Robustifying Sparse Matrix Multiplication

Karl Bringmann
ETH Zürich, Switzerland

Nick Fischer
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Vasileios Nakos
National and Kapodistrian University of Athens, Greece

Abstract

In the seminal sparse matrix multiplication problem the goal is to compute the product of two $n \times n$ matrices when the matrices are sparse, i.e., when the number of nonzeros in the input matrices m_{in} and/or the number of nonzeros in the output matrix m_{out} are much smaller than n^2. In this paper, we explore the generalized problem of (approximately) computing the k *largest* output entries, with an approximation error dependent solely on the smaller entries – from the viewpoint of sparse recovery, this can be seen as a *robust* variant of sparse matrix multiplication. Despite the substantial research dedicated to sparse matrix multiplication, almost no existing algorithms are robust in this sense. The one exception is Pagh's algorithm in time $\widetilde{O}(m_{in} + nk)$ [ITCS '12], and it remained open whether other algorithms can be similarly made robust.

Our principal contribution is a *black-box reduction* from robust sparse matrix multiplication to conventional sparse matrix multiplication with only polylogarithmic overhead. Specifically, we show that any sparse matrix multiplication algorithm with running time $T(n, m_{in}, m_{out})$ can be transformed into a robust algorithm running in time $\widetilde{O}(T(n, m_{in}, k))$. This reduction leverages an extensive toolkit from sparse recovery, and intriguingly, also involves solving a knapsack-type problem.

By plugging in the state-of-the-art algorithm for sparse matrix multiplication by Abboud, Bringmann, Fischer, and Künnemann [SODA'24], we achieve significantly improved bounds such as $O((m_{in} + k)^{1.346})$. Notably, in the regime where $k \geq m_{in}^{1.762}$, our reduction culminates in an *almost-optimal* $k^{1+o(1)}$-time algorithm.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases matrix multiplication, robustness, sparse recovery, Knapsack

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.157

Related Version *Full Version*: http://arxiv.org/abs/2607.01427

1 Introduction

Few problems have sparked as much theoretical and practical research in computer science as the quest for efficient *matrix multiplication* algorithms. Motivated by countless applications, several communities have contributed to the development of Strassen-like algebraic algorithms (see [2] for the current record $\omega < 2.3714$ on the matrix multiplication exponent), lower bounds [7, 56, 60, 45], and specialized practically fast algorithms [4, 26]. An especial focus lies on *sparse* matrix multiplication, where the goal is to achieve faster algorithms when either the *input-sparsity* m_{in} (the number of nonzero entries in A and B) or the *output-sparsity* m_{out} (the number of nonzero entries in AB) is much smaller than n^2. This problem has been studied extensively [30, 68, 3, 49, 54, 44, 65, 36, 29, 27, 43, 58, 18, 1, 5, 6], and many faster algorithms in terms of the three parameters n, m_{in}, m_{out} have been proposed.

In this paper we consider the generalization of sparse matrix multiplication where the output matrix AB is not necessarily sparse, but consists of $k \ll n^2$ *large, significant* entries plus up to n^2 small, insignificant entries which we regard as *noise* – our goal is to recover

the k significant entries (with some small approximation error depending on the noise). This problem is arguably very natural and well-motivated for various applications, for instance in a setting where the output matrix is sparse up to small measurement errors, or whenever the output matrix represents some numerical scores out of which only k scores are expected to be significant; see e.g. [14, 22, 17].

A concrete application in the context of information retrieval and pattern recognition is to search a database of documents (e.g., searching for a phrase on wikipedia). This task is commonly modelled as follows; see e.g. [22] for more details.[1] We fix a set of t *terms* (i.e., important phrases in a search query) and store a database of n documents by a *term-by-document matrix* $A \in \mathbb{R}^{n \times t}$. Here each column is associated to a term, each row is associated to a document, and each entry $A_{i,j}$ stores the frequency with which the term appears in the document. A search query is represented by a length-t vector b of term frequencies, so that the inner product Ab associates to each document a similarity score (called the *cosine similarity*) – the higher the score, the better the respective document matches the query. Therefore, evaluating multiple database queries is to compute the *large* entries in the matrix product AB (where the columns of B are the queries). Another equally important application is to compute all pairs of similar documents, which amounts to computing the large entries of AA^T [14].

Robustification via Sparse Recovery. Viewed through the lens of sparse recovery and compressed sensing, the problem we study is a natural *robustification* of the exact sparse matrix multiplication problem. Similar robustifications are common, e.g. for the seminal *sparse recovery* problem where the goal is to recover a k-sparse vector x from as few linear measurements as possible. In the *robust* sparse recovery problem the vector x is not necessarily k-sparse, but instead consists of k significant entries plus some noise. Formally, the goal is to find a vector x' such that

$$\|x - x'\|_2 \leq (1+\epsilon) \cdot \|x_{-k}\|_2 \qquad (\text{“}\ell_2/\ell_2\text{-guarantee”}),$$

or that

$$\|x - x'\|_\infty \leq \frac{1}{\sqrt{k}} \cdot \|x_{-k}\|_2 \qquad (\text{“}\ell_\infty/\ell_2\text{-guarantee”}).$$

Here, x_{-k} is the vector obtained from x after zeroing out the largest k coordinates. In this way, as desired, the approximation error only depends on the *insignificant* entries of x. The latter ℓ_∞/ℓ_2-guarantee is strictly stronger,[2] and also conceptually more desirable as it prevents missing relevant entries or false outliers. Both the ℓ_∞/ℓ_2 and the ℓ_2/ℓ_2 are very extensively studied in the field of sparse recovery, including Sparse Fourier transforms [31, 40, 33, 53, 41], sketching and streaming algorithms [38, 47], and reconstruction of signals from partial measurements [9, 10, 28, 55, 34, 52, 23], to name a few from a vast list. In the same spirit we formally define the *robust* version of sparse matrix multiplication as follows:

---

[1] This application has first been heuristically stated as an application of approximate matrix multiplication [14, 22, 17] as discussed in Section 1.2; however, upon closer inspection it turns out that this task is modelled more accurately by the robust sparse matrix multiplication problem described here. See also the discussion in [54].

[2] It is easy to verify that ℓ_∞/ℓ_2 is at least as strong as ℓ_2/ℓ_2; see for instance the proof of [55, Theorem 2.1]. To see that the guarantee is strictly stronger, consider a vector x with, say, $0.05k$ coordinates equal to $\frac{2}{\sqrt{k}}$ and where the total ℓ_2-mass of the other coordinates is 1. An algorithm with the ℓ_2/ℓ_2-guarantee (for $\epsilon = 0.1$, say) could as well detect no coordinate (i.e., $x' = 0$ is a feasible solution), whereas any algorithm with ℓ_∞/ℓ_2-guarantee *must* detect all the $0.05k$ special entries.

▶ **Definition 1** (Robust Sparse Matrix Multiplication). *Given matrices $A, B \in \mathbb{R}^{n\times n}$ in sparse representation, and given $k \geq 1$, compute a matrix $C \in \mathbb{R}^{n\times n}$ such that*

$$\|AB - C\|_\infty \leq \frac{1}{\sqrt{k}} \cdot \|(AB)_{-k}\|_F.$$

Here, $(AB)_{-k}$ denotes the matrix obtained from AB by zeroing out the largest k entries (in absolute value).

Note that this is indeed a generalization of sparse matrix multiplication (as by setting $k \geq m_{out}$, we force the error to be $\|(AB)_{-k}\|_F = 0$, and thus $C = AB$). The study of this problem was initiated in 2013 by Pagh [54] who developed the first robust sparse matrix multiplication algorithm in time $\widetilde{O}(m_{in} + kn)$.[3] In a nutshell, his algorithm relies on various tools from sparse recovery and interestingly encodes the problem as the multiplication of two *polynomials* (implemented by the Fast Fourier Transform). Among the vast collection of known sparse matrix multiplication algorithms [30, 68, 3, 49, 54, 44, 65, 36, 29, 27, 43, 58, 18, 1] Pagh's algorithm remained the only algorithm which is robust in the sense of Definition 1, and none of the other algorithms can be easily robustified.

This leaves open whether we could benefit from the ideas and improvements in the other sparse matrix multiplication algorithms to also obtain improved bounds for the robust problem. Specifically, many of these algorithms exploit Strassen-like algebraic fast matrix multiplication – can we obtain improved robust algorithms using fast matrix multiplication?

1.1 Main Result: A Reduction

Motivated by this question we investigated which algorithms can be robustified. We arrived, satisfyingly, at a very general solution: Our main contribution is *black-box reduction* from *robust* sparse matrix multiplication to *exact* sparse matrix multiplication with only polylogarithmic overhead:

▶ **Theorem 2** (Robustifying Reduction). *Suppose that sparse matrix multiplication of $n \times n$ matrices (with m_{in} nonzero inputs and m_{out} nonzero outputs) is in time $T(n, m_{in}, m_{out})$. Then robust sparse matrix multiplication is in time $O(n \log^4 n + \log^3 n \cdot T(n, m_{in}, k)) = \widetilde{O}(T(n, m_{in}, k))$ (by a randomized algorithm that succeeds with high probability).*

The reduction naturally builds on the broad toolkit provided by sparse recovery. Vaguely speaking, our main challenge is to estimate how the large and small entries are distributed among the columns of AB. The innovation in our solution lies in encoding this task as an optimization problem – specifically, an instance of Minimum-Cost Multiple-Choice Knapsack – and observing that this problem can be efficiently approximated. In Section 1.3 we give a more detailed overview.

As a consequence of Theorem 2 we obtain improved algorithms for robust sparse matrix multiplication in almost all regimes. To state these improvements precisely we introduce some notation: Let $\mathrm{MM}(x, y, z)$ denote the running time of algebraic dense rectangular matrix multiplication (i.e., the complexity of multiplying an $x \times y$ matrix by a $y \times z$ matrix). The state-of-the-art of sparse matrix multiplication is a recent algorithm due to Abboud, Bringmann, Fischer and Künnemann [1]. Taking all three parameters n, m_{in}, m_{out} into account this algorithm runs in time

[3] Pagh [54] referred to the problem as *compressed* matrix multiplication. We instead prefer the name *robust sparse* matrix multiplication to be more in line with the sparse recovery terminology.

$$\widetilde{O}\left(\max_{\substack{x,z\leq n\\ xz\leq m_{out}}}\min_{1\leq\Delta\leq n}\left(\Delta m_{in}+\mathrm{MM}\Big(x,\min\Big\{\frac{m_{in}}{\Delta},n\Big\},z\Big)\right)\right).$$

This complicated running time can often be conveniently bounded with respect to the matrix multiplication constants, e.g., by $O((m_{in}+m_{out})^{1.346})$ [1]. In light of Theorem 2 we immediately achieve the same running time for robust sparse matrix multiplication with k in place of m_{out}:

▶ **Corollary 3.** *Robust sparse matrix multiplication is in time $\widetilde{O}((m_{in}+k)^{1.346})$ (by a randomized algorithm that succeeds with high probability).*

This running time improves upon Pagh's $\widetilde{O}(m_{in}+nk)$-time algorithm in many regimes. Specifically, in the natural regime $m_{in}\leq k$, we obtain time $\widetilde{O}(k^{1.346})=\widetilde{O}(n^{0.692}k)$ since $k\leq n^2$. To allow for a clearer comparison, we also bound our running time as follows:

▶ **Corollary 4.** *Robust sparse matrix multiplication is in time $\widetilde{O}(m_{in}+\mathrm{MM}(n,n,\lceil\frac{k}{n}\rceil)\cdot n^{\epsilon})$ for any $\epsilon>0$ (by a randomized algorithm that succeeds with high probability).*[4]

That is, we can basically replace the term nk by the time complexity of multiplying an $n\times n$ by an $n\times\lceil\frac{k}{n}\rceil$ matrix, which can naively be upper bounded by $O(nk)$. Less naively, by using algebraic matrix multiplication we have[5]

$$\mathrm{MM}(n,n,\lceil\frac{k}{n}\rceil)=O(n^2+nk\cdot(n/k)^{3-\omega})=O(n^2+nk\cdot(n/k)^{0.6284}),$$

yielding a polynomial improvement over Pagh's running time $\widetilde{O}(m_{in}+nk)$ whenever $k\geq n^{1+\epsilon}$ and $nk\geq m_{in}^{1+\epsilon}$ (i.e., Pagh's algorithm does not already run in almost-linear time).

The Abboud–Bringmann–Fischer–Künnemann algorithm allows for various other trade-offs with respect to the relation between m_{in} and k, see [1]. Here we will only mention one last interesting setting: When k is sufficiently large, namely $k\geq m_{in}^{1.762}$, then we obtain an *unconditionally almost-optimal* algorithm:

▶ **Corollary 5.** *If $k\geq m_{in}^{1.762}$, then robust sparse matrix multiplication is in time $O(k^{1+\epsilon})$ for any $\epsilon>0$ (by a randomized algorithm that succeeds with high probability).*

Indeed, observe that any algorithm requires time $\Omega(k)$ simply to write the output and therefore the time in Corollary 5 is optimal up to the factor k^{ϵ}. This corollary is ultimately based on the well-known fact that matrix multiplication of sufficiently *rectangular* matrices is in almost-optimal time (i.e., $\mathrm{MM}(n,n,n^{0.31389})=O(n^{2+\epsilon})$ for any $\epsilon>0$, where the constant 0.31389, sometimes called the *dual matrix multiplication exponent*, is the state of the art in a series of improvements [15, 16, 24, 25, 64]).

---

[4] Let us remark that the overhead n^{ϵ} in Corollaries 4 and 5 is a technical artifact from the algebraic matrix multiplication machinery. Specifically, while many papers for the sake of simplicity state that matrix multiplication is in time $O(n^{\omega})$, the technically correct statement is that for every $\epsilon>0$, there is a matrix multiplication algorithm in time $O(n^{\omega+\epsilon})$. Here, as we are dealing with matrix multiplication very directly, we prefer the formally correct statement.

[5] This time bound is clear if $k\leq n$. Otherwise, we split both matrices into submatrices of size at most $\lceil\frac{k}{n}\rceil\times\lceil\frac{k}{n}\rceil$. To compute the original matrix product it then suffices to compute $O((n^2/k)^2)$ square matrix products of size $\lceil\frac{k}{n}\rceil\times\lceil\frac{k}{n}\rceil$ each of which runs in time $O((k/n)^{\omega})$. The total time is $O((n^2/k)^2\cdot(k/n)^{\omega})=O(nk\cdot(n/k)^{3-\omega})$. We remark that this running time can be improved by applying rectangular matrix multiplication for any $k=\Theta(n^{\gamma})$ with $1<\gamma\leq 2$.

1.2 Related Work: Approximate Matrix Multiplication

Our research is also closely related to the *approximate matrix multiplication* problem. Here the goal is similarly to approximate AB by some matrix C in the same sense of bounding some norm of the matrix $AB - C$ (such as the Frobenius or spectral norm). The conceptual difference is that AB is not assumed to be sparse or sparse up to noise, and therefore the goal is not to bound the error in terms of the insignificant entries in AB (i.e., $\|(AB)_{-k}\|_F$) but rather in terms of all entries in AB (i.e., say, $\|AB\|_F$). The typical guarantee is that one can compute some matrix C in time $\widetilde{O}(n^2c)$, for some parameter c, such that

$$\|AB - C\|_F \leq \frac{1}{\sqrt{c}}\|A\|_F\|B\|_F.$$

This can be achieved by two seminal approaches: *sampling*-based approaches due to Cohen and Lewis [14] and Drineas, Kannan and Mahoney [20, 21], and *projection*-based approaches as introduced by Sarlós [59]. These two approaches form the basis of a plethora of later refinements [66, 67] and extensions to other models such as streaming [13]. There was also some recent progress on incorporating fast matrix multiplication techniques in this setup [62].

All of these works are incomparable to our results: Approximate matrix multiplication is well-suited for dense and *flat* matrices (e.g., when most entries have roughly the same size) or whenever an ℓ_2/ℓ_2 guarantee is sufficient, whereas robust sparse matrix multiplication is preferable for *spiky* matrices or whenever an ℓ_∞/ℓ_2 guarantee is necessary. Having said that, one can of course simply apply robust matrix multiplication algorithms to the approximate setting (disregarding the strong robust guarantee). For instance, Pagh applied his algorithm to this setting to achieve a bound of

$$\|AB - C\|_F \leq \sqrt{\frac{n}{c}} \cdot \|AB\|_F$$

in time $\widetilde{O}(n^2c)$. This statement was also very recently obtained in an arXiv preprint [63] (though notably without the stronger robust guarantee). When applying our faster algorithm we obtain the strictly stronger accuracy bound of

$$\|AB - C\|_F \leq \sqrt{\frac{n}{c^{\frac{1}{\omega-2+\epsilon}}}} \cdot \|AB\|_F \leq \sqrt{\frac{n}{c^{2.963}}} \cdot \|AB\|_F$$

in the same time budget $\widetilde{O}(n^2c)$.[6] To reiterate, both Pagh's and our algorithm actually obtain stronger guarantees in terms of $\|(AB)_{-k}\|_F$ for some $k \geq \Omega(n/c)$.

1.3 Technical Overview

In this section we will briefly outline the main ideas behind our results. For illustration, we consider the following query problem. Let $X \in \mathbb{R}^{n\times n}$ be an unknown matrix. We can access X by querying the product LXR for a matrix $L \in \{-1, 0, 1\}^{\ell\times n}$ with column sparsity 1 and a matrix $R \in \{-1, 0, 1\}^{n\times r}$ with row sparsity 1 of our choice; the cost of this query is $\ell \cdot r$. The task is to compute an approximation X' of X with entry-wise error at most $\frac{1}{\sqrt{k}}\|X_{-k}\|_F$, using $\widetilde{O}(1)$ queries of total cost $\widetilde{O}(k)$.[7]

[6] To see this, bound the running time of our algorithm from Corollary 4 by $\mathrm{MM}(n, n, k/n) \leq O(n^4/k^2 \cdot \mathrm{MM}(k/n)) \leq O(n^{4-\omega+\epsilon}k^{\omega+\epsilon-2})$, for any arbitrarily small constant $\epsilon > 0$. Then, setting k such that $c = (k/n)^{\omega+\epsilon-2}$ the running time becomes $O(n^2c)$ and the guarantee becomes as claimed before.

[7] As we will later see, it is often sufficient to only consider one side of the sketch (say L) and leave the other side (say R) trivial. Moreover, it is of course equivalent to assume that the column-sparsity of L and row-sparsity of R is $\tilde{O}(1)$ instead of 1.

Connection to Robust Matrix Multiplication. Let us first explain the connection of the query problem to robust matrix multiplication. Suppose we want to multiply two m_{in}-sparse $n \times n$-matrices A and B. The overall goal is to *compress* the matrices $A, B \in \mathbb{R}^{n \times n}$ to *denser rectangular* matrices $A' \in \mathbb{R}^{x \times n}, B' \in \mathbb{R}^{n \times z}$ in such a way that the product of the outer dimensions becomes small, i.e., $xz \leq \widetilde{O}(k)$. Since the number of non-zeros in the matrix product $A'B'$ is trivially bounded by $xz \leq \widetilde{O}(k)$, we can apply in a black-box manner any sparse matrix multiplication algorithm to evaluate $A'B'$. The challenge is to design the compression in such a way that we can recover the large entries of AB from $A'B'$. (This general framework has been implicitly or explictly employed in various forms in previous work on *exact* matrix multiplication [35, 54, 36, 61, 29, 1], specifically, compared to [1] our goal is to replace the "densification" step by a robust alternative.)

The above query problem models this compression: We set the unknown matrix to $X := AB$. We can answer a query LXR as follows. First compute LA in time $O(m_{in})$, using that L has column sparsity 1. Symmetrically compute BR in time $O(m_{in})$, using that R has row sparsity 1. The remaining product $(LA)(BR)$ has outer dimensions ℓ and r. Thus, if each query has cost $\ell \cdot r = \widetilde{O}(k)$, then each product $(LA)(BR)$ is compressed in the above sense. Therefore, any algorithm for the query problem yields a reduction from robust matrix multiplication to sparse matrix multiplication.

In the remainder we discuss how to solve the query problem.

Warm-Up: The Perfectly Balanced Case. To demonstrate that the sparse recovery toolkit is applicable, consider first the toy case in which all columns in X contain exactly the same number of significant entries, i.e., $t := k/n$ many. Suppose further that the noise is evenly distributed among all columns, i.e.,

$$\|(Xe_j)_{-t}\|_2^2 \leq \frac{\|X_{-k}\|_F^2}{n}$$

for each column vector Xe_j of X (here e_j denotes the j-th unit vector).

The known ℓ_∞/ℓ_2-sparse recovery algorithms, e.g. [38, Theorem 1] can be phrased as follows (see Lemma 7): There is a matrix H of size $\widetilde{O}(t) \times n$ such that, given Hx for some unknown vector $x \in \mathbb{R}^n$, we can efficiently compute a vector x' so that $\|x - x'\|_\infty \leq \frac{1}{\sqrt{t}}\|x_{-t}\|_2$ (with good probability). Moreover, the matrix H has at most $\widetilde{O}(1)$ nonzero entries per column, and can be constructed in time $\widetilde{O}(n)$. A natural idea is to compute HX. Since H has column sparsity $\widetilde{O}(1)$, we can write it as a sum $H = L_1 + \ldots + L_{\widetilde{O}(1)}$ where each L_i has column sparsity 1. Then $L = L_i$ and the identity matrix $R = I_n$ is a valid query, and by summing up the query results over all i we obtain HX. This choice indeed satisfies our requirements: It uses $\widetilde{O}(1)$ many queries and, since H has $\widetilde{O}(t)$ rows and I_n has n columns, each query has cost $\widetilde{O}(t) \cdot n = \widetilde{O}(k)$. Moreover, from each column vector in HX we can efficiently recover an approximation x'_j of the corresponding column vector Xe_j in X with an error of

$$\|Xe_j - x'_j\|_\infty \leq \frac{1}{\sqrt{t}} \cdot \|(Xe_j)_{-t}\|_2 \leq \frac{1}{\sqrt{tn}} \cdot \|X_{-k}\|_F = \frac{1}{\sqrt{k}} \cdot \|X_{-k}\|_F. \tag{1}$$

This is exactly as hoped. However, we critically relied on the unrealistic assumption that both the k largest entries and the noise is evenly distributed among all columns.

The General Unbalanced Case. In the general case we cannot wish for this perfect distribution. Instead, we have to deal with some columns containing more significant entries and/or more noise than others. It could even be that one column has only one significant entry and the ℓ_2 mass of the rest of the elements is much larger than that entry; this means

that we cannot hope to recover that single element with ℓ_∞/ℓ_2 sparse recovery algorithm with $t = 1$ but have to assign a much larger budget. It turns out that the ideas from the toy case would generalize if we would know both the number of significant entries k_j as well as the noise $\|(Xe_j)_{-k_j}\|_2$ for each column j. In this case we could split the columns into polylog(n) groups where in each group all columns share the same statistics (up to constant factors); then the previous approach applies to each group. So perhaps a first instinct is to estimate k_j and $\|(Xe_j)_{-k_j}\|_2$ for each column in a preprocessing step. Unfortunately, this appears to be quite challenging – in particular, but not exclusively, because k_j depends on the entire matrix and not only on the j-th column.

Our solution is a more modest implementation of this approach, where our goal is to assign to each column a *budget* t_j. On the one hand, the budget should satisfy that

$$\frac{1}{\sqrt{t_j}} \cdot \|(Xe_j)_{-t_j}\|_2 \leq \frac{1}{\sqrt{k}} \cdot \|X_{-k}\|_F; \tag{2}$$

note that this is exactly the accuracy we require in (1). However, this condition can easily be enforced by assigning the generous budgets $t_j = n$ to each column. To avoid this we require, on the other hand, that the total budget is small, i.e.,

$$\sum_{j\in[n]} t_j \leq O(k). \tag{3}$$

With the approach outlined before we have reduced the whole problem to finding column budgets $t_1, \ldots, t_n$ satisfying both conditions (2) and (3). This is a natural subtask to which we refer as a "budget allocation sketch". Our design of a budget allocation sketch (Theorem 8) involves two steps that we outline next; they can be seen as the main innovation of this paper. We are confident that this tool finds more independent applications in the future. As proof of concept, in Appendix B we present a completely different application in a distributed communication scenario.

Budget Allocation: Step 1. Our first step is to estimate $\|X_{-k}\|_F$ in the query setting. This step is surprisingly intricate in light of estimating $\|x_{-k}\|_2$ for a *vector* x being already known [52, Lemma 1.5]. Applied to our setting this merely allows us to estimate $\|(Xe_j)_{-k}\|_2$ for the individual columns of X. More generally, we can estimate $\|(Xe_j)_{-t}\|_2$ for all powers $t = 2^0, 2^1, \ldots$. Here, by *estimating* we mean a bi-criteria approximation $v_{j,t}$ with constant estimation error as well as a constant loss in the threshold parameter t:

$$\Omega(1) \cdot \|(Xe_j)_{-O(t)}\|_2^2 \leq v_{j,t} \leq \|(Xe_j)_{-t}\|_2^2.$$

Our insight is that, given these approximations $v_{j,t}$, we can read off an estimate (more precisely, a bi-criteria approximation) of $\|X_{-k}\|_F$ as the optimal value of the following optimization problem: Select, for each column j, a value t_j (as a power of 2) in order to

$$\begin{aligned} \text{minimize} \quad & \sum_{j\in[n]} v_{j,t_j} \\ \text{subject to} \quad & \sum_{j\in[n]} t_j \leq 2k. \end{aligned}$$

For one direction of the claimed statement, it is easy to see that plugging in $t_j = k_j$ (rounded up to a power of 2) satisfies the constraint and therefore $\|X_{-k}\|_F$ provides a valid upper bound on the optimal value. For the other direction, we refer the reader to the formal proof of Lemma 10. Crucially, even solving the above optimization problem *approximately* suffices for our goal.

Computationally, it remains to solve this optimization problem. Our observation is that this problem can be cast as an instance of the *Minimum-Cost Multiple-Choice Knapsack* problem. Here an instance consists of n *items* each associated with a *cost* c_i and a *weight* w_i. Moreover, the items are partitioned into *bundles.* The goal is to select a subset $I \subseteq [n]$ of items – exactly one item from each bundle – so that the total weight is $\sum_{i \in I} w_i \leq W$ (for some given weight threshold W) and so that the total cost $\sum_{i \in I} c_i$ is minimized. This problem has not been explicitly considered before (to the best of our knowledge), but is closely related to the classical *0-1-Knapsack* problem (where the goal is to *maximize* the total *profit* subject to a weight constraint) and the *Multiple Choice Knapsack* problem (where items are similarly grouped into bundles). While both of these problems are NP-hard, it is known that they can be $(1+\epsilon)$-approximated in near-linear time $\widetilde{O}(n \operatorname{poly}(\epsilon^{-1}))$, as was shown for 0-1-Knapsack by Ibarra and Kim [32] and for Multiple Choice Knapsack by Rhee [57]. A long line of research has further optimized the polynomial dependence on ϵ for $0\overline{/}1\overline{/}$Knapsack [48, 42, 57, 11, 37, 19, 51, 12]. We employ Chan's framework for 0-1-Knapsack [11], as it transfers nicely also to the Minimum-Cost version of the problem (with some minor tweaks as described in Appendix A due to space constraints). The consequence is that we can $(1+\epsilon)$-approximate Minimum-Cost Multiple-Choice Knapsack in near-linear time, and thereby approximate $\|X_{-k}\|_F$ as desired.[8]

Budget Allocation: Step 2. The second step is to select the column budgets $t_1, \ldots, t_n$. In fact, simply taking values $t_1, \ldots, t_n$ of an (approximately) optimal solution to the previous optimization problem already gives a nontrivial budget allocation that will eventually lead to an ℓ_2/ℓ_2 guarantee for our query problem (i.e., we could compute a matrix X' such that $\|X - X'\|_F \leq (1+\epsilon) \cdot \|X_{-k}\|_F$). However, it turns out that with little extra effort we can do better and find budgets that lead to the stronger ℓ_∞/ℓ_2-guarantee: For each column j select the budget t_j to be the smallest value t satisfying that

$$\frac{v_{j,t}}{t} \leq \frac{w}{k},$$

where $w \approx \|X_{-k}\|_F$ is the approximation computed in step 1. In this way we make sure that no column receives a prohibitively small budget, and it follows from some calculations that the total budget is still bounded by $O(k)$ (though with a larger constant); see Lemma 12 for more details. Notably, the budget allocation routine runs altogether in *near-linear* time in the input sparsity.

1.4 Outline

We start with some preliminaries in Section 2. In Section 3 we give the formal proof of our key budget allocation step, deferring the discussion of the knapsack-type problem to Appendix A. We then conclude our the robustifying reduction in Section 4. Finally, in

[8] In light of this surprisingly involved algorithm to approximate $\|X_{-k}\|_F$, the critical reader might wonder whether we can simplify here. A first instinct could be to not approximate $\|X_{-k}\|_F$ at all, but instead to spend a logarithmic number of guesses for $\|X_{-k}\|_F$ (up to a constant factor). Unfortunately, this approach has two drawbacks: First, this approach would lead to a running time overhead of $\log(n\Delta)$ (where Δ is the largest entry in the given matrices) which we can avoid with the method outlined above (in other words, the approximation via Minimum-Cost Multiple Choice Knapsack has *strongly* polynomial running time). Second, even if we are willing to take this overhead into account, the more serious problem for our application to matrix multiplication is to verify which of the guesses was valid in the end – that is, we would have to verify whether $\|X - X'\|_\infty \leq \frac{1}{\sqrt{k}} \cdot \|X_{-k}\|_F$. This appears to be a quite nontrivial problem and we see no approach other than estimating $\|X_{-k}\|_F$.

Appendix B we present an interesting alternative application of our new budget allocation sketch for a distributed heavy hitter problem. Due to space constraints, the missing proofs for tools from sparse recovery are deferred to the full version of this paper.

2 Preliminaries

We set $[n] = \{1, \dots, n\}$, and write $\mathrm{polylog}(n) = (\log n)^{O(1)}$ and $\widetilde{O}(n) = n \cdot \mathrm{polylog}(n)$.

Matrix Notation. Let $A \in \mathbb{R}^{n \times n}$ be a matrix. We write $A[i,j]$ for the entry at position (i,j). The *Frobenius* norm is defined as $\|A\|_F = (\sum_{i,j} A[i,j]^2)^{1/2}$, and the ℓ_∞-norm is defined as $\|A\|_\infty = \max_{i,j} |A[i,j]|$. We write A_{-k} for the matrix obtained from A after zeroing out the largest k entries in absolute value (breaking ties in an arbitrary but consistent way). We say that A is *s-sparse* if it contains at most s nonzero entries, and we say that A is *s-column sparse* if each column of A contains at most s nonzero entries.

Machine Model. We work in the standard WordRAM model with word size $\Theta(\log n)$ (where n is the dimension). To simplify notation, throughout the paper we implicitly assume that all vectors $x \in \mathbb{R}^n$ and matrices $A \in \mathbb{R}^{m \times n}$ have entries that can be represented in a $O(\log n)$-bit fixed-precision format; the same applies to complex-valued vectors and matrices. This means that each entry can be stored in a machine word (or in a constant number of machine words).

Alternatively, our results also hold on the RealRAM model, where a machine word can store an arbitrary real number; in this case vectors and matrices can have arbitrary real entries.

Randomization. We say that an event happens *with high probability* if it happens with probability $1 - 1/n^c$ for an arbitrarily large prespecified constant c. Unless stated otherwise, all algorithms are Monte Carlo algorithms that succeed with high probability.

Tools from Sparse Recovery

Throughout we make extensive use of techniques from sparse recovery. Specifically, we will rely on the following two lemmas; the proofs are deferred to the full version of this paper. The following lemma is basically the same as [52, Lemma 1.4]. The only difference is that we use random signs instead of Gaussian random variables used in that proof in order to achieve anti-concentration needed for the left-hand side of the inequality.

▶ **Lemma 6** (ℓ_2-Tail Estimation). *Let $n \geq 1, n \geq s \geq 0$ be integers and let $\delta > 0$. In time and space $O(\log(1/\delta))$ we can sample a matrix $S^{(s)} \in \{-1, 0, 1\}^{m \times n}$ with $m = O(\log(1/\delta))$ rows and the following properties. (i) Any entry of $S^{(s)}$ can be computed in time $O(1)$. (ii) For any $x \in \mathbb{R}^n$, given $S^{(s)}x$ we can compute in time $O(\log(1/\delta))$ a value v which satisfies with probability at least $1 - \delta$ that*

$$\frac{1}{64} \cdot \|x_{-512s}\|_2^2 \leq v \leq \|x_{-s}\|_2^2.$$

The following lemma also follows from [38, 54, 46]. In the full version of this paper we give an accessible self-contained proof which is close in spirit to the one in [54].

▶ **Lemma 7** (Fast Heavy-Hitter Recovery). *Let $n \geq 1, s \geq 0$ be integers and let $\delta > 0$. In time and space $O(\log n \log(s/\delta))$ we can sample a matrix $H^{(s)} \in \{-1, 0, 1\}^{m \times n}$ with $m = O(s \log n \log(s/\delta))$ rows and the following properties. (i) The column sparsity of $H^{(s)}$ is $O(\log n \log(s/\delta))$, and any column of the $H^{(s)}$ can be computed in time $O(\log n \log(s/\delta))$. (ii) For any $x \in \mathbb{R}^n$, given $H^{(s)}x$ we can compute in time $O(m)$ an $O(s)$-sparse vector x' which satisfies with probability at least $1 - \delta$ that*

$$\|x - x'\|_\infty \leq \frac{1}{\sqrt{s+1}} \cdot \|x_{-s}\|_2.$$

3 Budget Allocation Sketch

The goal of this section is to prove the following Theorem 8 which captures the "budget allocation sketch" outlined before. This theorem forms the main technical ingredient for our robustifying reduction in Section 4. However, we expect Theorem 8 to have broader applicability, see for instance Appendix B for an alternative application in a distributed setup.

▶ **Theorem 8** (Budget Allocation Sketch). *There is a randomized construction of $S \in \mathbb{R}^{s \times n}$ with $s = O(\log^2 n)$, such that given $k \geq 0$ and SX for an unknown $X \in \mathbb{R}^{n \times d}$ with $d \leq n$ we can find in time $O(d \log^4 n)$ for each column j a number $t_j \geq 0$ such that with high probability:*

1. $\sum_{j \in [d]} t_j = O(k)$,
2. $\frac{\|(Xe_j)_{-t_j}\|_2^2}{t_j+1} \leq \frac{\|X_{-k}\|_F^2}{k+1}$ *for each* $j \in [d]$.

Furthermore, the matrix S can be sampled in time $O(\log^2 n)$, after which any entry can be computed in time $O(1)$, and it can be stored in $O(\log^2 n)$ space.

The construction of S and the proof of properties in the last sentence are simple, see Sections 3.1 and 3.2. For the main property, we first show that from SX we can accurately and efficiently approximate the total noise $\|X_{-k}\|_F$ in Section 3.3. Then in Section 3.4 we show how to allocate column budgets, thus finishing the proof of Theorem 8.

3.1 Construction of the Sketch

We write $T_n = \{0\} \cup \{2^\ell : 0 \leq \ell \leq \lfloor \log n \rfloor\} \cup \{n\}$. This set has the property that for all $0 \leq x \leq n$ (including $x = 0$), there is a number $t \in T_n$ such that $x \leq t \leq \min\{2x, n\}$.

For each $t \in T_n$ we apply Lemma 6 (with error parameter $\delta = n^{-100}$) to construct the matrix $S^{(t)}$. We construct the matrix S by stacking all matrices $S^{(t)}$, $t \in T_n$, on top of each other.

3.2 Easy Properties

Most properties of Theorem 8 can be easily verified: By Lemma 6 each matrix $S^{(t)}$ has $O(\log 1/\delta) = O(\log n)$ rows and can be sampled in time and space $O(\log n)$, so S has $O(\log^2 n)$ rows and can be sampled in time and space $O(\log^2 n)$. By Lemma 6 each entry of $S^{(t)}$ can be computed in time $O(1)$, so the same holds for S. It remains to construct the budgets $t_1, \ldots, t_d$.

3.3 Approximating the ℓ_2-Tail

We first show that from the sketch SX we can approximate the ℓ_2-mass of all columns of X after zeroing out the t largest entries, for any $t \in T_n$:

▶ **Lemma 9** (Column-Wise ℓ_2-Tail Estimation). *Let $X \in \mathbb{R}^{n\times d}$ with $d \le n$. Given SX in time $O(d\log^2 n)$ we can compute numbers $v_{j,t}$, for all $j \in [d]$ and $t \in T_n$, that satisfy with high probability in n:*

$$\frac{1}{64}\|(Xe_j)_{-512t}\|_2^2 \le v_{j,t} \le \|(Xe_j)_{-t}\|_2^2.$$

Proof. Fix $j \in [d]$ and $t \in T_n$. By construction of S, from SX we can read off $S^{(t)}X$. The j-column of this matrix is $S^{(t)}(Xe_j)$. Applying Lemma 6 yields a number $v = v_{j,t}$ with the claimed property

$$\frac{1}{64}\|(Xe_j)_{-512t}\|_2^2 \le v_{j,t} \le \|(Xe_j)_{-t}\|_2^2.$$

Since one application of Lemma 6 takes time $O(\log 1/\delta) = O(\log n)$ and we apply it $d\log n$ times (once for each $j \in [d]$ and $t \in T_n$), the total time is $O(d\log^2 n)$. ◀

Next, we combine these approximations to estimate $\|X_{-k}\|_F$:

▶ **Lemma 10** (Frobenius Norm Tail Estimation). *Let $X \in \mathbb{R}^{n\times d}$. Given SX and a number $k \ge 0$ in time $O(d\log^4 n)$ we can compute a number w satisfying with high probability in n:*

$$\frac{1}{128}\|X_{-1024k}\|_F^2 \le w \le \|X_{-k}\|_F^2.$$

Proof. We first compute the column-wise approximations $v_{j,t}$ for all $j \in [d]$ and $t \in T_n$ by Lemma 9. The idea is that we can read off w as the optimal value of the following Minimum-Cost Multiple-Choice Knapsack instance: View each pair $(j,t) \in [d] \times T_n$ as an *item* with *cost* $v_{j,t}$ and *weight* t. We group these items into d *bundles* of the form $\{(j,t) : t \in T_n\}$. The goal is to select from each bundle $j \in [d]$ exactly *one* item (j,t_j) such that the total weight of all items is at most $W = 2k$, and such that the total cost is minimized. Formally,

$$\begin{aligned} &\text{minimize} && \sum_{j\in[d]} v_{j,t_j} \\ &\text{subject to} && \sum_{j\in[d]} t_j \le 2k. \end{aligned}$$

Let OPT denote the optimal value of this optimization problem. As we will describe in detail in Appendix A, we can efficiently compute an approximation w satisfying $\frac{1}{2}\,\mathrm{OPT} \le w \le \mathrm{OPT}$ (specifically, we apply Theorem 13 in the upcoming Appendix A with parameter $\epsilon = \frac{1}{2}$).

It remains to prove that this value w is as desired. To this end, the following claim relates OPT to the ℓ_2-mass of the light entries in X:

▷ Claim 11. $\frac{1}{64}\|X_{-1024k}\|_2^2 \le \mathrm{OPT} \le \|X_{-k}\|_F^2$.

Proof. We start with the upper bound on OPT – that is, we construct a feasible solution to the optimization problem with value at most $\|X_{-k}\|_F^2$. Let us call the k largest entries in X (in absolute value) the *heavy* entries. Let k_j denote the number of heavy entries in the j-th column. We pick t_j to be the smallest value in T_n that exceeds k_j; by the construction of T_n we have $k_j \le t_j \le 2k_j$. The constraint $\sum_{j\in[d]} t_j \le \sum_{j\in[d]} 2k_j \le 2k$ is clearly satisfied.

Therefore, we have that

$$\begin{aligned}\mathrm{OPT} &\leq \sum_{j\in[d]} v_{j,t_j} \\ &\leq \sum_{j\in[d]} \|(Xe_j)_{-t_j}\|_2^2 \\ &\leq \sum_{j\in[d]} \|(Xe_j)_{-k_j}\|_2^2 \\ &= \|X_{-k}\|_F^2,\end{aligned}$$

where in the second step we have applied Lemma 9.

Next we show the lower bound on OPT. To this end, let $(t_1^*, \ldots, t_d^*)$ denote an optimal solution to the optimization problem. By Lemma 9 we have that

$$\begin{aligned}\mathrm{OPT} &= \sum_{j\in[d]} v_{j,t_j^*} \\ &\geq \tfrac{1}{64} \sum_{j\in[d]} \|(Xe_j)_{-512t_j^*}\|_2^2.\end{aligned}$$

This sum can be seen as the Frobenius norm of the matrix X after zeroing out some $512t_j^*$ entries from each column j. We zero out $\sum_{j\in[d]} 512t_j^* \leq 1024k$ entries in total, using that $(t_1^*, \ldots, t_d^*)$ is a feasible solution to the optimization problem. Therefore, $\mathrm{OPT} \geq \frac{1}{64}\|X_{-1024}\|_F^2$ as claimed. ◁

We finally analyze the running time. The call to Lemma 9 takes time $O(d\log^2 n)$. By Theorem 13 it takes time $O(d\log^4 n)$ to approximate the Minimum-Cost Multiple-Choice Knapsack problem on d bundles and $d \cdot |T_n| = O(d\log n) \leq O(n^2)$ items for $\epsilon = 1/2$. ◀

3.4 Allocating Column Budgets

Finally, we assign *budgets* $t_1, \ldots, t_d$ to the columns with the following two objectives in mind: On the one hand the budgets must be large enough such that the ℓ_2-tails per column, $\|(Xe_j)_{-t_j}\|_2$, are sufficiently small. On the other hand, for efficiency reasons the total budget should be bounded by $O(k)$. Compared to the overview, we remark that our formal statements here often involve quantities $\frac{1}{t+1}$ rather than $\frac{1}{t}$; this is necessary to take care of zero-budget columns.

▶ **Lemma 12** (Column Budgets). *Let $X \in \mathbb{R}^{n\times d}$ with $d \leq n$. Given SX and a number $k \geq 0$ in time $O(d\log^4 n)$ we can compute budgets $0 \leq t_1, \ldots, t_n \leq n$ that satisfy with high probability in n:*

$$\frac{\|(Xe_j)_{-t_j}\|_2^2}{t_j+1} \leq \frac{\|X_{-k}\|_F^2}{k+1}, \tag{4}$$

for all $j \in [d]$, and

$$\sum_{j\in[d]} t_j \leq 2^{25}(k+1) = O(k). \tag{5}$$

Proof. Compute the approximations $v_{j,t}$ (for $j \in [d]$ and $t \in T_n$) and w by Lemmas 9 and 10. For each $j \in [d]$, let $t_j' \in T_n$ denote the smallest value satisfying that

$$\frac{64v_{j,t_j'}}{t_j'+1} \leq \frac{w}{k+1};$$

note that $v_{j,n} = 0$ and thus there certainly is a value $t'_j \in T_n$ satisfying this inequality. Then we pick $t_j = \min\{512t'_j, n\}$. In what follows we argue that properties (4) and (5) are satisfied.

Property (4) follows from a simple calculation, using that $v_{j,\ell}$ and w are accurate approximations according to Lemmas 9 and 10:

$$\begin{aligned}\frac{\|(Xe_j)_{-t_j}\|_2^2}{t_j+1} &\leq \frac{\|(Xe_j)_{-512t'_j}\|_2^2}{t'_j+1}\\ &\leq \frac{64v_{j,t'_j}}{t'_j+1}\\ &\leq \frac{w}{k+1}\\ &\leq \frac{\|X_{-k}\|_F^2}{k+1}.\end{aligned}$$

For Property (5) we define some auxiliary quantities. Let $J \subseteq [d]$ denote the set of indices j satisfying that $t_j \neq 0$. Note that in the sum $\sum_{j\in[d]} t_j$ we can ignore all terms with $j \notin J$. Let $k' = 1024k$; we call the k' largest entries in X the *k'-heavy* entries. Let k'_j denote the number of k'-heavy entries in the j-th column of X. Then, for each $j \in J$, let $s_j \in T_n$ be the smallest value in T_n which is at least

$$\max\left\{k'_j,\, 2^{13}(k+1)\cdot\frac{\|(Xe_j)_{-k'_j}\|_2^2}{\|X_{-k'}\|_F^2}\right\}.$$

We emphasize that it is not our intention to compute s_j (as we have no way of learning k'_j); we merely define s_j for the analysis. We observe that $s_j > 0$ for all $j \in J$, as otherwise we would have $k'_j = 0$ and $\|Xe_j\|_2 = \|(Xe_j)_{-k'_j}\|_2 = 0$, and we would thus have selected $t_j = 0$. Therefore, and by the definition of T_n, it follows that

$$\max\left\{k'_j,\, 2^{13}(k+1)\cdot\frac{\|(Xe_j)_{-k'_j}\|_2^2}{\|X_{-k'}\|_F^2}\right\} \leq s_j \leq 2\cdot\max\left\{k'_j,\, 2^{13}(k+1)\cdot\frac{\|(Xe_j)_{-k'_j}\|_2^2}{\|X_{-k'}\|_F^2}\right\}.$$

Next, we argue that $t'_j \leq s_j$. Consider that

$$\begin{aligned}\frac{64v_{j,s_j}}{s_j+1} &\leq \frac{64\cdot\|(Xe_j)_{-s_j}\|_2^2}{s_j}\\ &\leq \frac{64\cdot\|(Xe_j)_{-s_j}\|_2^2\cdot\|X_{-k'}\|_F^2}{2^{13}(k+1)\cdot\|(Xe_j)_{-k'_j}\|_2^2}\\ &\leq \frac{\|X_{-k'}\|_F^2}{128(k+1)}\\ &\leq \frac{w}{k+1},\end{aligned}$$

where have again applied Lemmas 9 and 10. Consequently, from the minimality of the values t'_j it indeed follows that $t'_j \leq s_j$. Finally, we have that

$$\begin{aligned}
\sum_{j\in[d]} t_j &= \sum_{j\in J} t_j \\
&\le 512 \cdot \sum_{j\in J} t'_j \\
&\le 512 \cdot \sum_{j\in J} s_j \\
&\le 1024 \cdot \sum_{j\in J} \max\left\{ k'_j, 2^{13}(k+1) \cdot \frac{\|(Xe_j)_{-k'_j}\|_2^2}{\|X_{-k'}\|_F^2} \right\} \\
&\le 1024 \cdot \sum_{j\in J} \left(k'_j + 2^{13}(k+1) \cdot \frac{\|(Xe_j)_{-k'_j}\|_2^2}{\|X_{-k'}\|_F^2} \right) \\
&\le 1024 \cdot (2k' + 2^{14}(k+1)) \\
&\le 1024 \cdot (2^{11}k + 2^{14}(k+1)) \\
&\le 2^{25}(k+1),
\end{aligned}$$

proving Property (5).

Finally, consider the running time. The dominant contribution is the calls to Lemmas 9 and 10 in time $O(d \log^4 n)$. Then, having access to $v_{j,t}$ and w, we can easily compute the budgets $t_1, \ldots, t_n$ in time $O(d \log n)$. ◀

This completes the proof of Theorem 8, by combining Section 3.2 and Lemma 12. Due to space constraints we moved the proof of our Main Theorem 2 to Section 4.

4 Robustifying Reduction

The goal of this section is to prove our main theorem:

▶ **Theorem 2** (Robustifying Reduction). *Suppose that sparse matrix multiplication of $n \times n$ matrices (with m_{in} nonzero inputs and m_{out} nonzero outputs) is in time $T(n, m_{in}, m_{out})$. Then robust sparse matrix multiplication is in time $O(n \log^4 n + \log^3 n \cdot T(n, m_{in}, k)) = \widetilde{O}(T(n, m_{in}, k))$ (by a randomized algorithm that succeeds with high probability).*

Proof. The first step is to compute the column budgets $s_1, \ldots, s_n$ by Theorem 8. Specifically, we apply Theorem 8 for the matrix $X = AB$, and compute the product SX by first multiplying S with A, and multiplying the resulting $O(\log^2 n) \times n$ matrix with B. Let $t_1, \ldots, t_n$ denote the resulting column budgets satisfying the two properties

1. $\sum_{j\in[d]} t_j = O(k)$,
2. $\frac{\|(Xe_j)_{-t_j}\|_2^2}{t_j+1} \le \frac{\|X_{-k}\|_F^2}{k+1}$ for each $j \in [d]$.

We may round up each t_j to the smallest value in T_n while maintaining the two properties.

According to these budgets we partition the columns $[n]$ into blocks $J_s = \{j : t_j = s\}$ for $s \in T_n$. Let $B^{(s)}$ denote the matrix B restricted to the columns in J_s. In this terminology, our remaining goal is to compute the individual matrix products $AB^{(s)}$ for each $s \in T_n$.

Fix any $s \in T_n$ and let $H^{(s)}$ be the matrix obtained from Lemma 7 (with $\delta = n^{-100}$). We compute the matrix product $A^{(s)} := H^{(s)}A$ in time $O(m_{in} \log^2 n)$ exploiting that $H^{(s)}$ has column sparsity $O(\log^2 n)$ by Lemma 7. Moreover, $A^{(s)}$ has size $O(s \log^2 n) \times n$ and

is $O(m_{in} \log^2 n)$-sparse. Therefore, for some constant $\gamma > 0$ to be determined later, we can partition $A^{(s)}$ row-wise into $R = O(\log^2 n \cdot \gamma^{-1})$ submatrices $A_1^{(s)}, \dots, A_R^{(s)}$ each of which has size $\gamma s \times n$ and is m_{in}-sparse. Indeed, first split $A^{(s)}$ into chunks of γs rows, and whenever necessary subdivide a chunk into smaller subchunks with sparsity at most m_{in}; it is easy to see that the total number of chunks and subchunks does not exceed $O(\log^2 n \cdot \gamma^{-1})$. We then compute the matrix products $A_1^{(s)} B^{(s)}, \dots, A_R^{(s)} B^{(s)}$ using oracle calls to the efficient sparse matrix multiplication algorithm. Combining the rows of these resulting matrices appropriately, we obtain $A^{(s)} B^{(s)}$. Finally, we pass the column vectors of $A^{(s)} B^{(s)} = H^{(s)}(AB^{(s)})$ to Lemma 7 and obtain, for each $j \in J_s$, a vector c_j such that

$$\|Ab_j - c_j\|_\infty \leq \frac{1}{\sqrt{s+1}} \|(Ab_j)_{-s}\|_2.$$

After all $O(\log n)$ iterations, let C denote the $n \times n$ matrix with column vectors $c_1, \dots, c_n$. We return this matrix C.

The correctness follows from a simple calculation:

$$\begin{aligned} \|AB - C\|_\infty &= \max_{j \in [n]} \|Ab_j - c_j\|_\infty \\ &\leq \max_{j \in [n]} \frac{1}{\sqrt{t_j + 1}} \|(Ab_j)_{-t_j}\|_2 \\ &\leq \frac{1}{\sqrt{k+1}} \|(AB)_{-k}\|_F. \end{aligned}$$

In the last step we have applied Property (4) from Lemma 12.

We finally analyze the running time. The construction of the sketch matrix S takes polylogarithmic time, and the multiplications SA and $(SA)B$ take time $O(m_{in} \log^2 n)$ (since S has $O(\log^2 n)$ rows). Computing the budgets takes time $O(n \log^4 n)$ by Theorem 8. For any fixed $s \in T_n$ the time to compute the matrix $H^{(s)}$ is negligible. Computing the matrix product $A^{(s)} = H^{(s)} A$ by the naive algorithm takes time $O(m_{in} \log^2 n)$ given that $H^{(s)}$ is $O(\log^2 n)$-column sparse. Splitting $A^{(s)}$ into its submatrices runs in the same time $O(m_{in} \log^2 n)$. To bound the time to compute any matrix product $A_r^{(s)} B^{(s)}$, observe that this is a matrix of size $s \times |J_s|$. Since $\sum_{j \in [n]} t_j \leq 8c_2(c_1^2 + c_2)k$ by Property (5) of Lemma 12, we must have $|J_s| \leq 8c_2(c_1^2 + c_2)k/s$. In particular, the matrix $A_r^{(s)} B^{(s)}$ has at most $\gamma s \cdot 8c_2(c_1^2 + c_2)k/s$ nonzero entries, and thus, by choosing the constant $\gamma = 1/(8c_2(c_1^2 + c_2))$, the number of entries becomes $\leq k$. Since both matrices $A_r^{(s)}$ and $B^{(s)}$ further have size at most $n \times n$ and at most m_{in} nonzero entries, the running time of each product is $O(T(n, m_{in}, k))$. Recall that we globally iterate over $O(\log^2 n)$ choices for r and $|T_n| = O(\log n)$ choices for s, so the total time is

$$O(n \log^4 n + m_{in} \log^3 n + \log^3 n \cdot T(n, m_{in}, k)) = O(n \log^4 n + \log^3 n \cdot T(n, m_{in}, k)),$$

as claimed. ◀

4.1 Consequences of the Reduction

We finally show how the three corollaries follow from our main reduction. For Corollaries 3 and 5 we plug in the Abboud–Bringmann–Fischer–Künnemann algorithm [1] into Theorem 2 (with a different upper bounds on the running time), whereas for Corollary 4 it suffices to use rectangular matrix multiplication. In fact, even for Corollaries 3 and 5 it is not necessary to use the full power of [1], and we could instead directly apply the heavy/light algorithm á la Yuster–Zwick [68], generalized by Kaplan–Sharir–Verbin [39].

Proof of Corollary 3. [1, Theorem 1.4] proves that $T(n, m_{in}, m_{out}) = O((m_{in} + m_{out})^{1.346})$. From this, the claim is immediate by Theorem 2. ◀

Proof of Corollary 4. Let $\omega(\cdot,\cdot,\cdot)$ denote the exponent of algebraic rectangular matrix multiplication (i.e., such that $\mathrm{MM}(n^a, n^b, n^c) = O(n^{\omega(a,b,c)+\epsilon})$ for any $\epsilon > 0$). It is well-known that $\omega(\cdot,\cdot,\cdot)$ is a convex function [50, 8].

The running time of the Abboud–Bringmann–Fischer–Künnemann algorithm [1] can be bounded by

$$\widetilde{O}\left(m_{in} + \max_{\substack{x,z\le n\\ xz\le m_{out}}} \mathrm{MM}(x,n,z)\right).$$

Let γ be the constant such that $m_{out} = n^{\gamma+o(1)}$. Then we can bound this running time in terms of $\omega(\cdot,\cdot,\cdot)$ by

$$\widetilde{O}\left(m_{in} + \max_{0\le a,c\le 1} n^{\omega(a,1,c)+\epsilon}\right),$$

for any $\epsilon > 0$. From the convexity of $\omega(\cdot,\cdot,\cdot)$ it follows that the maximum is obtained in one of the corner cases $\omega(1,1,\gamma-1)$ or $\omega(\gamma-1,1,1)$. It is further known that $\omega(\cdot,\cdot,\cdot)$ is invariant under permuting its three arguments [50, 8], and thus the running time is $\widetilde{O}(m_{in} + n^{\omega(1,1,\gamma-1)+\epsilon}) = \widetilde{O}(m_{in} + \mathrm{MM}(n,n,\lceil \frac{k}{n} \rceil) \cdot n^{\epsilon})$.

Given this bound for sparse matrix multiplication, the statement follows immediately from Theorem 2. ◀

Proof of Corollary 5. A simple combination of [1, Theorem 1.7] with [1, Lemma 4.8] and the current state-of-the-art bounds for the dual matrix multiplication constant $\alpha \ge 0.31389$ yields that $T(n, m_{in}, m_{out}) = O(m_{out}^{1+\epsilon})$, for any $\epsilon > 0$, provided that $m_{out} \ge m_{in}^{1.76110} \ge (m_{in})^{1+\frac{1}{1+\alpha}}$. The statement now follows immediately from our main reduction in Theorem 2. ◀

References

1 Amir Abboud, Karl Bringmann, Nick Fischer, and Marvin Künnemann. The time complexity of fully sparse matrix multiplication. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 4670–4703. SIAM, 2024. doi:10.1137/1.9781611977912.167.

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

3 Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In Ronald Fagin, editor, *Database Theory - ICDT 2009, 12th International Conference, St. Petersburg, Russia, March 23-25, 2009, Proceedings*, volume 361 of *ACM International Conference Proceeding Series*, pages 121–126. ACM, 2009. doi:10.1145/1514894.1514909.

4 Grey Ballard, James Demmel, Olga Holtz, Benjamin Lipshitz, and Oded Schwartz. Communication-optimal parallel algorithm for Strassen's matrix multiplication. In Guy E. Blelloch and Maurice Herlihy, editors, *24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12, Pittsburgh, PA, USA, June 25-27, 2012*, pages 193–204. ACM, 2012. doi:10.1145/2312005.2312044.

5 Huck Bennett, Karthik Gajulapalli, Alexander Golovnev, and Evelyn Warton. Matrix multiplication verification using coding theory. In Amit Kumar and Noga Ron-Zewi, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2024, London School of Economics, London, UK, August 28-30, 2024*, volume 317 of *LIPIcs*, pages 42:1–42:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPIcs.APPROX/RANDOM.2024.42.

6 Huck Bennett, Karthik Gajulapalli, Alexander Golovnev, and Evelyn Warton. Output-sparse matrix multiplication using compressed sensing. *CoRR*, abs/2508.10250, 2025. doi:10.48550/arXiv.2508.10250.

7 Markus Bläser. A $5/2n^2$-lower bound for the rank of $n \times n$ matrix multiplication over arbitrary fields. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999, New York, NY, USA, October 17-18, 1999*, pages 45–50. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814576.

8 Markus Bläser. Fast matrix multiplication. *Theory Comput.*, 5:1–60, 2013. doi:10.4086/TOC.GS.2013.005.

9 Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Inf. Theory*, 52(2):489–509, 2006. doi:10.1109/TIT.2005.862083.

10 Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics*, 59(8):1207–1223, 2006. doi:10.1002/cpa.20124.

11 Timothy M. Chan. Approximation schemes for 0-1 knapsack. In Raimund Seidel, editor, *1st Symposium on Simplicity in Algorithms, SOSA 2018, New Orleans, LA, USA, January 7-10, 2018*, volume 61 of *OASIcs*, pages 5:1–5:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/OASIcs.SOSA.2018.5.

12 Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. A nearly quadratic-time FPTAS for knapsack. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 283–294. ACM, 2024. doi:10.1145/3618260.3649730.

13 Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 205–214. ACM, 2009. doi:10.1145/1536414.1536445.

14 Edith Cohen and David D. Lewis. Approximating matrix multiplication for pattern recognition tasks. *J. Algorithms*, 30(2):211–252, 1999. doi:10.1006/JAGM.1998.0989.

15 Don Coppersmith. Rapid multiplication of rectangular matrices. *SIAM J. Comput.*, 11(3):467–471, 1982. doi:10.1137/0211037.

16 Don Coppersmith. Rectangular matrix multiplication revisited. *J. Complex.*, 13(1):42–49, 1997. doi:10.1006/JCOM.1997.0438.

17 Humberto Madrid de La Vega, Valia Guerra, and Marielba Rojas. Sampling techniques for monte carlo matrix multiplication with applications to image processing. In Jesús Ariel Carrasco-Ochoa, José Francisco Martínez Trinidad, José Arturo Olvera-López, and Kim L. Boyer, editors, *Pattern Recognition - 4th Mexican Conference, MCPR 2012, Huatulco, Mexico, June 27-30, 2012. Proceedings*, volume 7329 of *Lecture Notes in Computer Science*, pages 45–54. Springer, 2012. doi:10.1007/978-3-642-31149-9_5.

18 Shaleen Deep, Xiao Hu, and Paraschos Koutris. Fast join project query evaluation using matrix multiplication. In David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1213–1223. ACM, 2020. doi:10.1145/3318464.3380607.

19 Mingyang Deng, Ce Jin, and Xiao Mao. Approximating knapsack and partition via dense subset sums. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2961–2979. SIAM, 2023. doi:10.1137/1.9781611977554.CH113.

20 Petros Drineas and Ravi Kannan. Fast monte-carlo algorithms for approximate matrix multiplication. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, Las Vegas, Nevada, USA, October 14-17, 2001*, pages 452–459. IEEE Computer Society, 2001. doi:10.1109/SFCS.2001.959921.

21 Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast monte carlo algorithms for matrices I: approximating matrix multiplication. *SIAM J. Comput.*, 36(1):132–157, 2006. doi:10.1137/S0097539704442684.

22 Sylvester David Eriksson-Bique, Mary Solbrig, Michael Stefanelli, Sarah Warkentin, Ralph Abbey, and Ilse C. F. Ipsen. Importance sampling for a monte carlo matrix multiplication algorithm, with application to information retrieval. *SIAM J. Sci. Comput.*, 33(4):1689–1706, 2011. doi:10.1137/10080659X.

23 Nick Fischer and Vasileios Nakos. ℓ_2/ℓ_2 sparse recovery via weighted hypergraph peeling. In *66th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2025, Sydney, Australia, December 14-17, 2025*, pages 1442–1457. IEEE, 2025. doi:10.1109/FOCS63196.2025.00075.

24 François Le Gall. Powers of tensors and fast matrix multiplication. In Katsusuke Nabeshima, Kosaku Nagasaka, Franz Winkler, and Ágnes Szántó, editors, *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, pages 296–303. ACM, 2014. doi:10.1145/2608628.2608664.

25 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the Coppersmith-Winograd tensor. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1029–1046. SIAM, 2018. doi:10.1137/1.9781611975031.67.

26 Jianhua Gao, Weixing Ji, Fangli Chang, Shiyu Han, Bingxin Wei, Zeming Liu, and Yizhuo Wang. A systematic survey of general sparse matrix-matrix multiplication. *ACM Comput. Surv.*, 55(12):244:1–244:36, 2023. doi:10.1145/3571157.

27 Leszek Gasieniec, Christos Levcopoulos, Andrzej Lingas, Rasmus Pagh, and Takeshi Tokuyama. Efficiently correcting matrix products. *Algorithmica*, 79(2):428–443, 2017. doi:10.1007/S00453-016-0202-3.

28 Anna C. Gilbert, Yi Li, Ely Porat, and Martin J. Strauss. Approximate sparse recovery: Optimizing time and measurements. *SIAM J. Comput.*, 41(2):436–453, 2012. doi:10.1137/100816705.

29 Dirk Van Gucht, Ryan Williams, David P. Woodruff, and Qin Zhang. The communication complexity of distributed set-joins with applications to matrix multiplication. In Tova Milo and Diego Calvanese, editors, *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 199–212. ACM, 2015. doi:10.1145/2745754.2745779.

30 Fred G. Gustavson. Two fast algorithms for sparse matrices: Multiplication and permuted transposition. *ACM Trans. Math. Softw.*, 4(3):250–269, 1978. doi:10.1145/355791.355796.

31 Haitham Hassanieh, Piotr Indyk, Dina Katabi, and Eric Price. Nearly optimal sparse Fourier transform. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 563–578. ACM, 2012. doi:10.1145/2213977.2214029.

32 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975. doi:10.1145/321906.321909.

33 Piotr Indyk, Michael Kapralov, and Eric Price. (Nearly) sample-optimal sparse Fourier transform. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 480–499. SIAM, 2014. doi:10.1137/1.9781611973402.36.

34 Piotr Indyk, Eric Price, and David P. Woodruff. On the power of adaptivity in sparse recovery. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 285–294. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.83.

35 Mark A. Iwen and Craig V. Spencer. A note on compressed sensing and the complexity of matrix multiplication. *Inf. Process. Lett.*, 109(10):468–471, 2009. doi:10.1016/J.IPL.2009.01.010.

36 Riko Jacob and Morten Stöckel. Fast output-sensitive matrix multiplication. In Nikhil Bansal and Irene Finocchi, editors, *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 766–778. Springer, 2015. doi:10.1007/978-3-662-48350-3_64.

37 Ce Jin. An improved FPTAS for 0-1 knapsack. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, Patras, Greece, July 9-12, 2019*, volume 132 of *LIPIcs*, pages 76:1–76:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.ICALP.2019.76.

38 Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. Fast moment estimation in data streams in optimal space. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 745–754. ACM, 2011. doi:10.1145/1993636.1993735.

39 Haim Kaplan, Micha Sharir, and Elad Verbin. Colored intersection searching via sparse rectangular matrix multiplication. In Nina Amenta and Otfried Cheong, editors, *Proceedings of the 22nd ACM Symposium on Computational Geometry, Sedona, Arizona, USA, June 5-7, 2006*, pages 52–60. ACM, 2006. doi:10.1145/1137856.1137866.

40 Michael Kapralov. Sparse Fourier transform in any constant dimension with nearly-optimal sample complexity in sublinear time. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 264–277. ACM, 2016. doi:10.1145/2897518.2897650.

41 Michael Kapralov, Ameya Velingker, and Amir Zandieh. Dimension-independent sparse Fourier transform. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2709–2728. SIAM, 2019. doi:10.1137/1.9781611975482.168.

42 Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an FPTAS for the knapsack problem. *J. Comb. Optim.*, 8(1):5–11, 2004. doi:10.1023/B:JOCO.0000021934.29833.6B.

43 Marvin Künnemann. On nondeterministic derandomization of Freivalds' algorithm: Consequences, avenues and algorithmic progress. In Yossi Azar, Hannah Bast, and Grzegorz Herman, editors, *26th Annual European Symposium on Algorithms, ESA 2018, Helsinki, Finland, August 20-22, 2018*, volume 112 of *LIPIcs*, pages 56:1–56:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ESA.2018.56.

44 Konstantin Kutzkov. Deterministic algorithms for skewed matrix products. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013, Kiel, Germany, February 27 - March 2, 2013*, volume 20 of *LIPIcs*, pages 466–477. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.STACS.2013.466.

45 J. M. Landsberg. New lower bounds for the rank of matrix multiplication. *SIAM J. Comput.*, 43(1):144–149, 2014. doi:10.1137/120880276.

46 Kasper Green Larsen, Jelani Nelson, Huy L. Nguyen, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, Hyatt Regency, New Brunswick, New Jersey, USA, October 9-11, 2016*, pages 61–70. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.16.

47 Kasper Green Larsen, Jelani Nelson, Huy L. Nguyen, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. *Commun. ACM*, 62(8):95–100, 2019. doi:10.1145/3339185.

48 Eugene L. Lawler. Fast approximation algorithms for knapsack problems. *Math. Oper. Res.*, 4(4):339–356, 1979. doi:10.1287/MOOR.4.4.339.

49 Andrzej Lingas. A fast output-sensitive algorithm for boolean matrix multiplication. *Algorithmica*, 61(1):36–50, 2011. doi:10.1007/S00453-010-9441-X.

50 Grazia Lotti and Francesco Romani. On the asymptotic complexity of rectangular matrix multiplication. *Theor. Comput. Sci.*, 23:171–185, 1983. doi:10.1016/0304-3975(83)90054-3.

51 Xiao Mao. $(1-\epsilon)$-Approximation of knapsack in nearly quadratic time. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 295–306. ACM, 2024. doi:10.1145/3618260.3649677.

52 Vasileios Nakos and Zhao Song. Stronger ℓ_2/ℓ_2 compressed sensing; without iterating. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 289–297. ACM, 2019. doi:10.1145/3313276.3316355.

53 Vasileios Nakos, Zhao Song, and Zhengyu Wang. (Nearly) sample-optimal sparse Fourier transform in any dimension; RIPless and filterless. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1568–1577. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00092.

54 Rasmus Pagh. Compressed matrix multiplication. *ACM Trans. Comput. Theory*, 5(3):9:1–9:17, 2013. doi:10.1145/2493252.2493254.

55 Eric Price and David P. Woodruff. (1 + eps)-Approximate sparse recovery. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 295–304. IEEE Computer Society, 2011. doi:10.1109/FOCS.2011.92.

56 Ran Raz. On the complexity of matrix product. *SIAM J. Comput.*, 32(5):1356–1369, 2003. doi:10.1137/S0097539702402147.

57 Donguk Rhee. Faster fully polynomial approximation schemes for knapsack problems. Master's thesis, MIT, 2015. URL: https://dspace.mit.edu/bitstream/handle/1721.1/98564/920857251-MIT.pdf.

58 Daniel S. Roche. Error correction in fast matrix multiplication and inverse. In Manuel Kauers, Alexey Ovchinnikov, and Éric Schost, editors, *Proceedings of the 2018 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2018, New York, NY, USA, July 16-19, 2018*, pages 343–350. ACM, 2018. doi:10.1145/3208976.3209001.

59 Tamás Sarlós. Improved approximation algorithms for large matrices via random projections. In *47th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2006, Berkeley, California, USA, October 21-24, 2006, Proceedings*, pages 143–152. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.37.

60 Amir Shpilka. Lower bounds for matrix product. *SIAM J. Comput.*, 32(5):1185–1200, 2003. doi:10.1137/S0097539702405954.

61 Morten Stöckel. *Randomized Primitives for Big Data Processing*. PhD thesis, IT-Universitetet i København, Denmark, 2015.

62 Yahel Uffenheimer and Omri Weinstein. (Approximate) matrix multiplication via convolutions. *CoRR*, abs/2510.22193, 2025. doi:10.48550/arXiv.2510.22193.

63 Yahel Uffenheimer and Omri Weinstein. Improved sparse recovery for approximate matrix multiplication. *CoRR*, abs/2602.04386, 2026. doi:10.48550/arXiv.2602.04386.

64 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In David P. Woodruff, editor, *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*, pages 3792–3835. SIAM, 2024. doi:10.1137/1.9781611977912.134.

65 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1867–1877. SIAM, 2014. doi:10.1137/1.9781611973402.135.

66 Yue Wu. A note on random sampling for matrix multiplication. *CoRR*, abs/1811.11237, 2018. arXiv:1811.11237.

67 Chuhan Yang and Christopher Musco. Efficient block approximate matrix multiplication. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman, editors, *31st Annual European Symposium on Algorithms, ESA 2023, Amsterdam, The Netherlands, September 4-6, 2023*, volume 274 of *LIPIcs*, pages 103:1–103:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ESA.2023.103.

68 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. doi:10.1145/1077464.1077466.

A Minimum-Cost Multiple-Choice Knapsack

In this section we design an efficient approximation scheme for the *Minimum-Cost Multiple-Choice Knapsack* problem; see the following Theorem 13. For this problem the input consists of n items with *costs* $c_1, \dots, c_n \geq 0$ and *weights* $w_1, \dots, w_n \geq 0$ along with a partitioning of $[n]$ into *bundles* $S_1, \dots, S_m$ and a weight threshold $W \geq 0$. The goal is to select a subset of items $I \subseteq [n]$, exactly one from each bundle, such that the total weight is at most $\sum_{i \in I} w_i \leq W$ and such that the total cost $\sum_{i \in I} c_i$ is minimized. Equivalently expressed as an integer program with indicator variables $x_1, \dots, x_n \in \{0, 1\}$, the task is to:

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in [n]} c_i x_i, \\ \text{subject to} \quad & \sum_{i \in [n]} w_i x_i \leq W, \\ & \sum_{i \in S_j} x_i = 1 \qquad \forall j \in [m]. \end{aligned}$$

▶ **Theorem 13.** *The Minimum-Cost Multiple-Choice Knapsack problem can be* $(1+\epsilon)$*-approximated in time* $O(n \log n + m\epsilon^{-2} \log^4(n/\epsilon)) \leq O(n\epsilon^{-2} \log^4(n/\epsilon))$.

To prove this theorem we borrow heavily from Chan's framework for 0-1 Knapsack [11]. While the dependence on ϵ is a secondary concern to us, we have to spend a little extra effort in order to cope with the Minimum-Cost variant of the problem. This extra difficulty is mainly due to the fact that we cannot trivially assume that all input numbers are from a $\text{poly}(n, \epsilon^{-1})$-size range. Instead, we employ the following lemma to precompute a very crude approximation:

▶ **Lemma 14.** *The Minimum-Cost Multiple-Choice Knapsack problem can be* n*-approximated in time* $O(n \log n)$.

Proof. In order to compute an n-approximation, it suffices to consider the relaxed problem with the objective to minimize $\max_i c_i x_i$ rather than $\sum_i c_i x_i$. Then, denoting the optimal value of this alternative problem by OPT', we have that $\text{OPT}' \leq \text{OPT} \leq n \cdot \text{OPT}'$. To compute OPT', let us sort and reorder the items such that $c_1 \geq \cdots \geq c_n$. Then we successively *delete* the items $1, \dots, n$. In each step we test whether the remaining instance is still feasible (i.e., in the k-th step we test whether there is a subset of $\{k, \dots, n\}$ that satisfies the weight and multiple-choice constraints). When k^* is the last feasible iteration it is not hard to see that $\text{OPT}' = c_{k^*}$.

To implement this strategy efficiently we precompute for each bundle S_j its right-to-left-minima, more precisely we compute the item of minimum weight in $S_j \cap \{k, \dots, n\}$ for each k. Note that these are at most $|S_j|$ distinct items, and we can compute them in sorted order by k via a linear-time scan over S_j. Following the strategy of iteratively deleting the items $1, \dots, n$, we maintain the smallest possible weight of all solutions:

$$w = \sum_{j \in [m]} \min_{i \in S_j} w_i.$$

We can initially sort the items, precompute right-to-left-minima and compute w in time $O(n \log n)$. Then in each step, when we delete some item k, we update the minimum weight of any item in S_j, by advancing in its sequence of right-to-left-minima in case k is a right-to-left-minimum in S_j. We update w accordingly (by subtracting w_k and adding the new minimum weight in the bundle S_j if k happened to be a right-to-left-minimum in S_j). Note that at any point in time we can easily check whether the current instance is feasible by testing whether $w \leq W$. The total time is $O(n \log n)$ as claimed. ◀

By running Lemma 14 in a preprocessing step we can compute a number $t \leq \mathrm{OPT} \leq n \cdot t$. Afterwards, we can delete all items with cost more than $n \cdot t$ and replace the cost of any item with cost less than $\frac{\epsilon}{n} t$ by $\frac{\epsilon}{n} t$; the latter transformation increases the total cost of any solution additively by at most $\epsilon t \leq \epsilon \, \mathrm{OPT}$. Then, by rescaling all costs by $\frac{\epsilon}{n} t$ we can assume that all costs are in the range $\{0\} \cup [1, n^2/\epsilon]$. Hence, this preprocessing ensures that all input numbers are from a $\mathrm{poly}(n, \epsilon^{-1})$-size range.

Preliminaries on Monotone Functions. Before we continue with the actual algorithm, we take a step back and introduce some terminology as in [11]. Let $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ be a nonincreasing function. We say that f has *complexity* s if it takes s distinct values. In particular, since f is nonincreasing this implies that f is a *staircase function* with s steps. Note that any staircase function can be concisely stored in space $O(s)$ by storing for each of the at most s different function values v the point $\min\{x : f(x) = v\}$.

Now consider two nonincreasing functions $f, g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$. Their $(\min, +)$-*convolution* is the function $f \oplus g$ defined by

$$(f \oplus g)(z) = \min_{\substack{x, y \in \mathbb{R}_{\geq 0} \\ x + y = z}} (f(x) + g(y));$$

it can easily be verified that $f \oplus g$ is also nonincreasing. Moreover, if the complexities of f and g are at most s and t, respectively, then the complexity of $f \oplus g$ is at most st, and we can also compute $f \oplus g$ in time $O(st)$.

Finally, we say that a function $\widetilde{f}$ is an $(1+\epsilon)$-*approximation* of another function f if for all points $x \in \mathbb{R}_{\geq 0}$ we have $f(x) \leq \widetilde{f}(x) \leq (1+\epsilon) f(x)$. Let $\mathrm{stair}_\epsilon(f)$ be the function in which each nonzero non-∞ value $f(x)$ is rounded up to the smallest power of $1+\epsilon$. By definition, $\mathrm{stair}_\epsilon(f)$ is an $(1+\epsilon)$-approximation of f with complexity at most

$$O\left(\log_{1+\epsilon} \frac{\max_{<\infty}(f)}{\min_{>0}(f)}\right) = O\left(\epsilon^{-1} \log \frac{\max_{<\infty}(f)}{\min_{>0}(f)}\right),$$

where we write $\max_{<\infty}(f)$ for the largest non-∞ function value and $\min_{>0}(f)$ for the smallest positive function value of f.

A Divide-and-Conquer Algorithm. For a subset of bundles $J \subseteq [m]$ we define the function $f_J : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ as follows. Let $f(w)$ denote the minimum cost among all solutions with weight at most w that select exactly one item from each bundle in J (and no item from the other bundles). We set $f(w) = \infty$ if there is no such solution.

▶ **Observation 15.** *f_J is nonincreasing and* $\mathrm{OPT} = f_{[m]}(W)$.

In light of this observation, in order to solve the Minimum-Cost Multiple-Choice Knapsack problem our goal is to compute the function $f_{[m]}$. In fact, as we are only shooting for a $(1+\epsilon)$-approximate solution our aim is to compute a $(1+\epsilon)$-approximation $\widetilde{f}_{[m]}$ of $f_{[m]}$. The simple insights behind the algorithm are that:

▶ **Observation 16.** *For any partition $J = J_1 \sqcup J_2$ it holds that $f_J = f_{J_1} \oplus f_{J_2}$.*

▶ **Observation 17.** *Let $\widetilde{f}$, $\widetilde{g}$ be $(1+\epsilon)$-approximations of f, g, respectively. Then $\widetilde{f} \oplus \widetilde{g}$ is a $(1+\epsilon)$-approximation of $f \oplus g$.*

Proof of Theorem 13. Let $\delta > 0$ be a parameter. We design a divide-and-conquer algorithm to recursively compute good approximations $\widetilde{f}_J$ of the functions f_J. At the base level, for any $j \in [m]$, we compute the functions $f_{\{j\}}(w) = \min\{c_i : i \in S_j, w_i \leq w\}$ exactly and then pick $\widetilde{f}_{\{j\}} := \mathrm{stair}_\delta(f_{\{j\}})$. For the recursive step we partition $J = J_1 \sqcup J_2$ arbitrarily into halves of (roughly) equal size and compute $\widetilde{f}_{J_1}$ and $\widetilde{f}_{J_2}$ recursively. We then compute their $(\min,+)$-convolution $\widetilde{f}_{J_1} \oplus \widetilde{f}_{J_2}$ and choose $\widetilde{f}_J := \mathrm{stair}_\delta(\widetilde{f}_{J_1} \oplus \widetilde{f}_{J_2})$. In this way we compute an approximation $\widetilde{f}_{[m]}$ in a binary-tree fashion with recursion depth $O(\log m)$.

By induction it is easy to verify that $\widetilde{f}_J$ is an $(1+\delta)^{d+1}$-approximation of f_J when J is at the d-th level of the recursion (where $d = 0$ is the base level). This is clear at the base level. At any higher level d we recursively obtain $(1+\delta)^d$-approximations $\widetilde{f}_{J_1}$ and $\widetilde{f}_{J_2}$, and thus $\widetilde{f}_{J_1} \oplus \widetilde{f}_{J_2}$ is a $(1+\delta)^d$-approximation of $f_{J_1} \oplus f_{J_2}$ by Observation 17, which in turn equals f_J by Observation 16. The transformation by $\mathrm{stair}_\delta(\cdot)$ worsens the approximation by another factor $(1+\delta)$.

At the root of the recursion the error has accumulated to $(1+\delta)^{O(\log m)}$, and therefore by setting $\delta = \Theta(\epsilon/\log m)$ we can compute a $(1+\epsilon)$-approximation of $f_{[m]}$. By Observation 15 we can read off an $(1+\epsilon)$-approximation of the given Minimum-Cost Multiple-Choice Knapsack instance.

Regarding the running time, note that all functions f_J and $\widetilde{f}_J$ have range $\{0\} \cup [1, n^2/\epsilon] \cup \{\infty\}$ due to the preprocessing with Lemma 14, and thus their $\mathrm{stair}_\delta(\cdot)$ approximations have complexity bounded by $s = O(\delta^{-1}\log(n/\epsilon))$. At the j-th leaf we spend time $O(|S_j|\log|S_j|)$ to sort the weights in the j-th bundle and to compute $f_{\{j\}}$. Then we spend time $O(s)$ to compute $\widetilde{f}_{\{j\}}$. Any other node in the recursion tree takes time $O(s^2)$ to compute the $(\min,+)$-convolution $\widetilde{f}_{J_1} \oplus \widetilde{f}_{J_2}$ plus time $O(s)$ to compute $\widetilde{f}_J$. As the recursion tree has $O(m)$ nodes, the total running time is

$$O\left(\sum_{j\in[m]} |S_j|\log|S_j| + ms^2\right) = O(n\log n + m\epsilon^{-2}\log^2(m)\log^2(n/\epsilon)) = O(n\epsilon^{-2}\log^4(n/\epsilon)),$$

as claimed. ◀

B Distributed Heavy Hitters

In this section we present another simple application of our new budget allocation sketch (Theorem 8) to a distributed/streaming heavy hitters problem.

In the usual turnstile streaming setting, the goal is to maintain a vector $x \in \mathbb{R}^n$, which is initially all-zeroes, under a stream of updates of the form $x_i := x_i + \Delta$, for given i, Δ. The goal is to answer interesting queries about x (approximately and with high probability), while minimizing the space usage. In the multi-pass setting, multiple passes over the stream of updates are allowed, and we want to make as few passes as possible.

We consider the following distributed variant of the usual turnstile multi-pass streaming setting. We have d players and an arbitrary partitioning of the coordinates $[n] = I_1 \cup \ldots \cup I_d$. The stream is split over the players, namely player j receives exactly the updates affecting the coordinates in I_j. In addition, there is a coordinator, which can communicate with the players. Again the goal is to answer interesting queries about x, while minimizing the number of passes over the stream, the total space used by the players, the number of rounds of communication, and the total amount of communication (as well as the running time used by the players and coordinator).

We study this setting for the heavy hitters problem: The task is that the coordinator knows the heavy hitters of x, more precisely the coordinator computes a set R of size $|R| = O(1/\epsilon^2)$ that with high probability contains each index i with $|x_i| \geq \epsilon \cdot \|x_{-1/\epsilon^2}\|_2$.

We show that this distributed streaming heavy hitters problem can be solved with 2 passes over the streams, $O((\epsilon^{-2} + d) \log^2 n)$ space in total (over all players and the coordinator, in words), 4 communication rounds, $O((\epsilon^{-2} + d) \log^2 n)$ total communication (in words), and total time $O(d \log^4 n + (\epsilon^{-2} + u) \log^2 n)$, where u is the total length of the input streams.

The Algorithm. For notational convenience, we encode the partitioning $I_1 \cup \ldots \cup I_d$ by the matrix $X \in \mathbb{R}^{d \times n}$ with $X[i, j] = x_i$ if $i \in I_j$, and 0 otherwise. Then the j-th column Xe_j is equal to the vector given by the stream of player j. Observe that computing the heavy hitters of x is equivalent to computing the heavy hitters of X. We run the following protocol.

1. The coordinator samples the matrix S from Theorem 8 and sends it to all players. (If all players have access to shared randomness, then this communication round is not necessary.)
2. Now the players make one pass over their streams, where player j computes $S(Xe_j)$. They send the results to the coordinator, who now knows SX.
3. The coordinator now uses Theorem 8 with $k := \lceil 1/\epsilon^2 \rceil$ to compute budgets $t_1, \ldots, t_d$. She informs each player of their respective assigned budgets t_j.
4. Now the players make a second pass over their streams, where player j computes the t_j-heavy hitters (i.e., identifies the entries in the stream larger than $1/\sqrt{t_j + 1}$ times the ℓ_2-mass of all but the t_j heaviest elements), for instance by Lemma 7. They send the heavy entries to the coordinator, who reports the union of the all the received entries.

Analysis of Communication, Space, and Time. The algorithm uses 4 communication rounds. The total communication (in words) is $O(d \log^2 n)$ from sending S and the vectors $S(Xe_j)$, plus $O(\epsilon^{-2} \log^2 n)$ from sending the heavy hitters; their total number is indeed

$$O\left(\sum_j t_j \log^2 n\right) = O(k \log^2 n) = O(\epsilon^{-2} \log^2 n).$$

The total space is $O(d \log^2 n)$ from each player j storing S, the vector $S(Xe_j)$, and the matrix $M^{(\ell_j)}$, plus $O(\epsilon^{-2} \log^2 n)$ from storing the vectors $M^{(\ell_j)}(Xe_j)$. The total time is $O(d \log^4 n + \epsilon^{-2} \log^2 n)$ from Theorem 8, plus $O(u \log^2 n)$ from the passes over the streams

of total length u. Note that each update in the stream involves exactly one column of S or the heavy hitter sketch, and we can compute the non-zero entries of each such column in time $O(\log^2 n)$, therefore each update can be performed in time $O(\log^2 n)$. The remaining steps are dominated by this running time.

Correctness. Since player j reports $O(t_j)$ heavy hitters, the number of heavy hitters ultimately reported by the coordinator is $\sum_j O(t_j) \leq O(k) = O(\epsilon^{-2})$. It remains to prove that with high probability we report all heavy hitters of X. Suppose that the i-th entry in the stream of player j satisfies $|X[i,j]| \geq \epsilon \|X_{-k}\|_F$. In this case the entry also exceeds $\|X_{-k}\|_F^2/(k+1)$, and thus, by Theorem 8, also $\|(Xe_j)_{-t_j}\|_2^2/(t_j+1)$. In other words, the i-th item in the stream is a heavy hitter that is reported with high probability in step 4.