Matrix sparsification and nested dissection over arbitrary fields

Noga Alon †
Raphael Yuster ‡

Abstract

The generalized nested dissection method, developed by Lipton, Rose, and Tarjan, is a seminal method for solving a linear system \( Ax = b \) where \( A \) is a symmetric positive definite matrix. The method runs extremely fast whenever \( A \) is a well-separable matrix (such as matrices whose underlying support is planar or avoids a fixed minor). In this work we extend the nested dissection method to apply to any non-singular well-separable matrix over any field. The running times we obtain essentially match those of the nested dissection method. An important tool is a novel method for matrix sparsification that preserves determinants and minors, and that guarantees that constant powers of the sparsified matrix remain sparse.

Key words. Gaussian elimination, linear system, nested dissection, matrix sparsification.

AMS subject classifications. 68W30, 15A15, 05C50

1 Introduction

Solving a linear system is the most basic, and perhaps the most important problem in computational linear algebra. Considerable effort has been devoted to obtaining algorithms that solve a linear system faster than the naive cubic implementation of Gaussian elimination.

For the rest of this introduction we assume that the system is given by \( Ax = b \), where \( A \) is a non-singular \( n \times n \) matrix over a field, \( b \) is an \( n \)-vector over that field, and \( x^T = (x_1, \ldots, x_n) \) is the vector of variables.

The fastest general algorithm for solving \( Ax = b \) was obtained by Bunch and Hopcroft [3], and by Ibarra, Moran, and Hui [11]. The algebraic complexity of both of these algorithms is \( O(n^\omega) \), where \( \omega < 2.376 \) is the matrix multiplication exponent [4].

If \( A \) is sparse and has only \( m \ll n^2 \) non-zero entries, faster algorithms exist. An important result of Wiedemann [24] asserts that if \( m = O(n) \) then a solution of \( Ax = b \) can be computed in \( \tilde{O}(n^2) \) time over finite fields. We note that solving sparse linear systems over finite fields has important applications in cryptography (see, e.g., [9]). Eberly et al. [5] solve \( Ax = b \) where \( A \) is any

---

\*This paper is based upon two conference proceedings papers. The first appeared in the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS’08) by the second named author [26]. The second appeared in the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS’10) by both authors [2].

†Schools of Mathematics and Computer Science, Tel Aviv University, Tel Aviv 69978, Israel and Institute for Advanced Study, Princeton, NJ 08540. E-mail: nogaa@post.tau.ac.il. Research supported in part by an ERC advanced grant, by a USA-Israeli BSF grant, by the Oswald Veblen Fund and by the Bell Companies Fellowship.

‡Department of Mathematics, University of Haifa, Haifa 31905, Israel. E-mail: raphy@math.haifa.ac.il

1The notation \( \tilde{O}(\cdot) \) is used to suppress polylogarithmic factors.
non-singular matrix with $O(n)$ nonzero bounded integer entries in bit complexity $\tilde{O}(n^{2.5})$. Spielman and Teng [21] obtained an almost linear time\(^2\) algorithm for approximately solving sparse symmetric diagonally-dominant linear systems.

In some important cases that arise in various applications, the matrix $A$ has additional structural properties in addition to being sparse. To make this notion more precise we need a definition. Let $A$ be an arbitrary $n \times n$ matrix. The *underlying graph* of $A$, denoted by $G_A$, is defined by the vertex set \{1, \ldots, n\} where, for $i \neq j$ we have an edge $ij$ if and only if $a_{i,j} \neq 0$ or $a_{j,i} \neq 0$ (the diagonal entries of $A$ play no role in the definition of $G_A$). Note that $G_A$ is always an undirected simple graph, while $A$ may or may not be symmetric.

The seminal *nested dissection* method of Lipton, Rose, and Tarjan [13], generalizing an earlier result of George [7], asserts that if $A$ is a symmetric positive definite matrix and the underlying graph $G_A$ has an appropriate separator tree (precise definitions will follow in the next section) then $Ax = b$ can be solved in $O(n^{\omega \beta})$ time, where $\beta \geq 1/2$ is a parameter of the separator tree. Notice that for $\beta < 1$ this implies, in particular, an algorithm whose algebraic complexity outperforms the $O(n^\omega)$ algorithm mentioned earlier. For example, it is known that $\beta = 1/2$ for planar graphs and for bounded genus graph (in these cases the separator tree can be constructed in $O(n \log n)$ time so one does not need to precondition its availability [14]). For graphs that exclude a fixed minor it is also known that $\beta = 1/2$ (although to initially construct a separator tree with this parameter requires $O(n^{1.5})$ time with present methods [1]).

However, the nested dissection method has several *algebraic* restrictions. The matrix needs to be *symmetric* (or Hermitian) and needs to be *positive definite*. The method does not apply to matrices over finite fields, or any other arbitrary field, unless they are assumed to be *symmetric pivoting-free* (exact definition will follow in the next section). Even if the matrix is, say, real, but either non-symmetric or non positive definite, the nested dissection method is not applicable.

Our main result in this paper is a new, modified version of the nested dissection method, that generalizes it so that all the above algebraic limitations are removed. Our method allows us to solve systems $Ax = b$ whenever $G_A$ has a $\beta$-separator tree. It applies to *any* such matrix, over *any* field; there are no algebraic restrictions.

There are several new techniques that are needed in order to obtain the main result. The first, and most notable, is a novel technique for matrix sparsification. This technique quickly sparsifies any given matrix so that after sparsification, each row and column has a bounded number of non-zero entries. The sparsification has the property that it is easy to derive the rank of the original matrix from the rank of the sparsified matrix, and the determinants, as well as certain minors of both matrices, are the same. Notice that the sparsified matrix has the property that constant powers of it remain sparse, which is not necessarily the case for the original matrix, even if it is sparse.

Another important technique is a new idea of partitioning a non-singular linear system into smaller (possibly rectangular) systems having unique solutions, and combining the solutions of these smaller systems into a solution of the original one.

We now formally state the main results that we obtain. We assume that the matrix of coefficients is non-singular (if this is not the case, our method detects this fact). The result here is stated for the

\(^2\)From here and throughout, unless otherwise noted, *time* means algebraic complexity. That is, each arithmetic operation in the field requires constant time.
specific graph families of planar graphs, bounded genus graphs, and $H$-minor free graphs, because we prefer to be concrete about running times. Notice, however, that the result applies to other hereditary families of graphs that exhibit small separators (see, e.g., [17, 22]). The full generic statement of the result appears in Section 3.

**Theorem 1.1** Let $A \in \mathbb{F}^{n \times n}$ be a non-singular matrix and let $b \in \mathbb{F}^n$. If $G_A$ is planar or has bounded genus, then $Ax = b$ can be solved in $O(n^{\omega/2}) < O(n^{1.19})$ time. If $G_A$ excludes some fixed minor, then $Ax = b$ can be solved in $O(n^{3\omega/(\omega+3)}) < O(n^{1.326})$ time. For the fields $\mathbb{R}, \mathbb{Q}, \mathbb{C}$ the algorithm is deterministic. For arbitrary fields (and, in particular, for finite fields) it is a randomized Las Vegas algorithm.

As noted earlier, the stated running times are given under the assumption that each arithmetic operation in the field takes $O(1)$ time (namely, the algorithms are measured in terms of their algebraic complexity). If $\mathbb{F}$ is a finite field whose number of elements is polynomial in $n$, then it is, indeed, true that each arithmetic operation takes $O(\log n)$ time (measured in bit operations) and hence the running times in Theorem 1.1 also measure actual bit complexity, up to a logarithmic factor. In the case where $A$ and $b$ have bounded integer entries independent of $n$ (and the system is to be solved over $\mathbb{Q}$) it is not difficult to show, using standard techniques, that the running times in Theorem 1.1, when measured in bit complexity, are multiplied by an additional $O(n)$ factor. Thus, for example, for the case of planar graphs we obtain an $O(n^{2.19})$ algorithm measured in bit complexity. For $H$-minor free graphs one can exploit a tradeoff with the combinatorial part of the algorithm and also solve the problem in $O(n^{2.19})$ bit complexity. Notice that this is quite close to the obvious $\Omega(n^2)$ lower bound, as the output may consist of $n$ rationals, each having numerators and denominators with $\Omega(n)$ digits.

Another minor point is that the stated running time $O(n^{\omega/2})$ assumes that $\omega > 2$, since the algorithm has an ingredient that runs in $\Theta(n \log n)$ time. Thus, if $\omega = 2$, the running time for planar graphs and bounded genus graphs is $O(n \log n)$, and not $O(n)$.

The rest of this paper is organized as follows. In Section 2 we establish the necessary tools for the proof of the main result. This section is split into five parts according to the nature of the tools used: matrix sparsification, linear algebra, graph theory, vertex splitting, and nested dissection. Section 3 contains the proof of the main result. This section is also split into parts in sync with the sub-algorithms applied in order to achieve the main result. Section 4 contains some concluding remarks.

## 2 Tools

### 2.1 Matrix sparsification

Throughout this paper $\mathbb{F}$ denotes an arbitrary field, unless stated otherwise. Recall that for a square matrix $X$, the **minor** $M_{i,j}(X)$ is the determinant of the matrix obtained from $X$ by removing row $i$ and column $j$. If $i = j$ then $M_{i,i}(X) = M_i(X)$ is the $i$'th **principal minor** of $X$. An important ingredient in the proof of our results is the following theorem, proved by the second author in [26].
Theorem 2.1 Let $A \in \mathbb{F}^{n \times n}$ be any matrix with $m$ non-zero entries. Another matrix $S \in \mathbb{F}^{n+2t \times n+2t}$ can be constructed in $O(m)$ time and which has the following properties:

1. $\det(A) = \det(S)$. Moreover, for all $i = 1, \ldots, n$ and $j = 1, \ldots, n$, $M_{i,j}(A) = M_{i,j}(S)$.
2. $\text{rank}(S) = \text{rank}(A) + 2t$.
3. $t = O(m)$.
4. Each row and column of $S$ has at most three non-zero entries.

Proof: We assume that $A$ is represented in a sparse form. That is, $A$ is given by $n$ lists $R_1, \ldots, R_n$ representing the rows where each element in $R_i$ is a pair $(j, a_{i,j})$ and $a_{i,j} \neq 0$. Each $R_i$ is sorted by the column indices. For implementation reasons we will maintain cross-referencing pointers between $(j, a_{i,j})$ in $R_i$ and $(i, a_{j,i})$ in $R_j$. Since the lists are initially sorted by the column indices, creating these cross-reference pointers requires $O(m)$ time.

It will be convenient to assume that $A$ has the property that $a_{i,j} \neq 0$ if and only if $a_{j,i} \neq 0$ for each pair of distinct indices $i, j$. We can always assume this since if only one of them is 0 we can rename it 0*, and at the end of the algorithm dispose of any $(i, 0*)$ entry in any list. As this assumption only increases $m$ by a factor of at most 2, it no effect on the claimed running time.

At step $t$ of the algorithm, the current matrix is denoted by $S_t$, its current order will be $n + 2t$, and its elements are denoted by $s_{i,j}^t$ or by $s_{i,j}$ if $t$ is clear from the context. Just like $A$, the matrix $S$ is represented by lists $R_i$ for $i = 1, \ldots, n + 2t$. We initially set $S_0 = A$. A single step of the algorithm constructs $S_{t+1}$ from $S_t$ by increasing the number of rows and columns of $S_t$ by 2 and by modifying constantly many entries of $S_t$. The algorithm halts when each row list of $S_t$ has at most three entries. Thus, in the final matrix $S_t$ we have that each row and column has at most 3 non-zero entries. At any step of the algorithm we will have $\det(S_{t+1}) = \det(S_t)$, $\text{rank}(S_{t+1}) = \text{rank}(S_t) + 2$, and $M_{i,j}(S_{t+1}) = M_{i,j}(S_t)$ for $i = 1, \ldots, n$ and $j = 1, \ldots, n$. Hence, in any step we will have $\det(S_t) = \det(A)$, $\text{rank}(S_t) = \text{rank}(A) + 2t$, and $M_{i,j}(S_t) = M_{i,j}(A)$ for $i = 1, \ldots, n$ and $j = 1, \ldots, n$.

At any step of the algorithm we denote by $r_i$ the number of elements in $R_i$ and denote by $F_t$ the set of indices of rows of $S_t$ that contain at least four non-zero entries. Notice that $F_0$ is initially constructed in $O(m)$ time. To assist in counting the number of steps of the algorithm we set

$$c_t = \sum_{i \in F_t} (r_i - 3).$$

Observe that initially $c_0 < m$.

We now describe a single step. As long as $F_t$ is not empty, let $i$ denote the first element of $F_t$. Hence, $r_i = |R_i| \geq 4$. Let $u$ and $v$ be two other distinct indices for which $s_{i,u} \neq 0$ and $s_{i,v} \neq 0$. We can locate $u$ and $v$ in constant time by looking at the first three entries of $R_i$ (we have to look at three entries since one of the first two entries might represent the diagonal element $s_{i,i}$). Let $X$ denote the $3 \times 3$ principal sub-matrix of $S_t$ obtained from rows $\{i, u, v\}$ and columns $\{i, u, v\}$. The matrix $S_{t+1}$ is constructed from $S_t$ as follows. The four elements $s_{i,u}, s_{i,v}, s_{u,i}, s_{v,i}$ of $X$ are replaced with zero. In other words, $s_{i,u}^{t+1} = s_{i,v}^{t+1} = s_{u,i}^{t+1} = s_{v,i}^{t+1} = 0$. The other 5 elements of $X$ remain the same. Two new rows are added and two new columns are added. The non-zero entries
\[
S_t = \begin{pmatrix}
  M_1 & M_2 \\
  s_{v,v} & s_{v,u} & s_{v,i} \\
  s_{u,v} & s_{u,u} & s_{u,i} \\
  s_{i,v} & s_{i,u} & s_{i,i}
\end{pmatrix}
\]

\[
S_{t+1} = \begin{pmatrix}
  M_1 & M_2 & 0 \\
  s_{v,v} & s_{v,u} & 0 & 0 & s_{v,i} \\
  s_{u,v} & s_{u,u} & 0 & 0 & s_{u,i} \\
  0 & 0 & -1 & 0 & 1 \\
  s_{i,v} & s_{i,u} & 0 & -1 & 0 
\end{pmatrix}
\]

Figure 1: A single step of the algorithm

in row \(n + 2t + 1\) are \(s_{n+2t+1,i} = -1\) and \(s_{n+2t+1,n+2t+2} = 1\). The non-zero entries in row \(n + 2t + 2\) are \(s_{n+2t+2,u} = s_{i,u}\), \(s_{n+2t+2,v} = s_{i,v}\), and \(s_{n+2t+2,n+2t+1} = -1\). The non-zero entries in column \(n + 2t + 1\) are \(s_{n+2t+2,n+2t+1} = 1\) and \(s_{n+2t+1,n+2t+1} = -1\). The non-zero entries in column \(n + 2t + 2\) are \(s_{u,n+2t+2} = s_{i,u}\), \(s_{v,n+2t+2} = s_{i,v}\), and \(s_{n+2t+1,n+2t+2} = 1\). This construction is visualized in Figure 1, where we assume, for the sake of clarity that \(i = n + 2t\), \(u = n + 2t - 1\), \(v = n + 2t - 2\), and hence \(X\) is the bottom right \(3 \times 3\) sub-matrix of \(S_t\).

Modifying the row lists \(R_u, R_v, R_i\) to reflect the changes between \(S_t\) and \(S_{t+1}\) takes constant time as there are at most two removals of elements that are directly pointed (here we use the cross-referencing pointers) and one insertion at the end of each list. Creating the new lists \(R_{n+2t+1}, R_{n+2t+2}\) requires constant time as well. Notice that the number of elements in \(R_i\) decreased by 1, since both \(s_{i,u}\) and \(s_{i,v}\) became zero in \(S_{t+1}\) and they were not zero in \(S_t\), and since \(s_{i,n+2t+1}\) is a new non-zero element which equals 1 in \(S_{t+1}\). Thus, \(r_i\) decreased by 1. If we now have \(r_i \leq 3\) we delete \(i\) from the head of \(F_i\) to obtain \(F_{t+1}\). Otherwise, \(F_{t+1} = F_t\) remains intact. Notice also that \(r_u\) did not change since \(s_{u,i}\) in \(S_t\) was “moved” to \(s_{u, n+2t+2}\) in \(S_{t+1}\). Similarly, \(r_v\) did not change. The value of \(r_{n+2t+1}\) is initialized to 2 and the value of \(r_{n+2t+2}\) is initialized to 3. It follows that \(c_{t+1} < c_t\) proving that the algorithm has less than \(m\) steps until it terminates. It remains to prove the following lemma:

**Lemma 2.2** \(\det(S_{t+1}) = \det(S_t), \ rank(S_{t+1}) = rank(S_t) + 2, \) and \(M_{i,j}(S_{t+1}) = M_{i,j}(S_t)\) for \(i = 1, \ldots, n\) and \(j = 1, \ldots, n\).

**Proof:** Consider first the matrix \(S'\) obtained from \(S_t\) by adding two rows and two columns all of which are zero except for the lower \(2 \times 2\) part, as depicted in Figure 2. Clearly, \(\det(S') = \det(S_t), \ rank(S') = rank(S_t) + 2, \) and the claimed minors are not affected. It remains to show that \(\det(S') = \det(S_{t+1}), \ rank(S') = rank(S_{t+1}), \) and \(M_{i,j}(S') = M_{i,j}(S_{t+1})\) for \(i = 1, \ldots, n\) and \(j = 1, \ldots, n\).
Let $R(j)$ denote the $j$th row of $S'$ and let $C(j)$ denote the $j$th column of $S'$. The following sequence of elementary operations does not change the determinants, the claimed minors, nor the rank, as they all involve multiplying one of the two newly added rows or columns by a scalar and add it to some original row or column. Furthermore, these operations transform $S'$ to $S_{t+1}$.

$$\begin{align*}
R(v) &= R(v) + s_{v,i} R(n + 2t + 1) \\
R(u) &= R(u) + s_{u,i} R(n + 2t + 1) \\
C(v) &= C(v) - s_{i,v} C(n + 2t + 1) \\
C(u) &= C(u) - s_{i,u} C(n + 2t + 1) \\
R(i) &= R(i) - R(n + 2t + 2) \\
C(i) &= C(i) - C(n + 2t + 2).
\end{align*}$$

2.2 Linear algebra

Let $Ax = b$ be a system of linear equations, where $A \in \mathbb{F}^{n \times n}$, $b \in \mathbb{F}^n$ and $x^T = (x_1, \ldots, x_n)$. Unless otherwise stated, $A$ is assumed to be non-singular. Our goal is to find the unique solution of the system, which is denoted by $c^T = (c_1, \ldots, c_n)$.

Let $B = [A|b]$ be the $n \times (n+1)$ matrix obtained by adding $b$ as the rightmost column of $A$. For $i = 1, \ldots, n+1$, let $B_i$ be the matrix obtained from $B$ by removing column $i$ (hence $B_{n+1} = A$). The following is an immediate consequence of Cramer’s rule [10] and the fact that permuting columns only changes the sign of the determinant.

**Fact 2.3** $c_i = \pm \det(B_i) \det(A)^{-1}$.

**Lemma 2.4** Let $Q = B^T B$. Then $\det(B_i)^2 = M_i(Q)$. Also, $\det(A)^2 = M_{n+1}(Q)$.

**Proof:** $B_i^T B_i$ is precisely the matrix obtained from $Q$ by removing row and column $i$. Likewise, $A^T A$ is obtained from $Q$ by removing the last row and the last column.

It follows from Fact 2.3 and from Lemma 2.4 that by computing the minors $M_1(Q), \ldots, M_{n+1}(Q)$, we obtain $c_1^2, \ldots, c_n^2$. But we are interested in $c_1, \ldots, c_n$, not in their squares. For this purpose, we
use the following interpolation argument. Let $a \in \mathbb{F}^n$ be the sum of the columns of $A$. Consider the linear system $Ay = (a + b)$. Notice that $(c_1, \ldots, c_n)$ is a solution of $Ax = b$ if and only if $(c_1 + 1, \ldots, c_n + 1)$ is a solution of $Ay = (a + b)$. So, by the same argument, we can compute the squares of the coordinates of the solution of $Ay = (a + b)$ which are $(c_1 + 1)^2, \ldots, (c_n + 1)^2$. To obtain $c_i$ we notice that if the field has characteristic 2 then $c_i^2$ already uniquely defines $c_i$. For other fields, we notice that $c_i = ((c_1 + 1)^2 - c_i^2 - 1)^{-1}$.

To summarize, we have shown that solving the system $Ax = b$ amounts to computing all the principal minors $M_1(Q), \ldots, M_{n+1}(Q)$ of the matrix $Q = B^T B$ where $B = [A|b]$. Computing all these principal minors quickly turns out to be a non-trivial task. For this purpose, we need to state and prove a few additional linear algebraic claims.

Although we are interested in solving systems $Ax = b$ where $A$ is a square non-singular matrix, we will need, in the course of our algorithm, to consider a more general setting. Let $Rx = h$ be a system of linear equations where $R \in \mathbb{F}^{m+p,n}$, $b \in \mathbb{F}^{m+p}$, and assume that the system has a unique solution (in particular, $\text{rank}(R) = n$). As before, let $B = [R|h]$ be the matrix obtained by adding $h$ as the last column of $R$. Notice that the dimensions of $B$ are $(n+p) \times (n+1)$. For $i = 1, \ldots, n+1$, let $B_i$ be the matrix obtained from $B$ by removing column $i$. We would like to generalize the above observation regarding the minors $M_i(B^T B)$ and the squares of the solutions of the system. It is not difficult to show that such a generalization holds for fields which satisfy $\text{rank}(R^T R) = \text{rank}(R)$ (such as the reals or the rationals). However, we require a generalization that applies to all fields.

**Lemma 2.5** Let $D$ be a diagonal matrix of order $n+p$, with elements taken from a field $\mathbb{F}' \supset \mathbb{F}$, and so that $\text{rank}(R^T DR) = \text{rank}(R) = n$. Let $Q = B^T DB$. Then $c_i^2 = M_i(Q)M_{n+1}(Q)^{-1}$.

**Proof:** We first prove that $M_{n+1}(Q) \neq 0$. Indeed, since $\text{rank}(R) = n$ we have that $\text{rank}(R^T DR) = n$ and hence $R^T DR$ is an $n \times n$ non-singular matrix. Since $B = [R|h]$ we have that $R^T DR$ is just the matrix obtained from $Q = B^T DB$ by removing the bottom row and rightmost column. As $R^T DR$ is non-singular, its determinant $M_{n+1}(Q)$ is nonzero.

For the rest of the proof, assume, without loss of generality, that the first $n$ rows of $R$ are linearly independent, and let $A$ denote the $n \times n$ nonsingular sub-matrix of $R$ corresponding to these rows. Let $B' = [A|h']$ where $h'$ is the truncation of $h$ to the first $n$ coordinates. Similarly, define $B'_i$ to be the matrix obtained from $B'$ by removing column $i$, and let $Q' = B'^T B'$. We already know from Fact 2.3 and from Lemma 2.4 that $c_i^2 = M_i(Q')M_{n+1}(Q')^{-1}$ and that $\text{det}(B'_i)^2 = M_i(Q')$. We will show that there is a scalar $f \neq 0$ so that $M_i(Q) = fM_i(Q')$, thereby obtaining the claimed result. Notice that if we show such a scalar $f$ exists then it must be nonzero since $M_{n+1}(Q) \neq 0$.

For a subset $J$ of $n$ row indices of $B$, let $B(J)$ denote the corresponding $n \times (n+1) \text{ sub-matrix}$, and let $B_i(J)$ denote the corresponding $n \times n$ matrix where column $i$ is removed. Now the following two cases may occur. Either $B(J)$ does not have rank $n$, in which case $\text{det}(B_i(J)) = 0$ for all $i$, or else the rows of $B(J)$ span the same $n$-dimensional subspace as $B'$ does (notice that $B' = B(\{1, \ldots, n\})$). In particular, for each $J$ there exists a constant $f_J$ so that $\text{det}(B_i(J)) = f_J \cdot \text{det}(B'_i)$ for all $i = 1, \ldots, n+1$.

Let $D_J$ denote the $n \times n$ diagonal matrix obtained from $D$ by selecting only the rows and columns corresponding to $J$. Define $f = \sum_J f_J^2 \text{det}(D_J)$. By the Cauchy-Binet formula (see, e.g., [10]),

$$M_i(Q) = \text{det}(B_i^T DB_i) = \sum_J \text{det}(B_i(J))^2 \text{det}(D_J) = \sum_J f_J^2 \text{det}(B'_i)^2 \text{det}(D_J) = f \cdot \text{det}(B'_i)^2 = fM_i(Q').$$
Recall that in $\mathbb{R}$ and $\mathbb{Q}$ we always have $\text{rank}(R^TR) = \text{rank}(R)$, so in these cases the diagonal matrix $D$ in Lemma 2.5 is simply irrelevant (in other words, just take $D = I$). Over $\mathbb{C}$ we can work with the conjugate transpose $R^*$ instead of $R^T$. Now, since $\text{rank}(R^TR) = \text{rank}(R)$ the result remains the same, except that we now obtain $|c_i|^2$ instead of $c_i^2$, and we can use the interpolation trick to recover $c_i$. But for arbitrary fields, how do we make sure that such a $D$ exists, and how do we compute one efficiently?

Existence is trivial; since $\text{rank}(R) = n$ there are $n$ rows of $R$ that are linearly independent, so let $J$ denote the subset of indices corresponding to these rows, and let $D$ be the diagonal matrix with $1$ in the diagonal positions corresponding to $J$ and zero otherwise.

Although to establish Lemma 2.5 we just require that $D$ has the property that $\text{rank}(R^TDR) = \text{rank}(R)$, our algorithm will require $R^TDR$ to have a much stronger property, which we now define.

Gaussian elimination of symmetric matrices can be performed on rows and columns simultaneously, as long as there is no pivoting. In step $i$ of the elimination, we already have that the top $i \times i$ block is a diagonal matrix. Assuming the entry in location $(i, i)$ (the pivot) is nonzero, we eliminate all the elements below it and to its right (simultaneously, as the matrix is symmetric). If the element in location $(i, i)$ is zero, one has to perform pivoting, namely, permute row (and column) $i$ with some other row (and column) $j > i$ so as to obtain a nonzero entry in location $(i, i)$.

We say that a symmetric matrix $C$ is pivoting-free if no pivoting occurs during the elimination process. In particular, $C$ is non-singular. Notice also that if $C$ is pivoting-free then the elimination process produces a decomposition $C = LDL^T$ where $L$ is a unit lower triangular matrix and $D$ is a diagonal matrix. Notice that, over $\mathbb{R}$ (resp. $\mathbb{C}$), any symmetric (resp. Hermitian) positive definite matrix is pivoting-free. (In this case the decomposition is known as the Cholesky decomposition). Hence, in this sense, pivoting-freeness is the analogue of symmetric positive-definite matrices for arbitrary fields.

In $\mathbb{R}$ ($\mathbb{C}$), for any matrix $A$ with full column rank $n$ we have that $A^TA$ (resp. $A^*A$) is symmetric (resp. Hermitian) positive definite, and thus pivoting-free. This, however, is not true for general fields. For example, over $\mathbb{F}_p$ it may be that already $(A^TA)(1, 1) = 0$ while $A$ is non-singular.

Our goal is to choose the diagonal matrix $D$ so that $R^TDR$ is pivoting-free with high probability. This can be proved in several ways, but we prefer the following proof.

**Lemma 2.6** Let $R \in \mathbb{F}^{n+p,n}$ have $\text{rank}(R) = n$. There is an $O(n + p)$ time algorithm that, with probability $1 - 1/n^2$, constructs a diagonal matrix $D$ of order $n + p$ so that $R^TDR$ is pivoting-free. If $\mathbb{F}$ has $q \leq n^4$ elements then the diagonal entries of $D$ are chosen at random from an extension field $\mathbb{F}'$ having at least $n^4$ elements. Otherwise, the diagonal entries of $D$ are chosen at random from some subset of $n^4$ elements of $\mathbb{F}$.

**Proof:** Consider the symbolic diagonal matrix $D = \text{diag}(x_1, \ldots, x_{n+p})$. Let $C = R^TDR$ be an $n \times n$ matrix over $\mathbb{F}[x_1, \ldots, x_{n+p}]$ and let $C_i$ be the top $i \times i$ block of $C$. We claim that $\text{det}(C_i) \neq 0$. Indeed, if $R_i$ denotes the first $i$ columns of $R$ then $C_i = R_i^TDR_i$. By the Cauchy-Binet formula, $\text{det}(C_i) = \sum_J \text{det}(R_i(J))^2 \text{det}(D_J)$ where $J$ ranges over all $i$-subsets of indices from $\{1, \ldots, n + p\}$ and $R_i(J)$ (resp. $D_J$) is the $i \times i$ sub-matrix of $R_i$ (resp. $D$) corresponding to these indices. Since $\text{det}(D_J)$ is just the monomial corresponding to the product of the variables with index in $J$, and all
these monomials are distinct, it suffices to prove that there exists \( J \) so that \( \text{det}(R_i(J)) \neq 0 \). Indeed, \( R \) has full column rank \( n \), so \( R_i \) has full column rank \( i \). Hence, there must be at least one \( i \times i \) sub-matrix of \( R_i \) that is non-singular. Since \( J \) ranges over all such matrices, the claim follows.

Having proved that \( \text{det}(C_i) \neq 0 \), we proceed as follows. Each \( \text{det}(C_i) \) is a nonzero polynomial of degree \( i \), and in particular, the product \( P(x_1, \ldots, x_{n+p}) = \pi_{i=1}^n \text{det}(C_i) \) is a polynomial of degree less than \( n^2 \). By the results of Schwartz [19] and Zippel [28], if \( S \subset \mathbb{F} \) has at least \( \lambda n^2 \) elements then a random assignment of elements of \( S \) to the variables yields a nonzero value with probability at least \( 1 - 1/\lambda \). We will use \( \lambda = n^2 \). If \( \mathbb{F} \) is finite and has only \( q \leq n^4 \) elements, we can use an extension field \( \mathbb{F}' \supset \mathbb{F} \) with at least \( n^4 \) elements. In order to construct an extension field \( \mathbb{F}' \) with \( q' \geq n^4 \) elements (notice that here \( r = O(\log n) \)) we just need to construct an irreducible polynomial of degree \( r \) over \( \mathbb{F} \). A probabilistic Las Vegas algorithm that performs this task in \( \tilde{O}(r^2 + r \log q) \) time (here \( \tilde{O} \) indicates an implicit polylogarithmic factor in \( r \)) is given in [20]. Thus, the time to construct \( \mathbb{F}' \) is not larger than the \( O(n+p) \) time required to randomly generate \( D \).

Now, assuming a successful random assignment, we now have that \( C = R^T D R \) is an \( n \times n \) matrix over \( \mathbb{F} \) (or \( \mathbb{F}' \), if we used an extension field). Furthermore, \( \text{det}(C_i) \neq 0 \) for all \( i = 1, \ldots, n \). We perform the Gaussian elimination of \( C \), and since Gaussian elimination does not cause the determinant of any top \( i \times i \) matrix to vanish (since elimination only involves elementary operations), we know that at step \( i \) of the elimination process, the current top \( i \times i \) matrix still has nonzero determinant. On the other hand, this top \( i \times i \) sub-matrix is diagonal, so we must have that the entry \((i, i)\) is nonzero. Hence \( C \) is pivoting-free.

Suppose that a square matrix \( Q \) of order \( n \) can be presented in the form \( Q = L K L^T \) where \( L \) is unit lower triangular, and \( K \) is a diagonal matrix. We present an efficient procedure for computing the minors \( M_i(Q), \ldots, M_n(Q) \), starting from some index \( t \). Let \( Q_t \) be the matrix obtained from \( Q \) by removing row and column \( i \) (so that \( M_i(Q) = \text{det}(Q_t) \)). Let \( L_t \) be the matrix obtained from \( L \) by removing row \( i \). Let \( K_t \) be the diagonal matrix obtained from \( K \) by removing row \( i \) and column \( i \). Observe that \( Q_t = L_t K_t L_t^T \). For \( j = 1, \ldots, n \), let \( L_{i,j} \) be obtained from \( L_t \) by removing column \( j \), and notice that \( L_{i,j} \) is square of order \( n-1 \). By the Cauchy-Binet formula we have:

**Fact 2.7**

\[
M_i(Q) = \text{det}(Q_t) = \text{det}(L_t K L_t^T) = \sum_{j=1}^n \text{det}(L_{i,j})^2 \text{det}(K_j) .
\]

We can use the fact that \( L \) is a unit triangular matrix and \( K \) is a diagonal matrix to speed up the computation of the determinants of \( L_{i,j} \) and \( K_j \).

**Lemma 2.8** If \( j < i \) then \( \text{det}(L_{i,j}) = 0 \). Consequently \( M_i(Q) = \sum_{j=1}^n \text{det}(L_{i,j})^2 \text{det}(K_j) \). For a given \( t \leq n \), all of the values \( \text{det}(L_{i,j}) \) for \( j \geq i \geq t \) can be computed in \( O(m_L + (n-t)^\omega) \) time, where \( m_L \) is the number of non-zero entries of \( L \). In particular, \( M_t(Q), M_{t+1}(Q), \ldots, M_n(Q) \) can all be computed in \( O(m_L + (n-t)^\omega) \) time.

**Proof:** We shall denote by \( \ell_{u,v} \) the entry of \( L \) in row \( u \) and column \( v \). Since \( L \) is unit triangular, we have \( \ell_{u,u} = 1 \) for \( u = 1, \ldots, n \). We also assume that \( L \) is represented in a sparse form using, say, row lists.
Consider first the case of $\det(L_{i,j})$ when $j < i$. Consider the top $j$ rows of $L_{i,j}$. These are $j$ vectors in $\mathbb{F}^{n-1}$, that may have non-zeros only in their first $j - 1$ coordinates. Hence, they are linearly dependent. Consequently, $\det(L_{i,j}) = 0$.

We fix $t$, and show how to compute all the values $\det(L_{i,j})$ for $t \leq i \leq j \leq n$. For this purpose we need to recall some additional facts from linear algebra. Let $L[t]$ denote the matrix obtained from $L$ by taking the lower right $(n + 1 - t) \times (n + 1 - t)$ block. Hence, if $t = 1$ then $L[t] = L$ and if $t = n$ then $L[t]$ is the singleton $\ell_{n,n} = 1$. Next, recall that the cofactor $C_{i,j}(L)$ of $L$ is defined as $(-1)^{i+j} \det(L_{i,j})$. As $L$ and $L[t]$ are triangular matrices, there is a clear connection between the cofactors of $L$ and the cofactors of $L[t]$, which we denote by $C_{i,j}(L[t])$. We have, for all $t \leq i \leq j \leq n$,

$$C_{i,j}(L) = C_{i-t+1,j-t+1}(L[t])\prod_{u=1}^{t-1} \ell_{u,u} = C_{i-t+1,j-t+1}(L[t]).$$

In particular, we have that for all $t \leq i \leq j \leq n$,

$$\det(L_{i,j}) = (-1)^{i+j}C_{i-t+1,j-t+1}(L[t]).$$

So, to determine all $\det(L_{i,j})$ we have to compute all the cofactors of $L[t]$. We need the following well-known fact (see, e.g., [10]).

**Fact 2.9** If $X$ is a non-singular matrix then $\text{adj}(X) = \det(X)X^{-1}$, where $\text{adj}(X)$ is the classical adjoint of $X$; namely $\text{adj}(X)^T$ is the cofactor matrix of $X$.

Since in our case $\det(L[t]) = 1$, we only need to show how to compute $L[t]^{-1}$ quickly. We need the following result of Bunch and Hopcroft [3].

**Lemma 2.10** If $X$ is a non-singular matrix of order $x$ then $X^{-1}$ can be computed in $O(x^{2x})$ time.

Recall that $L[t]$ is a non-singular matrix of order $n + 1 - t$. Also, trivially, it can be constructed from the row lists of $L$ in $O(m_L)$ time, where $m_L$ is the number of nonzero entries of $L$. Hence, $L[t]^{-1}$ can be computed in $O(m_L + (n - t)^{2x})$ time. To finish the proof we need to also compute the values $\det(K_j)$. Since $K$ is a diagonal matrix of order $n$, we have that $\det(K_j)$ is just the product of all the diagonal entries of $K$ except for the one in location $(j,j)$. Thus we can trivially compute all the $\det(K_j)$ in $O(n) \leq O(m_L)$ time. We have thus shown that the overall running time of the algorithm for computing $M_t(Q), M_{t+1}(Q), \ldots, M_n(Q)$ requires $O(m_L + (n - t)^{2x})$ time.

The expression $O(n - t)^{2x}$ in Lemma 2.8 seems rather large at first glance. However, as we shall see in Section 3, a crucial point is that we will only apply Lemma 2.8 for values of $t$ that are very large. For example, we will mostly use that $n - t = O(\sqrt{n})$. Another point is that in our applications of Lemma 2.8 the diagonal matrix $K$ will mostly contain a zero in its bottom diagonal entry. This means that in Lemma 2.8, only $\det(K_n)$ is nonzero. This simplifies the expression for computing $M_t(Q)$ in the statement of the lemma, when applied in our setting. However, it does not seem to help in improving the running time in Lemma 2.8. Furthermore, Lemma 2.8 as stated may be applicable in other cases where $K$ is a non-singular matrix.

10
2.3 Separator trees

We say that a graph $G = (V, E)$ has a $(k, \alpha)$-separation, if $V$ can be partitioned into three parts, $X, Y, Z$ such that $|X \cup Z| \leq \alpha|V|$, $|Y \cup Z| \leq \alpha|V|$, $|Z| \leq k$, and no edge has endpoints in both $X$ and $Y$. Hence, $X$ and $Y$ are separated by $Z$. We say that the partition $(X, Y, Z)$ exhibits a $(k, \alpha)$-separation, and that $Z$ is a $(k, \alpha)$-separator.

Lipton and Tarjan [14] proved that a planar graph with $n$ vertices has an $O(\sqrt{n})$, $2/3$-separation and that such a separation can be found in $O(n)$ time.

When the existence of an $(f(n), \alpha)$-separation can be proved for each $n$-vertex graph belonging to a hereditary family (closed under taking subgraphs), one can recursively continue separating each of the separated parts $X$ and $Y$ until the separated pieces are small enough. This yields a weak separator tree 3. Notice that being planar, having bounded genus $g$, as well as being $H$-minor free for any fixed graph $H$, are all examples of nontrivial hereditary families. More formally, we say that a graph $G = (V, E)$ with $n$ vertices has an $(f(n), \alpha)$-weak separator tree if there exists a full rooted binary tree $T$ such that the following holds:

(i) Each node $t \in T$ is associated with some $V_t \subset V$.
(ii) $V = \cup_{t \in T} V_t$, If $t \neq t'$ then $V_t \cap V_{t'} = \emptyset$.
(iii) For an internal node $t \in T$ and its children $t_1$ and $t_2$, let $T_t$ be the subtree rooted at $t$. Let $X = \cup_{t_1 \in T_t} V_t, Y = \cup_{t_2 \in T_t} V_t$ and $Z = V_t$. Then $(X, Y, Z)$ exhibits an $(f(|X| + |Y| + |Z|), \alpha)$-separation of the subgraph of $G$ induced by $X \cup Y \cup Z$.
(iv) If $t$ is a leaf, then $|V_t| = O(1)$.

By using divide and conquer, the result of Lipton and Tarjan mentioned above can be stated as follows, and even extended to bounded genus graphs by the result of Gilbert, Hutchinson, and Tarjan [6].

**Lemma 2.11** Let $g$ be a fixed nonnegative integer. Given an embedding of a graph $G$ with $n$ vertices on a surface with genus $g$, an $(O(\sqrt{n}), 2/3)$-weak separator tree for $G$ can be constructed in $O(n \log n)$ time.

We note that a linear time algorithm that embeds a graph with fixed genus $g$ in a surface of genus $g$ was obtained by Mohar [15]. The embedding is purely combinatorial and is given by a rotation system (a cyclic permutation $\pi_v$ of edges incident with $v$, representing their circular order around $v$ on the surface).

Alon, Seymour, and Thomas [1] extended the result of Lipton and Tarjan to $H$-minor free graphs. However, the running time of their algorithm is $O(n^{1.5})$ for every fixed $H$. Later, Reed and Wood [18] exhibited a more flexible algorithm which runs faster, at the expense of producing a larger separator.

**Lemma 2.12** Let $\epsilon \in [0, 1/2]$ be fixed and let $H$ be a fixed graph. There is an algorithm with running time $O(n^{1+\epsilon})$ that, given an $n$-vertex graph $G$, either reports that $G$ has an $H$-minor, or outputs an $(O(n^{2-\epsilon/3}), 2/3)$-separation. In particular, if $\epsilon \in (0, 1/2]$ then an $(O(n^{2-\epsilon/3}), 2/3)$-weak separator tree for $G$ can be constructed in $O(n^{1+\epsilon})$ time.

---

3In a strong separator tree the recursion is applied to $X \cup Z$ and $Y \cup Z$, while in a weak separator tree the recursion is applied only to $X$ and $Y$. 

11
We note that in a very recent result, Kawarabayashi and Reed [12] sketch an algorithm that finds an \((O(\sqrt{n}), 2/3)\)-separation in \(O(n^{1+\epsilon})\) time, for any fixed \(\epsilon > 0\).

### 2.4 Sparsification through weak separator trees

As shown in [26], each step in the sparsification algorithm of Theorem 2.1 corresponds to an operation on \(G_A\), the underlying graph of \(A\). Indeed, we can label \(G_A\) so that it encodes the matrix \(A\) itself, not only its underlying structure. Let \(a_{i,j}\) denote an entry of \(A\). We label vertex \(i\) of \(G_A\) with \(a_{i,i}\) and label an edge \(ij\) with the two labels \(a_{i,j}\) and \(a_{j,i}\). More conveniently, we can orient \(ij\) in two directions such that \((i,j)\) is labeled \(a_{i,j}\) and \((j,i)\) is labeled \(a_{j,i}\). Given this labeling, each step of the sparsification algorithm can be thought of as an operation which transforms the current labeled underlying graph to the next one. This operation (on unlabeled graphs) is known as \textit{vertex splitting} [25]. The next paragraph defines it formally.

Suppose that \(i\) is a vertex of \(G_A\) and \(u,v\) are two neighbors of \(i\). Modify \(G_A\) by adding two new vertices \(n + 1\) and \(n + 2\). Add new edges \(\{i, n + 1\}, \{n + 1, n + 2\}, \{n + 2, u\}\) and \(\{n + 2, v\}\), and delete the original edges \(\{i, u\}\) and \(\{i, v\}\). Label the new vertices \(n + 1\) and \(n + 2\) with 0. We label \((i, n + 1)\) with 1, label \((n + 1, i)\) with \(-1\), label \((n + 1, n + 2)\) with 1, label \((n + 2, n + 1)\) with \(-1\), label \((n + 2, u)\) with \(a_{u,v}\), label \((u, n + 2)\) with \(a_{u,i}\), label \((n + 2, v)\) with \(a_{v,i}\), label \((v, n + 2)\) with \(a_{v,j}\). This operation is termed \textit{labeled vertex-splitting} in [26]. As shown there, it is straightforward to verify that each step in the sparsification algorithm corresponds to a single labeled vertex splitting operation. Hence, in the notations of Theorem 2.1, the underlying graph \(G_{A'}\) is obtained from the underlying graph \(G_A\) by a sequence of \(t = O(m)\) labeled vertex splittings.

As is well-known, if \(G_A\) is a planar graph (or a bounded genus graph), then the vertex splitting operation can be chosen to preserve planarity (or the genus). One simply chooses the above neighbors \(u, v\) of \(i\) to be consecutive vertices in the clockwise ordering of the neighbors of \(i\) in the plane (or on the surface embedding). Thus, the resulting \(G_{A'}\) is also planar (resp. of the same genus). This naive topological argument no longer holds if we only know that \(G_A\) belongs to a hereditary family of graphs that has \((f(n), \alpha)\)-weak separator trees (for example, the family of \(H\)-minor free graphs for a fixed \(H\)). Nevertheless, in [27] it is proved that one can perform a sequence of vertex splittings on a given graph belonging to a \(\delta\)-sparse hereditary family, so that the resulting graph will also have a weak separator tree with the same parameters (up to a constant). A hereditary family of graphs \(\mathcal{F}\) is \(\delta\)-sparse if for all sufficiently large \(n\), any \(G \in \mathcal{F}\) with \(n\) vertices has at most \(\delta n^2\) edges. Notice that planar graphs, bounded genus graphs, and \(H\)-minor free graphs (for fixed \(H\)) are all examples of \(\delta\)-sparse families for a suitable choice of \(\delta\). The following result is proved in [27].

**Lemma 2.13** Let \(\mathcal{F}\) be a \(\delta\)-sparse hereditary family of graphs for which there exists an algorithm \(A\) that given an \(n\)-vertex graph in \(\mathcal{F}\), generates an \((O(n^3), 2/3)\)-separation in \(O(n^7)\) time. Then, given an \(n\)-vertex graph \(G \in \mathcal{F}\), there is a vertex-split graph \(G'\) of \(G\) of maximum degree 3 such that \(G'\) has an \((O(n^3), \alpha)\)-weak separator tree where \(\alpha < 1\) is a constant that only depends on the family \(\mathcal{F}\). Furthermore, a corresponding weak separator tree for \(G'\) can be constructed in \(O(n \log n + n^7)\) time.

It is important to note that, although Lemma 2.13 is stated in [27] only for \(H\)-minor free graphs (since this is what was needed there), its proof only uses the fact that the graphs belong to a \(\delta\)-sparse hereditary family for which there is an algorithm that finds good separators. Hence we prefer to
state it in this more general form. Also, in the statement of the lemma in [27], the graph $G'$ is only required to have bounded maximum degree $k$ (not necessarily maximum degree 3). However, notice that if one splits a vertex $v$ of degree $k$ several times until it has degree 3, then the sequence of splits corresponds to a tree on $O(k)$ vertices rooted at $v$. Thus, if the graph with maximum degree $k$ had an $(O(n^\beta), \alpha)$-weak separator tree then, as $k$ is bounded, the resulting graph with maximum degree 3 has an $(O(n^\beta), \alpha')$-weak separator tree as well. So, we prefer to state Lemma 2.13 in the “degree 3” form. Another important thing to note is that Lemma 2.13 applies to weak separator trees. It does not work for strongly separator trees. Finally, notice that for $H$-minor free graphs we can simply use $\mathcal{A}$ to be the algorithm of Reed and Wood stated in Lemma 2.12 with $\gamma = (1 + \epsilon)$ and $\beta = (2 - \epsilon)/3$.

Combining Theorem 2.1 and Lemma 2.13 we obtain the following corollary.

**Corollary 2.14** Let $\mathcal{F}$ be a $\delta$-sparse hereditary family of graphs for which there exists an algorithm $\mathcal{A}$ that given an $n$-vertex graph in $\mathcal{F}$, generates an $(O(n^3), 2/3)$-separation in $O(n^3)$ time. Then, given an a matrix $A \in \mathbb{F}^{n \times n}$ with $G_A \in \mathcal{F}$, another matrix $A' \in \mathbb{F}^{n+2t, n+2t}$ can be constructed and which has the following properties. For all $i = 1, \ldots, n$ and $j = 1, \ldots, n$, $M_{i,j}(A) = M_{i,j}(A')$. Furthermore, $t = O(n)$ and each row and column of $A'$ has at most three nonzero entries. The graph $G_{A'}$ has an $(O(n^3), \alpha)$-weak separator tree where $\alpha < 1$ is a constant that only depends on the family $\mathcal{F}$. The time to construct $A'$ and the corresponding weak separator tree of $G_{A'}$ is $O(n \log n + n^3)$.

### 2.5 Nested dissection

We briefly describe the generalized nested dissection method, developed by Lipton, Rose and Tarjan [13]. We follow the variant developed by Gilbert and Tarjan [8], as it applies to weak separator trees.

Suppose that $G = (V, E)$ is a graph with $n$ vertices, and $T$ is a weak separator tree of $G$. Recall that each node $x \in T$ is associated with a subset $V_x \subset V$. A $T$-elimination order of the vertices of $G$ is a bijective labeling from $V$ to $\{1, \ldots, n\}$ so that if $y$ is a child of $x$ in $T$ then the vertices in $V_y$ receive smaller labels than the vertices in $V_x$. Notice that the vertices of $V_{\text{root}}(T)$ receive the largest labels. Given $T$, it is straightforward to compute a $T$-elimination order in linear time by performing, say, a postorder traversal of the tree. So, from now on we assume that $V = \{1, \ldots, n\}$ and we identify a vertex with its label in the elimination order.

Let $C$ be a pivoting-free matrix and let $T_C$ be a weak separator tree of $G_C$ (and note that we now assume that row $i$ of $C$ is also the label in the $T_C$-elimination order of $G_C$). Orient the edges of $G_C$ from the lower ordered endpoint to the higher ordered endpoint. In particular, notice that vertex $n$ has out-degree 0. When performing the elimination of $C$, some entries that were zero in $C$ become nonzero during the elimination process. Such an entry is called a fill-in. So, let $G_C^*$ denote the fill-in graph after elimination. There is an edge $(u, v)$ in $G_C^*$ if $(u, v) \in G_C$ or if $(u, v)$ is a fill-in. It is easy to see (cf. [8]) that fill-ins cannot cross separators. Namely, if $(u, v)$ is an edge of $G_C^*$ then the node of $T_C$ containing $v$ is an ancestor of the node of $T_C$ containing $u$. Hence, if $T_C$ is an $(O(n^\beta), \alpha)$-weak separator tree then the out-degree of any vertex in $G_C^*$ is $O(n^\beta)$. Another important observation of Gilbert and Tarjan (see Theorem 4 in [8]) is that if the maximum degree of $G_C$ is bounded (that is, each row and column of $C$ has a bounded number of nonzero entries), then the total number of edges of $G_C^*$ is $O(n \log n)$.

During the elimination process, when we reach step $i$ and consider entry $(i, i)$, we only need to
Hence, doing naive elimination, the total operation count is $O(n^2)$ where $d^*(i)$ is the out-degree of $i$ in $G_C$. Since $\sum_{i=1}^n d^*(i) = O(n \log n)$, this already yields an $O(n^{1+\beta} \log n)$ operation count, even when using the naive method. A slightly more careful analysis given in [8] shows that the total naive operation count is only $O(n^{1+\beta})$. By plugging in the fast Gaussian elimination method of [3], instead of the naive method, Lipton, Rose, and Tarjan observed that the total operation count reduces to $O(n^{\omega})$. To summarize, the result of Gilbert and Tarjan is stated in the following lemma.

**Lemma 2.15** Let $C$ be a pivoting-free matrix of order $n$ with a bounded number of nonzero entries in each row and column, and assume that an $(O(n^\beta), \alpha)$-weak separator tree for $G_C$ is given, where $\beta \geq 1/2$. Then, a unit lower triangular matrix $L$ and a diagonal matrix $K$ can be constructed in $O(n^{\omega})$ time so that $C = LKL^T$. We need a minor modification of the above result. Let $w \in \mathbb{F}^{n+1}$ be any vector. Let $Q$ be the $(n+1) \times (n+1)$ matrix obtained from $C$ by adding $w$ as a last row and $w^T$ as a last column. Notice that $Q$ is not necessarily pivoting-free, but since $C$ is pivoting-free, the only location where we may encounter a diagonal zero during the elimination process is $(n+1, n+1)$ (but this, in turns, means that the whole last row and column at this final elimination stage is zero, so we are done). Clearly, if $T_C$ is an $(O(n^\beta), \alpha)$-weak separator tree for $G_C$ then it is also an $(O(n^\beta), \alpha)$-weak separator tree for $G_Q$ since we may assign vertex $n+1$ to the root separator. Also notice that $G_Q$ only has at most $n$ edges more than $G_C$, so it also has $O(n)$ edges as well. Also observe that the out-degree of any vertex in $G_Q^\ast$ is at most one larger than its out-degree in $G_C^\ast$ (since it may now have an edge to $n+1$). The new vertex, $n+1$, has out-degree 0. We therefore have:

**Lemma 2.16** Let $C$ be a pivoting-free matrix of order $n$ with a bounded number of nonzero entries in each row and column, and assume that an $(O(n^\beta), \alpha)$-weak separator tree for $G_C$ is given, where $\beta \geq 1/2$. Let $w \in \mathbb{F}^{n+1}$ be any vector. Let $Q$ be the $(n+1) \times (n+1)$ matrix obtained from $C$ by adding $w$ as a last row and $w^T$ as a last column. Then, a unit lower triangular matrix $L$ and a diagonal matrix $K$ can be constructed in $O(n^{\omega\beta})$ time so that $Q = LKL^T$.

Observe that if $\text{rank}(Q) = \text{rank}(C) = n$ then $Q$ is not pivoting-free and hence $K$ must have a unique zero in its diagonal, in position $(n+1, n+1)$.

3 Proof of the main result

We state and prove our main result in its more general setting, for which Theorem 1.1 is a special case.

**Theorem 3.1** Let $F$ be a $\delta$-sparse hereditary family of graphs for which there exists an algorithm $A$ that given an $n$-vertex graph in $F$, generates an $(O(n^\beta), 2/3)$-separation in $O(n^\gamma)$ time. Then, given a system of linear equations $Ax = b$ where $A \in \mathbb{F}^{n \times n}$ is non-singular, $b \in \mathbb{F}^n$, and $G_A \in F$, there is an algorithm that finds the unique solution of the system in $O(n^{\omega\beta} + n^{\gamma} + n \log n)$ time. For the
fields \(\mathbb{R}, \mathbb{Q}, \mathbb{C}\) the algorithm is deterministic. For arbitrary fields (and, in particular, for finite fields) it is a randomized Las Vegas algorithm.

The proof of Theorem 1.1 follows immediately from the more general Theorem 3.1. For planar graphs and bounded genus graphs we simply use \(\beta = 1/2\) and \(\gamma = 1\), following the aforementioned results of Lipton and Tarjan [14] and Gilbert, Hutchinson and Tarjan [6]. This gives a runtime of \(O(n^{\omega/2} + n \log n)\) for these graphs (which is \(O(n^{\omega/2})\) if \(\omega > 2\)). For the class of \(H\)-minor free graphs (where \(H\) is any fixed graph), we use the result of Reed and Wood, stated as Lemma 2.12, with \(\epsilon = (2 \omega - 3)/(3 + \omega)\) that implies using \(\beta = 3/(3 + \omega)\) and \(\gamma = 3\omega/(3 + \omega)\) in Theorem 3.1. Hence, the runtime obtained is this case is \(O(n^{3\omega/(3+\omega)})\).

The algorithm that proves Theorem 3.1 consists of two parts, which we denote by ALG\(_1\) and ALG\(_2\). The goal of the first part is to reduce the problem to a sparse setting that is suitable for solving a linear system recursively.

### 3.1 Algorithm ALG\(_1\)

This algorithm is given as input a linear system \(Ax = b\) where \(A \in \mathbb{F}^{n \times n}\) is non-singular, and \(b \in \mathbb{F}^n\). The matrix is represented via the labeled list representation of \(G_A\), and it is assumed that \(G_A \in \mathcal{F}\). The goal of ALG\(_1\) is to compute the unique solution \(c^T = (c_1, \ldots, c_n)\) of the system.

To achieve this goal, we first apply Corollary 2.14 to obtain a matrix \(A' \in \mathbb{F}^{n+2t,n+2t}\) and an \((O(n^3),\alpha)\)-weak separator tree for \(G_{A'}\) as in the statement of the corollary. The running time required for these constructions is \(O(n^7 + n \log n)\).

We also construct a vector \(b' \in \mathbb{F}^{n+2t}\) which is identical to \(b\) in the first \(n\) coordinates, and which is zero in the remaining \(2t\) coordinates. Since \(t = O(n)\), it takes only \(O(n)\) time to construct \(b'\).

We call ALG\(_2\) and provide it as input the system \(A'x = b'\) together with the separator tree for \(G_{A'}\). Notice that since \(\det(A') = \det(A)\) this system also has a unique solution. ALG\(_2\) returns the unique solution \((c'_1, \ldots, c'_{n+2t})\) of the system \(A'x = b'\). Now, ALG\(_1\) returns the first \(n\) coordinates \((c'_1, \ldots, c'_n)\) as its answer.

The overall running time of ALG\(_1\) (using ALG\(_2\) as a “black box” and not counting the running time of ALG\(_2\)) is \(O(n^7 + n \log n)\). To prove the correctness of ALG\(_1\) (assuming the correctness of ALG\(_2\)) we need to establish the following lemma.

**Lemma 3.2** Let \((c_1, \ldots, c_n)\) be the unique solution of \(Ax = b\) and let \((c'_1, \ldots, c'_{n+2t})\) be the unique solution of the system \(A'x = b'\). Then \(c_i = c'_i\) for \(i = 1, \ldots, n\).

**Proof:** Let \(B = [A|b]\) and let \(B' = [A'|b']\). Let \(B_i\) and \(B'_i\) be, respectively, the matrices obtained from \(B\) and \(B'\) by removing column \(i\). Since \(\det(A) = \det(A')\) it suffices, by Cramer’s rule, to prove that \(\det(B_i) = \det(B'_i)\) for \(i = 1, \ldots, n\). One way to obtain \(\det(B_i)\) is by expansion of the determinant by the last column of \(B_i\). Hence,

\[
\det(B_i) = \sum_{j=1}^{n} (-1)^{n+j} b_j M_{j,i}(A).
\]
Similarly, we can obtain $det(B'_i)$ by expansion of the determinant by the last column of $B'_i$.

$$det(B'_i) = \sum_{j=1}^{n+2t} (-1)^{n+2t+j}b'_j M_{j,i}(A') .$$

But $b'_j = b_j$ for $j = 1, \ldots, n$ and $b'_j = 0$ for $j = n + 1, \ldots, n + 2t$. Also, by Corollary 2.14, we have that $M_{j,i}(A') = M_{j,i}(A)$ for $i = 1, \ldots, n$ and $j = 1, \ldots n$. Hence, for $i = 1, \ldots, n$ we have

$$det(B'_i) = \sum_{j=1}^{n} (-1)^{n+2t+j}b_j M_{j,i}(A) = \sum_{j=1}^{n} (-1)^{n+j}b_j M_{j,i}(A) = det(B_i).$$

\[\square\]

### 3.2 Algorithm $ALG_2$

Algorithm $ALG_2$ is a recursive algorithm. To describe its input in general we need to define the underlying graph $G_R$ of a rectangular matrix $R$ with $n+p$ rows and $n$ columns. This is simply defined as the underlying graph of the matrix obtained from $R$ by padding $R$ with $p$ zero columns to the right. In other words, some vertices of $G_R$ may only represent rows, and do not have a corresponding column.

Throughout all its recursive calls, Algorithm $ALG_2$ will need to invoke Lemma 2.6 several times, in fact, up to $O(n)$ times, each time with different column dimension $n_\ell$ and row dimension $n_\ell+p_\ell$ of a suitable matrix $R_\ell$, where we will always have $n_\ell+p_\ell \leq O(n)$. Instead of randomly generating a diagonal matrix $D_\ell$ each time separately, it is more convenient to generate only one $O(n) \times O(n)$ matrix $D$, and use as $D_\ell$ the top $n_\ell+p_\ell$ block of $D$. Since Lemma 2.6 guarantees that $D_\ell$ will make $R_\ell^T D_\ell R_\ell$ pivoting-free with probability at least $1 - O(1/n^2)$, we have by the union bound that $D$ has the property that all the corresponding $D_\ell$’s throughout all the invocations will make the corresponding products $R_\ell^T D_\ell R_\ell$ pivoting-free with probability at least $1 - O(1/n)$. So, from here until the end of this subsection we assume that pivoting-freeness holds for all invocations.

Recall also that if $\mathbb{F}$ is one of $\mathbb{R}, \mathbb{Q}, \mathbb{C}$, then $D$ is not required (that is, $D = I$) and there is no randomization involved.

The input to $ALG_2$ is a linear system $Rx = h$ where $R \in \mathbb{F}^{n_1+p_1,n_1}$ and $h \in \mathbb{F}^{n_1+p_1}$. Furthermore, it is assumed that the system has a unique solution (in particular, $rank(R) = n_1$), that each row and column of $R$ has at most three nonzero entries, and that $p_1 \leq n_1$. An additional input to $ALG_2$ is an $(O(n_1^\alpha), \alpha)$-weak separator tree for $G_R$, which we denote by $T_R$. Let $Z$ denote the set of vertices of $G_R$ associated with the root of $T_R$, and let $N(Z)$ denote the neighbors of $Z$ in $G_R$. We assume, without loss of generality, that the vertices of $Z \cup N(Z)$ are the bottom indices of $R$ (if this is not the case we can just switch the order of the linked lists representing $G_R$ to ensure this fact).

The goal of $ALG_2$ is to compute the unique solution for the system $Rx = h$. Notice that the initial call to $ALG_2$ from $ALG_1$ satisfies these requirements with $p_1 = 0$, $R = A'$, and $n_1 = n + 2t$.

To achieve its goal, $ALG_2$ operates as follows. Recall that $C = R^T D_{n_1+p_1} R$ is assumed to be pivoting-free. Since $R$ has at most 3 nonzero entries in each row or column, $C$ has at most 7 nonzero entries in each row or column, and, furthermore, $C$ is constructed in $O(n_1)$ time.
Another important observation (see also [16] for the same observation) is that $G_C$ is the square of the graph $G_R$ (a vertex of $G_C$ has as its neighbors all the vertices within distance at most 2 in $G_R$). This observation shows that we can construct an $(O(n_1^2), \alpha)$-weak separator tree for $G_C$, denoted by $T_C$, in linear time, from the separator tree $T_R$ of $G_R$. It will be precisely the same tree, but the vertices of $G_C$ assigned to a node of $x \in T_C$ will be different from the vertices assigned to the same node in $T_R$. Let $V_x(R)$ be the set of vertices assigned to node $x$ in $T_R$. We define $V_x(C)$ as follows. For the root vertex, take $V_{root}(C) = Z \cup N(Z)$. Notice that since $G_R$ has maximum degree 3, $|Z \cup N(Z)| \leq 4|Z|$ and notice that $Z \cup N(Z)$ is now a separator of $G_C$, as required. Now, mark all the vertices of $V_{root}(C)$ as taken. For a general node $x \in T_C$, also take $V_x(C)$ to be the yet un-taken vertices of $V_x(R) \cup N(V_x(R))$, and so on.

In addition to constructing $C$, we also “construct” $Q = B^TDB$ where $B = [R|h]$. The only difference between $Q$ which is a square symmetric matrix of order $n + 1$, and $C = R\bar{T}D\bar{R}$, which is square symmetric of order $n$, is that $Q$ has an additional vector $w$ as a last row and last column. The extra time required to construct $w$ is clearly $O(n_1 + p_1)$.

We may now apply the algorithm of Lemma 2.16 to $C$ and $w$. As guaranteed by the lemma, we construct, in $O(n_1^\omega)$ time, matrices $L$ and $K$ so that $Q = LK^T$. By Lemma 2.8, we may now compute all the minors $M_i(Q)$ for $i \in Z \cup N(Z)$, since they (together with the index $n_1 + p_1 + 1$ which corresponds to the last row $w$) are the bottom indices of $Q$. The time required is $O(|Z \cup N(Z)|^\omega)$. Since $|Z| = O(n_1^\beta)$ and since $|N(Z)| \leq 3|Z|$, this is only $O(n_1^\omega)$ time.

By Lemma 2.5, we can use these computed minors to obtain $c_i^2$ for all $i \in Z \cup N(Z)$. To obtain $c_i$ we use the interpolation argument, as described after Lemma 2.4, working with the system $Rx = a + h$ where $a$ is the sum of the columns of $R$. Recall that this returns the squares $(c_i^2)^2$ for all $i \in Z \cup N(Z)$ and that $c_i$ is obtained by $c_i = (c_i^2 + 1)^2 - c_i^2$ for $i \in Z \cup N(Z)$ (recall also that if the field has characteristic 2 then $c_i^2$ already uniquely defines $c_i$ and there is no need to apply interpolation). We have thus shown how to compute, in $O(n_1^{\omega\beta})$ time, the values $c_i$ for all $i \in Z \cup N(Z)$.

But we are interested in the complete solution of $Rx = h$, not just the solution for the variables that correspond to the root separator $Z$ and its neighbors $N(Z)$. Here is where recursion comes into play. Let us simplify the system $Rx = h$ by replacing the unknowns $x_i$ for $i \in Z \cup N(Z)$ with their actual computed values. This possibly causes some equations to be eliminated (equations that involve only variables of $Z \cup N(Z)$), and some equations may become shorter. Since $R$ is represented in sparse form, the simplified system $R'x = h'$ is computed in $O(n_1 + p_1) = O(n_1)$ time.

The crucial argument is that we can now partition the simplified system into two sub-systems, each having a unique solution, and apply $ALG_2$ recursively to each subsystem. Let $(X, Y, Z)$ be the separation defined by the root of $T_R$. Indeed, let $R_1x = h_1$ be the system that corresponds, after simplification, only to equations that contain variables of $X \setminus N(Z)$. Notice that $R_1$ has precisely $n_{1,1} = |X \setminus N(Z)|$ columns, but may have more rows, possibly even $|X|$ rows. Hence $R_1$ has $n_{1,1} + p_{1,1}$ rows where $p_{1,1} \leq |N(Z)| = O(n_1^\beta)$. Similarly, $R_2$ has $n_{1,2} + p_{1,2}$ rows and $n_{1,2}$ columns where $n_{1,2} = |Y \setminus N(Z)|$ and $p_{1,2} = O(n_1^\gamma)$. Also, since $Rx = h$ had a unique solution, we also have that $R_i h = h_i$ each have a unique solution. Obtaining the separator trees for $R_1$ and $R_2$ is trivial. These are just the two subtrees rooted by the children of the root of $T_R$ (in
fact, we even gain a bit since vertices of \( N(Z) \) are also eliminated from the sets that correspond to nodes in these two subtrees). The recursion is hence on systems of total column sum \( n_{1,1} + n_{1,2} \leq n_1 \)
and \( n_{1,i} \leq \alpha n_1 \), and total row sum \( n_{1,1} + n_{1,2} + p_{1,1} + p_{1,2} \leq |X| + |Y| \leq n_1 \). The standard analysis of the total running time of all recursive calls is, therefore, \( O(n_1^{\omega_0^\beta} + n_1 \log n_1) \), as required.

The total running time of \( \text{ALG}_2 \), starting from the initial invocation from \( \text{ALG}_1 \), is, therefore, \( O(n^{\omega_0^\beta} + n \log n) \), as required. This completes the proof of Theorem 3.1, as the overall running time of all three parts is \( O(n^{\omega_0^\beta} + n^\gamma + n \log n) \).

4 Concluding remarks

The running times in Theorem 1.1, and, in its more general form of Theorem 3.1, are stated in terms of the matrix multiplication exponent, \( \omega \). The algorithm from [4] which yields \( \omega < 2.376 \), is only theoretical, and it presently has no practical implementation for reasonable values of \( n \). However, it is important to note that our main result, as well as the original nested dissection method, have practical implementations even if we use the naive matrix multiplication with \( \omega = 3 \). In this case, the running times of our algorithm, for the case of planar graphs, bounded-genus graphs, and also \( H \)-minor-free graphs all become \( O(n^{1.5}) \). It is important to note that if we use \( \omega = 3 \) (naive matrix multiplication) then all ingredients of our algorithm become practically implementable. In fact, for reasonable sizes of \( n \) (starting from several hundreds) one can use Strassen’s algorithm for fast matrix multiplication [23], which has an easy implementation, and for which \( \omega < 2.81 \).

As mentioned in the introduction, when the matrix \( A \) and the vector \( b \) have bounded integer entries, and the system is to be solved over \( \mathbb{Q} \), we can state our running times in terms of bit complexity. The standard approach in this case is to perform all operations over some large finite field \( \mathbb{F}_p \) where \( p \) is a prime which is larger than any (absolute value of a) determinant that we may encounter in our algorithm. This will cause all the (absolute) determinants and all the minors that are computed by our algorithm to have the same value in \( \mathbb{F}_p \) as they would have over the integers. Since \( n \times n \) matrices with bounded integer entries cannot have determinants that are larger than \( n^{O(n)} \), it suffices to choose \( p = n^{O(n)} \). Since in \( \mathbb{F}_p \), each arithmetic operation requires \( O(\log p) \) time, this amounts, in our case, to \( \tilde{O}(n) \) time (bit complexity) for each arithmetic operation. Hence, in the bounded integer case, the bit-complexity of Theorem 1.1 for planar graphs and bounded-genus graphs becomes \( \tilde{O}(n^{1+\omega/2}) \). For \( H \)-minor free graphs, we notice that Lemma 2.13 is purely combinatorial, and is applied only once in \( \text{ALG}_1 \). Hence, we can use it with \( \beta = 1/2 \) and \( \gamma = 1.5 \), where \( A \) is the algorithm from [1]. The overall bit complexity of Theorem 1.1 when applied to \( H \)-minor free graphs then becomes \( \tilde{O}(n^{1+\omega/2} + n^{1.5}) = \tilde{O}(n^{1+\omega/2}) \).

Finally, the very recent result of Kawarabayashi and Reed [12] mentioned after Lemma 2.12, when used instead of Lemma 2.12, yields in Theorem 1.1 a running time of \( O(n^{\omega/2}) \) also in the fixed excluded minor case, as long as \( \omega > 2 + \epsilon \) for any fixed \( \epsilon > 0 \).

References


