Deterministic algorithms for skewed matrix products*

Konstantin Kutzkov

IT University of Copenhagen
Denmark
konk@itu.dk

Abstract

Recently, Pagh presented a randomized approximation algorithm for the multiplication of real-valued matrices building upon work for detecting the most frequent items in data streams. We continue this line of research and present new *deterministic* matrix multiplication algorithms.

Motivated by applications in data mining, we first consider the case of real-valued, nonnegative n-by-n input matrices A and B, and show how to obtain a deterministic approximation of the weights of individual entries, as well as the entrywise p-norm, of the product AB. The algorithm is simple, space efficient and runs in one pass over the input matrices. For a user defined $b \in (0, n^2)$ the algorithm runs in time $O(nb+n\cdot\mathrm{Sort}(n))$ and space $O(n+b)$ and returns an approximation of the entries of AB within an additive factor of $\|AB\|_{E1}/b$, where $\|C\|_{E1} = \sum_{i,j} |C_{ij}|$ is the entrywise 1-norm of a matrix C and $\mathrm{Sort}(n)$ is the time required to sort n real numbers in linear space. Building upon a result by Berinde et al. we show that for skewed matrix products (a common situation in many real-life applications) the algorithm is more efficient and achieves better approximation guarantees than previously known randomized algorithms.

When the input matrices are not restricted to nonnegative entries, we present a new deterministic group testing algorithm detecting nonzero entries in the matrix product with large absolute value. The algorithm is clearly outperformed by randomized matrix multiplication algorithms, but as a byproduct we obtain the first $O(n^{2+\varepsilon})$-time deterministic algorithm for matrix products with $O(\sqrt{n})$ nonzero entries.

1998 ACM Subject Classification F.2.0 Analysis of algorithms and problem complexity

Keywords and phrases approximate deterministic memory-efficient matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.466

1 Introduction

The complexity of matrix multiplication is one of the fundamental problems in theoretical computer science. Since Strassen's sub-cubic algorithm for matrix multiplication over a ring from the late 1960's [33], the topic has received considerable attention, see [9] for a historical overview on the subject. It is conjectured that matrix multiplication admits an algorithm running in time $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$. For more than 20 years the record holder was the algorithm by Coppersmith and Winograd [14] running in time $O(n^{2.376})$. Recently two results improving on [14] were announced. In his PhD thesis Stothers [32] presents a refinement of the Coppersmith-Winograd algorithm running in time $O(n^{2.3737})$ and Vassilevska Williams [35] developes a general framework for obtaining a tighter upper bound on the complexity of the Coppersmith-Winograd algorithm. The latter yields the

* Work supported by the Danish National Research Council under the Sapere Aude program.

30th Symposium on Theoretical Aspects of Computer Science (STACS'13).
Editors: Natacha Portier and Thomas Wilke; pp. 466–477

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


best known bound of $O(n^{2.3727})$. Several algorithms computing exactly the matrix product for special classes have been designed. For example, algorithms with running time better than $O(n^{2.3727})$ are known for Boolean matrix multiplication with sparse output [25] or for the case when the input or output matrices are sparse [2, 34]. In a recent work Iwen and Spencer [23] present a new class of matrices whose product can be computed in time $O(n^{2+\varepsilon})$ by a deterministic algorithm: namely when the output matrix is guaranteed to contain at most $n^{0.29462}$ non-zero entries in each column (or by symmetry row). All improved algorithms use as a black-box the algebraic matrix multiplication algorithm which, unfortunately, is only of theoretical importance. It uses sophisticated algebraic approaches resulting in large constants hidden in the big-Oh notation and does not admit an efficient implementation. This motivates the need of simple "combinatorial-like" algorithms.

Approximate matrix multiplication. The first approximation algorithm with rigorously understood complexity by Cohen and Lewis is based on sampling [13]. For input matrices with nonnegative entries they show a concentration around the estimate for individual entries in the product matrix with high probability.

The amount of data to be handled has been growing at a faster rate than the available memory in modern computers. Algorithms for massive data sets, where only sequential access to the input is allowed, have become a major research topic in computer science in the last decade. Drineas et al. [18] first recognized the need for memory-efficient methods for matrix multiplication when access to single columns and rows of the input matrices is possible. They present a randomized algorithm for approximating the product of two matrices based on sampling. The complexity of the algorithm as well as its accuracy depend on user-defined parameters. The algorithm is "pass-efficient" since columns and rows of the input matrices are sequentially loaded into memory. The obtained approximation guarantees are expressed in terms of the Frobenius norm of the input matrices and the user-defined parameters but their method does not result in a strong guarantee for individual entries in the output matrix. Sarlós [30] observed that instead of sampling rows and columns one can use random projections to obtain a sketch of the matrix product. A notable difference to [18] is that by sketching one obtains an additive error for each individual entry depending on the 2-norm of the corresponding row and column vector in the input matrices.

Recently Pagh [28] introduced a new randomized approximation algorithm. Instead of sketching the input matrices and then multiplying the resulting smaller matrices, we treat the product as a stream of outer products and sketch each outer product. Using Fast Fourier Transformation in a clever way, Pagh shows how to efficiently adapt the Count-Sketch algorithm [11] to an outer product. The algorithm runs in one pass over the input matrices and provides approximation guarantees in terms of the Frobenius norm of their product.

Our contribution.

- A new algorithm for the case where the input matrices consist of nonnegative entries only. This is the first nontrivial deterministic approximation algorithm for the multiplication of nonnegative matrices in a streaming setting. Motivated by practical applications, we analyze the approximation guarantee and the algorithm complexity under the assumption that the entries adhere to Zipfian distribution. We compare it to previously known randomized algorithms and show that for certain natural settings it is more efficient and achieves better approximation guarantees.
- We present a new matrix multiplication algorithm for arbitrary real-valued input matrices by adapting the group testing algorithm for streams with updates in the turnstile model

outlined in [27] for detecting the entries with large absolute value in matrix product. As a byproduct we obtain the first deterministic algorithm running in $O(n^{2+\varepsilon})$ steps for matrix products with $O(\sqrt{n})$ nonzero entries.

Note that our algorithms easily generalize to rectangular matrix multiplication but for the ease of presentation we consider the case of square input matrices. Also, we will state the time complexity of our first algorithm using a function $\mathrm{Sort}(n)$ denoting the running time of a linear space deterministic sorting algorithm. Clearly, $\mathrm{Sort}(n) = O(n \log n)$ for comparison based sorting but under some assumptions on the elements to be sorted also better bounds are known, e.g. the $O(n \log\log n)$ time integer sorting algorithm by Han [21].

2 Preliminaries

2.1 Definitions

Linear algebra. Let $\mathbb{R}_+$ denote the field of nonnegative real numbers. Given matrices $A, B \in {\mathbb{R}_+}^{n\times n}$ we denote their product by $C := AB$. The ith row of a matrix A is written as $A_{i,*}$, the jth column as $A_{*,j}$. We use the term *entry* to identify a position in the matrix, not its value. Thus, *the weight* of the entry (i, j) in A is the value in the ith row and jth column, A_{ij}, $i, j \in [n]$, for $[n] := \{0, 1, \ldots, n-1\}$. When clear from the context however, we will omit weight. For example, nonzero entries will refer to entries whose weight is different from 0 and by heavy entries we mean entries with large weight.

The *outer product* of a column vector $u \in {\mathbb{R}_+}^n$ and a row vector $v \in {\mathbb{R}_+}^n$ is a matrix $uv \in {\mathbb{R}_+}^{n\times n}$ such that $uv_{i,j} = u_i v_j$, $i, j \in [n]$. The *rank* of a positive real number $a \in \mathbb{R}_+$ in a matrix A, denoted as $r_A(a)$, is the number of entries strictly smaller than a, plus 1. Note that a does not need to be present in A.

The *p-norm* of a vector $u \in \mathbb{R}^n$ is $\|u\|_p = (\sum_{i=1}^n |u_i|^p)^{\frac{1}{p}}$ for $p > 0$. Similarly, we define the *entrywise p-norm* of a matrix $A \in \mathbb{R}^{n\times n}$ as $\|A\|_{Ep} := (\sum_{i,j\in[n]} |A_{i,j}|^p)^{1/p}$ for $p \in \mathbb{N}$. The case $p = 2$ is the *Frobenius norm* of A denoted as $\|A\|_F$. The *k-residual entrywise p-norm* $\|A\|_{E^k p}$ is the entrywise p-norm of the matrix obtained from A after replacing the k entries with the largest absolute values in A, ties resolved arbitrarily, with 0.

Data streaming. Our algorithms have strong connection to data streaming, therefore we will use the respective terminology. A *stream* S is a sequence of N updates (i, v) for items $i \in \mathcal{I}$ and $v \in \mathbb{R}$. We assume $\mathcal{I} = [n]$. The *frequency* of i is $f_i = \sum_{(i,v)\in S} v$ and $\mathbf{f}_S = (f_0, \ldots, f_{n-1})$ is the *frequency vector* of the stream S. The *insert-only* model assumes $v > 0$ for all updates and in the *non-strict turnstile* model there are no restrictions on v and the values in $\mathbf{f}_S$ [27]. Similarly to matrix entries, we will also refer to the frequency of an item i as the *weight* of i. Items with weight above $||f||_1/b$, for a user-defined b, will be called *b-heavy hitters* or just *heavy hitters* when b is clear from the context. Ordering the items in S according to their absolute weight, the heaviest b items in S are called the *top-b entries* in S.

Skewed distributions. A common formalization of the skewness in real-life datasets is the assumption of *Zipfian distribution* [36]. The elements in a given set M over N different elements with positive weights follow *Zipfian distribution* with parameter $z > 0$ if the weight of the element of rank i is $\frac{|M|}{\zeta(z)i^z}$ where $\zeta(z) = \sum_{i=1}^N i^{-z}$ and $|M|$ denotes the total weight of elements in M. We will analyze only the case when the skew in the data is not light and $z > 1$. For $z > 1$, $\sum_{i=1}^N i^{-z}$ converges to a small constant. We will also use the facts that for $z > 1$, $\sum_{i=b+1}^N i^{-z} = O(b^{1-z})$ and for $z > 1/2$, $\sum_{i=b+1}^N i^{-2z} = O(b^{1-2z})$.

2.2 The column row method and memory efficient matrix multiplication

The naïve algorithm for the multiplication of input matrices $A, B \in \mathbb{R}^{n \times n}$ works by computing the inner product of $A_{i,*}$ and $B_{*,j}$ in order to obtain AB_{ij} for all $i, j \in [n]$. An alternative view of the approach is the *column row method* computing the sum of outer products $\sum_{i \in [n]} A_{*,i} B_{i,*}$. While this approach does not yield a better running time, it turns out to admit algorithmic modifications resulting in more efficient algorithms. Schnorr and Subramanian [31] and Lingas [25] build upon the approach and obtain faster algorithms for Boolean matrix multiplication. Assuming that A is stored as column-major ordered triples and B as row-major ordered triples [8], the approach yields a memory efficient algorithm since the matrix product AB can be computed in a single scan over the input matrices. Recently, Pagh [28] presented a new randomized algorithm combining the approach with frequent items mining algorithms [1, 11]. Inspired by this, we present another approach to modify the column-row method building upon ideas from deterministic frequent items mining algorithms [15, 16, 24, 26].

3 An algorithm for nonnegative matrix products

3.1 Intuition and key lemma

Recall first how the MAJORITY algorithm [6] works. We are given a multiset M of cardinality N and want to find a majority element, i.e. an element occurring at least $N/2 + 1$ times in M. While there are two distinct objects in M we remove them from M. It is easy to see that if there exists a majority element a, at the end only occurrences of a will be in M.

The FREQUENT algorithm [16, 24, 26] builds upon this simple idea and detects b-heavy hitters in an unweighted stream S of N updates $(i, 1)$ for items $i \in [n]$. We keep a *summary* of b distinct entries together with a counter lower bounding their weight. Whenever a new item i arrives we check whether it is already in the summary and, if so, update the corresponding counter. Otherwise, if there is an empty slot in the summary we insert i with a counter set to 1. In the case all b slots are occupied we decrease the weight of all items by 1 and proceed with the next item in the stream. The last step corresponds to removing $b + 1$ distinct items from the multiset of items occurring in S and a simple argument shows that b-heavy hitters will be in the summary after processing the stream. By returning the estimated weight of the item in the summary and 0 for not recorded items, the weight of each item is underestimated by at most $\|\mathbf{f}\|_1 / b$ where $\mathbf{f}$ is the frequency vector of the stream. Implementing the summary as a hash table and charging the cost of each item deletion to the cost incurred at its arrival the expected amortized cost per item update is constant. A sophisticated approach for decreasing the items weights in the summary leads to a worst case constant time per item update [16, 24].

Generalizing to nonnegative matrix multiplication by the column row method is intuitive. Assume the input matrices consist of {0,1}-valued entries only. We successively generate the n outer products and run the FREQUENT algorithm on the resulting stream associating entries with items. There are several problems to resolve: First, we want to multiply arbitrary nonnegative matrices, thus our algorithm has to handle weighted updates. Second, we have to consider $\Theta(n^3)$ occurrences of weighted items in the stream. Third, we cannot apply any more the amortized argument for the running time analysis since a group of $b - 1$ heavy items might be followed by many lighter items causing expensive updates of the summary and it is not obvious how to extend the deterministic approach from [16, 24] guaranteeing

function ComputeSummary

Require: matrices $A, B \in {\mathbb{R}_+}^{n \times n}$, summary $\mathcal{S}$ for b entries

for $i \in [n]$ **do**
  Denote by $R := A_{*,i} \cdot B_{i,*}$ the outer product of the ith column of A and ith row of B
  Find the weight w_{b+1}^R of the entry of rank $b+1$ in R
  Let $\mathcal{L}$ be the b entries in R with rank less than $b+1$, i.e. the largest b entries
  Decrease the weight of each entry in $\mathcal{L}$ by w_{b+1}^R
  for each entry e occurring in $\mathcal{S}$ *and* $\mathcal{L}$ **do**
    add e's weight in $\mathcal{L}$ to e's weight in $\mathcal{S}$
    remove e from $\mathcal{L}$
  Find the weight $w_{b+1}^{\mathcal{S}\cup\mathcal{L}}$ of the entry of rank $b+1$ in $\mathcal{S} \cup \mathcal{L}$, if any
  Update $\mathcal{S}$ to contain the largest b entries in $\mathcal{S} \cup \mathcal{L}$ and decrease their weight by $w_{b+1}^{\mathcal{S}\cup\mathcal{L}}$

function EstimateEntry

Require: Entry (i, j)

if (i, j) is in the summary $\mathcal{S}$ **then**
  return the weight of (i, j) in $\mathcal{S}$
else
  return 0

Figure 1 A high-level pseudocode description of the algorithm. In ComputeSummary we iterate over the n outer products and to each one of them apply Lemma 1 such that only the b heaviest entries remain. We update the summary with the entries output by the outer product. After processing the input matrices we can estimate the weight of an individual entry by checking the summary.

constant time updates in the worst case.

The first issue is easily resolved by the following

▶ Lemma 1. Let $\mathbf{f}$ be the frequency vector of an insert only stream S over a domain $[n]$. After successively decrementing t times the weight of at least b distinct items by $\Delta_i > 0$, $1 \le i \le t$, such that at each step $f_i \ge 0$ for all $0 \le i \le n-1$, it holds $f_k > 0$ for all b-heavy hitters, $k \in [n]$, for all $t \in \mathbb{N}$.

Proof. Since $f_i \ge 0$ for all $i \in [n]$ holds, the total decrease is bounded by $\|\mathbf{f}\|_\mathbf{1}$. A decrement of Δ_i in the weight of a given item is witnessed by the same decrement in the weights of at least $b-1$ different items. Thus, we have $b \sum_{i=1}^{t} \Delta_i \le \|\mathbf{f}\|_\mathbf{1}$ which bounds the possible decrease in the weight of a heavy hitter to $\|\mathbf{f}\|_\mathbf{1}/b$. ◀

In the next section we show that the specific structure of an outer product allows us to design efficient algorithms resolving the last two issues.

3.2 The algorithm

We assume that $A \in \mathbb{R}_+^{n \times n}$ is stored in column-major order and $B \in \mathbb{R}_+^{n \times n}$ in row-major order. We show how to modify the column row method in order to obtain an additive approximation of each entry in AB in terms of the entrywise 1-norm of AB.

Essentially, we run the Frequent algorithm for the stream of n outer products: we keep a summary $\mathcal{S}$ of b distinct items and for each outer product we want to update the

summary with the incoming weighted entries over the domain $[n] \times [n]$. The main difference is that for $b = o(n^2)$ we can use the specific structure of an outer product and update the summary in $o(n^2)$ steps. In ComputeSummary in Figure 1 for each of the n outer products we simulate the successive application of Lemma 1 until at most b entries with weight larger than 0 remain in the outer product. We then update $\mathcal{S}$ with the remaining entries.

Correctness.

▶ Lemma 2. Let w be the weight of an entry (i, j) in the product $C = AB$. After termination of ComputeSummary for the estimated weight $\overline{w}$ of w returned by EstimateEntry, $i, j \in [n]$, holds $\max(w - \|C\|_{E_1}/b, 0) \leq \overline{w} \leq w$.

Proof. The product AB equals $\sum_{i=0}^{n-1} a_i \cdot b_i$ for the columns $a_0, \ldots, a_{n-1}$ of A and the rows $b_0, \ldots, b_{n-1}$ of B. We consider each outer product as n^2 updates for different entries over the domain $[n] \times [n]$ in an insert only stream with positive real weights. We show how the algorithm updates the summary for a single outer product R. First, in line 3 the algorithm finds the entry of rank $b + 1$ in R. In line 4 we decrease the weight of the b largest entries by w^R_{b+1} which yields the same result as the following iterative procedure: While there are at least $b + 1$ nonzero entries in R, find the entry with smallest weight $w_{\min}$ in R and decrease the weight of all non-zero entries by $w_{\min}$. Equivalence holds because we always decrease the weight of an entry with the smallest weight and thus the decrease of the largest b entries weights can never exceed w^R_{b+1}. Also, the decrease can not be smaller than w^R_{b+1} since otherwise we would have more than b non-zero entries in the outer product. Thus, we always decrease by the same amount the weight of at least $b + 1$ different entries which by Lemma 1 guarantees the claimed approximation error. In lines 6–10 we apply essentially the same procedure again for the nonzero entries in the outer product and the entries in the summary. The remaining at most b nonzero entries constitute the updated summary. ◀

Running time. In the following lemmas we present efficient deterministic algorithms for the subroutines used in ComputeSummary. We concentrate how the algorithm updates the summary for a single outer product. Before presenting our approach, we give the main building blocks that will be used to achieve an efficient solution.

▶ Lemma 3. Given two sorted vectors $u, v \in {\mathbb{R}_+}^n$ we can find the entry of rank b in the outer product uv in time and space $O(n)$.

Proof. First note that we can ignore all $u_i = 0$ and $v_i = 0$ since they will result in a row, respectively column, in the outer product with all entries having weight 0, and clearly we do not need to update the summary with such entries. We reduce the problem to selection of the element of rank b in a Cartesian sum $X + Y = \{x + y : x \in X, y \in Y\}$ for sorted sets of real numbers X and Y. Setting $U = \{\log u_i : u_i \in u\}$ and $V = \{\log v_i : v_i \in v\}$ and searching in the Cartesian sum $U + V$ for the element of rank b corresponds to searching for the entry of rank b in the outer product uv, this follows from monotonicity of the $\log : \mathbb{R}_+ \backslash \{0\} \to \mathbb{R}$ function. The best known deterministic algorithm for selection in a Cartesian sum of two sorted sets [19] runs in time and space $O(n)$. ◀

▶ Lemma 4. Given vectors $u, v \in {\mathbb{R}_+}^n$, with elements sorted in descending order, we can output an implicit representation of the largest b elements from the outer product uv in a data structure $\mathcal{L}$ in time and space $O(n)$.

Proof. Assume we have found the entry of rank b as outlined in Lemma 3, let this element be c. Let i, j be two pointers for u and v respectively. Initialize $i = 0, j = n - 1$. Assume

i is fixed. We compare c to $u_i v_j$. While it is larger or equal, we move left v's pointer by decreasing j by 1. At the end we add the pair (i, j) to $\mathcal{L}$, denoting that the entries in ith row of uv bigger than c, and thus of rank less than b, are all (i, ℓ) for $\ell \leq j$. Then we go to the next row in uv by incrementing i and repeat the above while-loop starting with the current value of j. When $i = n$ or $j = 0$ we know that all entries smaller than c have been found. Correctness is immediate since the product $u_i v_j$ is monotonically increasing with i and decreasing with j, and thus for each row of the outer product we record the position of the entries smaller than c in $\mathcal{L}$. Both i and j are always incremented or respectively decremented, thus the running time is linear in n. We need to explicitly store only u, v and $\mathcal{L}$, this gives the claimed space usage. ◀

Next we present an efficient approach for updating the summary for a given outer product after finding the entries of rank at most b.

▶ Lemma 5. For a given outer product uv, $u, v \in {\mathbb{R}_+}^n$ we update the summary $\mathcal{S}$ in time $O(b + \text{sort}(n))$.

Proof. We first sort the vectors u and v in decreasing order according to the values u_i and v_i, respectively. Let us call the sorted vectors u^s and v^s. Each entry in u^s and v^s will be of the form (val, pos) such that $u_{pos} = val$ and $v_{pos} = val$, respectively, i.e. pos will record the position of a given value in u and v. We define the entry (i, j) in the outer product $u^s v^s$ as a $(val_u val_v, (pos_u, pos_v))$ such that $u^s_i = (val_u, pos_u)$ and $u^s_j = (val_v, pos_v)$. Comparing the entries on the $val_u val_v$ values, we can assume that we compute the outer product of two sorted vectors.

Assume we have computed the data structure $\mathcal{L}$ implicitly representing the largest b entries in $u^s v^s$, as shown in Lemma 4. Now we show how to update the summary with the entries in $\mathcal{L}$ in time $O(b + \text{sort}(n))$. We introduce a *position total order* on entries such that $(i_1, j_1) < (i_2, j_2)$ iff $i_1 n + j_1 < i_2 n + j_2$, $i, j \in [n]$. We will keep the entries in $\mathcal{S}$ in the summary sorted according to this order. Assume we can output the b heaviest entries from a given outer product sorted according to the position total order in $\mathcal{L}$. Then in a merge-like scan through $\mathcal{S}$ and $\mathcal{L}$ we update the entries in $\mathcal{S} \cap \mathcal{L}$, remove those from $\mathcal{L}$ and obtain a sorted data structure containing the entries from $\mathcal{S}$ and $\mathcal{L}$ in $O(b)$ steps. The entry of rank $b + 1$ in the set $\mathcal{L} \cup \mathcal{S}$, which has size at most $2b$, can be found in $O(b)$ by [4]. Thus, if the entries in $\mathcal{L}$ are sorted according to the position total order, updating the summary will run in $O(b)$ steps.

We output the b heaviest entries sorted according to the position total order by the following algorithm. Let $\mathcal{L}$ be implicitly given as a sorted column vector u^s and a sorted row vector v^s as described above, and $\ell \leq n$ integer pairs (q, r_q) denoting that in the qth row in the outer product $u^s v^s$ the first $r_q > 0$ entries have rank not more than b. Clearly, r_q will monotonically decrease as q increases. We start with $q = \ell$, namely the shortest interval, sort the r_q entries according to the position total order. We then decrease q by 1 and sort the next r_{q-1} entries according to the position total order. However, we observe that due to monotonicity $r_{q-1} \geq r_q$ and all elements from v^s appearing in the qth row of $u^s v^s$ also appear in the $(q-1)$th row. Thus, we can sort only the new $r_{q-1} - r_q$ elements and then merge the result with the already sorted r_q elements. We continue like this until the elements in each row of the outer product have been sorted. Then we sort the elements in the column vector u^s according to their position, keeping a pointer to the corresponding sorted subinterval of v^s for each entry in u^s. From this we build the set $\mathcal{L}$ with entries sorted according the position total order. By setting $r_{\ell+1} = 0$ the running time for the ℓ mergings and sortings amounts to $\sum_{i=0}^{\ell} (r_i + \text{Sort}(r_i - r_{i+1}))$. We can bound this sum by $O(b + \text{Sort}(n))$ since $\sum_{i=0}^{\ell} r_i = b$,

$\sum_{i=0}^{\ell}(r_i - r_{i+1}) \leq n$ and $\sum_{i=0}^{n-1} f(x_i) \leq f(\sum_{i=0}^{n-1} x_i)$ for a monotonically increasing convex function f and numbers x_i, $i \in [n]$, in its domain. ◀

3.3 Analysis of the approximation guarantee

The only remaining component is how to efficiently answer queries to the summary after processing all outer products. We use a static dictionary with constant look-up time. Observing that the entries are from a universe of size n^2, the best known result by Ružić [29] provides a construction in time $O(b \log^2 \log b)$ and space $O(b)$. Note that $b < n^2$, therefore as a first result we observe that Lemmas 2 and 5 immediately yield the following

▶ Lemma 6. Given $n \times n$-matrices A, B with non-negative real entries, there exists a deterministic algorithm approximating the weight of each entry in the product C of A and B within an additive error of $\|C\|_{E_1}/b$. The algorithm runs in time $O(nb + n\text{Sort}(n))$ and space $O(b + n)$ in one pass over the input matrices.

It was first observed by Bose et al. [5] that the FREQUENT algorithm guarantees tighter estimates for items with weight significantly larger than N/b in a stream of length N and summary of size b. Berinde et al. [3] develop a general framework for the analysis of so called *heavy-tolerant* counter based algorithms and show that FREQUENT falls in this class.

▶ Lemma 7. (Bose et al, [5]) For an entry (i, j) in $C = AB$ with weight $\alpha\|C\|_{E1}$, $\alpha b > 1$, after termination of COMPUTESUMMARY it holds $\widehat{C_{ij}} \geq C_{ij} - (1 - \alpha)\|C\|_{E1}/(b - 1)$ where $\widehat{C_{ij}}$ is the approximation of C_{ij} returned by ESTIMATEENTRY(i, j).

▶ Lemma 8. (Berinde et al, [3]) $\|C\|_{E^k 1}/(b - k)$ is an upper bound on the underestimation of any C_{ij} returned by ESTIMATEENTRY(i, j) for any $k \leq b$.

The above lemmas are important since they yield approximation guarantees depending on the residual k-norm of the matrix product, thus for skewed matrix products the approximation is much better than the one provided by Lemma 6.

Sparse recovery. The approximation of the matrix product $C = AB$ in [18, 28, 30] is analyzed in terms of the Frobenius norm of the difference of C and the obtained approximation $\widehat{C}$, i.e $\|C - \widehat{C}\|_F$. By simply creating a sparse matrix with all non-zero estimations in the summary we obtain an approximation of C: the so called k-sparse recovery of a frequency vector $\mathbf{f}$ aims at finding a vector $\widehat{\mathbf{f}}$ with at most k non-zero entries such that the p-norm $\|\mathbf{f} - \widehat{\mathbf{f}}\|_p$ is minimized.

As shown by Berinde et al. [3] the class of heavy-tolerant counter algorithms yields the best known bounds for the sparse recovery in the p-norm. The following Theorem 1 follows from Lemma 5 and their main result.

▶ Theorem 1. Let A, B be nonnegative $n \times n$ real matrices and $C = AB$ their product. There exists a one-pass approximation deterministic algorithm returning a matrix $\widehat{C}$ such that $\|C - \widehat{C}\|_{Ep} \leq (1 + \varepsilon)^{\frac{1}{p}}(\varepsilon/k)^{1-\frac{1}{p}}\|C\|_{E^k 1}$. The algorithm runs in time $O(n \cdot \text{Sort}(n) + (nk)/\varepsilon)$ and uses space $O(n + k/\varepsilon)$ for any $0 < \varepsilon < 1$ and $k \geq 1$.

Clearly, for $k/\varepsilon = o(n^2)$ the algorithm runs in subcubic time and subquadratic memory. In the next paragraph we show that for skewed output matrices ESTIMATEENTRY can provably detect the most significant entries even for modest summary sizes.

Zipfian distributions. In the full version of the paper we discuss the practical motivation for the assumption that the entries in the product adhere to a Zipfian distribution. The results stated below not only give a better understanding of the approximation yielded by the algorithm, but also allow direct comparison to related work.

▶ Lemma 9. (Berinde et al, [3]) If the entries weights in the product matrix follow a Zipfian distribution with parameter $z > 1$, then EstimateEntry with a summary of size b

1. approximates the weight of all entries with rank $i \leq b$ with additive error of $(1 - \frac{1}{\zeta(z)i^z})\frac{\|C\|_{E1}}{b-1}$.
2. estimates the weight of all entries with additive error of $\varepsilon\|C\|_{E1}$ for $b = O((\frac{1}{\varepsilon})^{\frac{1}{z}})$.
3. returns the largest k entries in the matrix product for $b = O(k)$.
4. returns the largest k entries in a correct order for $b = \Omega(k(\frac{k}{z})^{\frac{1}{z}})$.

3.4 Comparison to previous work

The randomized algorithm by Cohen and Lewis [13] for computing the product of nonnegative matrices yields an unbiased estimator of each entry and a concentration around the expected entry weight with high probability. However, their algorithm requires a random walk in a bipartite graph of size $\Theta(n^2)$ space and is thus not space efficient. It is difficult to compare the bounds returned by EstimateEntry to the bounds obtained in [18, 30], but it is natural to compare the guarantee of our estimates to the ones shown by Pagh [28].

The approximation error of the matrix estimation $\widehat{C}$ in [28], $\|C - \widehat{C}\|_F$, is bounded by $(n\|C\|_F)/\sqrt{b}$ with high probability. The running time is $O(n^2 \log n + b \log b \log n)$ and space usage is $O(n + b \log n)$. Our deterministic algorithm achieves an error guarantee of $(1 + \varepsilon)(\varepsilon/k)^{1-\frac{1}{p}}\|C\|_{E^k1}$ for the approximation $\|C - \widehat{C}\|_{Ep}$ for any $p > 0$. For a direct comparison we set $p = 2$, $k = 0$ and $b = \lceil 1/\varepsilon \rceil$ and obtain an approximation error of $\|C\|_{E1}/\sqrt{b}$ which is at most $(n\|C\|_F)/\sqrt{b}$ by Cauchy-Schwarz inequality. The time and space complexity of our algorithm is a polylogarithmic factor better. Note also that the approximation guarantee does not depend on the dimension n as in [28].

For individual entries we achieve an error bounded by $\min_{k\in[b]}\|C\|_{E^k1}/(b-k)$ while [28] shows that the error of the obtained estimates is bounded by $\|C\|_{E^{b/\kappa}1}/\sqrt{b}$ for a suitably chosen constant $\kappa > 2$.

Assuming Zipfian distribution with $z > 1$ the approximation error of the Frobenius norm of the matrix product in [28] is bounded by $O(nb^{-z}\|C\|_{E1})$ with high probability. By setting $k = 0$ our deterministic algorithm achieves $O(\|C\|_{E1}/\sqrt{b})$ for the Frobenius norm approximation error. For an $\varepsilon\|C\|_{E1}$-approximation of individual entries both [28] and our algorithm need a data structure, a sketch or a summary, of size $O((1/\varepsilon)^{\frac{1}{z}})$ but [28] needs to run $O(\log n)$ copies of the algorithm in parallel in order to guarantee that the estimates are correct with high probability. In the full version of the paper we show that the approximation guarantee of our algorithm is better than the one in [28] for some real data sets exhibiting higher skew. However, Pagh's algorithm achieves better bounds for lighter skew when $1/2 < z < 1$ and more important it is not restricted to nonnegative input matrices.

4 An algorithm for arbitrary real-valued matrices

In this section we show how to efficiently extend the deterministic streaming algorithm sketched in [15, 27] to matrix multiplication. The algorithm in [15, 27] works for streams in the non-strict turnstile model where updates are of the form (i, v) for an item i and $v \in \mathbb{R}$. We only give an overview of how it works, a thorough description will appear in the full version of the paper.

A majority item in a stream is an item whose absolute total weight is more than half of the absolute sum of total weights of all other items in the stream. In [15] the authors present an elegant group testing algorithm for finding a majority item in a stream in the non-strict turnstile model. The algorithm works by keeping a counter for each bit set to

1 in the binary representation of the items and a global counter for the total weight of all items processed so far. A new item is processed by updating the corresponding bit counters and the global counter. Once the stream is processed, a candidate for the majority item is constructed from the bit counters and the global counter. In a second pass we verify whether the candidate is indeed a majority item. Assuming the items are from the domain $[m]$, we need $O(\log m)$ counters. In the following we will call this algorithm BinaryMajority. A generalization of the algorithm to find the b items with nonzero weight after processing the stream is presented in Theorem 14 in [27]. Let P be a set of suitably chosen consecutive prime numbers and $|P|$ denote its cardinality. Each item $i \in [n]$ is distributed to p distinct groups, depending on the value $i \bmod p$, for each prime $p \in P$. In each such group we run BinaryMajority. The crucial observation is that for sufficiently large set of primes P, and sufficiently big primes in P, each of the b nonzero items will be isolated in at least one group and therefore BinaryMajority will detect it.

Generalizing to matrix multiplication again builds upon the column row method. We treat the matrix product as a stream of n updates to the n^2 entries by each outer product. We assume that $n = 2^\ell$ for some $\ell > 0$, otherwise we add less than n zero entries to each row and column vector such that the assumption holds. We number the n^2 entries of the matrix product as $0, 1, \ldots, n^2 - 1$ such that the entry in the position (i, j) is assigned a number $i2^\ell + j$, $0 \leq i, j \leq n - 1$. Now observe that the number of an entry in the outer product consists of 2ℓ bits and the term j determines only the least significant ℓ bits while the term $i2^\ell$ determines the most significant ℓ bits. In the sequence $0, 1, \ldots, n^2 - 1$ the elements having 1 in the kth position, $0 \leq k \leq 2\ell - 1$, are the ones in positions $\{2^k + i2^{k+1}, \ldots, (i+1)2^{k+1} - 1\}$, $0 \leq i \leq 2^{2\ell-k-1}$. Thus, given a column vector a of A and a row vector b of B the entries in the outer product ab, whose numbers have the kth bit equal to 1, are uniquely determined by the position of the contribution from either a or b. This means that in order to obtain the total contribution from all entries with the ℓth bit equal to 1, we simply need to nullify half of the entries in either a or b, and compute the sum over all entries weights in the outer product ab. The latter can be efficiently done by computing the sum of all entries weights in each vector and then multiplying the results. Thus, a majority candidate can be constructed in one pass over the input matrices, $O(n^2 \log n)$ steps and linear space.

For distributing the n^2 entries in ab to different groups, we apply the technique presented by Pagh [28]. Let p be a given prime number. We treat the column and row vectors a and b as polynomials of degree $p - 1$: $p_a = \sum_{i=0}^{n-1} a_i x^{i \cdot n \bmod p}$ and $p_b = \sum_{j=0}^{n-1} b_j x^{j \bmod p}$. p_a and p_b are then multiplied by Fast Fourier Transformation in time $O(p \log p)$. The resulting polynomial has degree $2(p - 1)$, thus we add the coefficients that have the same exponent modulo p. The coefficient in front of x^k, $k \in [p]$, is now exactly the total sum of the entries weights equal to k modulo p. The approach can be combined with BinaryMajority, which yields the following

▶ Theorem 2. Let A, B be real $n \times n$ matrices and $C = AB$ their product. If the absolute weight of each of the b entries with largest absolute weight is bigger than $\|C\|_{E^b 1}$, then there exists a deterministic algorithm computing the b heaviest entries exactly in time $O(n^2 + nb^2 \log^3 n \log^2 b)$ and space $O(n + b^2 \log^3 n \log b)$ in two passes over the input.

▶ **Corollary 1.** *Let A, B be real $n \times n$ matrices and $C = AB$ their product. If C has at most b nonzero entries then there exists a deterministic algorithm computing C exactly in time $O(n^2 + nb^2 \log^3 n \log^2 b)$ and space $O(n + b^2 \log^3 n \log b)$ in two passes over the input.*

4.1 Zipfian distribution

For the case when the absolute values of the entries in the outer product adhere to Zipfian distribution with parameter $z > 1$ we obtain the following

▶ Theorem 3. Let the absolute values of the entries weights in a matrix product adhere to Zipfian distribution. Then for user-defined $s > 0$ and $k > 0$ there exists a deterministic algorithm detecting the ks heaviest entries in the product in time $O(s(n^2 + nk^{\frac{2z}{z-1}} \log^3 n \log^2 k))$ and space $O(n + ks + k^{\frac{2z}{z-1}} \log^3 n \log k)$ in $2s$ passes over the input matrices.

4.2 Comparison to previous work

The algorithm seems to be only of theoretical interest. Note that its complexity is much worse than the one achieved by Pagh's randomized one-pass algorithm [28]: the b non-zero entries can be found in time $O(n^2 + nb \log n)$ and space $O(n + b \log n)$ with error probability $O(1/\text{poly}(n))$. Nevertheless since the best known space lower bound for finding b non-zero elements by a deterministic algorithm is $O(b \log n)$ there seems to be potential for improvement. For example Ganguly and Majumder [20] present improvements of the deterministic algorithm from [27] but their techniques are not suitable for our problem.

To the best of our knowledge this is the first deterministic algorithm for computing matrix products in time $O(n^{2+\varepsilon})$ for the case when the product contains at most $O(\sqrt{n})$ non-zero entries. The algorithm by Iwen and Spencer achieves this for an arguably more interesting class of matrix products, namely those with n^β, $\beta \leq 0.29462$, nonzero entries in each row, but the algorithm relies on fast rectangular matrix multiplication and its simple version runs in time $O(n^{2+\beta})$.

Acknowledgements. I would like to thank my supervisor Rasmus Pagh and the anonymous reviewers for valuable comments and suggestions.

References

1 N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci*, 58(1):137–147, 1999.

2 R. R. Amossen and R. Pagh. Faster join-projects and sparse matrix multiplications. *ICDT 2009*, 121–126.

3 R. Berinde, P. Indyk, G. Cormode, M. J. Strauss: Space-optimal heavy hitters with strong error bounds. *ACM Trans. Database Syst.* 35(4): 26 (2010)

4 M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, R. E. Tarjan. Time Bounds for Selection. *J. Comput. Syst. Sci. 7(4)*: 448–461 (1973)

5 P. Bose, E. Kranakis, P. Morin, Y. Tang. Bounds for Frequency Estimation of Packet Streams. *SIROCCO 2003*: 33–42

6 R. Boyer and S. Moore A Fast Majority Vote Algorithm *U. Texas Tech report*, 1982

7 S. Brin, R. Motwani, C. Silverstein. Beyond Market Baskets: Generalizing Association Rules to Correlations. *SIGMOD 1997*: 265–276

8 A. Buluç. Linear Algebraic Primitives for Parallel Computing on Large Graphs. *PhD thesis*, University of California, Santa Barbara.

9 P. Burgisser, M. Clausen, and M. A. Shokrollahi. Algebraic complexity theory. *Springer-Verlag*, 1997

10 A. Campagna and R. Pagh. Finding associations and computing similarity via biased pair sampling. *Knowl. Inf. Syst.* 31(3): 505–526 (2012)

11 M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci*, 312(1):3–15, 2004

12 E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, C. Yang. Finding Interesting Associations without Support Pruning. *IEEE Trans. Knowl. Data Eng.* 13(1): 64–78 (2001)

13 E. Cohen and D. D. Lewis. Approximating matrix multiplication for pattern recognition tasks. *Journal of Algorithms*, 30(2):211–252, 1999

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

15 G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. *ACM Transactions on Database Systems*, 30(1):249–278, 2005.

16 E. D. Demaine, A. López-Ortiz, J. I. Munro. Frequency Estimation of Internet Packet Streams with Limited Space. *ESA 2002*: 348–360

17 S. V. Dongen. Graph Clustering by Flow Simulation. *PhD thesis*, University of Utrecht, 2000

18 P. Drineas, R. Kannan, and M. W. Mahoney. Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication. *SIAM Journal on Computing*, 36(1):132–157, 2006

19 G. N. Frederickson, D. B. Johnson. The Complexity of Selection and Ranking in X+Y and Matrices with Sorted Columns. *J. Comput. Syst. Sci. 24(2)*: 197–208 (1982)

20 S. Ganguly and A. Majumder. Deterministic k-set structure. *PODS 2006*: 280–289

21 Y. Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. *J. Algorithms 50(1)*: 96–105 (2004)

22 J. Han, M. Kamber. Data Mining: Concepts and Techniques *Morgan Kaufmann* 2000

23 M. A. Iwen and C. V. Spencer. A note on compressed sensing and the complexity of matrix multiplication. *Inf. Process. Lett*, 109(10):468–471, 2009

24 R. M. Karp, S. Shenker, C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst. 28*: 51–55 (2003)

25 A. Lingas. A fast output-sensitive algorithm for boolean matrix multiplication. *ESA 2009*, 408–419.

26 J. Misra, D. Gries: Finding Repeated Elements. *Sci. Comput. Program.* 2(2): 143–152 (1982)

27 S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, Vol. 1, Issue 2, 2005

28 R. Pagh. Compressed Matrix Multiplication. *Proceedings of ACM Innovations in Theoretical Computer Science (ITCS)*, 2012

29 M. Ružić. Constructing Efficient Dictionaries in Close to Sorting Time. *ICALP (1) 2008*: 84–95

30 T. Sarlós. Improved Approximation Algorithms for Large Matrices via Random Projections. *FOCS 2006*: 143–152

31 C.-P. Schnorr, C. R. Subramanian. Almost Optimal (on the average) Combinatorial Algorithms for Boolean Matrix Product Witnesses, Computing the Diameter. *RANDOM 1998*: 218–231

32 A. J. Stothers. On the complexity of matrix multiplication. *Ph.D. thesis*, University of Edinburgh, 2010

33 V. Strassen. Gaussian Elimination is not Optimal. *Numer. Math.* 13, 354–356, 1969

34 R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, 1(1):2–13, 2005.

35 V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. *STOC 2012*, 887–898

36 G. Zipf. Human Behavior and The Principle of Least Effort. Addison-Wesley, 1949