Dynamic Rank, Basis, and Matching

Jan van den Brand
Georgia Institute of Technology, Atlanta, GA, USA

Vishal Kumar
Teachers College, Columbia University, New York, NY, USA

Daniel J. Zhang
Georgia Institute of Technology, Atlanta, GA, USA

Abstract

We study dynamic algorithms for maintaining fundamental algebraic properties of matrices, specifically, rank, basis, and full-rank submatrices, with applications to maximum matching on dynamic graphs. Prior dynamic algorithms for rank achieve subquadratic update times but scale with the matrix dimension n, and could not always maintain the corresponding objects such as a basis or maximum full-rank submatrix.

We present the first dynamic rank algorithms whose update time scales with the matrix rank r, achieving $\widetilde{O}(r^{1.405})$ time per entry-update and $\widetilde{O}(r^{1.528} + z)$ per column-update, where z is the number of changed entries. This extends to $\widetilde{O}(|M|^{1.405})$ edge-update time to maintain the size $|M|$ of a maximum matching. We also give dynamic algorithms for maintaining a column-basis subject to column-updates and a maximum full-rank submatrix subject to entry-updates.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Data structures design and analysis

Keywords and phrases Dynamic Graph Algorithm, Dynamic Algebraic Algorithm, Dynamic Matrix Inverse, Rank, Matching

Digital Object Identifier 10.4230/LIPIcs.ICALP.2026.45

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2605.09917`

Funding *Jan van den Brand*: NSF Award CCF-2338816 and CCF-2504994.
Daniel J. Zhang: NSF Award CCF-2338816.

1 Introduction

Dynamic algebraic algorithms maintain properties of matrices and vector spaces while the input undergoes changes. These algorithms, when used as subroutines, have led to improvements in various areas such as convex optimization [39, 24, 56, 57], network-flows [42, 46, 29, 27, 34], online algorithms [41, 53], failure-sensitive oracles [49, 75, 33, 50].

Many dynamic graph problems (where the goal is to maintain certain graph properties while the graph undergoes edge insertions and deletions) such as reachability [70, 48, 60], strong-connectivity [62] shortest path [71, 61, 28, 31], matching [72], near additive spanner [13, 25], have also been addressed using dynamic algebraic techniques. In fact, it has been shown that achieving subquadratic upper bounds for many of these problems requires the use of algebraic techniques. [1, 13].

This work focuses on two of the most fundamental properties of matrices and vector spaces: *rank* and *basis*. Formally, the problem is defined as follows: we are given an $n \times n$ matrix (i.e., a collection of vectors) that undergoes updates (e.g., column replacements). After each update, we must return the new rank or basis. The "*update-time*" refers to the


time required to compute and return the new rank or basis after an update. We consider both *column-updates* (where an entire column/vector is replaced) and *entry-updates* (where a single matrix entry is modified).

Dynamic Rank & Matching. Given a graph G, the size of a maximum matching is equal to half the rank of the graph's Tutte-matrix[1]. Hence, by maintaining the rank of a dynamic matrix under entry-updates, we can maintain the size of a maximum matching in a graph that undergoes edge insertions and deletions. The latter problem is known as dynamic maximum matching. Via further reductions, this also extends to other graph problems such as dynamic vertex-capacitated max-flow.

Dynamic maximum matching is a well-studied problem in the dynamic graph area with substantial amount of work in recent years for the approximate setting [11, 65, 23, 20, 9, 22, 8]. In the exact setting, the upper bounds for n-vertex graphs are $O(n^{1.528})$ [43], $O(n^{1.446})$ [72] and $O(n^{1.405})$ [32]. These upper bounds for exact matching are achieved via the reduction from matching to rank, i.e., despite the graph application, these dynamic algorithms are entirely algebraic and maintain the rank of the Tutte-matrix. The algebraic approach and slower progress in the exact setting can be explained via conditional lower bounds. The two highest corresponding conditional lower bounds for dynamic matching are $\Omega(n^{\omega-1}) \sim \Omega(n^{1.372})$ based on the triangle detection conjecture[2] [1], and $\Omega(n^{1.405})$ [32] based on the hinted-Mv conjecture[3]. These lower bounds imply that any upper bound must be algebraic (i.e., make use of fast matrix multiplication) to beat the naive $\widetilde{O}(n^2)$ upper bound[4] of recomputing the matching from scratch on dense graphs [30, 37]. Further, depending on which conjecture one believes, the current $O(n^{1.405})$ update time is conditionally optimal or nearly so (with only a $o(n^{0.04})$ gap to triangle detection for current bounds on ω). While these conditional lower bounds seemingly rule out substantial further progress in the exact setting, they only holds for distinguishing between matching-size (or rank) n vs $n-1$. Thus, the lower bounds do not apply to graphs with small maximum matchings (or matrices with small rank), leaving open the possibility of significant improvements for the low-rank/small-matching case.

Such improvements for low-rank matrices have previously been obtained in the static setting. While computing the rank of an $n \times n$ matrix $\mathbf{A}$ generally takes $O(n^\omega)$ time, [38] showed how to compute the rank in only $\widetilde{O}(\text{nnz}(\mathbf{A}) + r^\omega)$ time, where $\text{nnz}(\mathbf{A})$ is the number of non-zeros (i.e., time to read the input matrix $\mathbf{A}$) and r is the rank of the input matrix. So for low-rank matrices (e.g., $r < n^{1/2}$), the rank can be computed in nearly-linear time.

On the one hand, this highlights a gap in the dynamic setting: for sparse low-rank matrices, naïve recomputation of the rank from scratch via [38] is already faster than the current $O(n^{1.405})$ dynamic algorithms. So no non-trivial dynamic algorithm is known for such matrices. On the other hand, the result of [38] suggests that it may be possible to improve the dynamic complexity by having it depend on rank r rather than dimension n. Our first result shows this is indeed possible: The rank r can be maintained in $\widetilde{O}(r^{1.405})$ time per entry-update. This also implies a dynamic algorithm for maximum matching size with $O(|M|^{1.405})$ edge-update time, where $|M|$ is the size of the maximum matching.

---

[1] The Tutte-matrix [73] is the symbolic adjacency matrix where the signs are flipped along the diagonal, i.e. $\mathbf{A}_{i,j} = -\mathbf{A}_{j,i}$.

[2] Triangle conjecture: Given an n-vertex graph, triangles cannot be detected in $O(n^{\omega-\epsilon})$ time.

[3] The hinted-Mv conjecture [32] is a variation of the OMv conjecture [54], parametrized by the fast matrix multiplication exponent. The lower bound $\Omega(n^{1.405})$ is for current bounds on rectangular matrix multiplication [2].

[4] We use $\widetilde{O}(\cdot)$ to hide $\text{polylog}(n)$ factors.

Dynamic Basis and Full-Rank Submatrix. The rank of a set of vectors (or a matrix) is the size of the maximum linearly independent subset (or size of the maximum full-rank submatrix). Previous fastest dynamic rank data structures [72, 32, 26] can maintain the rank (i.e., *size* of basis/submatrix), but not the actual basis or submatrix itself. These algorithms work via a black-box reduction by [72], which reduces dynamic rank to dynamic matrix inverse (here one must maintain the inverse while the input matrix changes over time). Thus improvements for dynamic matrix inverse [32, 26, 4] have directly translated to better algorithms for dynamic rank and matching size.

However, when it comes to maintaining an actual basis, the only known algorithm is that by [43] which, rather than a black-box reduction, was a dynamic algorithm specifically designed for maintaining the rank[5]. Thus, this algorithm has not benefited from recent advances in dynamic matrix inversion, and thus the update time remains at $O(n^{1.528})$, which is slower than the $O(n^{1.405})$ update time for rank. If there was a black-box reduction from basis/full-rank submatrix to dynamic matrix inverse, similar to how there is a reduction from rank, then these issues would be resolved.

The ability to maintain the size, rather than the corresponding object, is a common limitation of dynamic algebraic algorithms. This issue extends to the applications of dynamic algebraic algorithms such as maintaining reachability, strong-connectivity, maximum matching, or shortest paths in dynamic graphs. [1] have proven that solving these problems in the dynamic setting faster than the trivial $\widetilde{O}(n^2)$ requires the use of fast matrix multiplication, i.e., algebraic techniques. However, given the algebraic nature, most dynamic algebraic algorithms maintain numbers/quantities rather than the respective combinatorial objects. For example, [71, 31, 61, 25] maintain distances (i.e., length of shortest path) rather than the paths. [72] maintains the size of the maximum matching rather than the matching. For reachability, [70, 40] maintain boolean reachability information, but not the respective paths. [32] maintains a boolean output whether the graph is strongly-connected, but not the strongly-connected components. This inability to maintain the underlying object is a major draw-back of dynamic algebraic methods and has been stated as open problems in [31, 32]. Recently, studying how to maintain objects via dynamic algebraic techniques has developed to an active area of research with first progress for paths: [59] extended dynamic reachability to path reporting, and [13, 3] can maintain shortest paths. [62] extends strong-connectivity to maintaining the strongly-connected components. However, for maintaining a maximum full-rank submatrix (rather than just the rank), or a maximum matching (rather than just its size), the question remains open.

In this work, we give the first reduction from basis (and maximum full-rank submatrix) to dynamic rank. The latter reduces to dynamic matrix inversion [72], thus, all recent improvements to dynamic matrix inverse now also extend to dynamic basis. As a full-rank submatrix corresponds to the vertex set used by the maximum matching, this is also the first fully dynamic algorithm that maintains information about the exact maximum matching beyond its size.

1.1 Our Results

Maintaining the Rank. Our first result are improved time complexities for maintaining the rank of a dynamic $n \times n$ matrix, scaling with the rank of the matrix rather than its dimension. In the following, $\text{nnz}(\mathbf{A})$ is the number of non-zero entries in $\mathbf{A}$, when given in sparse representation.

[5] [43] state their results for dynamic rank. It was not explicitly stated that they maintain basis or submatrix, but internally their algorithm maintains the positions of leading 1s of the row-echelon form. This allows to extract the basis.

	Entry Update		Column Update	
Output	New	Previous	New	Previous
Rank	$r^{1.405}$	$n^{1.405}$ [72, 32]	$z + r^{1.528}$	$n^{1.528}$ [72, 32]
Column-Basis	$n^{1.405}$ and $r^{1.528}$	$n^{1.528}$ [43][5]	$z + r^{1.528}$	
Submatrix	$n^{1.405}$ and $r^{1.528}$			

Figure 1 New and previous upper bounds for dynamically maintaining rank, basis, full-rank submatrix. Parameter n bounds the width and height of the input matrix, r is its rank, and z is the number of entries changed by the column update (i.e., input size of the update).

▶ **Theorem 1.** *There are randomized dynamic algorithms that maintain, with high probability, the rank of a dynamic $n \times n$ matrix $\mathbf{A}$. Both have preprocessing time $\widetilde{O}(\text{nnz}(\mathbf{A}) + \text{rank}(\mathbf{A})^{\omega})$ and their update times are:*

- $\widetilde{O}(\text{rank}(\mathbf{A})^{1.405})$ *time per entry-update,*
- $\widetilde{O}(\text{rank}(\mathbf{A})^{1.528} + z)$ *time per column-update, where z is the number of entries changed in the column.*

The dynamic algorithms are robust against output-adaptive adversaries[6]*.*

The term z in the column-update time accounts for reading the input (i.e., change of the column). For dense updates to low-rank matrices, the column-updates are nearly-linear time. The complexity dependence on $\text{rank}(\mathbf{A})$ is conditionally optimal, i.e., any further improvement would also improve the high-rank time complexities of $O(n^{1.405})$ and $O(n^{1.528})$. These are also the first non-trivial dynamic algorithms for low-rank matrices with $\text{nnz}(\mathbf{A}) < n^{1.405}$ or $n^{1.528}$ respectively.

Theorem 1 is obtained by developing a black-box reduction from maintaining the rank of $n \times n$ matrices to $r \times r$ matrices, where $r = \widetilde{O}(\text{rank}(\mathbf{A}))$, via techniques of [38].

▶ **Theorem 2.** *Assume there is a data structure that maintains the rank of a dynamic $n \times n$ matrix $\mathbf{A}$ with $U(n, z)$ column-update time, where z is the number of changed entries. Let $P(n)$ be the preprocessing time.*

Then there is a randomized dynamic algorithm for maintaining the rank of $\mathbf{A}$ after $\widetilde{O}(\text{nnz}(\mathbf{A}) + P(\text{rank}(\mathbf{A})))$ time preprocessing with

$$\widetilde{O}(z + U(\text{rank}(\mathbf{A}), z) + P(\text{rank}(\mathbf{A}))/\text{rank}(\mathbf{A}))$$

update time for column updates that change z entries. The output is correct with high probability and the dynamic algorithm is robust against output-adaptive adversaries.

Applying this reduction to the best-known dynamic rank algorithms ($O(n^{1.528})$ for column-updates, and $O(n^{1.405})$ for entry-updates [71, 32]), results in Theorem 1. These upper bounds depend on the current best bounds for fast rectangular matrix multiplication [2].

Since the rank of a graph's Tutte-matrix is twice the size of a maximum matching, Theorem 1 directly implies algorithms for dynamic maximum matching size. For bipartite graphs, a reduction by [58] allows us to maintain the weight of the maximum weight matching as well.

[6] Here the adversary sees the output of the data structure, but not the internal randomness.

▶ **Theorem 3.** *There are randomized dynamic algorithms that maintain with high probability*
- *the weight k of a maximum weight matching in a bipartite graph with $O(W \cdot k^{1.405})$ time per edge-update, where W is the weight of the updated edge.*
- *the size $|M|$ of the maximum matching M in a general graph in $O(|M|^{1.405})$ time per edge-update.*

The previous best bounds for these problems were $O(n^{1.405})$ for maximum matching and $O(W^{2.405} \cdot n^{1.405})$ for the weighted case [71, 32].

Maintaining a Basis. The dynamic algorithms of Theorem 1, and those of [71, 32] can only maintain the rank of a matrix (i.e., set of vectors) but not a corresponding basis. We extend the results to maintain a column-basis:

▶ **Theorem 4.** *There is a randomized dynamic algorithm that maintains with high probability a column-basis of a dynamic $n \times n$ matrix $\mathbf{A}$ in $\widetilde{O}(z + \text{rank}(\mathbf{A})^{1.528})$ time per column update. Here z is the number of entries changed during the column update.*

The dynamic algorithm is robust against output-adaptive adversaries.

In case of column-updates, the time complexity matches that of Theorem 1 for maintaining the rank. For entry-updates (i.e., when $z = 1$) there is a small slow-down compared to the $O(n^{1.405})$ update time for maintaining the rank. It still improves over the previous best bound $O(n^{1.528})$ for maintaining a basis under entry-updates [43].

Our result is the first to maintain a basis subject to column-updates. Column-updates to a matrix correspond to adding and removing vectors to some set $V = \{v_1, ..., v_n\}$ and maintaining a maximum linear independent subset of them. In this context, column-updates are arguably more natural than entry-updates.

Theorem 4 is obtained by developing a reduction and using the $O(n^{1.528})$ column-update time dynamic rank algorithm from [32].

▶ **Theorem 5.** *Given a dynamic algorithm for maintaining the rank of a dynamic matrix $\mathbf{A} \in \mathbb{F}^{n \times n}$ with column-update time $U(n)$ and processing time $P(n)$.*

Then there is dynamic algorithm maintaining a column-basis of $\mathbf{A} \in \mathbb{F}^{n \times n}$ with column-update time $\widetilde{O}(P(\text{rank}(\mathbf{A}))/\text{rank}(\mathbf{A}) + U(\text{rank}(\mathbf{A})) + z)$. Here z is number of entries that changed in the updated column.

The dynamic algorithm is robust against output-adaptive adversaries.

Maintaining a Full-Rank Submatrix. Given a matrix $\mathbf{A} \in \mathbb{F}^{n \times n}$, computing a full-rank submatrix corresponds to first constructing a column-basis (i.e., subset of columns of $\mathbf{A}$) and row-basis (i.e., subset of rows of $\mathbf{A}$), and then taking the submatrix induced by the row and column set[7].

Running Theorem 4 or [43] on $\mathbf{A}$ and $\mathbf{A}^\top$ would allow to maintain a maximum full-rank submatrix in $O(\text{rank}(\mathbf{A})^{1.528})$ or $O(n^{1.528})$ time. However, maintaining the rank can be done in $O(\text{rank}(\mathbf{A})^{1.405})$ time via Theorem 1 (or $O(n^{1.405})$ via [72, 32]).

To address these issues, we develop a new approach for maintaining a maximum full-rank submatrix. We show that maintaining such a submatrix (and thus also the rank of a matrix) reduces to merely detecting when a dynamic matrix becomes singular (i.e., non-invertible; $\det(\mathbf{A}) = 0$) for the first time.

[7] This only works when we have a column- and row-*basis*. It does not work when these are merely linearly independent column (or row) vectors. See full version for details.

▶ **Theorem 6.** *Assume there is a data structure that detects when a dynamic $n \times n$ matrix $\mathbf{A}$ becomes singular for the first time, with $U(n)$ time per entry-update. Let $P(n)$ be the preprocessing time.*

Then there is a dynamic algorithm for maintaining a maximum full-rank submatrix (i.e., sets $I, J \subset [n]$ with $\det(\mathbf{A}_{I,J}) \neq 0$ and $|I| = |J| = \text{rank}(\mathbf{A})$) after $O(P(n) + n^\omega)$ time preprocessing with $\widetilde{O}(U(n))$ entry-update time.
The reduction is randomized, correct with high probability, and works against an output-adaptive adversary.

A reduction from rank to singularity-detection was previously given by [72], but it could only maintain the rank, and not a corresponding object such as a basis or full-rank submatrix.

By applying our reductions to dynamic determinant algorithm of [32], we obtain the following result:

▶ **Theorem 7.** *There is a dynamic algorithm that maintains with high probability a $\text{rank}(\mathbf{A}) \times \text{rank}(\mathbf{A})$-sized full-rank submatrix of a dynamic $n \times n$ matrix $\mathbf{A}$. After each update, the data structure returns the set of row/column-indices that define the submatrix. The preprocessing time is $O(n^\omega)$ and the update time is $\widetilde{O}(n^{1.405})$ per entry update.*

This result also implies that we can maintain the vertex set of a maximum matching in a dynamic graph undergoing edge updates. This improves over previous algorithms, which could only maintain the size of the matching [72, 32, 26].

Remark: Combinatorial Approach. A combinatorial approach for maintaining a matching (besides recomputation from scratch in $\widetilde{O}(n^2)$ time [30, 37]) is to find an augmenting path in $O(n^2)$ time after each update. Beating quadratic time requires algebraic techniques [1].

Given how there is an $O(n^2)$ combinatorial and $O(n^{1.405})$ algebraic algorithm, it is natural to ask if there is also an $O(|M|^2)$ combinatorial algorithm matching the $O(|M|^{1.405})$ proven in Theorem 3. We observe that there is an $O(|M|^2)$ time combinatorial algorithm based on augmenting paths using vertex sparsification from [52].

▶ **Theorem 8.** *theoremthmCombinatorial There is a deterministic algorithm that maintains the maximum matching $M \subset E$ in a dynamic bipartite graph $G = (V, E)$ subject to edge updates in $O(|M|^2)$ time.*

1.2 Further Related Work

In addition to the conditional lower bounds stated in the introduction, there is also an $\Omega(n)$ conditional lower bound for dynamic maximum matching based on the OMv-conjecture [54].

Dynamic matching (maximal, maximum, weighted, etc.) is one of the most well-studied problems in the dynamic graph area, see e.g.,[7, 14, 15, 17, 16, 68, 74, 52, 66, 64, 21, 63, 36, 35, 66, 19, 18, 12, 6, 10, 67, 11, 65, 23, 20, 9, 22, 8] and references therein. Maintaining small matching/max-flow in the partially dynamic setting (only insertion/only deletions) was studied in [55, 51, 47]. Further dynamic algebraic problems that have been studied are dynamic characteristic polynomial [45], dynamic Frobenius normal form [61], dynamic matrix vector multiplication [5], dynamic convolution (polynomial multiplication) [69]. [44] studied dynamic algebraic algorithms from the perspective of algebraic circuit lower bounds.

Similar to the exact setting, in the better-than-2 approximate setting the dynamic matching algorithms from [11, 22, 20] can also only maintain the size but not the matching itself. Unfortunately our techniques for maintaining the vertex set seem to be difficult to extend to the approximate setting, because they rely on distinguishing size n vs $n-1$ (i.e., existence of a perfect matching).

1.3 Preliminaries/Notation

We write $[n]$ for set $\{1, ..., n\}$. For matrix $\mathbf{M} \in \mathbb{F}^{n \times n}$ and sets $A, B \subset [n]$, we let $\mathbf{M}_{A,B}$ be the submatrix consisting of the rows and columns with index in A and B respectively. Similarly, we let $\mathbf{M}_{-A,-B}$ be the matrix obtained via deleting the rows and columns with index in A and B respectively. For bipartite graphs $G = (U \cup W, E)$ with $|U| = |W| = n$, we define the biadjacency matrix $\mathbf{A} \in \mathbb{F}^{U \times W}$ via $\mathbf{A}_{u,w} = 1$ if $\{u, w\} \in E$ and $\mathbf{A}_{u,w} = 0$ otherwise. For permutation $\sigma : [n] \to [n]$ the sign of σ is $\text{sign}(\sigma) = (-1)^t$ where t is the number of inversions (i.e., number of pairs $i < j$ with $\sigma(i) > \sigma(j)$).

▶ **Lemma 9** (Schwartz-Zippel-Lemma). *Let $\mathbb{F}$ be a field and f be a non-zero polynomial of degree d with variables $x_1, ..., x_n$, i.e., $f \in \mathbb{F}[x_1, ..., x_n]$. Let $r_1, ..., r_n$ be i.i.d. uniformly sampled elements of $\mathbb{F}$. Then $\mathbb{P}[f(r_1, ..., r_n) = 0] \leq d/|\mathbb{F}|$.*

Intuitively, this lemma states that if there exists some input for f which evaluates to non-zero, then finding such an input is trivial by simply sampling a random input.

▶ **Lemma 10** ([38, Lemma 2.4&2.5]). *Given $n, k \in \mathbb{N}$, in $O(n)$ time we can construct a random $n \times r$ matrix $\mathbf{M} \in \mathbb{F}^{n \times r}$ with $r = \Theta(k)$, such that*

- *each column of $\mathbf{M}$ has at most $2\lceil n/r \rceil$ non-zero entries, and each row has 2 non-zero entries.*
- *for any matrix $\mathbf{A} \in \mathbb{F}^{m \times n}$, we have with probability at least $1 - O(1/k + k/|\mathbb{F}|)$ that $\text{rank}(\mathbf{AM}) = \min(\text{rank}(\mathbf{A}), r)$.*

The matrix $\mathbf{M}$ in Lemma 10 can be seen as a rank-preserving random projection. It projects m many n-dimensional vectors (the rows of $\mathbf{A}$) onto a smaller r-dimensional space such that with constant probability the rank is preserved. Since the new space is r-dimensional, the rank is preserved only when at most r, hence $\text{rank}(\mathbf{AM}) = \min(\text{rank}(\mathbf{A}), r)$. Further, the projection $\mathbf{M}$ is sparse, i.e., changing an entry of a row-vector $v^\top$ will change only 2 entries in its projection $v^\top \mathbf{M}$.

Remark. We can assume without loss of generality that field $\mathbb{F}$ has size $\text{poly}(n)$, otherwise we use a field extension of such size [38, Lemma 2.1]. Further, there is $k_0 = \Theta(1)$ such that for all $k > k_0$, the success probability of Lemma 10 is at least $1/2$. We can then further boost the success probability of Lemma 10 by sampling several $\mathbf{M}_1, ..., \mathbf{M}_\ell$ and returning $\max_{i \in [\ell]} \text{rank}(\mathbf{AM}_i)$. Then we have w.h.p., that the returned rank is $\min(\text{rank}(\mathbf{A}), r)$.

2 Technical Overview

We begin by outlining how to maintain a maximum sized full-rank submatrix (Theorem 7) in Section 2.1. The idea of the algorithm is based on constructing a gadget matrix by interpreting the full-rank submatrix problem graphically using connections between the determinant of a matrix and perfect matchings in graphs. Rather than a full-rank submatrix, we identify a vertex-set that must be deleted such that a perfect matching becomes possible.

Then, in Section 2.2, we explain how to improve the maintenance of rank (Theorem 1) and basis (Theorem 4) for low-rank matrices. These latter results are based on a random projection technique by [38].

2.1 Maintaining Full-Rank Submatrix (Section 3)

The task of finding a maximum sized full-rank submatrix for $n \times n$ matrix $\mathbf{A}$ is equivalent to finding a minimum set $S, T \subseteq [n]$ of rows and columns to delete from $\mathbf{A}$ such that $\det(\mathbf{A}_{-S,-T}) \neq 0$. Identifying these sets will rely on the following structural Lemma 13, proven in Section 3. In Lemma 13, matrix $\mathbf{A}$ is the original input matrix and $\mathbf{B}$ will be some gadget matrix that we construct.

▶ **Lemma 13.** *Suppose $\mathbf{A} \in R^{n\times n}$ and $\mathbf{B} \in R^{m\times m}$ with $m \geq n$ for some commutative ring R. Let*

$$\mathbf{H} = \left[\begin{array}{ccc|ccc|c} & & & x_1 & & & 0\dots0 \\ & \mathbf{A} & & & \ddots & & \vdots \\ & & & & & x_n & 0\dots0 \\ \hline y_1 & & & & & & \\ & \ddots & & & & & \\ & & y_n & & \mathbf{B} & & \\ \hline 0 & \dots & 0 & & & & \\ \vdots & & & & & & \\ 0 & \dots & 0 & & & & \end{array}\right].$$

Then,

$$\det(\mathbf{H}) = \sum_{S,T\subseteq[n],|S|=|T|} (-1)^{|S|} \det(\mathbf{A}_{-S,-T}) \det(\mathbf{B}_{-T,-S}) x^S y^T.$$

In particular, $\det(\mathbf{H}) \neq 0$ *as a polynomial in* $x_1, \dots, x_n, y_1, \dots, y_n$ *iff there exists* $S, T \subseteq [n]$ *with* $\det(\mathbf{A}_{-S,-T}) \neq 0$ *and* $\det(\mathbf{B}_{-T,-S}) \neq 0$.

Via the Schwartz-Zippel-Lemma 9 and replacing $x_1, ..., x_n, y_1, ..., y_n$ by random field elements, Lemma 13 can be restated as follows:

▶ **Fact 11.** *W.h.p. matrix $\mathbf{H}$ has $\det(\mathbf{H}) \neq 0$ if and only if there is $S, T \subset [n]$ where $\det(\mathbf{A}_{-S,-T}) \neq 0$ and $\det(\mathbf{B}_{-T,-S}) \neq 0$.*

We maintain whether $\det(\mathbf{H}) \neq 0$, for some gadget matrix $\mathbf{B}$. We construct $\mathbf{B}$ in such a way so that we know the collection of $S, T \subseteq [n]$ with $\det(\mathbf{B}_{-T,-S}) \neq 0$. Thus, when $\det(\mathbf{H}) \neq 0$, we know that $\det(\mathbf{A}_{-S,-T}) \neq 0$ for at least one of these S, T. In particular, by maintaining $\det(\mathbf{H})$ for gadget matrix $\mathbf{B}$, we can implement a binary search procedure to determine a minimum S, T with $\det(\mathbf{A}_{-S,-T}) \neq 0$.

Invariant and Update Procedure. Our dynamic algorithm will maintain the following invariant after each update to $\mathbf{A}$. We maintain sets $S, T \subseteq [n]$ and a matrix $\mathbf{B}$ such that

- **(i)** S, T are the unique subsets of $[n]$ for which $\det(\mathbf{B}_{-T,-S}) \neq 0$. Note that $\mathbf{B}$ is $m \times m$ with $m \geq n$ and $S, T \subset [n]$, so only the first n rows/columns are allowed to be deleted.
- **(ii)** $\mathbf{A}_{-S,-T}$ is a maximum sized full-rank submatrix of $\mathbf{A}$.

By property (ii), we know a full-rank submatrix of $\mathbf{A}$, which is exactly what our dynamic algorithm is supposed to return after each update. Property (i) will hold after we are done processing the update to $\mathbf{A}$, but uniqueness does not hold while processing the update, as we are performing a binary search to find one valid S, T.

To outline how we maintain these two properties, consider the following possibilities after we receive an entry-update to $\mathbf{A}$. Since a single entry $\mathbf{A}_{i,j}$ is updated, the rank of $\mathbf{A}$ can change by at most 1, giving us the following cases:

(1) The rank increases, so we can add one more row and column to the full-rank submatrix of $\mathbf{A}$. In particular, our task is to find and remove an index from S and an index from T, such that (ii) holds again.
(2) The rank decreases, so we must remove a row and column from the (previously) full-rank submatrix of $\mathbf{A}$. In particular, the task is to find and add an index to S and an index to T, such that (ii) holds again. This task is trivial since updating entry $\mathbf{A}_{i,j}$ means that the i-th row and j-th column are precisely the ones to be removed from the submatrix, i.e., $S \leftarrow S \cup \{i\}, T \leftarrow T \cup \{j\}$.
(3) The rank stays the same. We can reduce this to case (2)+(1): After an update to $\mathbf{A}_{i,j}$, we add i, j to S, T, then attempt to remove an index from S and from T again via the procedure of (1).

So our update routine is as follows: Before updating entry $\mathbf{A}_{i,j}$, we check if the update will make $\mathbf{A}_{-S,-T}$, and thus $\mathbf{H}$, singular. If it would, we add i to S and j to T before making the update. In either case, we update $\mathbf{A}_{i,j}$ and then attempt to find an index to remove from S and an index to remove from T and remove them. We do this last step (searching for and removing a pair of indices) twice.

We are left with describing the procedure for finding $s \in S$ and $t \in T$ such that after removal we still have $\det(\mathbf{A}_{-S,-T}) \neq 0$. To do this, we construct our gadget matrix $\mathbf{B}$ such that the following holds:

▶ **Property 12.** *Given $S, T \subset [n]$ and $C = \{l, ..., r\}, C' = \{l', ..., r'\}$, the matrix $\mathbf{B}$ has the property that all $S', T' \subset [n]$ with $\det(\mathbf{B}_{-T',-S'}) \neq 0$ are precisely the sets $S' = S \setminus \{c\}, T' = T \setminus \{c'\}$ for $c \in C \cap S, c' \in C' \cap T$.*

Thus, when $\det(\mathbf{H}) \neq 0$, then know there is some S', T' with $\det(\mathbf{A}_{-S',-T'}) \neq 0$, and by repeatedly halving the size of C, C' we can binary search for a single valid S', T'.

Given the binary search structure, it suffices to restrict to C, C' to be certain intervals. (When n is a power of two, they will have the form $\{k \cdot 2^\ell + 1, ..., (k+1)2^\ell\}$ for $\ell, k \in \mathbb{N}$). We show that for S, T and all the C, C' needed in the binary search, there is a matrix $\mathbf{B}$ with Property 12. Further, changing one entry in S, T and halving sets C, C' results in only $O(1)$ changed entries in $\mathbf{B}$. Thus, after an update to matrix $\mathbf{A}$, we only perform $O(\log n)$ updates to $\mathbf{B}$ and thus only $O(\log n)$ updates to $\mathbf{H}$. In particular, when given a dynamic algorithm maintaining $\det(\mathbf{H}) \stackrel{?}{=} 0$ with update time U, we have a dynamic algorithm maintaining a full-rank submatrix of $\mathbf{A}$ in update time $O(U \log n)$ (Theorem 7[8]).

Gadget Matrix B via Gadget Forest *F*. We get to choose matrix $\mathbf{B}$ in our algorithm. To have precise control about the possible sets S, T where $\det(\mathbf{B}_{-T,-S}) \neq 0$ we will use a graph theoretic perspective based on perfect matchings in bipartite graphs. For any $m \times m$ matrix $\mathbf{B}$ we can write the determinant as

$$\det(\mathbf{B}) = \sum_{\sigma \in \mathcal{S}_m} \operatorname{sign}(\sigma) \prod_{i=1}^{m} \mathbf{B}_{i,\sigma(i)}.$$

Here σ are permutations over $[m]$. When $\mathbf{B}$ represents a bipartite graph, i.e., edge $\{i, j\}$ exists if and only if $\mathbf{B}_{i,j} \neq 0$, then each non-zero product in above sum represents a valid

[8] Theorem 7 states that it suffices to detect when $\det(\mathbf{H}) = 0$ for the first time. That's because by our construction of $\mathbf{B}$, any updating causing $\det(\mathbf{H}) = 0$ will be followed by an update that reverts the previous one. So one can simply revert and skip the previous update that made the matrix singular.

perfect matching (given by $(i, \sigma(i))$) in the graph. The main issue is that $\text{sign}(\sigma) \in \{\pm 1\}$ could lead to cancellations if the perfect matching is not unique. To enforce uniqueness, we let the graph represented by $\mathbf{B}$ be a forest.

We let $\mathbf{B} \in \mathbb{F}^{m \times m}$ be the biadjacency matrix of a forest $F = ((U, V), E)$ on $2m$ vertices. Here $U = [m], V = [m]$ are the left/right vertices since a forest is bipartite. In particular, the row indices of $\mathbf{B}$ correspond to U and the column indices correspond to V. As such, $B_{-T,-S}$ is also forest since one only deleted the vertices $S \subset U$ and $T \subset V$ from forest F. Thus, the condition $\det(\mathbf{B}_{-T,-S}) \neq 0$ is equivalent to $F[U \setminus S, V \setminus T]$ (i.e., graph F after deleting vertices S and T) having a perfect matching.

Hence from now on, rather than argue about sets S, T with $\det(\mathbf{B}_{-T,-S}) \neq 0$, we will argue about vertex sets whose removal allows forest F to have a perfect matching.

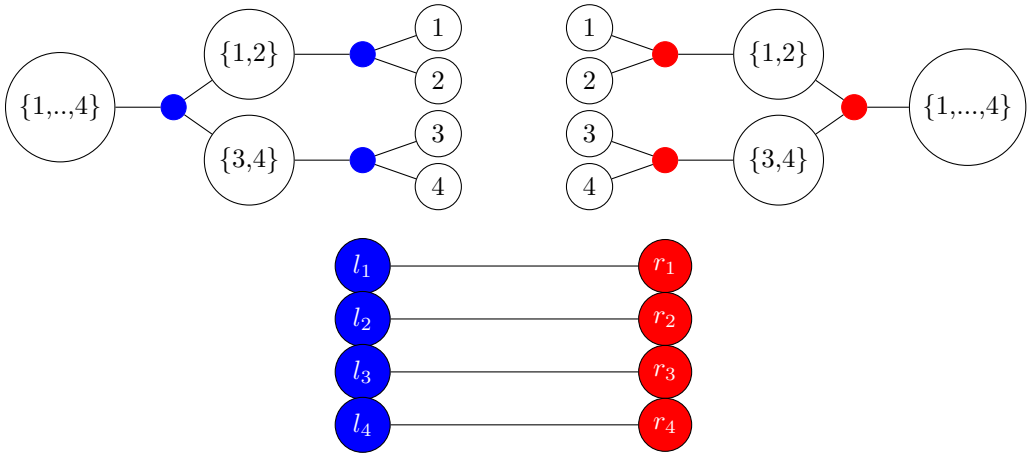

Figure 2 Base forest construction, before adding any dependence on $S, T, C, C' \subset [n]$.

Gadget Forest. *Base Gadget.* We first describe a forest F that does not yet depend on given sets S, T, C, C'. We will then modify the forest for some given S, T, C, C'. The forest is given by the following construction, depicted in Figure 2. We have two binary trees with n leaves where every internal vertex is split into two (a colored and a white vertex), with the colored vertex having the children and the white vertex having the parent. The leaves of the two trees are labeled with $1, ..., n$. In particular, sets $S, T \subset [n]$ of vertices that are to be deleted from F will correspond to the left/right leaves. Any white vertex is labeled by the set of leaves in that subtree, thus any possible set[9] C, C' corresponds to one white vertex of the tree on the left/right.

Additionally, we have $2n$ vertices $(\ell_1, ..., \ell_n$ on the left and $r_1, ..., r_n$ on the right respectively) connected to their horizontal neighbor, i.e., l_i connects to r_i. We refer to these vertices also as the "switch-vertices" and they will later be connected to other vertices in the forest based on some given sets S, T, C, C'.

Modification based on S and T. Recall that S, T must be the (unique) set of leaves to be deleted so the forest has a perfect matching. If we want the forest in Figure 2 to have a perfect matching, then each white tree node must be matched to its colored child. So

[9] Recall that C, C' are not arbitrary sets but facilitate a binary search. Thus they match the vertex labels of the binary tree.

the only way to have a perfect matching is if we were to delete all leaves. Since we want to have to delete only S, T, we do the following: we disconnect pairs of switch-vertices and connect them to leaves not in S, T. Then those leaves must be matched to the respective switch-vertices and must no longer be deleted. An example is given in Figure 3.

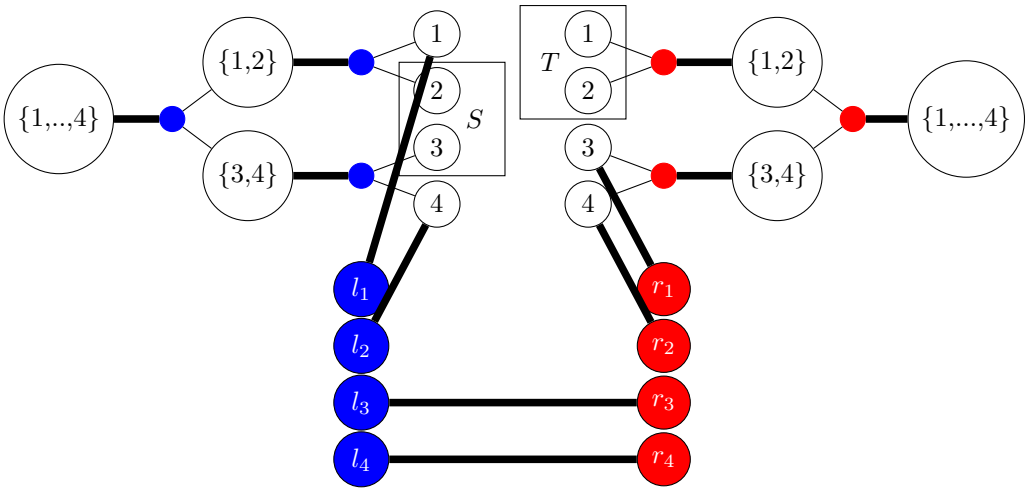

Figure 3 For $S = \{2, 3\}$ the left leaves $1, 4$ are connected to switch-vertices. For $T = \{1, 2\}$, the right leaves $3, 4$ are connected to switch-vertices. Set S, T are the unique set of leaves after whose deletion the forest has a perfect matching (bold lines).

Modification based on C and C'. Given C (and C'), we disconnect another pair of switch-vertices and connect them to the white vertices labeled with C (and C') of the left (and right) tree. Observe that, since a switch-vertex always has a single neighbor, it must be matched to that neighbor. So connecting a switch-vertex to some vertex v is equivalent to deleting both the switch-vertex and v, when it comes to the graph having a perfect matching. So to simplify Figure 4 we removed all vertices connected to switch-vertices.

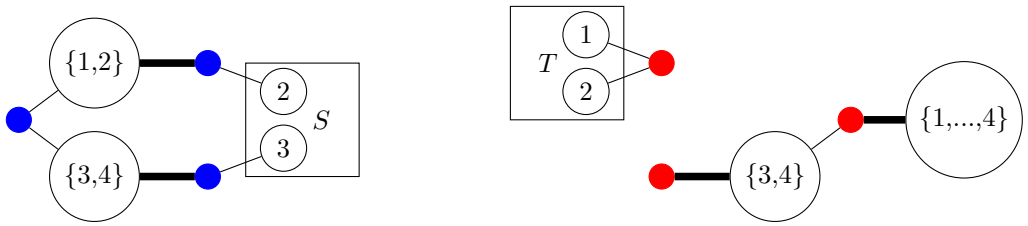

Figure 4 $S = \{1, 2\}, T = \{1, 2\}, C = \{1, 2, 3, 4\}, C' = \{1, 2\}$, vertices connected to switch-vertices have been removed for simplicity. Observe that $S' = S \setminus \{c\}$ for $c \in C \cap S$ are the only sets of leaves whose deletion allows the blue forest to have a perfect matching. The choice of c corresponds to an augmenting path from the unmatched blue vertex to c. Likewise $T' = T \setminus \{c'\}$ for $c' \in C' \cap T$ are the possible leaf deletions such that the red forest has a perfect matching.

Since internal tree vertex C (e.g., a set like $\{1, 2, 3, 4\}$) is removed, it can no longer be matched to its colored child, so the colored child must be matched to either the top or bottom half of set C (e.g., $\{1, 2\}$ or $\{3, 4\}$). We essentially get an augmenting path along the tree from the removed vertex C to one of the leaves. This only works if the leaf was not removed, i.e., not connected to a switch-vertex, but also the leaf must not be in S (since S is the set of leaves we want to delete). Thus one can show that, only deletion of leaves $S' = S \setminus \{c\}$

for any $c \in C \cap S$ allows the tree to have a perfect matching. The same argument holds for the other tree, deleting leaves $T' = T \setminus \{c'\}$ for any $c' \in C'$ allows the tree to have a perfect matching.

Thus, given S, T, C, C' we have a forest where the possible set of leaf deletions such that the forest has a perfect matching, are all of the form $S' = S \setminus \{c\}$, $T' = T \setminus \{c'\}$ for any $c \in C \cap S$, $c' \in C' \cap T$. Using the bi-adjacency matrix of this forest thus yields a matrix $\mathbf{B}$ with Property 12 required for our algorithm. Further, changing an entry in S or T, or replacing C or C' each will change only $O(1)$ edges in the forest (since we only reconnect two switch-vertices), i.e., $O(1)$ entries in matrix $\mathbf{B}$.

2.2 Maintaining Small Rank&Basis

The complexity of the submatrix-algorithm outlined in previous subsection scales in dimension n. Here we outline how to maintain small rank more efficiently (Theorem 1), and how to maintain a basis (Theorem 4) more efficiently for low-rank matrices. Our results rely on Lemma 10 (see preliminaries) by [38].

Maintaining just the rank is simple with the above lemma, and we start by outlining that algorithm. Then we extend the algorithm to also maintain a basis.

Maintaining Rank (Theorems 1 and 2). Let us assume for now that we just want to maintain $\text{rank}(\mathbf{A})$ up to some parameter k, i.e., we maintain $\min(\text{rank}(\mathbf{A}), k)$.

To do this, we first sample two matrices $\mathbf{L}, \mathbf{R}$ via Lemma 10 and then compute $\mathbf{L}^\top \mathbf{A} \mathbf{R}$. By Lemma 10 we have $\text{rank}(\mathbf{L}^\top \mathbf{A} \mathbf{R}) = \min(k, \text{rank}(\mathbf{A}\mathbf{R})) = \min(k, \text{rank}(\mathbf{A}))$. So by maintaining the rank of $\mathbf{L}^\top \mathbf{A} \mathbf{R}$, we maintain the rank of $\mathbf{A}$ up to k.

Further, by Lemma 10, there are only 2 non-zero entries per row in $\mathbf{L}, \mathbf{R}$. Thus an entry-update to $\mathbf{A}$ changes only 4 entries of $\mathbf{L}^\top \mathbf{A} \mathbf{R}$. Also observe that matrix $\mathbf{L}^\top \mathbf{A} \mathbf{R}$ is of size $O(k) \times O(k)$ due to width of $\mathbf{L}, \mathbf{R}$. Thus if we can maintain the rank of an $n \times n$ matrix in $U(n)$ update time, then we can now maintain $\min(\text{rank}(\mathbf{A}), k)$ in $4 \cdot (U(O(k))) = O(U(k))$ update time. We run $O(\log n)$ independent copies of this so at least one preserved the rank with high probability. Thus resulting in $\widetilde{O}(U(k))$ update time.

Next, we extend this to $\widetilde{O}(\text{rank}(\mathbf{A}))$ update time. To do this, we maintain the product $\mathbf{L}^\top \mathbf{A} \mathbf{R}$ for different powers of $k = 2^i$, $i = 0, ..., \log n$. Whenever the rank of $\mathbf{A}$ changes by constant factor, we initialize a new dynamic rank data structure for some $k = 2^i$ with $i = \lceil \log(\text{rank}(\mathbf{A})) \rceil + 1$. Until we must initialize a data structure for $k = 2^{i \pm 1}$, there must be at least 2^{i-1} updates. (Because each update can change the rank by at most 1.) So the initialization cost amortizes over $O(2^i)$ updates. This leads to an amortized update time of

$$\widetilde{O}(U(\text{rank}(\mathbf{A})) + P(\text{rank}(\mathbf{A}))/\text{rank}(\mathbf{A})),$$

where $P(n), U(n)$ are the preprocessing and update time complexity of a dynamic rank data structure for $n \times n$ matrices.

In case of column-updates, the same techniques works, because changing one column of $\mathbf{A}$ changes at most 2 columns of $\mathbf{L}^\top \mathbf{A} \mathbf{R}$. So there, too, the reduction has only a constant-factor blow-up for the number of updates.

Maintaining Basis (Theorems 4 and 5). When performing a column update, the rank can change by at most 1. In particular, if the updated column was already part of the basis then it might have to be removed or replaced. If the updated column was not part of the basis, then it might need to be added to the basis. In either case, the task of maintaining a basis reduces to the following problem: Given dynamic matrix $\mathbf{A}$ and a linearly-independent set of column-vectors $\mathbf{B}$, find if there is a column in $\mathbf{A}$ that can be added to $\mathbf{B}$.

Observe that if all columns of $\mathbf{A}$ are linearly dependent to $\mathbf{B}$, then any linear combination $\mathbf{A}v$ is also dependent to $\mathbf{B}$. Conversely, if $\mathbf{A}$ contains a column that is linearly independent, then there exists $v = e_i$ (a standard unit vector) such that $\mathbf{A}v$ is independent of $\mathbf{B}$. By Schwartz-Zippel-Lemma 9, this then implies[10] that for random vector v, we also have that $\mathbf{A}v$ is independent of $\mathbf{B}$ with high probability. Now assume we have a dynamic rank data structure supporting column updates. If adding column $\mathbf{A}v$ to $\mathbf{B}$ increases the rank, we know $\mathbf{A}$ contains an independent column. We can use this to binary-search for the first independent column in $\mathbf{A}$, by repeatedly zeroing out half the entries of v. Each step in the binary-search corresponds to adding and removing a column $\mathbf{A}v$ to $\mathbf{B}$. So we need $O(\log n)$ column-updates to find a new linearly independent column.

The last bottleneck is that computing $\mathbf{A}v$ takes $O(n^2)$ time. Thus we must maintain the possible $\mathbf{A}v$-products that occur during the binary-search, rather than computing them from scratch in each iteration.

During initialization, we precompute all $\mathbf{A}v$-products that could show up during binary-search. Then when a column of $\mathbf{A}$ changes, there are only $O(\log n)$ matrix-vector-products that must be updated. In particular, updating one product $\mathbf{A}v$ during a column-update to $\mathbf{A}$ takes only $O(z \log n)$ time, where z is the number of entries changed in $\mathbf{A}$ by the update.

In summary, let $U(n)$ be the update time of maintaining rank of a dynamic $n \times n$ matrix $\mathbf{H}$, subject to column-updates. Then we can maintain a column basis in time

$$O((U(n) + z) \log n).$$

This is $\widetilde{O}(n^{1.528} + z)$ update time using [32]. Since the reduction only relies on detecting whether $\operatorname{rank}(\mathbf{B})$ increases or decreases, we can run the rank data structure on $\mathbf{M}^\top \mathbf{B}$ for random matrix $\mathbf{M}$ from Lemma 10, improving the complexity dependence in n to $\operatorname{rank}(\mathbf{A})$. Similarly, instead of maintaining the products $\mathbf{A}v$ for the binary-search, we maintain $(\mathbf{M}^\top \mathbf{A})v$. Since an entry update to $\mathbf{A}$ changes only 2 entries of $O(\mathbf{M}^\top \mathbf{A})$, there is no complexity increase. This leads to Theorem 4, i.e., $\widetilde{O}(\operatorname{rank}(\mathbf{A})^{1.528} + z)$ update time.

3 Maintain Submatrices

In this section, we prove our reduction from maximum full-rank submatrix to singularity detection.

▶ **Theorem 6.** *Assume there is a data structure that detects when a dynamic $n \times n$ matrix $\mathbf{A}$ becomes singular for the first time, with $U(n)$ time per entry-update. Let $P(n)$ be the preprocessing time.*

Then there is a dynamic algorithm for maintaining a maximum full-rank submatrix (i.e., sets $I, J \subset [n]$ with $\det(\mathbf{A}_{I,J}) \neq 0$ and $|I| = |J| = \operatorname{rank}(\mathbf{A})$) after $O(P(n) + n^\omega)$ time preprocessing with $\widetilde{O}(U(n))$ entry-update time.
The reduction is randomized, correct with high probability, and works against an output-adaptive adversary.

As outlined in Section 2.1, the reduction is simple if we assume all non-zero entries of the matrix are random field elements, because then full-rankness is equivalent to perfect bipartite matching. The main challenge is to prove an equivalent result when the input matrix does

[10] Consider for instance the polynomial $\det([\mathbf{B}|\mathbf{A}v]^\top [\mathbf{B}|\mathbf{A}v])$ which is non-zero if and only if $[\mathbf{B}|\mathbf{A}v]$ (the matrix $\mathbf{B}$ with new column $\mathbf{A}v$ appended) is linearly independent.

not have random entries. To do this, we translate the graph reduction from Section 2.1 to matrix perspective. Hence we strongly recommend readers to first read Section 2.1 before reading Section 3.

The gadget-graph described in Section 2.1, is translated to a biadjacency matrix, whose determinant is analyzed in Section 3.1. That analysis requires careful handling of signs of permutations, which would not have been an issue in the random-entry case. In particular, we relate the determinant of the original input matrix $\mathbf{A}$ with the determinant of the biadjacency matrix that represents the binary trees in Section 2.1.

In Section 3.2, we analyze the determinant of the binary-tree-matrix and analyze the determinant behavior when performing the binary-search outlined in Section 2.1.

Lastly, in Section 3.3 we combine the results and prove that the binary-search yields an algorithm for maintaining a maximum full-rank submatrix.

3.1 Block Matrix Determinant

In this subsection, we prove the following lemma, which relates the determinants of matrix $\mathbf{A}$ and $\mathbf{B}$. Here matrix $\mathbf{A}$ will be the original input matrix whose submatrix we want to maintain. Matrix $\mathbf{B}$ will correspond to the biadjacency matrix of the binary-tree gadgets in Section 2.1. The lemma will be used in Section 3.3, so show that updates to $\mathbf{B}$ can be used to binary-search for suitable rows and columns in $\mathbf{A}$ that are linearly independent.

▶ **Lemma 13.** *Suppose $\mathbf{A} \in R^{n\times n}$ and $\mathbf{B} \in R^{m\times m}$ with $m \geq n$ for some commutative ring R. Let*

$$\mathbf{H} = \left[\begin{array}{ccc|ccc} & & & x_1 & & & 0\dots 0 \\ & \mathbf{A} & & & \ddots & & \vdots \\ & & & & & x_n & 0\dots 0 \\ \hline y_1 & & & & & & \\ & \ddots & & & & & \\ & & y_n & & \mathbf{B} & & \\ \hline 0 & \dots & 0 & & & & \\ \vdots & & & & & & \\ 0 & \dots & 0 & & & & \end{array}\right].$$

Then,

$$\det(\mathbf{H}) = \sum_{S,T\subseteq[n],|S|=|T|} (-1)^{|S|} \det(\mathbf{A}_{-S,-T}) \det(\mathbf{B}_{-T,-S}) x^S y^T.$$

In particular, $\det(\mathbf{H}) \neq 0$ *as a polynomial in* $x_1, \dots, x_n, y_1, \dots, y_n$ *iff there exists* $S, T \subseteq [n]$ *with* $\det(\mathbf{A}_{-S,-T}) \neq 0$ *and* $\det(\mathbf{B}_{-T,-S}) \neq 0$.

Proof. Let $\mathbf{X} \in R^{n\times m}, \mathbf{Y} \in R^{m\times n}$ be defined by $\mathbf{X}_{ii} = x_i$, $\mathbf{Y}_{ii} = y_i$ for $i \in [n]$ and $\mathbf{X}_{ij} = \mathbf{Y}_{ji} = 0$ for $i \neq j$.

$$\det(\mathbf{H}) = \det\begin{bmatrix}\mathbf{A} & \mathbf{X} \\ \mathbf{Y} & \mathbf{B}\end{bmatrix} = \det\left(\begin{bmatrix}\mathbf{I} & \mathbf{I} & & \\ & & \mathbf{I} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{A} & & & \mathbf{X} \\ & \mathbf{Y} & & \\ & & & \mathbf{B}\end{bmatrix}\begin{bmatrix}\mathbf{I} & \\ \mathbf{I} & \\ & \mathbf{I} \\ & \mathbf{I}\end{bmatrix}\right)$$

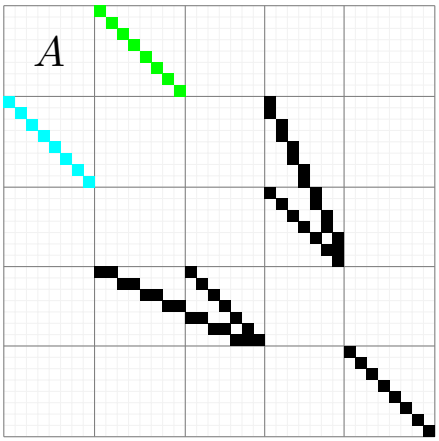

Figure 5 Shape of matrix $\mathbf{H}$ (Lemma 13), the green and cyan diagonals are the random $x_1, .., x_n$ and $y_1, ..., y_n$. The black boxes represent the non-zero entries of matrix $\mathbf{B}$, which is the biadjacency matrix of the forest given in Figure 2. The bottom right diagonal are the edges connecting left and right "switch-vertices." The black wedges represent the two binary trees.

By the Cauchy–Binet formula we can write $\det(\mathbf{H})$ as

$$\sum_{S,T\subseteq\binom{[2n+2m]}{n+m}} \det \begin{bmatrix} \mathbf{I} & \mathbf{I} & & \\ & & \mathbf{I} & \mathbf{I} \end{bmatrix}_{[n+m],S} \det \begin{bmatrix} \mathbf{A} & & \mathbf{X} & \\ & \mathbf{Y} & & \\ & & & \mathbf{B} \end{bmatrix}_{S,T} \det \begin{bmatrix} \mathbf{I} & \\ \mathbf{I} & \\ & \mathbf{I} \\ & \mathbf{I} \end{bmatrix}_{T,[n+m]}$$

Note that the first (resp. last) determinant is zero unless S (resp. T) selects exactly n of the first $2n$ and m of the last $2m$ columns (resp. rows). Thus, $\det(\mathbf{H})$ is equal

$$\sum_{S,T\subseteq[n],U,V\subseteq[m]} (-1)^{\alpha(S)+\alpha(T)+\beta(U)+\beta(V)} \det \begin{bmatrix} \mathbf{A}_{[n]\setminus S,[n]\setminus T} & & \mathbf{X}_{S,V} & \\ & \mathbf{Y}_{U,T} & & \\ & & & \mathbf{B}_{[m]\setminus U,[m]\setminus V} \end{bmatrix}$$

where we define

$$\alpha(S) = |\{(i,j) : i \in S, j \in [n] \setminus S, i < j\}| \text{ and } \beta(U) = |\{(i,j) : i \in [m] \setminus U, j \in U, i < j\}|.$$

The determinant is zero unless each of the four blocks is square, and $S = V$ and $T = U$. Note that for $S \subseteq [n]$, $\alpha(S) + \beta(S) = |S|(n - |S|)$. Thus, $(-1)^{\alpha(S)+\alpha(T)+\beta(U)+\beta(V)} = 1$, and

$$\det(\mathbf{H}) = \sum_{S,T\subseteq[n],|S|=|T|} (-1)^{|S|} \det(\mathbf{A}_{[n]\setminus S,[n]\setminus T}) \det(\mathbf{B}_{[m]\setminus T,[m]\setminus S}) x^S y^T$$

The $(-1)^{|S|}$ comes from swapping $\mathbf{X}_{S,V}$ and $\mathbf{Y}_{U,T}$ onto the diagonal. ◀

3.2 Binary-Search Gadget Matrix

By choosing $\mathbf{B}$, we can check if a certain subcollection of submatrices of $\mathbf{A}$ all have determinant zero via Lemma 13. Intuitively, matrix $\mathbf{B}$ represents the biadjacency matrix of the forest in Section 2.1, e.g., Figure 2. An example of what $\mathbf{H}$ looks like for such $\mathbf{B}$ is given in Figure 5.

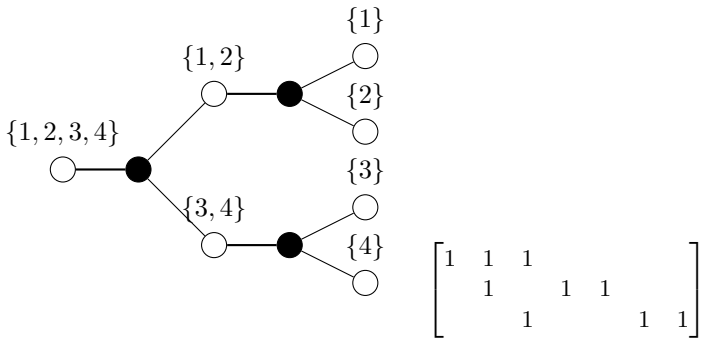

Figure 6 $\mathcal{T}_{1,4}$ and its biadjacency matrix.

We start by formally defining the graph gadget in this Section 3.2. Then in Section 3.3 we use those gadget to facilitate the binary-search for new rows/columns to be added to a full-rank submatrix.

First, we define the structure of the trees. Figure 6 shows their structure.

▶ **Definition 14.** *For integer $r \geq l \geq 1$, let $\mathcal{T}_{l,r}$ be a rooted tree defined recursively as follows:*

- *If $l = r$, then $\mathcal{T}_{l,r}$ consists only of a root labeled $\{l\}$*
- *If $l < r$, then $\mathcal{T}_{l,r}$ consists of a root labeled $\{l, \ldots, r\}$ with an (unlabeled) child vertex whose children are the roots of $\mathcal{T}_{l,\lfloor \frac{l+r}{2} \rfloor}$ and $\mathcal{T}_{\lfloor \frac{l+r}{2} \rfloor + 1,r}$ and*

Note that each vertex of $\mathcal{T}_{l,r}$ is labeled by the set of its descendant leaves.

We write $\mathcal{L}_{l,r}$ for the set of vertex-labels used in $\mathcal{T}_{l,r}$.

▶ **Definition 15.** *Let $\mathcal{L}_{l,r}$ be the labels of vertices in $\mathcal{T}_{l,r}$. Observe that*

$$\mathcal{L}_{l,r} = \begin{cases} \{\{l, \ldots, r\}\} \cup \mathcal{L}_{l,\lfloor \frac{l+r}{2} \rfloor} \cup \mathcal{L}_{\lfloor \frac{l+r}{2} \rfloor + 1,r} & l < r \\ \{\{l\}\} & l = r \end{cases}$$

In addition to the binary-trees, we also have additional "switch-vertices" at the bottom, used to "switch off" vertices. The graph in Definition 16 represents the two trees and the additional switch-vertices at the bottom. The vectors $a, b \in (\mathcal{L}_{1,n})^k$ in Definition 16 represent which k vertices of the tree are currently switched off, i.e., which tree vertices are connected to switch-vertices. In particular, a_i is the vertex switched off by connecting it to the ith switch-vertex.

▶ **Definition 16.** *For $a, b \in (\mathcal{L}_{1,n})^k$, where $0 \leq k \leq n$, let $B_n(a, b)$ be the graph consisting of $\mathcal{T}_{1,n}$ and a copy $\mathcal{T}'_{1,n}$ and an additional $2n$ vertices $u_1, \ldots, u_n, v_1, \ldots, v_n$ and additional edges $(u_1, \widehat{a}_1), \ldots, (u_k, \widehat{a}_k)$, $(v_1, \widehat{b}'_1), \ldots, (v_k, \widehat{b}'_k), (u_{k+1}, v_{k+1}), \ldots, (u_n, v_n)$, where $\widehat{a}_k$ is the vertex in $\mathcal{T}_{1,n}$ with label a_k, and $\widehat{b}'_k$ is the vertex in $\mathcal{T}'_{1,n}$ with label b_k.*

The biadjacency matrix of this graph will later be used as our matrix $\mathbf{B}$ in Lemma 13. Here we want to the first n rows/columns of $\mathbf{B}$ correspond to the leafs of the trees.

▶ **Definition 17.** *For $a, b \in (\mathcal{L}_{1,n})^k$, let $\mathbf{B}_n(a, b) \in \mathbb{R}^{(4n-2)\times(4n-2)}$ be the biadjacency matrix of $B_n(a, b)$, where the leaves of $\mathcal{T}_{1,n}$ are the first n vertices on the left side, and the leaves of $\mathcal{T}'_{1,n}$ are the first n vertices on the right side.*

The binary-tree and switch-vertices will be used to perform a binary-search. Since graph B has two trees, it will be useful to also define notation where we consider only a single tree with a few switched off vertices. (Observe that in the following definition, only the

switch-vertices attached to the tree exist. There are no switch-vertices that are not attached to some tree-vertex.) Further, as Lemma 13 argues about $\mathbf{B}_{-T,-S}$ (i.e., rows/column with index in T and S removed), we also want notation that describes the corresponding vertex deletion to the tree.

▶ **Definition 18.** *For $S \subseteq \{l,\dots,r\}$ and $a_1,\dots,a_k \in \mathcal{L}_{l,r}$, let Let $\mathcal{T}_{l,r}^{a_1,\dots,a_k}$ denote the tree $\mathcal{T}_{l,r}$ with k switch-vertices attached to vertices with labels $a_1,\dots,a_k$ respectively, and let $\mathcal{T}_{l,r}^{a_1,\dots,a_k}(-S)$ be $\mathcal{T}_{l,r}^{a_1,\dots,a_k}$ with vertices with label $\{i\}$ for $i \in S$ removed.*

The property $\det(\mathbf{B}) \neq 0$ corresponds to graph B having a perfect matching. The following Lemma 19 describes under which conditions our tree has a perfect matching.

▶ **Lemma 19.** *Suppose $i_1,\dots,i_k \in \{l,\dots,r\}$ Then, $\mathcal{T}_{l,r}^{\{i_1\},\dots,\{i_k\}}(-S)$ has a perfect matching iff $S = \{l,\dots,r\} \setminus \{i_1,\dots,i_k\}$ and $|S| = r+l-1-k$.*

Proof. Suppose $\mathcal{T}_{l,r}^{\{i_1\},\dots,\{i_k\}}(-S)$ has a perfect matching. Then, $i_1,\dots,i_k \notin S$, as otherwise their switch-vertices will be isolated. Furthermore, $i_1,\dots,i_k$ must be distinct, as otherwise the neighborhood of the k switch-vertices will have fewer than k vertices, which violates Hall's condition. By the requirement that both sides of the bipartition have the same number of vertices, we get that $|S| = r-l+1-k$. Thus, $S = \{l,\dots,r\} \setminus \{i_1,\dots,i_k\}$.

Conversely, suppose $S = \{l,\dots,r\} \setminus \{i_1,\dots,i_k\}$ where $|S| = r-l+1-k$, so that $i_1,\dots,i_k$ are distinct. We can obtain a perfect matching in $\mathcal{T}_{l,r}^{\{i_1\},\dots,\{i_k\}}(-S)$ by matching $\{i_1\},\dots,\{i_k\}$ to its switch-vertex, and the remaining labeled vertices to their child in $\mathcal{T}_{l,r}$. ◀

Lemma 19 concerns the case where the leafs of the tree are connected to switch-vertices. To facilitate the binary-search, we also want to connect one other tree-vertex to a switch-vertex. The following Lemma 20 states under which conditions such a graph has a perfect matching.

▶ **Lemma 20.** *Suppose $i_1,\dots,i_{k-1} \in \{l,\dots,r\}$ and $I \in \mathcal{L}_{l,r}$. Then, $\mathcal{T}_{l,r}^{\{i_1\},\dots,\{i_{k-1}\},I}(-S)$ has a perfect matching iff $S = \{l,\dots,r\} \setminus \{i_1,\dots,i_{k-1},i'\}$ for some $i' \in I$ such that $i_1,\dots,i_{k-1},i'$ are distinct.*

Proof. We will prove the lemma by induction on $r-l$. Let $m = \lfloor \frac{l+r}{2} \rfloor$.

Perfect Matching ⇒ condition on S. Suppose a perfect matchings exists. We must have $i_1,\dots,i_{k-1}$ distinct, as otherwise the neighborhood of the $k-1$ switch-vertices will have fewer than $k-1$ vertices, which violates Hall's condition.

- Suppose that $I \in \mathcal{L}_{m+1,r}$.
  The vertex labeled $\{l,\dots,r\}$ must be matched to its child, and so

  $$\mathcal{T}_{l,m}^{\mathcal{L}_{l,m} \cap \{\{i_1\},\dots,\{i_{k-1}\},I\}}(-S \cap \{l,\dots,m\}) \text{ and}$$

  $$\mathcal{T}_{m+1,r}^{\mathcal{L}_{m+1,r} \cap \{\{i_1\},\dots,\{i_{k-1}\}\}}(-S \cap \{m+1,\dots,r\})$$

  must have perfect matchings.
  By Lemma 19, $S \cap \{l,\dots,m\} = \{l,\dots,m\} \setminus \{i_1,\dots,i_{k-1}\}$.
  By the inductive hypothesis, $S \cap \{m+1,\dots,r\} = \{m+1,\dots,r\} \setminus \{i_1,\dots,i_{k-1},i'\}$ for some $i' \in I \setminus \{i_1,\dots,i_{k-1}\}$.
  Thus, $S = (S \cap \{l,\dots,m\}) \cup (S \cap \{m+1,\dots,r\}) = \{l,\dots,r\} \setminus \{i_1,\dots,i_{k-1},i'\}$ where $i_1,\dots,i_{l-1},i'$ are distinct.
- The case where $I \in \mathcal{L}_{l,m}$ is similar.

- Now, suppose $I = \{l, \ldots, r\}$. Then, the vertex labeled I must be matched to its switch-vertex, and its child matched to either the vertex labeled $\{l, \ldots, m\}$ or $\{m+1, \ldots, r\}$.
  - In the first case, both

    $$\mathcal{T}_{l,m}^{\mathcal{L}_{l,m} \cap \{\{i_1\}, \ldots, \{i_{k-1}\}, \{l, \ldots, m\}\}}(-S \cap \{l, \ldots, m\}) \text{ and}$$

    $$\mathcal{T}_{m+1,r}^{\mathcal{L}_{m+1,r} \cap \{\{i_1\}, \ldots, \{i_{k-1}\}\}}(-S \cap \{m+1, \ldots, r\})$$

    have perfect matchings.
    By the inductive hypothesis, $S \cap \{l, \ldots, m\} = \{l, \ldots, m\} \setminus \{i_1, \ldots, i_{k-1}, i'\}$ for some $i' \in \{l, \ldots, m\} \setminus \{i_1, \ldots, i_{k-1}\}$.
    By Lemma 19, $S \cap \{m+1, \ldots, r\} = \{m+1, \ldots, r\} \setminus \{i_1, \ldots, i_{k-1}\}$.
    Thus, $S = \{l, \ldots, r\} \setminus \{i_1, \ldots, i_{k-1}, i'\}$ for $i' \in \{l, \ldots, m\} \setminus \{i_1, \ldots, i_{k-1}\}$.
  - In the second case, both

    $$\mathcal{T}_{l,m}^{\mathcal{L}_{l,m} \cap \{\{i_1\}, \ldots, \{i_{k-1}\}\}}(-S \cap \{l, \ldots, m\}) \text{ and}$$

    $$\mathcal{T}_{m+1,r}^{\mathcal{L}_{m+1,r} \cap \{\{i_1\}, \ldots, \{i_{k-1}\}, \{m+1, \ldots, r\}\}}(-S \cap \{m+1, \ldots, r\})$$

    have perfect matchings.
    Again, by Lemma 19 and the inductive hypothesis, $S = \{l, \ldots, r\} \setminus \{i_1, \ldots, i_{k-1}, i'\}$ for $i' \in \{m+1, \ldots, r\} \setminus \{i_1, \ldots, i_{k-1}\}$.

In all cases we have $S = \{l, \ldots, r\} \setminus \{i_1, \ldots, i_{k-1}, i'\}$ for $i' \in I \setminus \{i_1, \ldots, i_{k-1}\}$.

Condition on $S \Rightarrow$ Perfect Matching. Now, we show the converse. Suppose $S = \{l, \ldots, r\} \setminus \{i_1, \ldots, i_{k-1}, i'\}$ for some $i' \in I$ such that $i_1, \ldots, i_{k-1}, i'$ are distinct. We can explicitly construct a perfect matching of $\mathcal{T}_{l,r}^{\{i_1\}, \ldots, \{i_{k-1}\}, I}(-S)$ as follows:

Take the symmetric difference of the unique perfect matching of $\mathcal{T}_{l,r}(-\{l, \ldots, r\})$ with the path from I to $\{i'\}$, and match $\{i_1\}, \ldots, \{i_{k-1}\}, I$ to their switch vertices. ◀

▶ **Lemma 21.** *$B_n((\{i_1\}, \ldots, \{i_{k-1}\}, I), (\{j_1\}, \ldots, \{j_{k-1}\}, J))$ has a matching covering all vertices except T on the left and S on the right iff $T = [n] \setminus \{i_1, \ldots, i_{k-1}, i'\}$ and $S = [n] \setminus \{j_1, \ldots, j_{l-1}, j'\}$ for some $i' \in I$ and $j' \in J$, and zero such matchings otherwise.*

Proof. Note that $B((a_1, \ldots, a_k), (b_1, \ldots, b_k))$ is isomorphic to a disjoint union of $\mathcal{T}_{1,n}^{a_1, \ldots, a_k}$ and $\mathcal{T}_{1,n}^{b_1, \ldots, b_k}$ and $n - k$ isolated edges.

$B((a_1, \ldots, a_k), (b_1, \ldots, b_k))$ has a perfect matching covering all vertices except T on the left and S on the right iff $\mathcal{T}_{1,n}^{a_1, \ldots, a_k}(-T)$ and $\mathcal{T}_{1,n}^{b_1, \ldots, b_k}(-S)$ both have perfect matchings. ◀

The matrix $\mathbf{B}_n((\{i_1\}, \ldots, \{i_{k-1}\}, I), (\{j_1\}, \ldots, \{j_{k-1}\}, J))_{-T,-S}$ is the biadjacency matrix of the graph $B_n((\{i_1\}, \ldots, \{i_{k-1}\}, I), (\{j_1\}, \ldots, \{j_{k-1}\}, J))$ after removing vertices S and T. Further, since that graph is a forest, if it has a perfect matching, then it has at most one perfect matching. Thus

$$\det(\mathbf{B}_n((\{i_1\}, \ldots, \{i_{k-1}\}, I), (\{j_1\}, \ldots, \{j_{k-1}\}, J))_{-T,-S}) \neq 0$$

if and only if the graph has a perfect matching, even without replacing the non-zero entries with random field elements. Schwartz-Zippel-Lemma is not needed. Thus we get Corollary 22.

▶ **Corollary 22.** $\det(\mathbf{B}_n((\{i_1\}, \ldots, \{i_{k-1}\}, I), (\{j_1\}, \ldots, \{j_{k-1}\}, J))_{-T,-S}) \neq 0$ *iff $T = [n] \setminus \{i_1, \ldots, i_{k-1}, i'\}$ and $S = [n] \setminus \{j_1, \ldots, j_{l-1}, j'\}$ for some $i' \in I$ and $j' \in J$.*

3.3 Maintaining Maximum Full-Rank Submatrix

In this section, we prove our reduction Theorem 6 for maintaining a maximum full-rank submatrix $\mathbf{A}_{I,J}$ ($I, J \subset [n]$) of dynamic matrix $\mathbf{A}$. The idea is to greedily add row/column indices to I, J, where the correct indices are located via a binary-search. First, observe that such a greedy procedure works, as proven in Lemma 23.

▶ **Lemma 23.** *Suppose* $\text{rank}(\mathbf{A}) > k$. *Then, for any* $k \times k$ *full-rank submatrix of* $\mathbf{A}$, *there is some row and column that can be added to make a* $(k+1) \times (k+1)$ *full-rank submatrix.*

Proof. Suppose $\text{rank}(\mathbf{A}_{I,J}) = k$ for $|I| = |J| = k$.

I indexes k linearly independent rows of $\mathbf{A}$, so there is some $i \notin I$ such that $I \cup \{i\}$ indexes $k+1$ linearly independent rows. In other words,

$$\text{rank}(\mathbf{A}_{I\cup\{i\},[n]}) = k+1$$

Now, J indexes k linearly independent columns of $\mathbf{A}_{I\cup\{i\},[n]}$, so there must be some j such that $J \cup \{j\}$ indexes $k+1$ linearly independent columns of $\mathbf{A}_{I\cup\{i\},[n]}$. Then,

$$\text{rank}(\mathbf{A}_{I\cup\{i\},J\cup\{j\}}) = k+1$$

as desired. ◀

Next, we show how to find new row/column indices via binary-search, when having access to a dynamic algorithm that detects when matrix $\mathbf{H}$ (Lemma 13) has $\det(\mathbf{H}) = 0$.

▶ **Lemma 24.** *Suppose* $\det(\mathbf{A}_{\{i_1,\ldots,i_k\},\{j_1,\ldots,j_k\}}) \neq 0$. *We can find* (i', j') *such that*

$$\det(\mathbf{A}_{\{i_1,\ldots,i_k,i'\},\{j_1,\ldots,j_k,j'\}}) \neq 0,$$

or determine that $\text{rank}(\mathbf{A}) = k$, *in* $O(\log n)$ *calls to a dynamic singularity detection algorithm.*

Algorithm 1 Augment via binary search.

Input : $\mathbf{A}$ and $i_1, \ldots, i_k, j_1, \ldots, j_k$ such that $\det(\mathbf{A}_{\{i_1,\ldots,i_k\},\{j_1,\ldots,j_k\}}) \neq 0$
Output: (i', j') such that $\det(A_{\{i_1,\ldots,i_k,i'\},\{j_1,\ldots,j_k,j'\}}) \neq 0$, or that none exist

if $\det(\mathbf{H}((i_1, \ldots, i_k, \{1, \ldots, n\}), (j_1, \ldots, j_k, \{1, \ldots, n\}))) = 0$ **then**
  return *none exist*
$l \leftarrow 1, r \leftarrow n$;
while $l < r$ **do**
  if $\det(\mathbf{H}((i_1, \ldots, i_k, \{l, \ldots, \lfloor \frac{l+r}{2} \rfloor\}), (j_1, \ldots, j_k, \{1, \ldots, n\}))) = 0$ **then**
    $l \leftarrow \lfloor \frac{l+r}{2} \rfloor + 1$;
  else
    $r \leftarrow \lfloor \frac{l+r}{2} \rfloor$;
$l' \leftarrow 1, r' \leftarrow n$;
while $l' < r'$ **do**
  if $\det(\mathbf{H}((i_1, \ldots, i_k, \{l, \ldots, r\}), (j_1, \ldots, j_k, \{l', \ldots, \lfloor \frac{l'+r'}{2} \rfloor\}))) = 0$ **then**
    $l' \leftarrow \lfloor \frac{l'+r'}{2} \rfloor + 1$;
  else
    $r' \leftarrow \lfloor \frac{l'+r'}{2} \rfloor$;
return (l, l')

Proof. See Algorithm 1. Here matrix $\mathbf{H}((i_1, ..., i_k, I), j_1, ..., j_k, J))$ is the matrix $\mathbf{H}$ from Lemma 13 where we use $\mathbf{B}((i_1, ..., i_k, I), (j_1, ..., j_k, J))$ (Definition 17).

By Lemma 13, $\det(\mathbf{H}) \neq 0$ iff there is $S, T \subset [n]$ with $\det(\mathbf{A}_{-S,-T}) \neq 0$ and $\det(\mathbf{B}_{-T,-S}) \neq 0$. Corollary 22 states that $\det(\mathbf{B}_{-T,-S}) \neq 0$ happens iff

$$T = [n] \setminus \{i_1, ..., i_k, i'\}, \quad S = [n] \setminus \{j_1, ..., j_k, j'\}$$

for some $i' \in I, j' \in J$. Thus $\det(\mathbf{H}) \neq 0$ iff there is $i' \in I, j' \in J$ with $0 \neq \det(\mathbf{A}_{-S,-T}) = \det(\mathbf{A}_{\{i_1,...,i_k,i'\},\{j_1,...,j_k,j'\}})$. Hence we can binary-search over sets I, J to find such i', j', which is precisely what we do in Algorithm 1.

Observe that each iteration changes only $O(1)$ entries of $\mathbf{B}$ to relink the switch-vertices. Further, it suffices to have a data structure that detects for the first time when $\det(\mathbf{H}) = 0$, because we can simply revert the previous update. The matrix $\mathbf{H}$ initially has $\det(\mathbf{H}) \neq 0$ when initializing with $\mathbf{B}((i_1, ..., i_k), (j_1, ..., j_k))$. ◀

We can now prove our main reduction Theorem 6.

▶ **Theorem 6.** *Assume there is a data structure that detects when a dynamic $n \times n$ matrix $\mathbf{A}$ becomes singular for the first time, with $U(n)$ time per entry-update. Let $P(n)$ be the preprocessing time.*

Then there is a dynamic algorithm for maintaining a maximum full-rank submatrix (i.e., sets $I, J \subset [n]$ with $\det(\mathbf{A}_{I,J}) \neq 0$ and $|I| = |J| = \operatorname{rank}(\mathbf{A})$) after $O(P(n) + n^\omega)$ time preprocessing with $\widetilde{O}(U(n))$ entry-update time.
The reduction is randomized, correct with high probability, and works against an output-adaptive adversary.

Proof. We maintain the invariant that $\det(\mathbf{A}_{\{i_1,...,i_k\},\{j_1,...,j_k\}}) \neq 0$ and no larger submatrix has nonzero determinant. Further, we always run a dynamic singularity algorithm on matrix $\mathbf{M}$ (as defined in Lemma 13) using $\mathbf{B}_n(I, J)$. When $\mathbf{A}_{I,J}$ is full-rank, then $\mathbf{M}$ is also full-rank.

Initialization. To initialize, first find initial maximum sets I, J with $\det(\mathbf{A}_{I,J}) \neq 0$ in $O(n^\omega)$ time. Then construct $\mathbf{M}$ and initialize the dynamic singularity data structure. Thus initialization takes $O(P(n) + n^\omega)$ time.

Update. If we update entry (i, j) and $\det(\mathbf{A}_{I,J})$ becomes zero (implying $i \in I$ and $j \in J$), then $\det(\mathbf{M})$ becomes zero. When we detect this, revert the update, remove i, j from I and J (update $\mathbf{M}$ accordingly), then update $\mathbf{A}$ (and $\mathbf{M}$) again. We now have $\det(\mathbf{M}) \neq 0$ since $\det(\mathbf{A}_{I,J}) \neq 0$ for the new smaller I, J. We perform one binary-search via Lemma 13 to see if maybe new indices can be added to I, J. If so, add the indices to I, J and update $\mathbf{M}$.

If we update entry (i, j) and $\det(\mathbf{A}_{I,J})$ remains nonzero, we also perform one binary-search via Lemma 13 to see if maybe new indices can be added to I, J. If so, add the indices to I, J and update $\mathbf{M}$.

Overall, we perform at most $O(\log(n))$ updates to the dynamic singularity algorithm by Lemma 24. So the update time is $O(U(n) \log n)$. ◀

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.

2 Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2005–2039. SIAM, 2025. doi:10.1137/1.9781611978322.63.

3 Anastasiia Alokhina and Jan van den Brand. Fully dynamic shortest path reporting against an adaptive adversary. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3027–3039. SIAM, 2024. doi:10.1137/1.9781611977912.108.

4 Emile Anand, Jan van den Brand, Mehrdad Ghadiri, and Daniel J. Zhang. The bit complexity of dynamic algebraic formulas and their determinants. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson, editors, *51st International Colloquium on Automata, Languages, and Programming, ICALP 2024, July 8-12, 2024, Tallinn, Estonia*, volume 297 of *LIPIcs*, pages 10:1–10:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.ICALP.2024.10.

5 Emile Anand, Jan van den Brand, and Rose McCarty. The structural complexity of matrix-vector multiplication. *CoRR*, abs/2502.21240, 2025. doi:10.48550/arXiv.2502.21240.

6 Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.7.

7 Sepehr Assadi, Aaron Bernstein, and Aditi Dudeja. Decremental matching in general graphs. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 11:1–11:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.11.

8 Sepehr Assadi, Sanjeev Khanna, and Peter Kiss. Improved bounds for fully dynamic matching via ordered ruzsa-szemerédi graphs. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2971–2990. SIAM, 2025. doi:10.1137/1.9781611978322.96.

9 Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani. Fully dynamic matching: -approximation in polylog update time. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3040–3061. SIAM, 2024. doi:10.1137/1.9781611977912.109.

10 Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully dynamic maximal matching in O (log n) update time. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 383–392. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.89.

11 Soheil Behnezhad. Dynamic algorithms for maximum matching size. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 129–162. SIAM, 2023. doi:10.1137/1.9781611977554.CH6.

12 Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 382–405. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00032.

13 Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New techniques and fine-grained hardness for dynamic near-additive spanners. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1836–1855. SIAM, 2021. doi:10.1137/1.9781611976465.110.

14 Aaron Bernstein, Aditi Dudeja, and Zachary Langley. A framework for dynamic matching in weighted graphs. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 668–681. ACM, 2021. doi:10.1145/3406325.3451113.

15 Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A deamortization approach for dynamic spanner and dynamic maximal matching. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1899–1918. SIAM, 2019. doi:10.1137/1.9781611975482.115.

16 Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2015. doi:10.1007/978-3-662-47672-7_14.

17 Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 692–711. SIAM, 2016. doi:10.1137/1.9781611974331.CH50.

18 Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 398–411. ACM, 2016. doi:10.1145/2897518.2897568.

19 Sayan Bhattacharya and Peter Kiss. Deterministic rounding of dynamic fractional matchings. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 27:1–27:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.27.

20 Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic $(1+\epsilon)$-approximate matching size in truly sublinear update time. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1563–1588. IEEE, 2023. doi:10.1109/FOCS57990.2023.00095.

21 Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. Dynamic algorithms for packing-covering lps via multiplicative weight updates. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1–47. SIAM, 2023. doi:10.1137/1.9781611977554.CH1.

22 Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. Dynamic matching with better-than-2 approximation in polylogarithmic update time. *J. ACM*, 71(5):33:1–33:32, 2024. doi:10.1145/3679009.

23 Joakim Blikstad and Peter Kiss. Incremental $(1-\epsilon)$-approximate dynamic matching in $o(\text{poly}(1/\epsilon))$ update time. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 22:1–22:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.22.

24 Jan van den Brand. Unifying matrix data structures: Simplifying and speeding up iterative algorithms. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 1–13. SIAM, 2021. doi:10.1137/1.9781611976496.1.

25 Jan van den Brand, Sebastian Forster, and Yasamin Nazari. Fast deterministic fully dynamic distance approximation. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1011–1022. IEEE, 2022. doi:10.1109/FOCS54457.2022.00099.

26 Jan van den Brand, Sebastian Forster, Yasamin Nazari, and Adam Polak. On dynamic graph algorithms with predictions. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3534–3557. SIAM, 2024. doi:10.1137/1.9781611977912.126.

27 Jan van den Brand, Yu Gao, Arun Jambulapati, Yin Tat Lee, Yang P. Liu, Richard Peng, and Aaron Sidford. Faster maxflow via improved dynamic spectral vertex sparsifiers. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 543–556. ACM, 2022. doi:10.1145/3519935.3520068.

28 Jan van den Brand and Adam Karczmarz. Deterministic fully dynamic SSSP and more. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 2312–2321. IEEE, 2023. doi:10.1109/FOCS57990.2023.00142.

29 Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and l1-regression in nearly linear time for dense instances. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 859–869. ACM, 2021. doi:10.1145/3406325.3451108.

30 Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 919–930. IEEE, 2020. doi:10.1109/FOCS46700.2020.00090.

31 Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 436–455. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00035.

32 Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 456–480. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00036.

33 Jan van den Brand and Thatchaphol Saranurak. Sensitive distance and reachability oracles for large batch updates. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 424–435. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00034.

34 Jan van den Brand and Daniel J. Zhang. Faster high accuracy multi-commodity flow from single-commodity techniques. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 493–502. IEEE, 2023. doi:10.1109/FOCS57990.2023.00036.

35 Moses Charikar and Shay Solomon. Fully dynamic almost-maximal matching: Breaking the polynomial worst-case time barrier. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 33:1–33:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.33.

36 Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 370–381. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00031.

37 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Almost-linear-time algorithms for maximum flow and minimum-cost flow. *Commun. ACM*, 66(12):85–92, 2023. doi:10.1145/3610940.

38 Ho Yee Cheung, Tsz Chiu Kwok, and Lap Chi Lau. Fast matrix rank algorithms and applications. *J. ACM*, 60(5):31:1–31:25, 2013. doi:10.1145/2528404.

39 Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *J. ACM*, 68(1):3:1–3:39, 2021. doi:10.1145/3424305.

40 Camil Demetrescu and Giuseppe F. Italiano. Trade-offs for fully dynamic transitive closure on dags: breaking through the o(n^2 barrier. *J. ACM*, 52(2):147–156, 2005. doi:10.1145/1059513.1059514.

41 Yichuan Deng, Zhao Song, and Omri Weinstein. Discrepancy minimization in input-sparsity time. *CoRR*, abs/2210.12468, 2022. doi:10.48550/arXiv.2210.12468.

42 Sally Dong, Yu Gao, Gramoz Goranci, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Guanghao Ye. Nested dissection meets ipms: Planar min-cost flow in nearly-linear time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 124–153. SIAM, 2022. doi:10.1137/1.9781611977073.7.

43 Gudmund Skovbjerg Frandsen and Peter Frands Frandsen. Dynamic matrix rank. *Theor. Comput. Sci.*, 410(41):4085–4093, 2009. doi:10.1016/J.TCS.2009.06.012.

44 Gudmund Skovbjerg Frandsen, Johan P. Hansen, and Peter Bro Miltersen. Lower bounds for dynamic algebraic problems. *Inf. Comput.*, 171(2):333–349, 2001. doi:10.1006/INCO.2001.3046.

45 Gudmund Skovbjerg Frandsen and Piotr Sankowski. Dynamic normal forms and dynamic characteristic polynomial. *Theor. Comput. Sci.*, 412(16):1470–1483, 2011. doi:10.1016/J.TCS.2010.11.049.

46 Yu Gao, Yang P. Liu, and Richard Peng. Fully dynamic electrical flows: Sparse maxflow faster than goldberg-rao. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 516–527. IEEE, 2021. doi:10.1109/FOCS52979.2021.00058.

47 Gramoz Goranci and Monika Henzinger. Efficient data structures for incremental exact and approximate maximum flow. In *ICALP*, volume 261 of *LIPIcs*, pages 69:1–69:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.69.

48 Gramoz Goranci, Adam Karczmarz, Ali Momeni, and Nikos Parotsidis. Fully dynamic algorithms for transitive reduction. In Keren Censor-Hillel, Fabrizio Grandoni, Joël Ouaknine, and Gabriele Puppis, editors, *52nd International Colloquium on Automata, Languages, and Programming, ICALP 2025, July 8-11, 2025, Aarhus, Denmark*, volume 334 of *LIPIcs*, pages 92:1–92:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ICALP.2025.92.

49 Fabrizio Grandoni and Virginia Vassilevska Williams. Faster replacement paths and distance sensitivity oracles. *ACM Trans. Algorithms*, 16(1):15:1–15:25, 2020. doi:10.1145/3365835.

50 Yong Gu and Hanlin Ren. Constructing a distance sensitivity oracle in o(nˆ2.5794 M) time. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 76:1–76:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.76.

51 Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set, maximum flow and maximum matching. In *SOSA*, pages 86–91. SIAM, 2021. doi:10.1137/1.9781611976496.10.

52 Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, Berkeley, CA, USA, October, 26-29, 2013*, pages 548–557. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.65.

53 Christopher Harshaw, Fredrik Sävje, Daniel A. Spielman, and Peng Zhang. Balancing covariates in randomized experiments using the gram-schmidt walk. *CoRR*, abs/1911.03071, 2019. arXiv:1911.03071.

54 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.

55 Monika Rauch Henzinger. A static 2-approximation algorithm for vertex connectivity and incremental approximation algorithms for edge and vertex connectivity. *J. Algorithms*, 24(1):194–220, 1997. doi:10.1006/JAGM.1997.0855.

56 Haotian Jiang, Tarun Kathuria, Yin Tat Lee, Swati Padmanabhan, and Zhao Song. A faster interior point method for semidefinite programming. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 910–918. IEEE, 2020. doi:10.1109/FOCS46700.2020.00089.

57 Haotian Jiang, Yin Tat Lee, Zhao Song, and Sam Chiu-wai Wong. An improved cutting plane method for convex optimization, convex-concave games, and its applications. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 944–953. ACM, 2020. doi:10.1145/3357713.3384284.

58 Ming-Yang Kao, Tak Wah Lam, Wing-Kin Sung, and Hing-Fung Ting. A decomposition theorem for maximum weight bipartite matchings. *SIAM J. Comput.*, 31(1):18–26, 2001. doi:10.1137/S0097539799361208.

59 Adam Karczmarz, Anish Mukherjee, and Piotr Sankowski. Subquadratic dynamic path reporting in directed graphs against an adaptive adversary. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1643–1656. ACM, 2022. doi:10.1145/3519935.3520058.

60 Adam Karczmarz and Piotr Sankowski. Fully dynamic shortest paths and reachability in sparse digraphs. In Kousha Etessami, Uriel Feige, and Gabriele Puppis, editors, *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*, volume 261 of *LIPIcs*, pages 84:1–84:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.84.

61 Adam Karczmarz and Piotr Sankowski. Sensitivity and dynamic distance oracles via generic matrices and frobenius form. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1745–1756. IEEE, 2023. doi:10.1109/FOCS57990.2023.00106.

62 Adam Karczmarz and Marcin Smulewicz. On fully dynamic strongly connected components. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, volume 274 of *LIPIcs*, pages 68:1–68:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.68.

63 Peter Kiss. Improving update times of dynamic matching algorithms from amortized to worst case. *CoRR*, abs/2108.10461, 2021. arXiv:2108.10461.

64 Peter Kiss. Deterministic dynamic matching in worst-case update time. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 94:1–94:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.94.

65 Peter Kiss. Deterministic dynamic matching in worst-case update time. *Algorithmica*, 85(12):3741–3765, 2023. doi:10.1007/S00453-023-01151-X.

66 Hung Le, Lazar Milenkovic, Shay Solomon, and Virginia Vassilevska Williams. Dynamic matching algorithms under vertex updates. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPIcs*, pages 96:1–96:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.96.

67 Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In Leonard J. Schulman, editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 457–464. ACM, 2010. doi:10.1145/1806689.1806753.

68 David Peleg and Shay Solomon. Dynamic (1 + eps)-approximate matchings: A density-sensitive approach. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 712–729. SIAM, 2016. doi:10.1137/1.9781611974331.CH51.

69 John H. Reif and Stephen R. Tate. On dynamic algorithms for algebraic problems. *J. Algorithms*, 22(2):347–371, 1997. doi:10.1006/JAGM.1995.0807.

70 Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 509–517. IEEE Computer Society, 2004. doi:10.1109/FOCS.2004.25.

71 Piotr Sankowski. Subquadratic algorithm for dynamic shortest distances. In Lusheng Wang, editor, *Computing and Combinatorics, 11th Annual International Conference, COCOON 2005, Kunming, China, August 16-29, 2005, Proceedings*, volume 3595 of *Lecture Notes in Computer Science*, pages 461–470. Springer, 2005. doi:10.1007/11533719_47.

72 Piotr Sankowski. Faster dynamic matchings and vertex connectivity. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 118–126. SIAM, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283397.

73 William T Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947.

74 David Wajc. Rounding dynamic matchings against an adaptive adversary. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 194–207. ACM, 2020. doi:10.1145/3357713.3384258.

75 Oren Weimann and Raphael Yuster. Replacement paths and distance sensitivity oracles via fast matrix multiplication. *ACM Trans. Algorithms*, 9(2):14:1–14:13, 2013. doi:10.1145/2438645.2438646.