Matrix sparsification for rank and determinant computations via nested dissection

Raphael Yuster
University of Haifa
Haifa 31905, Israel
raphy@math.haifa.ac.il

Abstract

The nested dissection method developed by Lipton, Rose, and Tarjan is a seminal method for quickly performing Gaussian elimination of symmetric real positive definite matrices whose support structure satisfies good separation properties (e.g. planar). One can use the resulting LU factorization to deduce various parameters of the matrix.

The main results of this paper show that we can remove the three restrictions of being "symmetric", being "real", and being "positive definite" and still be able to compute the rank and, when relevant, also the absolute determinant, while keeping the running time of nested dissection.

Our results are based, in part, on an algorithm that, given an arbitrary square matrix A of order n having m non-zero entries, creates another square matrix B of order $n + 2t = O(m)$ with the property that each row and each column of B contains at most three *nonzero entries, and, furthermore, $rank(B) = rank(A) + 2t$ and $det(B) = det(A)$. The running time of this algorithm is only $O(m)$, which is optimal.*

1 Introduction

Computing the rank and the determinant of a matrix is one of the most basic algebraic problems and considerable effort is devoted to obtaining efficient algorithms for these tasks (see, e.g. [8]). Both of these problems can be solved as by-products of Gaussian elimination. An old result of Hopcroft and Bunch [2] asserts that Gaussian elimination of a matrix requires asymptotically the same number of algebraic operations needed for matrix multiplication. It follows, in particular, that the algebraic complexity of rank and determinant computations of an $n \times n$ matrix over a field is $O(n^\omega)$ where $\omega < 2.376$ is the *matrix multiplication exponent* [3]. In fact, no faster algorithm is known for general $n \times n$ matrices.

If A is sparse and has only $m \ll n^2$ non-zero entries, it seems likely at first glance that computing rank and determinant should be easier. Unfortunately, it is generally not known to be the case and Gaussian elimination is unlikely to be helpful. This can be attributed to the negative result of Yannakakis [19] who proved that controlling the number of fill-ins (entries that were originally zero and become non-zero during the Gaussian elimination process due to pivoting) is an NP-Hard problem. If we allow randomness then faster algorithms are known. Wiedemann's algorithm [18] implies a Monte Carlo rank algorithm for a matrix over an arbitrary field which requires $\tilde{O}(n^2 + nm)$ algebraic operations. Recently, a randomized Las Vegas algorithm for computing the rank was obtained by Eberly et al [4]. In the case where $m = \Theta(n)$ the expected running time of their algorithm is $O(n^{3-1/(\omega-1)}) \leq O(n^{2.28})$.

In some important cases that arise in various applications, the matrix has additional structural properties in addition to being sparse. To make this notion precise we need a definition. Let A be an arbitrary $n \times n$ matrix. The *representing graph* of A, denoted 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 $a_{j,i} \neq 0$. Notice that the diagonal elements of A play no role in the definition of G_A. Also notice that G_A is always an undirected simple graph, while A may or may not be symmetric. G_A is sometimes called the *support structure* of A.

The seminal *nested dissection* method of Lipton, Rose, and Tarjan [9], generalizing an earlier result of George [6], asserts that if A is a symmetric positive definite matrix and the graph G_A is represented by an appropriate separator tree (we will define this representation formally in Section 3) then Gaussian elimination on A can be performed with $O(n^{\omega\beta})$ arithmetic operations, where β 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. 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 [10]). For graphs with an excluded fixed minor it is also known that $\beta = 1/2$ (although to initially construct a separation with this parameter requires $O(n^{1.5})$ time with present methods [1]).

However, the nested dissection method has several limitations. The matrix needs to by *symmetric* and needs to be *positive definite* (it is not difficult to modify the method so that it applies also to symmetric positive semidefinite matrices of the form AA^T). In particular, the matrices are assumed to be matrices over the reals. The method does not apply to matrices over finite fields (not even over $GF(2)$). Even if the matrix is real, but either non-symmetric or non positive semidefinite, the nested dissection method is not applicable. The main results of this paper show that we can overcome all of these limitations if we wish to compute ranks or absolute determinants.

An important tool in achieving the main results of this paper is an algorithm that takes as input an arbitrary $n \times n$ matrix A (over any field or integral domain) with m non-zero entries, and outputs, in $O(m)$ time, another $(n+2t) \times (n+2t)$ matrix B with $det(B) = det(A)$ and $rank(B) = rank(A) + 2t$ where $t = O(m)$. The important feature of B is that each row and column have at most *three* non-zero entries. As usual for sparse matrices, we assume that 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 $a_{i,j} \neq 0$, and each R_i is sorted by the column indices (if sorting is not assumed then an additional $\log n$ factor is needed in the running time). Without loss of generality we assume that A has no zero rows, and hence $m \geq n$.

Theorem 1.1 *Let A be a square matrix of order n with m non-zero entries. Another square matrix B of order $n + 2t$ with $t = O(m)$ is constructed in $O(m)$ time so that $det(B) = det(A)$ and $rank(B) = rank(A) + 2t$. Furthermore, each row and column of B contains at most three non-zero entries.*

The usefulness of Theorem 1.1 stems from the fact that constant powers of B, as well as BDB^T (where D is any diagonal matrix), are very sparse matrices, while this is not true for the constant powers of A nor for AA^T. In many cases (for example, over the reals), we know that $rank(BB^T) = rank(B) = rank(A) + 2t$ and also that $det(BB^T) = det(A)^2$. Since BB^T is symmetric, and positive semidefinite (over the reals), then the nested dissection method may apply if we can also guarantee that G_B has a good separator tree assuming G_A has one (guaranteeing this, in general, is not an easy task). Over finite fields we also need to overcome other difficulties, such as the fact that $rank(BB^T) = rank(B)$ does not always hold. Nevertheless, we show how to overcome these difficulties as well. We now state the results that we obtain.

Theorem 1.2 *Let $A \in F^{n \times n}$. If G_A has bounded genus then $rank(A)$ can be computed in $\tilde{O}(n^{\omega/2}) < O(n^{1.19})$ time. The algorithm is deterministic if $F = \Re$ and randomized if F is a finite field. If G_A has a fixed excluded minor then $rank(A)$ can be computed in $\tilde{O}(n^{3\omega/(\omega+3)}) < O(n^{1.326})$ time. The algorithm is deterministic if $F = \Re$ and randomized if F is a finite field.*

The stated running times are given under the assumption that each arithmetic operation in the field takes $\tilde{O}(1)$ time (namely, the algorithms are measured in terms of their algebraic complexity). If F is a finite field whose number of elements is polynomial in n then it is, indeed, true that each arithmetic operation takes $\tilde{O}(1)$ time and hence the running times in Theorem 1.2 also measure actual bit complexity. In the case of matrices with bounded integer entries over the reals extra care needs to be taken. In this case, the running times in Theorem 1.2 measure actual bit complexities only if we allow the algorithms to become randomized (as in the finite field case).

For the case of computing absolute determinants of matrices over the reals we obtain the following.

Theorem 1.3 *Let $A \in \Re^{n \times n}$. If G_A has bounded genus then $|det(A)|$ can be computed in $\tilde{O}(n^{\omega/2}) < O(n^{1.19})$ time (algebraic complexity). If G_A has a fixed excluded minor then then $|det(A)|$ can be computed in $\tilde{O}(n^{3\omega/(\omega+3)}) < O(n^{1.326})$ (algebraic complexity). If A has bounded integer entries then, in both cases, $|det(A)|$ can be computed in $\tilde{O}(n^{1+\omega/2}) < O(n^{2.19})$ time (bit complexity).*

The rest of this paper is organized as follows. In Section 2 we prove Theorem 1.1. Theorems 1.2 and 1.3 are proved in Section 3. The final section contains concluding remarks and open problems.

2 The sparsification algorithm

Proof of Theorem 1.1: As noted in the introduction, we assume that the given $n \times n$ matrix 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.

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 $(\cdot, 0^*)$ entry in any list. We assume in the proof that $0^* \neq 0$ in comparisons, but $0^* = 0$ in arithmetic computations. As these assumptions only increase m by a factor of at most 2, they have no effect on the claimed running time.

For implementation reasons we will maintain cross-referencing pointers between $(j, a_{i,j})$ in R_i and $(i, a_{j,i})$ in

$$B_t = \begin{array}{c} \\ v \\ u \\ i \end{array} \left(\begin{array}{c|ccc} M_1 & \multicolumn{3}{c} M_2 \\ \hline & b_{v,v} & b_{v,u} & b_{v,i} \\ M_3 & b_{u,v} & b_{u,u} & b_{u,i} \\ & b_{i,v} & b_{i,u} & b_{i,i} \end{array} \right)$$

$$B_{t+1} = \begin{array}{c} \\ v \\ u \\ i \\ n+2t+1 \\ n+2t+2 \end{array} \left(\begin{array}{c|ccc|cc} M_1 & \multicolumn{3}{c|}{M_2} & \multicolumn{2}{c} 0 \\ \hline & b_{v,v} & b_{v,u} & 0 & 0 & b_{v,i} \\ M_3 & b_{u,v} & b_{u,u} & 0 & 0 & b_{u,i} \\ & 0 & 0 & b_{i,i} & 1 & 0 \\ \hline 0 & 0 & 0 & -1 & 0 & 1 \\ & b_{i,v} & b_{i,u} & 0 & -1 & 0 \end{array} \right)$$

Figure 1. A single step of the algorithm

R_j. Since the lists are initially sorted by the column indices, creating these cross-reference pointers requires $O(m)$ time.

At step t of the algorithm, the current matrix is denoted by B_t, its current order will be $n + 2t$, and its elements are denoted by $b^t_{i,j}$ or by $b_{i,j}$ if t is clear from the context. Just like A, the matrix B is represented by lists R_i for $i = 1, \ldots, n + 2t$. B_t also has the property that $b_{i,j} \neq 0$ if and only if $b_{j,i} \neq 0$. We initially set $B_0 = A$. A single step of the algorithm constructs B_{t+1} from B_t by increasing the number of rows and columns of B_t by 2 and by modifying constantly many entries of B_t. The algorithm halts when each row list of B_t has at most three entries. Thus, in the final matrix B_t we have that each row and column has at most 3 non-zero entries. At any step of the algorithm we will have $det(B_{t+1}) = det(B_t)$ and $rank(B_{t+1}) = rank(B_t) + 2$. Hence, in any step we will have $det(B_t) = det(A)$ and $rank(B_t) = rank(A) + 2t$.

At any step of the algorithm we let r_i denote the number of elements in R_i. Furthermore, we let F_t be the set of indices of rows of B_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

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

Observe that initially $s_0 < m$.

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 $b_{i,u} \neq 0$ and $b_{i,v} \neq 0$. 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

$$B' = \begin{array}{c} \\ v \\ u \\ i \\ n+2t+1 \\ n+2t+2 \end{array} \left(\begin{array}{c|ccc|cc} M_1 & \multicolumn{3}{c|}{M_2} & \multicolumn{2}{c} 0 \\ \hline & b_{v,v} & b_{v,u} & b_{v,i} & 0 & 0 \\ M_3 & b_{u,v} & b_{u,u} & b_{u,i} & 0 & 0 \\ & b_{i,v} & b_{i,u} & b_{i,i} & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 1 \\ & 0 & 0 & 0 & -1 & 0 \end{array} \right)$$

Figure 2. The matrix B'

represent the diagonal element $b_{i,i}$). Let X denote the 3×3 principal sub-matrix of B_t obtained from rows $\{i, u, v\}$ and columns $\{i, u, v\}$. The matrix B_{t+1} is constructed from B_t as follows. The four elements $b_{i,u}, b_{i,v}, b_{u,i}, b_{v,i}$ of X are replaced with zero. In other words, $b^{t+1}_{i,u} = b^{t+1}_{i,v} = b^{t+1}_{u,i} = b^{t+1}_{v,i} = 0$. The other 5 elements of X remain the same. Two new rows are added and two new columns are added. The non-zero entries in row $n + 2t + 1$ are $b^{t+1}_{n+2t+1,i} = -1$ and $b^{t+1}_{n+2t+1,n+2t+2} = 1$. The non-zero entries in row $n + 2t + 2$ are $b^{t+1}_{n+2t+2,u} = b^t_{i,u}$, $b^{t+1}_{n+2t+2,v} = b^t_{i,v}$, and $b^{t+1}_{n+2t+2,n+2t+1} = -1$. The non-zero entries in column $n+2t+1$ are $b^{t+1}_{i,n+2t+1} = 1$ and $b^{t+1}_{n+2t+2,n+2t+1} = -1$. The non-zero entries in column $n+2t+2$ are $b^{t+1}_{u,n+2t+2} = b^t_{u,i}$, $b^{t+1}_{v,n+2t+2} = b^t_{v,i}$, and $b^{t+1}_{n+2t+1,n+2t+2} = 1$. This construction is visualized in Figure 1, where we assume, for the sake of clarity (and actually, without loss of generality, since flipping two rows and two columns has no effect on the determinant nor the rank) that $i = n + 2t$, $u = n + 2t - 1$, $v = n + 2t - 2$, and hence X is the bottom right 3×3 sub-matrix of B_t.

Modifying the row lists R_u, R_v, R_i to reflect the changes between B_t and B_{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 $b_{i,u}$ and $b_{i,v}$ became zero in B_{t+1} and they were not zero in B_t, and since $b_{i,n+2t+1}$ is a new non-zero element which equals 1 in B_{t+1}. Thus, r_i decreased by 1. If we now have $r_i \leq 3$ we delete i from the head of F_t to obtain F_{t+1}. Otherwise, $F_{t+1} = F_t$ remains intact. Notice also that r_u did not change since $b_{u,i}$ in B_t was "moved" to $b_{u,n+2t+2}$ in B_{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 $s_{t+1} < s_t$ proving that the algorithm has less than m steps until it terminates. It remains to prove the following lemma:

Lemma 2.1 *$det(B_{t+1}) = \det(B_t)$ and $rank(B_{t+1}) = rank(B_t) + 2$.*

Proof: Consider first the matrix B' obtained from B_t by adding two rows and two columns all of which are zero except for the lower 2×2 part, as depicted in Figure 2. Clearly, $det(B') = det(B_t)$ and $rank(B') = rank(B_t) + 2$. It remains to show that $det(B') = det(B_{t+1})$. Let $R(j)$ denote the jth row of B' and let $C(j)$ denote the jth column of B'. The following sequence of elementary operations does not change the determinant nor the rank and transform B' to B_{t+1}.

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

■

3 Ranks and determinants of matrices with a separator structure

3.1 Graph-theoretic tools

A graph G' is a *minor* of a graph G if G' can be obtained from a subgraph of G by contracting edges. A graph is *H-minor free* if H is not a minor of G. An H-minor free graph is also said to have *excluded H-minor*. An *H-model* in G is a set of vertex-disjoint connected subgraphs $\{X_v : v \in V(H)\}$ indexed by the vertices of H, such that for every edge $uv \in E(H)$, there is an edge $xy \in E(G)$ with $x \in X_u$ and $y \in X_v$. Clearly G has an H-minor if and only if G has an H-model.

The *genus* of a surface in R^3 is the largest number of non intersecting simple closed curves that can be drawn on the surface without separating it. The sphere is the simplest nontrivial surface in R^3. It has genus 0. The *genus* of a graph is the smallest integer g such that the graph can be embedded in an (orientable) surface of genus g, without edge crossings. Planar graphs are therefore of genus 0. Genus 1 graphs are graphs that can be embedded on the *torus*. As a special case of a seminal result of Robertson and Seymour [14], graphs of genus g are exactly the graphs which have no minor in a finite family $F = F_g$ of graphs. The classical Kuratowski-Wagner Theorem states that $F_0 = \{K_5\,, K_{3,3}\}$ (see [17]). Classes of H-minor free graphs are much more general, however, than classes of genus g graphs. For example, the class of K_5-free graphs contains all the planar graphs, and many other graphs, but there is no bounded genus surface on which all the graphs from this family can be embedded.

We say that a graph $G = (V, E)$ has a (k, α)-*separation*, if V can be partitioned into three parts, A, B, C so that $A \cap B = \emptyset$, $|A \cup C| \leq \alpha|V|$, $|B \cup C| \leq \alpha|V|$, $|C| \leq k$, and if $uv \in E$ and $u \in A$ then $v \notin B$. We say that A and B are separated by C, that C is a separator and that the partition (A, B, C) *exhibits* a (k, α)-separation.

By a seminal result of Lipton and Tarjan [10] a planar graph with n vertices has an $(O(\sqrt{n}), 2n/3)$-separation. In fact, they also show how to compute such a separation in linear 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 A and B until the separated pieces are small enough. This yields a separator tree. Notice that having genus g, as well as being H-minor free, is a hereditary property. More formally, we say that a graph $G = (V, E)$ with n vertices has an *$(f(n), \alpha)$-separator tree* if there exists a full rooted binary tree T so that the following holds:

(i) Each $t \in V(T)$ is associated with some $V_t \subset V$.
(ii) The root of T is associated with V.
(iii) If $t_1, t_2 \in V(T)$ are the two children of $t \in V(T)$ then $V_{t_1} \subset V_t$ and $V_{t_2} \subset V_t$. Furthermore, if $A = V_{t_1}$, $B = V_{t_2}$ and $C = V_t \setminus (V_{t_1} \cup V_{t_2})$ then (A, B, C) exhibits an $(f(|V_t|), \alpha)$-separation of $G[V_t]$ (the subgraph induced by V_t).
(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 (see also Gilbert and Tarjan [7] for a simplified version of their algorithm), and even extended to bounded genus graphs by the result of Gilbert, Hutchinson, and Tarjan [5].

Theorem 3.1 *Let g be a fixed nonnegative integer. Given an embedding of a graph G with n vertices on a surface with genus g, an $(O(\sqrt{n}), 2/3)$-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 [11]. The embedding is purely combinatorial and is given by a *rotation system* (a cyclic permutation π_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. Recently, Reed and Wood [13] exhibited a more flexible algorithm which runs faster, but produces a larger separator.

Theorem 3.2 *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 outputs:* (a) *an H-model of G, or* (b) *a partition (A, B, C) that exhibits an $(O(n^{(2-\epsilon)/3}), 2/3)$-separation.*

3.2 Rank-preserving nested dissection over a field

Lipton, Rose, and Tarjan [9] and Gilbert and Tarjan [7] proved the following result.

Theorem 3.3 *Let C be a symmetric positive definite $n \times n$ matrix. If, for some positive constant $\alpha < 1$, and for some constant $\beta \geq 1/2$, G_C has bounded degree and an $(O(n^\beta), \alpha)$-separator tree, and such a tree is given, then Gaussian elimination on C can be performed with $O(n^{\omega\beta})$ arithmetic operations. The resulting LU factorization of C is given by matrices L and D, $C = LDL^T$, where L is unit lower-triangular and has $\tilde{O}(n^{2\beta})$ non-zero entries, and D is diagonal.*

We note that the requirement that G_C has bounded degree is not needed in Theorem 3.3 if we use a *strong* separator tree (in a strong separator tree the separator vertices at each node of the tree also appear in both children of the node). See [7] for a detailed description of both versions.

Theorem 3.3 cannot be used directly in order to prove Theorem 1.2 and Theorem 1.3. The are several reasons for that. First, notice that the matrix A in Theorems 1.2 and 1.3 may have more than some bounded number of non-zero entries in a row or a column. This problem can, of course, be eliminated by applying Theorem 1.1 to the matrix A, obtaining a matrix B with at most three non-zero entries in each row or column. But then, we are no longer guaranteed that G_B has a good separator tree as G_A had. We will show how to solve this problem. Next, even if we do assume that G_B has a good separator tree we still cannot apply Theorem 3.3 to it, since B is not necessarily a symmetric positive definite matrix. In the case of matrices over the reals we solve this problem as follows. We compute the matrix $C = BB^T$ and recall that (over the reals) $rank(C) = rank(B)$ and $det(C) = det(B)^2$. Furthermore, now C is symmetric and positive semidefinite. Still, we need to prove that G_C has a good separator tree (and compute such a tree) and we need to justify why the nested dissection method works for such positive semidefinite matrices as well. In the case of matrices over finite fields the situation is even more complicated. We are no longer guaranteed that $rank(C) = rank(B)$ nor do we have a notion of positive (semi)definiteness is such fields. We need to overcome these problems as well.

The nested dissection algorithm of Theorem 3.3 is purely combinatorial, and uses fast matrix multiplication as a black box. In fact, the *algebraic* requirement that the matrix be positive definite is needed only to guarantee that no diagonal zero is encountered during the elimination process (so that no row and column pivoting is required). We say that a symmetric matrix C is *pivoting-free* if, whenever a diagonal zero in location (i, i) is encountered during Gaussian elimination of C, then both line i and column i of the current matrix can be completely ignored without affecting the rank.

Notice that for the purpose of computing the rank, we can replace the positive definiteness requirement in Theorem 3.3 with the requirement that C be pivoting-free (if we encounter a diagonal zero we can just skip row and column i entirely).

Now, suppose that B is any real matrix and $C = BB^T$. Then C is symmetric and positive semidefinite (and is positive definite if B is non-singular). It is an immediate exercise to see that C is pivoting-free. Indeed, during the elimination the current matrix we are working on is of the form ZZ^T for some Z. It therefore suffices to prove that if Z is any matrix over the reals and $(ZZ^T)(i, i) = 0$ then $(ZZ^T)(i, j) = (ZZ^T)(j, i) = 0$ for all j. This follows immediately from Lemma 19 in [12], or simply by noticing that the only way the inner product of the i'th line of Z with itself can be zero is if the i'th line is entirely zero. In particular, C is pivoting free. We therefore have:

Lemma 3.4 *Let B be a real $n \times n$ matrix and let $C = BB^T$. If, for some positive constant $\alpha < 1$, and for some constant $\beta \geq 1/2$, G_C has bounded degree and an $(O(n^\beta), \alpha)$-separator tree, and such a tree is given, then Gaussian elimination on C can be performed with $O(n^{\omega\beta})$ arithmetic operations. The resulting LU factorization of C is given by matrices L and D, $C = LDL^T$, where L is unit lower-triangular and has $\tilde{O}(n^{2\beta})$ non-zero entries, and D is diagonal.*

Unfortunately, if B is a matrix over some finite field F, then Lemma 3.4 no longer applies. Indeed, the inner product of a non-zero vector with itself can be zero in such fields. For example, over $GF(2)$ we have $(1, 1) \times (1, 1) = 0$. To overcome this problem we proceed as follows. Let $q = |F|$ denote the number of elements of F. Let r be the smallest positive integer for which $q^r > 2n^2$. Let F' be an extension field of F with $|F'| = q^r$ elements (if $q > 2n^2$ then we just set $F' = F$). Let $R = diag(r_1, \ldots, r_n)$ be a random diagonal matrix where each r_i is chosen from F' independently. We prove that:

Lemma 3.5 *Let $B \in F^{n\times n}$ be any matrix. For a randomly chosen $R = diag(r_1, \ldots, r_n)$ where $r_i \in F'$, the matrix $C[R] = BRB^T$ is pivoting-free with probability greater than $1/2$.*

proof: Let $X = diag(x_1, \ldots, x_n)$ denote the variable diagonal matrix where $X(i, i) = x_i$ for $i = 1, \ldots, n$. Con-

sider the symmetric matrix $C[X] = BXB^T$ over the polynomial ring $F[x_1, \ldots, x_n]$. Let $C[X]_i$ denote the top $i \times i$ sub-matrix of $C[X]$ and let B_i denote the top $i \times n$ rectangular sub-matrix of B. Clearly, $C[X]_i = B_i X B_i^T$. Now, suppose $rank(B_i) = t_i$. Let B'_i be a $t_i \times n$ sub-matrix of B_i having full row rank. We choose the t_i rows lexicographically so that B'_i is a sub-matrix of B'_{i+1}. By the Cauchy Binet formula, the determinant of $C[X]'_i = B'_i X (B'_i)^T$ is a non-zero polynomial of degree t_i in the variables $x_1, \ldots, x_n$. Denote this polynomial by $f_i(x_1, \ldots, x_n)$. In particular, this means that $rank(C[X]'_i) = t_i$. But $C[X]'_i$ is a sub-matrix of $C[X]_i$. It follows that $rank(C[X]_i) = t_i$ as well.

Now, when substituting $r_1, \ldots, r_n$ for $x_1, \ldots, x_n$ and using the similar notations $C[R]_i$ and $C[R]'_i = B'_i R (B'_i)^T$ we get that the determinant of $C[R]'_i$ is just $f_i(r_1, \ldots, r_n)$. By the results of Zippel [21] and Schwartz [15], the probability that a random substitution of elements from a field of size q^r for the variables of a polynomial of degree t_i causes the polynomial to vanish with probability less than $t_i/q^r \leq n/q^r < 1/(2n)$. Applying the union bound for $i = 1, \ldots, n$ we have that with probability greater than $1/2$, non of the polynomials $f_1, \ldots, f_n$ vanish. In particular, all of the $C[R]'_i$ are non-singular and hence $rank(C([R]'_i) = t_i$ and hence $rank(C[R]_i) = t_i$ as well.

It remains to show that in the case where $rank(C[R]_i) = t_i$ for $i = 1, \ldots, n$, we have that $C[R]$ is pivoting-free. Consider the matrix B^0 obtained from B by replacing each row i with zeros if and only if $B'_i = B'_{i-1}$ (namely, if row i is a linear combination of previous rows). Clearly, $rank(B^0) = rank(B)$ and each of the $C[R]'_i$ are also sub-matrices of $C^0[R] = B^0 (B^0)^T$. Hence, $rank(C^0[R]_i) = t_i$ as well. We therefore have that performing Gaussian elimination on $C[R]$ and ignoring rows and column whenever diagonal zeros are encountered (as if these rows and column were entirely zero) produces precisely the same result as performing Gaussian elimination on $C^0[R]$, which has the property that whenever a diagonal zero is encountered, then the entire row and column are also zero. It follows that at the end of the elimination the resulting diagonal matrix D that is obtained has $rank(D) = rank(C^0[R]) = rank(C[R]) = t_n$. ■

Using Lemma 3.5, we obtain a lemma analogous to Lemma 3.4 for finite fields.

Lemma 3.6 *Let B be an $n \times n$ over a finite field F with q elements. Let r be the smallest positive integer so that $q^r > 2n^2$ and let F' be an extension field of F with q^r elements. For a randomly chosen $R = diag(r_1, \ldots, r_n)$ where $r_i \in F'$, let $C[R] = BRB^T$. If, for some positive constant $\alpha < 1$, and for some constant $\beta \geq 1/2$, $G_{C[R]}$ has bounded degree and an $(O(n^\beta), \alpha)$-separator tree, and such a tree is given, then there is algorithm that correctly computes $rank(B)$ with probability greater than $1/2$ and which performs $O(n^{\omega\beta})$ arithmetic operations.*

In order to construct the extension field F' with q^r elements we just need to construct an irreducible polynomial of degree r over F_q. 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 [16]. Since in our case $r = O(\log n)$, this has no effect on the running time of our algorithm. Also notice that each arithmetic operation in Lemma 3.6 is an operation in F'. As each such operation corresponds to a polylogarithmic number of operations in F, the overall effect is only a polylogarithmic increase in the running time.

3.3 Labeled vertex splitting

The representing graph G_A of an $n \times n$ matrix A can be labeled to encode the matrix A itself. We can label vertex i 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 direction so 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 described in Section 2 can be thought of as an operation which transforms the current labeled representing graph to the next one.

Suppose G is a labeled representing graph of an $n \times n$ matrix B and suppose that i is a vertex of G and u, v are two neighbors of i. Modify G 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 $\{i, v\}$. We 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 $b_{i,u}$, label $(u, n+2)$ with $b_{u,i}$, label $(n+2, v)$ with $b_{i,v}$, label $(v, n+2)$ with $b_{v,i}$. We call this operation *labeled vertex-splitting*. It is straightforward to verify that each step in the sparsification algorithm corresponds to one labeled vertex splitting (see Figure 1). Equivalently, in the notations of Theorem 1, the representing graph G_B is obtained from the representing graph G_A by a sequence of $t = O(m)$ labeled vertex splittings. Another way to view this is as follows. Suppose we start from the labeled G_A and perform any sequence of t labeled vertex splittings, obtaining a labeled G_B. Then, $rank(B) = rank(A) + 2t$ and $det(B) = det(A)$.

If G_A is a planar graph (resp. bounded genus graph), then the vertex splitting operation can be chosen to preserve planarity (resp. 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 embedding (resp. surface embedding). Thus, the resulting G_B is also planar (resp. of the same genus). This argument no longer holds if we only know that G_A is H-minor free. Nevertheless, in [20] it is proved that one can at least guarantee

that G_B has a separator tree with essentially the same parameters. More precisely, the following result is proved in [20].

Lemma 3.7 *Let ALG be an algorithm that given an n-vertex H-minor free graph, generates a partition (A, B, C) that exhibits an $(O(n^\beta), 2/3)$-separation in $O(n^\gamma)$ time (here $\gamma > 1$ and $\beta \geq 1/2$). Then, given an H-minor free graph G with n vertices, there is a vertex-split graph G' of G of bounded maximum degree (bound depending only on H) so that G' has an $(O(n^\beta), \alpha)$-separator tree where $\alpha < 1$ is a constant that only depends on H. Furthermore, a corresponding separator tree for G' can be constructed in $O(n^\gamma)$ time.*

3.4 Completing the proofs of theorems 1.2 and 1.3

We prove the theorems for the case of fixed minor free graphs, as the case for bounded genus graphs is easier (one applies Theorem 3.1 instead of Theorem 3.2).

Let $A \in F^{n\times n}$, and assume that G_A does not have some fixed H-minor. We begin by applying Lemma 3.7 on G_A where algorithm ALG is the algorithm of Theorem 3.2 taken with the choice $\epsilon = (2\omega-3)/(3+\omega)$. Hence using the notations of Lemma 3.7 we have $\beta = (2-\epsilon)/3 = 3/(3+\omega)$ and $\gamma = 1+\epsilon = 3\omega/(3+\omega)$. Lemma 3.7 constructs a vertex-split graph G' with bounded maximum degree and a corresponding $(O(n^{3/(3+\omega)}), \alpha)$-separator tree for G_B, and does this in $O(n^{3\omega/(3+\omega)})$ time. Considering the vertex splittings as labeled ones, let B be the matrix for which $G' = G_B$ is the graph representing B. Notice that B is an $(n+2t)\times(n+2t)$ matrix where $rank(B) = rank(A)+2t$ and $det(B) = det(A)$. Furthermore, since G_A is H-minor free, its number of edges is $O(n)$ and hence $t = O(n)$ as well. Thus, B is an $O(n) \times O(n)$ square matrix.

Consider first the case where $F = \Re$. We can compute $C = BB^T$ in $O(n)$ time using the naïve algorithm, since each row and each column of B has only a bounded number of nonzero entries (in general, if each row and column have at most d nonzero entries then C can be computed with at most d^2n arithmetic operations). Notice that the graph G_C representing G is simply the graph obtained by squaring the graph G_B (namely, two vertices of G_B having a common neighbor become direct neighbors in G_C). Also notice that G_C has bounded degree (at most the square of the maximum degree of G_B). As shown in [12], an $(O(n^{3/(3+\omega)}), \alpha)$-separator tree for G_C is obtained from the $(O(n^{3/(3+\omega)}), \alpha)$-separator tree for G_B by simply taking each separator in the tree of G_B together with its neighborhood in G_B to be a separator vertex of G_C (we now see why the bounded degree requirement is crucial). We can now apply lemma 3.4 to C which requires $O(n^{\omega\beta}) = O(n^{3\omega/(3+\omega)})$ time in arithmetic operations. From The obtained LU factorization LDL^T we notice that $det(B)^2 = det(C) = det(D)$ and that $det(D)$ can be computed with just $O(n)$ arithmetic operations by multiplying the diagonal elements of D. Similarly, $rank(D) = rank(C) = rank(B)$ can be computed by just counting the number of zeros in the diagonal of D. The overall running time of the algorithm, counting each arithmetic operation as having unit cost, is $O(n^{3\omega/(3+\omega)})$, as required. This completes the proof of Theorem 1.2 for case $F = \Re$ and also the first part of Theorem 1.3.

Consider next the case where F is a finite field. The only difference here is that we now apply Lemma 3.6. As in the case for the matrix C in the previous paragraph, computing $C[R] = BRB^T$ is performed in $O(n)$ time using the naïve algorithm. Again $G_{C[R]}$ has an $(O(n^{3/(3+\omega)}), \alpha)$-separator tree that is obtained from the corresponding separator tree of G_B. By Lemma 3.6 we correctly compute $rank(B)$ with probability greater than $1/2$ using $O(n^{3\omega/(3+\omega)})$ time, counting each arithmetic operation as having unit cost. To increase our success probability of correctly computing $rank(B)$ to $1-1/2^k$ we can apply the algorithm k times and take the largest of the ranks that are found. This completes the proof of Theorem 1.2 for the finite field case.

Finally, we consider the case where A is a matrix with bounded integer entries. We repeat the first part of the algorithm where ALG is applied using Theorem 3.2 with $\epsilon = 1/2$ (which amounts to using as ALG the algorithm of Alon, Seymour, and Thomas [1]). Now we obtain an $(O(n^{3/2})), \alpha)$-separator tree for G_B, in $O(n^{1.5})$ time. The remaining arguments stay precisely the same. We apply lemma 3.4 to C which requires $\tilde{O}(n^{\omega/2})$ time in arithmetic operations and obtain the LU factorization LDL^T. But now, if we want to measure actual bit complexity we notice that since A has bounded integer entries, so does B and so does C. Thus, the numerators and denominators of the rationals that are obtained during the Gaussian elimination process have only $\tilde{O}(n)$ bits, and the bit complexity of each arithmetic operation is therefore $\tilde{O}(n)$. Thus, the overall bit complexity is $\tilde{O}(n^{1+\omega/2})$. By multiplying the rational numbers in the diagonal of D (in $\tilde{O}(n^2)$ time) we obtain $det(C)$ and thus also $det(B)$ and $det(A)$. This completed the second part of Theorem 1.3. ■

The last paragraph also shows that if A has bounded integer entries then the bit complexity of computing $rank(A)$ is also $\tilde{O}(n^{1+\omega/2})$. However, for the purpose of computing the rank we can actually do much better if we allow randomness, as stated in the paragraph after the statement of Theorem 1.2. Indeed, proceeding as in the proof of Theorem 1.2 for the real case, we notice again that the matrix $C = BB^T$ has bounded integer entries, say their absolute values are bounded by some K. We claim that we can perform the Gaussian elimination of C over a large finite field $GF(p)$

where $p = O(n^2)$ is a prime, and we have already shown how to do *this* in *bit complexity* which is $\tilde{O}(n^{3\omega/(3+\omega)})$ (recalling that each arithmetic operation in $GF(p)$ takes only $\tilde{O}(\log n)$ time). Indeed, assume that $rank(C) = t$ over the reals. Thus, C has a $t \times t$ sub-matrix C' whose rank is t, and hence $\det(C') \neq 0$. On the other hand, by Hadamard's Inequality $|det(C')| \leq (tK^2)^{t/2} < (nK)^n$. Since an integer x has $O(\log x)$ prime divisors, choosing a random prime $p = O(nK) \leq O(n^2)$ guarantees that, w.h.p., $det(C') \neq 0$ also in $GF(p)$, and, in particular, $rank(C) = t$ also in $GF(p)$.

4 Concluding remarks and open problems

We have shown how matrix sparsification, together with several additional arguments, enables us to apply the nested dissection method and deduce the rank of matrices that are neither symmetric, nor positive definite. In fact, the matrices can be taken from any finite field. Over the reals, the algorithms are deterministic, but over finite fields, our algorithms need randomness. It would be interesting to show if there is a deterministic algorithm with the same complexity, also for the finite field case.

It would also be interesting to find additional applications for matrix sparsification other than for rank and determinant computations of well-structured matrices. Perhaps the following is plausible: Suppose A is any $n \times n$ matrix with $m = O(n^{1+\epsilon})$ non-zero entries. Can we compute A^k (k fixed) faster than the known methods, possibly in $O(n^2)$ time, if ϵ is a sufficiently small constant? The idea is to use Theorem 1.1 and compute B in $O(n^{1+\epsilon})$ time, then compute B^j for $i = 1, \ldots, f(k)$ in $O(n^{1+\epsilon})$ time (we can certainly do this as B is both row-sparse and column-sparse), and then deduce (this would be the non-trivial part, if at all possible) the entries of A^k from some easy manipulation of the entries of the B^j.

Acknowledgment

I thank Mark Giesbrecht for helpful comments.

References

[1] N. Alon, P.D. Seymour, and R. Thomas. A separator theorem for nonplanar graphs. *Journal of the American Mathematical Society*, 3(4):801–808, 1990.

[2] J. Bunch and J. Hopcroft. Triangular factorization and inversion by fast matrix multiplication. *Mathematics of Computation*, 28:231–236, 1974.

[3] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[4] W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. Faster inversion and other black box matrix computations using efficient block projections. *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)* ACM Press, 2007

[5] J.R. Gilbert, J.P. Hutchinson, and R.E. Tarjan. A separator theorem for graphs of bounded genus. *Journal of Algorithms* 5(3):391–407, 1984.

[6] A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10:345–363, 1973.

[7] J.R. Gilbert and R.E. Tarjan. The analysis of a nested dissection algorithm. *Numerische Mathematik*, 50(4):377-404, 1987.

[8] E. Kaltofen and G. Villard. On the complexity of computing determinants. *Computational Complexity*, 378:91–130, 2004.

[9] R.J. Lipton, D.J. Rose, and R.E. Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358, 1979.

[10] R.J. Lipton and R.E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[11] B. Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics* 12(1):6–26, 1999.

[12] M. Mucha and P. Sankowski. Maximum matchings in planar graphs via Gaussian elimination. *Algorithmica* 45(1):3–20, 2006.

[13] B. Reed and D.R. Wood. Fast separation in a graph with an excluded minor. *Proceedings of the 2005 European Conference on Combinatorics, Graph Theory and Applications (EUROCOMB)*, 45–50, 2005.

[14] N. Robertson and P.D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory Series B*, 92(2):325–357, 2004.

[15] J.T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, 1980.

[16] V. Shoup. Fast construction of irreducible polynomials over finite fields. *Journal of Symboic Computation*, 17(5):371–391, 1994.

[17] C. Thomassen. Kuratowski's Theorem. *Journal of Graph Theory*, 5:225–241, 1981.

[18] D.H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Transactions on Information Theory*, 32(1):54-62, 1986.

[19] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 2(1):77–79, 1981.

[20] R. Yuster and U. Zwick. Maximum matching in graphs with an excluded minor. *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* 108–117, 2007.

[21] R. Zippel. Probabilistic algorithms for sparse polynomials. *Proceedings of Symbolic and Algebraic Computation (EUROSAM)*, Lecture Notes in Computer Science 72:216–226, 1979.