Fully Polynomial-Time Algorithms Parameterized by Vertex Integrity Using Fast Matrix Multiplication

Matthias Bentert ✉
University of Bergen, Norway

Klaus Heeger ✉
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Tomohiro Koana ✉
Algorithmics and Computational Complexity, Technische Universität Berlin, Germany

Abstract

We study the computational complexity of several polynomial-time-solvable graph problems parameterized by *vertex integrity*, a measure of a graph's vulnerability to vertex removal in terms of connectivity. Vertex integrity is the smallest number ι such that there is a set S of $\iota' \leq \iota$ vertices such that every connected component of $G - S$ contains at most $\iota - \iota'$ vertices. It is known that the vertex integrity lies between the well-studied parameters *vertex cover number* and *tree-depth*. Our work follows similar studies for vertex cover number [Alon and Yuster, ESA 2007] and tree-depth [Iwata, Ogasawara, and Ohsaka, STACS 2018].

Alon and Yuster designed algorithms for graphs with small vertex cover number using fast matrix multiplications. We demonstrate that fast matrix multiplication can also be effectively used when parameterizing by vertex integrity ι by developing efficient algorithms for problems including an $O(\iota^{\omega-1}n)$-time algorithm for Maximum Matching and an $O(\iota^{(\omega-1)/2}n^2) \subseteq O(\iota^{0.687}n^2)$-time algorithm for All-Pairs Shortest Paths. These algorithms can be faster than previous algorithms parameterized by tree-depth, for which fast matrix multiplication is not known to be effective.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Matchings and factors; Mathematics of computing → Graph algorithms; Theory of computation → Shortest paths

Keywords and phrases FPT in P, Algebraic Algorithms, Adaptive Algorithms, Subgraph Detection, Matching, APSP

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.16

Funding *Matthias Bentert*: Supported by the European Research Council (ERC) project LOPRE (819416) under the Horizon 2020 research and innovation program.
Klaus Heeger: Supported by Deutsche Forschungsgemeinschaft (DFG) project NI 369/16.
Tomohiro Koana: Supported by the DFG project DiPa (NI 369/21).

Acknowledgements We thank Vincent Borko for fruitful discussions regarding the results for finding small induced subgraphs and anonymous reviewers for their constructive feedback which, in particular, helped improving the running time of our algorithm for All-Pairs Shortest Paths.

1 Introduction

Parameterized complexity provides a powerful framework for studying NP-hard problems. The main idea behind parameterized algorithms is to analyze the running time in terms of the input size $|I|$ as well as a parameter k, some measure of the input instance. A problem is *fixed-parameter tractable* or *FPT* for short, if it admits an *FPT algorithm*, an algorithm running in time $f(k) \cdot |I|^{O(1)}$ time, where f is a function solely depending on k. In the past decade, a line of research dubbed "FPT in P" has emerged, where the goal is a more

refined parameterized analysis of polynomial-time-solvable problems [1, 5, 7, 13, 20, 25, 27]. Although the function f usually has to be at least exponential when working with NP-hard problems, this is not true for problems in P. FPT algorithms where f is a polynomial function are called *fully polynomial-time algorithms* and are of course desirable.

We study graph problems in this work. Let n and m be the number of vertices and edges, respectively. Also, let vc be the vertex cover number and td be the tree-depth (see Section 2 for definitions). Alon and Yuster [2] demonstrated that fast matrix multiplication can be used effectively for graphs with a (not necessarily small) vertex cover, developing algorithms for Maximum Matching and All-Pairs Shortest Paths (APSP) that run in $O(n^\omega)$ time (where $\omega < 2.372$ is the matrix multiplication exponent) even when $\mathrm{vc} = \Theta(n)$. More recently, Iwata et al. [26] proposed a divide-and-conquer framework in the design of fully polynomial-time algorithms parameterized by tree-depth. For instance, they showed that Maximum Matching can be solved in $O(m \cdot \mathrm{td})$ time.

In this work, we consider the parameter *vertex integrity*, a parameter that lies between vertex cover number and tree-depth. The vertex integrity ι of a graph G is the smallest integer such that G contains a set S of $\iota' \leq \iota$ vertices whose deletion results in a graph whose connected components each have at most $\iota - \iota'$ vertices. Many problems can be solved in $O(nm)$ time and thus in $O(\iota n^2)$ time, since $m \in O(\iota n)$. As the relation $\mathrm{td} \leq \iota \leq \mathrm{vc}+1$ holds for any graph, an algorithm that runs in $O(m \cdot \mathrm{td})$ time (e.g., for Maximum Matching) also runs in time $O(\iota^2 n)$. These bounds become $O(n^3)$ when $\iota = \Theta(n)$. However, many problems can be solved in faster $O(n^\omega)$ time using fast matrix multiplication. We aim to close this gap by developing fully polynomial-time algorithms that run in $O(n^\omega)$ time even when $\iota = \Theta(n)$. Such algorithms are called *adaptive*, and are optimal unless there is an (unparameterized) algorithm that runs faster than $O(n^\omega)$ time. For many problems, the discovery of such an algorithm would be a breakthrough, given that these $O(n^\omega)$-time algorithms were developed decades ago and have not been improved since.

Our approach. Before describing our results, let us briefly discuss our approach (see Section 2 for details). Let S be a *k-separator*, a vertex set of size at most $k' \leq k$ such that each connected component of $G - S$ has size $k - k'$. Throughout the paper, we will make the assumption that a k-separator is given as input. We remark that our algorithms do not require an (optimal) ι-separator to compute the correct solution. However, the running times of our algorithm will depend on k and to achieve the claimed running times, we require a k-separator with $k \in O(\iota)$. Although $G - S$ may have $O(n)$ connected components, we may assume that there are $\Theta(n/k)$ "components" (which are not necessarily connected; see Section 2 for details), each with $O(k)$ vertices. For every component C, we can use the $O(n^\omega)$-time algorithm to solve the instance on $G[C]$ or $G[S \cup C]$, which takes $O(k^\omega \cdot n/k) = O(k^{\omega-1} n)$ time. The next step is to combine solutions for $O(n/k)$ instances, which varies depending on the problem. For instance, this is trivial for the problem of finding a triangle, as a triangle must be contained in $S \cup C$ for some component C. For other problems, e.g., finding a maximum matching, this step requires a more sophisticated approach.

Main results. There are three main results in this work.

The first result concerns the problem of finding an induced copy of a graph H. Vassilevska Williams et al. [38] gave an $O(n^\omega)$-time algorithm that finds an induced copy of H when H is a graph on four vertices that is not a clique K_4 or its complement $\overline{K_4}$. Their randomized algorithm is based on computing the number of induced copies of H modulo some integer q which they show to be computable from A^2 in linear time, where A is the adjacency matrix. We observe that the "essential" part of A^2 can be computed in $O(\iota^{\omega-1} n)$ time, leading to $O(\iota^{\omega-1} n)$-time algorithms.

Secondly, we develop an $O(\iota^{\omega-1}n)$-time algorithm for finding a maximum matching. We start by showing that whether a graph contains a perfect matching can be determined in $O(\iota^{\omega-1}n)$ time. Tutte [36] observed that the Tutte matrix is nonsingular if and only if the graph has a perfect matching. By the Schwartz-Zippel lemma, we can test its nonsingularity in randomized $O(n^\omega)$ time. We can thus test whether each component in $G-S$ has a perfect matching in $O(\iota^{\omega-1}n)$ time. However, there might be a vertex in S that must be matched to a vertex in $G-S$. To handle these cases, we use *Schur complements*. The task of finding a maximum matching is more intricate. Lovász [30] generalized Tutte's observation by stating that the rank of the Tutte matrix (which can be computed in randomized $O(n^\omega)$ time) equals twice the size of a maximum matching. It was only decades later that $O(n^\omega)$-time algorithms for finding one were discovered. Mucha and Sankowski [31] and Harvey [24] gave such algorithms. We show how to adapt the latter to obtain an $O(\iota^{\omega-1}n)$-time algorithm for finding a maximum matching.

Lastly, we study APSP on unweighted graphs. Seidel [34] showed that APSP can be solved in $O(n^\omega \log n)$ time. Alon and Yuster [2] later developed an algorithm that runs in $O(\mathrm{vc}^{\omega-2}\, n^2)$ time (they actually provide a stronger bound using rectangular matrix multiplication). We show that APSP can be solved in $O(\iota^{\omega-2}n^2)$ time when the graph has constant diameter. We were not able to obtain an adaptive algorithm in general, but we give an $O(\iota^{(\omega-1)/2}n^2) \subseteq O(\iota^{0.687}n^2)$-time algorithm. When parameterizing by vc, we can effectively replace every vertex not in the vertex cover with edges of weight two connecting their neighbors. Thus, the $O(Wn^\omega)$-time algorithm [17, 35] for weighted APSP, where W is the maximum weight, finds all pairwise distances between vertices in the vertex cover in $O(\mathrm{vc}^\omega)$ time. For vertex integrity, we show how to replace every component with edges of weight $O(\iota)$. To compute distances between pairs of vertices with at least one vertex not in the k-separator, we use the known subcubic-time algorithm for computing min-plus matrix multiplication for *bounded-difference matrices*, matrices in which the difference of two adjacent entries in a row is constant [10].

Previous work on vertex integrity. The notion of vertex integrity was introduced by Barefoot et al. [3]. The vertex integrity ι can be much smaller than n, e.g., it is known that $\iota \in O(n^{2/3})$ on K_h-minor free graphs [4]. The Vertex Integrity problem, i.e., computing an ι-separator is NP-hard. A straightforward branching algorithm solves Vertex Integrity in $O(\iota^\iota \cdot n)$ time (see [15]). A greedy algorithm can find an $O(\iota^2)$-separator in linear time. There is also a polynomial-time algorithm that can find an $O(\iota \log \iota)$-separator [29]. FPT algorithms parameterized by vertex integrity gained increased attention recently [6, 15, 16, 22, 28]. In particular, see Gima et al. [22] for an extensive list of problems that are W[1]-hard for tree-depth but become FPT when parameterized by vertex integrity.

2 Preliminaries

We use standard notation from graph theory. Unless stated otherwise, all appearing graphs are undirected. Further, V denotes the set of vertices in the graph, E its set of edges, n its number of vertices, and m its number of edges. We denote an edge between two vertices u and v by uv. A *walk* of length ℓ is a sequence $v_1, \ldots, v_\ell$ of (not necessarily distinct) vertices such that $v_iv_{i+1} \in E$ for all $i \in [\ell-1]$, where $[j] := \{1, \ldots, j\}$ for any integer j. A walk whose vertices are all pairwise distinct is a *path*. The *adjacency matrix* of G is the $V \times V$-matrix A with $A[u,v] = 1$ if and only if $uv \in E$, and $A[u,v] = 0$ otherwise (where $A[u,v]$ is the entry of A indexed by u and v).

Graph parameters. For a graph G, the *vertex integrity* is the smallest integer ι such that G contains a set S (called *ι-separator*) of $\iota' \leq \iota$ vertices whose deletion results in a graph whose connected components each have at most $\iota - \iota'$ vertices. The *vertex cover number* vc is the smallest cardinality of a vertex cover, a set that contains at least one endpoint of every edge. The *tree-depth* td is the smallest depth of a rooted forest F with vertex set V such that G can be *embedded* in F, i.e., for every edge xy in G, x is an ancestor of y or vice versa. The *feedback vertex number* is the smallest cardinality of a feedback vertex set, a set that contains at least one vertex of every cycle.

Decomposition. Here, we describe the decomposition with respect to a k-separator, which will be used throughout the paper. Let S be a k-separator. Typically in our algorithms, we spend $O(k^\omega)$ time for every connected component in $G - S$. Since $G - S$ may have $\Omega(n)$ connected components, this would result in a running time of $O(k^\omega n)$, which is often worse than a more straightforward algorithm. Thus, we will do the following to bound the number of "components" by $O(n/k)$: Basically, we put together some connected components C and construct a collection $\mathcal{T}$ of sets, each (except for possibly the last one) containing between k and $2k-1$ vertices. More precisely, we start with $\mathcal{T} = \emptyset$ and process the connected components of $G - S$ one by one as follows. If every set $T \in \mathcal{T}$ has at least k vertices, then add $\{C\}$ to $\mathcal{T}$, and otherwise replace the set $T \in \mathcal{T}$ with $|T| < k$ by $T \cup C$. Since every connected component C has at most k vertices, every set $T \in \mathcal{T}$ (except for possibly the last set which may be smaller) contains between k and $2k-1$ vertices. Let $\mathcal{T} = \{T_1, \ldots, T_\nu\}$. It is easy to see that $\nu \leq n/k + 1$. In our algorithms, we will always assume that the decomposition $(S; T_1, \ldots, T_\nu)$ of V is given. Note that given a k-separator, the decomposition can be computed in linear time.

Basic operations in matrix multiplication time. For $n \times n$-matrices A, B, one can compute the following in $O(n^\omega)$ time: (i) the product AB, (ii) the determinant $\det A$, (iii) the inverse A^{-1}, and (iv) a row/column basis of A (see e.g., [9]). More generally, for a $k \times n$-matrix A and an $n \times k$-matrix B, one can compute the product AB in $O(k^{\omega-1}n)$ time by dividing A and B into n/k blocks of size $k \times k$. The rank of A can be computed using Gaussian elimination using $O(k^{\omega-1}n)$ arithmetic operations [8]. Throughout the paper, we will use a word RAM model with word size $O(\log n)$. If $\mathbb{F}$ is a field of size $\mathrm{poly}(n)$, we will assume that addition and multiplication take $O(1)$ time.

Matrices and Matchings. For a subset X of rows and a subset Y of columns, we denote by $A[X, Y]$ the restriction of the matrix A to rows X and columns Y. For a set X of rows (or columns), we will use the shorthand $A[X]$ for $A[X, X]$. The i-th power of the adjacency matrix A correspond to the number of walks of length i, i.e., $A^i[u, v]$ equals the number of u-v-walks of length i in G.

Note that for any graph with a k-separator S and decomposition $(S; T_1, \ldots, T_\nu)$ it holds that there is no edge between a vertex in T_i and a vertex in T_j for any $i \neq j$. We can therefore represent the adjacency matrix A of the graph as follows.

$$A = \begin{array}{c} \\ S \\ \bar{S} \end{array} \begin{array}{c} \begin{array}{cc} S & \bar{S} \end{array} \\ \begin{bmatrix} \gamma & \beta \\ \beta^T & \alpha \end{bmatrix} \end{array}, \text{ for } \alpha = \begin{array}{c} \\ T_1 \\ \vdots \\ T_\nu \end{array} \begin{array}{c} \begin{array}{ccc} T_1 & \cdots & T_\nu \end{array} \\ \begin{bmatrix} \alpha_1 & \cdots & O \\ \vdots & \ddots & \vdots \\ O & \cdots & \alpha_\nu \end{bmatrix} \end{array} \text{ with } \beta = \begin{array}{c} \\ S \end{array} \begin{array}{c} \begin{array}{ccc} T_1 & \cdots & T_\nu \end{array} \\ \begin{bmatrix} \beta_1 & \cdots & \beta_\nu \end{bmatrix} \end{array}. \quad (1)$$

Here, $\bar{S} = T_1 \cup \cdots \cup T_\nu$ and the matrices $\alpha_i, \beta_i, \gamma$ all have size $O(k) \times O(k)$. Many of our algorithms will use this representation and exploit the sparseness when computing e.g., matrix multiplications and determinants.

We denote the (unique) finite field with 2^q many elements by $\mathrm{GF}(2^q)$. Note that this field has characteristic 2, i.e., $x + x = 0$ for every $x \in \mathrm{GF}(2^q)$. For a graph $G = (V, E)$, the Tutte matrix (also known as the skew adjacency matrix) A whose rows and columns are indexed by $V = \{v_1, \ldots, v_n\}$ is defined by

$$A[u,v] = \begin{cases} +x_{uv} & \text{if } u = v_i, v = v_j \text{ with } i < j \text{ and } uv \in E(G) \\ -x_{uv} & \text{if } u = v_i, v = v_j \text{ with } j < i \text{ and } uv \in E(G) \\ 0 & \text{otherwise,} \end{cases}$$

where x_{uv} is a variable associated with the edge uv. The Tutte matrix A is *skew-symmetric*, i.e., $A = -A^T$. The Pfaffian of a skew-symmetric matrix A indexed by V is defined as

$$\mathrm{pf}(A) = \sum_{M \in \mathcal{M}} \sigma_M \prod_{uv \in M} A[u,v],$$

where $\mathcal{M}$ is the set of all perfect matchings of $(V, \binom{V}{2})$ and $\sigma_M \in \{\pm 1\}$ is the sign of M. We will assume that the field has characteristic 2 (implying $-1 = 1$), and thus the precise definition of σ_M is not important for us. (This assumption is not essential to our algorithm but it will simplify the notation.)

The following are well-known facts about skew-symmetric matrices (see e.g., [23, 32]).

▶ **Lemma 1.** *For a skew-symmetric matrix A, we have $\det A = \mathrm{pf}(A)^2$.*

In particular, a skew-symmetric matrix A is nonsingular if and only if $\mathrm{pf}(A) \neq 0$.

▶ **Lemma 2.** *For a skew-symmetric matrix A, if X is a row (or column) basis, then $A[X]$ is nonsingular.*

The next is immediate from the definition of Pfaffians.

▶ **Lemma 3** (row expansion). *For a skew-symmetric matrix A indexed by V and $v \in V$ over a field of characteristic 2, we have $\mathrm{pf}(A) = \sum_{v' \in V \setminus \{v\}} A[v,v'] \cdot \mathrm{pf}(\widehat{A}_{v,v'})$, where $\widehat{A}_{v,v'}$ is the matrix where the rows and columns indexed by v and v' are deleted.*

Proofs of statements marked with $\star$ are omitted from the conference version and can be found in a full version of this paper.

3 Finding Subgraphs

In this section, we develop adaptive algorithms for finding four-vertex subgraphs. There are eleven non-isomorphic graphs with 4 vertices: the clique on four vertices (K_4) and its complement ($\overline{K_4}$), the diamond ($K_4 - e$) and its complement ($\overline{K_4 - e}$), the claw ($K_{1,3}$) and its complement ($\overline{K_{1,3}}$), the paw ($K_{1,3} + e$) and its complement ($\overline{K_{1,3} + e}$), the cycle on four vertices (C_4) and its complement ($\overline{C_4}$), and the path on four vertices (P_4) (which is its own complement). Note that $+e$ and $-e$ indicate the insertion of an edge or the deletion of any edge, respectively. A linear-time algorithm is known for detecting whether the input graph contains an induced P_4 [12]. For K_4 and $\overline{K_4}$, the currently fastest algorithm runs in $O(n^{3.257})$ [18, 21] and for all other graphs the best known algorithm is by Williams et al. [38] and runs in $O(n^\omega)$ time. Their approach can be summarized as follows.

- Let G be an undirected graph and let A be its adjacency matrix. Let $H = (V', E')$ be a four-vertex graph that is none of K_4, $\overline{K_4}$, C_4, $\overline{C_4}$. There is an integer $2 \leq q_H \leq 6$ such that if we can compute $A^2[u, v]$ for every edge $uv \in E(G)$ in time t, then we can compute the number of induced copies of H in G modulo q_H in $O(n + m + t)$ time. See [38, Lemma 4.1] for details. (Some equations provided in [38] require $A^3[v]$ for every $v \in V$. However, this can be computed in $O(m)$ time if $A^2[u, v]$ is given for every edge uv.)
- Let $q \geq 2$ be an integer and let G, H be two undirected graphs. Let G' be an induced subgraph of G obtained by independently deleting each vertex with probability $1/2$. If G contains H as an induced subgraph, then the number of induced copies of H in G' modulo q is not 0 with probability at least $2^{-|V(H)|}$. (For our applications, $|V(H)| = 4$, so this probability is at least $1/16$.)

We show that when a k-separator is given, one can test in $O(k^{\omega-1}n)$ time whether there is an induced copy of H for each four-vertex graph H except for K_4 and $\overline{K_4}$. We start with all graphs except for K_4, $\overline{K_4}$, C_4, $\overline{C_4}$. Using the framework by Williams et al. [38], it suffices to show how to compute $A^2[u, v]$ for every edge $uv \in E(G)$. Clearly, it requires $\Omega(n^2)$ time to compute the square A^2. Our key observation is that the relevant part of A^2 can be computed in $O(k^{\omega-1} \cdot n)$ time. (Incidentally, A^2 can be computed in $O(k^{\omega-2}n^2)$ time; see Lemma 18.)

▶ **Lemma 4.** *Given a graph G and a k-separator S, we can compute $A^2[u, v]$ for every edge $uv \in E(G)$ in $O(k^{\omega-1}n)$ time.*

Proof. We use the decomposition $(S; T_1, \ldots, T_\nu)$ described in Section 2 and suppose that the adjacency matrix A has the form given in Equation (1). Note that

$$A^2 = \begin{array}{c} \\ S \\ T_1 \\ \vdots \\ T_\nu \end{array} \begin{array}{c} \begin{array}{cccc} S & T_1 & \cdots & T_\nu \end{array} \\ \begin{bmatrix} \zeta & \eta_1 & \cdots & \eta_\nu \\ \eta_1^T & \delta_1 & \cdots & - \\ \vdots & \vdots & \ddots & \vdots \\ \eta_\nu^T & - & \cdots & \delta_\nu \end{bmatrix} \end{array}, \text{ where } \begin{array}{l} \zeta = \gamma^2 + \beta\beta^T, \\ \eta_i = \gamma\beta_i + \beta_i\alpha_i, \text{ and} \\ \delta_i = \beta_i^T\beta_i + \alpha_i^2. \end{array}$$

Note that computing ζ takes $O(\nu \cdot k^\omega) = O(k^{\omega-1}n)$ time and computing each of the $O(\nu)$ submatrices η_i or δ_i takes $O(k^\omega)$ time. The $-$ represents pairs where the corresponding vertices belong to different T_i and are therefore non-adjacent. We thus do not need to compute these values. Thus, we can compute all relevant values in $O(k^{\omega-1}n)$ time. ◀

By Lemma 4, an induced copy of $H \notin \{K_4, \overline{K_4}, C_4, \overline{C_4}\}$ can be detected in $O(\iota^{\omega-1}n)$ time. We show next that C_4 and $\overline{C_4}$ can also be detected in $O(\iota^{\omega-1}n)$ time.

▶ **Proposition 5** (⋆). *Given a graph G, a k-separator, and a graph $H \in \{C_4, \overline{C_4}\}$, we can test whether G contains H as an induced subgraph in $O(k^{\omega-1}n)$ time.*

Thus, we obtain the following.

▶ **Proposition 6.** *Given a graph G, a k-separator, and a graph H with four vertices that is not K_4 or $\overline{K_4}$, we can test whether G contains an induced copy of H in $O(k^{\omega-1}n)$ time.*

We can also find an induced copy with a constant overhead using a standard self-reduction.

▶ **Corollary 7** (⋆). *Given a graph, a k-separator, and a graph H with four vertices that is not K_4 or $\overline{K_4}$, we can find an induced copy of H in $O(k^{\omega-1}n)$ time if it exists.*

Finally, let us also remark on the detection of cliques K_ℓ and independent sets $\overline{K_\ell}$. Let $t_\ell(n)$ be the time complexity of finding K_ℓ (or $\overline{K_\ell}$) on an n-vertex graph. (It is known, for instance, that $t_4(n) \in O(n^{3.257})$ [18] using fast rectangular matrix multiplication [21].) Since any K_ℓ must fully be contained in $G[S \cup T_i]$ for some i, it can be detected in $O(t_\ell(\iota)/\iota \cdot n)$ time. For the detection of $\overline{K_\ell}$, note that if $n/\iota \geq \ell$, then there is an independent set of size ℓ. Thus, we may assume that $n \leq \iota\ell$. For constant ℓ, we can thus find $\overline{K_\ell}$ in $O(t_\ell(\iota))$ time.

In the full version of the paper, we also study the problem of finding short(est) cycles. In particular, we show that *girth*, i.e., the length of a shortest cycle, can be computed in $O(\iota^{\omega-1}n)$ time.

4 Matching

As mentioned in the introduction, Lovász [30] showed that the cardinality of a maximum matching can be determined in randomized $O(n^\omega)$ time. Several decades later, two algorithms have been developed to find a maximum matching [24, 31]. In this section, we show the following.

▶ **Theorem 8.** *Given a graph and a k-separator, we can find a maximum matching in randomized $O(k^{\omega-1}n)$ time.*

In the full version of the paper, we show that the existence of a perfect matching can be checked in $O(k^{\omega-1}n)$ time, where k is the feedback vertex number of the input graph, that is, the minimum number of vertices to delete in order to turn the graph into a forest.

Tutte [36] showed that G has a perfect matching if and only if its symbolic Tutte matrix A is nonsingular. Lovász [30] later showed that the rank of A equals twice the size of maximum matching. To avoid computation over a multivariate polynomial ring, we will assume that each variable x_{uv} is instantiated with an element chosen from a finite field $\mathbb{F}$ uniformly at random. (We will assume that $|\mathbb{F}| = \mathrm{GF}(2^{c\lceil \log n \rceil})$ for a sufficiently large constant $c > 0$.) Since the determinant of the symbolic Tutte matrix has degree at most n, by the Schwartz–Zippel lemma [33, 39] (which states that a non-zero polynomial of total degree d over a finite field $\mathbb{F}$ is, when evaluated at a uniformly randomly coordinate, non-zero with probability at least $1 - d/|\mathbb{F}|$), if G has a perfect matching and $\mathbb{F}$ is of size at least δn, then A is nonsingular with probability at least $1 - 1/\delta$.

Let A be a block matrix $A = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}$. If α is nonsingular, then the determinant of A is $\det(A) = \det(\alpha) \cdot \det(C)$, where $C = \delta - \gamma\alpha^{-1}\beta$ is the Schur complement (see e.g., [32]). Thus, assuming that α is nonsingular, A is nonsingular if and only if $\delta - \gamma\alpha^{-1}\beta$ is nonsingular. A simple application of the Schur complement yields the following.

▶ **Lemma 9.** *Consider a matrix A of the following form. Then, provided that α is nonsingular, A is nonsingular if and only if the following matrix A' is nonsingular.*

$$A = \begin{bmatrix} \alpha & \beta & 0 \\ -\beta^T & \gamma & \zeta \\ 0 & -\zeta^T & \eta \end{bmatrix} \qquad A' = \begin{bmatrix} \gamma & \zeta \\ -\zeta^T & \eta \end{bmatrix} - \begin{bmatrix} -\beta^T \\ 0 \end{bmatrix} \alpha^{-1} \begin{bmatrix} \beta & 0 \end{bmatrix} = \begin{bmatrix} \gamma + \beta^T\alpha^{-1}\beta & \zeta \\ -\zeta^T & \eta \end{bmatrix}$$

Harvey's algorithm. There is a randomized $O(n^\omega)$-time algorithm to find a perfect matching in a graph (if one exists) due to Harvey [24]. We describe the algorithm outline here. The main idea is to delete edges as long as at least one perfect matching remains until we are left with a graph in which every vertex has exactly one neighbor. Here, deleting an edge

corresponds to setting the corresponding entries in the Tutte matrix to zero. Whether an edge is *deletable*, i.e., whether there remains a perfect matching after its deletion, can be determined in $O(1)$ time after computing the inverse of the Tutte matrix in a preprocessing step. This is essentially due to the fact that for a nonsingular matrix A and column vectors u and v, matrix $A + uv^T$ is nonsingular if and only if $1 + v^T A^{-1} u \neq 0$. When an edge is deleted, the inverse needs to be updated and the Sherman-Morrison-Woodbury formula (which is not relevant for us) is employed in Harvey's algorithm. However, it takes $O(n^2)$ time when this update is implemented naively. The crux of Harvey's algorithm lies in a recursive scheme that essentially allows to perform the update in $O(1)$ time. The algorithm uses a subroutine called DeleteEdgesCrossing, which takes two disjoint vertex sets R and S as input and iteratively deletes all deletable edges uv with $u \in R$ and $v \in S$. This subroutine runs in $O(n^\omega)$ time and our algorithm will also use it.

4.1 Detecting a perfect matching

In this section, we will describe a randomized $O(k^{\omega-1}n)$-time algorithm to test whether a given graph with a k-separator S contains a perfect matching. We assume that the Tutte matrix A has the form of Equation (1) and that it is instantiated randomly from a field $\mathbb{F} = \mathrm{GF}(2^{c\lceil \log n \rceil})$. For each component T_i, we find a basis $T_i' \subseteq T_i$ of $A[T_i]$ in $O(k^\omega)$ time. Note that $A[T_i']$ is nonsingular by Lemma 2. Let $T_i^* = T_i \setminus T_i'$, $S_i = S_{i-1} \cup T_i^*$ for $S_0 = S$, and $S^* = S_\nu$. Note that if G has a perfect matching M, then at least $|T_i^*|$ vertices of T_i are matched to S. Thus, if $\sum_{i\in[\nu]} |T_i^*| > |S|$, we can conclude that there is no perfect matching. Otherwise, we have $|S^*| \leq 2k$. Now, consider for each component T_i' the matrix

$$B_i = \begin{array}{c} \\ T_i' \\ S^* \\ V \setminus (S^* \cup \bigcup_{j\in[i-1]} T_j') \end{array} \begin{array}{c} \begin{array}{ccc} T_i' & S^* & V \setminus (S^* \cup \bigcup_{j\in[i-1]} T_j') \end{array} \\ \begin{bmatrix} \alpha_i & \beta_i & 0 \\ -\beta_i^T & \gamma_i & \zeta_i \\ 0 & -\zeta_i^T & \eta_i \end{bmatrix} \end{array},$$

where $\gamma_1 = \gamma$, $\gamma_{i+1} = \gamma_i + \beta_i^T \alpha_i^{-1} \beta_i$, and all entries except for γ_i for $i > 1$ are identical to the corresponding entries of A. Note that α_i and β_i may differ from α_i and β_i in Equation (1), but if $T_j' = T_j$ for all $j \in [\nu]$, then α_i and β_i here coincide with α_i and β_i from Equation (1). Note that the matrix B_1 coincides with the Tutte matrix A. Hence, we only need to test whether B_1 is nonsingular. We do this by iteratively computing B_{i+1} from B_i in $O(k^\omega)$ time. For notational convenience, we will assume that there is an empty component $T_{\nu+1}'$. Since $\alpha_i = A[T_i']$ is nonsingular, by Lemma 9 the matrix B_i is nonsingular if and only if B_{i+1} is nonsingular. The nonsingularity of $B_{\nu+1} = \gamma_{\nu+1}$ can be tested in $O(k^\omega)$ time since it has size $O(k) \times O(k)$. Note that our algorithm takes $O(k^\omega)$ time for each $i \in [\nu + 1]$ and thus $O(\nu k^\omega) = O(k^{\omega-1}n)$ time overall. If A is nonsingular, then our algorithm correctly concludes that there is a perfect matching. By the Schwartz–Zippel lemma, we know that if G admits a perfect matching, then the probability that A is nonsingular is at least $1 - n/|\mathbb{F}| \geq 1/2$. This leads to the following result.

▶ **Proposition 10.** *Given a graph and a k-separator, we can test whether it has a perfect matching in randomized $O(k^{\omega-1}n)$ time.*

In the full version of the paper, we show that whether there is a perfect matching can be checked in $O(k^{\omega-1}n)$ time, where k is the feedback vertex number. We use a similar approach based on the Schur complement. The major difference is that computing $(A[V \setminus S])^{-1}$ would

require $\Omega(n^2)$ time (A is the Tutte matrix and S is a feedback vertex set). Instead, we compute $(A[V \setminus S])^{-1}A[S, V \setminus S]$ via dynamic programming in $O(kn)$ time. To that end, we first show that for entries with both coordinates in S in the Tutte matrix, it suffices to use values in $\{0, 1\}$ instead of variables. This allows us to give a simple characterization of the entries of $(A[V \setminus S])^{-1}$ in terms of alternating paths. Our dynamic program uses a recurrence relation based on this.

▶ **Proposition 11** (⋆). *Deciding whether a given graph G contains a perfect matching can be done in randomized $O(k^{\omega-1}n)$ time, where k is the feedback vertex number of G.*

4.2 Finding a perfect matching

We next give an algorithm to find a perfect matching (if one exists). We will use the same notation as before and we will again assume that α_i is nonsingular for every $i \in [\nu]$. Note that if the submatrix γ_1 is also nonsingular, then it is easy to find a perfect matching since the corresponding submatrices $\gamma_1 = A[S^*]$ and $\alpha_i = A[T_i']$ are all nonsingular and therefore there exists a perfect matching in $G[S^*]$ and $G[T_i']$ for every $i \in [\nu]$. We can find these using one of the randomized $O(n^\omega)$-time algorithms [24, 31]. This procedure takes $O(k^\omega\nu) = O(k^{\omega-1}n)$ time. However, in general, we cannot assume that γ_1 is nonsingular.

To deal with the case that γ_1 is singular, we use Harvey's aforementioned algorithm. Our algorithm first computes γ_i for each $i \in [\nu + 1]$. We then proceed inductively in decreasing order from $i = \nu+1$ to $i = 1$. Our algorithm finds a set $\widetilde{S}_i \subseteq \widetilde{S}_{i+1} \subseteq S^*$ (we set $\widetilde{S}_{\nu+1} = S^*$ for notational convenience) and a matching M_i which saturates every vertex in $(\widetilde{S}_{i+1} \setminus \widetilde{S}_i) \cup T_i'$. The matching M_i is the union of two matchings M_i' and M_i'', where M_i' is a perfect matching between $\widetilde{S}_{i+1} \setminus \widetilde{S}_i$ and $T_i' \setminus T_i''$ for $T_i'' \subseteq T_i'$ and M_i'' is a perfect matching in $G[T_i'']$ (see Lemmas 13 and 14).

We will maintain the invariant that $\gamma_i[\widetilde{S}_i]$ is nonsingular. For $i = \nu + 1$, we have that $\gamma_{\nu+1}[\widetilde{S}_{\nu+1}] = B_{\nu+1}$ is nonsingular. Moreover, if this invariant is maintained, then $\gamma_1[\widetilde{S}_1]$ is nonsingular and thus $G[\widetilde{S}_1]$ contains a perfect matching that can be combined with all previous M_i to obtain a perfect matching for G. To find the set $\widetilde{S}_i$ for $i \in [\nu]$, consider the following submatrix B_i' of B_i whose rows and columns are indexed by $\widetilde{S}_{i+1} \cup T_i'$:

$$B_i' = \begin{bmatrix} \alpha_i & \beta_i[T_i', \widetilde{S}_{i+1}] \\ -\beta_i^T[\widetilde{S}_{i+1}, T_i'] & \gamma_i[\widetilde{S}_{i+1}] \end{bmatrix}.$$

Note that for a nonzero entry in B_i', if one of the coordinates is in T_i', then the corresponding edge exists in G but this is not necessarily the case if both coordinates are in $\widetilde{S}_{i+1}$. We first show that B_i' is nonsingular.

▶ **Lemma 12** (⋆). *For each $i \in [\nu]$, the matrix B_i' is nonsingular.*

Since B_i' is nonsingular, we can compute $(B_i')^{-1}$ and apply the subroutine DeleteEdges-Crossing of Harvey [24] on the bipartition $(\widetilde{S}_{i+1}, T_i')$ in $O(k^\omega)$ time. (This subroutine requires the inverse to be given.) Essentially, we "delete" (i.e., change to zero) all deletable entries with one coordinate in $\widetilde{S}_{i+1}$ and the other in T_i'. We then get a skew-symmetric matrix C_i that is identical to B_i' except that some entries in $C_i[\widetilde{S}_{i+1}, T_i']$ and $C_i[T_i', \widetilde{S}_{i+1}]$ are set to zero. Let $\widetilde{S}_i$ be those vertices $v \in \widetilde{S}_{i+1}$ such that $C_i[T_i', v]$ is a zero vector and T_i'' be those vertices $u \in T_i'$ such that $C_i[\widetilde{S}_{i+1}, v]$ is a zero vector. Each remaining nonzero entry in $C_i[\widetilde{S}_{i+1}, T_i']$ and $C_i[T_i', \widetilde{S}_{i+1}]$ is undeletable. We show that all edges corresponding to these entries form a perfect matching M_i' between $\widetilde{S}_{i+1} \setminus \widetilde{S}_i$ and $T_i' \setminus T_i''$.

▶ **Lemma 13.** *The set* $M'_i = \{uv \mid C_i[u,v] \neq 0, u \in \widetilde{S}_{i+1} \setminus \widetilde{S}_i, v \in T'_i \setminus T''_i\}$ *is a matching in* G *with high probability.*

Proof. For a vertex $u \in \widetilde{S}_{i+1} \setminus \widetilde{S}_i$, assume that there are two vertices v, v' such that $C_i[u,v] \neq 0$ and $C_i[u,v'] \neq 0$. We show that this leads to a contradiction when the variables are treated as indeterminates. Let $\widehat{C}_{iu,w}$ be the result of deleting the rows and columns indexed by u and w from C_i. By Lemma 3, we have

$$\operatorname{pf}(C_i) = C_i[u,v] \operatorname{pf}(\widehat{C}_{iu,v}) + C_i[u,v'] \operatorname{pf}(\widehat{C}_{iu,v'}) + p, \tag{2}$$

where $p = \sum_{w \in \widetilde{S}_{i+1} \cup T'_i,\, w \notin \{v,v'\}} C_i[u,w] \operatorname{pf}(\widehat{C}_{iu,w})$.

By the assumption that deleting the corresponding edge uv or uv' (i.e., setting $x_{uv} = 0$ or $x_{uv'} = 0$) results in a singular matrix (this is due to Harvey's algorithm), we have

$$C_i[u,v] \operatorname{pf}(\widehat{C}_{iu,v}) + p = 0 \text{ and } C_i[u,v'] \operatorname{pf}(\widehat{C}_{iu,v'}) + p = 0.$$

We thus obtain $C_i[u,v] \operatorname{pf}(\widehat{C}_{iu,v}) = C_i[u,v'] \operatorname{pf}(\widehat{C}_{iu,v'}) = -p$. Note that $p \neq 0$, since otherwise $\operatorname{pf}(C_i) = -p = 0$ by Equation (2), which is a contradiction because C_i is nonsingular by Harvey's algorithm. The left hand side $C_i[u,v] \operatorname{pf}(\widehat{C}_{iu,v})$ is multiplied by a variable $C_i[u,v] = x_{u,v}$ but this variable does not appear on the right hand side $C_i[u,v'] \operatorname{pf}(\widehat{C}_{u,v'})$. Thus, $Q = C_i[u,v] \operatorname{pf}(\widehat{C}_{iu,v}) - C_i[u,v'] \operatorname{pf}(\widehat{C}_{iu,v'})$ is a polynomial that is not identically zero, which is a contradiction. We thus have exactly one vertex $v \in T'_i \setminus T''_i$ such that $C_i[u,v] \neq 0$. Since Q has degree at most n, this evaluates to non-zero with high probability by the Schwartz–Zippel lemma. An analogous argument shows that for every $v \in T'_i \setminus T''_i$, there is exactly one $u \in \widetilde{S}_{i+1} \setminus \widetilde{S}_i$ such that $C_i[u,v] \neq 0$. Since every nonzero entry in $C_i[\widetilde{S}_{i+1}, T_i]$ corresponds to an edge in G, we are done. ◀

We can now show that our main invariant can be maintained with high probability.

▶ **Lemma 14** (⋆). *The matrices* $\gamma_i[\widetilde{S}_i]$ *and* $\alpha_i[T''_i]$ *are nonsingular with high probability.*

Concluding this section, we prove our main result.

▶ **Proposition 15** (⋆). *Given a graph with a perfect matching and a* k*-separator, we can find a perfect matching in* $O(k^{\omega-1}n)$ *time.*

In the full version of the paper, we prove Theorem 8, that is, we show how to find a maximum matching in $O(k^{\omega-1}n)$ time, when given a k-separator.

5 All-Pairs Shortest Paths

We study the well-known All-Pairs Shortest Paths problem (APSP) on unweighted graphs. APSP on unweighted graphs can be solved in $O(n^\omega)$-time using matrix multiplication [34]. In this section, we show an algorithm which is faster than $O(n^\omega)$ when k is sufficiently small, that is, $k \in O(n^{0.541})$.

▶ **Theorem 16.** *Given an (unweighted undirected) graph and its* k*-separator, APSP can be solved in* $O(k^{(\omega-1)/2}n^2) \subseteq O(k^{0.687}n^2)$*-time.*

APSP is closely connected to the min-plus-product. In fact, solving APSP is subcubic equivalent to computing the min-plus product of two matrices [19]. The *min-plus product* $A \star B$ of a $p \times q$-matrix A with a $q \times r$-matrices B is the $p \times r$-matrix C

with $C[i,j] = \min_{k\in[q]} A[i,k] + B[k,j]$. While it is conjectured that there is no algorithm computing the min-plus product of two $n \times n$-matrices in $O(n^{3-\epsilon})$ time for any $\epsilon > 0$, there is a subcubic-time min-plus matrix multiplication algorithm for bounded-difference matrices, i.e., matrices in which the difference of two adjacent entries in a row is small [10]. Although the distance matrix D (which contain pairwise distances) does not necessarily have bounded differences, for two adjacent vertices u and v, the difference between $D[u,w]$ and $D[u,v]$ for any $w \in V$ is at most 1. The "standard" decomposition of $G - S$ into $T_1, \dots, T_\nu$ from Section 2 is not convenient in this case, since two vertices in T_i may have distance $\Theta(n)$, e.g., when T_i is not connected. Our algorithm first modifies the given graph so that we have bounded-difference distance matrices. The main idea is that, for a k-separator S, we can join connected components from $G - S$ by adding copies of vertices in S. We will also ensure that there is a Hamiltonian path through each component (this step is essentially also used by Deng et al. [14]). This ensures that if we rearrange the rows and columns according to the Hamiltonian path, then two consecutive rows correspond to adjacent vertices and the distances between either of them and some third vertex differ by at most one.

▶ **Lemma 17** ($\star$). *Given a graph G and a k-separator S, we can compute in $O(n+m)$ time a graph G' and its $O(k)$-separator S' such that $G' - S'$ has connected components $T'_1, \dots, T'_\nu$ for $\nu' \in O(n/k)$, each of which has a Hamiltonian path H_i, and $|S'| = |T'_1| = \cdots = |T'_\nu|$. Moreover, given the distance matrix D' for G', one can compute the distance matrix D of G in $O(n^2)$ time.*

Using Lemma 17, we now show that APSP on unweighted graphs can be solved faster than $O(n^\omega)$ if the vertex integrity is sufficiently small and $\omega > 2$.

Proof of Theorem 16. If $k \geq n^{0.6}$, then $k^{(\omega-1)/2}n^2 > n^\omega$ and we apply the $O(n^\omega \log n)$-time algorithm [34]. Otherwise, $k \leq n^{0.6}$ and we use the following algorithm to solve APSP.

1. We apply Lemma 17, resulting in a graph G and sets $S, T_1, \dots, T_\nu$.
2. For every i, we solve APSP on $G[S \cup T_i]$. Let $D_i = D'_i[T_i, S]$, where D'_i is the distance matrix of $G[S \cup T_i]$.
3. We compute an edge-weighted graph G_S where the vertex set is S, the edge set is $\binom{S}{2}$, and $w(uv) = 1$ if $uv \in E(G)$, and $w(uv) = \min_i D_i(uv)$ otherwise. We solve APSP on this weighted graph and call the resulting distance matrix D_S. As we show later, $D_S[u,v]$ is the distance from u to v in G.
4. For each $i, j \in [\nu]$, we compute $D^*_i := D_i \star D_S$. and $D^*_{i,j} := D^*_i \star D_j^T$.
5. Return a symmetric matrix D^* where the upper triangular part of D^* is defined by

$$D^*[u,v] = \begin{cases} D_S[u,v] & u,v \in S, \\ D^*_i[u,v] & u \in S \text{ and } v \in T_i, \\ \min\{D_i[u,v], D^*_{i,i}[u,v]\} & u,v \in T_i \text{ for some } i \in [\nu], \\ D^*_{i,j}[u,v] & u \in T_i \text{ and } v \in T_j \text{ for } i \neq j. \end{cases}$$

First we show the correctness of the algorithm. For $u, v \in S$, Step 3 guarantees that $D^*[u,v] = D_S[u,v]$ is the distance between u and v in the weighted graph G_S. As each edge $u'v'$ in G_S corresponds to a path in $S \cup T_i$ of length $w(u'v')$ for some $i \in [\nu]$, it follows that each path in G_S corresponds to a walk in G of the same length. Thus, $D_S[u,v]$ is not smaller than a shortest u-v-path in G. On the other hand, let P be a shortest u-v-path in G. Let $u = s_1, \dots, s_\ell = v$ be the vertices of P in S (in the order they appear in P). By construction, for each $i < \ell$, there are no vertices in S between s_i and s_{i+1} in P. Hence, this subpath of P is fully contained in $G[S \cup T_j]$ for some connected component T_j and we

have $w(s_i, s_{i+1}) \leq \text{dist}_{G[S \cup T_j]}(s_i, s_{i+1})$. Consequently, we have that P corresponds to a path in the weighted graph G_S and $D_S(u, v)$ is not larger than the length of a shortest u-v-path in G. Thus, $D_S[u, v]$ equals the distance between u and v in G and $D^*[u, v]$ has therefore been computed correctly.

Next, assume that $u \in T_i$ for some $i \in [\nu]$ and $v \in S$ (the case $u \in S$ and $v \in T_i$ is analogous). A shortest u-v-path then consists of a shortest u-s-path in $G[S \cup T_i]$ and a shortest s-v-path in G for some $s \in S$ (since each vertex in T_i only has neighbors in $T_i \cup S$). This implies $\text{dist}_G[u, v] = \min_{s \in S} D_i[u, s] + D_S[s, v] = D_i^*[u, v]$.

Next, assume that $u \in T_i$ and $v \in T_i$ for some $i \in [\nu]$. Then a shortest u-v-path either stays completely in T_i or it decomposes into a shortest u-s-path in $G[S \cup T_i]$, followed by a shortest s-s'-path in G, and finally followed by a shortest s'-v-path in $G[S \cup T_i]$ for some $s, s' \in S$. In the first case, the distance between u and v equals to $D_i[u, v]$. In the second case, the distance between u and v equals

$$\min_{s,s' \in S} \text{dist}_{G[S \cup T_i]}(u, s) + \text{dist}_G(s, s') + \text{dist}_{G[S \cup T_i]}(s', v) = D_{i,i}^*[u, v].$$

Thus, $D^*[u, v]$ contains the distance between u and v.

Finally, assume that $u \in T_i$ and $v \in T_j$ for some $i \neq j \in [\nu]$. A shortest u-v-path then decomposes into a shortest u-s-path in $G[S \cup T_i]$, followed by a shortest s-s'-path in G, and finally followed by a shortest s'-v-path in $G[S \cup T_j]$ for some $s, s' \in S$. Consequently, we have $\text{dist}(u, v) = \min_{s,s' \in S} D_i[u, s] + D_S[s, s'] + D_j[s', v] = D_{i,j}^*[u, v]$.

It remains to analyze the running time of the algorithm. Step 1 runs in linear time by Lemma 17. Step 2 solves APSP on $O(n/k)$ many instances, each with $O(k)$ vertices. As APSP on unweighted graphs with k vertices can be solved in $O(k^\omega \log(k))$ time, Step 2 runs in $O(nk^{\omega-1} \log(k))$ time.

Step 3 first computes a weighted graph on $O(k)$ vertices. Each edge can be computed in $O(n/k)$ time. As there are $O(k^2)$ many edges, computing the graph takes $O(n \cdot k)$ time. Solving weighted APSP on this graph then can be done in $O(k^3)$ time [37]. As we assumed $k \leq n^{0.6}$, this step runs in $O(n^{1.8})$ time.

Step 4 computes for each pair $(i, j) \in [n/k]$ the min-plus product of D_i, D_S, and D_j. We will show that we can compute these min-plus products in $O(k^{(3+\omega)/2})$ time (which is faster than the state-of-the-art algorithm for computing arbitrary min-plus products). The trick is to use the fact that D_i and D_i^* have bounded difference and then use a result by Chi et al. [10] stating that the min-plus product of two matrices of dimension $n \times n$ can be computed in $O(n^{(3+\omega)/2}) \subseteq O(n^{2.687})$ time if one of the matrices has bounded difference. While it is not true a priori that D_i and D_i^* have bounded difference, we will order the rows according to the Hamiltonian path H_i. This ensures that two consecutive vertices are adjacent, which implies that consecutive entries in a row differ by at most one. Hence, each computation of the min-plus product can be done in $O(k^{(3+\omega)/2})$ time, as $|S| = |T_1| = \cdots = |T_\nu| = O(k)$ by Lemma 17. Overall, Step 4 runs in $O((n/k)^2 \cdot k^{(3+\omega)/2}) = O(n^2 \cdot k^{(\omega-1)/2})$ time.

Lastly, Step 5 runs in $O(n^2)$ time. The overall running time of $O(k^{(\omega-1)/2} \cdot n^2)$ follows from the observation that the running time for Step 4 dominates the running time of Steps 1, 2, 3, and 5 as $k \leq n$ and $\omega < 2.9$. ◀

Adaptive algorithm for bounded diameter. We give an adaptive algorithm for constant diameter. The crucial observation here is that we can compute the product of the adjacency matrix of a graph with any other matrix in $O(\iota^{\omega-2} n^2)$ time.

▶ **Lemma 18.** *Given a graph G with adjacency matrix A, a k-separator S for G, and an $n \times n$-matrix M, the matrices AM and MA can be computed in $O(k^{\omega-2} n^2)$ time.*

Proof. Suppose that the adjacency matrix has the form as described in Equation (1). Then,

$$AM = \begin{bmatrix} \gamma M[S,V] + \beta M[\overline{S},V] \\ \beta^T M[S,V] + \alpha M[\overline{S},V] \end{bmatrix}.$$

A closer inspection reveals that all the computation can be done in $O(k^{\omega-2}n^2)$ time: First observe that $\gamma M[S,V]$ can be computed in $O(k^{\omega-1}n)$ time. For the other terms, note that

$$\begin{aligned} \beta M[\overline{S},V] &= \begin{bmatrix} \beta M[\overline{S},S] & \beta M[\overline{S},T_1] & \dots & \beta M[\overline{S},T_\nu] \end{bmatrix}, \\ \beta^T M[S,V] &= \begin{bmatrix} \beta^T M[\overline{S},S] & \beta^T M[\overline{S},T_1] & \dots & \beta^T M[\overline{S},T_\nu] \end{bmatrix}, \\ \alpha M[\overline{S},V] &= \begin{bmatrix} \alpha_1 M[T_1,\overline{S}] & \alpha_2 M[T_2,\overline{S}] & \dots & \alpha_\nu M[T_\nu,\overline{S}] \end{bmatrix}^T. \end{aligned}$$

Since there are $O(n/k)$ submatrices and each takes $O(k^{\omega-1}n)$ time to compute, AM can be computed in $O(k^{\omega-2}n^2)$ time. Note that $MA = (AM^T)^T$ can be computed analogously. ◀

We obtain our algorithm from the folklore observation that for any two vertices $u, v \in V$, there is a walk of length exactly length d between u and v if and only if $A^d[u,v] \neq \emptyset$.

▶ **Proposition 19.** *Given a graph G and a k-separator S, APSP can be solved in $O(dk^{\omega-2}n^2)$ time where d is the diameter of G.*

Proof. Let A be the adjacency matrix of G. We compute matrices $B^1, \dots, B^d \in \{0,1\}^{n \times n}$ recursively as follows. We start with $B^1 := A$. Matrix B^{i+1} is computed by multiplying B^i with A and replacing all non-zero entries by 1. Note that $A^i[u,v] = 0$ if and only if $B^i[u,v] = 0$. Thus, the distance between two vertices $u \neq v$ is the minimum i such that $B^i[u,v] \neq 0$. By Lemma 18, we can multiply any matrix with A in $O(k^{\omega-2}n^2)$ time. Thus, we can compute $B^1, \dots, B^d$ in $O(dk^{\omega-2}n^2)$ time. ◀

Note that d can be of order $\Omega(k^2)$ as the vertex integrity in a cycle with n vertices is in $O(\sqrt{n})$ and its diameter is $\lfloor n/2 \rfloor$.

6 Conclusion

In this work, we investigated the parameter vertex integrity in search for more efficient algorithms in the FPT-in-P paradigm. We exhibited that for many problems, the structure of graphs with a small vertex integrity allows us to harness the power of fast matrix multiplication. In particular, we designed randomized $O(\iota^{\omega-1}n)$-time algorithms for finding a four-vertex subgraph (which is not a K_4 or a $\overline{K_4}$) and a maximum matching. We also showed that unweighted APSP can be solved in $O(\iota^{(\omega-1)/2}n^2)$ time using min-plus product of bounded-differences matrices, leaving open whether there is an $O(\iota^{\omega-2}n^2)$-time algorithm. More broadly, we wonder whether a similar approach using fast matrix multiplication can be used for graphs of bounded tree-depth or tree-width. Existing approaches (e.g. [11, 20, 26]) do not seem amenable to fast matrix multiplication.

References

1 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '16)*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

2 Noga Alon and Raphael Yuster. Fast algorithms for maximum subset matching and all-pairs shortest paths in graphs with a (not so) small vertex cover. In *Proceedings of the 15th Annual European Symposium on Algorithms (ESA '07)*, pages 175–186, 2007. doi:10.1007/978-3-540-75520-3_17.

3 Curtis A Barefoot, Roger Entringer, and Henda Swart. Vulnerability in graphs - A comparative survey. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 1(38):13–22, 1987.

4 D. Benko, C. Ernst, and Dominic Lanphier. Asymptotic bounds on the integrity of graphs and separator theorems for graphs. *SIAM Journal on Discrete Mathematics*, 23(1):265–277, 2009. doi:10.1137/070692698.

5 Matthias Bentert, Till Fluschnik, André Nichterlein, and Rolf Niedermeier. Parameterized aspects of triangle enumeration. *Journal of Computer and System Sciences*, 103:61–77, 2019. doi:10.1016/j.jcss.2019.02.004.

6 Hans L. Bodlaender, Tesshu Hanaka, Yasuaki Kobayashi, Yusuke Kobayashi, Yoshio Okamoto, Yota Otachi, and Tom C. van der Zanden. Subgraph isomorphism on graph classes that exclude a substructure. *Algorithmica*, 82(12):3566–3587, 2020. doi:10.1007/s00453-020-00737-z.

7 Karl Bringmann, Thore Husfeldt, and Måns Magnusson. Multivariate analysis of orthogonal range searching and graph distances. *Algorithmica*, 82(8):2292–2315, 2020. doi:10.1007/s00453-020-00680-z.

8 James R. Bunch and John E. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28(125):231–236, 1974. doi:10.1090/S0025-5718-1974-0331751-8.

9 Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic complexity theory*. Springer, 1997. doi:0.1007/978-3-662-03338-8.

10 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC '22)*, pages 1529–1542. ACM, 2022. doi:10.1145/3519935.3520057.

11 Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985. doi:10.1137/0214017.

12 Derek G. Corneil, Yehoshua Perl, and Lorna K. Stewart. A linear recognition algorithm for cographs. *SIAM Journal on Computing*, 14(4):926–934, 1985. doi:10.1137/0214065.

13 David Coudert, Guillaume Ducoffe, and Alexandru Popa. Fully polynomial FPT algorithms for some classes of bounded clique-width graphs. *ACM Transactions on Algorithms*, 15(3):33:1–33:57, 2019. doi:10.1145/3310228.

14 Mingyang Deng, Yael Kirkpatrick, Victor Rong, Virginia Vassilevska Williams, and Ziqian Zhong. New additive approximations for shortest paths and cycles. In *Proceedings of the 49th International Colloquium on Automata, Languages, and Programming (ICALP '22)*, pages 50:1–50:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.50.

15 Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/s00453-016-0127-x.

16 Pavel Dvorák, Eduard Eiben, Robert Ganian, Dusan Knop, and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP: Programs with few global variables and constraints. *Artificial Intelligence*, 300:103561, 2021. doi:10.1016/j.artint.2021.103561.

17 Pavlos Eirinakis, Matthew D. Williamson, and K. Subramani. On the Shoshan-Zwick algorithm for the all-pairs shortest path problem. *Journal of Graph Algorithms and Applications*, 21(2):177–181, 2017. doi:10.7155/jgaa.00410.

18 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, 2004. doi:10.1016/j.tcs.2004.05.009.

19 Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT '71)*, pages 129–131. IEEE Computer Society, 1971. doi:10.1109/SWAT.1971.4.

20 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, Michal Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms*, 14(3):34:1–34:45, 2018. doi:10.1145/3186898.

21 François Le Gall. Faster algorithms for rectangular matrix multiplication. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS '12)*, pages 514–523. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.80.

22 Tatsuya Gima, Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, and Yota Otachi. Exploring the gap between treedepth and vertex cover through vertex integrity. *Theor. Comput. Sci.*, 918:60–76, 2022. doi:10.1016/j.tcs.2022.03.021.

23 Chris D. Godsil. *Algebraic combinatorics*. Chapman and Hall, 1993.

24 Nicholas J. A. Harvey. Algebraic algorithms for matching and matroid problems. *SIAM Journal on Computing*, 39(2):679–702, 2009. doi:10.1137/070684008.

25 Falko Hegerfeld and Stefan Kratsch. On adaptive algorithms for maximum matching. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP '19)*, pages 71:1–71:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.71.

26 Yoichi Iwata, Tomoaki Ogasawara, and Naoto Ohsaka. On the power of tree-depth for fully polynomial FPT algorithms. In *Proceedings of the 35th Symposium on Theoretical Aspects of Computer Science (STACS '18)*, pages 41:1–41:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.41.

27 Stefan Kratsch and Florian Nelles. Efficient parameterized algorithms for computing all-pairs shortest paths. In *Proceedings of the 37th International Symposium on Theoretical Aspects of Computer Science (STACS '20)*, pages 38:1–38:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.38.

28 Michael Lampis and Valia Mitsou. Fine-grained meta-theorems for vertex integrity. In *Proceedings of the 32nd International Symposium on Algorithms and Computation (ISAAC '21)*, pages 34:1–34:15, 2021. doi:10.4230/LIPIcs.ISAAC.2021.34.

29 Euiwoong Lee. Partitioning a graph into small pieces with applications to path transversal. *Mathematical Programming*, 177(1-2):1–19, 2019. doi:10.1007/s10107-018-1255-7.

30 László Lovász. On determinants, matchings, and random algorithms. In *Proceedings of the 2nd International Symposium on Fundamentals of Computation Theory (FCT '79)*, pages 565–574. Akademie-Verlag, Berlin, 1979.

31 Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *Proceedings of the 45th Symposium on Foundations of Computer Science (FOCS '04)*, pages 248–255, 2004. doi:10.1109/FOCS.2004.40.

32 Kazuo Murota. *Matrices and matroids for systems analysis*. Springer Science & Business Media, 1999. doi:10.1007/978-3-642-03994-2.

33 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980. doi:10.1145/322217.322225.

34 Raimund Seidel. On the all-pairs-shortest-path problem. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC '92)*, pages 745–749. ACM, 1992. doi:10.1145/129712.129784.

35 Avi Shoshan and Uri Zwick. All pairs shortest paths in undirected graphs with integer weights. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS '99)*, pages 605–615, 1999. doi:10.1109/SFFCS.1999.814635.

36 William T. Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947. doi:10.1112/jlms/s1-22.2.107.

37 Richard Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.

38 Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '15)*, pages 1671–1680, 2015. doi:10.1137/1.9781611973730.111.

39 Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the 2nd International Symposium on Symbolic and Algebraic Manipulation (EUROSM '79)*, pages 216–226. Springer, 1979. doi:10.1007/3-540-09519-5_73.