Minimizing and Computing the Inverse Geodesic Length on Trees

Serge Gaspers
UNSW Sydney, Australia
Data61, CSIRO, Sydney, Australia
sergeg@cse.unsw.edu.au

Joshua Lau[1]
UNSW Sydney, Australia
joshua.lau@unsw.edu.au

Abstract

For any fixed measure H that maps graphs to real numbers, the MinH problem is defined as follows: given a graph G, an integer k, and a target τ, is there a set S of k vertices that can be deleted, so that $H(G-S)$ is at most τ? In this paper, we consider the MinH problem on trees.

We call H *balanced on trees* if, whenever G is a tree, there is an optimal choice of S such that the components of $G-S$ have sizes bounded by a polynomial in n/k. We show that MinH on trees is Fixed-Parameter Tractable (FPT) for parameter n/k, and furthermore, can be solved in subexponential time, and polynomial space, whenever H is additive, balanced on trees, and computable in polynomial time.

A particular measure of interest is the Inverse Geodesic Length (IGL), which is used to gauge the efficiency and connectedness of a graph. It is defined as the sum of inverse distances between every two vertices: $IGL(G) = \sum_{\{u,v\}\subseteq V} \frac{1}{d_G(u,v)}$. While MinIGL is $W[1]$-hard for parameter treewidth, and cannot be solved in $2^{o(k+n+m)}$ time, even on bipartite graphs with n vertices and m edges, the complexity status of the problem remains open in the case where G is a tree. We show that IGL is balanced on trees, to give a $2^{O\left((n\log n)^{5/6}\right)}$ time, polynomial space algorithm.

The *distance distribution* of G is the sequence $\{a_i\}$ describing the number of vertex pairs distance i apart in G: $a_i = |\{\{u,v\} : d_G(u,v) = i\}|$. Given only the distance distribution, one can easily determine graph parameters such as diameter, Wiener index, and particularly, the IGL. We show that the distance distribution of a tree can be computed in $O(n\log^2 n)$ time by reduction to polynomial multiplication. We also extend the result to graphs with small treewidth by showing that the first p values of the distance distribution can be computed in $2^{O(\mathrm{tw}(G))}n^{1+\varepsilon}\sqrt{p}$ time, and the entire distance distribution can be computed in $2^{O(\mathrm{tw}(G))}n^{1+\varepsilon}$ time, when the diameter of G is $O(n^{\varepsilon'})$ for every $\varepsilon' > 0$.

2012 ACM Subject Classification Mathematics of computing → Trees; Mathematics of computing → Graph algorithms

Keywords and phrases Trees, Treewidth, Fixed-Parameter Tractability, Inverse Geodesic Length, Vertex deletion, Polynomial multiplication, Distance distribution

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.59

Related Version A full version of the paper is available: `https://arxiv.org/abs/1811.03836`.

Funding *Serge Gaspers*: Serge Gaspers is the recipient of an Australian Research Council (ARC) Future Fellowship (FT140100048).

Acknowledgements We thank David Harvey and Ray Li for fruitful discussions and feedback.

[1] Corresponding author

1 Introduction

The *Inverse Geodesic Length (IGL)* is a widely-used measure for quantifying the connectedness and efficiency of a given graph or network. In mathematical chemistry, it is also known as the *Harary Index* [39], and in network science as the *(global) efficiency* [14].

To test the resilience of a graph to vertex failures, the problem of minimizing a particular measure by deleting a fixed number of vertices has been studied extensively [29, 25, 20]. In these cases, heuristics have been used to choose which vertices to delete, and their effect has been assessed using the chosen measure. In particular, Szczepánski et al. [34] chose IGL as the measure to be minimized when examining this problem. Nonetheless, only recently has the exact optimization problem itself (MinIGL) been studied.

Veremyev et al. [36] formulated MinIGL as a special case of the Distance-Based Critical Node Detection Problem (DCNP), and reduced the problem to Integer Linear Programming. Aziz et al. [4] observed that MinIGL is NP-complete, since it corresponds to Vertex Cover when $\tau = 0$, but it is also both NP-complete, and $W[1]$-hard for parameter k, on both split and bipartite graphs. Najeebullah [30] showed that, under the Exponential Time Hypothesis of Impagliazzo and Paturi [22], MinIGL cannot be solved in $2^{o(k+n+m)}$ time, even on bipartite graphs. On the positive side, it was shown that MinIGL is Fixed-Parameter Tractable (FPT) for parameter twin (or vertex) cover number, and also for $\omega + k$, where ω is the neighbourhood diversity of the graph. In another paper, Aziz et al. [3] showed that MinIGL is $W[1]$-hard for parameter treewidth. The complexity status of MinIGL when the input graph is a tree was stated as an open question by Aziz et al. [4, 3], and in open problem sessions of IWOCA 2017 and the Sydney Algorithms Workshop 2017.

In Section 3, we examine MinIGL on trees, giving the following results.

▶ **Theorem 1.1.** MinIGL *is FPT for parameter n/k on trees.*

▶ **Theorem 1.2.** *There is a $2^{O((n \log n)^{5/6})}$ time, $O(n^3)$ space algorithm for* MinIGL *on trees, on a real RAM.*

To do so, we prove more general versions of these results, for the MinH problem in the case when H is additive, balanced on trees, and computable in polynomial time.

We give a Dynamic Programming (DP) algorithm that solves MinIGL by matching ordered trees to the structure of the given tree, to give a forest with $n - k$ vertices and minimum IGL. The running time of this algorithm is exponential in L, but polynomial in n, where L is the size of the largest tree in this forest. Since H is balanced, L is bounded by a polynomial in n/k, so MinH is FPT for parameter n/k. Proving that IGL is balanced on trees then gives Theorem 1.1. Choosing this DP algorithm when k is large compared to n, and a simple brute-force algorithm otherwise, gives Theorem 1.2.

IGL has been used to identify key protein residues [9], compare the robustness of botnet structures [16], and assess the impact of attacks on power grids [40]. Thus, the ability to compute the IGL of a graph efficiently serves practical purpose in identifying characteristics of real-world networks.

Since the IGL of a graph can easily be computed from its distance distribution, we examine the problem of computing the distance distribution of trees. By combining the relatively well-known techniques of centroid decomposition and fast polynomial multiplication, we obtain the following result on trees.

▶ **Theorem 1.3.** *The distance distribution of a tree with n vertices can be computed in $O(n \log^2 n)$ time on a log-RAM.*

We extend this result to graphs with small treewidth. This is of practical note, as real-world graphs for which IGL is an indicator of strength – such as electrical grids [2] and road transport networks [27] – have been found to have relatively small treewidth.

The distance distribution of a graph can be trivially computed from the All Pairs Shortest Paths (APSP). The output of APSP is of size n^2, so any APSP algorithm requires $\Omega(n^2)$ time. On graphs with treewidth k, APSP can be computed in $O(kn^2)$ time [31], so we seek algorithms that find the distance distribution with a subquadratic dependence on n. Abboud et al. [1] proved that, under the Orthogonal Vectors Conjecture (OVC), there is no algorithm that distinguishes between graphs of diameter 2 and 3 in $2^{o(k)}n^{2-\varepsilon}$ time. Williams [38] showed that the OVC is implied by the Strong Exponential Time Hypothesis (SETH) of Impagliazzo, Paturi and Zane [22, 23]. Since the distance distribution of a graph immediately gives its diameter, this hardness result also applies to computing the distance distribution. We prove the following result.

▶ **Theorem 1.4.** *The prefix $a_1, \ldots, a_p$ of the distance distribution of a graph with n vertices and treewidth k can be computed in $2^{O(k)}n^{1+\varepsilon}\sqrt{p}$ time on a log-RAM, for any $\varepsilon > 0$.*

In particular, the number of relevant values of p is at most the graph's diameter, so when the diameter is $O(n^{\varepsilon'})$ for every $\varepsilon' > 0$, we obtain a $2^{O(k)}n^{1+\varepsilon}$ time algorithm to compute the distance distribution. This matches the known hardness bounds above, in the sense that under the OVC, (or the stronger SETH), the dependence on k must be $2^{\Omega(k)}$ when the dependence on n is subquadratic.

Cabello and Knauer [12] reduced the problem of computing the Wiener index [37] (the sum of distances between every two vertices) to orthogonal range queries in $k-1$ dimensions. They did so by applying a divide-and-conquer strategy that divides the graph with small separators that are found efficiently. Abboud et al. [1] adapted this approach to find radius and diameter. We take a similar approach, but reduce computing the distance distribution to the following problem rather than to orthogonal range queries.

If v and w are vectors in $\mathbb{R}^d$, write $v < w$ if each coordinate of w is strictly greater than the corresponding coordinate in v. In this case, we say that w (strictly) *dominates* v. We define the RedBluePolynomial problem as follows.

RedBluePolynomial	
Input:	r red points $R_1, \ldots R_r$, and b blue points $B_1, \ldots, B_b$ in $\mathbb{R}^d$, along with corresponding non-negative integer values $r_1, \ldots, r_r$, and $b_1, \ldots, b_b$, respectively.
Question:	Determine the non-zero coefficients of the polynomial $\sum_{(p,q):R_p<B_q} x^{r_p+b_q}$, as a list of (exponent, coefficient) pairs.

This problem can be solved naively in quadratic time, but we seek a more efficient solution in the case when the value of each point is bounded.

To our knowledge, this problem is new, and a variant of a well-known counting problem, which asks for the number of red points dominated by each blue point. Chan and Pătraşcu [13] showed that this variant can be solved in $O(n\sqrt{\log n})$ time on a Word RAM, using word operations to facilitate efficient counting. Bentley [6] gave a multidimensional divide-and-conquer approach for a similar problem, which Monier [28] showed had complexity $O(dn \cdot B(n,d))$ where $B(n,d) = \binom{d+\lceil \log n \rceil}{d}$.

Bringmann et al. [11] used this fact to show that the method employed by Cabello and Knauer [12], and Abboud et al. [1] can, in fact, be used to compute the Wiener index, radius, and diameter of graphs with treewidth k in $2^{O(k)}n^{1+\varepsilon}$ time for any $\varepsilon > 0$, by proving that $B(n,k) = 2^{O(k)}n^{\varepsilon}$. Furthermore, Husfeldt [21] gave an improved $2^{O(k)}n$ time algorithm for

computing diameter and radius in the case where the graph also has constant diameter. However, it was noted that this result only pertains to the existence of pairs of vertices at certain distances, and not to counting the number of such pairs. Thus, the result does not directly give further insight to computing distance distributions.

We follow Bentley's method, where it suffices to consider the one-dimensional case, $d = 1$. We resolve this case using square-root decomposition and fast polynomial multiplication. Applying the approach of Bringmann et al. to analyse the running time of this approach gives Theorem 1.4. A detailed discussion of this algorithm is given in Section 4.

Due to space constraints, we omit and abbreviate proofs to some of the more straightforward results, and refer the reader to the full version of this paper for more details.

2 Preliminaries

Let $G = (V, E)$ be a graph and suppose $u, v, w \in V$. We define the *distance* $d_G(u, v)$ between u and v to be the fewest number of edges in any path from u to v, or ∞ if no such path exists, with the convention that $\frac{1}{\infty} = 0$.

In Section 3, we consider the problem when the provided graph is a tree T. In this case, precisely one simple path exists between every pair $\{u, v\} \subseteq V$. Define $\mathcal{P}_T(u, v)$ to be the set of vertices along the simple path from u to v in T, including the endpoints u and v.

Observe that $d_T(u, w) + d_T(w, v) = d_T(u, v)$ if and only if $w \in \mathcal{P}_T(u, v)$. For a vertex w, we also define $\mathcal{P}_T^{-1}(w)$ to be the set of all (unordered) pairs of vertices whose path in T passes through w. Formally, $\mathcal{P}_T^{-1}(w) = \{\{u, v\} \subseteq V : w \in \mathcal{P}_T(u, v)\}$.

A vertex u is a *centroid* of T if the maximum size of a connected component in $T - u$ is minimized. We will use the following results, concerning centroids.

▶ **Lemma 2.1** (Jordan [24]). *Every tree has either one centroid or two adjacent centroids. If a centroid is deleted from a tree, each tree in the remaining forest contains no more than* $\frac{n}{2}$ *vertices, where* n *is the number of vertices in the original tree.*

▶ **Lemma 2.2.** *Let* u *be a centroid of a tree* T *with* $n \geq 2$ *vertices. Then,* $|\mathcal{P}_T^{-1}(u)| \geq \frac{n^2}{4}$.

Proof. See the full version of the paper. ◀

In Section 4 we also consider the problem of computing the IGL, using the tree decompositions of graphs with small treewidth. A *tree decomposition* of G is a tree $\mathcal{T}$ whose vertices (called *nodes*) are $\{1, \ldots, I\}$ and a sequence $\mathcal{V}_1, \ldots, \mathcal{V}_I$ of subsets of V (called *bags*) such that

1. $V = \bigcup_{i=1}^{I} \mathcal{V}_i$;
2. If $uv \in E$, then $\{u, v\} \subseteq \mathcal{V}_i$ for some i;
3. $\mathcal{V}_a \cap \mathcal{V}_c \subseteq \mathcal{V}_b$ whenever $b \in \mathcal{P}_{\mathcal{T}}(a, c)$.

The *width* of such a tree decomposition is $\max_{i=1}^{I} |\mathcal{V}_i| - 1$. The *treewidth* $\mathrm{tw}(G)$ of G is the minimum width among all tree decompositions of G.

2.1 Model of computation

We establish our results on models of computation that closely reflect what is available to programmers of high-level languages on physical computing devices today.

In Section 3, we solve MinIGL by explicitly computing the minimum IGL that can be obtained by deleting k vertices from the given tree. We perform this on the *real RAM* formulated by Shamos [33], which allows addition, subtraction, multiplication, division and

comparisons of real numbers in constant time, but does not support rounding a value to the nearest integer, or modulo as native operations. This permits efficiently adding and comparing contributions of distances between vertices to the IGL.

In Section 4, we reduce the problem of computing the IGL of a graph to finding its distance distribution. We solve this on a *log-RAM* introduced by Fürer [19], which is a Word RAM that also supports constant time arithmetic operations (including multiplication, integer modulo, and division) on words of length $O(\log n)$. Fürer showed that on a log-RAM, multiplication of two n-bit integers can be done in $O(n)$ time, using either the approach of Schönhage and Strassen [32] (performing a complex polynomial-based Fast Fourier Transform (FFT) and maintaining sufficient precision), or that of Fürer [18] (performing an FFT over a ring of polynomials).

We extend this to integer polynomials with bounded coefficients, as follows.

▶ **Lemma 2.3.** *Suppose P and Q are integer polynomials of degree n whose coefficients are non-negative integers, such that their product PQ has coefficients not exceeding some integer m. Then, the coefficients of PQ can be computed from the coefficients of P and Q in $O(n \log m)$ time on a log-RAM.*

Proof. This can be done using Kronecker substitution [26]. See the full version of the paper for further details. ◀

3 MinIGL on Trees

In this section, we give a new subexponential time, polynomial space algorithm for MinH on trees, when H satisfies the following properties. We use this to prove Theorems 1.1 and 1.2, by showing that IGL also satisfies these properties.

▶ **Definition 3.1** (Additivity). *We say that a measure H on graphs is* additive *if $H(G_1 \oplus G_2) = H(G_1) + H(G_2)$ for any vertex-disjoint graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, where $G_1 \oplus G_2$ is the graph $(V_1 \dot\cup V_2, E_1 \dot\cup E_2)$.*

Call a forest *L-trimmed* if none of its trees contain more than L vertices. In the same way, call a subset of vertices in a tree *L-trimming* if their deletion gives an L-trimmed forest.

▶ **Definition 3.2** (Balanced on trees). *We say that a measure H is* balanced (on trees) *if there exist positive constants c_H and t_H, such that, for any* Yes*-instance (T, k, τ) of* MinH *on a tree T with n vertices, there exists a witness that is $c_H\,(n/k)^{t_H}$-trimming.*

Hereafter, we will assume that the value of H on a forest is computable in $O(n^\alpha)$ time, and $O(n^\beta)$ space, on a real RAM, where $\alpha, \beta \geq 1$ are constants. We also assume that such a value can be stored in a constant number of words on a real RAM.

We prove Theorems 1.1 and 1.2 by giving compatible algorithms for MinH on trees, then complete the proof by showing that IGL satisfies the same properties that H does. Now it is easy to see that there is a naïve, brute-force algorithm for MinH.

▶ **Lemma 3.3.** *There is an $O(n^{k+\alpha})$ time, $O(n^\beta)$ space algorithm for* MinH *on a tree, on a real RAM.*

Proof. We simply try all $\binom{n}{k} = O(n^k)$ subsets of k vertices. The value of H on the forest that remains after each subset has been removed can be computed in $O(n^\alpha)$ time and $O(n^\beta)$ space. ◀

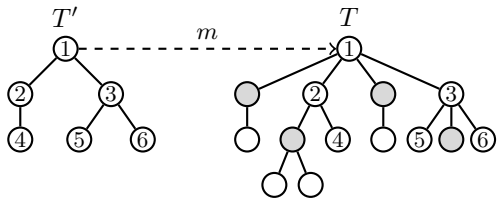

Figure 1 Mapping the vertices of T' to T in Lemma 3.4. Note that T' is an ordered tree, and that children (and their subtrees) must be mapped in order. Shaded vertices will be deleted, and we recursively solve for the subtrees rooted at their children.

If k is small, this algorithm may be efficient. When k is large, the vertices forming an optimal solution will leave a forest of relatively small trees after they are deleted, since H is balanced. We use this property to develop an alternate, more efficient algorithm for MinH in this case. Let $L = c_H (n/k)^{t_H}$. Our algorithm minimizes H, considering only L-trimming subsets of k vertices. The running time of this algorithm is exponential in L, but polynomial in n, so it is fast when k is large, relative to n.

▶ **Lemma 3.4.** *Let $T = (V, E)$ be a tree with n vertices. There is an $O\left(\frac{4^L}{\sqrt{L}}\left(n^2 + L^{\alpha-1}\right)\right)$ time, $O\left(nkL + L^\beta\right)$ space algorithm on a real RAM, which finds the minimum value of $H(T - S)$, among all L-trimming subsets S of k vertices.*

Proof. We root T arbitrarily and employ DP to compute this minimum value for every subtree and budget, in two cases: the case where the root of the subtree is deleted, and the case where it is not. Denote these minimum values by $f(u, b)$ and $g(u, b)$, respectively, for the subtree rooted at u and budget b. The leaves of the tree form the base cases for this algorithm, and the final answer is derived from the minimum of $f(\text{root}, k)$ and $g(\text{root}, k)$. It remains to give recurrences for f and g.

In the case where u is deleted, we simply need to distribute the remaining $b - 1$ deletions among the subtrees rooted at each child of u. Let the children of u be $v_1, \ldots, v_{ch_T(u)}$ in a fixed order. Our recurrence takes the form of another DP algorithm: let $f'(u, i, b')$ be the minimum value of f distributing a budget of b' deletions among the subtrees rooted at the first i children of u. Our recurrence is as follows:

$$f'(u, i, b') = \min_{0 \le b'' \le b'} (\min(f(v_i, b''), g(v_i, b'')) + f'(u, i - 1, b' - b''))$$

and we have that $f(u, b) = f'(u, ch_T(u), b - 1)$.

If u is not deleted, it will be the root of some tree with no more than L vertices after our chosen subset has been deleted. We fix the structure (formally, an *ordered tree*) for this rooted tree, and attempt to match the vertices in this structure to vertices in the subtree rooted at u. Formally, let the structure be an ordered tree T' over $L' \le L$ vertices. Let its vertex set be $V' = \{1, \ldots, L'\}$ and, without loss of generality, suppose 1 is its root. We seek a total, injective mapping $m : V' \to V$ satisfying the following conditions.

1. $m(1) = u$;
2. Suppose p and p' are the parents of q and q' in T and T', respectively. If $m(q') = q$ then $m(p') = p$;
3. Let p and p' be vertices in T and T' such that their children are, in order, $q_1, \ldots, q_{ch_T(p)}$ and $q'_1, \ldots, q'_{ch_{T'}(p')}$, respectively. If $m(q'_{j_1}) = q_{i_1}$, $m(q'_{j_2}) = q_{i_2}$ and $j_1 \le j_2$, then $i_1 \le i_2$. That is, children are matched in order.

Note that the structure of the chosen ordered tree uniquely characterises the value of H on the component containing u, since H is only defined on unlabelled graphs, and is additive, so this value is independent of the structure of other components.

Let v be some vertex in T. If v is mapped to by m, then v is a part of this component. Otherwise, if v is not mapped to by m but its parent is, then v must be a vertex chosen for deletion, and so we should recursively consider each of its childrens' subtrees.

This implies a DP approach to determine the optimal choice of m, similar to that of f'. We let $g'(u, i, b', u', j)$ be the minimum value (of H) induced by a mapping which maps u' to u and maps the first j children of u' among the first i children of u with a total budget of b' deletions in the subtree rooted at u. This value *does not* include the contributions of vertex pairs in T which both end up in the current component (are mapped to by m).

We have a choice to either delete the ith child v_i of u, or map it to the jth child v'_j of u'. In both cases, we allocate a budget of $b'' \leq b'$ deletions to the subtree rooted at v_i. This gives the following recurrence:

$$g'(u, i, b', u', j) = \min_{0 \leq b'' \leq b'} \min \begin{cases} g'(u, i-1, b'-b'', u', j) + f(v_i, b''), \\ g'(u, i-1, b'-b'', u', j-1) + g'(v_i, ch_T(v_i), b'', v'_j, ch_{T'}(v'_j)) \end{cases}$$

and $g(u, b) = \min_{T'}(H(T') + g'(u, ch_T(u), b, 1, ch_{T'}(1)))$. This concludes the description of the algorithm.

A detailed analysis of the time and space complexity of this algorithm is given in the full version of the paper, using the key result that there are $O\left(\frac{4^L}{L\sqrt{L}}\right)$ unlabelled, ordered trees on L or fewer vertices [17, 35]. ◀

Since $n^2 + L^{\alpha-1} = O\left(n^{\max(2,\alpha-1)}\right)$, and $L \leq c_H \left(\frac{n}{k}\right)^{t_H}$, it follows that MinH is FPT for parameter $\frac{n}{k}$.

▶ **Corollary 3.5.** *Suppose H is a measure on graphs, that is additive, balanced on trees, and computable in polynomial time on trees, on a real RAM. Then* MinH *is FPT for parameter n/k on trees.*

With an appropriate threshold, we can combine the approaches of Lemma 3.3 and Lemma 3.4 to give a subexponential time, polynomial space algorithm for MinH.

▶ **Corollary 3.6.** *Suppose H is a measure on graphs, that is additive, balanced on trees, and computable in polynomial time on trees, on a real RAM. Then there is a $2^{O\left((n \log n)^{t_H/(t_H+1)}\right)}$ time, polynomial space algorithm for* MinH *on trees, where t_H is the constant given in Definition 3.2.*

Proof. Lemma 3.3 gives us an $O(n^{k+\alpha}) = 2^{O(k \log n)}$ time algorithm for MinH on a tree. Lemma 3.4 gives us an alternate $O\left(4^{c_H (n/k)^{t_H}} n^{\max(2,\alpha-1)}\right) = 2^{O\left((n/k)^{t_H} + \log n\right)}$ time algorithm for the same problem. Note that the memory consumption of both algorithms is bound by $O(n^{\max(3,\beta)})$, so they are both polynomial in space.

Let $k^* = n^{t_H/(t_H+1)} \log^{-1/(t_H+1)} n$. We select the former algorithm when $k \leq k^*$, and the latter algorithm otherwise. In both cases, our running time is bound by $2^{O\left((n \log n)^{t_H/(t_H+1)}\right)}$, as required. ◀

We now prove that IGL satisfies the requirements of Corollary 3.5 and Corollary 3.6. IGL is clearly additive, since pairs of vertices belonging to different components contribute $\frac{1}{\infty} = 0$ to the IGL. We can easily compute the IGL in $O(n^2)$ time, and $O(n)$ space, on the

real-RAM by traversing from each vertex. Hence, it remains to show that IGL is balanced on trees: it suffices to show that there is a constant t_{IGL}, such that any subset of vertices whose deletion minimizes the IGL is $O\left((n/k)^{t_{\text{IGL}}}\right)$-trimming.

To do so, we choose to reason about the decrease in IGL caused by the removal of a subset of k vertices, rather than the IGL itself. Maximizing this decrease (which we call *utility*) is equivalent to minimizing the IGL of the graph after removal.

▶ **Definition 3.7** (Utility). *Let $G = (V, E)$ be a graph. Then the utility of some $S \subseteq V$ is:*

$$\mathcal{U}_G(S) = IGL(G) - IGL(G - S).$$

If $S = \{v\}$, we write $\mathcal{U}_G(v)$ instead of $\mathcal{U}_G(\{v\})$, which we call the utility of v in G.

Suppose $S = S' \dot\cup \{v\}$ is a subset of k vertices in a tree T with maximum utility. Necessarily, v must have maximum utility in $T - S'$. This means that v has no less utility than any vertex in its component in $T - S'$, and that it also has no less utility than the optimal vertex in any other component. In this vein, we would like to consider the case when $k = 1$ so we can reason about the individual optimality of each vertex in an optimal solution.

We use the following upper and lower bounds on the utility of the optimal choice of vertex in this case. The proofs of these bounds are straightforward, and provided in the full version of the paper.

▶ **Lemma 3.8.** *Let $T = (V, E)$ be a tree with $n \geq 2$ vertices. Then, $\max_{v \in V} \mathcal{U}_T(v) \geq n/2$.*

▶ **Lemma 3.9.** *Let $G = (V, E)$ be a tree with n vertices. Then $\mathcal{U}_G(v) \leq IGL(G) \leq \frac{1}{2}n(n-1)$ for any vertex $v \in V$.*

Next, we show that the removal of a vertex with maximum utility leaves the remaining forest somewhat balanced. Specifically, it is never the case that one tree in this forest is so large that it contains all but $o(n^{1/4})$ vertices.

▶ **Theorem 3.10.** *Let $T = (V, E)$ be an unweighted tree with $n \geq 3$ vertices and suppose $v \in V$ minimizes $IGL(T - v)$. Further, suppose C is a connected component in $T - v$ containing l vertices and let $r = n - l - 1$ be the number of vertices in $T - v$ not in C. Then, there is a constant $0 < c < 1$ independent of n such that $r \geq cn^{1/4}$.*

Proof. We may assume $l \geq 1$, since the case when $l = 0$ is trivial. We may also assume that $r \geq 1$, since if $r = 0$, v is a leaf, which contradicts its optimality since $n \geq 3$.

Since T is a tree, each neighbour of v belongs to a different component in $T - v$. Suppose x_C is the neighbour of v in C and let C' be the subtree $T[V(C) \cup \{v\}]$. Thus, v is a leaf of C'. We use this structure (pictured in Figure 2) to give two different, but related, upper bounds for the utility $\mathcal{U}_T(v)$ of v in T.

▷ Claim 3.11. $\mathcal{U}_T(v) \leq \frac{1}{2}r(r+1) + (r+1)\mathcal{U}_{C'}(v)$.

Proof. Let us upper bound $\mathcal{U}_T(v)$ by considering the utility of v in C' and also in $T - V(C)$. There are $n - l$ vertices in $T - V(C)$, so by Lemma 3.9, we have that $\mathcal{U}_{T-V(C)}(v) \leq \frac{1}{2}(n-l)(n-l-1) = \frac{1}{2}r(r+1)$. This accounts for the pairs of vertices disconnected by the deletion of v in $T - V(C)$.

We still need to consider such pairs where one vertex is in C, and the other is in $T - V(C)$ (this includes v). Since v is a leaf in C', the only pairs of vertices connected in C' that are disconnected in $C = C' - v$ are those of the form $\{v, v_C\}$, where v_C ranges over $V(C)$.

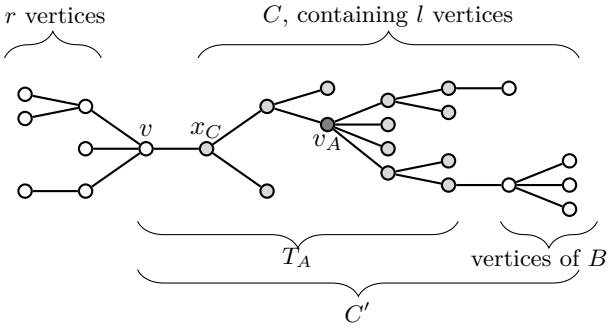

Figure 2 Layout of the vertices of T, in Theorem 3.10. Shaded vertices are in A, and are no more than $D = 5$ away from v. The value of D here has chosen for example's sake, and is not the true value constructed in the proof.

Now let u be a vertex in $V \setminus V(C)$. The path from u to v_C must pass through v, and thus $d_T(u, v_C) \geq d_T(v, v_C)$. Hence, the contribution of each disconnected $\{u, v_C\}$ pair is at most that of $\{v, v_C\}$ towards $\mathcal{U}_T(v)$. Putting these inequalities together gives us

$$\begin{aligned}
\mathcal{U}_T(v) &= \sum_{\{p,q\} \in \mathcal{P}_T^{-1}(v)} \frac{1}{d_T(p,q)} \\
&= \mathcal{U}_{T-V(C)}(v) + \sum_{\substack{u \in V \setminus V(C) \\ v_C \in V(C)}} \frac{1}{d_T(u, v_C)} \\
&\leq \frac{1}{2} r(r+1) + |V \setminus V(C)| \sum_{v_C \in V(C)} \frac{1}{d_T(v, v_C)} \\
&\leq \frac{1}{2} r(r+1) + (r+1) \sum_{v_C \in V(C)} \frac{1}{d_T(v, v_C)} \\
&= \frac{1}{2} r(r+1) + (r+1) \mathcal{U}_{C'}(v),
\end{aligned}$$

as required. ◁

▷ Claim 3.12. $\mathcal{U}_T(v) \leq rn$.

Proof. Since v is a leaf of C', it is distance 1 away from its sole neighbour, and only this neighbour, in C'. Also, the only pairs disconnected by v's removal in C' are those containing v itself. Now there are $l - 1$ other vertices in C', each at least distance 2 away from v. Hence, $\mathcal{U}_{C'}(v) \leq 1 + \frac{l-1}{2} = \frac{l+1}{2} = \frac{n-r}{2}$.

Since $r \geq 1$, we know that $r + 1 \leq 2r$. Hence, by Claim 3.11

$$\begin{aligned}
\mathcal{U}_T(v) &\leq r^2 + 2r\mathcal{U}_{C'}(v) \\
&\leq r^2 + r(n-r) \\
&= rn,
\end{aligned}$$

as required. ◁

Since the utility of deleting v is maximal among all vertices, and $n \geq 2$, we know $\mathcal{U}_T(v) \geq n/2$ from Lemma 3.8. Combining this with Claim 3.11 and rearranging gives

$$\mathcal{U}_{C'}(v) \geq \frac{n - r(r+1)}{2(r+1)}. \tag{1}$$

Suppose, for a contradiction, that $r < \frac{1}{15}n^{1/4}$. Since r is purported to be relatively small, $\mathcal{U}_{C'}(v)$ must be rather large (note it is proportional to n). Intuitively, this implies that many vertices in C' are close to v, and hints towards a more central choice of vertex to delete. We will formally show that such a vertex exists, and is a more optimal choice.

Fix some distance D. We can divide the vertices of C into two groups, A and B: those at most distance D from v in C' (and thus, also in T) and those that are not, respectively. Suppose that $|A| = t$ and that $|B| = |V(C)| - t$. We have the following upper bound:

$$\mathcal{U}_{C'}(v) \leq t + \frac{|V(C)| - t}{D+1} \leq t + \frac{n-t}{D+1}, \tag{2}$$

because each vertex in B is at least distance $D+1$ away from v, and $|V(C)| \leq n$. Note that we do not account for v itself, since the distance to itself does not contribute to its utility.

Recall that $r < \frac{1}{15}n^{1/4}$. It is easy to see that $r(r+1) \leq n/2$. Combining this with (1) and (2) gives us the following inequality:

$$\frac{n}{4(r+1)} \leq \mathcal{U}_{C'}(v) \leq t + \frac{n-t}{D+1},$$

from which we can obtain

$$tD \geq \frac{n(D+1)}{4(r+1)} - n.$$

If we choose $D = 8(r+1) - 1$, it holds that $t \geq \frac{n}{D} = \frac{n}{8r+7} \geq \frac{n}{15r}$.

Consider the subgraph (a tree) T_A induced by the vertex set $A \dot\cup \{v\}$. T_A contains at least two vertices as v and x_C both must be in A. Also, since T_A is a tree, by Lemma 2.1 it must have a centroid. Let one of the centroids of T_A be v_A. The diameter of T_A is at most $2D$, since every vertex in T_A is at most distance D from v. Combining this with Lemma 2.2, we have

$$\mathcal{U}_{T_A}(v_A) \geq \frac{t^2}{8D} \geq \frac{n^2}{8D^3} \geq \frac{n^2}{8(15)^3 r^3}.$$

Now every pair in T_A that is disconnected by the deletion of v_A is also disconnected in T by the deletion of v_A, so $\mathcal{U}_{T_A}(v_A) \leq \mathcal{U}_T(v_A)$. Also, by the optimality of v in T, we have that $\mathcal{U}_T(v_A) \leq \mathcal{U}_T(v)$. Hence, using the result of Claim 3.12, we can conclude that

$$\frac{n^2}{8(15)^3 r^3} \leq \mathcal{U}_{T_A}(v_A) \leq \mathcal{U}_T(v) \leq rn.$$

Thus, we have that $r^4 \geq \frac{n}{8(15)^3} \geq \frac{n}{15^4}$, so $r \geq \frac{1}{15}n^{1/4}$, which is a contradiction. The result follows with a choice of $c = \frac{1}{15}$. ◀

We can use this result to finally upper bound the number of vertices in any component after an optimal set of vertices has been removed.

▶ **Theorem 3.13.** *Let $T = (V, E)$ be a tree with n vertices, and let $S \subseteq V$ be some subset of vertices such that $|S| = k \geq 1$. There exists a positive constant c', independent of T and k, such that whenever S minimizes $IGL(T - S)$, S is $\left(c'\,(n/k)^5\right)$-trimming.*

Proof. We will call the components of $T - S$ *remaining components* and denote each of them by their vertex set. Suppose the remaining components are $R = \{R_1, R_2, \ldots, R_{|R|}\}$, where $R_i \subseteq V$ and $S \dot\cup R_1 \dot\cup \ldots \dot\cup R_{|R|} = V$. We need to show that $|R_i| \leq c'\,(n/k)^5$ for each R_i.

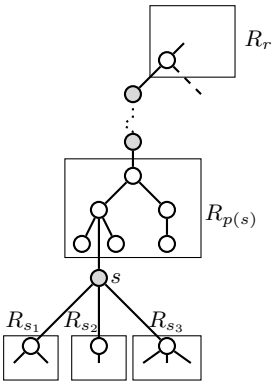

Figure 3 Bounding the size of the largest remaining component in Theorem 3.13. In this case, the parent of s is not the root and s has $ch(s) = 3$ children in T'. The shaded vertices are those in S.

We first construct a new graph $T' = (V', E')$ by collapsing each of the remaining components. Formally, $V' = R \dot\cup S$, and, for each $R_i \in R$ and $s \in S$, $\{R_i, s\} \in E'$ if and only if there exists some $r \in R_i$ such that $\{r, s\} \in E$. It can be seen that T' is necessarily a tree, and that every R_i is only incident to elements in S. For the remainder of the proof, we further assume that every element in S is only incident to remaining components in T': if this is not the case, one can add a "dummy" remaining component with cardinality zero between every pair of adjacent elements of S in T'.

Let R_r be a remaining component containing at least as many vertices as any other remaining component. Note that $|R_r| > 0$: it is never a "dummy". It suffices to show the upper bound holds for R_r. We root T' at R_r. Since $k > 0$, there are strictly fewer than n vertices among the remaining components R. Hence, by the Pigeonhole Principle, there must be some $s \in S$ such that the children $R_{s_1}, R_{s_2}, \ldots, R_{s_{ch(s)}}$ of s in T' together contain fewer than n/k vertices. Let the parent of s in T' be $R_{p(s)}$. See Figure 3.

Since S is optimal, s must be an optimal choice of vertex to delete in an instance of MinIGL with graph $T - (S \setminus \{s\})$ and a budget of 1 deletion. In particular, it must also be the optimal choice of vertex to delete in the component containing s in $T - (S \setminus \{s\})$. Hence, we may apply Theorem 3.10 to $T - (S \setminus \{s\})$, in that component to give

$$\frac{n}{k} > \sum_{i=1}^{ch(s)} |R_{s_i}| \geq c(|R_{p(s)}| + |R_{s_1}| + \cdots + |R_{s_{ch(s)}}|)^{1/4} \geq c|R_{p(s)}|^{1/4},$$

since $c > 0$, where c is the constant in Theorem 3.10. Thus, we have $|R_{p(s)}| \leq c^{-4} (n/k)^4$.

We now have two cases: if the parent $R_{p(s)}$ of s in T' is the root, R_r, or if it is not the root. If $R_{p(s)}$ is the root, then $p(s) = r$, so $|R_r| \leq c^{-4} (n/k)^4$. Otherwise, s is not a child of the root, and so s must have been a more optimal choice than the best choice in the component induced by R_r in $T - (S \setminus \{s\})$. Since this component contains $|R_r|$ vertices, the best choice had utility at least $\frac{|R_r|}{2}$, by Corollary 3.8. Now the paths that pass through s in $T - (S \setminus \{s\})$ must have one endpoint in some R_{s_j} and the other either in another $R_{s'_j}$ or in $R_{p(s)}$. This is the case since no path can have both endpoints in $R_{p(s)}$. Hence, there are at most $(n/k + 1)(n/k + |R_{p(s)}|)$ such pairs, accounting also for those paths starting at s. Since each of these paths have length at least 1, we have that

$$\frac{|R_r|}{2} \leq \mathcal{U}_{T-(S\setminus\{s\})}(s) \leq \left(\frac{n}{k} + 1\right)\left(\frac{n}{k} + |R_{p(s)}|\right) \leq 2\frac{n}{k}\left(\frac{n}{k} + |R_{p(s)}|\right),$$

because $k \leq n$. Thus

$$\frac{|R_r|}{2} \leq 2\frac{n}{k}\left(\frac{n}{k} + c^{-4}\left(\frac{n}{k}\right)^4\right) \leq 4c^{-4}\left(\frac{n}{k}\right)^5,$$

because $0 < c < 1$. Hence, $|R_r| \leq 8c^{-4}(n/k)^5$ and the result follows with a choice of $c' = 8c^{-4}$. ◀

Thus, we can choose $c_{\mathrm{IGL}} = c'$ and $t_{\mathrm{IGL}} = 5$, showing that IGL is indeed balanced on trees. This gives Theorem 1.1 and Theorem 1.2.

4 Computing the IGL

Computing the IGL of a graph is trivial once its distance distribution has been determined. In this section, we describe algorithms for efficiently computing the distance distribution of trees, and extend these ideas to graphs with small treewidth.

4.1 Trees

To compute the distance distribution on trees, we present a divide-and-conquer method (commonly known as the *centroid decomposition*, as used in [7]) as follows. We pick a vertex and compute the contribution to the distance distribution of all paths passing through that vertex, using fast polynomial multiplication. Then, we delete the vertex from the tree, and recurse on the remaining connected subtrees. We first provide a method that efficiently computes this contribution.

▶ **Lemma 4.1.** *Let $T = (V, E)$ be an unweighted tree with n vertices and suppose $r \in V$. Then, the contribution to the distance distribution of all pairs in $\mathcal{P}_T^{-1}(r)$ can be found in $O(n \log n)$ time on a log-RAM.*

Proof. We begin by rooting the tree at r. Suppose the children of r are $s_1, \ldots, s_{ch(r)}$ and let $S_1, \ldots, S_{ch(r)}$ denote the set of vertices in the subtrees rooted at each child, respectively. With the addition of $S_0 = \{r\}$, the sets S_i form a partition of V.

We perform a depth-first search from r, to find $d_T(r,u) = d_T(u,r)$ for each vertex u and construct a sequence of distance polynomials $P_0, P_1, \ldots, P_{ch(r)}$, where $P_i(x) = \sum_{w \in S_i} x^{d_T(r,w)}$. This takes $O(n)$ time, storing each distance polynomial in coefficient form: there are at most n terms overall. Now let

$$P(x) = \left(\sum_{0 \leq i \leq ch(r)} P_i(x)\right)^2 - \left(\sum_{0 \leq i \leq ch(r)} P_i^2(x)\right) = \sum_{0 \leq j \leq n} b_j x^j.$$

We observe that

$$b_j = 2|\{\{u,v\} \in \mathcal{P}_T^{-1}(r) : u \neq v \text{ and } d_T(u,v) = j\}|, \tag{3}$$

that is, b_j is twice the number of pairs of distinct vertices which have a path of length j passing through r. Thus, the required contribution to the distance distribution can be read off from the coefficient form of $P(x)$. The result follows by computing this efficiently from the coefficients of each P_i by applying Lemma 2.3, and observing that the degree of $\sum_{0 \leq i \leq ch(r)} P_i(x)$, and the sum of the degrees of the P_i's are both at most n. ◀

If we always pick r in Lemma 4.1 to be a centroid of the tree, Lemma 2.1 ensures that each vertex can appear in at most $\log_2 n + 1$ trees throughout the execution of our divide-and-conquer algorithm. A centroid must always exist (also by Lemma 2.1), and we can find one in linear time by recursively computing, then examining, subtree sizes. This gives Theorem 1.3.

▶ **Theorem 1.3.** *The distance distribution of a tree with n vertices can be computed in $O(n \log^2 n)$ time on a log-RAM.*

If we only wish to determine the first p values of the distance distribution of T, we can modify Lemma 4.1 to run in $O(n + p \log n)$ time, by discarding all terms with degree greater than p when constructing the polynomials. Thus, the expensive multiplication step costs $O(p \log n)$ time by Lemma 2.3, and we obtain Theorem 4.2 as a corollary.

▶ **Theorem 4.2.** *The prefix $a_1, \ldots, a_p$ of the distance distribution of a tree with n vertices can be computed in $O(n \log n + p \log^2 n)$ time on a log-RAM.*

4.2 Graphs with small treewidth

Here, we extend the ideas used in the previous section to prove Theorem 1.4.

Let $G = (V, E)$ be an undirected graph with n vertices, whose edges each have a non-negative weight. We describe a modification of the method of Cabello and Knauer [12], to recursively reduce the task of computing the distance distribution of G to solving instances of RedBluePolynomial over points in $O(\mathrm{tw}(G))$ dimensions, with values at most p.

In time $2^{O(k)}n$, we can compute a tree decomposition of G of width at most $k = 5 \cdot \mathrm{tw}(G) + 4$ containing at most $O(kn)$ nodes [10]. Using a common technique, we can transform this decomposition into a *nice* tree decomposition with $N = O(kn)$ nodes (see, for example [15]). The nodes of a nice tree decomposition form a rooted binary tree.

Let A be a subset of vertices. A *portal* of A is a vertex in A which has, as a neighbour, some vertex outside A. If these portals are contained in some set $S \subseteq A$, we can partition the vertices of the graph into three sets: $A \setminus S, S$ and $V \setminus A$, such that every path from a vertex in A to a vertex in $V \setminus A$ passes through some vertex in S.

Since the nice tree decomposition is a binary tree, there is some edge ij in the decomposition whose removal splits the decomposition's tree into two components I and J (containing nodes i and j, respectively), each containing between $\frac{N}{3}$ and $\frac{2N}{3}$ nodes. Let A be the set of vertices that appear in component I, and let S be the intersection $B_i \cap B_j$ of the bags corresponding to nodes i and j. Necessarily, S must contain all the portals of A due to properties of the tree decomposition. Moreover, $|B_i \cap B_j| \leq \min(|B_i|, |B_j|) \leq k + 1$.

Given this fixed A, recursively find the distance distribution among all pairs of vertices in A as follows. First, perform Dijkstra's algorithm from all vertices in S. For every pair of vertices in S, add an edge whose weight equals the length of the shortest path between them. After these edges are added, the length of the shortest path in G between any pair of vertices in A can be found by only considering paths passing through the vertices of A. Hence, we remove all vertices in $V \setminus A$, and recurse on this smaller graph. Note that I is a valid tree decomposition for this new graph, since $i \in I$, and all the added edges have their endpoints in B_i. Thus, we do not need to find another tree decomposition for this new graph, and its treewidth does not exceed k.

In the same way, we recursively find the distance distribution induced by the pairs of vertices in $(V \setminus A) \dot{\cup} S$. Between these two sets of pairs, we have counted pairs of vertices in S twice, so we subtract the distance distribution induced by these pairs using the shortest paths already computed.

Finally, we must compute the distance distribution among shortest paths between the remaining pairs of vertices: these are the pairs in $(A \setminus S) \times (V \setminus A)$. Let the vertices in S be $s_1, \ldots, s_{|S|}$. For every a in $A \setminus S$ and every b in $V \setminus A$, we associate (a, b) with precisely one vertex in s through which some shortest path between the vertices passes. More formally, we will associate (a, b) with the only s_i such that

$$\begin{aligned} d_G(a, s_i) + d_G(s_i, b) &< d_G(a, s_j) + d_G(s_j, b) && \text{for all } j < i, \text{ and} \\ d_G(a, s_i) + d_G(s_i, b) &\leq d_G(a, s_j) + d_G(s_j, b) && \text{for all } j > i. \end{aligned}$$

By rearranging, and observing that all distances are integers, we deduce that this is precisely when

$$\begin{aligned} d_G(a, s_i) - d_G(a, s_j) &< d_G(s_j, b) - d_G(s_i, b) && \text{for all } j < i, \text{ and} \\ d_G(a, s_i) - d_G(a, s_j) &< d_G(s_j, b) - d_G(s_i, b) + 1 && \text{for all } j > i. \end{aligned}$$

Note that all these distances are known from our application of Dijkstra's algorithm from each vertex in S. Since any path from a to b must pass through S, it follows from these inequalities that $d_G(a, s_i) + d_G(s_i, b) = d_G(a, b)$. For each vertex s_i in turn, we will compute the contribution of all pairs of vertices associated with s_i to the distance distribution. We do so by reducing this task to an instance of RedBluePolynomial.

Our instance will have points in $|S| - 1$ dimensions: one dimension for each $j \neq i$. For each $a \in A \setminus S$, create a red point with coordinate $d_G(a, s_i) - d_G(a, s_j)$ in the dimension corresponding to j, and value $d_G(a, s_i)$, corresponding to each $s_j \neq s_i$. Similarly, for each $b \in V \setminus A$, create a blue point with coordinate $d_G(s_j, b) - d_G(s_i, b)$, for each $j < i$, and $d_G(s_j, b) - d_G(s_i, b) + 1$ for each $j > i$, with value $d_G(s_i, b)$. Importantly, we omit any points with value greater than p: these cannot contribute to the prefix we are trying to compute. Hence, we have created no more than n points in all, each with a non-negative integer value no greater than p. The coefficient of x^l produced by our instance of RedBluePolynomial corresponds to the number of pairs associated with s_i that are distance l apart. This concludes the description of our reduction.

Naturally, we now turn our attention to solving RedBluePolynomial. Naively, this can be done in quadratic time by considering every pair of points. However, when values are bounded – such as in our instance – we can solve the problem more efficiently.

For a given instance of RedBluePolynomial, let $n = r + b$ be the total number of points and suppose the value of each point does not exceed some integer $v \geq 0$. Below, we give solutions with time complexity parameterized by both n and v. We consider the 1-dimensional case, then extend this result to higher dimensions recursively.

▶ **Lemma 4.3.** *When $d = 1$, there is an algorithm that solves* RedBluePolynomial *in $O(n\sqrt{v \log n} + n \log n)$ time on a log-RAM.*

Proof. Sort the red and blue points together in non-decreasing order of the coordinate, placing blue points earlier in the order when there are ties. Let t be a positive integer no greater than n. Assign points to groups of size no more than t by placing the first t points, in order, into a group G_1, followed by the next t points in order into a group G_2, and so on, so we create $\lceil \frac{n}{t} \rceil$ groups in all. An example is given in Figure 4.

We will separately consider pairs of points that both belong to the same group, and those that belong to different groups. In each group, consider every pair of points, and check if they contribute a term to the polynomial. This takes $O(nt)$ time over all groups.

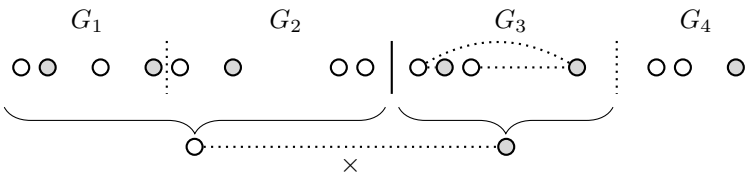

Figure 4 Square root decomposition in Lemma 4.3. The empty dots represent red points, and the shaded dots represent blue points. When processing G_3, we consider pairs of points within the group where the red point precedes the blue point. We then consider cross-group pairs whose blue point is in G_3 using fast polynomial multiplication.

It remains to consider pairs that belong to different groups: call these *cross-group* pairs. For each blue point in G_i, we must add an extra term for each red point among $G_1, \ldots, G_{i-1}$. Thus, the total cross-group contribution of all pairs with a blue point in G_i can be written as the following product of two polynomials.

$$\sum_{B_q \in G_i} \sum_{R_p \in G_1 \cup \cdots \cup G_{i-1}} x^{r_p + b_q} = \left(\sum_{R_p \in G_1 \cup \cdots \cup G_{i-1}} x^{r_p} \right) \left(\sum_{B_q \in G_i} x^{b_q} \right)$$

To compute these contributions, iterate over each group in order, maintaining the coefficient form of the polynomial representing all red points in groups processed thus far. This corresponds to the first multiplicand on the right hand side. We can quickly construct the second multiplicand directly from the elements in this group. Note that the degree of both multiplicands does not exceed v, and that the coefficients of the product do not exceed n^2. Hence, we can compute the product of these two polynomials in $O(v \log n)$ time by Lemma 2.3, so we can compute the cross-group contributions in $O(\frac{n}{t} v \log n)$ time.

Combining these parts with an appropriate choice of t gives the required result. ◀

▶ **Theorem 4.4.** *There is an algorithm that solves* RedBluePolynomial *in* $2^{O(d)} n^{1+\varepsilon} \sqrt{v}$ *time on a log-RAM, for every* $\varepsilon > 0$.

Proof. When $d = 1$, we use Lemma 4.3. Otherwise, we will use the divide-and-conquer method of Bentley [6] to reduce the problem to smaller dimensions.

First, combine the red and blue points into one list and apply divide-and-conquer as follows. Let x_m be the median value among the first coordinate of all points. This can be found in $O(n)$ time [8]. We divide the list into two halves as follows. First assign those points with first coordinate less than x_m into the first half, and those with first coordinate greater than x_m into the second half. Among those with first coordinate precisely x_m, assign blue points to the first half until the first half has $\frac{n}{2}$ points. Assign the remaining points to the second half. This assignment can be done in $O(n)$ time and has the property that if $R_p < B_q$, then either both points belong to the same half, or they belong to the first and second half, respectively.

Next, recursively compute the contribution of both groups to the final polynomial. The remaining pairs that may contribute terms to the result must have a red point in the first half, and a blue point in the second half. Since the ordering guarantees that all points in the first half have a first coordinate no greater than those in the second half, we project the red points in the first half together with the blue points in the second half onto a $(d-1)$-dimensional space by simply ignoring the first coordinate of each point. We then solve RedBluePolynomial for this set of points in $d-1$ dimensions recursively.

The time and space complexity of this algorithm follows from the results of Monier [28] and Bringmann et al. [11], and applying an additional multiplicative factor of $\sqrt{v}$. A full analysis can be found in the full version of this paper. ◀

An analysis of the algorithm we have described in this section gives Theorem 1.4.

▶ **Theorem 1.4.** *The prefix $a_1, \ldots, a_p$ of the distance distribution of a graph with n vertices and treewidth k can be computed in $2^{O(k)}n^{1+\varepsilon}\sqrt{p}$ time on a log-RAM, for any $\varepsilon > 0$.*

Proof. To find the contribution of pairs in $(A \setminus S) \times (V \setminus A)$, we solve $|S| \leq k+1$ instances of RedBluePolynomial in $|S| - 1 \leq k$ dimensions, using the result of Theorem 4.4. As our algorithm performs divide-and-conquer over the nodes of the tree decomposition, each vertex induces the creation of a point in $O((k+1)\log(kn)) = O(k \log n)$ instances of RedBluePolynomial. Hence, since the time complexity of Theorem 4.4 is superadditive with respect to n, the total running time over all instances of RedBluePolynomial is $2^{O(k)}n^{1+\varepsilon} \log n\sqrt{p} = 2^{O(k)}n^{1+\varepsilon'}\sqrt{p}$ for any $\varepsilon' > 0$.

Since we are working on a (nice) tree decomposition with $O(kn)$ nodes, the running time of finding an appropriate dividing edge in the tree, and performing k Dijkstra's per instance are negligible compared to that of solving our instances of RedBluePolynomial. The result follows from the fact that $k = O(\mathrm{tw}(G))$. ◀

This result can easily be extended to directed graphs, and graphs with bounded edge weights, with some modifications, and a suitable choice of p. On graphs with unit weight edges, setting $p = n - 1$ determines the entire distance distribution.

▶ **Corollary 4.5.** *The distance distribution of an undirected graph G with n vertices, edges of unit weight and treewidth $tw(G)$ can be computed in $2^{O(tw(G))}n^{3/2+\varepsilon}$ time on a log-RAM.*

5 Conclusion

We have provided a general method to solve MinH on trees in subexponential time and polynomial space, whenever H is additive, balanced on trees, and computable in polynomial time. We used this to give a $2^{O((n \log n)^{5/6})}$ time, polynomial space algorithm for MinIGL, by proving that IGL is balanced on trees. Our proof ideas can be used to show that other measures (such as the Wiener index), are also balanced on trees.

For graphs with treewidth k, we have shown that in $2^{O(k)}n^{3/2+\varepsilon}$ time, one can compute the entire distance distribution of the input graph. Compared to the $O(kn^2)$ time algorithm for computing APSP [31], our dependence on n is a factor of $O(\sqrt{n})$ less, though our dependence on k is exponential. Our algorithm is a $O(\sqrt{n})$ factor slower than the current best-known $2^{O(k)}n^{1+\varepsilon}$ time algorithm for diameter [1]. For graphs with diameter $O(n^{\varepsilon'})$ for all $\varepsilon' > 0$, including graphs with polylogarithmic diameter, the extra factor becomes $O(n^{\varepsilon})$ for any $\varepsilon > 0$, when compared to the current best-known $2^{O(k)}n$ time algorithm for diameter [21] in this setting. This might be expected, as the distance distribution implies the diameter, and is implied by the APSP, but we find it somewhat surprising that the distance distribution can be computed faster than APSP on graphs with small treewidth.

Our results can be immediately applied to compute any measure of a graph that is a function of the distance distribution. However, they are difficult to adapt to measures that compute properties of individual vertices in the graph, as we exploit properties exclusive to counting pairs that are certain distances apart, without expressly considering which vertices belong to such pairs. In particular, this means that our results are unlikely to directly provide further insight into the efficient computation of related measures, such as the task of computing closeness centrality [5] of every vertex in a given graph.

References

1 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and Fixed Parameter Subquadratic Algorithms for Radius and Diameter in Sparse Graphs. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

2 Karla Atkins, Jiangzhuo Chen, V. S. Anil Kumar, and Achla Marathe. The structure of electrical networks: a graph theory based analysis. *International Journal of Computational Intelligence Systems*, 5(3):265–284, 2009. doi:10.1504/IJCIS.2009.024874.

3 Haris Aziz, Serge Gaspers, Edward J. Lee, and Kamran Najeebullah. Defender Stackelberg Game with Inverse Geodesic Length as Utility Metric. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018)*, pages 694–702. ACM, 2018.

4 Haris Aziz, Serge Gaspers, and Kamran Najeebullah. Weakening Covert Networks by Minimizing Inverse Geodesic Length. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, (IJCAI 2017)*, pages 779–785. IJCAI, 2017. doi:10.24963/ijcai.2017/108.

5 Alex Bavelas. Communication Patterns in Task-Oriented Groups. *The Journal of the Acoustical Society of America*, 22(6):725–730, 1950. doi:10.1121/1.1906679.

6 Jon Louis Bentley. Multidimensional divide-and-conquer. *Communications of the ACM*, 23(4):214–229, 1980. doi:10.1145/358841.358850.

7 Davide Bilò, Feliciano Colella, Luciano Gualà, Stefano Leucci, and Guido Proietti. A faster computation of all the best swap edges of a tree spanner. In *Proceedings of the 22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO 2015)*, volume 9439 of *LNCS*, pages 239–253. Springer, 2015. doi:10.1007/978-3-319-25258-2_17.

8 Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

9 Csaba Böde, István A. Kovács, Máté S. Szalay, Robin Palotai, Tamás Korcsmáros, and Péter Csermely. Network analysis of protein dynamics. *FEBS Letters*, 581(15):2776–2782, 2007. doi:10.1016/j.febslet.2007.05.021.

10 Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. A $c^k n$ 5-Approximation Algorithm for Treewidth. *SIAM Journal on Computing*, 45(2):317–378, 2016. doi:10.1137/130947374.

11 Karl Bringmann, Thore Husfeldt, and Måns Magnusson. Multivariate Analysis of Orthogonal Range Searching and Graph Distances. In *13th International Symposium on Parameterized and Exact Computation (IPEC 2018)*, volume 115 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:13, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.IPEC.2018.4.

12 Sergio Cabello and Christian Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Computational Geometry: Theory and Applications*, 42(9):815–824, 2009. doi:10.1016/j.comgeo.2009.02.001.

13 Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 161–173. SIAM, 2010. doi:10.1137/1.9781611973075.15.

14 Paolo Crucitti, Vito Latora, Massimo Marchiori, and Andrea Rapisarda. Efficiency of scale-free networks: error and attack tolerance. *Physica A: Statistical Mechanics and its Applications*, 320:622–642, 2003. doi:10.1016/S0378-4371(02)01545-5.

15 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

16 David Dagon, Guofei Gu, Christopher P. Lee, and Wenke Lee. A Taxonomy of Botnet Structures. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC 2007)*, pages 325–339. IEEE Computer Society, 2007. doi:10.1109/ACSAC.2007.44.

17 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 1st edition, 2009.

18 Martin Fürer. Faster Integer Multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009. doi:10.1137/070711761.

19 Martin Fürer. How fast can we multiply large integers on an actual computer? In *Proceedings of the 11th Latin American Symposium on Theoretical Informatics (LATIN 2014)*, volume 8392 of *LNCS*, pages 660–670. Springer, 2014. doi:10.1007/978-3-642-54423-1_57.

20 Murad Hossain, Sameer Alam, Tim Rees, and Hussein Abbass. Australian Airport Network Robustness Analysis : A Complex Network Approach. In *Proceedings of the 36th Australasian Transport Research Forum (ATRF 2013)*, pages 1–21, 2013.

21 Thore Husfeldt. Computing Graph Distances Parameterized by Treewidth and Diameter. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*, volume 63 of *LIPIcs*, pages 16:1–16:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPICS.IPEC.2016.16.

22 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/JCSS.2000.1727.

23 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

24 Camille Jordan. Sur Les Assemblages des Lignes. *Journal für die Reine und Angewandte Mathematik*, 70:185–190, 1869.

25 István A. Kovács and Albert-László Barabási. Destruction perfected. *Nature*, 524(7563):38–39, 2015. doi:10.1038/524038a.

26 Leopold Kronecker. *Grundzüge einer arithmetischen Theorie der algebraischen Grössen*. G. Reimer, 1882.

27 Silviu Maniu, Pierre Senellart, and Suraj Jog. An Experimental Study of the Treewidth of Real-World Graph Data. In *22nd International Conference on Database Theory (ICDT 2019)*, volume 127 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:18, 2019. doi:10.4230/LIPIcs.ICDT.2019.12.

28 Louis Monier. Combinatorial solutions of multidimensional divide-and-conquer recurrences. *Journal of Algorithms*, 1(1):60–74, 1980. doi:10.1016/0196-6774(80)90005-X.

29 Flaviano Morone and Hernán A. Makse. Influence maximization in complex networks through optimal percolation. *Nature*, 524(7563):65–68, 2015. doi:10.1038/nature14604.

30 Kamran Najeebullah. *Complexity of Optimally Attacking and Defending a Network*. PhD thesis, UNSW Sydney, 2018.

31 Léon Planken, Mathijs de Weerdt, and Roman van der Krogt. Computing All-Pairs Shortest Paths by Leveraging Low Treewidth. *Journal of Artificial Intelligence Research*, 43:353–388, 2012. doi:10.1613/jair.3509.

32 A. Schönhage and V. Strassen. Schnelle Multiplikation großer Zahlen. *Computing*, 7(3-4):281–292, 1971. doi:10.1007/BF02242355.

33 Ian Michael Shamos. *Computational geometry*. PhD thesis, Yale University, 1978.

34 Piotr L. Szczepański, Tomasz P. Michalak, and Talal Rahwan. Efficient algorithms for game-theoretic betweenness centrality. *Artificial Intelligence*, 231:39–63, 2016. doi:10.1016/j.artint.2015.11.001.

35 Kevin Topley. Computationally Efficient Bounds for the Sum of Catalan Numbers. Technical Report 1601.04223, ArXiv, 2016. arXiv:1601.04223.

36 Alexander Veremyev, Oleg A. Prokopyev, and Eduardo L. Pasiliao. Critical nodes for distance-based connectivity and related problems in graphs. *Networks*, 66(3):170–195, 2015. doi:10.1002/net.21622.

37 Harry Wiener. Structural Determination of Paraffin Boiling Points. *Journal of the American Chemical Society*, 69(1):17–20, 1947. doi:10.1021/ja01193a005.

38 Ryan Williams. A New Algorithm for Optimal Constraint Satisfaction and Its Implications. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, volume 3142 of *LNCS*, pages 1227–1237. Springer, 2004. doi:10.1007/978-3-540-27836-8_101.

39 Bo Zhou, Xiaochun Cai, and Nenad Trinajstić. On Harary index. *Journal of Mathematical Chemistry*, 44(2):611–618, 2008. doi:10.1007/s10910-007-9339-2.

40 Yihai Zhu, Jun Yan, Yan Sun, and Haibo He. Revealing cascading failure vulnerability in power grids using risk-graph. *IEEE Transactions on Parallel and Distributed Systems*, 25(12):3274–3284, 2014. doi:10.1109/TPDS.2013.2295814.