Sketching via hashing: from heavy hitters to compressed sensing to sparse fourier transform

The MIT Faculty has made this article openly available. Please share how this access benefits you. Your story matters.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>As Published</td>
<td><a href="http://dx.doi.org/10.1145/2463664.2465217">http://dx.doi.org/10.1145/2463664.2465217</a></td>
</tr>
<tr>
<td>Publisher</td>
<td>Association for Computing Machinery</td>
</tr>
<tr>
<td>Version</td>
<td>Author's final manuscript</td>
</tr>
<tr>
<td>Accessed</td>
<td>Mon Feb 04 02:01:03 EST 2019</td>
</tr>
<tr>
<td>Citable Link</td>
<td><a href="http://hdl.handle.net/1721.1/86898">http://hdl.handle.net/1721.1/86898</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>Creative Commons Attribution-Noncommercial-Share Alike</td>
</tr>
<tr>
<td>Detailed Terms</td>
<td><a href="http://creativecommons.org/licenses/by-nc-sa/4.0/">http://creativecommons.org/licenses/by-nc-sa/4.0/</a></td>
</tr>
</tbody>
</table>
Sketching via Hashing: from Heavy Hitters to Compressive Sensing to Sparse Fourier Transform

Piotr Indyk
MIT
indyk@mit.edu

Categories and Subject Descriptors
F.2.1 [Analysis of Algorithms and Problem Complexity]: Numerical Algorithms and Problems—Computations on matrices; Computation of transforms (e.g., fast Fourier transform)

General Terms
Algorithms

Keywords
Dimensionality reduction, hashing, sparse Fourier transform, sparsity

1. INTRODUCTION

Sketching via hashing is a popular and useful method for processing large data sets. Its basic idea is as follows. Suppose that we have a large multi-set of elements \( S = \{a_1, \ldots, a_s\} \subset \{1 \ldots n\} \), and we would like to identify the elements\(^1\) that occur “frequently” in \( S \). The algorithm starts by selecting a hash function \( h \) that maps the elements into an array \( c[1 \ldots m] \). The array entries are initialized to 0. Then, for each element \( a \in S \), the algorithm increments\(^2\) \( c[h(a)] \). At the end of the process, each array entry \( c[j] \) contains the count of all data elements \( a \in S \) mapped to \( j \). It can be observed that if an element \( a \) occurs frequently enough in the data set \( S \), then the value of the counter \( c[h(a)] \) must be large. That is, “frequent” elements are mapped to “heavy” buckets. By identifying the elements mapped to heavy buckets and repeating the process several times, one can efficiently recover the frequent elements, possibly together with a few extra ones (false positives).

\(^1\)These elements are often referred to as heavy hitters or elephants.

\(^2\)Typically, the value is incremented by 1. However, some algorithms such as Count Sketch [CCF02] or the pre-identification procedure of [GGI+02b] use randomly chosen increments.

Variants of this method have originated in several fields, including databases [FSGM+98, CM03a, CM03b], computer networks [FCAB98, EV02] (cf. [BM04]) and theoretical computer science [CCF02, GGI+02b, CM04]. One of the key features of this method is that it allows to approximate the counts of the elements using very limited storage while making only a single pass over the data. As a result, the method has become one of the staples in the field of data stream computing [Mut05]. However, this was just the beginning. Over the last decade, this approach has been used to design improved algorithms for remarkably diverse tasks such as compressive sensing, dimensionality reduction and sparse Fourier transforms. In this survey we give a brief overview of how hashing is used in the aforementioned applications.

In order to apply the approach to those tasks, the first step is to view the hashing process as a linear mapping of the characteristic vector \( x \) of the set \( S \) to the vector \( c \). Specifically, for any \( j = 1 \ldots m \), 
\[
c[j] = \sum_{a \in S} x_a \]
This can be written as \( c = Ax \) where \( A \) is a sparse binary \( m \times n \) matrix. The algorithmic benefit of using such mappings is due to the sparsity of the matrix \( A \) (which makes it easy to perform various tasks such as matrix-vector multiplication efficiently) as well as the overall simplicity of the hashing process.

2. COMPRESSED SENSING

In compressed sensing [Don06, CRT06] one is given the vector \( Ax \) and the goal is to recover an approximation \( x' \) to \( x \) that is \( k \)-sparse, i.e., that has at most \( k \) non-zero entries. The approximation should (approximately) minimize the error \( \|x' - x\|_p \) for some choice of the \( \ell_p \) norm. Note that for any value of \( p \), the error \( \|x - x'\|_p \) is minimized when the approximation \( x' \) consists of the \( k \) largest (in magnitude) coefficients of \( x \). This problem has numerous applications in signal processing or imaging, where signals are quite sparse, possibly after applying an appropriate change-of-basis transform. In those applications compressed sensing allows one to recover a good approximation to a signal \( x \) from only few “measurements” \( Ax \). In particular, the result of [CRT06] shows that one can recover a \( k \)-sparse approximation to \( x \) using only \( m = O(k \log(n/k)) \) measurements, and it is known that this bound cannot be improved [Don06, DIPW10, FPRU10]. The bound is achieved using matrices \( A \) with random i.i.d. Gaussian or Bernoulli entries. Unfortunately, any operation on such matrices takes \( O(nm) \)
time, which makes the recovery algorithms somewhat slow for high values of $n$.

It was observed in [CM06] (cf. [GI10]) that the algorithm of [CCF02] yields a recovery procedure and a matrix $A$ with $O(k \log n)$ measurements, which is not too far from the optimal bound (although the recovery procedure is only correct with high probability). At the same time, thanks to the sparsity of the matrix $A$, the approximation $x'$ can be computed in only $O(n \log n)$ time. Compressive sensing via sparse matrices has attracted a considerable interest in the literature, see e.g., [SBMO6, CM06, WGR07, GSTV07, XH07, SBB10, WWR10, KDXH11, Ind08, LMP\(^+\)08, BGI\(^+\)08, IR08, BIR08, JXH09, GLPS10, GM11, PR12, BCG12] or a survey [GI10]. In particular, the results of [BGI\(^+\)08, IR08, BIR08, GLPS10] show that sparse matrices can match the optimal $O(k \log(n/k))$ measurement bound achieved via fully random matrices while supporting faster algorithms, albeit in some cases providing somewhat weaker approximation guarantees.

### 3. DIMENSIONALITY REDUCTION

A mapping from $x$ to $Ax$ can be used to reduce the dimensionality of general (non-sparse) vectors $x$, as per the Johnson-Lindenstrauss theorem [JL84]. The original theorem used random dense matrices, which necessitated $O(nm)$ matrix-vector multiplication time. Faster dimensionality reduction is possible by using structured matrices that support multiplication procedures [AC10, AL11, KG11, NPW12], but the reduced dimension is either sub-optimal or restricted. Moreover, the running times of those procedures do not scale with the sparsity of the vector $x$. In contrast, the line of research on sparse dimensionality reduction matrices [SPD\(^+\)09, WDL\(^+\)09, DKS10, BOR10, KN12] has led to matrices with optimal reduced dimension bounds that are supported by algorithms with run-time $O(km)$, where $k$ is the number of non-zero entries in $x$ and $\epsilon > 0$ is an approximation parameter that is arbitrarily close to 0. Using such matrices, [CW13] (see also [NN12, MM13]) recently showed almost linear time approximate algorithms for sparse regression and low-rank approximation, the key problems in numerical linear algebra.

### 4. SPARSE FOURIER TRANSFORM

The Discrete Fourier Transform (DFT) maps an $n$-dimensional signal $x$ sampled in time domain into an $n$-dimensional spectrum $\hat{x}$. The widely used Fast Fourier Transform algorithm performs this task in $O(n \log n)$ time. It is not known whether this algorithm can be further improved. However, it is known that one can compute DFT significantly faster for signals whose spectrum is (approximately) sparse. Such sparsity is common for many data sets occurring in signal processing, imaging and communication. For such signals, one may hope for faster algorithms.

The first algorithms of this type were designed for the Hadamard Transform, i.e., the Fourier Transform over the Boolean cube [KM91, Lev93] (cf. [GLS99, Go99]). Soon, algorithms for the complex Fourier transform were discovered

\(^3\)This issue can be alleviated by using random structured matrices, which support matrix-vector product in $O(n \log n)$ time [CT06, RV06, CVG13, NPW12]. However, even the best construction due to [NPW12] requires the number of measurements to be $O(\log^2 n)$ times larger than optimal, at least in theory.

\(^4\)See also [GOS\(^+\)11] which applied a similar method to the related problem of testing Fourier sparsity.

### 5. REFERENCES


[BM04] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A


M. A. Iwen. Combinatorial sublinear-time Fourier algorithms. Foundations of


