Constant-Time Retrieval with $O(\log m)$ Extra Bits

Martin Dietzfelbinger
Technische Universität Ilmenau, Germany
martin.dietzfelbinger@tu-ilmenau.de

Stefan Walzer
Technische Universität Ilmenau, Germany
stefan.walzer@tu-ilmenau.de

Abstract

For a set $\mathcal{U}$ (the *universe*), *retrieval* is the following problem. Given a finite subset $S \subseteq \mathcal{U}$ of size m and $f\colon S \to \{0,1\}^r$ for a small constant r, build a data structure D_f with the property that for a suitable query algorithm query we have $\mathsf{query}(D_f, x) = f(x)$ for all $x \in S$. For $x \in \mathcal{U} \setminus S$ the value $\mathsf{query}(D_f, x)$ is arbitrary in $\{0,1\}^r$. The number of bits needed for D_f should be $(1+\varepsilon)rm$ with *overhead* $\varepsilon = \varepsilon(m) \geq 0$ as small as possible, while the query time should be small. Of course, the time for constructing D_f is relevant as well.

We assume fully random hash functions on $\mathcal{U}$ with constant evaluation time are available. It is known that with $\varepsilon \approx 0.09$ one can achieve linear construction time and constant query time, and with overhead $\varepsilon_k \approx e^{-k}$ it is possible to have $O(k)$ query time and $O(m^{1+\alpha})$ construction time, for arbitrary $\alpha > 0$. Furthermore, a theoretical construction with $\varepsilon = O((\log\log m)/\sqrt{\log m})$ gives constant query time and linear construction time. Known constructions avoiding all overhead, except for a seed value of size $O(\log\log m)$, require logarithmic query time.

In this paper, we present a method for treating the retrieval problem with overhead $\varepsilon = O((\log m)/m)$, which corresponds to $O(1)$ extra memory words ($O(\log m)$ bits), and an extremely simple, constant-time query operation. The price to pay is a construction time of $O(m^2)$. We employ the usual framework for retrieval data structures, where construction is effected by solving a sparse linear system of equations over the 2-element field $\mathbb{F}_2$ and a query is effected by a dot product calculation. Our main technical contribution is the design and analysis of a new and natural family of sparse random linear systems with m equations and $(1+\varepsilon)m$ variables, which combines good locality properties with high probability of having full rank.

Paying a larger overhead of $\varepsilon = O((\log m)/m^{\alpha})$, the construction time can be reduced to $O(m^{1+\alpha})$ for arbitrary constant $0 < \alpha < 1$. In combination with an adaptation of known techniques for solving sparse linear systems of equations, our approach leads to a highly practical algorithm for retrieval. In a particular benchmark with $m = 10^7$ we achieve an order-of-magnitude improvement over previous techniques with $\varepsilon = 0.24\%$ instead of the previously best result of $\varepsilon \approx 3\%$, with better query time and no significant sacrifices in construction time.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Retrieval, Hashing, Succinct Data Structure, Randomised Data Structure, Structured Gaussian Elimination, Method of Four Russians

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.24

1 Introduction

A *retrieval data structure* for a universe $\mathcal{U}$ (a set) and r-bit values represents a function from $\mathcal{U}$ to $R = \{0,1\}^r$ with prescribed values on a set $S \subseteq \mathcal{U}$ of size m. We need an algorithm construct that builds the data structure and an operation query. The input for construct is a function $f\colon S \to R$ (given as a list of argument-value pairs), the result is a data structure D_f, whose binary length we denote by $|D_f|$. The query algorithm query has two inputs, D_f and $x \in \mathcal{U}$, and returns an element of R. We require that

$\mathsf{query}(D_f, x) = f(x)$, for all $x \in S$.

36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019).
Editors: Rolf Niedermeier and Christophe Paul; Article No. 24; pp. 24:1–24:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The values $\mathsf{query}(D_f, x)$ for $x \in \mathcal{U} \setminus S$ are irrelevant.

Relevant parameters of a retrieval data structure are (1) the space $|D_f|$ in bits in terms of $m = |S|$ and r, (2) the running time of query, and (3) the running time of construct. In this paper, we focus on minimising (1) while keeping (2) a constant, as small as possible. As for (3), it is kept in the $O(m^{1+\alpha})$ range. We make some efforts to extend the range of practical usability of our approach. (This is only partly reflected in this paper.) Note that storing all pairs $(x, f(x))$ for $x \in S$ in a dictionary data structure is not good enough for our purposes, as in general this requires $|D_f| = rm + \Omega(m \log m)$ bits. Since it is not necessary to decide membership in S or a subset of S, space $|D_f| = O(rm)$ is sufficient.

There is a close connection between retrieval data structures and perfect hashing. Most of the perfect hash functions that get by with linear space utilise a retrieval data structure with $r = 2$, see [6, 7, 15]. (Exceptions are the optimal theoretical construction in [16] and Hash-Displace-Compress in [4].) Conversely, if a hash function $g\colon \mathcal{U} \to [p]$ is given that is perfect on S, one can store $f(x)$, $x \in S$, in position $T[g(x)]$ of a table $T[0..p-1]$ to obtain a retrieval data structure with space pr plus the space for storing g. However, if r is small and the goal is a retrieval structure with very small space overhead, this detour via perfect hashing is inefficient.

1.1 Basic Data Structure

The basic setup of the data structure is well known and well studied. For some $n = n(m) \geq m$, a set $W \subseteq \{0,1\}^n$ of (typically sparse) vectors is chosen. One hash function or several hash functions are used to map elements x of $\mathcal{U}$ to $row(x) \in W$.

The data structure D_f consists of row[1] and a vector $\vec{z} = (z_1, \dots, z_n) \in (\{0,1\}^r)^n$, which in construct is chosen in such a way that the query algorithm

$$\mathsf{query}(D_f, x) = \bigoplus_{\substack{1 \leq i \leq n \\ (row(x))_i = 1}} z_i, \quad \text{for } x \in \mathcal{U}, \tag{1}$$

yields $f(x)$ for $x \in S$. To carry out $\mathsf{query}(D_f, x)$, one only has to find the components z_i with $(row(x))_i = 1$ and perform $\bigoplus$, the bitwise XOR, of these components.

Note that in this construction the $r \geq 1$ components of elements of R are just treated independently (or "in parallel"). In order to simplify notation, we concentrate on the case $r = 1$ from here on. (The generalisation is immediate. The query time has to be multiplied by r.) In this case (1) turns into

$$f(x) = \langle row(x), \vec{z} \rangle, \tag{2}$$

where $\langle \cdot, \cdot \rangle$ is the dot product of n-bit vectors. For construct one has to solve a linear system $(\langle row(x), \vec{z} \rangle = f(x))_{x \in S}$ over $\mathbb{F}_2$ for the vector $\vec{z}$ of "unknowns". For this to be possible it is sufficient that the $m \times n$ matrix $A(S, row)$ with rows $row(x)$, $x \in S$, has full row rank. We assume that $row(x)$, $x \in U$, are stochastically independent and identically distributed, and we may simply write $A_{m,n}$ for a random variable distributed as $A(S, row)$, if the distribution of row is clear from the context.

[1] In all our constructions we assume that fully random independent hash functions $(\varphi_{i,j})_{i,j}$ with suitable range $\{1, \dots, \mathrm{poly}(m)\}$ are available for free and can be evaluated on $x \in \mathcal{U}$ in constant time. Fixing a seed value j, one gets $(h_0, h_1, \dots) = (\varphi_{0,j}, \varphi_{1,j}, \dots)$ from which row is constructed. Hence, an index j suffices to identify row.

Algorithm
construct($S \subseteq \mathcal{U}, f\colon S \to \{0,1\}$):
  pick $row\colon S \to W \subseteq \{0,1\}^{(1+\varepsilon)|S|}$
  solve $(\langle row(x), \vec{z}\rangle = f(x))_{x\in S}$ for $\vec{z}$
  restart with new row if unsolvable
  return $D = (row, \vec{z})$

Algorithm
query($D_f = (row, \vec{z}), x \in \mathcal{U}$):
  return $\langle row(x), \vec{z}\rangle$

Figure 1 The general framework for retrieval data structures for $R = \{0,1\}$.

1.2 Previous Work and Relevant Techniques

Construction time and query time depend very much on the structure of the vectors in W. Most earlier works [6, 7, 8, 11, 15] chose W as the set of vectors with constant *Hamming weight* k, i.e. k entries of $row(x)$ are 1, for some $k \geq 3$. The operation query$(x, \vec{z})$ then just reads the k positions given by $row(x)$ in $\vec{z}$, which results in running time $O(k)$ for a query. (Only in [11, 21] sets W with vectors of Hamming weight $\Theta(\log m)$ are considered. Applied in the obvious way, this will lead to query times of $\Theta(\log m)$.) There is a simple case, which also admits linear construction time, namely when the rows of $A_{m,n}$ can be brought into triangular form by row and column *exchanges* alone. This is equivalent to the k-uniform hypergraph $G_{m,n}$ with incidence matrix $A_{m,n}^{\mathsf{T}}$ being *peelable*, i.e. having an empty *2-core* [18]. In this case we do not even need a field to compute in (and no linear algebra), but (1) can be taken to be a formula over any group $(R, \oplus)$ (like $\mathbb{Z}/m\mathbb{Z}$ with addition modulo m). This approach is underlying the constructions in [6, 7, 8, 15, 18, 23]. The common feature utilised in these works is that for fixed $k \geq 2$ there is a density threshold $c_k^{\triangle}$ such that, for arbitrary constant $\delta > 0$, $G_{m,n}$ is peelable with high probability (whp) for $m < (c_k^{\triangle} - \delta)n$ and not peelable whp for $m > (c_k^{\triangle} + \delta)n$. A description of these thresholds can be found in [18], a proof in [19]. The largest threshold value is $c_3^{\triangle} \approx 0.81847$. In [22] it was shown that rows with nonuniform weight (e.g. 3 for about 88% of keys and 21 for the rest) can raise the threshold to over 0.92. It is open if with constant average Hamming weight a quotient m/n arbitrarily close to 1 will still lead to peelability whp.

As our aim is to achieve $m = cn$ for $c = 1 - o(1)$, we have to give up peelability. A standard approach [2, 11, 15] uses bit vectors with constant weight $k \geq 3$ and in the construct routine requires solving the linear system (2) over $\mathbb{F}_2$. In [10] it was claimed and in [20] it was rigorously proved that there are thresholds c_k^* such that for $c < c_k^*$ and $m = cn$, $m, n \to \infty$ we have solvability whp, and for $c > c_k^*$ and $m = cn$, $m, n \to \infty$ we have unsolvability whp. These thresholds are the same as for simple orientability of k-uniform hypergraphs [10, 13, 14]. (Numerical values for some k can be found in [10].) The values $1 - c_k^*$ approach e^{-k} as k increases, and hence c_k^* is quite close to 1, but for constant k a constant gap remains. Queries take time $\Theta(k)$ and are not cache efficient, since k random components of $\vec{z}$ are accessed. In [11] it was shown (utilising a result from [9]) that with $k = k(m) = O(\log m)$, one can achieve solvability for $m = n$, with constant probability. A similar observation was made in [21]. The query time increases to $O(\log m)$.

A serious obstacle in these methods for reducing the overhead is that one has to solve a linear system without the advantage of peelability. So it is necessary to address the running time of this task. Gaussian elimination, applied naively, needs time $\Theta(m^3)$, which already for moderately large m becomes infeasible. Exploiting the fact that the rows are sparse, a variant of Wiedemann's algorithm [24] will reduce the time to $O(m^2 \log^2 m)$ (for fixed k) and to $O(m^2 \log m)$ (for $k = k(x) = O(\log m)$). Alternatively, one may use clever

variants of Gaussian elimination ("structured" in [17] and "lazy" in [15]), which try to delay the system filling up with 1's. As far as we know, there is no mathematical analysis of such techniques, although in experiments they drastically reduce the running time of naive Gaussian elimination on sparse random matrices, outperforming Wiedemann's algorithm for small to medium inputs. So, we do not use these techniques in our theoretical analysis, but we utilise them in our experimental implementations. Another standard speedup technique for Gaussian elimination, of course, is word-level parallelism, where the rows of the linear system are split into machine words and row operations can be performed by a sequence of bitwise boolean XORs. If the word length is w, the running times may be divided by w. Finally, the "Technique of Four Russians" is applicable to Gaussian elimination. By filling a lookup-table with certain precomputed row sums, it achieves a speedup by a factor of $\Theta(\log m)$; for a description see [3].

A last, very important technique for achieving feasible construction times for large retrieval data structures is *input partitioning*. It has been used in many works on dictionary-like data structures, see, e.g., [2, 11, 15, 21]. Let $[n]$ denote the set $\{1, \ldots, n\}$. Using a "level-1" hash function $h_0 \colon \mathcal{U} \to [m/C]$, for some C (which w.l.o.g. divides m), one splits S into *chunks* $S_i = h_0^{-1}(\{i\}) \cap S$, for $i = 1, \ldots, m/C$. The expected size of each chunk is C, and if h_0 is fully random and C is not too small, one has $|S_i| = O(C)$ for all i, whp. One sets up a separate retrieval data structure $D_{f \restriction S_i}$ for each S_i. When using Gaussian elimination with $|C| = m^\alpha$ to construct each $D_{f \restriction S_i}$, the total construction time is reduced to $O((m/C) \cdot C^3) = O(mC^2) = O(m^{1+2\alpha})$, and with Wiedemann's algorithm [24] we get construction time $O((m/C) \cdot C^2 \log^2(C)) = O(m^{1+\alpha} \log^2 m)$. The downside is that one gets an additional "outer overhead" for saving for each i a pointer to $D_{f \restriction S_i}$. Using the offsets $\sum_{1 \le i < j} |D_{f \restriction S_i}|$ for this purpose costs $O((m/C) \log m)$ bits. This space we have to pay for the reduction in construction time. In [11] and [21] chunks of size $O(\sqrt{\log m})$ are used. An extra auxiliary data structure is employed to accommodate "bad" keys from chunks that overflow. Moreover, these papers use lookup tables for solving tiny systems of equations ($O(\sqrt{\log m})$ variables and equations). While using sublogarithmic chunk sizes has good properties in theory, it does not seem to be competitive in practice, see [2].[2]

1.3 Our Contribution

There are three degrees of freedom in the retrieval framework described above, regarding both theory and implementation:

- **(F1)** What is the set W of "sparse" vectors that is the range of *row*?
- **(F2)** Which method is used for solving the linear system?
- **(F3)** How, if at all, do we partition the input to reduce the influence of a high running time of the solver?

The main contribution of this paper is to propose and analyse a new answer to **(F1)**. The effect is that the query time is now constant – it involves only access to two memory locations and a small dot product calculation –, while the overhead drops to $m^{-\alpha}$ for a constant $\alpha \in (0, 1)$, or even to $O((\log m)/m)$, which means that $n = m + O(\log m)$. The method is very simple and natural: The 1's in $row(x)$ are concentrated in two blocks of size $O(\log m)$, so that the non-zero part of $row(x)$ fits in $O(1)$ words in the (standard) word RAM model.

[2] In [12] it is explained how one can justify the full-randomness assumption by partitioning S and employing an auxiliary data structure of size $C^{1+\Omega(1)}$ to provide fully random hash functions on each chunk. We do not discuss this aspect of partitioning here.

(This computational model was also used in [11, 21], and it is the basis of the speedup of the Four Russians method.) We are not aware of this idea having been used in the context of retrieval structures or perfect hashing before.[3] Its appeal is in its simplicity and in its apparent practicality. (Our experiments show that a significant reduction in overhead is possible for sets S of realistic size, with construction times comparable to the other approaches.) Let $n = m \cdot (1+\varepsilon)$. We describe $W \subseteq \{0,1\}^n$ and a distribution of $row(x) \in W$ as follows. Fix a *block size* $\ell = O(\log n)$ that divides n. For $b \in [n/\ell]$ and $p \in \{0,1\}^\ell$ we let $B_{b,p} = 0^{b\ell-\ell}p0^{n-b\ell} \in \{0,1\}^n$. Two block indices $b_1, b_2 \in [n/\ell]$ and two patterns $p_1, p_2 \in \{0,1\}^\ell$ are chosen uniformly at random, and $row(x)$ is set to be $B_{b_1,p_1} \oplus B_{b_2,p_2}$, with at most two non-zero blocks. Bit parallelism allows computing dot products involving $row(x)$, and thus answer queries, in $O(1)$ time. It is the main contribution of this work to establish that an overhead of $\Theta(\frac{\log m}{m})$ is achievable using this approach (we call it "inner overhead" if we want to distinguish it from the "outer overhead" needed due to the use of partitioning techniques). The theoretical analysis of the probability of obtaining a solvable system of equations, meaning that $A_{m,n}$ has full row rank, uses a first moment argument. Note that our result is also a significant theoretical improvement. Previously, overhead $\varepsilon = m^{-\delta}$ for constant δ required query cost $O(\log m)$ while query cost $O(1)$ required overhead $\varepsilon = \Omega(\frac{\log\log m}{\sqrt{\log m}})$, see Table 1.

▶ **Theorem 1** (Main Theorem). *Assume the context of a word RAM with word length $w = \Theta(\log n)$ and access to fully random hash functions*[4]*.*

- **(i)** *For any $r < m$ there is an r-bit retrieval data structure with overhead $\varepsilon = O(\frac{\log m}{m})$ and a construction that succeeds in time $O(\frac{m^3}{w \log m})$ whp. Queries take $O(r)$ time and access two contiguous segments of memory.*
- **(ii)** *An alternative construction based on Wiedemann's algorithm runs in time $O(rm^2)$ whp.*

Construction time can be improved by taking **(F3)** into account: randomly partition the input into chunks of expected size C and use the construction from Theorem 1 on each of the $\frac{m}{C}$ chunks. For each chunk, we need to store a pointer to its data and a seed for the function row used in the successful construction. This requires $O(\frac{m}{C}\log m)$ and $O(\frac{m}{C}\log\log m)$ extra bits, respectively. We then get:

▶ **Corollary 2.** *Under the same conditions as Theorem 1, we have:*

- **(i)** *For any $C = m^\alpha$ $(0 < \alpha \leq 1)$ and any $r < C$ there is an r-bit retrieval data structure with overhead $\varepsilon = O(\frac{\log m}{C})$ and a construction that succeeds in time $O(\frac{mC^2}{w \log C})$ whp. Queries take $O(r)$ time and access three contiguous segments of memory*[5]*.*
- **(ii)** *A alternative construction based on Wiedemann's algorithm runs in time $O(rmC)$.*

Table 1 summarises previous work and the new construction if the effect of partitioning is taken into account. Also, obvious improvements achieved by replacing Gaussian elimination by Wiedemann's algorithm are reflected. The choice of C constitutes a trade-off between construction time and total overhead, which is the sum of the overhead from the chunks ("inner") and from organising the data structures for the single chunks ("outer"). For $C = m$ there are no chunks, the construction time is maximal, and only the "inner" overhead is

[3] Possibly it was vaguely anticipated in [21], where the author suggested using "sparse equations that are more or less local".

[4] For any universe $\mathcal{U}$ and any finite domain D we assume oracle access to fully random functions $(h_i : \mathcal{U} \to D)_{i\in\mathbb{N}}$, meaning we need to only store an index i to describe such a function. This assumption is motivated by the observation that good (pseudo-)randomness is usually not an issue in practice.

[5] In practice it is reasonable to expect two cache faults per query.

Table 1 Comparison of previous work and the results of this paper. Where query times are not enclosed in O-Notation the number vaguely counts accesses to random memory locations. The column $t_{\mathsf{construct}}$ reports the construction times given in the respective paper and alternatively times improved by utilising Wiedemann's algorithm [24]. Regarding the results of [11], the better thresholds from [20] are substituted.

Paper	t_{query}	$t_{\mathsf{construct}}$	"inner overhead" + "outer overhead"	Practical?
[7, 18]	3	$O(m)$	$0.23 + 0$	✓
[22]	$O(1)$	$O(m)$	$0.087 + 0$	✓
[11]	k	$O(m^3)$ or $O(m^2 \log^2 m)$	$\mathrm{e}^{-k} + o(\mathrm{e}^{-k}) + 0$	✗
[11]	$O(k)$	$O(m)$	$\mathrm{e}^{-k} + o(\mathrm{e}^{-k}) + \Omega((\log m)^{-1/4})$	✗
[11, 21]	$\log m$	$O(m^3)$ or $O(m^2 \log^2 m)$	$0 + O(\frac{\log\log m}{m})$	✗
[21]	$O(1)$	$O(m)$	$0 + \Omega(\frac{\log\log m}{\sqrt{\log m}})$	✗
[2]	$O(k)$	$O(mC^2)$ or $O(mC \log^2 C)$	$\mathrm{e}^{-k} + o(\mathrm{e}^{-k}) + \Omega(C^{-1/2})$	(✓)
[15]	3 [or 4]	$O(\frac{mC^2}{w})$ or $O(mC \log^2 C)$	0.09 [or 0.024] $+ \Theta(\frac{\log m}{C})$	✓
⟨**new**⟩	2	$O(\frac{mC^2}{w \log C})$ or $O(mC)$	$\Theta(\frac{\log m}{C}) + \Theta(\frac{\log m}{C})$	✓

relevant. Both from a theoretical and practical point of view the reduction from "ε is constant" or "$\varepsilon = O(\frac{\log\log m}{\sqrt{\log m}})$" to a polynomially small overhead is significant. Pleasingly, our approach compares very favourably with previous results in practical benchmarks, as shown in Table 2 and explained in Section 5.

Table 2 Comparison of our algorithm in the form presented in Section 5 to the arguably best-so-far results reported in [15]. We achieve much smaller overhead with comparable run times.

	overhead	Construction [µs/key]	Lookup [ns]
[15] $k = 3$	9%	1.12	210
[15] $k = 4$	3%	1.75	236
⟨this paper⟩	0.24%	2.6	75–125

Structure of the paper. In Section 2 we define a matrix $A^{\ell}_{m,n}$ and a related graph $G^{\ell}_{m,n}$ that formally capture the problem of constructing our retrieval data structure. In Section 3 we show that $A^{\ell}_{m,n}$ has full rank whp, which is the main ingredient used to prove our Main Theorem in Section 4. Lastly, in Section 5 we briefly present an implementation of our approach. A discussion of practical improvements we employed is postponed to the full version of this paper.

2 The Construction Problem in Matrix and Graph Terminology

We now formalise our idea of using "vectors with coefficients within two blocks".

Let $S \subseteq \mathcal{U}$ be a set annotated with $f\colon S \to \{0,1\}$, $\ell \in \mathbb{N}$ the *block size*, $|S| = m$ and $n \geq m$ with $\ell \mid n$. Moreover, we pick a uniformly random function $h\colon \mathcal{U} \to [n/\ell] \times [n/\ell] \times \{0,1\}^{\ell} \times \{0,1\}^{\ell}$ with components we call $h = (b_1, b_2, p_1, p_2)$ that implicitly characterise *row*. Together, (S, f, m, n, ℓ, h) is an instance of the construction problem for our retrieval data structures. The task to be solved can be expressed in two equivalent ways.

Matrix terminology. For $b \in [n/\ell]$ and $p \in \{0,1\}^\ell$ let $B_{b,p} = 0^{b\ell-\ell}p0^{n-b\ell} \in \{0,1\}^n$. Then each $x \in S$ is identified with the equation $\langle B_{b_1(x),p_1(x)} \oplus B_{b_2(x),p_2(x)}, \vec{z}\rangle = f(x)$ where $\vec{z} \in \{0,1\}^n$ is a vector of unknowns. Together, S is a system of equations $A\vec{z} = \vec{b}$ where $\vec{b} = (f(x))_{x\in S}$. The matrix $A = A^\ell_{m,n}$ will be examined thoroughly in Section 3.

Graph terminology. The problem instance can also be conveniently captured as a graph $G = G^\ell_{n,m} = ([n/\ell], S)$ with labels. Each vertex corresponds to a block of ℓ variables and $x \in S$ is identified with an edge $\{b_1(x), b_2(x)\}$ where the incidence $(x, b_1(x))$ is labelled with $p_1(x)$, the incidence $(x, b_2(x))$ is labelled with $p_2(x)$ and x itself is labelled with $f(x)$. A solution is now a vertex labelling $x\colon [n/\ell] \to \{0,1\}^\ell$ with $\langle p_1(x), x(b_1(x))\rangle \oplus \langle p_2(x), x(b_2(x))\rangle = f(x)$ for all $x \in S$.

We will borrow notions from graph theory in algebraic discussions, when convenient. For instance, we may speak of the degree of a block of variables or a connected set of equations, meaning the degree of a corresponding vertex or the connectedness of a corresponding set of edges.

Loops and parallel edges. It is possible to have $b_1(x) = b_2(x) = b$ for some $x \in S$. Then $row(x)$ contains $p_1 \oplus p_2$ in block b and G has a loop at vertex b with two labels p_1 and p_2. Moreover two distinct elements $x \neq x'$ may be associated with the same two blocks. In this case G is a multigraph.

Forbidding the all-zero pattern. Let $h^*\colon \mathcal{U} \to [n/\ell]\times[n/\ell]\times(\{0,1\}^\ell\setminus\{0^\ell\})\times(\{0,1\}^\ell\setminus\{0^\ell\})$ be uniformly random. Compared to h, the pattern 0^ℓ is forbidden in h^*. This gives rise to random matrices $A^{\ell*}_{m,n}$ and graphs $G^{\ell*}_{m,n}$ with higher probability of admitting solutions, see Proposition 3.

3 Full Rank of the Linear Systems

We now provide the main ingredient for Theorem 1, establishing that the matrices $A^\ell_{cn,n}$ defined in Section 2 have full rank whp. We also consider two natural variations concerning $A^{\ell*}_{cn,n}$.

Throughout this section, logarithms have base 2, $\bar{c}$ is a shorthand for $1-c$ and *with high probability* (whp) refers to a probability of $1 - n^{-\varepsilon}$ for some $\varepsilon > 0$.

▶ **Proposition 3.** *Let $\beta = 27$ and $\gamma = 1/4$. Then we have:*

- **(i)** *If $2\ell = 2\ell(n) \geq (1+\delta)\log n$ for $\delta > 0$, then $A^\ell_{cn,n}$ has independent rows whp, provided that $\bar{c} \geq \max\{2^{-\gamma\ell}, \beta\log(n)/n\}$.*
- **(ii)** *If $\ell = \ell(n) = \omega(1)$ then $A^{\ell*}_{cn,n}$ has independent rows with probability $1-o(1)$, provided that $\bar{c} \geq \max\{2^{-\gamma\ell}, \beta\log(n)/n\}$.*
- **(iii)** *If ℓ is a large enough constant, then $A^{\ell*}_{cn,n}$ has independent rows with probability $\Theta(1)$, provided that $\bar{c} \geq 2^{-\gamma\ell}$.*

Remarks.

- For $2\ell = \log n$ the matrix $A^\ell_{cn,n}$ has dependent rows with constant probability, simply because the number of all-zero rows is binomially distributed with expectation $cn \cdot 2^{-2\ell} =$

$c = \Theta(1)$. In this sense **(i)** is best possible. This motivates considering $A^{\ell*}_{cn,n}$ where all-zero rows are far less likely[6].

- For $\ell = \Theta(1)$ the probability that $A^{\ell*}_{cn,n}$ has two identical rows is $\Theta(1)$. This implies that $\ell = \omega(1)$ is best possible in **(ii)** and the probability of $\Theta(1)$ is best possible in **(iii)**.
- We have no reason to believe that $\gamma = 1/4$ and $\beta = 27$ are "best possible" or even "good". Note that for $\ell = 4\log n$ the bound on $\bar{c}$ becomes $\bar{c} \geq \beta \log(n)/n = \Theta(\frac{\log n}{n})$.
- We conjecture that for each $\ell \geq 2$ there is a threshold value $c^*_\ell \in (0,1)$ such that for $c < c^*_\ell$ the matrix $A^{\ell*}_{cn,n}$ has independent rows with probability at least $1/2$ and for $c > c^*_\ell$ it has dependent rows whp.

3.1 Proof of Proposition 3 (i)

Recall from Section 2 how $A = A^\ell_{cn,n}$ is obtained from S via a random hash function $h = (b_1, b_2, p_1, p_2)$ mapping elements to rows. If A does not have independent rows, then this is *witnessed* by a non-trivial subset $W \subseteq S$ of elements such that the corresponding rows of A sum to zero. We use a first moment calculation to show that whp no *inclusion-minimal* witness Y exists. We fix two parameters of candidate sets Y: The number $s = |W|$ of elements/rows, with $1 \leq s \leq m = cn$, and the number $t = |B| \in [n/\ell]$, where $B = \bigcup_{w\in W}\{b_1(w), b_2(w)\}$ is the set of variable blocks involved in at least one of the rows.

There are $\binom{m}{s}$ ways to choose Y, and $\binom{n/\ell}{t}$ ways to choose B. The probability that the rows corresponding to Y involve exactly the blocks from B is

$$\Pr[B = \bigcup_{w\in W}\{b_1(w), b_2(w)\}] \leq \prod_{w\in W} \Pr[b_1(w) \in B \wedge b_2(w) \in B] = (\tfrac{t}{n/\ell})^{2s}.$$

The event that the rows corresponding to W sum to zero is the intersection of the independent events that the rows sum to zero within each block $b \in B$. Its probability is therefore

$$\prod_{b\in B} \Pr\Big[\bigoplus_{\substack{(w,i)\in W\times\{0,1\}\\ b_i(w)=b}} p_i(w) = 0^\ell \;\Big|\; \exists (w,i) \in W \times \{0,1\} : b_i(w) = b\Big] = \prod_{b\in B} 2^{-\ell} = 2^{-\ell t}.$$

In the following, it is often convenient to deal with the fraction $\sigma = s/m$ of rows and the fraction $\tau = t/(n/\ell) = \ell t/n$ of blocks involved in a witness. Accordingly, we define $O(n^2/\ell)$ values $p_{\sigma,\tau}$, where $p_{\sigma,\tau}$ is the probability that some set of equations involving σm rows and exactly $\tau n/\ell$ blocks is a minimal witness. This gives

$$p_{\sigma,\tau} \leq \binom{m}{s}\binom{n/\ell}{t}(\tfrac{t}{n/\ell})^{2s}2^{-\ell t} = \binom{m}{\sigma m}\binom{n/\ell}{\tau n/\ell}\tau^{2\sigma m}2^{-\tau n}. \tag{3}$$

We now list a few bounds that will be useful later. Throughout, $\sigma \in \{\frac{1}{m}, \ldots, \frac{m}{m} = 1\}$ and $\tau \in \{\frac{1}{n/\ell}, \ldots, \frac{n/\ell}{n/\ell} = 1\}$.

▶ **Lemma 4.** *Let $\bar{\tau} = 1 - \tau$, $\bar{c} = 1 - c$, and let H be the binary entropy function. Then*

(a) $\frac{\ell}{n}\log(p_{\sigma,\tau}) \leq c\ell H(\sigma) + H(\tau) + 2\sigma c\ell \log\tau - \ell\tau = c\ell(H(\sigma) + \sigma\log\tau^2) + H(\tau) - \ell\tau.$

(b) $\frac{\ell}{n}\log(p_{\sigma,\tau}) \leq \ell(c\log(1+\tau^2) - \tau) + H(\tau).$

(c) $-\log\bar{c} \leq \ell/4$ (c1) $\qquad c \geq \frac{3}{4}$ (c2) $\qquad \bar{c} \geq 27\log(n)/n$ (c3)

(d) *All minimal witnesses satisfy $t \leq s + 1$.*

[6] An all-zero row requires an element x with hash value $h(x)$ fulfilling $b_1(x) = b_2(x)$ and $p_1(x) = p_2(x)$.

(e) $\log(1+\tau^2) \le \begin{cases} \tau \cdot 2\log\frac{5}{4} \le \frac{2}{3}\tau & \text{if } 0 < \tau \le \frac{1}{2}, \\ 1 - 2\bar{\tau} \cdot (1 - \log\frac{5}{4}) \le 1 - \frac{4}{3}\bar{\tau} & \text{if } \frac{1}{2} \le \tau \le 1, \\ \tau & \text{if } 0 < \tau \le 1. \end{cases}$

(f) $-\log\tau \le 2\bar{\tau}$ *if* $\frac{1}{2} \le \tau \le 1$.

(g) $-\tau_1 \log\tau_1 \le -\tau_2 \log\tau_2$ *for* $0 < \tau_1 < \tau_2 < \frac{1}{4}$.

(h) $H(\tau) \le -\tau\log\tau + 2\tau$ *if* $\tau \le \frac{1}{2}$.

(i) $H(\tau_1) < H(\tau_2)$ *for* $0 < \tau_1 < \tau_2 \le \frac{1}{2}$ *and* $H(\tau) = H(\bar{\tau})$ *for* $0 < \tau \le 1$.

(j) *If* $s \ge t$ *and* $\tau < 1/\ell$ *then* $\frac{\ell}{n}\log(p_{\sigma,\tau}) \le -\tau\ell/2$.

The claims of Lemma 4 can be verified with simple calculations, found in Section 3.3.

Different arguments will be used to get bounds on $p_{\sigma,\tau}$ for different ranges of τ. The sum of all $p_{\sigma,\tau}$ belonging to the same case will be $n^{-\varepsilon}$ for some $\varepsilon > 0$, which implies that A has full rank whp. In the proofs, we refer to parts of Lemma 4 by their labels.

Case 1: $c \le \tau \le 1$.

$$\frac{\ell}{n}\log(p_{\sigma,\tau}) \overset{\text{(b,e)}}{\le} \ell(c\tau - \tau) + H(\tau) \overset{\text{(i)}}{\le} -\ell\tau\bar{c} + H(\bar{c}) \overset{\text{(h)}}{\le} -\ell\tau\bar{c} - \bar{c}\log\bar{c} + 2\bar{c}$$
$$\overset{\text{(c1)}}{\le} -\ell c\bar{c} + \bar{c}\ell/4 + 2\bar{c} = -\bar{c}(\ell c - \ell/4 - 2) \overset{\text{(c2)}}{\le} -\bar{c}\ell/2.$$

This gives a bound of:

$$p_{\sigma,\tau} \le 2^{-\bar{c}n/2} \overset{\text{(c3)}}{\le} n^{-27/2}.$$

Multiplying with $O(n^2)$ choices for σ and τ, this still gives a bound of $O(n^{-23/2})$.

Case 2: $1/2 \le \tau \le c$.

$$\frac{\ell}{n}\log(p_{\sigma,\tau}) \overset{\text{(b,e)}}{\le} \ell(c(1-\tfrac{4}{3}\bar{\tau}) - \tau) + H(\tau) \overset{\text{(h,i)}}{\le} \ell(\bar{\tau} - \tfrac{4}{3}\bar{\tau}) + 2\bar{\tau} - \bar{\tau}\log\bar{\tau}$$
$$\le \bar{\tau}(-\ell/3 + 2 - \log\bar{c}) \overset{\text{(c1)}}{\le} \bar{\tau}(-\ell/3 + 2 + \ell/4) \le -\bar{c}\ell/13.$$

From this we obtain a bound $p_{\sigma,\tau} = n^{-27/13}$ and proceed as in Case 1.

Case 3: $\bar{c} \le \tau \le 1/2$.

$$\frac{\ell}{n}\log(p_{\sigma,\tau}) \overset{\text{(b,e,f)}}{\le} \ell(\tfrac{2}{3}\tau - \tau) + H(\tau) \overset{\text{(h)}}{\le} -\tau\ell/3 - \tau\log\tau + 2\tau$$
$$\le \tau(-\ell/3 - \log\bar{c} + 2) \overset{\text{(c1)}}{\le} \tau(-\ell/3 + \ell/4 + 2) \le -\tau\ell/13 \le -\bar{c}\ell/13.$$

This is the same bound as in Case 2.

Case 4: $8\log(n)/n < \tau < 1/\ell$. Assuming $s \ge t$ for the moment, we may apply (j) to obtain $\frac{\ell}{n}\log(p_{\sigma,\tau}) \le -\tau\ell/2$. This gives $p_{\sigma,\tau} = 2^{-\tau n/2} \le 2^{-4\log n} \le n^{-4}$.
Inconveniently, (d) only gives $s \ge t-1$ instead of $s \ge t$ and the $O(n)$ cases with $s = t-1$ are not yet handled. Luckily, changing s by 1 (or equivalently σ by $1/m$) affects the upper bound in Equation (3) by at most $O(n^2)$ and the combined contribution of the cases in question is bounded by $O(n) \cdot O(n^2) \cdot n^{-4} = O(n^{-1})$.

Case 5: $2 \le t$ and $\ell^2 t^2 \le \frac{n}{2\mathrm{e}}$. We refine Equation (3) to get (recall $\sigma = \frac{s}{m}$, $\tau = \frac{\ell t}{n}$)

$$\begin{aligned} p_{\sigma,\tau} &\le \binom{m}{s}\binom{n/\ell}{t}\left(\tfrac{t}{n/\ell}\right)^{2s} 2^{-\ell t} \le \left(\frac{m\mathrm{e}}{s}\right)^s \left(\frac{en/\ell}{t}\right)^t \left(\frac{t}{n/\ell}\right)^{2s} 2^{-\ell t} \\ &= \left(\frac{\mathrm{ce}\ell^2 t^2}{sn}\right)^s \left(\frac{\mathrm{en}}{t\ell 2^\ell}\right)^t = \left(\frac{\mathrm{ce}\ell^2 t^2}{sn}\right)^{1+(s-t+1)+(t-2)} \left(\frac{\mathrm{en}}{t\ell 2^\ell}\right)^{2+(t-2)} \\ &= \left(\frac{\mathrm{ce}\ell^2 t^2}{sn}\right)\left(\frac{\mathrm{en}}{t\ell 2^\ell}\right)^2 \left(\frac{\mathrm{ce}^2\ell t}{s2^\ell}\right)^{t-2} \left(\frac{\mathrm{ce}\ell^2 t^2}{sn}\right)^{s-t+1} \\ &\le \frac{\mathrm{ce}^3 n}{2^{2\ell}}(\tfrac{1}{2})^{t-2}(\tfrac{1}{2})^{s-t+1} = \frac{\mathrm{ce}^3 n}{2^{2\ell}}(\tfrac{1}{2})^{s-1} \end{aligned}$$

where the last inequality used $\ell/2^\ell \le c\mathrm{e}^2/4$ (which holds for n sufficiently large), the upper bound on ℓt and $t/s \le 2$. It is crucial that the exponents $t-2$ and $s-t+1$ are nonnegative; for the latter exponent this is because of (d). The sum over all applicable s and t is dominated by the contribution for $t=2$ and $s=1$, since

$$\sum_{s\ge 1}\sum_{2\le t\le s+1} p_{\sigma,\tau} \le \sum_{s\ge 1}\sum_{2\le t\le s+1} \frac{\mathrm{ce}^3 n}{2^{2\ell}}(\tfrac{1}{2})^{s-1} = \frac{\mathrm{ce}^3 n}{2^{2\ell}}\sum_{s\ge 1} s\cdot(\tfrac{1}{2})^{s-1} = \frac{4\mathrm{ce}^3 n}{2^{2\ell}}.$$

Finally, using the assumption $2\ell \ge (1+\delta)\log n$, and thus $2^{2\ell} \ge n^{1+\delta}$, we obtain

$$\frac{4ce^3 n}{2^{2\ell}} = 4ce^3 n^{-\delta} = O(n^{-\delta}).$$

Case 5': $1 = t$ and $\ell \le n^{1/4}$. The argument from Case 5 essentially works, but the trivial bound $s \ge t-1 = 0$ needs to be replaced with $s \ge 1$. We get

$$\begin{aligned} \sum_{s\ge 1} p_{\sigma,\tau} &\le \sum_{s\ge 1}\binom{m}{s}\tfrac{n}{\ell}\left(\tfrac{\ell}{n}\right)^{2s} 2^{-\ell} \le \sum_{s\ge 1}\tfrac{n}{\ell\cdot 2^\ell}\left(\frac{m\ell^2}{sn^2}\right)^s \\ &= \tfrac{\mathrm{ce}\ell}{2^\ell}\sum_{s\ge 1}\left(\frac{\mathrm{ce}\ell^2}{sn}\right)^{s-1} \le O(n^{-1/2})\sum_{s\ge 1}\left(n^{-1/2}\right)^{s-1} = O(n^{-1/2}). \end{aligned}$$

Finally, we need to check that the case distinction is complete. If $\ell = \omega(\log n)$ then $t=1$ corresponds to $\tau = \ell/n \ge 27\log(n)/n = \bar{c}$ (using (c)) and Cases 1–3 already cover the entire range of τ. For more interesting values of $\ell = O(\log n)$ Cases 3 and 4 overlap due to (c) and Cases 4 and 5 overlap since the upper bound $\ell^2 t^2 \le \frac{n}{2\mathrm{e}}$ corresponds to $\tau = \ell t/n = O(n^{-1/2})$.

3.2 Adjustments for Proposition 3 (ii) and (iii)

We first outline how the argument from **(i)** needs to be modified to prove **(ii)**. Firstly, the probability that a sum within a block $b \in B$ is $\vec{0}$ is no longer $2^{-\ell}$. Assuming $\deg(b) = k$ and incidences labelled with uniformly random values $p_1, \ldots, p_k \in \{0,1\}^k - \{\vec{0}\}$ it is

$$\begin{aligned} &\Pr[p_1 \oplus \ldots \oplus p_k = \vec{0}] \\ =&\Pr[p_k = p_1 \oplus \ldots \oplus p_{k-1} \mid p_1 \oplus \ldots \oplus p_{k-1} \neq \vec{0}] \cdot \Pr[p_1 \oplus \ldots \oplus p_{k-1} \neq \vec{0}] \\ \le&\Pr[p_k = p_1 \oplus \ldots \oplus p_{k-1} \mid p_1 \oplus \ldots \oplus p_{k-1} \neq \vec{0}] = 1/(2^\ell - 1). \end{aligned}$$

The additive difference to the bound on $\frac{\ell}{n}\log(p_{\sigma,\tau})$ in Lemma 4(a) is

$$\tfrac{\ell}{n}\log((2^\ell/(2^\ell-1))^{\tau n/\ell}) = \tau\log(1+1/(2^\ell-1)) < 2t/(2^\ell-1) < 4\tau 2^{-\ell} \le 4\tau\bar{c}^4.$$

This is of lower order than the upper bound required in Cases 1 to 4 and thus inconsequential.

The only case where work is needed is Case 5 where the bound $\ell \geq (1+\delta)\log n$ is not available. Since the all-zero vector is forbidden as a coefficient vector for blocks of an equation, we know that minimal witnesses are not only connected but have minimum degree 2. The most extreme cases are then not trees with $s = t-1$, but cycles with $s = t$. The dominating term is then upper bounded by $\frac{c^2 e^4 \ell^2}{(2^\ell-1)^2}$, which is $o(1)$ because $\ell = \omega(1)$.

For **(iii)** we argue as in **(ii)**, except that when ℓ is constant the dominating term $\frac{c^2 e^4 \ell^2}{(2^\ell-1)^2}$ of Case 5 does not vanish for $n \to \infty$. However, if ℓ is large enough then the sum is less than 1, yielding a constant probability that no Case-5-type witness exists. Witnesses of other types have vanishing probability as before.

3.3 Proof of Lemma 4

Proof.

(a) This follows from Equation (3) after taking logarithms and multiplying by ℓ/n on both sides, using the standard approximation $\log \binom{n}{k} \leq nH(\frac{k}{n})$.

(b) This is obtained from (a) by observing that $H(\sigma) + \sigma \log \tau^2$ is concave as a function of σ and assumes its unique maximum value at $\sigma^* = \sigma^*(\tau) = \frac{\tau^2}{1+\tau^2}$.

(c) This is part of the assumption of Proposition 3**(i)**.

(d) If $W \subseteq S$, viewed as a subgraph of $G = G^\ell_{cn,n}$ (see Section 2), has two connected components $W = W_1 \cup W_2$, then the rows corresponding to W sum to zero if and only if the rows corresponding to W_1 and W_2 sum to zero individually, as they involve disjoint sets of variable blocks. In that case, W is not a minimal witness for dependence. In other words, we can restrict our attention to *connected* sets W. From this $t \leq s+1$ follows, the upper bound being attained if W corresponds to a tree in G.

(e) Since $g(\tau) = \log(1+\tau^2)$ is convex on $[0,1]$, we may the obtain upper bounds on g by linearly interpolating between the values $g(0) = 0$, $g(\frac{1}{2}) = \log \frac{5}{4}$ and $g(1) = 1$.

(f) Using that $g(\tau) = -\log \tau$ is convex we may obtain bounds on g by linearly interpolating between the values $g(1/2) = 1$, $g(1) = 0$.

(g) The function $g(\tau) = -\tau \log \tau$ is clearly continuous and its unique maximum is easily determined to be at $\tau = 1/\mathrm{e} > 1/4$, which implies the claim.

(h) $H(\tau) = -\tau \log \tau - \bar{\tau} \log \bar{\tau} \leq -\tau \log \tau - \log \bar{\tau} \overset{\text{(f)}}{\leq} -\tau \log \tau + 2\tau$.

(i) These properties of the entropy function are well known and easily checked.

(j) From $\sigma cn = s \geq t = \tau n/\ell$ we get $\sigma \geq \frac{\tau}{c\ell}$. Using the upper bound on τ we continue with $\sigma \geq \frac{\tau^2}{c} \geq \frac{\tau^2}{1+\tau^2}$. This means that all values permitted for σ exceed the argument $\sigma^* = \frac{\tau^2}{1+\tau^2}$ from (b) that maximises $H(\sigma) + \sigma \log \tau^2$. Again by concavity of this function we may refine the upper bound from (a) by substituting the smallest admissible value $\sigma = \frac{\tau}{c\ell}$. This yields:

$$\begin{aligned}\tfrac{\ell}{n} \log(p_{\sigma,\tau}) &\leq c\ell H(\tfrac{\tau}{c\ell}) + 2\tau \log \tau + H(\tau) - \ell\tau \overset{\text{(h)}}{\leq} -\tau \log(\tfrac{\tau}{c\ell}) + 2\tau + \tau \log \tau + 2\tau - \ell\tau \\ &= \tau \log(c\ell) + 4\tau - \ell\tau = \tau(\log(c\ell) - \ell + 4) \leq -\tau\ell/2.\end{aligned}$$

◀

4 Proof of the Main Theorem

With Proposition 3 in place, we can now prove Theorem 1.

Proof of Theorem 1.

(i) Aiming to apply Proposition 3**(i)**, we pick $\ell := 4\lceil \log m \rceil$, $\bar{c}' := \frac{27 \log m}{m}$, $c' := 1 - \bar{c}'$ and n as the least multiple of ℓ exceeding m/c'. We generate the matrix $A = A^{\ell}_{m,n}$ as defined in Section 2. For $c := m/n$ we can derive that by construction $\bar{c} = 1 - c = 1 - m/n \geq 1 - c' = \bar{c}' = \frac{27 \log m}{m}$ and then clearly $\bar{c} \geq 2^{-\ell/4} = O(m^{-1})$ holds as well. Thus Proposition 3**(i)** implies that A has full rank whp. Assume $r = 1$ for now. Solving a corresponding system $A\vec{z} = \vec{b}$ yields a retrieval data structure occupying n bits. The overhead is $O(\frac{\log m}{m})$ since

$$\frac{n}{m} - 1 \leq \frac{m/c' + \ell}{m} - 1 = O(\tfrac{\log m}{m}) + \tfrac{1-c'}{c'} \leq O(\tfrac{\log m}{m}) + \bar{c}' = O(\tfrac{\log m}{m}).$$

Construction time is dominated by the time to solve the linear system. We employ the Method of Four Russians [3] – a variant of Gaussian elimination – which requires $O(m^2/\log m)$ row additions. As rows contain $n + 1 = O(m)$ bits and w bits can be added in one word operation, we obtain a total runtime of $O(\frac{m^3}{w \log m})$. Queries access $\lceil \frac{\ell}{w} \rceil = O(1)$ memory words in two contiguous areas of memory, and require $O(1)$ bit-wise and operations as well as a parity operation. Query times are thus $O(1)$.
If $r > 1$ we need to solve $A \cdot X = B$ with $B \in \mathbb{F}_2^{m \times r}$ for $X \in \mathbb{F}_2^{n \times r}$. Solving the linear system for several right hand sides simultaneously comes at negligible additional cost. For cache efficient queries, blocks of X of size $\ell \times r$ should be stored contiguously, and each block should be stored column-wise.

(ii) We use Wiedemann's algorithm [24] to solve the system $A\vec{z} = \vec{b}$. As the algorithm only works for regular matrices we must first append $n - m$ rows to the full-rank matrix $A \in \mathbb{F}_2^{m \times n}$ such that the resulting square matrix $A' \in \mathbb{F}_2^{n \times n}$ is regular. It is well known that when picking rows uniformly from $\mathbb{F}_2^n$ this succeeds with probability $\Theta(1)$. We also append $n - m$ zeroes to b to obtain b'. The running time for solving the regular system $A'\vec{z} = \vec{b'}$ with Wiedemann's algorithm is dominated by the running time of $O(n)$ matrix-vector multplications involving A'. Note that multiplications with A can be carried out in time $O(m)$ using word operations. The additional rows of A' increase this by $O((m-n)n/w) = O(\frac{m \log m}{w})$, which is also $O(m)$ if $w = \Omega(\log m)$. The total runtime for $r = 1$ is thus $O(m^2)$. For $r > 1$, the algorithm must be repeated for each bit. ◀

In Section 5 we demonstrate that the approach in **(i)** admits a particularly efficient implementation in practice.

5 Experiments and Practical Considerations

5.1 Experimental Overhead

The benchmark uses the universe $\mathcal{U} = \text{ASCII}^*$ of strings, $S \subset \mathcal{U}$ is taken as the first $m = 10^7$ URLs from a `eu-2015-host` dataset gathered by [5] with ≈80 bytes per key, and for simplicity $f : \mathcal{U} \to \{0,1\}$ is taken to be the parity of the string length[7].

For hashing, we use $\mathsf{murmur} = \texttt{MurmurHash3\_x64\_128} : (\{0,1\}^8)^* \times \{0,1\}^{32} \to \{0,1\}^{128}$ [1], which conveniently has a second parameter. We use $(\mathsf{murmur}(\cdot, s))_{s \in \{0,1\}^{32}}$ as though it were a sequence of random independent hash functions. A hash function on $\mathcal{U}$ can thus

[7] Since the sequence of operations performed by the algorithm does not depend on f except, possibly, in the rare cases where singular linear systems are involved, the choice of f is largely inconsequential.

Table 3 Overview of all bits used in the data structure. The concrete values on the right correspond to a run on a data set with $m = 10^7$ keys, chunk size $C = 10^4$, $\ell = 16$ and $\varepsilon = 0.0005$. In that run $\lceil \log(1 + \max_i s_i) \rceil = 2$ and $\lceil \log(1 + \max_i d_i) \rceil = 9$.

Number of bits	bits used for	per element
m	entropy lower bound	1.000000
εm	intended inner overhead	0.000500
$\sum_i n_i - (1+\varepsilon)m$	padding ensuring $\ell \mid n_i$	0.000716
$\lceil \log(1 + \max_i s_i) \rceil \cdot m/C$	seed for each chunk	0.000200
$\lceil \log(1 + \max_i d_i) \rceil \cdot m/C$	offset info for each chunk	0.000900
[not discussed]	various global counters	0.000062
	all of the above	1.002378

be identified simply by s, the *seed*. One such hash function $h_0 : \mathcal{U} \to [\lceil m/C \rceil]$ partitions S into chunks $S_i = \{x \in S \mid h_0(x) = i\}$ for $0 \le i < \lceil m/C \rceil$ where the average chunk size was chosen as $C = 10^4$. The actual chunk sizes $m_i = |S_i|$ vary slightly around C.

For each i let n_i be the least multiple of the block size $\ell = 16$ that is at least $(1+\varepsilon)m_i$. Here, $\varepsilon = 0.0005$ is the intended inner overhead. Note that $n_i - (1+\varepsilon)m_i$ has an expectation of roughly $\frac{\ell-1}{2} = 7.5$. Within each chunk we generate and solve a system $A^{\ell}_{m_i,n_i}\vec{z_i} = \vec{b_i}$ yielding $\vec{z_i} \in \{0,1\}^{n_i}$. Construction is repeated with a new seed if necessary. Let s_i be the seed of the first successful construction for chunk i.

The vectors $\vec{z_i}$ are concatenated into one bit string $\vec{z}$. Let $o_i = \sum_{j<i} n_i/\ell$ be the offset (counted in blocks) where z_i starts within z. We store the values $d_i = o_i - \lfloor \frac{i-1}{\lceil m/C \rceil} |\vec{z}| \rfloor \approx o_i - \mathbb{E}[o_i]$ instead of the values o_i as their binary representation is typically only half as long.

Finally, let $\hat{d} := \max_i d_i$ and $\hat{s} := \max_i s_i$. In addition to $\vec{x}$, m, C, $\hat{d}$ and $\hat{s}$ we need to store the meta data $((s_i, d_i))_i$ for the chunks using $(\lceil \log(\hat{d}+1) \rceil + \lceil \log(\hat{s}+1) \rceil) \lceil m/C \rceil$ bits. A full account of everything that needs to be saved with concrete numbers is given in Table 3.

5.2 Experimental Runtimes

All tests were performed on an desktop computer with an Intel® Core i7-2600 Processor @ 3.40GHz. A direct comparison to results from [15] is given in Table 2.[8]

Construction. To solve the sparse linear system $A\vec{z} = \vec{b}$ in a chunk, we first employ a heuristic that reduces the system to a system $A'\vec{z}' = \vec{b}'$ that is dense but substantially smaller than A. We dub this step *BlockedLazyGauss* as it is heavily inspired by the LazyGauss algorithm from [15]. In our case of $\ell = 16$ only 15% of the variables from A remain in A'. The reduced system is then solved using the Method of Four Russians.

To highlight the influence of the chunk size C, we now consider construction time *per key*, which is $O(\frac{C^2}{w \log C})$. In Figure 2 we report the runtimes of our solver for $A^{\ell}_{m,n}\vec{z} = \vec{b}$, as well as the relative contribution of the LazyGauss and the Four-Russian phases. The time per key of ≈1.8µs reported there for $C = 10^4$ is also the main component of the time per key of ≈2.6µs for the complete construction algorithm. The additional time is mostly spent on

[8] Note that the implementations may be optimised to different degrees, and despite the fact that very similar CPUs were used, runtime comparisons should not be overinterpreted. The authors of [15] estimate a "tight C implementation [of their algorithm] would be about twice as fast".

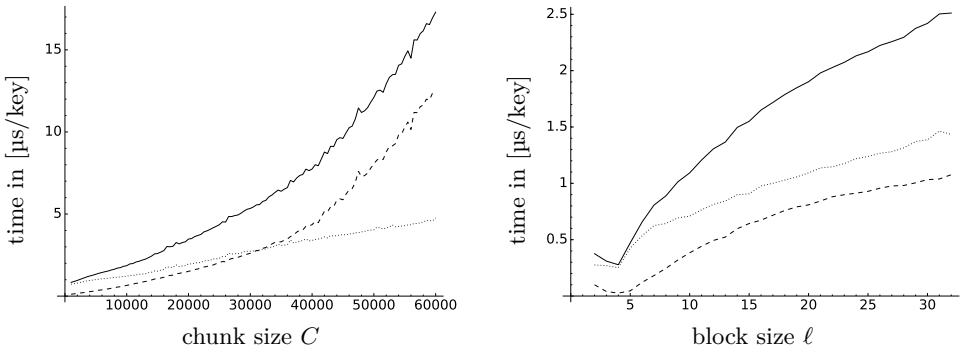

Figure 2 Time per key of our linear system solver with ⋰ representing the time for the BlockedLazyGauss-phase, ⋱ the time for the FourRussian-phase and ╱ the sum. On the left the block size is $\ell = 16$ and the chunk size C varies. On the right $C = 10^4$ and ℓ varies. The number of equations per chunk is $0.9995C$ and C, respectively.

streaming the key from a zipped file (≈0.3µs), hashing it, sorting it into the correct chunk as well as allocating and initialising the linear systems. Only a fraction of ≈0.005 of the linear systems fail to have full rank and require a restart for the chunk.

The work on the 10^3 chunks can be parallelised in a straightforward way, which brings construction time down to 1.1µs per key, using 4 cores with 2 logical processors each.

Query. A query involves computing hash values, accessing two ℓ-bit words, and very cheap and, xor and parity operations. In our experiments, computing hash values took ≈35ns. Overall query time was ≈75ns for $m = 10^6$ (ℓ, ε, C as above), when the retrieval data structure could reasonably be expected to reside in cache. Time increased to ≈125ns for $m = 10^8$, where the retrieval data structure certainly did not fit into cache.

6 Conclusion and Future Work

We introduced a new variant of constructing a retrieval data structure for m elements and range $\{0,1\}^r$ on the basis of the classical method of transforming keys into the rows of a linear system of equations over $\mathbb{F}_2$, and using the solution vector as the data structure. The new idea of having $O(\log m)$ many 1 entries in a row, concentrated in two blocks, in combination with word parallelism, gives constant query time on a word RAM. The construction time can be reduced by both exact and heuristic methods so as to achieve space $(1 + O((\log m)/m))mr = mr + O(\log m)r$ in theory and $1.0024m$ in realistic experiments with $r = 1$.[9] Future work could examine:

- Is it possible to achieve constant access time and additive overhead $O(\log \log n)$ by a variant of our construction using a square system? (This would be the case if such a systems had full rank with constant probability.)
- Study the behaviour of systems of equations as considered here for fields $\mathbb{F}_q$ of constant size $q > 2$.

[9] Table 3 suggests that we can obtain $\approx 1.0012mr + 0.0012m$ for general r.

References

1 Austin Appleby. MurmurHash3, 2012. URL: https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp.

2 Martin Aumüller, Martin Dietzfelbinger, and Michael Rink. Experimental Variations of a Theoretically Good Retrieval Data Structure. In *Proc. 17th ESA*, pages 742–751, 2009. doi:10.1007/978-3-642-04128-0_66.

3 Gregory V. Bard. *Algebraic Cryptanalysis*, chapter The Method of Four Russians, pages 133–158. Springer US, Boston, MA, 2009. doi:10.1007/978-0-387-88757-9_9.

4 Djamal Belazzougui, Fabiano Cupertino Botelho, and Martin Dietzfelbinger. Hash, Displace, and Compress. In *Proc. 17th ESA*, pages 682–693, 2009. doi:10.1007/978-3-642-04128-0_61.

5 Paolo Boldi, Andrea Marino, Massimo Santini, and Sebastiano Vigna. BUbiNG: Massive crawling for the masses. In *Proc. 23rd WWW'14*, pages 227–228, 2014. doi:10.1145/2567948.2577304.

6 Fabiano Cupertino Botelho, Rasmus Pagh, and Nivio Ziviani. Simple and Space-Efficient Minimal Perfect Hash Functions. In *Proc. 10th WADS*, pages 139–150, 2007. doi:10.1007/978-3-540-73951-7_13.

7 Fabiano Cupertino Botelho, Rasmus Pagh, and Nivio Ziviani. Practical Perfect Hashing in Nearly Optimal Space. *Inf. Syst.*, pages 108–131, 2013. doi:10.1016/j.is.2012.06.002.

8 Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables. In *Proc. 15th SODA*, pages 30–39, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982797.

9 Colin Cooper. On the rank of random matrices. *Random Structures & Algorithms*, 16(2):209–232, 2000. doi:10.1002/(SICI)1098-2418(200003)16:2<209::AID-RSA6>3.0.CO;2-1.

10 Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight Thresholds for Cuckoo Hashing via XORSAT. In *Proc. 37th ICALP (1)*, pages 213–225, 2010. doi:10.1007/978-3-642-14165-2_19.

11 Martin Dietzfelbinger and Rasmus Pagh. Succinct Data Structures for Retrieval and Approximate Membership (Extended Abstract). In *Proc. 35th ICALP (1)*, pages 385–396, 2008. doi:10.1007/978-3-540-70575-8_32.

12 Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci.*, 380(1-2):47–68, 2007. doi:10.1016/j.tcs.2007.02.054.

13 Nikolaos Fountoulakis and Konstantinos Panagiotou. Sharp Load Thresholds for Cuckoo Hashing. *Random Struct. Algorithms*, 41(3):306–333, 2012. doi:10.1002/rsa.20426.

14 Alan M. Frieze and Páll Melsted. Maximum Matchings in Random Bipartite Graphs and the Space Utilization of Cuckoo Hash Tables. *Random Struct. Algorithms*, 41(3):334–364, 2012. doi:10.1002/rsa.20427.

15 Marco Genuzio, Giuseppe Ottaviano, and Sebastiano Vigna. Fast Scalable Construction of (Minimal Perfect Hash) Functions. In *Proc. 15th SEA*, pages 339–352, 2016. doi:10.1007/978-3-319-38851-9_23.

16 Torben Hagerup and Torsten Tholey. Efficient minimal perfect hashing in nearly minimal space. In *Proc. 18st STACS*, pages 317–326, 2001. doi:10.1007/3-540-44693-1_28.

17 Brian A. LaMacchia and Andrew M. Odlyzko. Solving large sparse linear systems over finite fields. In *CRYPTO '90, 10th Annual International Cryptology Conference*, pages 109–133, 1990. doi:10.1007/3-540-38424-3_8.

18 Bohdan S. Majewski, Nicholas C. Wormald, George Havas, and Zbigniew J. Czech. A Family of Perfect Hashing Methods. *Comput. J.*, pages 547–554, 1996. doi:10.1093/comjnl/39.6.547.

19 Michael Molloy. The pure literal rule threshold and cores in random hypergraphs. In *Proc. 15th SODA*, pages 672–681, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982896.

20 Boris Pittel and Gregory B. Sorkin. The Satisfiability Threshold for k-XORSAT. *Combinatorics, Probability & Computing*, 25(2):236–268, 2016. doi:10.1017/S0963548315000097.

21 Ely Porat. An Optimal Bloom Filter Replacement Based on Matrix Solving. In *Proc. 4th CSR*, pages 263–273, 2009. doi:10.1007/978-3-642-03351-3_25.

22 Michael Rink. Mixed Hypergraphs for Linear-Time Construction of Denser Hashing-Based Data Structures. In *Proc. 39th SOFSEM*, pages 356–368, 2013. doi:10.1007/978-3-642-35843-2_31.

23 Steven S. Seiden and Daniel S. Hirschberg. Finding succinct ordered minimal perfect hash functions. *Inf. Process. Lett.*, pages 283–288, 1994. doi:10.1016/0020-0190(94)00108-1.

24 Douglas H. Wiedemann. Solving Sparse Linear Equations Over Finite Fields. *IEEE Transactions on Information Theory*, pages 54–62, 1986. doi:10.1109/TIT.1986.1057137.