Time-Optimal APSP and Matrix Multiplication in Classes of Linear Neighborhood Complexity

Édouard Bonnet* Julien Duron† Marcin Pilipczuk‡ Marek Sokołowski§
Szymon Toruńczyk¶

August 26, 2026

Abstract

The notion of *linear neighborhood complexity* is a very general structural assumption on a graph class, covering most classes of sparse graphs such as planar graphs, graphs excluding a fixed (topological) minor, or bounded expansion graphs, as well as many structured classes of dense graphs, such as graphs of bounded clique-width, twin-width, merge-width, or flip-width.

In this work, we present $\mathcal{O}(n^2)$-time optimal algorithms for n-vertex graphs coming from a class of linear neighborhood complexity for the following problems:

- All-Pairs Shortest Paths,
- the multiplication of the adjacency matrix M of the input graph with any $n \times n$ matrix. More specifically, after a quadratic preprocessing, we can multiply M with any n-vector in $\mathcal{O}(n)$ time.

This solves several questions raised in [Bonnet, Kim, Geniet, Moon; ICALP '26], and improves and generalizes results in several other recent papers [Bonnet, Giocanti, Ossona de Mendez, Thomassé; STACS '23], [Bannach, Marwitz, Tantau; STACS '24], [Anand, van den Brand, McCarty; NeurIPS '26], [Kozma, Opler '26], and [Cardinal, McCarty, Yuditsky '26]. We also extend our results to classes of bounded VC density.

In classes of linear neighborhood complexity, we also give a triangle-detection algorithm in randomized linear time $\mathcal{O}(n+m)$ in n-vertex m-edge graphs, a K_4-detection algorithm in randomized $\mathcal{O}(n \log^5 n + m \log n)$ or deterministic $\mathcal{O}(n^2)$ time, and a K_5-detection algorithm in randomized $\mathcal{O}(n \log^9 n + m \log^5 n)$ time.

*CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR 5668, 69342 Lyon, France. Homepage: http://perso.ens-lyon.fr/edouard.bonnet. Email: edouard.bonnet@ens-lyon.fr.

†Institute of Informatics, University of Warsaw, Poland. Email: j.duron@uw.edu.pl

‡Institute of Informatics, University of Warsaw, Poland. Homepage: https://www.mimuw.edu.pl/~malcin/. Email: m.pilipczuk@uw.edu.pl.

§Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany. Homepage: https://mnbvmar.github.io. Supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant number 559177164. Email: msokolow@mpi-inf.mpg.de.

¶Institute of Informatics, University of Warsaw, Poland. Homepage: https://www.mimuw.edu.pl/~szymtor/. Email: szymtor@uw.edu.pl.

Contents

1 Introduction

In recent years, we have witnessed tremendous progress in understanding very general structured graph classes and their algorithmic properties.

Classic studies of structured graph classes tackle topologically constrained graphs such as planar graphs, or, more generally, graph classes excluding a minor [40]. This is probably the limit of improved tractability for some algorithmic problems, e.g., network design problems, where topological restrictions on the input graph seem necessary. However, for problems that treat unweighted graphs and interactions between vertices only within a bounded distance, we expect to find efficient algorithms in more general graph classes. Problems of this type include NP-hard problems such as Minimum Dominating Set or Maximum Independent Set (where one can hope for approximation or parameterized algorithms that are believed to be impossible for general graphs) and polynomial-time solvable problems such as multiple shortest path computations (where one can hope to beat known lower bounds that hold in general graphs).

Two decades ago, Nešetřil and Ossona de Mendez [43, 44, 45, 46] initiated the study of *bounded expansion* and *nowhere dense* graph classes. Subsequent research confirmed that the latter are the limit of tractability of first-order model checking (a meta-problem capturing most problems that concern bounded-distance interactions between vertices) among subgraph-closed graph classes [30]. However, the assumption of being subgraph-closed does not allow this theory to capture *dense* structured graph classes. In attempts to understand the limit of tractability of first-order model checking on graphs in full generality, width parameters twin-width [16], flip-width [48], and merge-width [24] were introduced. Boundedness of the last two is conjectured to be equivalent; furthermore, tractability of the first-order model checking problem in hereditary graph classes is conjectured to be delimited by the slightly more general notions of *almost bounded flip-width* or *almost bounded merge-width* [24]. Research on algorithms for classes of bounded twin-width/merge-width/flip-width includes not only parameterized algorithms for first-order model checking [16, 28, 24], but also parameterized algorithms for specific problems [13, 14, 29, 4], approximation algorithms [13, 7, 22], polynomial kernels [15], and polynomial-time algorithms [13, 37, 31, 12, 23].

However, most of the above-mentioned algorithms require a witness of low twin-width or merge-width to be given as part of the input. The existence of polynomial-time or parameterized algorithms providing appropriate witnesses remains a central open question in the area.

First-order formulas are able to speak about any fixed distance between vertices, but some problems are concerned only with direct adjacencies. This makes them potentially efficiently solvable on even larger classes of graphs. A promising concept in this direction is the notion of *linear neighborhood complexity* [27]. We say that a graph class $\mathcal{C}$ has *linear neighborhood complexity* if there exists a constant $c > 0$ such that for every $G \in \mathcal{C}$ and $\emptyset \neq A \subseteq V(G)$ the number of neighborhood traces on A, i.e., $|\{N(v) \cap A \mid v \in V(G)\}|$, is at most $c|A|$. We also say that $\mathcal{C}$ has VC density ρ if there exists a constant $c > 0$ such that for every $G \in \mathcal{C}$ and $\emptyset \neq A \subseteq V(G)$, $|\{N(v) \cap A \mid v \in V(G)\}| \leqslant c|A|^\rho$.

Linear neighborhood complexity not only covers all classes of bounded twin-width, flip-width, and merge-width but also, for example, classes with bounded *weak 2-coloring number*, such as the class of 2-subdivisions of all graphs. An even more general notion—extending graph *degeneracy* to the dense setting—is bounded *symmetric-difference degeneracy (sd-degeneracy)* [11]. For a graph G, it is the least number c such that G can be reduced into a single vertex by iteratively finding two distinct vertices u, v with $|N(v) \triangle N(u)| \leqslant c$ and removing one of them from the graph. (This notion of a decomposition is called an *sd-degeneracy sequence*. See Section 2 for precise definitions.) We refer to Fig. 1 for an overview of the properties of graph classes mentioned here.

The first algorithmic applications of the linear neighborhood complexity assumption stem from

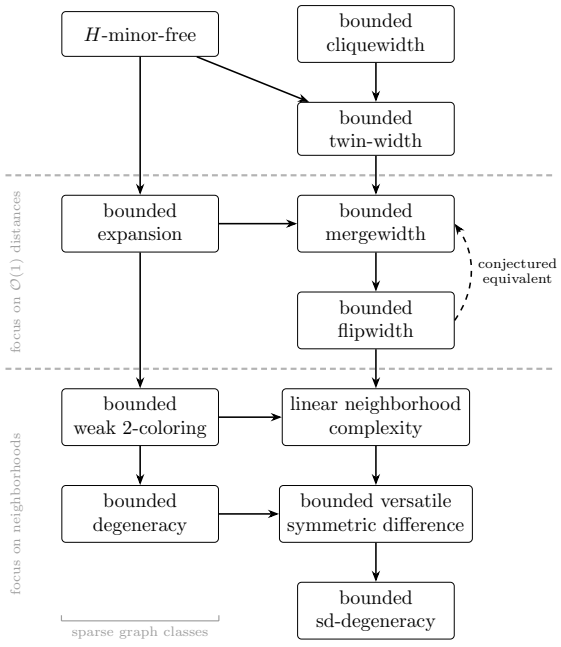

Figure 1: Properties of hereditary graph classes and implications among them.

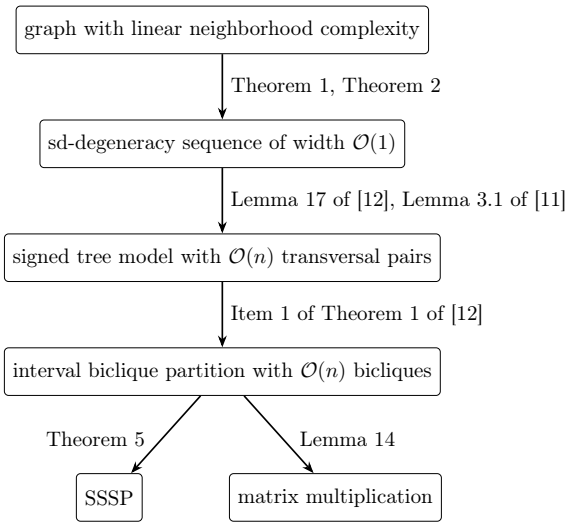

Figure 2: Diagram of the algorithmic pipeline.

the seminal work of Welzl [49], which implies that for a given n-vertex graph from such a class, one can order its vertices in polynomial time so that every neighborhood is a union of $\mathcal{O}(\log n)$ many intervals. The runtime was improved to $\mathcal{O}((m+n)\log n)$ in very recent work with $\mathcal{O}(\log^2 n)$ intervals for each vertex [23] instead of $\mathcal{O}(\log n)$. Prior works [19, 25] achieve $\mathcal{O}(\log^2 n)$ intervals on average per vertex in the same running time. In turn, such a representation can be transformed into an sd-degeneracy sequence of width $\mathcal{O}(\log^2 n)$ [12].

Furthermore, [11] introduced a representation of the edge set of the graph, called a *signed tree model*. A signed tree model of a graph G is a binary tree with leaves labeled with $V(G)$ and additional transversal edges which can be positive or negative, and which fully determine the adjacency in G (see Section 2 for a precise definition). Subsequent work [12] transformed signed tree models into *interval biclique partitions*, first defined in [13]. An interval biclique partition into b bicliques of a graph G is a labeling of $V(G)$ with $\{1, 2, \ldots, n\}$ together with a partition of $E(G)$ into b bicliques whose sides are intervals of $\{1, 2, \ldots, n\}$. By pipelining the results of [12, 11], we can first turn an sd-degeneracy sequence of width d into a signed tree model with $\mathcal{O}(dn)$ transversal pairs in $\mathcal{O}(dn+m)$ time, and then into an interval biclique partition with $\mathcal{O}(dn)$ bicliques in $\mathcal{O}(dn\log n+m)$ time.

While sd-degeneracy sequences, sparse signed tree models, and interval biclique partitions are relatively simple and often efficiently computable, they bypass the need for involved decomposition notions in classes of bounded twin-width/merge-width/flip-width, when the problem at hand lends itself to their algorithmic uses.

1.1 Our results

Our first contributions are two time-optimal algorithms for computing sd-degeneracy sequences of bounded width in classes of linear neighborhood complexity.

Theorem 1. *For every graph class $\mathcal{C}$ of linear neighborhood complexity, there is an $\mathcal{O}_{\mathcal{C}}(n^2)$-time algorithm that, given an n-vertex graph $G \in \mathcal{C}$, outputs an sd-degeneracy sequence of G of width $\mathcal{O}_{\mathcal{C}}(1)$.*

Note that the running time of Theorem 1 is optimal for graphs with $\Theta(n^2)$ edges. This resolves an open problem of [12]. For graphs with a subquadratic number of edges, we provide a linear-time randomized algorithm.

Theorem 2. *For every graph class $\mathcal{C}$ of linear neighborhood complexity, there is a randomized algorithm that, given a graph $G \in \mathcal{C}$, outputs an sd-degeneracy sequence of G of width $\mathcal{O}_{\mathcal{C}}(1)$ in expected time $\mathcal{O}_{\mathcal{C}}\big(|V(G)| + |E(G)|\big)$.*

Previously available algorithms would either proceed by naively constructing an sd-degeneracy sequence of width $\mathcal{O}(1)$, which takes time $\mathcal{O}(n^4)$ using a direct implementation (or $\mathcal{O}(n^3)$ after realizing that $\mathcal{C}$ has bounded versatile symmetric difference, see definition below), or by following Welzl's approach, which currently yields an sd-degeneracy sequence of width $\mathcal{O}(\log^2 n)$ in time $\mathcal{O}((n+m)\log n)$ [23, 12], where $n = |V(G)|$ and $m = |E(G)|$.

To prove Theorem 1, we introduce the notion of *bounded versatile symmetric difference*, where we require that every induced subgraph with n vertices contains $\Theta(n)$ disjoint vertex pairs u, v satisfying $|N(u) \triangle N(v)| \leqslant c$ for some constant c.

On one hand, a simple application of Haussler's packing lemma (cf. Theorem 33) proves that linear neighborhood complexity implies bounded versatile symmetric difference. On the other hand, we show how to use the "versatility" strengthening, together with a known data structure for longest common extension of [39], to compute sd-degeneracy sequences in $\mathcal{O}(n^2)$ time.

We will actually prove generalizations of Theorems 1 and 2 to classes of VC density ρ.

Theorem 3. *For every real number $\rho \geqslant 1$ and every graph class $\mathcal{C}$ of VC density ρ, there is an $\mathcal{O}_{\mathcal{C}}(n^{3-\frac{1}{\rho}})$-time algorithm that, given an n-vertex graph $G \in \mathcal{C}$, outputs an sd-degeneracy sequence of G of width $\mathcal{O}_{\mathcal{C}}(n^{1-\frac{1}{\rho}})$.*

Theorem 4. *For every real number $\rho \geqslant 1$ and every graph class $\mathcal{C}$ of VC density ρ, there is a randomized algorithm that, given a graph $G \in \mathcal{C}$, outputs an sd-degeneracy sequence of G of width $\mathcal{O}_{\mathcal{C}}(n^{1-\frac{1}{\rho}})$ in expected time $\mathcal{O}_{\mathcal{C}}\big(|V(G)| + |E(G)|\big)$.*

With the aforementioned results of [12, 11], Theorem 1 allows us to compute in $\mathcal{O}(n^2)$ time a signed tree model with $\mathcal{O}(n)$ transversal pairs and an interval biclique partition with $\mathcal{O}(n)$ bicliques. Our second contribution is an efficient algorithm using the latter for shortest-path computation.

Theorem 5. *There is an $\mathcal{O}(n+b)$-time algorithm that, given an interval biclique partition of an n-vertex graph G with b bicliques and $s \in V(G)$, outputs a shortest-path tree from vertex s in G.*

Note that the running time of Theorem 5 is independent of m, the number of edges of G, and in fact is usually sublinear in m. Theorem 5 bypasses the use of the DAG-compression technique [5], which is a way to solve shortest-path problems from interval biclique partitions that incurs, in the current state of knowledge, extra superconstant factors.

By applying Theorem 5 to every vertex of the graph and pipelining it with Theorem 1 and the aforementioned results of [12, 11], we obtain a time-optimal algorithm for All-Pairs Shortest Path (APSP for short) in classes of linear neighborhood complexity.

Theorem 6. *For every graph class $\mathcal{C}$ of linear neighborhood complexity,* All-Pairs Shortest Path *can be solved in $\mathcal{O}_{\mathcal{C}}(n^2)$ time on n-vertex graphs of $\mathcal{C}$.*

Previous work on APSP includes an $\mathcal{O}(n^2 \log^2 n)$-time algorithm in classes of bounded twin-width, and in time $\mathcal{O}(n^2 \log n)$ on graphs of bounded radius-1 merge-width given with a witness [12]. More recently, the $\mathcal{O}(n^2 \log^2 n)$-time APSP algorithm was extended to classes of linear neighborhood complexity [18]. In [13], an $\mathcal{O}(n^2 \log n)$-time APSP algorithm was given on graphs of bounded twin-width given with their contraction sequence, which was then improved to $\mathcal{O}(n^2)$-time in the paper introducing DAG-compressions [5].

Note that Theorems 2 and 5, together with [12, 11], also give a randomized $\mathcal{O}(kn + m)$-time algorithm for Multi-Source Shortest Path from k sources on n-vertex m-edge graphs. Another consequence of Theorem 1 and results in [12] (see Lemma 14) is:

Theorem 7. *For every graph class $\mathcal{C}$ of linear neighborhood complexity, there is an $\mathcal{O}_{\mathcal{C}}(n^2)$-time algorithm that, given any adjacency matrix M of an n-vertex graph of $\mathcal{C}$, yields a data structure that computes the matrix-vector product MX in time $\mathcal{O}_{\mathcal{C}}(n)$ for any n-vector X.*

In particular, the product MN can be computed in $\mathcal{O}_{\mathcal{C}}(n^2)$ time for any *$n \times n$ matrix N.*

We refer to Fig. 2 for a diagram of the algorithmic pipeline discussed in the previous few paragraphs.

Theorem 6 answers [12, Question 2] positively, far beyond classes of bounded twin-width (for which the question was asked). Theorems 6 and 7 improve APSP and matrix-multiplication algorithms in [14, 2, 12, 35, 18], all of which have extra polylog factors, and extend results in [36, 5], which work on graphs of bounded clique-width and on graphs of bounded twin-width given with contraction sequences, respectively.

By using Theorem 3 rather than Theorem 1, one can solve All-Pairs Shortest Path in graphs of VC density ρ (or multiply the adjacency matrix of graphs of VC density ρ with any matrix) in time $\mathcal{O}(n^{3-1/\rho})$. However, we observe that fast matrix multiplication and Seidel's algorithm [47] have a better running time in general instances as soon as $\rho > 1.591$.

We also obtain a time-optimal triangle-detection algorithm in classes of linear neighborhood complexity, which leverages an sd-degeneracy sequence of constant width.

Theorem 8. *For every graph class $\mathcal{C}$ of linear neighborhood complexity,* Triangle Detection *can be solved in expected time $\mathcal{O}_{\mathcal{C}}(n + m)$ on n-vertex m-edge graphs of $\mathcal{C}$.*

Let us recall that on general graphs, the fastest known Triangle Detection algorithm runs in time $\mathcal{O}(\min(n^{\omega}, m^{2\omega/(\omega+1)}))$ [1] where ω is the exponent of square matrix multiplication, which currently implies $\mathcal{O}(\min(n^{2.372}, m^{1.407}))$ time.

Utilizing both an sd-degeneracy sequence and a Welzl order of the input graph, we obtain randomized almost-linear algorithms for finding 4-vertex and 5-vertex cliques, and a deterministic $\mathcal{O}(n^2)$-time algorithm for K_4 detection (time-optimal within dense graphs).

Theorem 9. *For every graph class $\mathcal{C}$ of linear neighborhood complexity, a K_4 subgraph can be found in expected $\mathcal{O}_{\mathcal{C}}(n \log^5 n + m \log n)$ or in deterministic $\mathcal{O}_{\mathcal{C}}(n^2)$ time on n-vertex m-edge graphs of $\mathcal{C}$.*

Theorem 10. *For every graph class $\mathcal{C}$ of linear neighborhood complexity, a K_5 subgraph can be found in expected time $\mathcal{O}_{\mathcal{C}}(n \log^9 n + m \log^5 n)$ on n-vertex m-edge graphs of $\mathcal{C}$.*

Prior to our work, no linear-time algorithm was known for Triangle Detection in the much more restricted classes of bounded clique-width, nor was an almost-linear algorithm known in classes of bounded twin-width (where, for both, the input is a mere graph without a witness of low width).

2 Preliminaries

We index the word positions from 1. In particular, $w = w[1]w[2]\cdots w[s]$ for every word w of length s. We let $\log x$ denote the base-2 logarithm of x. For integers $\ell \leqslant r$, let $[\ell, r]$ be the integer interval spanning all values from ℓ to r, inclusively, and for integer n, let $[n] = [1, n]$. Graphs are simple (without loops or parallel edges), undirected, and finite. For a graph G, we denote its vertex- and edge-set as $V(G)$ and $E(G)$, respectively.

2.1 Neighborhood complexity

For a graph G, let $\pi_G(m)$ be the maximum number of distinct neighborhoods of vertices in G restricted to a set of at most m vertices, i.e.,

$$\pi_G(m) := \max_{X \subseteq V(G), |X| \leqslant m} |\{N_G(v) \cap X : v \in V(G)\}|.$$

We say that G has *c-linear neighborhood complexity* (or c-LNC) if $\pi_G(m) \leqslant c \cdot m$ for all positive $m \in \mathbb{N}$. Class $\mathcal{C}$ has *linear neighborhood complexity* if there exists a constant c such that all graphs $G \in \mathcal{C}$ have c-LNC. We say that $\mathcal{C}$ has *VC density* ρ if there exists a constant c such that for all $G \in \mathcal{C}$, $\pi_G(m) \leqslant cm^\rho$ for all positive m. Finally, we say that G has *(c, ρ)-polynomial neighborhood complexity* (or (c, ρ)-PNC) if $\pi_G(m) \leqslant c \cdot m^\rho$ for all positive $m \in \mathbb{N}$.

2.2 Symmetric difference and sd-degeneracy

In this paper, we define the *symmetric difference* of two vertices u, v in a graph G as

$$\mathrm{sd}_G(u, v) := |N_G(u) \triangle N_G(v)|,$$

where $\triangle$ denotes the symmetric difference of two sets: $X \triangle Y := (X \setminus Y) \cup (Y \setminus X)$. There are slight variations in defining the symmetric difference of u, v (depending on whether or not u, v themselves count), but they only differ by a constant additive term. We say that two distinct vertices u, v are *d-near-twins* in G if $\mathrm{sd}_G(u, v) \leqslant d$, and are *twins* if $\mathrm{sd}_G(u, v) = 0$.

The *symmetric difference* $\mathrm{sd}(G)$ of a graph G is the least nonnegative integer d such that for every induced subgraph H of G with at least two vertices, there are two d-near-twins in H.

An *sd-degeneracy sequence* of an n-vertex graph G is a list $(u_1, v_1), \ldots, (u_{n-1}, v_{n-1})$ of pairs of vertices of G such that

- $u_i \neq v_i$ for every $i \in [n-1]$,
- $V(G) = \{u_1, u_2, \ldots, u_{n-2}, u_{n-1}, v_{n-1}\}$, and
- there is no $i < j$ such that $u_i \in \{u_j, v_j\}$.

Informally, it is an elimination sequence where only the first element of the pair is removed at each step. The *width* of the sd-degeneracy sequence $(u_1, v_1), \ldots, (u_{n-1}, v_{n-1})$ is defined as

$$\max_{i \in [n-1]} \mathrm{sd}_{G - \{u_1, u_2, \ldots, u_{i-1}\}}(u_i, v_i).$$

The *sd-degeneracy* of G (where 'sd' stands for symmetric difference) is the least integer d such that G admits an sd-degeneracy sequence of width d.

Equivalently, the class of graphs of sd-degeneracy at most d can be defined by induction on the number of vertices: a graph G has sd-degeneracy at most d if either $|V(G)| \leqslant 1$, or there is a pair of distinct vertices $u, v \in V(G)$ such that $\mathrm{sd}_G(u, v) \leqslant d$ and the graph $G - u$ has sd-degeneracy at most d.

We say that a hereditary graph class $\mathcal{C}$ has *bounded versatile symmetric difference* if there is an integer $d := d_{\mathcal{C}} > 0$ such that for every n-vertex graph $G \in \mathcal{C}$, there are at least $\lfloor n/d \rfloor$ disjoint pairs of vertices u, v satisfying $\mathrm{sd}_G(u, v) \leqslant d$. For $\alpha \in [0, 1)$, we say that $\mathcal{C}$ has *versatile symmetric difference of exponent* α if there is a constant $d := d_{\mathcal{C}} > 0$ such that for every n-vertex graph $G \in \mathcal{C}$, there are $\lfloor n/d \rfloor$ disjoint pairs of vertices u, v in G satisfying $\mathrm{sd}_G(u, v) \leqslant dn^{\alpha}$. Thus *versatile symmetric difference of exponent 0* coincides with *bounded versatile symmetric difference*.

2.3 Signed tree models and interval biclique partitions

We recall the definition of signed tree models for the sake of self-containedness. However, we will not use them in the rest of the paper. Thus, from this subsection, one can only read the definition of interval biclique partitions (given just after Lemma 11) and Lemma 13.

Signed tree models, originally defined in [11], generalize the tree models introduced in the context of twin-width [17, 13]. In a rooted tree T and $u, v \in V(T)$, by $u \preccurlyeq_T v$ we denote that u is an ancestor of v. A pair of vertices of a rooted tree T is a *transversal pair* of T if it is not between an ancestor and descendant. We say that two transversal pairs u_1v_1 and u_2v_2 *cross* if one of u_1, v_1 is a strict ancestor of one of u_2, v_2, and conversely, one of u_2, v_2 is a strict ancestor of one of u_1, v_1.

A *full* binary tree is a rooted binary tree where every internal node has exactly two children. A *signed tree model* is a triple $\mathcal{T} = (T, A, B)$ where T is a full binary tree, A and B are two disjoint sets of transversal pairs of T, and $A \cup B$ does not contain any crossing pair. The transversal pairs of A and B are called *negative edges* and *positive edges*, respectively.

We say that a transversal pair $u'v'$ *covers* the pair $u, v \in V(T)$ if $u' \preccurlyeq_T u$ and $v' \preccurlyeq_T v$ and there is no transversal pair $u''v'' \neq u'v'$ in $A \cup B$ with $u' \preccurlyeq_T u'' \preccurlyeq_T u$ and $v' \preccurlyeq_T v'' \preccurlyeq_T v$. We denote by $L(T)$ the set of leaves of T. The graph $G_{\mathcal{T}}$ represented by the signed tree model $\mathcal{T}$ has vertex set $L(T)$, and $u, v \in L(T)$ are adjacent in $G_{\mathcal{T}}$ if and only if the pair u, v is covered by a positive transversal pair in $\mathcal{T}$.

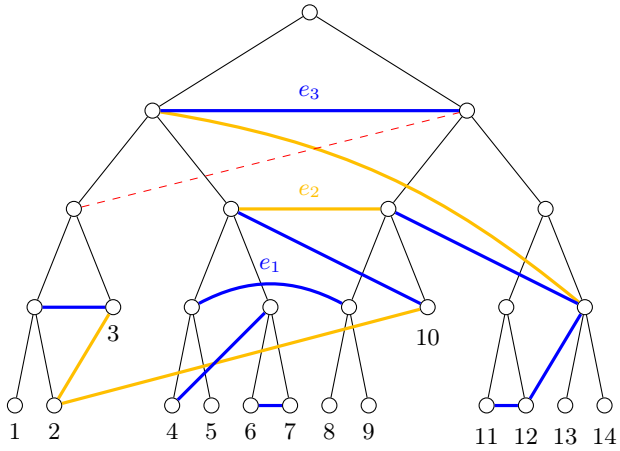

Figure 3: A signed tree model of a 14-vertex graph, with A in amber and B in blue. The topmost amber edge and the dashed red edge cross (which is not in $A \cup B$). Vertex 8 is adjacent to 4 because 4,8 is covered by the positive edge e_1, and to 2 because 2,8 is covered by the positive edge e_3, but 8 is not adjacent to 7 because 7,8 is covered by the negative edge e_2.

A graph class $\mathcal{C}$ has *sparse signed tree models* if there is a constant c such that every $G \in \mathcal{C}$ admits a signed tree model (T, A, B) with $|A \cup B| \leqslant c \cdot |V(T)|$. There is an efficient algorithm that turns sd-degeneracy sequences of bounded width into sparse signed tree models.

Lemma 11 (Lemma 18 of [12], Lemma 3.1 of [11]). *There is an $\mathcal{O}(dn+m)$-time algorithm that, given an sd-degeneracy sequence of an n-vertex m-edge graph G of width d, outputs a signed tree model of G with $\mathcal{O}(dn)$ transversal pairs.*

Assume that the graph G comes with a total ordering of its vertices; for convenience, assume that the vertex set of G is $[n]$, ordered naturally. We then define the *interval biclique partition* of G as the partitioning of the edge set of G into bicliques, where each side of each biclique is an integer subinterval of $[n]$. The following lemma implies that a small-size signed tree model can be efficiently turned into a small-size interval biclique partition.

Lemma 12 (Item 1 of Theorem 1 of [12]). *There is an $\mathcal{O}(p \log n)$-time algorithm that, given a signed tree model of an n-vertex graph G with p transversal pairs, outputs an interval biclique partition of G with $\mathcal{O}(p)$ bicliques.*

We will rely on the following consequence of Lemmas 11 and 12.

Lemma 13. *There is an $\mathcal{O}(dn \log n + m)$-time algorithm that, given an sd-degeneracy sequence of an n-vertex m-edge graph G of width d, outputs an interval biclique partition of G with $\mathcal{O}(dn)$ bicliques.*

2.4 Matrix multiplication

An interval biclique partition with $\mathcal{O}(n)$ bicliques of an n-vertex graph G allows one to multiply any adjacency matrix of G with any n-vector in $\mathcal{O}(n)$ time. More precisely:

Lemma 14 (Theorem 33 in [12]). *For every abelian group $\mathcal{A}$, there exists an algorithm that takes on input an interval biclique partition with b bicliques of an n-vertex graph G with adjacency matrix M and an n-vector $X \in \mathcal{A}^n$ and computes the product MX, in time $\mathcal{O}(n+b)$ and $\mathcal{O}(n+b)$ group operations.*

Note that, as the adjacency matrix M is a $\{0,1\}$-matrix, the multiplication of M with a vector of group elements is well-defined. This lemma has a short proof that encodes M into an $n \times n$ $\{-1,0,1\}$-matrix with $\mathcal{O}(b)$ nonzero entries. As a direct consequence of Lemma 14, one can multiply M with any $n \times n$ matrix in time $\mathcal{O}(n(n+b))$ (assuming the group operations take constant time).

3 Finding an sd-degeneracy sequence deterministically

In this section, we provide an algorithm that computes an sd-degeneracy sequence of constant width in graphs of bounded versatile symmetric difference. The algorithm is time-optimal for dense graphs (i.e., n-vertex graphs with $\Theta(n^2)$ edges). More generally, the algorithm computes an sd-degeneracy sequence of width $\mathcal{O}(n^\alpha)$ in graphs of versatile symmetric difference of exponent α.

It uses classic data structures for strings. Given a word w of finite length s, a longest common extension (LCE) query $i, j \in [s]$ asks for the largest integer k such that $w[i]w[i+1]\cdots w[i+k-1] = w[j]w[j+1]\cdots w[j+k-1]$.

Theorem 15 (Section 4 of [39]). *Fix a finite alphabet Σ. There is an $\mathcal{O}(s)$-time algorithm, that given a word w of length s over Σ, builds a data structure that answers each LCE query on w in constant time.*

Theorem 15 is originally shown by reduction to least common ancestor (LCA) in the suffix tree. To avoid computing the suffix tree, one can alternatively go through suffix array, longest common prefix (LCP) array, and range minimum query (RMQ). See [6] for a simple proof of optimal LCA and RMQ data structures.

Theorem 15 yields a surprisingly fast check for d-near-twinness in graphs.

Theorem 16. *There is an $\mathcal{O}(n^2)$-time algorithm that, given an n-vertex graph G, builds a data structure that given a query $u, v \in V(G), d \in \mathbb{N}$ reports if $sd_G(u, v) \leqslant d$ in $\mathcal{O}(d)$ time.*

Proof. We build a binary word w of length n^2 by concatenating the adjacency vectors of all vertices. Formally, assuming $V(G) = [n]$, for every $a, b \in [n]$, $w[(a-1)n + b] = 1$ if $ab \in E(G)$ and $w[(a-1)n + b] = 0$ otherwise. The word w can easily be built in $\mathcal{O}(n^2)$ time.

By Theorem 15, we build, in $\mathcal{O}(n^2)$ time, a data structure $\mathcal{D}_w$ that answers each LCE query on w in constant time. Given a query $u, v \in V(G), d \in \mathbb{N}$, we proceed as follows. We initialize $p \leftarrow 1$ and $q \leftarrow d$.

While $p \leqslant n$ and $q \geqslant 0$, we feed $\mathcal{D}_w$ with the query $(u-1)n + p, (v-1)n + p$. (Note that, for $p = 1$, those are the positions where the adjacency vectors of u and v, respectively, start.) Let k be the query answer, found in constant time. If $p + k \leqslant n$, we decrement q by 1 and set $p \leftarrow p + k + 1$, else we exit the while loop.

When we exit the while loop, if $q \geqslant 0$, we report that $\mathrm{sd}_G(u, v) \leqslant d$, and otherwise ($q = -1$) that $\mathrm{sd}_G(u, v) > d$.

Time. We have already checked that $\mathcal{D}_w$ is built in $\mathcal{O}(n^2)$ time. Note that each query u, v, d makes at most $d + 1$ LCE queries. Indeed, in the end of the while loop, we decrement q (initially set to d) by 1 or we exit the loop, whereas continuing looping requires that q is nonnegative. Hence a query is met in time $(d + 1)\mathcal{O}(1) + \mathcal{O}(1) = \mathcal{O}(d)$.

Correctness. Let $1, p_1, \ldots, p_h$ be the successive values p is set to. In particular, $h \leqslant d$. The following properties hold:

- $1 < p_1 < \ldots < p_h \leqslant n + 1$;
- for every $i \in [h]$, $w[(u-1)n + p_i - 1] \neq w[(v-1)n + p_i - 1]$;
- for every $j \in [p_h - 1]$, if $w[(u-1)n + j] \neq w[(v-1)n + j]$ then there exists $i \in [h]$ such that $j = p_i - 1$.

The main invariant is that there are exactly i vertices with label smaller than p_i adjacent to exactly one of u, v. As we check both if more than d such positions where found ($q \leqslant 0$) and if the current position overflows to the next adjacency vectors ($p \leqslant n$), we correctly decide whether or not $\mathrm{sd}_G(u, v) \leqslant d$ holds when exiting the while loop. □

In particular, we can list *all* pairs of d-near-twins in quadratic time, when d is constant.

Corollary 17. *There is an $\mathcal{O}(dn^2)$-time algorithm that, given an n-vertex graph and a positive integer d, lists all the pairs of vertices $u, v \in V(G)$ such that $sd_G(u, v) \leqslant d$.*

Proof. We compute in $\mathcal{O}(n^2)$ time the data structure $\mathcal{D}$ of Theorem 16. For every $u, v \in V(G)$ with $u \neq v$, we present $\mathcal{D}$ with the query u, v, d. If the answer is positive ($\mathrm{sd}_G(u, v) \leqslant d$), we add u, v to the output pairs. Overall, this takes $\mathcal{O}(dn^2)$ time. □

We obtain the objective of this section by repeatedly appling Corollary 17.

Theorem 18. *Let $\mathcal{C}$ be a (hereditary) class of versatile symmetric difference of exponent α. There is an $\mathcal{O}_{\mathcal{C}}(n^{2+\alpha})$-time algorithm that, given an n-vertex graph of $\mathcal{C}$, outputs an sd-degeneracy sequence of G of width $\mathcal{O}_{\mathcal{C}}(n^{\alpha})$.*

Proof. Let $d := d_{\mathcal{C}} > 0$ be such that for every n-vertex graph $G \in \mathcal{C}$, there are $\lfloor n/d \rfloor$ disjoint pairs of vertices u, v satisfying $\mathrm{sd}_G(u, v) \leqslant dn^{\alpha}$. We initialize an sd-degeneracy sequence $\mathcal{S}$ of G of width dn^{α} to the empty list. We also set $G_1 := G$ and $n_1 := n$.

While the remaining graph G_i has $n_i \geqslant d$ vertices, we do the following.

By Corollary 17, we list all the pairs of dn^{α}-near-twins of G_i in $\mathcal{O}(dn^{\alpha}n_i^2)$ time. We then greedily build a list $\{u_1, v_1\}, \ldots, \{u_h, v_h\}$ of $\max(1, \lfloor \frac{1}{2} \lfloor n_i/d \rfloor \rfloor)$ *disjoint* pairs of dn^{α}-near-twins of G_i in $\mathcal{O}(n_i)$ time. (Indeed, if one fixes a set D disjoint pairs, then the choice of any other pair consumes at most two pairs of D.) We append this list to $\mathcal{S}$. We then set $G_{i+1} := G_i - \{u_1, \ldots, u_h\}$ and $n_{i+1} := |V(G_{i+1})|$. This finishes the body of the while loop.

When we exit the while loop, we finish the sd-degeneracy sequence arbitrarily.

Correctness. If the algorithm terminates, it does output an sd-degeneracy sequence of G of width dn^{α} since we only append to $\mathcal{S}$ pairs of dn^{α}-near-twins in the current graph, and remove their first elements in batches of disjoint pairs.

Time. At every loop iteration, we add at least one pair to the sequence and remove their first elements from the graph. Thus, the algorithm does terminate, and goes through the successive induced subgraphs of G: $G_1, G_2, \ldots, G_p$ with $p \leqslant n$. Let $c := 1 - \frac{1}{4d}$.

Claim 19. *For every $i \in [p-1]$, $n_{i+1} \leqslant cn_i$.*

Proof. Indeed, if $n_i < 4d$, then $n_{i+1} \leqslant cn_i$ since $n_{i+1} \leqslant n_i - 1$ (we remove at least one vertex of G_i to form G_{i+1}). If instead, $n_i \geqslant 4d$, we conclude because $\max(1, \lfloor \frac{1}{2} \lfloor n_i/d \rfloor \rfloor) = \lfloor \frac{1}{2} \lfloor n_i/d \rfloor \rfloor \geqslant \frac{1}{4d} \cdot n_i$. □

By Claim 19, for every $i \in [p]$, $n_i \leqslant c^{i-1}n$. So the overall running time is

$$\sum_{i \in [p]} \mathcal{O}(dn^{\alpha}n_i^2) \leqslant \mathcal{O}(dn^{\alpha}) \cdot n^2 \sum_{i \in [p]} c^{2i-2} \leqslant \frac{1}{1-c} \cdot \mathcal{O}_{\mathcal{C}}(n^{2+\alpha}) = \mathcal{O}_{\mathcal{C}}(n^{2+\alpha}),$$

and we conclude. □

Theorem 3 is then a consequence of Theorems 18 and 33. For classes of bounded versatile symmetric difference (exponent 0), we get the following corollary (which, by Theorem 33, implies Theorem 1).

Corollary 20. *Let $\mathcal{C}$ be a (hereditary) class of bounded versatile symmetric difference. There is an $\mathcal{O}_{\mathcal{C}}(n^2)$-time algorithm that, given an n-vertex graph of $\mathcal{C}$, outputs an sd-degeneracy sequence of G of width $\mathcal{O}_{\mathcal{C}}(1)$.*

4 SSSP in graphs given with an interval biclique partition

In this section we prove Theorem 5, which for convenience we recall below.

Theorem 5. *There is an $\mathcal{O}(n + b)$-time algorithm that, given an interval biclique partition of an n-vertex graph G with b bicliques and $s \in V(G)$, outputs a shortest-path tree from vertex s in G.*

We begin by introducing a data structure problem for reporting interval intersections, called Interval-Intersect, and we show how to solve this problem in total linear time. In Interval-Intersect, we are given as input a unary-encoded integer $N \geqslant 1$ (the universe size) and a finite set A of labels, where every label $a \in A$ is assigned an integer interval $I_a \subseteq [N]$. Initially, all labels are active. The data structure is required to answer the following online queries: given an integer interval $J \subseteq [N]$, report all active labels $a \in A$ with the corresponding intervals I_a intersecting J, and deactivate all reported labels.

Subsequently, we will use the resulting data structure to show the linear-time algorithm for Theorem 5. In Section 4.1, we prove the following.

Lemma 21. *There exists a data structure for* Interval-Intersect *in the word RAM model of computation that processes an instance with universe size N, set of labels A, and q queries in $\mathcal{O}(N + |A| + q)$ time.*

The purpose of encoding N in unary is to ensure that every element of the universe $[N]$ fits within a single RAM word; hence we can assume $N < 2^b$ where b is the size of the machine word.

We show now to use Lemma 21 to solve SSSP in linear time in graphs with a given interval biclique partition, namely, we prove Theorem 5.

Proof of Theorem 5. Recall that $[n]$ is the vertex set of the graph, and let the ith biclique in the edge partition ($i \in [b]$) have sides A_i and B_i; each A_i and B_i is a subinterval of $[n]$.

In the beginning, we set up two instances of the linear-time data structure for Interval-Intersect, both with universe $[n]$:

- D_{gather}, with labels from $[n]$, where the label $i \in [n]$ is assigned the interval $[i, i]$;
- D_{scatter}, with labels $\bigcup_{i \in [b]} \{a_i, b_i\}$, with the interval A_i assigned to a_i and the interval B_i assigned to b_i.

The active items in D_{gather} are the vertices undiscovered by the run of the breadth-first search (BFS); hence, we initially query D_{gather} with $[s, s]$, where s is the source of the BFS, to deactivate the label s. On the other hand, the active items in D_{scatter} represent the sides of the bicliques that do not contain vertices visited by BFS.

We perform the usual BFS in G, with the usual queue holding the vertices discovered by the search (initially only s) that are yet to be visited. When a vertex $v \in [n]$ is visited by the search, we find the yet undiscovered neighbors of v as follows. We first use D_{scatter} to enumerate the not yet processed sides of the bicliques by querying D_{scatter} for the interval $[v, v]$. Suppose D_{scatter} returns a_i (resp., b_i) as one of the labels; then, we have that $v \in A_i$ (resp., $v \in B_i$). In this case, all vertices in B_i (resp., A_i) should become discovered if not yet discovered. In order to ensure that, we query D_{gather} with the interval B_i (resp., A_i), and enqueue all vertices returned as a result of the query, marking v as the parent of each of them. We repeat this process for each label reported by D_{scatter}.

The correctness of the algorithm is clear as it is simply a breadth-first search with a bespoke method for listing the yet undiscovered neighbors. To see the efficiency, observe that we issue at most n queries to D_{scatter}, and it reports at most $\mathcal{O}(b)$ labels throughout the entire run of SSSP. Thus, at most $\mathcal{O}(b)$ queries to D_{gather} are issued. Hence both data structures for Interval-Intersect process all respective queries in total $\mathcal{O}(n+b)$ time. Moreover, it is clear that outside of the queries, the SSSP implementation above takes time linear in the size of the input; the claim follows. □

4.1 Interval intersections data structure

It remains to prove Lemma 21.

Our approach resembles the *tabulation technique* and the *method of Four Russians*, commonly used in algorithm design in the word RAM model of computation; see e.g. [3, 26, 8, 6]. Namely, we first show a data structure with a worse complexity guarantee $\mathcal{O}(N \log N + |A| + q)$ (Lemma 22). Using Lemma 22, we then show how to bootstrap a linear-time data structure for universes of size $n \leqslant N$, given access to a linear-time data structure for universes of size $\log n$ (Lemma 23). Next, in the word RAM model, we give a linear-time data structure for universes of size $\sqrt{\log N}$, assuming $\mathcal{O}(N)$-time prior preprocessing (Lemma 24). Applying Lemma 23 twice to Lemma 24 implies Lemma 21. We implement this strategy below.

Lemma 22 (Quasi-linear interval intersections)**.** *There exists a data structure for* Interval-Intersect *that processes an instance with universe size N, set of labels A, and q queries in $\mathcal{O}(N \log N + |A| + q)$ time.*

Proof. On initialization, we build an auxiliary directed graph H with $\mathcal{O}(N \log N + |A|)$ vertices and arcs, as follows. Let $L = \lfloor \log N \rfloor$. For integer $k \in [0, L]$, let the *kth interval layer* $\mathcal{I}_k$ comprise all integer intervals $[i, i + 2^k - 1] \subseteq [N]$. For every such k and every interval $I \in \mathcal{I}_k$, we introduce two vertices: an outvertex u_I and an invertex v_I. For every $a \in A$, we also introduce a vertex a.

We construct arcs in H as follows. Every $I \in \mathcal{I}_k$ with $k \geqslant 1$ is a disjoint union of two integer intervals $I_1, I_2 \in \mathcal{I}_{k-1}$; add arcs $u_I \to u_{I_1}$, $u_I \to u_{I_2}$ (so that arcs between outvertices are directed toward lower-indexed layers) and arcs $v_{I_1} \to v_I$, $v_{I_2} \to v_I$ (so arcs between invertices are oriented toward higher-indexed layers). Also, for every $I = [i, i] \in \mathcal{I}_0$, add an arc $u_I \to v_I$. It is easy to verify that now, for every $I \in \mathcal{I}_k$ and $x \in [N]$, there is a path from u_I to $u_{[x,x]}$ if and only if $x \in I$; and by the same token, there is a path from $v_{[x,x]}$ to v_I if and only if $x \in I$. It follows that for an outvertex u_J and an invertex v_I, a path from u_J to v_I exists if and only if the intervals J and I intersect. Finally, for every $a \in A$ with the corresponding interval I_a, write I_a as a (not necessarily disjoint) union of two intervals $I_1, I_2 \in \mathcal{I}_k$ for some $k \in [0, L]$ (where possibly $I_1 = I_2$). Then add arcs $v_{I_1} \to a$, $v_{I_2} \to a$. This guarantees that for any outvertex u_J, a path from u_J to a in H exists if and only if J and I_a intersect. See Fig. 4.

Apart from the graph H, we also initialize a visited bit vis_v for each $v \in V(H)$, initially unset. Then consider a query J. As above, write J as a union of two intervals $J_1, J_2 \in \mathcal{I}_k$ for some $k \in [0, L]$. Then perform two graph searches in H from the outvertices u_{J_1}, u_{J_2}, visiting only previously unvisited vertices of H, and marking these vertices as visited. For each newly visited vertex $a \in A$, we report a as the label corresponding to the interval I_a intersecting J.

The correctness of the data structure is clear. For the efficiency of the data structure, observe that H has size $\mathcal{O}(N \log N + |A|)$ and can be constructed in time proportional to its size. Then, each of the q queries spawns at most two graph searches in H that only visit previously unvisited vertices. Therefore, all queries are processed in total time $\mathcal{O}(N \log N + |A| + q)$, as required. □

We then show how to use Lemma 22 to bootstrap a data structure for tiny universes to handle an exponentially larger universe. In the next lemma, b is the size of the machine word.

Lemma 23 (Bootstrapping for interval intersections)**.** *Let $N_{\max} < 2^b$ be an integer. Suppose there exists a data structure for* Interval-Intersect *that processes instances with universe of size $N \leqslant \log N_{\max}$, set of labels A, and q queries in $\mathcal{O}(N + |A| + q)$ time. Then there exists a data structure for* Interval-Intersect *that processes instances with universe of size $N \leqslant N_{\max}$, the set of labels A, and q queries in $\mathcal{O}(N + |A| + q)$ time.*

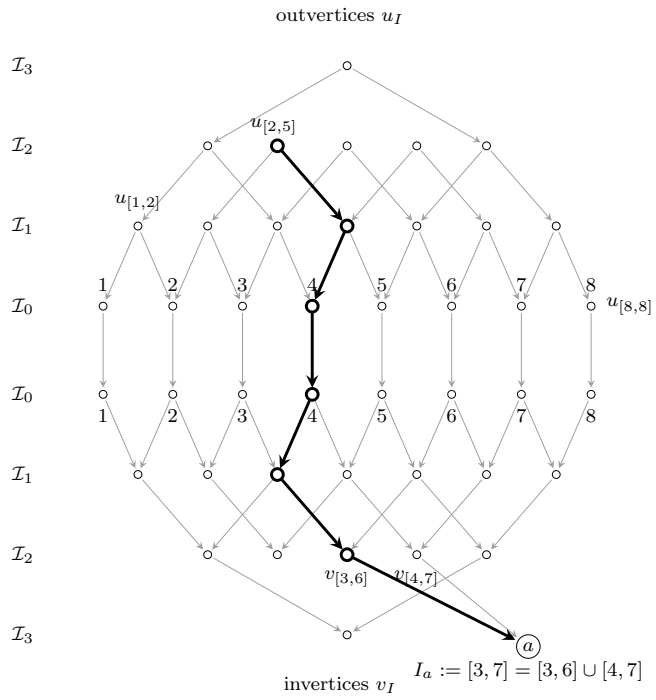

Figure 4: The graph H with $N = 8$ and a single interval $I_a = [3, 7]$. There is a directed path from $u_{[2,5]}$ to a. Note that, on the contrary, one cannot reach a from $u_{[1,2]}$ or from $u_{[8,8]}$.

Proof. We construct the promised data structure as follows. Let $\ell = \max(1, \lfloor \log N \rfloor)$ and partition $[N]$ into $m = \lceil \frac{N}{\ell} \rceil$ blocks $B_1, \ldots, B_m$ of length ℓ; the last block is shorter if needed. For an interval $I \subseteq [N]$, define:

- $\mathsf{inter}(I) \subseteq [m]$ to be the (nonempty) interval of indices of blocks that intersect I;
- $\mathsf{core}(I) \subseteq [m]$ to be the (possibly empty) interval of indices of blocks that are fully contained within I;
- $\mathsf{fringe}(I)$ to be the collection (of size at most 2) of nonempty intervals $I \cap B_i$ over all blocks $i \in \mathsf{inter}(I) \setminus \mathsf{core}(I)$.

See Fig. 5 for an illustration. All of the above can be computed from I in constant time. Also, whenever I is a subinterval of B_i, let $\mathsf{rel}_i(I) \subseteq [\ell]$ be the position of the interval I relative to B_i, i.e., the interval I with both endpoints shifted by $-(i-1)\ell$.

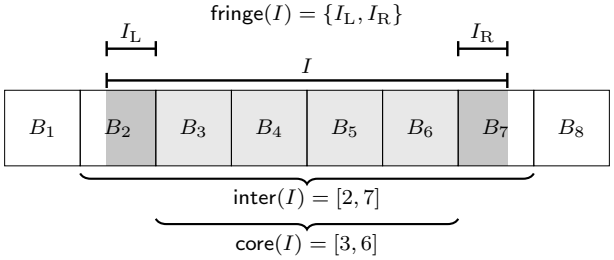

Figure 5: The objects $\mathsf{inter}(I)$, $\mathsf{core}(I)$, and $\mathsf{fringe}(I)$ for $m = 8$ and some interval I.

We instantiate three types of auxiliary data structures for Interval-Intersect.

- D_{core} holds universe $[m]$. For every $a \in A$ with nonempty $\mathsf{core}(I_a)$, it holds the interval $\mathsf{core}(I_a)$ with label a. D_{core} is implemented via Lemma 22.
- D_{fringe} holds universe $[m]$. For every $a \in A$ and every fringe interval $I' \in \mathsf{fringe}(I_a)$ within an ith block, it holds the interval $[i, i]$ with label a.[1] D_{fringe} is also implemented via Lemma 22.
- For each block $i \in [m]$, the data structure D_i carries universe $[\ell]$. For every $a \in A$ and every fringe interval $I' \in \mathsf{fringe}(I_a)$ within the ith block, it holds the interval $\mathsf{rel}_i(I')$ labeled a. Each D_i is implemented via the assumed linear-time data structure for small universes.

Moreover, for each $a \in A$, we introduce a bit active_a, initially set, representing whether the label a is still active.

Given a query interval $J \subseteq [N]$, we perform the following queries to auxiliary data structures and gather all the returned labels.

- Query D_{core} with $\mathsf{inter}(J)$.
- Query D_{fringe} with $\mathsf{core}(J)$ as long as $\mathsf{core}(J)$ is nonempty.
- For each $J' \in \mathsf{fringe}(J)$, let J' be contained within the jth block. Then query D_j with $\mathsf{rel}_j(J')$.

For each returned label a, if a is still active, we deactivate a (i.e., unset active_a) and add a to the result label set. This concludes the description of the data structure; we now argue its correctness and efficiency.

Correctness. Let I_a be an input interval labeled a and J be a query interval. First, suppose a is reported by the query. Then, a was still active before the query. If D_{core} reported a, then $\mathsf{core}(I_a) \cap \mathsf{inter}(J) \neq \emptyset$, which easily implies $I_a \cap J \neq \emptyset$. Next, if D_{fringe} reported a, then a fringe subinterval $I' \in \mathsf{fringe}(I_a)$ resides in a block belonging to $\mathsf{core}(J)$; hence $I' \cap J \neq \emptyset$ and so $I_a \cap J \neq \emptyset$. Finally, if a is returned by D_i, then there exist $I' \in \mathsf{fringe}(I_a)$ and $J' \in \mathsf{fringe}(J)$, both within the ith block, so that $I' \cap J' \neq \emptyset$; hence trivially $I_a \cap J \neq \emptyset$.

Conversely, suppose $I_a \cap J \neq \emptyset$ and a is active. Let x be any element of the intersection and suppose x is within the ith block. First, if $i \in \mathsf{core}(I_a)$, then D_{core} definitely reports a since J intersects the ith block. Otherwise, $x \in I'$ for some $I' \in \mathsf{fringe}(I_a)$. If $i \in \mathsf{core}(J)$, then D_{fringe} reports a as the data structure carries the interval $[i, i]$ labeled a. In the opposite case, $x \in J'$ for some $J' \in \mathsf{fringe}(J)$ and so $I' \cap J' \neq \emptyset$. Therefore, D_i returns a. We conclude that an active label a is output if and only if $I_a \cap J \neq \emptyset$, which implies the correctness of our implementation.

Time. Suppose q queries are processed by our data structure. Both D_{core} and D_{fringe} maintain a universe of size m, hold at most $\mathcal{O}(|A|)$ labels, and accept at most q queries. Thus both of these auxiliary data structures initialize and process the queries in total time $\mathcal{O}(m \log m + |A| + q) = \mathcal{O}(N + |A| + q)$. Then, the data structures D_i have total universe size $m\ell = \mathcal{O}(N)$, carry at most $\mathcal{O}(|A|)$ labels in total, and accept $\mathcal{O}(q)$ queries in total. By our assumption, they all process all the queries in total time $\mathcal{O}(N + |A| + q)$. Therefore, the time complexity of our constructed data structure is also linear. □

[1] Technically, this may cause two intervals to be assigned to the same label a. Formally, in this case, we create two labels a_1, a_2 and assign them to the two intervals. Then, whenever a_i is reported by D_{fringe}, we translate the label a_i to a on the fly.

Note that in Lemma 23, the constant hidden in the $\mathcal{O}$ notation of the resulting data structure is greater than the constant hidden in the prerequisite data structure.

It remains to give a linear-time data structure for sufficiently small universe sizes.

Lemma 24 (Efficient interval intersections for tiny universes)**.** *Let b be the size of the machine word and $\ell \leqslant \sqrt{b}$. There exists a data structure for any instance of* Interval-Intersect *with the universe size at most ℓ that takes a finite set A of labels and processes q queries in total time $\mathcal{O}(\ell + |A| + q)$. The data structure requires a global table, which can be computed given only ℓ in time $\mathcal{O}(2^{\ell^2})$.*

Proof. Fix ℓ as the universe size; if the universe size of an instance is smaller than ℓ, we can artificially increase it to ℓ. Define the *state* of an instance of Interval-Intersect to be the set of *active* integer intervals of $[\ell]$: the intervals assigned to the active labels of the instance. Note that there are exactly $2^{\binom{\ell+1}{2}}$ different states since there are $\binom{\ell+1}{2}$ integer subintervals of $[\ell]$. We represent states in memory as bitmasks of length $\binom{\ell+1}{2}$; this is permitted in our computation model as $b \geqslant \binom{\ell+1}{2}$.

In the preprocessing stage, we initialize a global two-dimensional table T as follows. For a state S and an interval $J \subseteq [\ell]$, we define $T[S][J] = (S_{\text{in}}, S_{\text{out}})$, where S_{in} is the set of intervals of S intersecting J, and S_{out} is the remaining intervals of S. Observe that when an instance of Interval-Intersect is in state S, then performing a query J on it transitions it to state S_{out} and returns all labels with the corresponding intervals in S_{in}. Let also, for a state S and an interval $I \in S$, the value $F[S][I]$ be an integer such that, for each S, the mapping from intervals $I \in S$ to the values $F[S][I]$ is a bijection between S and $[|S|]$. We also explicitly compute an $\mathcal{O}(\ell^2)$-sized array M mapping integer subintervals of $[\ell]$ to their bit index within the state. The arrays can be constructed in time $2^{\binom{\ell+1}{2}}\ell^{\mathcal{O}(1)} \in \mathcal{O}(2^{\ell^2})$.

We now give an implementation of the data structure. On initialization, we compute the initial state S_{init} from ℓ and A using M. Then create buckets $B_1, \ldots, B_{|S_{\text{init}}|}$ of labels from A, and for each label $a \in A$, insert a—corresponding to an interval $I_a \in S_{\text{init}}$—into bucket $B_{F[S_{\text{init}}][I_a]}$. This can be done in total $\mathcal{O}(\ell + |A|)$ time. Let also S track the current state of the data structure; initially $S = S_{\text{init}}$. Then, for a given query J, let $(S_{\text{in}}, S_{\text{out}}) = T[S][J]$. We update the state of the data structure to S_{out} and return all labels of the intervals in S_{in} (i.e., all labels in the buckets $B_{F[S_{\text{init}}][I]}$ for $I \in S_{\text{in}}$). This can clearly be done in time $\mathcal{O}(1+t)$, where t denotes the total number of returned labels. Therefore, all queries are processed in total time $\mathcal{O}(\ell + |A| + q)$. □

Given Lemmas 23 and 24, the proof of Lemma 21 is direct.

Proof of Lemma 21. Fix the universe size $N < 2^b$, and let $\ell = \max(1, \lceil \log \log N \rceil)$. By Lemma 24, after preprocessing in time $\mathcal{O}(2^{\ell^2}) = o(N)$, we can construct a data structure for Interval-Intersect processing instances with a universe of size at most ℓ, the set of labels A, and q queries in time $\mathcal{O}(\ell + |A| + q)$. Applying Lemma 23 the first time with $N_{\max} = 2^\ell$, we get a data structure for universes of size $n \leqslant 2^\ell$ with total time $\mathcal{O}(n + |A| + q)$. Applying Lemma 23 once again with $N_{\max} = N$, we end with the required data structure. □

5 Wrap-up of the algorithmic corollaries

The proof of Theorem 6 now simply wraps the results of previous sections together.

Theorem 6. *For every graph class $\mathcal{C}$ of linear neighborhood complexity,* All-Pairs Shortest Path *can be solved in $\mathcal{O}_{\mathcal{C}}(n^2)$ time on n-vertex graphs of $\mathcal{C}$.*

Proof. By Theorem 33, $\mathcal{C}$ has bounded versatile symmetric difference. Thus, by Corollary 20, we compute an sd-degeneracy sequence of G of constant width in time $\mathcal{O}(n^2)$. By Lemma 13, we get an interval biclique partition of G with $\mathcal{O}(n)$ bicliques in further $\mathcal{O}(n \log n + |E(G)|) = \mathcal{O}(n^2)$ time. We finally invoke Theorem 5 from every vertex as source, and conclude. □

Theorem 7. *For every graph class $\mathcal{C}$ of linear neighborhood complexity, there is an $\mathcal{O}_{\mathcal{C}}(n^2)$-time algorithm that, given any adjacency matrix M of an n-vertex graph of $\mathcal{C}$, yields a data structure that computes the matrix-vector product MX in time $\mathcal{O}_{\mathcal{C}}(n)$ for any n-vector X.*

In particular, the product MN can be computed in $\mathcal{O}_{\mathcal{C}}(n^2)$ time for any *$n \times n$ matrix N.*

Proof. As in the proof of Theorem 6, we obtain an interval biclique partition of G with $\mathcal{O}(n)$ bicliques in $\mathcal{O}(n^2)$ time. We conclude with Lemma 14. □

6 Finding an sd-degeneracy sequence in randomized linear time

In this section, we prove Theorem 4: we show how to compute an sd-degeneracy sequence of width $\mathcal{O}(n^{1-1/\rho})$ in expected linear time in graphs of VC density ρ. In particular, this yields an expected linear-time algorithm that outputs sd-degeneracy sequences of constant width in classes of linear neighborhood complexity (Theorem 2). The proof relies on a result in learning theory and VC-combinatorics related to Haussler's packing lemma (Theorem 32).

A *set system* on a domain D is a set $\mathcal{F}$ of subsets of D. For $A \subseteq D$, denote

$$\mathcal{F}\lfloor_A := \{F \cap A : F \in \mathcal{F}\}.$$

The *shatter function* of a set system $\mathcal{F}$ is the function $\pi_{\mathcal{F}} : \mathbb{N} \to \mathbb{N}$ defined by

$$\pi_{\mathcal{F}}(m) := \max\{|\mathcal{F}\lfloor_A| : A \subseteq D, |A| \leqslant m\}.$$

The *VC dimension* of a set system $\mathcal{F}$ is the largest integer d such that there exists a subset $A \subseteq D$ of size d with $\mathcal{F}\lfloor_A = 2^A$.

Given a finite domain X, an *s-sample* is a tuple $S = (x_1, \ldots, x_s) \in X^s$ of s elements of X. We will consider s-samples of X drawn uniformly at random, that is, each element x_i of S is drawn independently and uniformly at random from X. (Note that an s-sample can contain the same element multiple times.)

Given a set F and an s-sample $S = (x_1, \ldots, x_s)$, denote $F \cap S := F \cap \{x_1, \ldots, x_s\}$. For a set system $\mathcal{F}$ on a domain X and integer $s \geqslant 1$, denote

$$\mathcal{F}\lfloor_s := \{(S, F \cap S) \mid S \in X^s, F \in \mathcal{F}\},$$

that is, the set of all restrictions of sets in $\mathcal{F}$ to all possible samples of size s. The next lemma originates in [38, Lemma 9] as part of a proof of Haussler's packing lemma. It is in the spirit of the classical PAC learning model, and states that sets from a set system of bounded VC dimension can be learned from a small random sample.

Lemma 25 ([38])**.** *Let $\mathcal{F}$ be a set system on a finite domain X of VC dimension at most d. For any $\varepsilon > 0$ and integer $s \geqslant 1$, there exists a map $f : \mathcal{F}\lfloor_s \to 2^X$ such that for every $F \in \mathcal{F}$,*

$$\mathop{\mathbb{P}}_{S \sim X^s}\left(|F \triangle f(S, F \cap S)| < \frac{d}{\varepsilon s} \cdot |X|\right) \geqslant 1 - \varepsilon,$$

where $S \sim X^s$ denotes that S is an s-sample of X, drawn uniformly at random.

The next algorithm relies on the existence of the map f as above, but does not require to compute it explicitly.

Lemma 26. *Fix real numbers $c, \rho \geqslant 1$ and $\delta > 0$, and let $\alpha := 1 - 1/\rho$. There is a Las Vegas randomized algorithm that, given an n-vertex graph G with (c,ρ)-polynomial neighborhood complexity and a set $U \subseteq V(G)$ with $|U| \geqslant \max(\delta n, 2)$, outputs in expected time*

$$\mathcal{O}_{c,\rho,\delta}\left(n + \sum_{u \in U} \deg_G(u)\right)$$

a set of $\Omega_{c,\rho,\delta}(|U|)$ disjoint pairs of $\mathcal{O}_{c,\rho,\delta}(n^\alpha)$-near-twins contained in U.

Proof. We may assume that $\delta \leqslant 1$. If $|U| < 2^{\rho+1}c$, then $n \leqslant |U|/\delta = \mathcal{O}_{c,\rho,\delta}(1)$. An arbitrary maximal pairing of U has size at least $|U|/4$, and every pair has symmetric difference at most $n = \mathcal{O}_{c,\rho,\delta}(n^\alpha)$, so we are done. Henceforth assume that $|U| \geqslant 2^{\rho+1}c$.

We first consider the case when U contains no pair of vertices which are twins in G. Let

$$\mathcal{F} := \{N_G(u) \mid u \in U\}$$

be the set system with domain $V(G)$ of neighborhoods of vertices of U. The members of $\mathcal{F}$ are distinct, and, since G has (c,ρ)-polynomial neighborhood complexity,

$$\pi_{\mathcal{F}}(m) \leqslant \pi_G(m) \leqslant c \cdot m^\rho$$

for every positive integer m. Let d_0 be the VC dimension of $\mathcal{F}$. Since $|\mathcal{F}| = |U| \geqslant 2$, we have $d_0 \geqslant 1$, and

$$2^{d_0} \leqslant \pi_{\mathcal{F}}(d_0) \leqslant c \cdot d_0^\rho.$$

Consequently, there is an integer $d := d(c,\rho)$, fixed independently of the input, such that $d_0 \leqslant d$.

Let

$$s := \left\lfloor \left(\frac{|U|}{2c}\right)^{1/\rho} \right\rfloor, \qquad \varepsilon := \frac{1}{16}, \qquad k := \left\lceil \frac{2dn}{\varepsilon s} \right\rceil.$$

Our lower bound on $|U|$ ensures that

$$s \geqslant \frac{1}{2}\left(\frac{|U|}{2c}\right)^{1/\rho} \qquad \text{and} \qquad cs^\rho \leqslant \frac{|U|}{2}.$$

Moreover,

$$k \leqslant 1 + \frac{4dn(2c)^{1/\rho}}{\varepsilon |U|^{1/\rho}} \leqslant \mathcal{O}_{c,\rho,\delta}\left(n^{1-\frac{1}{\rho}}\right).$$

Fix a map f as in Lemma 25, using the upper bound d on the VC dimension. For an s-sample S, we call an element $F \in \mathcal{F}$ *bad* (with respect to S) if

$$|F \mathbin{\triangle} f(S, F \cap S)| \geqslant \frac{k}{2}.$$

By Lemma 25, for any fixed $F \in \mathcal{F}$, the probability over the random choice of S that F is bad is at most ε. Thus, the expected number of bad elements of $\mathcal{F}$ is at most $\varepsilon|U|$. By Markov's inequality,

$$\mathop{\mathbb{P}}_{S \sim V(G)^s}\left(|\{F \in \mathcal{F} \mid F \text{ is bad with respect to } S\}| > |U|/8\right) \leqslant \frac{\varepsilon|U|}{|U|/8} = 8\varepsilon = \frac{1}{2}.$$

One attempt of the algorithm draws an s-sample S of $V(G)$ and lets $\widehat{S} \subseteq V(G)$ be its underlying set. Consider the set system

$$\mathcal{F}\lfloor_{\widehat{S}} := \{N_G(u) \cap \widehat{S} \mid u \in U\}.$$

We have

$$|\mathcal{F}\lfloor_{\widehat{S}}| \leqslant \pi_{\mathcal{F}}(|\widehat{S}|) \leqslant c|\widehat{S}|^{\rho} \leqslant cs^{\rho} \leqslant \frac{|U|}{2}.$$

For each $\widehat{F} \in \mathcal{F}\lfloor_{\widehat{S}}$, let the *bucket* $B(\widehat{F})$ be the set of sets $F \in \mathcal{F}$ such that $F \cap \widehat{S} = \widehat{F}$. Thus, the buckets partition $\mathcal{F}$.

Inside every bucket, we arbitrarily choose a maximum set of pairwise disjoint pairs, leaving at most one set unpaired. Let $\mathcal{P}$ be the collection of pairs obtained from all the buckets, and let $\mathcal{P}_{\text{good}} \subseteq \mathcal{P}$ consist of those pairs $\{F_1, F_2\}$ such that $|F_1 \triangle F_2| \leqslant k$. If $|\mathcal{P}_{\text{good}}| \geqslant |U|/8$, we output the corresponding pairs of vertices in U; otherwise, we discard the attempt and resample S.

Claim 27. *One attempt takes expected time* $\mathcal{O}(n + \sum_{u \in U} \deg_G(u))$.

Proof of claim. Sampling S takes $\mathcal{O}(n)$ time. Computing $\mathcal{F}\lfloor_{\widehat{S}}$ and partitioning $\mathcal{F}$ into buckets takes expected time $\mathcal{O}(n + \sum_{u \in U} \deg_G(u))$ by scanning the adjacency lists of the vertices in U. Building the candidate pairs takes $\mathcal{O}(|U|)$ time.

Given a candidate pair $\{F_1, F_2\}$, computing $F_1 \triangle F_2$ takes $\mathcal{O}(|F_1| + |F_2|)$ time. As the candidate pairs are disjoint, checking all of them takes $\mathcal{O}(|U| + \sum_{u \in U} \deg_G(u))$ time. ⌟

Claim 28. *Every attempt succeeds with probability at least* $1/2$.

Proof of claim. Let F_1, F_2 lie in the same bucket. Then $F_1 \cap S = F_2 \cap S$, and hence

$$|F_1 \triangle F_2| \leqslant |F_1 \triangle f(S, F_1 \cap S)| + |F_2 \triangle f(S, F_2 \cap S)|.$$

Therefore, every pair in $\mathcal{P} \setminus \mathcal{P}_{\text{good}}$ contains a bad element of $\mathcal{F}$. As the candidate pairs are disjoint, the number of such pairs is at most the number of bad elements.

If the buckets are $B_1, \ldots, B_t$, then $t = |\mathcal{F}\lfloor_{\widehat{S}}| \leqslant |U|/2$ and

$$|\mathcal{P}| = \sum_{j=1}^{t} \left\lfloor \frac{|B_j|}{2} \right\rfloor \geqslant \frac{|U| - t}{2} \geqslant \frac{|U|}{4}.$$

With probability at least $1/2$, at most $|U|/8$ elements of $\mathcal{F}$ are bad, and on that event

$$|\mathcal{P}_{\text{good}}| \geqslant |\mathcal{P}| - \frac{|U|}{8} \geqslant \frac{|U|}{8}.$$

Thus, the attempt succeeds. ⌟

The expected number of attempts is at most two, so Claim 27 proves the claimed expected running time when U contains no pair of twins in G.

In the general case, construct a graph G' by adding, for every $u \in U$, a new private pendant vertex x_u adjacent only to u. Then the vertices of U have pairwise distinct neighborhoods in G'. We claim that G' has $(c+2, \rho)$-polynomial neighborhood complexity. Indeed, fix a nonempty set $A \subseteq V(G')$, and put $A_0 := A \cap V(G)$ and $A_1 := A \setminus V(G)$. The traces of the old vertices are determined by their traces on A_0, except that at most $|A_1|$ of them can acquire a private pendant

element. The new pendant vertices have traces among $\emptyset$ and the singletons $\{u\}$ for $u \in A_0$. Consequently, the total number of traces on A is at most

$$c|A|^\rho + |A| + 1 \leqslant (c+2)|A|^\rho.$$

Also, $|V(G')| \leqslant 2n$, $|U| \geqslant (\delta/2)|V(G')|$, and $\deg_{G'}(u) = \deg_G(u) + 1$ for every $u \in U$. We can therefore apply the twin-free case to G' and U with constants $c+2$, ρ, and $\delta/2$. This application has the claimed expected running time. Finally, deleting the new pendant vertices cannot increase a symmetric difference, so the returned pairs satisfy the asserted bound in G as well. □

We now present the randomized linear-time algorithm computing an sd-degeneracy sequence.

Theorem 4. *For every real number $\rho \geqslant 1$ and every graph class $\mathcal{C}$ of VC density ρ, there is a randomized algorithm that, given a graph $G \in \mathcal{C}$, outputs an sd-degeneracy sequence of G of width $\mathcal{O}_{\mathcal{C}}(n^{1-\frac{1}{\rho}})$ in expected time $\mathcal{O}_{\mathcal{C}}\big(|V(G)| + |E(G)|\big)$.*

Proof. Let $c \geqslant 1$ be such that every graph in $\mathcal{C}$ has (c,ρ)-polynomial neighborhood complexity. Write $n := |V(G)|$, $m := |E(G)|$, and $\alpha := 1 - 1/\rho$. We build a sequence $G_1, G_2, \ldots$ of induced subgraphs of G, with $G_1 := G$. Every G_i still has (c,ρ)-polynomial neighborhood complexity.

In every iteration i with $n_i := |V(G_i)| \geqslant 3$, let $U_i \subseteq V(G_i)$ be the set of $\lceil n_i/2 \rceil$ vertices of G_i with smallest degrees, breaking ties arbitrarily. By Lemma 26, applied to G_i, U_i, and $\delta = 1/2$, in expected time

$$\mathcal{O}_{c,\rho}\left(n_i + \sum_{u \in U_i} \deg_{G_i}(u)\right)$$

we find a set $\mathcal{P}_i$ of $\Omega_{c,\rho}(n_i)$ disjoint pairs of $\mathcal{O}_{c,\rho}(n_i^\alpha)$-near-twins inside U_i. We order every pair of $\mathcal{P}_i$ arbitrarily, append the ordered pairs to the output sequence, and obtain G_{i+1} by deleting the first vertex of every pair. When at most two vertices remain, we append their pair if there are two vertices, and terminate.

Claim 29. *The output is an sd-degeneracy sequence of G of width $\mathcal{O}_{c,\rho}(n^\alpha)$.*

Proof of claim. Consider a pair (u,v) from a batch $\mathcal{P}_i$ when it is reached in the output sequence. Earlier pairs of the same batch are vertex-disjoint from (u,v), and the graph then present is an induced subgraph of G_i. Hence, its symmetric difference is no larger than it was in G_i, where (u,v) is a pair of $\mathcal{O}_{c,\rho}(n_i^\alpha)$-near-twins. Since $n_i \leqslant n$, this is $\mathcal{O}_{c,\rho}(n^\alpha)$. If the algorithm terminates with two vertices, their symmetric difference is at most 2, and hence at most $2n^\alpha$. ⌟

Claim 30. *The expected running time is $\mathcal{O}_{c,\rho}(n+m)$.*

Proof of claim. We maintain $\deg_{G_i}(v)$ for every current vertex v, initialized in $\mathcal{O}(n+m)$ time. When vertices are deleted, we update the degrees of their surviving neighbors; all such updates cost $\mathcal{O}(n+m)$ in total. In iteration i, a linear-time selection algorithm applied to these maintained degrees finds the $\lceil n_i/2 \rceil$ vertices of smallest degree in $\mathcal{O}(n_i)$ time.

By conditional linearity of expectation, the remaining expected work is bounded by

$$\mathcal{O}_{c,\rho}\left(\sum_i n_i + \sum_i \sum_{u \in U_i} \deg_{G_i}(u)\right). \tag{1}$$

Every iteration removes $\Omega_{c,\rho}(n_i)$ vertices. Thus, the sequence $(n_i)_i$ decreases geometrically, and

$$\sum_i n_i = \mathcal{O}_{c,\rho}(n).$$

For the second term of (1), we have

$$\sum_i \sum_{u \in U_i} \deg_{G_i}(u) = \sum_{r \geqslant 1} |S_r|, \qquad \text{with } S_r := \{(i,u) \mid u \in U_i \text{ and } \deg_{G_i}(u) \geqslant r\}.$$

For every r with $S_r \neq \emptyset$, let i_r be the smallest index such that U_{i_r} contains a vertex of degree at least r. The geometric decrease and $|U_i| = \Theta(n_i)$ imply

$$|S_r| \leqslant \sum_{i \geqslant i_r} |U_i| = \mathcal{O}_{c,\rho}(|U_{i_r}|).$$

For every $r \geqslant 1$, let

$$A_r := \{v \in V(G) \mid \deg_G(v) \geqslant r\}.$$

Because U_{i_r} consists of the lower half of the degree order and contains a vertex of degree at least r, every vertex of $V(G_{i_r}) \setminus U_{i_r}$ has degree at least r in G_{i_r}, and hence also in G. Therefore,

$$\left\lfloor \frac{n_{i_r}}{2} \right\rfloor \leqslant |A_r| \qquad \text{and consequently} \qquad |U_{i_r}| \leqslant |A_r| + 1.$$

Since $S_r = \emptyset$ for $r \geqslant n$ and $\sum_{r \geqslant 1} |A_r| = 2m$, we obtain

$$\sum_i \sum_{u \in U_i} \deg_{G_i}(u) = \sum_{r=1}^{n-1} |S_r| \leqslant \mathcal{O}_{c,\rho}\left(\sum_{r=1}^{n-1} (|A_r| + 1)\right) = \mathcal{O}_{c,\rho}(n+m).$$

Substituting both estimates into (1) proves the claim. ⌟

We conclude by Claims 29 and 30. □

7 Finding Small Patterns

We next design a randomized linear-time algorithm that returns a triangle (if one exists) in graphs from a class of linear neighborhood complexity. The main idea is to use an sd-degeneracy sequence of the input graph to guide the search. In the next three (almost-)linear algorithms we assume that the edges of the input graph are preprocessed into a static dictionary in $\mathcal{O}(n+m)$ expected time and $\mathcal{O}(n+m)$ space, supporting adjacency queries in $\mathcal{O}(1)$ expected time.

Theorem 8. *For every graph class $\mathcal{C}$ of linear neighborhood complexity,* Triangle Detection *can be solved in expected time $\mathcal{O}_{\mathcal{C}}(n+m)$ on n-vertex m-edge graphs of $\mathcal{C}$.*

Proof. Let G be any input graph. By Theorem 2, we get an sd-degeneracy sequence $(u_1, v_1), \ldots, (u_{n-1}, v_{n-1})$ of G of width $d = \mathcal{O}_{\mathcal{C}}(1)$ in randomized time $\mathcal{O}_{\mathcal{C}}(n+m)$.

For every $i \in [n-1]$, we define $G_i := G - \{u_1, \ldots, u_{i-1}\}$. Note that to find a triangle in $G = G_1$ (if any), it is enough to return a triangle of G_1 that contains u_1 or obtain the guarantee that *not* all the triangles of G_1 contain u_1, and recursively apply this strategy to G_2. Indeed, at each step we either find a triangle in G_i or move to G_{i+1} with the promise that G_{i+1} contains a triangle if and only if G_i contains one.

We set

$$A_i := N_{G_i}(u_i) \setminus N_{G_i}(v_i) \quad \text{and} \quad X_i := N_{G_i}(u_i) \cap N_{G_i}(v_i)$$

for every $i \in [n-1]$.

For i going from 1 to $n-2$, the algorithm proceeds as follows. Compute A_i in time $\mathcal{O}(|N_{G_i}(u_i)|)$ by filtering out the vertices of $N_{G_i}(u_i)$ that are adjacent to v_i. By assumption, $|A_i| \leqslant d$. For every $a \in A_i$ and every $b \in N_{G_i}(u_i)$ check if $ab \in E(G_i)$. If so, return the triangle u_iab (break from the for loop, and terminate). Even if all the checks are unsuccessful, this takes $\mathcal{O}\left((d+1) \cdot |N_{G_i}(u_i)|\right)$ time. If no triangle is found, remove u_i from G_i in time $\mathcal{O}(|N_{G_i}(u_i)|)$ to obtain G_{i+1} (and move to the next iteration of the for loop).

Correctness. If there is a triangle in G_i that contains u_i but the algorithm does not return one at the ith iteration of the for loop, it is of the form u_ixy with $\{x, y\} \subseteq X_i$. Then G_{i+1} still contains at least the triangle v_ixy.

Running time. Computing the sd-degeneracy sequences takes randomized time $\mathcal{O}_{\mathcal{C}}(n+m)$. The rest of the algorithm takes time

$$\mathcal{O}\left(\sum_{i \in [n-2]} (d+1) \cdot (|N_{G_i}(u_i)| + 1)\right) = \mathcal{O}_{\mathcal{C}}(n+m). \qquad \square$$

We now build on the scheme of the previous proof to detect a 4-vertex clique. The randomized almost linear algorithm relies on both an sd-degeneracy sequence *and* a Welzl order.

Theorem 9. *For every graph class $\mathcal{C}$ of linear neighborhood complexity, a K_4 subgraph can be found in expected $\mathcal{O}_{\mathcal{C}}(n \log^5 n + m \log n)$ or in deterministic $\mathcal{O}_{\mathcal{C}}(n^2)$ time on n-vertex m-edge graphs of $\mathcal{C}$.*

Proof. We obtain an sd-degeneracy sequence $(u_1, v_1), \ldots, (u_{n-1}, v_{n-1})$ of the input graph G of width $d = \mathcal{O}_{\mathcal{C}}(1)$ in randomized $\mathcal{O}_{\mathcal{C}}(n+m)$ time by Theorem 2, or deterministic $\mathcal{O}(n^2)$ time by Theorem 1. For every $i \in [n-1]$, we define G_i, A_i, X_i as in the proof of Theorem 8.

Our general strategy remains the same. Now, for every $a \in A_i$, we do not merely need to know if $Z_{i,a} \neq \emptyset$ but rather whether $G_i[Z_{i,a}]$ contains an edge, where

$$Z_{i,a} := N_{G_i}(u_i) \cap N_{G_i}(a).$$

We start describing the deterministic $\mathcal{O}(n^2)$-time algorithm, which naturally uses Theorem 1 as opening step.

Let M be an arbitrary adjacency matrix of G, on which we compute the data structure of Theorem 7 (associated sparse matrix) in $\mathcal{O}_{\mathcal{C}}(n^2)$ time. For i going from 1 to $n-3$, the algorithm proceeds as follows. It computes A_i in time $\mathcal{O}(|N_{G_i}(u_i)|)$. For every $a \in A_i$, it computes the indicator (column) vector $z_{i,a}$ of $Z_{i,a}$ in time $\mathcal{O}(n)$.

We observe that

$$z_{i,a}^T M z_{i,a} = 2|E(G_i[Z_{i,a}])|,$$

and compute the matrix-vector product $y_{i,a} := Mz_{i,a}$ in time $\mathcal{O}_{\mathcal{C}}(n)$, by Theorem 7. We now check in $\mathcal{O}(n)$ time whether there is a $j \in [n]$ such that both $z_{i,a}[j]$ and $y_{i,a}[j]$ are nonzero. If this happens, we know that there is at least one edge in $G_i[Z_{i,a}]$ incident to the jth vertex, say x, of G. We can find the other endpoint of such an edge, say y, in $\mathcal{O}(n)$ time. We return the 4-vertex clique $\{u_i, a, x, y\}$ (and terminate).

If no K_4 is found, we remove u_i from G_i in time $\mathcal{O}(|N_{G_i}(u_i)|)$ to obtain G_{i+1} (and move to the next iteration of the for loop). This concludes the algorithm.

Each iteration takes $\mathcal{O}((d+1)n) = \mathcal{O}_{\mathcal{C}}(n)$ time because $|A_i| \leqslant d$. The preprocessing of M takes $\mathcal{O}_{\mathcal{C}}(n^2)$ time. Therefore, the overall running time is $\mathcal{O}_{\mathcal{C}}(n^2)$. The correctness follows as in the proof of Theorem 8.

We now describe the randomized $\mathcal{O}_{\mathcal{C}}(n \log^5 n + m \log n)$-time algorithm, which first calls Theorem 2. In addition to the sd-degeneracy sequence, we will rely on a Welzl order for G; more specifically, in time $\mathcal{O}_{\mathcal{C}}((n+m)\log n)$ we compute a vertex ordering $\prec$: $w_1 w_2 \dots w_n$ such that the neighborhood of every vertex of G is the union of $\mathcal{O}_{\mathcal{C}}(\log^2 n)$ $\prec$-intervals [23]. We further compute, for every $v \in V(G)$, the $\prec$-intervals $I_1(v), \dots, I_{h_v}(v)$ with $h_v = \mathcal{O}_{\mathcal{C}}(\log^2 n)$ such that $N_G(v) = \biguplus_{j \in [h_v]} I_j(v)$; overall this takes time $\mathcal{O}(n+m)$. We observe that the orderings $u_1 u_2 \dots$ and $w_1 w_2 \dots$ are a priori unrelated. For every $v \in V(G)$, we denote by $\pi(v)$ the integer i such that $v = w_i$.

We create the following planar point set P, with $|P| = 2m$. For every $uv \in E(G)$, we add points $(\pi(u), \pi(v))$ and $(\pi(v), \pi(u))$ to P. In $\mathcal{O}(m \log n)$ preprocessing time, a data structure $\mathcal{D}_P$ can be computed that answers rectangle query (given an axis-parallel rectangle R, yield a point in $R \cap P$) in time $\mathcal{O}(\log n)$ [20, Theorem 2].

Let

$$\widehat{Z}_{i,a} := N_G(u_i) \cap N_G(a),$$

and note the G subscripts (so in particular $\widehat{Z}_{i,a} \supseteq Z_{i,a}$). We proceed as the deterministic $\mathcal{O}_{\mathcal{C}}(n^2)$-time algorithm, except we decide if $G[\widehat{Z}_{i,a}]$ (not $G_i[Z_{i,a}]$) has an edge with the $\prec$-intervals $I_j(v)$ and $\mathcal{D}_P$. (We do not compute M nor $z_{i,a}$.) We compute $\widehat{Z}_{i,a}$ as the intersection of

$$\biguplus_{j \in [h_{u_i}]} I_j(u_i) \text{ and } \biguplus_{j \in [h_a]} I_j(a).$$

This intersection is itself the union of $\mathcal{O}_{\mathcal{C}}(\log^2 n)$ $\prec$-intervals, say $J_1, \dots, J_p$, computed in $\mathcal{O}_{\mathcal{C}}(\log^2 n)$ time. For every $j, k \in [p]$, we test if $E_G(J_j, J_k)$ is nonempty with the query rectangle $J_j \times J_k$ on $\mathcal{D}_P$. If so, we return the corresponding 4-vertex clique, and terminate. This test takes $\mathcal{O}_{\mathcal{C}}(\log^5 n)$ time in total, hence the claimed $\mathcal{O}_{\mathcal{C}}(n \log^5 n + m \log n)$ running time. □

We can go one step further and find 5-vertex cliques in randomized almost linear time.

Theorem 10. *For every graph class $\mathcal{C}$ of linear neighborhood complexity, a K_5 subgraph can be found in expected time $\mathcal{O}_{\mathcal{C}}(n \log^9 n + m \log^5 n)$ on n-vertex m-edge graphs of $\mathcal{C}$.*

Proof. The proof exactly follows the second part of that of Theorem 9. We now need to detect a triangle of G between a triple J_h, J_j, J_k of $\prec$-intervals of $\widehat{Z}_{i,a} = N_G(u_i) \cap N_G(a)$ with $h, j, k \in [p]$. Instead of the planar point set P, we build the following integral point set Q in $\mathbb{Z}^4$. To simplify the presentation, we assume that $V(G) = [n]$ and that $1, 2, \dots, n$ is the Welzl order computed in time $\mathcal{O}_{\mathcal{C}}((n+m)\log n)$ such that the neighborhood of every vertex of G is the union of $\mathcal{O}_{\mathcal{C}}(\log^2 n)$ $\prec$-intervals [23].

For every $uv \in E(G)$, let

$$N_G(u) \cap N_G(v) = [\ell_1, r_1] \uplus \dots \uplus [\ell_s, r_s]$$

where each $[\ell_t, r_t]$ is an $\prec$-interval and $s = \mathcal{O}_{\mathcal{C}}(\log^2 n)$, and add the point

$$q_{u,v,t} := (u, v, \ell_t, -r_t)$$

to Q, for each $t \in [s]$. We build in time $\mathcal{O}(|Q| \log^3 |Q|) = \mathcal{O}_{\mathcal{C}}(m \log^5 n)$ a 4-dimensional orthogonal range data structure $\mathcal{D}_Q$ that, given a query box, returns in time $\mathcal{O}(\log^3 |Q|) = \mathcal{O}_{\mathcal{C}}(\log^3 n)$ one point of Q contained in the box, if one exists [21, Theorem 5.11].

Now for every triple J_h, J_j, and $J_k = [\alpha_k, \beta_k]$, we query the box $J_h \times J_j \times (-\infty, \beta_k] \times (-\infty, -\alpha_k]$ on $\mathcal{D}_Q$. If the query returns a point $q_{u,v,t} \in Q$, it holds that $u \in J_h$, $v \in J_j$, $uv \in E(G)$, $\ell_t \leqslant \beta_k$, and $r_t \geqslant \alpha_k$. In particular, the interval $[\ell_t, r_t]$ of $N_G(u) \cap N_G(v)$ intersects J_k. Let z be any vertex in $[\ell_t, r_t] \cap J_k$. It holds that uvz is a triangle in $\widehat{Z}_{i,a} \supseteq J_k$, and we return the 5-vertex clique $\{u_i, a, u, v, z\}$. If no K_5 is found, u_i is removed and the algorithm continues with G_{i+1} (next iteration).

Each iteration takes time $\mathcal{O}_{\mathcal{C}}((\log^2 n)^3 \log^3 n) = \mathcal{O}_{\mathcal{C}}(\log^9 n)$, hence the overall running time $\mathcal{O}_{\mathcal{C}}(n \log^9 n + m \log^5 n)$. □

One should, however, not expect an almost-linear dependence on n uniformly over all clique sizes k. The Exponential Time Hypothesis (ETH) asserts that there exists a constant $\lambda > 0$ such that 3-SAT on n variables cannot be solved in time $O(2^{\lambda n})$ [33].

Theorem 31. *There is a class $\mathcal{C}$ of linear neighborhood complexity such that k-Clique cannot be solved in time $f(k)\, n^{o(k/\log k)}$ for any function f, unless the ETH fails.*

Proof. By [41, Theorem 1.3] specialized to bipartite 3-regular graphs, as stated in [34, Theorem 1.4], unless the ETH fails, 2-CSP with alphabet Σ and q constraints cannot be solved in time $g(q)\, |\Sigma|^{o(q/\log q)}$ for any computable function g.

Let Γ be such an instance, and $H = (X, Y; E)$ be its constraint graph. Thus $|E| = q$. For every constraint $e = uv \in E$, where $u \in X$ and $v \in Y$, let $C_e \subseteq \Sigma^2$ be the set of pairs allowed by e. We construct the cell gadget from the proof of [10, Theorem 3] with $r = 3$ and tile set C_e: each of its five *blocks* contains a copy of (a, b) for every $(a, b) \in C_e$. Denote the two border blocks at the ends of its horizontal (resp. vertical) path by L_e and R_e (resp. by U_e and D_e), and the central block by M_e. Each block is a clique, and every copy of (a, b) in M_e is adjacent to every vertex in $L_e \cup R_e \cup U_e \cup D_e$ except the four copies of (a, b).

For every $u \in X$, let e_1, e_2, e_3 be the edges of H incident to u. For every $i \in [3]$, we add edges between R_{e_i} and $L_{e_{i \bmod 3+1}}$, making two vertices adjacent precisely when their first coordinates differ. Similarly, for each variable $v \in Y$, we arbitrarily list the edges of H incident to v as f_1, f_2, f_3 and, for every $i \in [3]$, add edges between D_{f_i} and $U_{f_{i \bmod 3+1}}$, making two vertices adjacent precisely when their second coordinates differ.

Denote the resulting graph by G_Γ. By the correctness argument of [10], Γ is satisfiable if and only if $\alpha(G_\Gamma) \geqslant 5q$, and

$$|V(G_\Gamma)| = 5 \sum_{e \in E} |C_e| \leqslant 5q|\Sigma|^2.$$

Moreover, by the radius-2 merge sequence from [10], $\mathsf{mw}_2(G_\Gamma) = \mathsf{mw}_{r-1}(G_\Gamma) \leqslant 4r - 3 = 9$. Observe indeed that the upper bound on the width of the merge sequences does not depend on which pairs of border blocks from distinct cells are connected. Let $\mathcal{C}$ consist of the complements of all graphs G_Γ. By [9, Theorem 1.5], the graphs G_Γ form a class of linear neighborhood complexity. This property is preserved under complementation.

Now an algorithm for k-Clique on $\mathcal{C}$ running in time $f(k) n^{o(k/\log k)}$, applied to $\overline{G_\Gamma}$ with $k = 5q$, would decide Γ in time

$$f(5q)\big(5q|\Sigma|^2\big)^{o(q/\log q)} = g(q)\, |\Sigma|^{o(q/\log q)}.$$

The polynomial time needed to construct G_Γ is absorbed in this bound. This contradicts the ETH lower bound of 2-CSP. □

AI Disclosure. A number of proofs in this work have been developed with the assistance of ChatGPT 5.5 Pro.

- The idea to use a longest-common-extension data structure, critical to Theorem 1, has been brought up by ChatGPT.
- The details of the bootstrapping process presented in Lemma 23 and Lemma 24 were developed with ChatGPT's assistance. (The data structure of Lemma 22 was developed by the authors.)
- A number of technical details in the proof of Theorem 2 have been developed with ChatGPT's assistance.

The suspected generalizations in Theorems 3 and 4 were confirmed by GPT-5.6 Sol Pro, which also found the extensions in Theorems 9 and 10 of the triangle-detection algorithm. In each case, while ChatGPT suggested pieces of the write-up, the final write-up is due to the authors and has been polished and verified by us.

References

[1] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. doi:10.1007/BF02523189.

[2] Emile Anand, Jan van den Brand, and Rose McCarty. The structural complexity of matrix-vector multiplication. *Advances in Neural Information Processing Systems*, 38:37145–37183, 2026.

[3] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradžev. On economical construction of the transitive closure of a directed graph. *Doklady Akademii Nauk SSSR*, 194(3):487–488, 1970. In Russian.

[4] Jakub Balabán, Daniel Mock, and Peter Rossmanith. Solving partial dominating set and related problems using twin-width. In Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak, editors, *50th International Symposium on Mathematical Foundations of Computer Science, MFCS 2025, Warsaw, Poland, August 25-29, 2025*, volume 345 of *LIPIcs*, pages 13:1–13:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025. URL: https://doi.org/10.4230/LIPIcs.MFCS.2025.13, doi:10.4230/LIPICS.MFCS.2025.13.

[5] Max Bannach, Florian Andreas Marwitz, and Till Tantau. Faster Graph Algorithms Through DAG Compression. In Olaf Beyersdorff, Mamadou Moustapha Kanté, Orna Kupferman, and Daniel Lokshtanov, editors, *41st International Symposium on Theoretical Aspects of Computer Science (STACS 2024)*, volume 289 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.STACS.2024.8, doi:10.4230/LIPIcs.STACS.2024.8.

[6] Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, Lecture Notes in Computer Science, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

[7] Pierre Bergé, Édouard Bonnet, Hugues Déprés, and Rémi Watrigant. Approximating highly inapproximable problems on graphs of bounded twin-width. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, Hamburg, Germany, March*

7-9, 2023, volume 254 of *LIPIcs*, pages 10:1–10:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.STACS.2023.10, doi:10.4230/LIPICS.STACS.2023.10.

[8] Omer Berkman and Uzi Vishkin. Finding level-ancestors in trees. *J. Comput. Syst. Sci.*, 48(2):214–230, 1994. doi:10.1016/S0022-0000(05)80002-9.

[9] Marthe Bonamy and Colin Geniet. χ-boundedness and neighbourhood complexity of bounded merge-width graphs. *CoRR*, abs/2504.08266, 2025. URL: https://doi.org/10.48550/arXiv.2504.08266, arXiv:2504.08266, doi:10.48550/ARXIV.2504.08266.

[10] Édouard Bonnet, Maël Dumas, and Julien Duron. Independent set hardness in graphs of bounded twin-width and low-radius merge-width, IPEC 2026. URL: https://arxiv.org/abs/2607.00244, arXiv:2607.00244.

[11] Édouard Bonnet, Julien Duron, John Sylvester, and Viktor Zamaraev. Symmetric-difference (degeneracy) and signed tree models. In Rastislav Královic and Antonín Kucera, editors, *49th International Symposium on Mathematical Foundations of Computer Science, MFCS 2024, August 26-30, 2024, Bratislava, Slovakia*, volume 306 of *LIPIcs*, pages 32:1–32:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.MFCS.2024.32, doi:10.4230/LIPICS.MFCS.2024.32.

[12] Édouard Bonnet, Colin Geniet, Eun Jung Kim, and Sungmin Moon. Fast shortest path in graphs with sparse signed tree models and applications. In Sayan Bhattacharya, Danupon Nanongkai, Michael Benedikt, and Gabriele Puppis, editors, *53rd International Colloquium on Automata, Languages, and Programming, ICALP 2026, Royal Holloway, University of London, Egham, United Kingdom, July 7-10, 2026*, volume 374 of *LIPIcs*, pages 40:1–40:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026. URL: https://doi.org/10.4230/LIPIcs.ICALP.2026.40, doi:10.4230/LIPICS.ICALP.2026.40.

[13] Édouard Bonnet, Colin Geniet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width III: max independent set, min dominating set, and coloring. *SIAM Journal on Computing*, 53(5):1602–1640, 2024. doi:10.1137/21M142188X.

[14] Édouard Bonnet, Ugo Giocanti, Patrice Ossona de Mendez, and Stéphan Thomassé. Twin-width V: linear minors, modular counting, and matrix multiplication. In Petra Berenbrink, Patricia Bouyer, Anuj Dawar, and Mamadou Moustapha Kanté, editors, *40th International Symposium on Theoretical Aspects of Computer Science, STACS 2023, Hamburg, Germany, March 7-9, 2023*, volume 254 of *LIPIcs*, pages 15:1–15:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.STACS.2023.15, doi:10.4230/LIPICS.STACS.2023.15.

[15] Édouard Bonnet, Eun Jung Kim, Amadeus Reinald, Stéphan Thomassé, and Rémi Watrigant. Twin-width and polynomial kernels. *Algorithmica*, 84(11):3300–3337, 2022. URL: https://doi.org/10.1007/s00453-022-00965-5, doi:10.1007/S00453-022-00965-5.

[16] Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width I: tractable FO model checking. *J. ACM*, 69(1):3:1–3:46, 2022. doi:10.1145/3486655.

[17] Édouard Bonnet, Jaroslav Nešetřil, Patrice Ossona de Mendez, Sebastian Siebertz, and Stéphan Thomassé. Twin-width and permutations. *Log. Methods Comput. Sci.*, 20(3), 2024. URL: https://doi.org/10.46298/lmcs-20(3:4)2024, doi:10.46298/LMCS-20(3:4)2024.

[18] Jean Cardinal, Rose McCarty, and Yelena Yuditsky. Biclique decompositions from Welzl orders, 2026. URL: https://arxiv.org/abs/2606.09785, arXiv:2606.09785.

[19] Timothy M. Chan, Hsien-Chih Chang, Jie Gao, Sándor Kisfaludi-Bak, Hung Le, and Da Wei Zheng. Truly subquadratic time algorithms for diameter and related problems in graphs of bounded VC-dimension. In *66th Annual Symposium on Foundations of Computer Science, FOCS 2025, Sydney, Australia, December 14-17, 2025*, 2025. URL: https://arxiv.org/abs/2510.16346.

[20] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988. doi:10.1137/0217026.

[21] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition.* Springer, 2008. doi:10.1007/978-3-540-77974-2.

[22] Karolina Drabik, Maël Dumas, Colin Geniet, Jakub Nowakowski, Michał Pilipczuk, and Szymon Toruńczyk. Variants of merge-width and applications. *CoRR*, abs/2602.23867, 2026. URL: https://doi.org/10.48550/arXiv.2602.23867, arXiv:2602.23867, doi:10.48550/ARXIV.2602.23867.

[23] Jan Dreier and Clemens Kuske. Near-linear time computation of Welzl orders on graphs with linear neighborhood complexity. *CoRR*, abs/2602.14625, 2026. URL: https://doi.org/10.48550/arXiv.2602.14625, arXiv:2602.14625, doi:10.48550/ARXIV.2602.14625.

[24] Jan Dreier and Szymon Toruńczyk. Merge-width and first-order model checking. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, STOC '25, page 1944–1955, New York, NY, USA, 2025. Association for Computing Machinery. doi:10.1145/3717823.3718259.

[25] Lech Duraj, Filip Konieczny, and Krzysztof Potepa. Better diameter algorithms for bounded vc-dimension graphs and geometric intersection graphs. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, Royal Holloway, London, United Kingdom, September 2-4, 2024*, volume 308 of *LIPIcs*, pages 51:1–51:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.ESA.2024.51, doi:10.4230/LIPICS.ESA.2024.51.

[26] Harold N. Gabow and Robert Endre Tarjan. A linear-time algorithm for a special case of Disjoint Set Union. *J. Comput. Syst. Sci.*, 30(2):209–221, 1985. doi:10.1016/0022-0000(85)90014-5.

[27] Jakub Gajarský, Petr Hliněný, Jan Obdržálek, Sebastian Ordyniak, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, and Somnath Sikdar. Kernelization using structural parameters on sparse graph classes. *Journal of Computer and System Sciences*, 84:219–242, 2017. doi:10.1016/j.jcss.2016.09.002.

[28] Jakub Gajarský, Michał Pilipczuk, Wojciech Przybyszewski, and Szymon Toruńczyk. Twin-width and types. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, Paris, France, July 4-8, 2022*, volume 229 of *LIPIcs*, pages 123:1–123:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.ICALP.2022.123, doi:10.4230/LIPICS.ICALP.2022.123.

[29] Robert Ganian, Filip Pokrývka, André Schidler, Kirill Simonov, and Stefan Szeider. Weighted model counting with twin-width. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, Haifa, Israel, August 2-5, 2022*, volume 236 of *LIPIcs*, pages 15:1–15:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.SAT.2022.15, doi:10.4230/LIPICS.SAT.2022.15.

[30] Martin Grohe, Stephan Kreutzer, and Sebastian Siebertz. Deciding first-order properties of nowhere dense graphs. *J. ACM*, 64(3):17:1–17:32, 2017. doi:10.1145/3051095.

[31] Martin Grohe and Daniel Neuen. Isomorphism for tournaments of small twin width. *TheoretiCS*, 5, 2026. URL: https://doi.org/10.46298/theoretics.26.4, doi:10.46298/THEORETICS.26.4.

[32] David Haussler. Sphere packing numbers for subsets of the boolean n-cube with bounded Vapnik-Chervonenkis dimension. *J. Comb. Theory A*, 69(2):217–232, 1995.

[33] 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.

[34] Karthik C. S., Dániel Marx, Marcin Pilipczuk, and Uéverton S. Souza. Conditional lower bounds for sparse parameterized 2-CSP: A streamlined proof. In Merav Parter and Seth Pettie, editors, *2024 Symposium on Simplicity in Algorithms, SOSA 2024, Alexandria, VA, USA, January 8-10, 2024*, pages 383–395. SIAM, 2024. doi:10.1137/1.9781611977936.35.

[35] László Kozma and Michal Opler. Fast and simple multiplication of bounded twin-width matrices. *CoRR*, abs/2602.20023, 2026. URL: https://doi.org/10.48550/arXiv.2602.20023, arXiv:2602.20023, doi:10.48550/ARXIV.2602.20023.

[36] Stefan Kratsch and Florian Nelles. Efficient parameterized algorithms for computing all-pairs shortest paths. *Discret. Appl. Math.*, 341:102–119, 2023. URL: https://doi.org/10.1016/j.dam.2023.07.029, doi:10.1016/J.DAM.2023.07.029.

[37] Stefan Kratsch, Florian Nelles, and Alexandre Simon. On triangle counting parameterized by twin-width. *CoRR*, abs/2202.06708, 2022. URL: https://arxiv.org/abs/2202.06708, arXiv:2202.06708.

[38] Andrey Kupavskii and Nikita Zhivotovskiy. When are epsilon-nets small? *Journal of Computer and System Sciences*, 110:22–36, 2020. URL: https://www.sciencedirect.com/science/article/pii/S0022000020300027, doi:https://doi.org/10.1016/j.jcss.2019.12.006.

[39] Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *J. Comput. Syst. Sci.*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.

[40] László Lovász. Graph minor theory. *Bulletin of the American Mathematical Society*, 43(1):75–86, 2006. doi:10.1090/S0273-0979-05-01088-8.

[41] Dániel Marx. Can you beat treewidth? *Theory Comput.*, 6(1):85–112, 2010. URL: https://doi.org/10.4086/toc.2010.v006a005, doi:10.4086/TOC.2010.V006A005.

[42] Jiří Matoušek. *Geometric Discrepancy: An Illustrated Guide*, volume 18 of *Algorithms and Combinatorics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. doi:10.1007/978-3-642-03942-3.

[43] Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion i. decompositions. *Eur. J. Comb.*, 29(3):760–776, 2008. URL: https://doi.org/10.1016/j.ejc.2006.07.013, doi:10.1016/J.EJC.2006.07.013.

[44] Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion II. algorithmic aspects. *Eur. J. Comb.*, 29(3):777–791, 2008. URL: https://doi.org/10.1016/j.ejc.2006.07.014, doi:10.1016/J.EJC.2006.07.014.

[45] Jaroslav Nešetřil and Patrice Ossona de Mendez. First order properties on nowhere dense structures. *J. Symb. Log.*, 75(3):868–887, 2010. URL: https://doi.org/10.2178/jsl/1278682204, doi:10.2178/JSL/1278682204.

[46] Jaroslav Nešetřil and Patrice Ossona de Mendez. On nowhere dense graphs. *Eur. J. Comb.*, 32(4):600–617, 2011. URL: https://doi.org/10.1016/j.ejc.2011.01.006, doi:10.1016/J.EJC.2011.01.006.

[47] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995. URL: https://www.sciencedirect.com/science/article/pii/S0022000085710781, doi:https://doi.org/10.1006/jcss.1995.1078.

[48] Szymon Toruńczyk. Flip-width: Cops and robber on dense graphs. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 663–700. IEEE, 2023. doi:10.1109/FOCS57990.2023.00045.

[49] Emo Welzl. Partition trees for triangle counting and other range searching problems. In Herbert Edelsbrunner, editor, *Proceedings of the Fourth Annual Symposium on Computational Geometry, Urbana-Champaign, IL, USA, June 6-8, 1988*, pages 23–33. ACM, 1988. doi:10.1145/73393.73397.

A Neighborhood complexity and versatile symmetric difference

We give a shorter (albeit non-algorithmic) proof for the inclusion of class families that can already be deduced in Lemma 26. In Theorem 33, we derive versatile symmetric difference from VC density thanks to the following fundamental result of Haussler.

Theorem 32 (Haussler's packing lemma [32])**.** *Let $\mathcal{F}$ be a set system of VC dimension d_0 with $|\mathcal{F}| > 1$ on a finite domain D, and let $c, \rho \geqslant 1$ be real numbers such that $\pi_{\mathcal{F}}(m) \leqslant c \cdot m^{\rho}$ for every positive integer m. Then $d_0 \leqslant \mathcal{O}(\log c + \rho \log \rho)$, and there are distinct $F, F' \in \mathcal{F}$ such that*

$$|F \triangle F'| \leqslant d_0 \cdot c^{1/\rho} \cdot \frac{|D|}{|\mathcal{F}|^{1/\rho}} = \mathcal{O}_{c,\rho}\left(\frac{|D|}{|\mathcal{F}|^{1/\rho}}\right).$$

Proof. Since $|\mathcal{F}| > 1$, we have $d_0 \geqslant 1$. Hence,

$$2^{d_0} \leqslant \pi_{\mathcal{F}}(d_0) \leqslant c \cdot d_0^{\rho},$$

which implies that $d_0 \leqslant \mathcal{O}(\log c + \rho \log \rho)$.

Let

$$\delta := \min\{|F \triangle F'| : F, F' \in \mathcal{F}, F \neq F'\}.$$

By Theorem 1 of [32], more precisely by the proof[2] of [42, Lemma 5.14],

$$|\mathcal{F}| \leqslant c \cdot \left(\frac{d_0 |D|}{\delta}\right)^{\rho}.$$

Rearranging gives

$$\delta \leqslant d_0 \cdot c^{1/\rho} \cdot \frac{|D|}{|\mathcal{F}|^{1/\rho}},$$

as required. □

We can now prove the main result of this section. Note that a stronger algorithmic result is proved in Lemma 26.

Theorem 33. *For every fixed pair of real numbers $c, \rho \geqslant 1$, there is a constant $d := d(c, \rho)$ such that every n-vertex graph G with (c, ρ)-polynomial neighborhood complexity contains at least $\lfloor n/8 \rfloor$ disjoint pairs of vertices u, v satisfying $sd_G(u, v) \leqslant dn^{1-1/\rho}$.*

Consequently, every hereditary graph class of VC density ρ has versatile symmetric difference of exponent $1 - 1/\rho$. In particular, every class of linear neighborhood complexity has bounded versatile symmetric difference.

Proof. Fix an n-vertex graph G with (c, ρ)-polynomial neighborhood complexity. Let $U \subseteq V(G)$ be a largest subset of vertices without twins in G, that is, $N_G(u) \neq N_G(u')$ for any distinct $u, u' \in U$. For each $u \in U$, let $C(u)$ be the set of vertices in $V(G)$ that are twins of u. Then $\{C(u) : u \in U\}$ forms a partition of $V(G)$. We split the argument depending on whether $|U| \leqslant n/2$.

Case $|U| \leqslant n/2$: For every $u \in U$, pairing vertices arbitrarily inside $C(u)$ gives $\lfloor |C(u)|/2 \rfloor$ disjoint pairs of twins. Thus, the total number of pairs is at least

$$\sum_{u \in U} \left\lfloor \frac{|C(u)|}{2} \right\rfloor \geqslant \frac{1}{2} \sum_{u \in U} (|C(u)| - 1) = \frac{n - |U|}{2} \geqslant \frac{n}{4}.$$

[2]The constant C_2 on page 158 of [42] is $c \cdot d_0^{\rho}$ in our notation, and the last inequality on page 159, for $\mathcal{P} = \mathcal{F}$, gives $|\mathcal{F}| = |\mathcal{P}| \leqslant c \cdot (d_0 |D|/\delta)^{\rho}$.

In particular, this gives at least $\lfloor n/8 \rfloor$ pairs satisfying the claimed bound.

Case $|U| > n/2$: Consider the set system of neighborhoods

$$\mathcal{F}_1 := \{N_G(u) \mid u \in U\}$$

over the domain $V(G)$. Its members are distinct and $|\mathcal{F}_1| = |U| > n/2$. Every subsystem $\mathcal{F}' \subseteq \mathcal{F}_1$ satisfies

$$\pi_{\mathcal{F}'}(m) \leqslant \pi_G(m) \leqslant c \cdot m^\rho$$

for every positive integer m. By Theorem 32, there is a constant $k := k(c, \rho)$ such that every subsystem $\mathcal{F}'$ with at least two members contains distinct F, F' with

$$|F \triangle F'| \leqslant k \cdot \frac{n}{|\mathcal{F}'|^{1/\rho}}.$$

The neighborhood $N_G(u)$ will be referred to as F^u when considered as a member of the set system. Starting with $\mathcal{F}_1$, repeat $\lfloor n/8 \rfloor$ times: choose such a pair F^{u_i}, F^{v_i} and remove both sets to obtain $\mathcal{F}_{i+1}$. Before every choice, fewer than $n/4$ sets have been removed, so the current subsystem has more than $n/4$ members and the preceding application of Theorem 32 applies. The corresponding vertex pairs $\{u_i, v_i\}$ are disjoint, and every chosen pair satisfies

$$|N_G(u_i) \triangle N_G(v_i)| \leqslant k \cdot \frac{n}{|\mathcal{F}_i|^{1/\rho}} < k \cdot 4^{1/\rho} n^{1-1/\rho}.$$

Taking $K := k \cdot 4^{1/\rho}$ proves the first assertion. For the consequence, if $\mathcal{C}$ has VC density ρ, choose a constant c witnessing this fact and apply the first assertion to every graph in $\mathcal{C}$. With $d := \max(8, \lceil K \rceil)$, each n-vertex graph in $\mathcal{C}$ has at least $\lfloor n/d \rfloor$ disjoint pairs with symmetric difference at most $dn^{1-1/\rho}$. Thus, $\mathcal{C}$ has versatile symmetric difference of exponent $1 - 1/\rho$. □