \( d(u, v) \in \{ d(u, x_0) - j, d(u, x_0) - j + 1 \} \). In particular, \( u \notin T(v, S + v) \) if \( d(u, s_u) \leq d(u, x_0) - j \). Furthermore, \( u \notin T(v, S + v) \) if \( d(u, s_u) > d(u, x_0) - j + 1 \). Otherwise, \( d(u, s_u) = d(u, x_0) - j + 1 \), and so we have \( u \in T(v, S + v) \) if and only if \( d(u, v) = d(u, x_0) - j \), which by Lemma 3.13 is equivalent to \( \sigma(u) < \sigma(v) \). As a result, the contribution to \( \ell^+(v, S + v) \) of all vertices \( u' \) such that \( d(u', x_0) > j \) must equal \( \lambda^+(v) \).

Our claim directly follows from this case analysis.

From another application of Lemma 3.13, where we process distance layers from \( x_0 \) in a different way than what we did for Lemma 3.14, we obtain the following complementary result.

**Lemma 3.15.** Let \( G = (V, E) \) be a proper interval graph, let \( x_0 \) be the vertex with the minimum left boundary in some proper realization of \( G \), and let \( \sigma \) be as in Lemma 3.13. For every nonnegative cost function \( \pi \), and every
proper vertex list $S$, we can compute the values $\Lambda(v) = \max\{\ell_x(s, S + v) | s \in S\}$, for every vertex $v \not\in S$, in $\mathcal{O}(m + n \log n)$ time.

**Proof.** We need some notations for what follows. Let $e(x_0) = \max_{v \in V} d(x_0, v)$. For every $j$ with $0 \leq j \leq e(x_0)$, let $X_j = \{v \in V | d(x_0, v) = j\}$. Finally, for every vertex $v$, let $s_v$ be the unique site of $S$ such that $v \in T(s_v, S)$.

Roughly, the algorithm consists of two phases. During its first phase, we compute the Voronoi diagram of $G$ with respect to $S$, and we create a max-heap $H$ storing all sites in $S$. Then, during the second and main phase, we process the heap in order to compute $\Lambda(v)$, for every $v \in V \setminus S$. More precisely, we iterate over the distance layers $X_0, X_1, \ldots, X_{e(x_0)}$ and, for every $j$ with $0 \leq j \leq e(x_0)$, we compute all values $\Lambda(v)$, for $v \in X_j \setminus S$, sequentially.

Intuitively, when we consider a vertex $v \in X_j \setminus S$, the key of every site $s$ in $H$ must be equal to $\ell_x(s, S + v)$.

**First phase.** We compute $\text{Vor}(G, S)$, which by Proposition 2.4 can be done in $\mathcal{O}(n + m)$ time. We compute, also in $\mathcal{O}(n + m)$ time, the distances $d(x_0, v), d(v, S)$ for every vertex $v$. Doing so, we can partition $V$ in $X_0, X_1, \ldots, X_{e(x_0)}$. Finally, we create a max-heap $H$ and, for every site $s$ of $S$, we insert $s$ into $H$ with key equal to $key(s) = \ell_x(s, S)$.

Given $\text{Vor}(G, S)$, we can compute all the keys in $\mathcal{O}(m)$ time. Every insertion in a heap can be done in $\mathcal{O}(\log n)$ time. Therefore, we can initialize the heap $H$ in $\mathcal{O}(n \log n)$ time.

Throughout every loop $j$ of the second phase, we maintain the following invariant for $H$: for every site $s$ of $S$,

$$key(s) = \ell_x(s, S) - \sum \{\pi(u) | u \in T(s, S) \text{ and } d(u, s_v) > |j - d(u, x_0)| + 1\}$$

Intuitively, this is because any such vertex $u$ must be in the territory of $v \not\in S$, provided $v \in X_j$. For that, before we start the $j^{th}$ loop, we need to actualize the keys of all sites. More precisely, for every vertex $u$:

- We claim that we must set $key(s_u) = key(s_v) - \pi(u)$ if the following conditions hold:

  $$\begin{cases} d(u, s_v) \geq 2 \\
  j = \max \{0, d(u, x_0) - d(u, s_v) + 2\} \end{cases}$$

  Indeed, if $d(u, s_v) \leq 1$, then there is no $j'$ such that $d(u, s_v) > |j' - d(u, x_0)| + 1$. Otherwise, for $j \leq d(u, x_0)$, we have $d(u, s_v) > d(u, x_0) - j + 1$ if and only if $j > d(u, x_0) - d(u, s_v) + 1$.

- We claim that we must set $key(s_u) = key(s_v) + \pi(u)$ if the following conditions hold:

  $$\begin{cases} d(u, s_v) \geq 2 \\
  j = d(u, s_v) + d(u, x_0) - 1 \end{cases}$$

  Indeed, if $d(u, s_v) \geq 2$ and $j \geq d(u, x_0)$, then we have $d(u, s_v) > j - d(u, x_0) + 1$ if and only if $j < d(u, s_v) + d(u, x_0) - 1$.

For every vertex $u$, there are at most two loops such that we modify $key(s_u)$. Furthermore, given the distance layers $X_0, X_1, \ldots, X_{e(x_0)}$, and given $\text{Vor}(G, S)$, the indices $j = \max \{0, d(u, x_0) - d(u, s_v) + 2\}$ and $j = d(u, s_v) + d(u, x_0) - 1$ of these two loops can be precomputed in $\mathcal{O}(1)$ time. Therefore, we need to modify any key $\mathcal{O}(n)$ times during the second phase. Overall, we can maintain our invariant throughout the second phase in $\mathcal{O}(n \log n)$ time.

**Second phase.** We are now ready to describe the main loop of the algorithm. We consider each distance layer $X_0, X_1, \ldots, X_{e(x_0)}$ sequentially. Let $j$ with $0 \leq j \leq e(x_0)$ be fixed. We compute $\Lambda(v)$ for every vertex $v \in X_j \setminus S$.

For that, we define the frontier $F_j$, as follows:

$$F_j = \{u \in V \setminus X_j | d(u, s_v) = |j - d(u, x_0)| + 1\}.$$ 

Intuitively, given a vertex $v \in X_j \setminus S$, the vertices $u$ in the frontier may or may not be in $T(v, S + v)$ depending on whether $\sigma(u)$ is bigger or smaller than $\sigma(v)$. Determining, for every site $s$, the vertices in $F_j \cap T(s, S)$ that belong to $v$’s territory, is the main difficulty in our algorithm. In what follows, we find it convenient to bipartition $F_j$ in $F_j^\leq, F_j^\geq$ such that:

$$F_j^\leq = \{u \in F_j | d(x_0, u) < j\}$$

$$F_j^\geq = \{u \in F_j | d(x_0, u) > j\}$$

We subdivide the loop in many intermediate steps:
1. For every site $s$ of $S$, we set:

$$key(s) = key(s) - \sum \{\pi(u) | u \in F_j^c \cap T(s, S)\}$$

Since we are given $\text{Vor}(G, S)$, for every vertex $u \in F_j^c$ we can access to $s_u$ in $O(1)$ time; then, we can decrease $key(s_u)$ by $\pi(u)$, and reordering the heap, in $O(\log n)$ time. Hence, this whole step can be done in $O(|F_j^c| \log n)$ time by scanning the vertices of $F_j^c$.

2. We order the vertices of $F_j \cup X_j$ according to $\sigma$. This can be done in $O(|F_j| \log n + |X_j| \log n)$ time by using any fast sorting algorithm.

3. We scan the vertices of $F_j \cup X_j$ by increasing $\sigma$ values:

- **Case of a vertex $u \in F_j^c$.** We set $key(s_u) = key(s_u) + \pi(u)$.
- **Case of a vertex $u \in F_j^c$.** We set $key(s_u) = key(s_u) - \pi(u)$.
- **Case of a vertex $v \in X_j \setminus S$.** First, if $d(s_v, v) = 1$, then we set $key(s_v) = key(s_v) - \pi(v)$. We set $\Lambda(v)$ to be the maximum key in $H$. Finally, if $d(s_v, v) = 1$, then we reset $key(s_v) = key(s_v) + \pi(v)$.

This can be done in $O(\log n)$ time per vertex considered.

4. We reset the keys in $H$, namely, for every site $s$ of $S$, we set:

$$key(s) = key(s) + \sum \{\pi(u) | u \in F_j^c \cap T(s, S)\}$$

It takes $O(|F_j^c| \log n)$ time.

The total running time for the $j$th loop, excluding the cost for maintaining our invariant for $H$, is in $O(|F_j| \log n + |X_j| \log n)$. Furthermore, every vertex $u$ can appear in the frontier $F_j$ for at most two indices $j'$, and both indices can be pre-computed in $O(1)$ time per vertex. Hence, the overall running time of the second phase is in $O(n \log n)$ plus the total cost for maintaining the invariant for $H$, which is also in $O(n \log n)$.

In order to prove correctness of our algorithm, let $j$ with $0 \leq j \leq \epsilon(x_0)$ be fixed. Let $v \in X_j \setminus S$ be arbitrary. In order to prove that we correctly computed $\Lambda(v)$, it suffices to prove that at the time we consider this vertex during the $j$th loop we have $key(s) = \ell_x(s, S + v)$ for every site $s$ of $S$. For that, recall that by Lemma 3.13, for every vertex $u \in F_j$ we have that $|j - d(u, x_0)| \leq d(u, v) \leq |j - d(u, x_0)| + 1$. Furthermore, $d(u, v) = |j - d(u, x_0)|$ if and only if: $u \in F_j^c$ and $\sigma(u) > \sigma(v)$; or $u \in F_j^c$ and $\sigma(u) < \sigma(v)$. As a result,

$$key(s) = \ell_x(s, S) - \sum \{\pi(u) | u \in T(s, S) \land d(u, s_u) > |j - d(u, x_0)| + 1\}
- \sum \{\pi(u) | u \in F_j^c \cap T(s, S)\}
+ \sum \{\pi(u) | u \in F_j^c \cap T(s, S) \land \sigma(u) < \sigma(v)\}
- \sum \{\pi(u) | u \in F_j^c \cap T(s, S) \land \sigma(u) < \sigma(v)\}
- \mathbf{1}[s = s_v, d(s, v) = 1] \cdot \pi(v)$$

$$= \ell_x(s, S) - \sum \{\pi(u) | u \in T(s, S) \setminus \{v\} \land d(u, s_u) > |j - d(u, x_0)| + 1\}
- \sum \{\pi(u) | u \in F_j^c \cap T(s, S) \land \sigma(u) > \sigma(v)\}
- \sum \{\pi(u) | u \in F_j^c \cap T(s, S) \land \sigma(u) < \sigma(v)\}
- \mathbf{1}[s = s_v] \cdot \pi(v)$$

$$= \ell_x(s, S) - \sum \{\pi(u) | u \in T(s, S) \setminus \{v\} \land d(u, s_u) > |j - d(u, x_0)| + 1\}
- \sum \{\pi(u) | u \in F_j^c \cap T(s, S) \land d(u, v) = |j - d(u, x_0)|\}
- \sum \{\pi(u) | u \in F_j^c \cap T(s, S) \land d(u, v) = d(u, x_0) - j\}
- \mathbf{1}[s = s_v] \cdot \pi(v)$$

$$= \ell_x(s, S) - \sum \{\pi(u) | u \in T(s, S) \land d(u, v) < d(u, s)\}$$

The following theorem summarizes our results in this subsection.

**Theorem 3.16.** We can solve $\text{BALANCED VERTEX}$ in $O(m + n \log n)$ time if $G$ is a proper interval graph.
Proof. We compute a proper realizer, which can be done in \( \mathcal{O}(n + m) \) time [12]. Then, in additional \( \mathcal{O}(n) \) time, we compute the vertex \( x_0 \) of minimum left boundary and the ordering \( \sigma \) of Lemma 3.13. We are done applying Lemmas 3.14 and 3.15.

4 CONCLUSION

In this paper, we introduced a new problem on graph Voronoi diagrams, and we completely settled its complexity for general graphs under plausible complexity assumptions. We left open whether our conditional time lower bound could also hold under weaker complexity hypotheses.

We proposed various ways to tackle with our new problem on tree-like and path-like topologies. In this respect, two immediate questions following our work could be whether our results can be extended to bounded-treewidth graphs with nonconstant number of sites, and to interval graphs.

ACKNOWLEDGMENTS

This work was supported by a grant of the Ministry of Research, Innovation and Digitalization, CCCDI–UEFISCDI, project number PN-III-P2-2.1-PED-2021-2142, within PNCDI III.

DATA AVAILABILITY STATEMENT

Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

ORCID

Guillaume Ducoffe

https://orcid.org/0000-0003-2127-5989

REFERENCES

[1] A. Abboud, V. Vassilevska Williams, and J. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. Proceedings of the twenty-seventh annual symposium on discrete algorithms (SODA), SIAM. 2016 377–391.
[2] G. Adelson-Velsky and E. Landis, An algorithm for the organization of information, Proc USSR Acad Sci 146 (1992), 263–266.
[3] F. Aurenhammer, Voronoi diagrams—a survey of a fundamental geometric data structure, ACM Comput Surv 23 (1991), no. 3, 345–405.
[4] S. Bandyapadhyay, A. Banik, S. Das, and H. Sarkar, Voronoi game on graphs, Theor. Comput. Sci. 562 (2015), 270–282.
[5] M. Bender and M. Farach-Colton, The level ancestor problem simplified, Theor. Comput. Sci. 321 (2004), no. 1, 5–12.
[6] J. Bentley, Decomposable searching problems, Inf. Process. Lett. 8 (1979), no. 5, 244–251.
[7] J. A. Bondy and U. S. R. Murty, Graph theory, Springer-Verlag, London, 2008.
[8] K. Bringmann, T. Husfeldt, and M. Magnusson, Multivariate analysis of orthogonal range searching and graph distances, Algorithmica. 82 (2020), no. 8, 1–24.
[9] S. Cabello, Subquadratic algorithms for the diameter and the sum of pairwise distances in planar graphs, ACM Trans Algorithms 15 (2018), no. 2, 21.
[10] S. Cabello and C. Knauer, Algorithms for graphs of bounded treewidth via orthogonal range searching, Comput. Geom. 42 (2009), no. 9, 815–824.
[11] V. Cohen-Addad, S. Dahlgaard, and C. Wulff-Nilsen. Fast and compact exact distance oracle for planar graphs. Paper presented at: 2017 IEEE 58th annual symposium on foundations of computer science (FOCS), IEEE. 2017 962–973.
[12] D. Corneil, A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs, Discret. Appl. Math. 138 (2004), no. 3, 371–379.
[13] R. Downey and M. Fellows, Parameterized complexity, Springer Science & Business Media, New York, 2012.
[14] M. Erwig, The graph Voronoi diagram with applications, Netw. Int. J. 36 (2000), no. 3, 156–163.
[15] M. Garey and D. Johnson, Computers and intractability: A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences, W.H. Freeman and Company, New York, 1979.
[16] C. Gavoille and C. Paul, Optimal distance labeling for interval graphs and related graph families, SIAM J. Discret. Math. 22 (2008), no. 3, 1239–1258.
[17] P. Gawrychowski, H. Kaplan, S. Mozes, M. Sharir, and O. Weimann. Voronoi diagrams on planar graphs, and computing the diameter in deterministic \( \mathcal{O}(n^{5/3}) \) time. Proceedings of the twenty-ninth annual symposium on discrete algorithms (SODA), SIAM. 2018 495–514.
[18] A. Goldman, Optimal center location in simple networks, Transp. Sci. 5 (1971), no. 2, 212–221.
[19] L. Hakimi and M. Labbé, “A locational problem associated with the Voronoi partition of a network,” ORSA/TIMS National meeting (April), ORSA/TIMS, Washington D.C., 1988.
[20] L. Hakimi, M. Labbê, and E. Schmeichel, The Voronoi partition of a network and its implications in location theory, ORSA J. Comput. 4 (1992), no. 4, 412–417.
[21] F. Hurtado, R. Klein, E. Langetepe, and V. Sacristán, The weighted farthest color Voronoi diagram on trees and graphs, Comput. Geom. 27 (2004), no. 1, 13–26.
[22] J. Kleinberg and E. Tardos, Algorithm design, Pearson Education India, Boston, 2006.
[23] T. Körhonen. A single-exponential time 2-approximation algorithm for treewidth. Paper presented at: 2021 IEEE 62nd annual symposium on foundations of computer science (FOCS), IEEE. 2022 184–192.
[24] K. Mehlhorn, A faster approximation algorithm for the Steiner problem in graphs, Inf. Process. Lett. 27 (1988), no. 3, 125–128.
[25] I. Núñez and A. Soto-Mota. Hospital saturation and risk of death without receiving mechanical ventilation in hospitalized COVID-19 patients: a city-wide analysis. Technical report, medRxiv 2021.

[26] A. Okabe, T. Satoh, T. Furuta, A. Suzuki, and K. Okano. Generalized network Voronoi diagrams: Concepts, computational methods, and applications. Int. J. Geogr. Inf. Sci. 22 (2008), no. 9, 965–994.

[27] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time, J ACM 46 (1999), no. 3, 362–394.

[28] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications, Theor. Comput. Sci. 348 (2005), no. 2-3, 357–365.

How to cite this article: G. Ducoffe, Balancing graph Voronoi diagrams with one more vertex, Networks. 83 (2024), 368–389. https://doi.org/10.1002/net.22198