One Type Fits All: A General Analysis of Representation Trees for Subset Sum, LWE and Decoding

Andre Esser

Technology Innovation Institute, Abu Dhabi, UAE
andre.esser@tii.ae

Abstract. The representation technique is the basis of the best known algorithms for random subset sum, of combinatorial attacks on small LWE keys and of information set decoding. It builds the solution in a search tree, the representation tree, whose analysis has so far required a combinatorial derivation by hand for every variant.
We introduce a general analysis technique for any representation tree, which replaces this derivation by a few short entropy formulas and turns the whole analysis into one explicit optimization problem, so that a new variant needs no new analysis. This technique alone improves the asymptotic exponents for almost all applications we consider. We then extend this analysis by adding filters, known from nearest-neighbor search, to every level of the best known trees and optimize them jointly with all other parameters. Finally, we improve the filtering technique by partitioning the enumerated vectors into a sparse and a dense part, which lets the filters act in a more concentrated way, with significant gains especially for subset sum and LWE. Further, we reduce the number of optimization variables for the filters from quadratic to linear, which makes optimizations feasible that were previously out of reach.
Heuristically, random subset sum is solved in time $T = 2^{0.2620n}$ and memory $M = 2^{0.2120n}$ instead of $T = M = 2^{0.2830n}$, and LWE keys following the distribution used in Dilithium are recovered in time $2^{0.2954n}$ instead of $2^{0.378n}$. We also obtain the fastest combinatorial algorithms for ternary LWE keys and LWE keys as in Kyber, the fastest algorithms for decoding over $\mathbb{F}_q$ for small $q \geq 3$, ternary decoding with large weight, decoding with restricted errors and Lee metric decoding. Further, we improve on the best known algorithms for nearest-neighbor search over $\mathbb{F}_q$ and on the best known time-memory trade-offs for subset sum.

Keywords: Representation technique · Subset sum · LWE · Information set decoding · Nearest-neighbor search · Time-memory trade-off

1 Introduction

The representation technique, introduced by Howgrave-Graham and Joux [33], is one of the major techniques of combinatorial cryptanalysis. It is the basis of the fastest known algorithms for random subset sum [8,11], of the fastest known

information set decoding algorithms over $\mathbb{F}_2$ [39,10,40,14] and for further settings such as decoding over $\mathbb{F}_q$, $q \geq 3$ [41], ternary decoding with large weight [15,43], decoding with restricted errors [3] and decoding in the Lee metric [50]. With the Meet-LWE trees [38,29,27] it also gives the best known combinatorial attacks on LWE keys with small coordinates, as used in the standardized schemes Kyber (ML-KEM) [12] and Dilithium (ML-DSA) [23]. Let us take subset sum as a running example: given random integers $a_1, \ldots, a_n$ modulo 2^n and a target t, find $\mathbf{e} \in \{0,1\}^n$ with $\sum_i a_i e_i = t \bmod 2^n$. In the other problems vectors replace the a_i, and for LWE the equation holds up to a small error.

The technique writes the solution as a sum $\mathbf{e} = \mathbf{x} + \mathbf{y}$ of two vectors. The ones of $\mathbf{e}$ can be distributed over $\mathbf{x}$ and $\mathbf{y}$ in exponentially many ways, the *representations* of $\mathbf{e}$, while it suffices to find one of them. The algorithm therefore introduces constraints on the specific representation it finds. Namely, it fixes a random value for some bits of the *label* $\phi(\mathbf{x}) := \sum_i a_i x_i \bmod 2^n$, letting $\phi(\mathbf{y}) = t - \phi(\mathbf{x}) \bmod 2^n$ such that in expectation about one representation satisfies those restrictions. Candidates for $\mathbf{x}$ and $\mathbf{y}$ are then built recursively from sparser vectors in a tree-based fashion. Two lists are *joined* by forming all pairs $(\mathbf{x}, \mathbf{y})$ between them whose labels match on further bits, and keeping $\mathbf{x}+\mathbf{y}$ only if it contains every *digit*, that is, every value of an entry, as often as its level dictates. For the top level of the subset sum tree this latter restriction requires that $\mathbf{x}+\mathbf{y}$ contains $n/2$ zeros and $n/2$ ones. Lower levels may contain further digits, to increase the number of representations. Becker, Coron and Joux [8] add the digit -1, and with the further digit 2 Bonnetain et al. [11] obtain the running time $2^{\tau n}$ with the *exponent* $\tau = 0.2830$.

Every such algorithm comes with an optimization problem. Its exponent is a non-convex function of many parameters, such as the frequency of each digit and the number of label bits on every level, and has to be minimized numerically [11]. To set up this function, the analysis of a join needs two numbers, the number of representations of a vector and the fraction of the pairs that match the restriction on its digits imposed by the next level. So far both rest on a cumbersome combinatorial derivation by hand for the digit set of the tree [8,11,28,38,29,27,17,3,50]. Every new variant requires a new derivation, and every further digit or level adds parameters. In general, the number of parameters grows quadratically with the number of digits and linearly with the depth of the tree.

To carry out such an analysis by hand, published works often simplify it. Such simplifications restrict the depth of the trees, the digits or the freedom in the optimization, or approximate certain quantities. For LWE keys, the analyses of [38,29] for example fix the number of label bits of every level by the number of representations of that level, which is valid but not optimal [27]. For the Lee metric, the analysis of [50] replaces the number of representations by a lower bound. For restricted errors, the analyses of [3,4] limit the set of digits that occur in the lists. Such simplifications give correct but suboptimal algorithms.

Additional techniques often add more parameters. One such technique is a consistency filter introduced in the context of nearest-neighbor search by May and Ozerov [40]. Traditionally, a join finds all pairs whose labels match and then

discards all but an exponentially small fraction of them, because their sums do not have the dictated digit frequencies. A filter avoids forming most of the discarded pairs. Ozerov [44] uses filters for subset sum, at the price of introducing one variable for every pair of digits that the two vectors of a pair can have in the same position [44, Theorem 17], a number that grows quadratically with the number of digits. He applies a filter on a single level only, stating that the optimization problem is already sophisticated and that more filters lead to unreliable results in his optimizations [44, Sect. 4.2.2]. For the tree of [11], with its four digits and three join levels, filters at all joins come with 31 such variables. To date, the subset sum tree of [11], its time-memory trade-offs [28] and the Meet-LWE trees [38,29,27] use no filters.

Our Contribution. We show that the derivations by hand can be avoided. We introduce a general analysis technique for any representation tree, which replaces them by a few short entropy formulas for every digit set and turns the whole tree into one optimization problem with explicit formulas. We then embed filtering at every join into this general analysis. For a filter we prove that a number of parameters linear in the number of digits suffices, improving on Ozerov's quadratic modeling. We then show that filters can be further improved by partitioning the enumerated vectors into a sparse and a dense part. The filter then mostly acts on the dense parts, where the consistency loss is concentrated.

The introduced framework needs none of the simplifications above, and a new variant is evaluated without a new analysis. With it we improve on the previous best exponents in all settings of the problems mentioned above, where Table 1 gives a selection of achieved exponents in various settings in comparison to previous work. Note that for most of them this already holds without filters, demonstrating the gain from a unified analysis. With filters at every join, random subset sum is solved heuristically in time $2^{0.2719n}$ instead of $2^{0.2830n}$, and in time $2^{0.2620n}$ if the positions are additionally partitioned into a dense and a sparse part.

A General Analysis Technique. For a pair $(\mathbf{x}, \mathbf{y})$ we count how often each digit of $\mathbf{x}$ meets each digit of $\mathbf{y}$ in the same position, and call these counts the *joint type* of the pair. They determine how often each digit occurs in $\mathbf{x}$, in $\mathbf{y}$ and in $\mathbf{x} + \mathbf{y}$, so we describe a join by the joint type of the pairs that it keeps, the *consistent* pairs. For every digit set, the two numbers that the analysis needs are then short entropy formulas in the joint type (Lemma 1). The whole tree becomes one optimization problem, whose variables are the joint types and the numbers of enforced label bits of all levels.

Filters. A filter avoids the construction of many of the inconsistent pairs by using the dependence of the two vectors of a consistent pair: if the sum has digits in $\{0, 1\}$, a 2 in $\mathbf{x}$ must meet a -1 in $\mathbf{y}$. We show that filters can be naturally defined for the joint types of our analysis, allowing for a simple embedding into our framework. We then prove that the minimization of the filter running time

		this work		
problem	previous best	no filters	with filters	partitioning
subset sum	0.2830 [11]	0.2827	0.2719	**0.2620**
LWE, ternary, weight $n/2$	0.3218 [27]	0.3193	0.2935	**0.2684**
LWE, Kyber768/1024	0.357 [29]	0.3397	0.2997	**0.2761**
LWE, Kyber512	0.371 [29]	0.3458	0.3084	**0.3006**
LWE, Dilithium	0.378 [29]	0.3620	0.3107	**0.2954**
ternary decoding, hardest	0.243 [43]	0.2304	0.2179	**0.2148**
restricted errors, $\mathbb{F}_{157}$, $\omega = 0.8$	0.2517 [3]	0.2239	0.1962	**0.1931**
decoding over $\mathbb{F}_4$	0.2028 [41]	0.1995	**0.1972**	**0.1972**
Lee metric, $\mathbb{Z}_{47}$	0.660 [7]	0.4058	**0.3580**	**0.3580**

Table 1. Time exponents of previous work and of this work. Restricted errors with $z = |E| = 2$ and rate 0.5. Decoding over $\mathbb{F}_4$ and in the Lee metric at full distance, worst-case exponent over the rate.

has a closed form, so that a filter is given by at most two numbers per digit, and further symmetries halve this number. Additionally, we show that arbitrary far- or nearest-neighbor search is a special case of filtered joins and derive closed-form formulas for nearest-neighbor search over $\mathbb{F}_q$, which our decoding analysis uses. This translation yields the fastest known algorithm for nearest-neighbor search over $\mathbb{F}_q$, outperforming an earlier adaptation by Hirose [31].

Filters increase the depth of the optimal trees. A representation tree thins out the solution level by level until the vectors of the lowest level are sparse enough to be enumerated by a meet-in-the-middle, and the faster it thins out, the fewer levels are needed. Summands that are nearly as dense as their sum give the sum many more representations, which is why Becker, Coron and Joux [8] and Bonnetain et al. [11] obtained improvements by increasing the digit set. However, the denser the summands in comparison to the sum, the lower the fraction of consistent pairs. In a tree where inconsistent pairs need to be enumerated, thinning out more quickly at the cost of a lower number of representations is optimal. A filter, however, avoids forming most of the inconsistent pairs. As a result, the best trees with filters mostly use summands that are nearly as dense as their sum, so that more levels are needed to reach the same sparsity.

Partitioning. So far a join prescribes one joint type for all n positions. We instead split the positions into two parts, fixed once at random for the whole tree, and give each part its own joint type and its own filter at every join. A pair is then consistent if it has the defined joint types on both parts. Without filters this split does not change the running time. A filter gains at most what is lost to inconsistent pairs, and works best if the consistency loss is high. Therefore, we find that the optimal tree splits vectors into one sparse part using few different digits, and one dense part that uses many different digits. On the sparse part most pairs are consistent, while it provides relatively few representations. The dense part provides many representations, while most pairs are inconsistent.

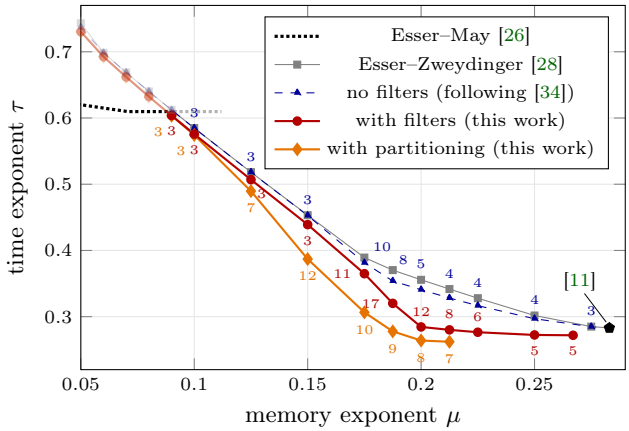


Fig. 1. Time-memory trade-off for random subset sum. The numbers give the depth of our trees without filters (blue), with filters (red) and with filters and partitioning (orange), numbers state optimal tree depth.

The filter then concentrates mostly on the dense part where it recovers most of the inconsistency loss. This is why the partitioning technique works best for applications where the fraction of consistent pairs drops rapidly with the density of the vectors. This is the case for subset sum and LWE, where digits are added over the integers. For decoding problems, where digits are added modulo q and the consistency loss converges, partitioning gives smaller gains.

Asymptotic Exponent Improvements. Leveraging the generality of our analysis, we optimize depth, digit set and filters including the introduced partitioning technique for all considered applications. For *subset sum*, the optimal tree achieves the time exponent 0.2620 and lies at depth 6, uses the digit set $\{-3,\ldots,3\}$ and incorporates partitioned filters at every join. Furthermore, this construction enables the best known time-memory trade-off for subset sum under any memory bound $2^{\mu n}$ with $\mu \geq 0.088$ (see Fig. 1). For small *LWE keys*, filters lower the exponents of Meet-LWE trees for all ternary key weights in Table 3 and for the key distributions of Kyber and Dilithium, and together with partitioning they lower the exponent by 0.0388 to 0.0826 compared to the best previous trees from [29,27]. Our optimal trees have depth between 5 and 20 and use digit sets up to $\{-5,\ldots,5\}$, whereas previous works were restricted to $\{-3,\ldots,3\}$ and used further simplifications. For *decoding* over $\mathbb{F}_q$, we obtain the first known improvements over the BJMM adaptation of Meurer [41] for all $3 \leq q \leq 27$ at half distance and for all $3 \leq q \leq 16$ at full distance. The improvement mostly stems from the incorporation of the $\mathbb{F}_q$ nearest-neighbor search, which is a special case of our filtered joins, on the top join of the tree. This contrasts with previous works [31,36], which found no gain by incorporating nearest-neighbor techniques over $\mathbb{F}_q$ as used in the best decoders over $\mathbb{F}_2$.

Our analysis technique also covers restricted errors [3], the problem behind the signature scheme CROSS [4], the Lee metric [50] and ternary decoding with large weight as in Wave [18,5]. For all of these applications we obtain significantly reduced exponents. All running times rely on the standard heuristic that lists behave like random sets (Heuristic 1), and all exponents are asymptotic upper bounds, attained by parameters we provide.

A Library for Representation Trees. We provide the Python library `jointtrees` (`https://github.com/Memphisd/jointtrees`), which makes it straightforward to obtain the optimization problem of any representation tree for each of the applications in this work, and solves it numerically. For example, optimizing a subset sum tree with the digits $\{-3, \dots, 3\}$, depth 10 and a filter at every join becomes

```
optimize(SubsetSum(), filtered(depth=10, digits=range(-3, 4)))
```

Additionally, we publish all code and witness files needed to recompute the results of this work in a public GitHub repository at `https://github.com/Memphisd/one-type-fits-all`.

Relation to Earlier Work. In terms of joint types, earlier works used related concepts: Ozerov [44] describes a pair of vectors by its joint distribution, and Bricout et al. [15] count ternary representations with a 3×3 table. These techniques are similar to the joint types we introduce, but those earlier works did not go beyond specific applications, which is why every recent representation analysis is still carried out by hand. In terms of filters, earlier works introduced them for nearest-neighbor search [21,40,1,25], and for problems related to this work they have mostly been used in lattice sieving [9,22] and decoding over $\mathbb{F}_2$ [40,14].

Outline. Section 2 fixes notation, Sections 3 and 4 present the analysis technique and the filters, Sections 5 to 7 apply them to subset sum, LWE keys and decoding over $\mathbb{F}_q$, and Section 8 covers three further decoding settings.

Concurrent Work. Shortly before this work, Howgrave-Graham [32] independently added filters to every join of the subset sum tree and of May's Meet-LWE tree [38] for ternary keys. The analysis also describes a join by a joint type and uses the entropy formulas of Lemma 1. Howgrave-Graham states them for integer digits without proof, while we prove them for every digit set, including digits in $\mathbb{F}_q$. For the running time [32] takes the cost formulas of the two trees from [11] and [38], whereas we derive them once for arbitrary digits and linear labels (Section 3.3). This derivation thus covers all applications of this work and can be adapted to further settings in a straightforward manner, while [32] names trees for decoding and other trees for ternary LWE keys as open problems. For LWE keys our exponents are lower than those of [32] already without filtering, as [32] carries many of the restrictions from earlier work in terms of digit sets, depths and analysis simplifications. For subset sum [32] obtains the exponent

0.2690 in comparison to our 0.2620, where the difference comes from the introduced partitioning technique. Further applications, time-memory trade-offs and the partitioning technique are not covered in [32]. We give a more detailed comparison in Appendix D.

2 Preliminaries

We write vectors in bold lower case and matrices in bold upper case, for example $\mathbf{x}$ and $\mathbf{A}$, and we denote the k-th entry of a vector $\mathbf{x}$ by x_k. Probability distributions and tuples of constant length are not set in bold.

Exponents. All logarithms are base 2. We write $A \simeq B$ if $A = B \cdot 2^{o(n)}$, where n is the length of the solution, and $\widetilde{\mathcal{O}}(\cdot)$ suppresses factors poly-logarithmic in the dominant term, which for exponential terms are polynomial factors. Unless stated otherwise, running times, memory and list sizes are stated as $2^{\tau n}$ up to factors $2^{o(n)}$, and we call the constant τ an *exponent*, like every other constant that multiplies n in a power of 2 or of q. Fractions of n such as $n/4$ are treated as integers, since rounding changes all counts by polynomial factors only.

Entropy. For a probability distribution $p = (p_1, \ldots, p_m)$ we define the *entropy* $H(p) := -\sum_i p_i \log p_i$, with the convention $0 \log 0 = 0$. The binary entropy function is $h(x) := -x \log x - (1-x)\log(1-x)$. We simplify multinomial coefficients via Stirling's formula as

$$\binom{n}{p_1 n, \ldots, p_m n} \simeq 2^{H(p)n}. \tag{1}$$

The left-hand side counts the vectors of length n in which the i-th of m symbols appears $p_i n$ times.

Random Subset Sum. We use the following problem as a basis for introducing the general techniques in Sections 3 and 4.

Definition 1 (Random Subset Sum Problem). *Let* $\mathbf{a} := (a_1, \ldots, a_n) \in \mathbb{Z}_{2^n}^n$ *be drawn uniformly at random. For a random* $\mathbf{e} \in \{0,1\}^n$ *with* $n/2$ *ones, let* $t := \sum_i a_i e_i \bmod 2^n$. *The* random subset sum problem *is, given* $(\mathbf{a}, t)$*, to find any* $\mathbf{e}' \in \{0,1\}^n$ *satisfying* $\sum_i a_i e'_i = t \bmod 2^n$. *We call any such* $\mathbf{e}'$ *a* solution *and* $(\mathbf{a}, t)$ *an* instance.

Labels and Constraints. In every problem of this work we are given a random matrix $\mathbf{A}$ and a vector $\mathbf{t}$, which we call the *target*. We search for a vector $\mathbf{x}$ from a predefined set, for instance the vectors in $\{0,1\}^n$ or the vectors with few nonzero entries, whose *label* $\phi(\mathbf{x}) := \mathbf{A}\mathbf{x}$ equals $\mathbf{t}$, for LWE up to a small error. For subset sum the matrix $\mathbf{A}$ has the single row $\mathbf{a}$ and the target $\mathbf{t}$ the single entry t, both modulo 2^n, so that the label of $\mathbf{x} \in \mathbb{Z}^n$ is $\phi(\mathbf{x}) = \sum_i a_i x_i \bmod 2^n$. Labels are linear, $\phi(\mathbf{x}+\mathbf{y}) = \phi(\mathbf{x}) + \phi(\mathbf{y})$. A label has $\ell_{\max} n$ bits, the largest number of bits

a constraint can have, and a *constraint on ℓn bits* fixes a value on ℓn of them. For subset sum the label has n bits, so that $\ell_{\max} = 1$, and a constraint on ℓn bits fixes the label modulo $2^{\ell n}$. For LWE and decoding the label is a vector with coordinates modulo a prime q or in $\mathbb{F}_q$, and every coordinate counts $\log q$ bits, so that $\ell_{\max} n$ is the number of coordinates times $\log q$ and a constraint on ℓn bits fixes $\ell n / \log q$ of the coordinates. We state the details for each application in its section.

The Heuristic. The analyses of representation trees (Section 3) use the following standard heuristic [33,8,39,10,11,28,38]. Several works prove the heuristic, or the parts of it that their analyses need, for their trees and parameters [33,8,39,41,42,19,37].

Heuristic 1. *The labels of distinct vectors of the same level behave like independent uniformly random values on the bits that their constraints leave free, and every list behaves like a uniformly random subset of the vectors of its type (Section 3.1) that satisfy its constraints.*

Our analyses use it to replace the size of a list, the number of label matches of a join and the number of representations (Section 3) of the solution that satisfy all constraints by their expected values.

Numerical Optimization and Witnesses. The parameters of our trees are obtained from a non-convex optimization problem (Section 3.3), which we solve numerically. Every exponent we report is attained by parameters that we provide and is therefore an upper bound on the optimum of this problem. These parameters are published as *witness* files, from which our code recomputes every exponent. Our exponents are rounded up at the last digit shown, the differences stated later are computed from the unrounded values and rounded down, and published values are quoted as published.

3 Representation Trees Described by Joint Types

The representation technique of Howgrave-Graham and Joux [33] writes the solution as a sum $\mathbf{x}+\mathbf{y}$ of two vectors, and we call such a pair $(\mathbf{x}, \mathbf{y})$ a *representation* of the solution. Since the nonzero entries of the solution can be distributed over $\mathbf{x}$ and $\mathbf{y}$ in many ways, the solution has exponentially many representations, allowing one to introduce linear constraints on those representations. Candidates for $\mathbf{x}$ and $\mathbf{y}$ are then computed in a tree-like fashion.

In this section we describe a representation tree by one joint type per level. We then express the two counts that the analysis needs, the number of representations and the fraction of pairs whose sum has the required digit frequencies, by entropies of the joint type. Finally, we state the optimization problem based on joint types. This description is the same for all problems of this work. The application sections therefore refer back to this problem and only state their specifics, such as the digits and the labels.

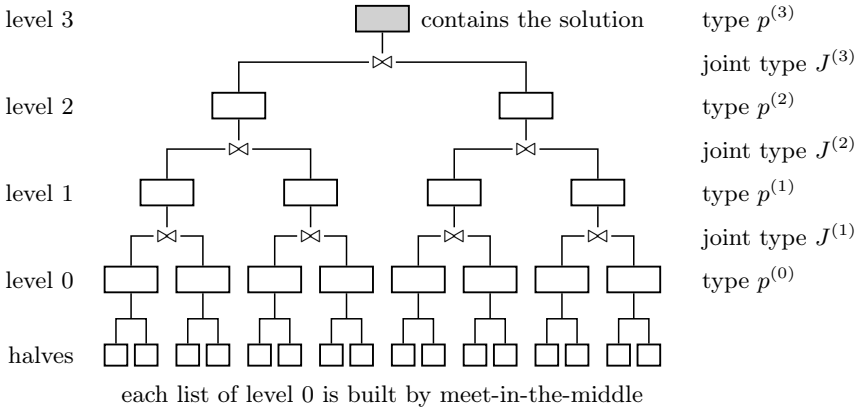


Fig. 2. A representation tree with three join levels, its types $p^{(i)}$ and joint types $J^{(i)}$.

3.1 The Tree

A tree with d *join levels*, which we also call a tree of *depth* d, has 2^d lists on level 0, 2^{d-i} lists on level i, and one list on level d that contains the solution (Fig. 2). The *join into level* i combines two lists of level $i-1$ into one list of level i. It searches for pairs $(\mathbf{x}, \mathbf{y})$, with $\mathbf{x}$ from the first and $\mathbf{y}$ from the second list, and puts the sum $\mathbf{x}+\mathbf{y}$ of every pair it keeps into the new list. A pair is kept if the label of $\mathbf{x}+\mathbf{y}$ matches a predefined value on some bits and $\mathbf{x}+\mathbf{y}$ is a vector that the next level admits. Earlier works state the second condition through the frequencies of the digits of $\mathbf{x}+\mathbf{y}$, where a digit is a value of an entry. We state it through the joint type of $(\mathbf{x}, \mathbf{y})$, that is, how often each digit of $\mathbf{x}$ meets each digit of $\mathbf{y}$. Up to a polynomial factor, the earlier condition is the case of the joint type of largest entropy (see Appendix A).

Types and Joint Types. Let $\mathcal{A} \subset \mathbb{Z}$ be a finite set of *digits*, for instance $\mathcal{A} = \{-1, 0, 1, 2\}$. The *type* of a vector $\mathbf{x} \in \mathcal{A}^n$ is the distribution $p = (p_i)_{i \in \mathcal{A}}$ of its digits, that is, $p_i := |\{k : x_k = i\}|/n$. By Eq. (1) the number of vectors of type p is $\simeq 2^{H(p)n}$.

The *joint type* of a pair of vectors $\mathbf{x}, \mathbf{y} \in \mathcal{A}^n$ is the table $J = (J_{ij})_{i,j \in \mathcal{A}}$ with

$$J_{ij} := |\{k : x_k = i \text{ and } y_k = j\}|/n.$$

Fig. 3 shows an example. The joint type determines the types of $\mathbf{x}$, of $\mathbf{y}$ and of $\mathbf{x}+\mathbf{y}$. The row sums $\sum_j J_{ij}$ give the type of $\mathbf{x}$ and the column sums $\sum_i J_{ij}$ give the type of $\mathbf{y}$. The sums over the anti-diagonals,

$$p^+_s := \sum_{i+j=s} J_{ij},$$

give the type p^+ of the sum $\mathbf{x}+\mathbf{y}$. We call a pair (i, j) with $i+j=s$ a *split* of the digit s. Note that p^+ is indexed by the sums $s = i+j$ of two digits, which

$\mathbf{x}$	0	0	1	0	2	−1	−1	1
$\mathbf{y}$	0	0	0	1	−1	2	1	−1
$\mathbf{x}+\mathbf{y}$	0	0	1	1	1	1	0	0

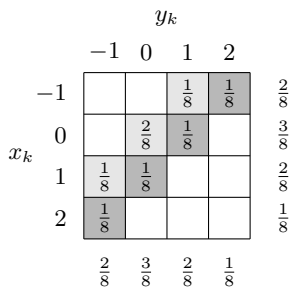


Fig. 3. A pair $(\mathbf{x}, \mathbf{y})$ of length $n = 8$ and its joint type with row and column sums. Light entries have $i + j = 0$ and dark entries have $i + j = 1$.

need not be digits of $\mathcal{A}$. From Section 7 on, the digits are elements of $\mathbb{F}_q$ or $\mathbb{Z}_q$, and sums of digits are then taken in that field or ring. A joint type is the joint distribution of the pair of digits of $\mathbf{x}$ and $\mathbf{y}$ in a uniformly random position, so that its entropy $H(J)$ is defined as in Section 2, with J as a distribution on the pairs (i, j). Eq. (1) applied to the pairs of digits shows that the number of pairs $(\mathbf{x}, \mathbf{y})$ of joint type J is $\simeq 2^{H(J)n}$.

Levels and Joins. All vectors on level i have the same type $p^{(i)}$ (Fig. 2), and the type $p^{(d)}$ of the top level is that of the solution, which for subset sum is $(\frac{1}{2}, \frac{1}{2})$. Subscripts of types and joint types index digits, whereas a superscript (i) denotes the level. The join into level i is described by a joint type $J^{(i)}$, and it keeps a pair $(\mathbf{x}, \mathbf{y})$ only if the joint type of the pair is $J^{(i)}$, in which case we call the pair *consistent*. The row and column sums of $J^{(i)}$ are the type $p^{(i-1)}$ of the two lists that the join combines, and its anti-diagonal sums are the type $p^{(i)}$ of the list it produces. Choosing the joint types $J^{(d)}, J^{(d-1)}, \ldots, J^{(1)}$ from the top down thus fixes the types of all levels.

3.2 Representations and Consistency from Joint Types

The analysis of a tree needs two numbers for every join. The first is the number of pairs of the join that sum to the same vector, which determines how many label bits the tree may constrain. The second is the fraction of all pairs that are consistent, which determines the list sizes. The following lemma states both numbers as entropy expressions of the joint type, for every digit set.

Lemma 1 (Counts of a Joint Type). *Let J be a joint type whose row sums and column sums are p and whose anti-diagonal sums are p^+.*

(a) Every vector $\mathbf{z}$ of type p^+ can be written as $\mathbf{z} = \mathbf{x} + \mathbf{y}$ with a pair $(\mathbf{x}, \mathbf{y})$ of joint type J in 2^{rn} ways up to polynomial factors, where $r = H(J) - H(p^+)$.

(b) Among all pairs $(\mathbf{x}, \mathbf{y})$ of vectors of type p, a fraction of 2^{-cn} up to polynomial factors is consistent, that is, has joint type J, where $c = 2H(p) - H(J)$.

Proof. By Eq. (1) there are, up to polynomial factors, $2^{H(J)n}$ pairs of joint type J, $2^{H(p^+)n}$ vectors of type p^+ and $2^{2H(p)n}$ pairs of vectors of type p. Every pair of joint type J sums to a vector of type p^+, and since permuting the positions maps the representations of a vector one-to-one to those of the permuted vector, all vectors of type p^+ have the same number of representations, which proves (a). For (b), note that the row and the column sums of the joint type J are both p, so that both vectors of a pair of joint type J have type p, and that there are $2^{2H(p)n}$ pairs of vectors of type p, up to polynomial factors. The fraction is the quotient $2^{(H(J)-2H(p))n}$ of the two counts. □

Put differently, the exponent r of the number of representations is the entropy that is lost when a pair is replaced by its sum.

3.3 The Optimization Problem

Besides the joint types, a tree has one more parameter per level, the *constraint size* $\ell^{(i)}$ of level i. The lists of level 0 are constrained on $\ell^{(0)}n$ label bits. The join into level $i < d$ adds $\ell^{(i)}n$ new bits, so that the lists of level i are constrained on $(\ell^{(0)} + \cdots + \ell^{(i)})n$ bits. Since labels are linear, the constraint of a list of level i agrees on the bits of the levels below with the sum of the constraints of the two lists it is built from, and only its $\ell^{(i)}n$ new bits can be chosen. The top join adds the remaining $\ell^{(d)}n$ of the $\ell_{\max}n$ bits of the label, that is, $\ell^{(d)} = \ell_{\max} - (\ell^{(0)} + \cdots + \ell^{(d-1)})$. In the following all sizes are exponents, that is, a list of level i has $2^{L^{(i)}n}$ elements and we call $L^{(i)}$ its size.

List Sizes. Let $D^{(i)} := H(p^{(i)})$, so that there are $2^{D^{(i)}n}$ vectors of type $p^{(i)}$. A list of level 0 contains all vectors of type $p^{(0)}$ that satisfy its constraint, which gives $L^{(0)} = D^{(0)} - \ell^{(0)}$. We write $T^{(0)}$ and $M^{(0)}$ for the time and memory exponents of building one such list. Unless stated otherwise, the lists of level 0 are built by meet-in-the-middle. It enumerates the $2^{D^{(0)}n/2}$ vectors of length $n/2$ and type $p^{(0)}$, once as left and once as right halves, and combines a left and a right half whose labels add up to the constraint. This finds the vectors of type $p^{(0)}$ whose two halves both have type $p^{(0)}$, an inverse-polynomial fraction of all vectors of type $p^{(0)}$, in time and memory

$$T^{(0)} = \max\big(D^{(0)}/2,\ L^{(0)}\big), \qquad M^{(0)} = D^{(0)}/2. \tag{2}$$

For the join into level i there are $2^{2L^{(i-1)}n}$ pairs, of which $2^{(2L^{(i-1)}-\ell^{(i)})n}$ have labels that match on the $\ell^{(i)}n$ new bits. By Lemma 1(b) a $2^{-c^{(i)}n}$ fraction of them is consistent, where $c^{(i)} := 2H(p^{(i-1)}) - H(J^{(i)})$. This gives

$$L^{(i)} = 2L^{(i-1)} - \ell^{(i)} - c^{(i)} \qquad \text{for } 1 \le i < d. \tag{3}$$

Eq. (3) counts pairs, and it gives the size of the list only if different pairs yield different sums. A list of level i cannot contain more than the $2^{(D^{(i)}-\ell^{(0)}-\cdots-\ell^{(i)})n}$ vectors of type $p^{(i)}$ that satisfy its constraints. Below this number the sums of

the pairs are distinct up to a constant fraction under Heuristic 1. Therefore, to ensure distinct elements in all lists, we require the *saturation constraints*

$$L^{(i)} \leq D^{(i)} - (\ell^{(0)} + \cdots + \ell^{(i)}) \qquad \text{for } 1 \leq i < d. \tag{4}$$

Correctness. The tree finds the solution if at least one representation of the solution satisfies all constraints. Let $r^{(i)} := H(J^{(i)}) - H(p^{(i)})$ be the exponent of the number of representations at the join into level i from Lemma 1(a). The solution is a sum of two vectors of level $d-1$ in $2^{r^{(d)}n}$ ways, each of these two vectors is a sum of two vectors of level $d-2$ in $2^{r^{(d-1)}n}$ ways, and so on. In total the solution has

$$2^{(r^{(d)}+2r^{(d-1)}+\cdots+2^{d-1}r^{(1)})n}$$

representations as a sum of 2^d vectors of level 0. In such a representation, the labels of the 2^{d-i} vectors of level i add up to the label $\mathbf{t}$ of the solution. The values that the constraints of the 2^{d-i} lists of level i dictate must therefore add up to $\mathbf{t}$ on the constrained bits. The new bits of the first $2^{d-i}-1$ constraints are chosen at random, and the last constraint is $\mathbf{t}$ minus the sum of the others. A fixed representation satisfies the randomly chosen constraints of all levels with probability $2^{-\sum_i (2^{d-i}-1)\ell^{(i)}n}$. Hence, we expect at least one representation to satisfy all constraints if

$$\sum_{i=0}^{d-1} \big(2^{d-i}-1\big)\ell^{(i)} \;\leq\; \sum_{i=1}^{d} 2^{d-i} r^{(i)}. \tag{5}$$

The left-hand side counts the constrained bits and the right-hand side the available representations. Following [28], we call Eq. (5) the *correctness constraint*.

Repetitions. Generally, larger constraints give smaller lists. Under a memory bound it can therefore be beneficial to violate the correctness constraint and compensate by repeating the tree construction for new random constraints [28]. Only the upper part of the tree has to be rebuilt for a repetition that changes the constraints of the upper levels. To quantify this, we define analogously to [28] the *deficit* of level i as

$$\Delta^{(i)} := \max\Big(0,\; \sum_{j=0}^{i} \big(2^{d-j}-1\big)\ell^{(j)} - \sum_{j=1}^{d} 2^{d-j} r^{(j)}\Big) \qquad \text{for } 0 \leq i \leq d. \tag{6}$$

The deficit is the amount by which the constraints of the levels up to i exceed the available representations. A representation of the solution satisfies the constraints of the levels up to i for one out of $2^{\Delta^{(i)}n}$ choices of these constraints in expectation. The part of the tree from level i upwards is therefore built $2^{\Delta^{(i)}n}$ times in total. If the correctness constraint (5) holds, all deficits are zero and every level is built once.

Algorithm 1 Representation Tree

Require: instance $(\mathbf{A}, \mathbf{t})$, joint types $J^{(1)}, \dots, J^{(d)}$, constraint sizes $\ell^{(0)}, \dots, \ell^{(d-1)}$
Ensure: an element of the top list $\mathcal{L}_1^{(d)}$, a vector $\mathbf{x}$ of type $p^{(d)}$ with $\phi(\mathbf{x}) = \mathbf{t}$
1: compute $\Delta^{(0)}, \dots, \Delta^{(d)}$ via Eq. (6) and set $\Delta^{(-1)} := 0$
2: **return** BUILD(0)

3: **procedure** BUILD(i) ▷ *build level i, then all levels above it*
4: **for** $2^{(\Delta^{(i)} - \Delta^{(i-1)})n}$ times **do** ▷ *once if all deficits are zero*
5: choose the $\ell^{(i)}n$ new bits of the constraints of $\mathcal{L}_1^{(i)}, \dots, \mathcal{L}_{2^{d-i}-1}^{(i)}$ at random
6: set the constraint of $\mathcal{L}_{2^{d-i}}^{(i)}$ to $\mathbf{t}$ minus the sum of the others
7: **if** $i = 0$ **then**
8: build $\mathcal{L}_1^{(0)}, \dots, \mathcal{L}_{2^d}^{(0)}$ ▷ *Eq. (2), or as the application states*
9: **else**
10: $\mathcal{L}_j^{(i)} \leftarrow \mathcal{L}_{2j-1}^{(i-1)} \bowtie \mathcal{L}_{2j}^{(i-1)}$ for $1 \le j \le 2^{d-i}$ ▷ *joint type $J^{(i)}$, $\ell^{(i)}n$ new bits*
11: **if** $i = d$ and $\mathcal{L}_1^{(d)} \neq \emptyset$ **then return** an element of $\mathcal{L}_1^{(d)}$
12: **else if** $i < d$ and BUILD($i+1$) finds a solution **then return** it
13: **return** nothing found

The Algorithm. Algorithm 1 builds the tree from the bottom up. We write $\mathcal{L}_1^{(i)}, \dots, \mathcal{L}_{2^{d-i}}^{(i)}$ for the lists of level i. The loop of BUILD(i) runs $2^{(\Delta^{(i)} - \Delta^{(i-1)})n}$ times per call, so that level i is built $2^{\Delta^{(i)}n}$ times in total. The join $\bowtie$ in line 10 enumerates the pairs whose labels match on the new bits and keeps those of joint type $J^{(i)}$. Section 4 replaces this line by a filtered join.

Time and Memory. A join into level i reads its two input lists, enumerates all pairs with matching labels and tests each of them, which takes time

$$T^{(i)} = \max\bigl(L^{(i-1)},\ 2L^{(i-1)} - \ell^{(i)}\bigr). \tag{7}$$

For $i = d$ the join uses the remaining $\ell^{(d)}n$ label bits. Since level i is built $2^{\Delta^{(i)}n}$ times, the time and memory exponents of the tree are

$$\tau = \max_{1 \le i \le d} \bigl(T^{(0)} + \Delta^{(0)},\ T^{(i)} + \Delta^{(i)}\bigr), \qquad m = \max\bigl(M^{(0)},\ L^{(0)}, \dots, L^{(d-1)}\bigr). \tag{8}$$

The memory is the largest stored list, where the top list is not stored, since its elements are returned as they are found. The parameters of the tree are now obtained from the optimization problem

$$\begin{array}{lll} \text{minimize} & \tau \text{ of Eq. (8), with the deficits } \Delta^{(i)} \text{ of Eq. (6),} & \\ & \text{over } J^{(1)}, \dots, J^{(d)} \text{ and } \ell^{(0)}, \dots, \ell^{(d-1)} & \\ \text{subject to} & L^{(i)} \le D^{(i)} - (\ell^{(0)} + \cdots + \ell^{(i)}), \quad 1 \le i < d & \text{saturation (4)} \\ & m \le \mu & \text{memory bound} \end{array} \tag{9}$$

for a given memory bound $2^{\mu n}$. Without a memory bound the last constraint is dropped and all deficits are zero, i.e., Eq. (5) holds.

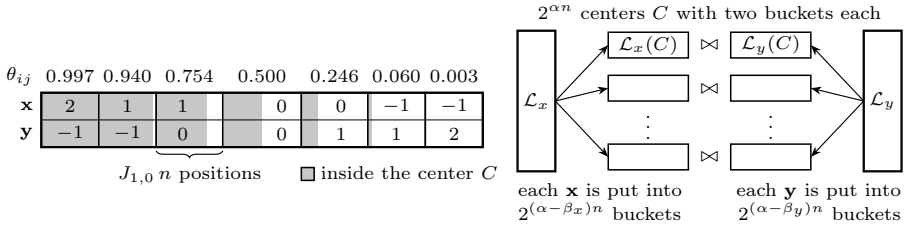


Fig. 4. A center C on a consistent pair (left) and the buckets of the filtered join (right).

We omit from Eq. (9) the constraints that keep the parameters in their domain, namely that the entries of a joint type are non-negative and have the required sums, $\ell^{(i)} \geq 0$, $L^{(i)} \geq 0$ and $\ell^{(0)} + \cdots + \ell^{(d-1)} \leq \ell_{\max}$. We call the optimization problem (9) with its cost formulas the *model*. The applications add their own parameters and constraints, which we state where they arise.

4 Filters for a Joint Type

The join of Section 3 enumerates all pairs whose labels match on the new bits, and by Lemma 1(b) it discards all but a 2^{-cn} fraction of them as inconsistent. In this section we replace it by a join that never forms most of the inconsistent pairs, following the consistency filters of Ozerov [44]. The cost of such filters has been derived for nearest-neighbor search [21,40,25] and for sieving [9,22], and we state it here generally for the joint type of a join of a representation tree. For the digits of subset sum we first show that the digits of a consistent pair are negatively correlated, which leads to conditions on single vectors that keep a consistent pair with larger probability than a pair of independent vectors. We then define the filter for an arbitrary joint type whose row and column sums agree, i.e., $\mathbf{x}$ and $\mathbf{y}$ have the same type, and derive the cost of the filtered join. Finally, we show how to reduce the number of optimization variables introduced by a filter in comparison to previous works and embed filters into the analysis of Section 3.3.

Basic Idea. Consider a pair $(\mathbf{x}, \mathbf{y})$ over the digits $\{-1, 0, 1, 2\}$ whose sum has digits in $\{0, 1\}$. The two digits at a position add up to 0 or 1, so a digit 2 of $\mathbf{x}$ meets a digit -1 of $\mathbf{y}$, and a digit -1 of $\mathbf{x}$ meets a digit 1 or 2 of $\mathbf{y}$. Put differently, where $\mathbf{x}$ has a large digit, $\mathbf{y}$ has a small one. Two independent vectors of the same type show no such dependence. Filters use this correlation to discard vectors before any pairs are formed.

Put simply, we choose a random set C of half of the positions. From the left list we keep only the vectors that have a predefined number of every digit inside C, with more than half of their large digits, and from the right list only those with more than half of their small digits inside C. Each of the two conditions concerns a single vector, so both lists are reduced separately, and labels are matched only among the vectors that are kept. For a consistent pair the two

conditions are positively dependent: if $\mathbf{x}$ has many of its large digits inside C, then $\mathbf{y}$ has many of its small digits there. Fig. 4 (left) shows this for the top join of a tree with the digits $\{-1, 0, 1, 2\}$: the positions of a consistent pair are grouped by the two digits, and the number above a group is its fraction inside C. A consistent pair therefore satisfies both conditions with a larger probability than a pair of independent vectors, by an exponential factor that we quantify in Section 4.2.

A single set C still keeps only a small fraction of the consistent pairs. We therefore repeat the procedure with many independent sets, until every consistent pair is kept by at least one of them. The cost of a filter is thus determined by three quantities: the number of sets that are needed, and the fractions of the left and of the right list that one set keeps. In the following we compute them for an arbitrary joint type whose row and column sums agree.

4.1 Centers

We fix a joint type J whose row sums and column sums are both p. Recall that a pair $(\mathbf{x}, \mathbf{y})$ of vectors of type p is consistent if its joint type is J. A *center* of *center size* $0 < u < 1$ is a set $C \subset \{1, \dots, n\}$ of un positions, chosen uniformly at random. A *filter* is a triple (u, δ^x, δ^y) of a center size and two tuples $\delta^x = (\delta^x_i)_{i \in \mathcal{A}}$ and $\delta^y = (\delta^y_i)_{i \in \mathcal{A}}$ with $\sum_i \delta^x_i = \sum_i \delta^y_i = 0$, the *tilts*. A join uses one filter and many centers. A center *accepts* a vector $\mathbf{x}$ of the left list if the restriction of $\mathbf{x}$ to C has type $p + \delta^x$, that is, if every digit i appears exactly $u(p_i + \delta^x_i)n$ times among the positions in C. It accepts a vector $\mathbf{y}$ of the right list if the restriction of $\mathbf{y}$ to C has type $p + \delta^y$. The filter outlined above has $u = 1/2$, the tilt δ^x is positive on the large digits, and $\delta^y = -\delta^x$.

Fraction of a List Accepted by a Center. Let $\mathbf{x}$ be a vector of type p that a center accepts on the left. Its un digits inside C have type $p + \delta^x$, and since $\mathbf{x}$ has type p, its $(1-u)n$ digits outside C have type $p - \frac{u}{1-u}\delta^x$. Counting both parts by Eq. (1), a fixed center accepts a $2^{-\beta_x n}$ fraction of the vectors of type p, where

$$\beta_x = H(p) - u\,H\big(p + \delta^x\big) - (1-u)\,H\Big(p - \tfrac{u}{1-u}\delta^x\Big). \tag{10}$$

Here the first term counts all vectors of type p, the second term the arrangements of the digits inside C and the third term the arrangements outside C. For the right list we obtain β_y as the same expression in δ^y. We call the exponents β_x and β_y the *list reductions* of the filter.

Probability That a Center Accepts a Consistent Pair. Fix a consistent pair $(\mathbf{x}, \mathbf{y})$. We sort its positions into groups, where the group (i, j) consists of the $J_{ij}n$ positions at which $\mathbf{x}$ has digit i and $\mathbf{y}$ has digit j. With respect to this pair, a set C is described by the fraction $\theta_{ij} \in [0, 1]$ of every group that it contains (Fig. 4, left). The center accepts the pair, that is, both $\mathbf{x}$ and $\mathbf{y}$, if and only if

$$\sum_j J_{ij}\theta_{ij} = u\big(p_i + \delta^x_i\big) \quad \text{for every } i, \qquad \sum_i J_{ij}\theta_{ij} = u\big(p_j + \delta^y_j\big) \quad \text{for every } j. \tag{11}$$

The left-hand sides are the numbers of digits i of $\mathbf{x}$ and of digits j of $\mathbf{y}$ inside C, divided by n. We call fractions $\theta = (\theta_{ij})$ that satisfy Eq. (11) *admissible*. Summing the first condition over i gives

$$u = \sum_{i,j} J_{ij}\theta_{ij}, \tag{12}$$

since $\sum_i p_i = 1$ and $\sum_i \delta_i^x = 0$. In other words, the numbers of positions $J_{ij}\theta_{ij}n$ that C contains from the groups (i,j) add up to the un positions of C.

For given admissible fractions θ_{ij}, a random center C of un positions contains exactly the fraction θ_{ij} of the positions of every group (i,j) with probability (compare Eq. (1))

$$\prod_{i,j} \binom{J_{ij}n}{\theta_{ij}J_{ij}n} \Big/ \binom{n}{un} \;\simeq\; 2^{-K_u(\theta)n}, \qquad \text{where} \quad K_u(\theta) := h(u) - \sum_{i,j} J_{ij}\, h(\theta_{ij}). \tag{13}$$

The fractions take only polynomially many values, so a random center accepts the pair with probability $2^{-\alpha n}$ up to polynomial factors, where

$$\alpha := \min\big\{K_u(\theta) : \theta \text{ admissible}\big\}. \tag{14}$$

Put differently, among all ways in which a center can accept the pair, the most frequent one determines the probability. Consequently, about $2^{\alpha n}$ random centers are needed until every consistent pair is accepted by one of them, and we call the exponent α the *center count* of the filter.

4.2 The Filtered Join

The join receives two lists $\mathcal{L}_x, \mathcal{L}_y$ of 2^{Ln} vectors of type p each, and it has to output all consistent pairs in $\mathcal{L}_x \times \mathcal{L}_y$ whose labels match on ℓn new bits. For the join into level i of Section 3, the quantities L, ℓ, p and J are $L^{(i-1)}$, $\ell^{(i)}$, $p^{(i-1)}$ and $J^{(i)}$, and we omit the level index here. The filtered join (Algorithm 2) draws $2^{\alpha n} \cdot \mathrm{poly}(n)$ random centers. This number ensures that every consistent pair is accepted by at least one center, except with probability $2^{-\Omega(n)}$. For every center C, the left bucket $\mathcal{L}_x(C)$ and the right bucket $\mathcal{L}_y(C)$ contain the vectors of $\mathcal{L}_x$ and of $\mathcal{L}_y$ that C accepts. The join matches labels only between the two buckets of a center and keeps the label matches that are consistent (Fig. 4, right).

Cost. Recall that a fixed center accepts a $2^{-\beta_x n}$ fraction of the vectors of type p (Eq. (10)). By symmetry under permuting the positions, a random center accepts a fixed left vector with the same probability, up to a polynomial factor. A left vector is therefore inserted into $2^{(\alpha-\beta_x)n}$ buckets in expectation, and a right vector into $2^{(\alpha-\beta_y)n}$ buckets. Under Heuristic 1, every center in turn receives $2^{(L-\beta_x)n}$ left and $2^{(L-\beta_y)n}$ right vectors in expectation, and a $2^{-\ell n}$ fraction of the pairs in its two buckets match on the new bits. Each label match is tested

Algorithm 2 Filtered Join

Require: lists $\mathcal{L}_x, \mathcal{L}_y$ of 2^{Ln} vectors of type p, joint type J, number ℓn of new bits, filter (u, δ^x, δ^y) with center count α
Ensure: all $(\mathbf{x}, \mathbf{y}) \in \mathcal{L}_x \times \mathcal{L}_y$ of joint type J whose labels match on the ℓn new bits

$\mathcal{O} \leftarrow \emptyset$ ▷ *set, stores every pair once*
draw $2^{\alpha n} \cdot \mathrm{poly}(n)$ random centers
for all $\mathbf{x} \in \mathcal{L}_x$ and all centers C that accept $\mathbf{x}$ **do**
  insert $\mathbf{x}$ into the left bucket $\mathcal{L}_x(C)$ ▷ $2^{(\alpha-\beta_x)n}$ *centers per* $\mathbf{x}$
for all $\mathbf{y} \in \mathcal{L}_y$ and all centers C that accept $\mathbf{y}$ **do**
  insert $\mathbf{y}$ into the right bucket $\mathcal{L}_y(C)$ ▷ $2^{(\alpha-\beta_y)n}$ *centers per* $\mathbf{y}$
for all centers C **do**
  find all $(\mathbf{x}, \mathbf{y}) \in \mathcal{L}_x(C) \times \mathcal{L}_y(C)$ whose labels match on the ℓn new bits
  add those of joint type J to $\mathcal{O}$
return $\mathcal{O}$

for consistency in time $\mathrm{poly}(n)$. Summed over all centers, the time exponent of the insertions and of the label matches is

$$T = \max\bigl(L + \alpha - \beta_x,\ L + \alpha - \beta_y,\ 2L - \ell + \alpha - \beta_x - \beta_y\bigr). \tag{15}$$

We call the first two terms the *bucketing terms* and the third term the *checking term*.

Let us compare Eq. (15) with the cost $\max(L, 2L - \ell)$ of the join without filter (Eq. (7)). The filter examines $2^{(\beta_x+\beta_y-\alpha)n}$ times fewer label matches. This is a reduction if $\alpha < \beta_x + \beta_y$, that is, if a center accepts a consistent pair with larger probability than a pair of independent vectors. The join without filter examines 2^{cn} label matches per pair that it outputs, where $c = 2H(p) - H(J)$ (Lemma 1(b)), and the filtered join examines $2^{(c-(\beta_x+\beta_y-\alpha))n}$. Thus, the reduction is at most the factor 2^{cn}. On the other hand, a center that accepts a pair accepts each of its two vectors, which implies $\alpha \geq \max(\beta_x, \beta_y)$, so the bucketing terms are at least the input size L. A filter therefore lowers the cost of a join only if the label matches dominate the cost of the join without filter, and its parameters trade off the bucketing terms against the checking term.

Finding the Accepting Centers. Eq. (15) counts every vector once for each center that accepts it, so the join has to find the accepting centers of a vector in time proportional to their number. Testing every vector against all $2^{\alpha n}$ centers would instead cost $2^{(L+\alpha)n}$. However, a standard technique removes this overhead by using random product codes [40,44,9]. These use *product centers*, which consist of one random center on each of $k = \lceil \sqrt{n} \rceil$ blocks of n/k positions, and store a vector in the bucket $C = C_1 \times \cdots \times C_k$ only if every center C_i accepts on its block. With these centers the filtered join attains the cost of Eq. (15) in time, with only a $2^{o(n)}$ memory overhead beyond its lists. For completeness, we give the details on this technique in Appendix C.

Complexity of the Filtered Tree. We now insert the filtered join into the tree of Section 3. From here on i denotes the level of the tree again, and digits are

indexed by j. The join in line 10 of Algorithm 1 is replaced by Algorithm 2 with product centers. The filtered join outputs the same pairs as the join without filter, so the list sizes, the numbers of representations and all constraints of Section 3.3 remain unchanged. If the join into level i uses a filter with exponents $\alpha^{(i)}, \beta_x^{(i)}, \beta_y^{(i)}$, its cost $T^{(i)}$ in Eq. (8) is given by Eq. (15) instead of Eq. (7). The parameters of one filter per join become additional variables of the optimization problem of Section 3.3. These are the prices $z_j^{(i)}$ of the digits, one set per list for a general filter, together with the factor $g^{(i)}$ that sets the center size where it is free. All exponents of a filter are explicit functions of these variables and of the joint type $J^{(i)}$, so the joint types, the constraint sizes and the filters of all levels are optimized together.

4.3 Parameterizing a Filter

The list reductions of Eq. (10) are explicit in the filter, but the center count of Eq. (14) is a minimum over the admissible fractions, with one variable θ_{ij} per pair of digits (i, j) with $J_{ij} > 0$. We call such a pair of digits *occurring*. Evaluating a filter given by its tilts thus requires solving an optimization problem in up to $|\mathcal{A}|^2$ variables, for every filter at every join of the tree. The next lemma shows that the minimizing fractions are determined by one positive number for every digit of $\mathbf{x}$ and one for every digit of $\mathbf{y}$, and that every choice of these numbers gives a filter with an explicit center count, so that describing filters by these numbers avoids the inner minimization.

Lemma 2 (Closed Form of the Center Count). *Let J be a joint type whose row sums and column sums are both p.*

(a) Let (u, δ^x, δ^y) be a filter for which there exist admissible fractions that lie strictly between 0 and 1 on all occurring pairs. Then the minimum in Eq. (14) is attained by fractions θ of the form

$$\theta_{ij} = \frac{w_i^x w_j^y}{1 + w_i^x w_j^y} \qquad \text{for all occurring pairs } (i, j) \tag{16}$$

for positive numbers w_i^x and w_j^y, $i, j \in \mathcal{A}$.

(b) Conversely, let w_i^x and w_j^y, $i, j \in \mathcal{A}$, be arbitrary positive numbers and define θ by Eq. (16). Then the row and column sums of Eq. (11) define a filter (u, δ^x, δ^y), and θ attains the minimum in Eq. (14) for this filter, that is, $\alpha = K_u(\theta)$.

Proof. Given in Appendix B. □

In words, the most frequent way in which a center accepts a consistent pair contains the group (i, j) with odds $\theta_{ij}/(1-\theta_{ij}) = w_i^x w_j^y$, the product of a factor for the digit of $\mathbf{x}$ and a factor for the digit of $\mathbf{y}$.

Two Symmetric Cases. By Lemma 2, a filter that satisfies the assumption of part (a) is given by the numbers w_i^x and w_j^y of Eq. (16), of which $2|\mathcal{A}|-1$ are free, since multiplying all w_i^x and dividing all w_j^y by the same constant does not change θ. A general filter thus adds $2|\mathcal{A}|-1$ optimization variables per join. In the following we reduce this number to $|\mathcal{A}|$ for two cases that are optimal in many of our applications. We call J *symmetric* if $J_{ij} = J_{ji}$ for all $i, j \in \mathcal{A}$, and joint types are symmetric from here on unless stated otherwise. The optimal filters then often take one of two forms, namely $w^y = w^x$ (same tilt, as it implies $\delta_i^x = \delta_i^y$) or $w^y = 1/w^x$ (opposite tilts: a digit that a center favors in $\mathbf{x}$ it disfavors in $\mathbf{y}$, with $\delta_i^x = -\delta_i^y$ exactly for $u = 1/2$). In both cases a filter is given by one positive number w_i per digit, and we call $z_i := \log w_i$ the *price* of digit i. The larger the price of a digit, the larger the fraction of the positions with this digit in $\mathbf{x}$ that lie inside the center. In both cases we let the center size u be a free parameter. To this end, we multiply all odds $w_i^x w_j^y$ in Eq. (16) by a common factor $g > 0$. Every fraction θ_{ij} then increases with g, from 0 for $g \to 0$ to 1 for $g \to \infty$, so the center size $u = \sum_{i,j} J_{ij}\theta_{ij}$ increases with g as well, and every $0 < u < 1$ corresponds to exactly one g. The optimization then runs over g instead of u. With the fraction $\theta_i^x := u(p_i + \delta_i^x)/p_i = \sum_j J_{ij}\theta_{ij}/p_i$ of the digits i of $\mathbf{x}$ that lie inside C, defined for $p_i > 0$, Eq. (10) reads

$$\beta_x = h(u) - \sum_i p_i\, h(\theta_i^x), \tag{17}$$

and analogously for β_y.

If the prices of the two digits of a consistent pair are positively correlated, both lists get the same tilt. If they are negatively correlated, the two lists get opposite tilts. For binary digits the first case means that the two vectors are closer than two independent vectors, and the second that they are farther apart. For the same tilt, $w_i^y = w_i^x =: w_i$, the fractions are

$$\theta_{ij} = \frac{g\, w_i w_j}{1 + g\, w_i w_j} \qquad \text{for all occurring pairs } (i,j). \tag{18}$$

Like J, they are symmetric in i and j, so both lists get the same tilt and $\beta_x = \beta_y$. For the opposite tilt, $w_i^x =: w_i$ and $w_i^y = 1/w_i$, the fractions are

$$\theta_{ij} = \frac{g\, w_i}{g\, w_i + w_j} \qquad \text{for all occurring pairs } (i,j). \tag{19}$$

By Lemma 2(b), every choice of the prices and of g gives a filter with the explicit exponents $\alpha = K_u(\theta)$ and β_x, β_y as above.

4.4 Partitioning a Filter

So far a filtered join uses a single joint type on all n positions. We now extend this to joins that may instead partition the positions into parts, each covering a fixed fraction of them, and give each part its own joint type and filter. A pair is consistent if it has the predefined joint type on every part. We consider a partitioning of the positions into two parts, fixed once at random and the same for every list of the tree, each with its own joint types and filters at every level.

Cost of a Partitioned Filter. A center is then one product center per part, and accepts a vector when every part's center accepts the vector's restriction to that part, i.e., both parts match the respective tilts. The list reductions β_x, β_y (Eq. (10)) and the center count α (Eq. (14)) add up over the positions, so each is the sum of its values on the parts, weighted by the fraction of the positions each part covers. More precisely, if we split the positions into parts of εn and $(1-\varepsilon)n$ positions with part $k \in \{1,2\}$ attaining $(\beta_x)_{(k)}$, we have $\beta_x = \varepsilon(\beta_x)_{(1)} + (1-\varepsilon)(\beta_x)_{(2)}$, and analogously for β_y and α. The filtered join then has the cost of Eq. (15) with these exponents, as Appendix C shows for a join with several parts. The counts $r^{(i)}$ and $c^{(i)}$ of Lemma 1 and $D^{(i)}$ in Eq. (4) are the weighted averages of the counts of their parts (similar to β_x) and the optimization problem (9) introduces variables for the joint types and filters of both parts and the fraction ε.

5 Application: Random Subset Sum

In this section we apply the filtered tree of Section 4 to the random subset sum problem (Definition 1).

Setup Let us give the subset sum specifics on digits, label sizes and the filters used.

Digits and Types. All levels below the top level use a subset of the digits $\mathcal{A} = \{-3, \dots, 3\}$. We obtain our best result for the full set and, additionally, compare against digit sets of earlier works such as $\{-1, 0, 1, 2\}$ [11], $\{-1, 0, 1\}$ [8] and $\{0, 1\}$ [33].

Labels and Lists of Level 0. Recall that $\ell_{\max} = 1$ and that the lists of level 0 are built by meet-in-the-middle (Eq. (2)).

Joins and Filters. All joins use opposite tilts with one price per digit and a free center size (Eq. (19)). The filter of the join into level i is thus given by the prices $z_j^{(i)}$ of the digits j and its center size $u^{(i)}$.

Numerical Results The variables of the optimization problem (9) are the joint types $J^{(1)}, \dots, J^{(d)}$, the constraint sizes $\ell^{(0)}, \dots, \ell^{(d-1)}$ and the prices and center sizes of the filters. When considering partitioned filters (Section 4.4) the optimization problem includes joint types, prices and a center size for both parts as well as ε. Without a filter at any join we obtain the trees of [11,8,33] in our model, and we call this baseline *no filters*.

In Table 2 we state the exponents of our trees without and with filters, next to the published values, where a dash marks a setting without a published value. The last column states the exponents with filters when the positions are partitioned into two parts and the bold entry is the best achieved exponent.

		no filters		with filters		with filters and
digits	d	published	this work	published	this work	partitioning
$\{0,1\}$	3	0.3370 [33]	0.3372	0.329 [44]	0.3270	0.3270
$\{-1,0,1\}$	3	0.2890 [11]	0.2892	0.287 [44]	0.2844	0.2843
$\{-1,0,1\}$	4	–	0.2896	–	0.2800	0.2798
$\{-1,0,1,2\}$	3	0.2830 [11]	0.2829	–	0.2791	0.2791
$\{-1,0,1,2\}$	4	–	0.2835	–	0.2726	0.2707
$\{-1,0,1,2\}$	5	–	0.2835	–	0.2724	0.2668
$\{-2,\ldots,2\}$	3	–	0.2827	–	0.2789	
$\{-2,\ldots,2\}$	6	–	0.2833	–	0.2719	0.2621
$\{-3,\ldots,3\}$	6	–	0.2833	–	0.2719	**0.2620**

Table 2. Time exponents for random subset sum.

We achieve the best exponent for a tree in depth 6 using filtered joins on every level with digits in $\{-3,\ldots,3\}$ below the top partitioned into two parts. In this tree the first part keeps its vectors sparse and the second part dense, which gives more representations but fewer consistent pairs. Most of this consistency loss falls on the second part, where the filters are much stronger and recover most of it (for details see Appendix I). For each digit set in the table we increased the depth until the resulting exponent did not change by our rounding rule. Adding ± 4 led to improvements below 10^{-5}.

Increasing the depth to 7 allows us to lower the memory from $2^{0.2560n}$ down to $2^{0.2120n}$ at the same time exponent as stated in the following theorem.

Theorem 1. *Under Heuristic 1, random subset sum can be solved in time* $2^{0.2620n}$ *and memory* $2^{0.2120n}$.

Proof. We run Algorithm 1 with the parameters of Table 11 in Appendix I in depth 7, with the digits $\{-3,\ldots,3\}$, partitioned into two parts (Section 4.4) and the filtered join of Algorithm 2 at every join. Under Heuristic 1 and since the parameters satisfy the saturation constraints (4) and the correctness constraint (5), the tree is built once and one representation of the solution is expected to reach the top list (Section 3.3). The lists of level 0 are built by the algorithm of Schroeppel and Shamir [47], which has the time of meet-in-the-middle (Eq. (2)) and half its memory, here $2^{0.2572n}$ and $2^{0.1286n}$. By Theorem 2 every join costs at most $2^{0.2620n}$, and the largest list has size $2^{0.2120n}$. □

Interestingly, in contrast to unfiltered representation trees, the introduction of filters leads to optimal trees with running time larger than their memory complexity. This points to improved behavior under memory bounds, which we study in the following.

Time-Memory Trade-Off We now bound the memory by $2^{\mu n}$. Recall from Section 3.3 that the constraint sizes that the bound requires may violate the

correctness constraint (5). The deficits of Eq. (6) are then positive, and the upper part of the tree is repeated with fresh random constraints.

Under a memory bound, the memory $2^{D^{(0)}n/2}$ of meet-in-the-middle can dominate the memory of the tree. As in [28], we then build the lists of level 0 by the 7-dissection of Dinur et al. [20], which with memory $2^{M^{(0)}n}$, $D^{(0)}/7 \leq M^{(0)} \leq D^{(0)}/4$, takes time $2^{T^{(0)}n}$ with

$$T^{(0)} = \max\big(D^{(0)}/2,\ 2(D^{(0)} - M^{(0)})/3,\ L^{(0)}\big). \tag{20}$$

We then set $M^{(0)} = \min(\mu, D^{(0)}/4)$ and add the constraint $D^{(0)} \leq 7\mu$.

In Fig. 1 we compare the trade-off of [28], which uses depth 3, with our trees without and with filters for memory bounds between $\mu = 0.05$ and $\mu = 0.275$. Kulpe [34] observed that under a memory bound trees with more levels can improve on the trade-offs from [28]. We therefore re-optimized the construction from [28] using larger depth. The numbers in the figure give the smallest depth at which our trees reach the plotted exponent, in blue without filters, in red with filters and in orange with filters and partitioning. The blue, red and orange lines are optimized with the digit set $\{-2, \ldots, 2\}$ until a depth increase yields no better exponent.

We find that filters lower the time exponent for every memory bound in Fig. 1 compared to [28,34], which makes the new trade-off the best known for any $\mu \geq 0.088$. Partitioning lowers the exponent further for any $\mu \geq 0.1$. As an example, at $\mu = 0.2$ the exponent drops by 0.0712 from 0.3557 [28] to 0.2845 with filters, and by 0.0917 to 0.2640 with filters and partitioning. Re-optimized trees without filters reach 0.3407 at this memory bound, so 0.0561 of the improvement is due to the filters and 0.0205 due to partitioning.

We observe a strong increase of the optimal depth for intermediate memory bounds. With filters the depth is 3 for $\mu \leq 0.15$, lies between 6 and 17 for $0.175 \leq \mu \leq 0.225$ and is 5 for $\mu \geq 0.25$. With filters and partitioning the depth is 3 for $\mu \leq 0.1$, lies between 7 and 12 for $0.1 < \mu$. A larger depth reduces the deficits $\Delta^{(i)}$ of Eq. (6), that is, the repetitions of the tree, which is why a larger depth is optimal in this regime already without filters. Filters additionally favor a larger depth, since deeper trees have denser vectors, which increases the consistency loss that the filters exploit. Partitioning, like a larger depth, serves as another technique to reduce the deficits, which is why filtered-only trees are sometimes deeper. With filters the deficits are nonzero for any $\mu \leq 0.2250$, with filters and partitioning only for $\mu \leq 0.1875$. For $0.1 \leq \mu \leq 0.2$ the largest deficit with partitioning is below the one with filters at every memory bound, which is why the time exponent with partitioning decreases faster in μ over this range.

6 Application: Small LWE Keys

In this section we apply the filtered tree of Section 4 to the enumeration of LWE keys with small coordinates. The best known combinatorial algorithms for this task follow the Meet-LWE trees of May [38] and are given by Glaser

and May [29] and by Esser, Mukherjee and Sarkar [27]. We first state where Meet-LWE, described as a tree of Section 3, differs from the subset sum tree of Section 5, and then give the exponents without and with filters, and with partitioning.

Instances and Keys. We consider the LWE problem for keys and errors with small coordinates.

Definition 2 (LWE Problem with Small Keys). *Let q be a prime that is polynomial in n, and let $\eta \geq 1$ be a constant. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ be drawn uniformly at random, let the* key $\mathbf{s} \in \{-\eta, \ldots, \eta\}^n$ *be drawn from a known key distribution and the* error $\mathbf{e} \in \{-\eta, \ldots, \eta\}^n$ *from an error distribution, and let $\mathbf{b} := \mathbf{A}\mathbf{s}+\mathbf{e} \bmod q$. The* LWE problem with small keys *is, given $(\mathbf{A}, \mathbf{b})$, to find $\mathbf{s}$.*

Note that in this section the vector that the tree searches for is the key $\mathbf{s}$, and $\mathbf{e}$ denotes the LWE error. We consider three kinds of keys. *Ternary keys* are vectors $\mathbf{s} \in \{-1, 0, 1\}^n$ with $\omega n/2$ entries equal to 1 and $\omega n/2$ entries equal to -1, and we call ωn the weight of the key. The second kind has coordinates that follow the centered binomial distribution $B(\eta)$ on $\{-\eta, \ldots, \eta\}$, which Kyber uses with $\eta = 3$ (Kyber512) and $\eta = 2$ (Kyber768 and Kyber1024). The third kind has coordinates that are uniform on $\{-2, \ldots, 2\}$, denoted $U(2)$ and used in Dilithium. For $B(\eta)$ and $U(2)$ we restrict to keys in which every digit appears with its expected frequency, which holds for a random key with probability $1/\mathrm{poly}(n)$ [29, Lemma 1]. In all three cases the key is then a vector of a known type, which is the type $p^{(d)}$ of the top level of a tree with d join levels.

Setup Let us give the LWE specifics on digits, labels and the filters used.

Digits and Types. Without filters, all levels below the top level use the digits $\{-2, \ldots, 2\}$ for ternary keys, as in the variant Rep-2 of [38]. For $B(2)$ and $U(2)$ we use $\{-3, \ldots, 3\}$ and for $B(3)$ the digits $\{-4, \ldots, 4\}$, that is, one digit more in each direction than the key has. For $B(3)$ this set is larger than the digits $\{-3, \ldots, 3\}$ of [29]. With filters, larger digit sets lower the exponent further, and we use $\{-3, \ldots, 3\}$ for ternary keys and $\{-5, \ldots, 5\}$ for $B(2)$, $B(3)$ and $U(2)$.

Labels and Lists of Level 0. The label of a vector $\mathbf{x}$ is $\phi(\mathbf{x}) = \mathbf{A}\mathbf{x} \bmod q$ and the target is $\mathbf{b}$, which the label of the key matches only up to the error, $\phi(\mathbf{s}) = \mathbf{b}-\mathbf{e}$. Since q is polynomial in n, every coordinate of the label carries $\log q$ bits, so the constraints of all levels together fix only $\mathcal{O}(n/\log q) = o(n)$ coordinates. The algorithm guesses the error on these coordinates, which costs a factor $2^{o(n)}$ since each coordinate of $\mathbf{e}$ takes one of $2\eta+1$ values, and uses $\mathbf{b}$ minus the guessed error as the target $\mathbf{t}$ of Algorithm 1. The lists of level 0 are built by meet-in-the-middle (Eq. (2)).

Joins and Filters. As for subset sum, the top join matches the remaining label coordinates exactly, since the error is guessed on them. The conditions of Section 3.3 thus apply without change. Every join, including the top join, may use

		this work					
key	published	no filters	d	with filters	d	with filters and partitioning	d
ternary, $\omega = 0.3$	0.2893 [27]	0.2884	5	0.2787	5	**0.2505**	10
ternary, $\omega = 0.375$	0.3070 [27]	0.3042	5	0.2889	7	**0.2596**	14
ternary, $\omega = 0.441$	0.3182 [27]	0.3139	5	0.2924	11	**0.2647**	15
ternary, $\omega = 0.5$	0.3218 [27]	0.3193	5	0.2935	17	**0.2684**	16
ternary, $\omega = 0.62$	0.3308 [27]	0.3287	5	0.2949	20	**0.2741**	20
ternary, $\omega = 0.667$	0.3392 [27]	0.3320	5	0.2955	20	**0.2759**	20
$B(2)$ (Kyber768/1024)	0.357 [29]	0.3397	7	0.2997	17	**0.2761**	17
$B(3)$ (Kyber512)	0.371 [29]	0.3458	8	0.3084	13	**0.3006**	13
$U(2)$ (Dilithium)	0.378 [29]	0.3620	8	0.3107	14	**0.2954**	14

Table 3. Time exponents for the enumeration of LWE keys, with depth $d \leq 20$.

a filter. These are the same filters as for subset sum (Section 5), with opposite tilts, one price per digit and a free center size.

Numerical Results We solve the optimization problem (9) without a memory bound, and require the correctness constraint (5), so that every level is built once. The trees without filters are as in Section 5, and so are the variables, including the prices and center sizes of the filters. As for subset sum, we also optimized over the depth d until the rounded exponent did not improve anymore or we reached the cap of $d = 20$. The trees with filters and partitioning for $B(\eta)$ and $U(2)$ use the depths found optimal with filters only.[1]

In Table 3 we state the time exponents for all key distributions, next to the best published values of [27] for ternary keys[2] and of [29] for $B(\eta)$ and $U(2)$. The columns d give the depth of the tree behind each exponent. Overall, we obtain large gains across all distributions, for example for ternary keys at $\omega = 0.3$ we improve the exponent from 0.2893 to 0.2505, and for $U(2)$ from 0.3620 to 0.2954. We note that already our re-optimization without filters improves on all published results. For [38,29] this stems from those works fixing the constraint size of every level to its number of representations, whereas in our trees the constraint sizes are optimization variables, limited only by the saturation constraint (4) and the correctness constraint (5). In addition, our trees use larger digit sets and correspondingly deeper trees.

[1] At the time of writing, the experiments with larger depth are still running. A later revision will deliver these numbers.

[2] For some weights [35] states lower exponents, obtained from a tree that joins three lists at a time. Its analysis counts only the matching triples and the list sizes, not the pairs from two of the lists that must be enumerated to find the triples, so we could not reproduce these exponents.

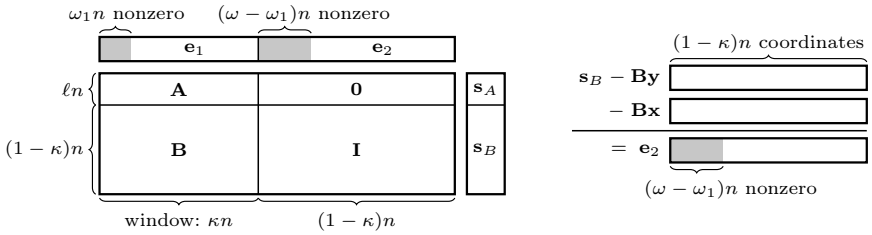


Fig. 5. The transformed parity-check matrix and the filtered top join.

7 Application: Decoding Random Linear Codes

The fastest known information set decoders for random linear codes over $\mathbb{F}_2$ combine a representation tree with joins that search for pairs of vectors of a given small Hamming distance [40,13,14]. Such a join is a filtered join (Section 4) that matches no label bits and only checks the pairs in the two buckets of every center for consistency, in time that is quadratic in the bucket size. We call it a *nearest-neighbor join* in the following. The integration of these filters over $\mathbb{F}_q$ for $q \geq 3$ has been attempted in several prior works [31,30,36], with limited success. In this work we provide the first integration that beats Meurer's BJMM adaptation [41] for all $3 \leq q \leq 27$ at half distance and for all $3 \leq q \leq 16$ at full distance, and that matches it for $q = 27$ at full distance.

The Decoding Problem. A random linear code of length n and dimension Rn over $\mathbb{F}_q$ is given by a uniformly random parity-check matrix $\mathbf{H} \in \mathbb{F}_q^{(1-R)n \times n}$, where R is the *rate* of the code. Throughout, q is a constant. We call the number of nonzero entries of a vector its *weight* and denote it by $\mathrm{wt}(\cdot)$.

Definition 3 (Syndrome Decoding Problem). *Let* $\mathbf{H} \in \mathbb{F}_q^{(1-R)n \times n}$ *be drawn uniformly at random. For a random* error $\mathbf{e} \in \mathbb{F}_q^n$ *of weight* ωn*, let* $\mathbf{s} := \mathbf{He}$ *be its* syndrome*. The* syndrome decoding problem *is, given* $(\mathbf{H}, \mathbf{s})$ *and* ω*, to find any* $\mathbf{e}' \in \mathbb{F}_q^n$ *of weight* ωn *satisfying* $\mathbf{He}' = \mathbf{s}$*.*

The number of vectors of weight xn in $\mathbb{F}_q^n$ is $q^{H_q(x)n}$ up to polynomial factors, with the q-ary entropy function

$$H_q(x) := x \log_q(q-1) - x \log_q x - (1-x) \log_q(1-x).$$

In *full-distance* decoding the weight is $\omega = H_q^{-1}(1-R)$, the relative Gilbert–Varshamov distance of the code, and in *half-distance* decoding it is half of this value. The works [41,31,30] state running times as $q^{Fn} = 2^{(F \log q)n}$, so their exponent F corresponds to the exponent $F \log q$ in this work.

7.1 From Decoding to a Representation Tree

Information set decoding permutes the columns of $\mathbf{H}$ at random and transforms the permuted matrix by row operations into the matrix of Fig. 5 (left), where

$\mathbf{A}$ has ℓn rows, $\mathbf{B}$ has $(1-\kappa)n$ rows, both have κn columns, and $\kappa := R + \ell$. The same row operations turn the syndrome into $(\mathbf{s}_A, \mathbf{s}_B)$. We call the first κn columns the *window* and the $(1-\kappa)n$ coordinates of $\mathbf{s}_B$ the *syndrome part*. Let $\mathbf{e}_1$ denote the part of the permuted error on the window and $\mathbf{e}_2$ the part on the remaining $(1-\kappa)n$ columns, the syndrome part. Then $\mathbf{He} = \mathbf{s}$ reads $\mathbf{Ae}_1 = \mathbf{s}_A$ and $\mathbf{Be}_1 + \mathbf{e}_2 = \mathbf{s}_B$. The decoder searches for an $\mathbf{e}_1$ of weight $\omega_1 n$, where $\omega_1 \leq \omega$ is a parameter. Such an $\mathbf{e}_1$ exists if the permutation places exactly $\omega_1 n$ of the ωn nonzero entries of $\mathbf{e}$ on the window, and then $\mathbf{e}_2 = \mathbf{s}_B - \mathbf{Be}_1$ has weight $(\omega - \omega_1)n$. Finding $\mathbf{e}_1$ is therefore a problem of the form of Section 2, with the matrix $\mathbf{A}$, the label $\phi(\mathbf{x}) := \mathbf{Ax}$ of a vector $\mathbf{x} \in \mathbb{F}_q^{\kappa n}$ and the target $\mathbf{s}_A$. We solve it with the tree of Section 3. The vectors of the tree have length κn, and types and joint types are fractions of these positions. All exponents remain exponents per n, so every quantity on the window, $D^{(i)}$, $r^{(i)}$, $c^{(i)}$ and the exponents of window filters, is multiplied by κ, and the exponents of the syndrome filter by $1-\kappa$.

Number of Runs. The decoder runs the tree once per permutation until a permutation places exactly $\omega_1 n$ nonzero entries of $\mathbf{e}$ on the window. A random permutation does so with probability

$$\mathcal{P} := \binom{\kappa n}{\omega_1 n}\binom{(1-\kappa)n}{(\omega-\omega_1)n} \Big/ \binom{n}{\omega n} \simeq 2^{-\left(h(\omega) - \kappa\, h(\frac{\omega_1}{\kappa}) - (1-\kappa)\, h(\frac{\omega-\omega_1}{1-\kappa})\right)n}, \quad (21)$$

so the tree is run $\mathcal{P}^{-1}$ times in expectation.

Random Column Scalings. To find $\mathbf{e}_1$ via Algorithm 1 we have to fix its top level type. We fix this type to be the type in which each nonzero digit appears equally often, i.e., $\big(1-\omega_1/\kappa, \omega_1/\big(\kappa(q-1)\big), \ldots, \omega_1/\big(\kappa(q-1)\big)\big)$. To ensure this type is met, together with the permutation, the decoder multiplies every column of $\mathbf{H}$ by a fresh uniformly random nonzero element of $\mathbb{F}_q$. If column k is multiplied by c_k, the error of the new matrix has the entry e_k/c_k, so its nonzero entries are independent and uniformly random nonzero digits, for every $\mathbf{e}$. Hence, since q is constant, $\mathbf{e}_1$ contains every nonzero digit equally often with probability $1/\mathrm{poly}(n)$, and the number of runs remains $\mathcal{P}^{-1}$ up to a polynomial factor.

Filters on the Window and on the Syndrome Part. Every join of the tree may be a filtered join (Section 4.2), and we call its filter a *window filter*. A pair $(\mathbf{x}, \mathbf{y})$ of the two lists of level $d-1$ gives the solution $\mathbf{e}_1 = \mathbf{x} + \mathbf{y}$ if it is consistent, if it matches the target, i.e., $\mathbf{A}(\mathbf{x}+\mathbf{y}) = \mathbf{s}_A$, and if $\mathbf{e}_2 = \mathbf{s}_B - \mathbf{B}(\mathbf{x}+\mathbf{y})$ has weight $(\omega-\omega_1)n$. The last condition concerns the vectors $\mathbf{Bx}$ and $\mathbf{s}_B - \mathbf{By}$, which differ exactly in the nonzero entries of $\mathbf{e}_2$ (Fig. 5, right). To find such pairs, the top join may apply a second filter, which acts on the digits of $\mathbf{Bx}$ and $\mathbf{s}_B - \mathbf{By}$ (the syndrome part) rather than the digits of $\mathbf{x}$ and $\mathbf{y}$ (the window), as for subset sum and LWE. We call it a *syndrome filter*. The vectors of the top join then consist of two parts, the window and the syndrome part, each with its own filter (similar to the description in Section 4.4), and the join matches the remaining $\ell^{(d)} n$ label bits.

Algorithm 3 states the pseudocode of the decoder.

Algorithm 3 Decoding with a Filtered Tree

Require: parity-check matrix $\mathbf{H} \in \mathbb{F}_q^{(1-R)n \times n}$, syndrome $\mathbf{s}$, weight ωn
parameters ℓ, ω_1, tree parameters $J^{(1)}, \dots, J^{(d)}$, $\ell^{(0)}, \dots, \ell^{(d-1)}$,
one window filter per join and the syndrome filter of the top join
Ensure: $\mathbf{e} \in \mathbb{F}_q^n$ of weight ωn with $\mathbf{He} = \mathbf{s}$

repeat ▷ $\mathcal{P}^{-1}$ *times in expectation, Eq.* (21)
  permute and scale the columns of $\mathbf{H}$ at random, compute $\mathbf{A}$, $\mathbf{B}$, $\mathbf{s}_A$, $\mathbf{s}_B$
  build $\mathcal{L}_1^{(d-1)}$, $\mathcal{L}_2^{(d-1)}$ with labels $\mathbf{Ax}$ and target $\mathbf{s}_A$, ▷ *Algorithm 1*
    where every join is a filtered join with its window filter ▷ *Algorithm 2*
  keep only the $\mathbf{x}$ with $\mathbf{Bx}$ and the $\mathbf{y}$ with $\mathbf{s}_B - \mathbf{By}$ of type $(1/q, \dots, 1/q)$
  find the pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{L}_1^{(d-1)} \times \mathcal{L}_2^{(d-1)}$ with
    joint type $J^{(d)}$ on the window, ▷ *window filter*
    $\mathrm{wt}\big((\mathbf{s}_B - \mathbf{By}) - \mathbf{Bx}\big) = (\omega - \omega_1)n$, ▷ *syndrome filter*
    $\mathbf{A}(\mathbf{x} + \mathbf{y}) = \mathbf{s}_A$ on the remaining $\ell^{(d)} n$ label bits
  if a pair $(\mathbf{x}, \mathbf{y})$ was found **then**
    return $\big(\mathbf{x} + \mathbf{y},\ \mathbf{s}_B - \mathbf{B}(\mathbf{x} + \mathbf{y})\big)$ with scalings and permutation undone
until a solution is returned

Cost of Algorithm 3. Algorithm 3 repeats the tree construction in expectation $\mathcal{P}^{-1}$ times, so with the time exponent τ of the tree (Eq. (8)) its running time is $\mathcal{P}^{-1} 2^{\tau n}$, which has the exponent

$$\tau + \frac{1}{n} \log \mathcal{P}^{-1}. \tag{22}$$

The optimization problem (9) minimizes this exponent instead of τ, with ℓ and ω_1 as additional variables.

As in Section 4.4, the center count of the top join is $\alpha^{(d)} = \kappa\, \alpha_{\mathrm{w}} + (1 - \kappa)\, \alpha_{\mathrm{s}}$, where α_{w} is the center count of its window filter and α_{s} that of its syndrome filter. The same holds for the list reductions $\beta_x^{(d)}$ and $\beta_y^{(d)}$. In the following we allow for window, syndrome or window and syndrome filters at the top, where a filter that is not used contributes zero. The cost $T^{(d)}$ of the top join is then given by Eq. (15) with these exponents.

Before we state the setup for decoding over $\mathbb{F}_q$, the next section derives the syndrome filter, which is a nearest-neighbor search over $\mathbb{F}_q$.

7.2 Nearest Neighbors over $\mathbb{F}_q$ as a Filtered Join

In this section we show that finding close pairs of vectors from two lists is a special case of a filtered join and derive its complexity formula over $\mathbb{F}_q$. As in Section 4 we omit the level index, and i and j index digits. The top join in Algorithm 3 on the syndrome part solves the following problem [40,31].

Definition 4 (Nearest-Neighbor Problem over $\mathbb{F}_q$). *Let $0 < \gamma < 1 - 1/q$ and $L > 0$ be constants. Given two lists $\mathcal{L}_x, \mathcal{L}_y \subseteq \mathbb{F}_q^{n'}$ of $2^{Ln'}$ uniformly random vectors each, the* nearest-neighbor problem over $\mathbb{F}_q$ *is to find the pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{L}_x \times \mathcal{L}_y$ at Hamming distance $\gamma n'$, that is, with $\mathrm{wt}(\mathbf{x} - \mathbf{y}) = \gamma n'$.*

In our algorithm, the two lists consist of the vectors $\mathbf{Bv}$ for $\mathbf{v} \in \mathcal{L}_1^{(d-1)}$ and $\mathbf{s}_B - \mathbf{Bw}$ for $\mathbf{w} \in \mathcal{L}_2^{(d-1)}$, of length $n' = (1-\kappa)n$. The vectors $\mathbf{Bv}$ and $\mathbf{s}_B - \mathbf{Bw}$ are uniformly random vectors, and such vectors have exactly the type $(1/q, \dots, 1/q)$ with probability $1/\mathrm{poly}(n)$. Since the filters of Section 4 are defined for lists of one type, we keep in both lists only the vectors of this type (see line 4 of Algorithm 3), which changes all counts by polynomial factors only.

The Used Joint Type. The vector $\mathbf{s}_B - \mathbf{Bw} = \mathbf{Bv} + \mathbf{e}_2$ differs from $\mathbf{Bv}$ exactly on the support of $\mathbf{e}_2$, which consists of $\gamma n'$ coordinates. On each of the other $(1-\gamma)n'$ coordinates both vectors have the same digit, and this digit is i with probability $1/q$. On each coordinate of the support the two digits differ by the entry of $\mathbf{e}_2$, which is a uniformly random nonzero digit by the column scalings and independent of $\mathbf{Bv}$. There, every pair (i, j) of different digits thus has the same probability $1/(q(q-1))$. With probability $1/\mathrm{poly}(n)$ every pair of digits occurs exactly as often as expected, and the pair $(\mathbf{Bv}, \mathbf{s}_B - \mathbf{Bw})$ then has the joint type J_γ with

$$(J_\gamma)_{ii} = \frac{1-\gamma}{q} \qquad \text{and} \qquad (J_\gamma)_{ij} = \frac{\gamma}{q(q-1)} \quad \text{for } i \neq j. \tag{23}$$

Given J_γ, the filtered join of Section 4.2 without label bits solves the problem of Definition 4. The general definition of these filters then introduces $2q-1$ free parameters for one join, namely one for each digit and list, up to a common factor (Section 4.3). In the following we further simplify those choices to reduce the number of optimization variables and to obtain closed-form formulas for α and β to simplify the running time exponent.

A Simplified Filter. Since $(J_\gamma)_{ii} > 1/q^2$ for $\gamma < 1 - 1/q$, a digit of $\mathbf{x}$ makes the same digit of $\mathbf{y}$ more likely than in a pair of independent vectors, and we use the same tilt on both lists (Section 4.3). Recall from Eq. (18) that a center of un' positions then contains the fraction $\theta_{ij} = g\, w_i w_j/(1 + g\, w_i w_j)$ of the positions in which $\mathbf{x}$ has digit i and $\mathbf{y}$ has digit j, where $w_i = 2^{z_i}$ for the price z_i of digit i and the factor $g > 0$ is determined by the center size through $\sum_{i,j} (J_\gamma)_{ij} \theta_{ij} = u$ (see Eq. (12)). In the following we restrict to filters that single out one digit, say 0. That is, the price z_0 is a variable and all other prices are $z_i = 0$.[3] The fraction θ_{ij} then depends only on the number $t \in \{0, 1, 2\}$ of zeros among the digits i and j, in which case we can write $\theta_t := g\, 2^{tz_0}/(1 + g\, 2^{tz_0})$ with $\theta_{ij} = \theta_t$ if and only if i and j together contain t zeros.

Let ν_t be the fraction of the positions at which exactly t of the two digits are zero. By Eq. (23), both digits are zero on the fraction $\nu_2 = (J_\gamma)_{00} = \frac{1-\gamma}{q}$ of the positions. Exactly one digit is zero for the $2(q-1)$ pairs $(0, j)$ and $(j, 0)$ with $j \neq 0$, so $\nu_1 = 2(q-1) \cdot \frac{\gamma}{q(q-1)} = \frac{2\gamma}{q}$, and the remaining positions make

[3] A numerical analysis with general filters has in all instances converged to this setting, similar to the observations in [31]. We therefore state this special case to obtain compact closed formulas for the running time.

up $\nu_0 = 1 - \nu_2 - \nu_1 = 1 - \frac{1+\gamma}{q}$. This yields a center size of $u = \sum_{i,j}(J_\gamma)_{ij}\theta_{ij} = \sum_{t=0}^{2}\nu_t\theta_t$. Now with the center count $\alpha = K_u(\theta)$ of Eq. (13) we obtain

$$\alpha = h(u) - \sum_{i,j}(J_\gamma)_{ij}\, h(\theta_{ij}) = h(u) - \sum_{t=0}^{2}\nu_t\, h(\theta_t). \tag{24}$$

The list reduction (Eq. (17)) needs the fraction θ_i^x of the positions with digit i in $\mathbf{x}$ that lie inside the center. From the used joint type (see Eq. (23)) we have that among the positions where $\mathbf{x}$ has the digit 0, $\mathbf{y}$ has the digit 0 on a fraction $1-\gamma$ and a nonzero digit on a fraction γ, so $\theta_0^x = (1-\gamma)\theta_2 + \gamma\theta_1$. Among the positions where $\mathbf{x}$ has a digit $i \neq 0$, $\mathbf{y}$ has the digit 0 on a fraction $\frac{\gamma}{q-1}$, so $\theta_i^x = \big(1 - \frac{\gamma}{q-1}\big)\theta_0 + \frac{\gamma}{q-1}\theta_1$. Every digit has the share $1/q$ in $\mathbf{x}$, and both lists have the same tilt, so $\beta := \beta_x = \beta_y$.

$$\beta = h(u) - \frac{1}{q}\, h\big((1-\gamma)\theta_2 + \gamma\theta_1\big) - \frac{q-1}{q}\, h\Big(\Big(1 - \frac{\gamma}{q-1}\Big)\theta_0 + \frac{\gamma}{q-1}\,\theta_1\Big). \tag{25}$$

The filter thus has the parameters z_0 and u, or equivalently z_0 and g, and the following proposition states the cost of the join.

Proposition 1 (Nearest Neighbors over $\mathbb{F}_q$). *Let γ, L and the lists $\mathcal{L}_x, \mathcal{L}_y$ be as in Definition 4, with both lists restricted to their vectors of type $(1/q, \ldots, 1/q)$. Under Heuristic 1, all pairs of joint type J_γ with one vector from each list can be found in time $2^{(\tau_{\mathrm{nn}}+o(1))n'}$, where*

$$\tau_{\mathrm{nn}} = \tau_{\mathrm{nn}}(L,\gamma) := \min_{z_0,\, g} \max\big(L + \alpha - \beta,\ 2L + \alpha - 2\beta\big)$$

with α from Eq. (24) and β from Eq. (25).

Proof. By Lemma 2(b) every choice of z_0 and g gives a same-tilt filter with the center count α of Eq. (24) and the list reduction β of Eq. (25). The two terms are the bucketing term and the checking term of Eq. (15) for a join without label bits and $\beta_x = \beta_y = \beta$, and Theorem 2 attains this cost. □

Comparison with the Analysis of Hirose. Hirose [31, Algorithm 1] gave a similar algorithm for nearest-neighbor search over $\mathbb{F}_q$. In Table 4 we compare the exponent of the analysis of Hirose [31, Theorem 1] with that of Proposition 1. We observe that Proposition 1 obtains a lower exponent for every $q \geq 3$ in the table and the same exponent only for $q = 2$. Translated to our analysis, the difference lies in the center count. The center count α_{H} of [31] considers only the centers that contain the fraction u of every group in which $\mathbf{x}$ and $\mathbf{y}$ differ, while α is the minimum over all admissible fractions (Eq. (14)). Embedding this nearest-neighbor translation into Stern's ISD algorithm [48] we obtain, in contrast to [31], improvements also for $q > 2$ (we give details in Appendix E).

7.3 Setup

Let us give the specifics of decoding over $\mathbb{F}_q$ on digits, labels and the filters used.

q	L	γ	no filter	Hirose [31]	Proposition 1
2	0.10	0.05	0.20	**0.1056**	**0.1056**
3	0.07925	0.10	0.1585	0.0882	**0.0858**
3	0.1585	0.40	0.3170	0.2841	**0.2362**
4	0.20	0.20	0.40	0.2544	**0.2332**
8	0.15	0.10	0.30	0.1650	**0.1583**

Table 4. Time exponents τ_{nn} of the nearest-neighbor join over $\mathbb{F}_q$.

Digits and Types. The digits are the elements of $\mathbb{F}_q$, and sums of digits are taken in $\mathbb{F}_q$ (Section 3.1). A nearest-neighbor join only filters for the weight of a vector. The weight does not distinguish between the $q-1$ nonzero digits a coordinate can have. We therefore use joint types whose entries $J^{(i)}_{jj'}$ depend only on which of the digits j, j' and $j+j'$ are zero. This leaves five cases: both digits are zero, only j is zero, only j' is zero, both are nonzero with sum zero, and both are nonzero with nonzero sum. Since the joint type is symmetric (Section 4.3), its entries sum to one (Section 3.1) and the weight of the sum is fixed by the level above, such a joint type is determined by two variables for $q \geq 3$.

This implies that all nonzero digits have the same share in the type of a level, so a type is determined by the weight, for example $p_0^{(d)} = 1 - \omega_1/\kappa$ and $p_j^{(d)} = \omega_1/(\kappa(q-1))$ for $j \neq 0$ on the top level. Since q is constant, the number of vectors of this type is, up to polynomial factors, the number of all vectors of weight $\omega_1 n$, and likewise for the levels below. The tree thus requires that in $\mathbf{e}_1$ all nonzero digits appear equally often, which we ensure via the column scalings of Section 7.1.

Labels and Lists of Level 0. The label $\mathbf{Ax}$ has ℓn coordinates of $\log q$ bits each (Section 2), so that $\ell^{(0)} + \cdots + \ell^{(d)} = \ell_{\max} = \ell \log q$. In contrast to subset sum, the label length $\ell \leq 1 - R$ is a parameter. The lists of level 0 are built by meet-in-the-middle (Eq. (2)).

Joins and Filters. The top join uses a syndrome filter and no window filter, and all other joins of the tree are joins without filter that naively ensure the defined weight on the window. Therefore, the top join in Algorithm 3 is the filtered join of Algorithm 2 on the lists of the vectors $\mathbf{Bv}$ and $\mathbf{s}_B - \mathbf{Bw}$, with the filter from the previous section for $\gamma = \frac{\omega-\omega_1}{1-\kappa}$ as its syndrome filter, and it matches the remaining $\ell^{(d)} n$ label bits exactly. The lists have size $2^{L^{(d-1)}n}$. By Eq. (15) the top join costs

$$T^{(d)} = \min_{z_0,\, g} \max\Big(L^{(d-1)} + (1-\kappa)(\alpha-\beta),\ 2L^{(d-1)} - \ell^{(d)} + (1-\kappa)(\alpha-2\beta)\Big), \quad (26)$$

which replaces Eq. (7) for the top join in Eq. (8) and requires $\gamma < 1 - 1/q$. For $\ell^{(d)} = 0$ this is $(1-\kappa)\,\tau_{\mathrm{nn}}(L,\gamma)$ with $L = \frac{L^{(d-1)}}{1-\kappa}$ (Proposition 1). Every pair $(\mathbf{v}, \mathbf{w})$ that the join finds is checked for the joint type $J^{(d)}$ on the window naively.

	half distance			full distance		
		this work			this work	
q	published	no filters	with filters	published	no filters	with filters
2	0.0494 [10]	0.0494	0.0473	**0.0951** [14]	0.1019	0.0953
3	0.0763 [41]	0.0751	**0.0730**	0.1634 [41]	0.1598	**0.1561**
4	0.0931 [41]	0.0920	**0.0900**	0.2028 [41]	0.1995	**0.1972**
5	0.1053 [41]	0.1042	**0.1025**	0.2321 [41]	0.2294	**0.2278**
7	0.1221 [41]	0.1212	**0.1201**	0.2744 [41]	0.2724	**0.2718**
8	0.1284 [41]	0.1275	**0.1266**	0.2907 [41]	0.2888	**0.2884**
11	0.1423 [41]	0.1416	**0.1410**	0.3270 [41]	0.3262	**0.3261**
16	0.1569 [41]	0.1564	**0.1560**	0.3672 [41]	**0.3667**	**0.3667**
27	0.1744 [41]	0.1741	**0.1740**	**0.4164** [41]	**0.4164**	**0.4164**

Table 5. Worst-case time exponents for decoding over $\mathbb{F}_q$. For $q \geq 3$ the column published is the BJMM adaptation of [41] with one level of representations. [41, Table 7.1] states its exponent at full distance for $q = 4, 8, 16$, and the other entries are our evaluation of it. Published exponents F of a time q^{Fn} are stated as $F \log q$.

Remark 1 (Join Architecture). Note that this join architecture is different from the one found optimal in the $\mathbb{F}_2$ setting, where pure nearest-neighbor joins on independent blocks of the syndrome are optimal [14,16,24]. However, those joins yield only a tiny speedup over joins without filter on lower levels [40,13], which we find to fully disappear for $q \geq 3$.

7.4 Numerical Results

We optimize the running time exponent of Algorithm 3 in Eq. (22) in the half- and full-distance setting. For all instances depth 2 is optimal, except for $q \leq 4$ at full distance, where depth 3 is optimal. The variables are the joint types and constraint sizes of the tree, ℓ, ω_1 and, for a filtered top join, the center size (via g) and one price z_0 of the digit 0. We report the *worst-case exponent*, i.e., the largest exponent over a grid of rates with step 0.002.

Reproduction of Published Values. Our model contains several previous algorithms as special cases. For $q = 2$ we obtain BJMM [10] when restricting to a top join without filter ($z_0 = 0$) and the algorithm of May and Ozerov [40] for a nearest-neighbor top join ($\ell^{(d)} = 0$), and its depth-optimized version from [13] for a nearest-neighbor top join with $d = 3$. Note that the best algorithm of Both and May [14] for $q = 2$ is not an instance of Algorithm 3 as stated, since Algorithm 3 uses a syndrome filter at the top join only. Over $\mathbb{F}_q$ our algorithm contains Meurer's BJMM adaptation [41], which is the best known $\mathbb{F}_q$ decoder for $q \geq 3$ to date, as a special case when using top joins without filter.

Results. In Table 5 we state the worst-case exponents at half and at full distance, next to those of previously known algorithms, where bold marks the lowest

known exponent for the q of each row. Where our algorithms contain previous algorithms as special cases we re-optimized those exponents in our model.

In our re-optimization we found improvements over [41] in the full-distance setting for $q = 4, 8, 16$ already without filters by increasing to depth 2, while Meurer only evaluated depth 1.

On top of a representation tree, the filtered top join lowers the half-distance exponent in comparison to the best known exponents for every $q \geq 3$ in the table, while the gain decreases for increasing q. The diminishing gains are a consequence of the known fact that representation trees, and enumeration strategies in general, become less favorable for growing q [41]. At full distance it lowers the exponent for all $3 \leq q \leq 16$, for $q = 16$ only through the larger depth. Since the optimal instantiations use no filter on lower joins, we obtain no gain from partitioning.

Further $\mathbb{F}_q$ Decoders. The decoders of [36], a Both–May adaptation to $\mathbb{F}_q$, and of [30], BJMM with a nearest-neighbor top join, both use the nearest-neighbor join of [31]. The exponents stated in [36] lie above those of [41], and as already observed in [36], evaluating [30, Theorem 1] at the parameters of its Tables 3 and 4 gives exponents above those stated there and above those of Table 5.

8 Further Decoding Settings

We apply Algorithm 3 with small changes to three further decoding problems and give the details in Appendices F to H.

Ternary Decoding with Large Weight. Decoding over $\mathbb{F}_3$ an error of weight ωn with $\omega > 2/3$ underlies the signature scheme Wave [18,5]. As in [15,43], the decoder shifts the error on the window by the all-one vector so that the tree builds a sparse vector, and since such instances have many solutions, some levels use the joins of Wagner's algorithm [49], which add two vectors with disjoint supports. The joint types are free, and at the hardest instance the exponent drops from 0.243 [43] to 0.2148 (Table 8).

Decoding with Restricted Errors. In restricted syndrome decoding [2], which underlies the signature scheme CROSS [4], the nonzero entries of the error lie in a subgroup $E \subset \mathbb{F}_q^*$ of order z. The decoder scales the columns by random elements of E, and its lists use the digit 0 and the sums and differences of two elements of E, with free joint types and a window filter at every join. For $q = 157$, $R = 0.5$, $z = 2$ and $\omega = 0.8$ the exponent drops from 0.2517 [3] to 0.1931 (Table 9).

Decoding in the Lee Metric. Over $\mathbb{Z}_q$ the Lee weight of a digit j is $\min(j, q - j)$, and the Lee weight of a vector is the sum over its entries [50]. The decoder searches for an error of the most frequent type of the given Lee weight, in which the share of a digit falls by a constant factor with every unit of Lee weight, and

it scales no columns, since scaling changes the Lee weight. The syndrome filter gives every digit a price proportional to its Lee weight, every join also uses a window filter, and for $q = 47$ at full distance the worst-case exponent drops from 0.660 [7] to 0.3580 (Table 10).

Acknowledgments. The author used AI assistance (innovium.ai) to support the analysis and conduct the implementation, as well as to improve the writing, style and flow of this paper.

References

1. Andoni, A., Laarhoven, T., Razenshteyn, I., Waingarten, E.: Optimal hashing-based time-space trade-offs for approximate near neighbors. In: SODA 2017. pp. 47–66 (2017)
2. Baldi, M., Battaglioni, M., Chiaraluce, F., Horlemann-Trautmann, A.L., Persichetti, E., Santini, P., Weger, V.: A new path to code-based signatures via identification schemes with restricted errors. arXiv:2008.06403 (2020)
3. Baldi, M., Bitzer, S., Pavoni, A., Santini, P., Wachter-Zeh, A., Weger, V.: Generic decoding of restricted errors. In: IEEE International Symposium on Information Theory (ISIT) (2023), arXiv:2303.08882
4. Baldi, M., et al.: CROSS: Codes and restricted objects signature scheme. Submission to the NIST call for additional post-quantum signatures, https://www.cross-crypto.com (2023)
5. Banegas, G., Carrier, K., Chailloux, A., Couvreur, A., Debris-Alazard, T., Gaborit, P., Karpman, P., Loyer, J., Niederhagen, R., Sendrier, N., Smith, B., Tillich, J.P.: WAVE. Submission to the NIST call for additional post-quantum signature schemes, Round 1 (2023)
6. Bariffi, J., Bartz, H., Liva, G., Rosenthal, J.: On the properties of error patterns in the constant Lee weight channel. In: International Zurich Seminar on Information and Communication (IZS). pp. 44–48 (2022), arXiv:2110.01878
7. Bariffi, J., Khathuria, K., Weger, V.: Information set decoding for Lee-metric codes using restricted balls. arXiv:2205.12903 (2022)
8. Becker, A., Coron, J.S., Joux, A.: Improved generic algorithms for hard knapsacks. In: EUROCRYPT 2011. LNCS, vol. 6632, pp. 364–385 (2011)
9. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. In: SODA 2016. pp. 10–24 (2016)
10. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In: EUROCRYPT 2012. LNCS, vol. 7237, pp. 520–536 (2012)
11. Bonnetain, X., Bricout, R., Schrottenloher, A., Shen, Y.: Improved classical and quantum algorithms for subset-sum. In: ASIACRYPT 2020. LNCS, vol. 12492, pp. 633–666 (2020)
12. Bos, J.W., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS – Kyber: A CCA-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367 (2018)
13. Both, L., May, A.: Optimizing BJMM with nearest neighbors: Full decoding in $2^{2n/21}$ and McEliece security. In: WCC 2017 – The Tenth International Workshop on Coding and Cryptography (2017)

14. Both, L., May, A.: Decoding linear codes with high error rate and its impact for LPN security. In: PQCrypto 2018. LNCS, vol. 10786, pp. 25–46 (2018)
15. Bricout, R., Chailloux, A., Debris-Alazard, T., Lequesne, M.: Ternary syndrome decoding with large weight. In: SAC 2019. LNCS, vol. 11959, pp. 437–466 (2019)
16. Carrier, K., Debris-Alazard, T., Meyer-Hilfiger, C., Tillich, J.P.: Statistical decoding 2.0: Reducing decoding to LPN. In: ASIACRYPT 2022. LNCS, vol. 13794, pp. 477–507 (2022)
17. Chi-Domínguez, J.J., Esser, A., Kunzweiler, S., May, A.: Low memory attacks on small key CSIDH. In: ACNS 2023. LNCS, vol. 13906, pp. 276–304 (2023)
18. Debris-Alazard, T., Sendrier, N., Tillich, J.P.: Wave: A new family of trapdoor one-way preimage sampleable functions based on codes. In: ASIACRYPT 2019. LNCS, vol. 11921, pp. 21–51 (2019)
19. Devadas, S., Ren, L., Xiao, H.: On iterative collision search for LPN and subset sum. In: TCC 2017. LNCS, vol. 10678, pp. 729–746 (2017)
20. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In: CRYPTO 2012. LNCS, vol. 7417, pp. 719–740 (2012)
21. Dubiner, M.: Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. IEEE Transactions on Information Theory **56**(8), 4166–4179 (2010)
22. Ducas, L., Esser, A., Etinski, S., Kirshanova, E.: Asymptotics and improvements of sieving for codes. Cryptology ePrint Archive, Report 2023/1577; EUROCRYPT 2024 (2024)
23. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems **2018**(1), 238–268 (2018)
24. Esser, A.: Revisiting nearest-neighbor-based information set decoding. In: IMACC 2023. LNCS, vol. 14421, pp. 34–54 (2023)
25. Esser, A., Kübler, R., Zweydinger, F.: A faster algorithm for finding closest pairs in Hamming metric. In: FSTTCS 2021 (2021)
26. Esser, A., May, A.: Low weight discrete logarithm and subset sum in $2^{0.65n}$ with polynomial memory. In: EUROCRYPT 2020. LNCS, vol. 12107, pp. 94–122 (2020)
27. Esser, A., Mukherjee, A., Sarkar, S.: Memory-efficient attacks on small LWE keys. Journal of Cryptology (2024), extended version of ASIACRYPT 2023
28. Esser, A., Zweydinger, F.: New time-memory trade-offs for subset sum – improving ISD in theory and practice. In: EUROCRYPT 2023. LNCS, vol. 14008, pp. 360–390 (2023)
29. Glaser, T., May, A.: How to enumerate LWE keys as narrow as in Kyber/Dilithium. Cryptology ePrint Archive, Report 2022/1337; CANS 2023 (2023)
30. Gueye, C.T., Klamti, J.B., Hirose, S.: Generalization of BJMM-ISD using May-Ozerov nearest neighbor algorithm over an arbitrary finite field $\mathbb{F}_q$. In: C2SI 2017. LNCS, vol. 10194, pp. 96–109 (2017)
31. Hirose, S.: May-Ozerov algorithm for nearest-neighbor problem over $\mathbb{F}_q$ and its application to information set decoding. In: SecITC 2016. LNCS, vol. 10006, pp. 115–126 (2016), cryptology ePrint Archive, Report 2016/237
32. Howgrave-Graham, N.: Filtered merges make deeper representation trees pay: Subset sum and Meet-LWE. Cryptology ePrint Archive, Report 2026/2802 (2026)
33. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: EUROCRYPT 2010. LNCS, vol. 6110, pp. 235–256 (2010)

34. Kulpe, A.: Time-memory tradeoffs for subset sum and decoding. Master's thesis, Ruhr-Universität Bochum; https://github.com/alexkulpe/time-memory-tradeoffs-for-subset-sum-and-decoding (2024)
35. Lee, E., Lee, J., Son, Y., Wang, Y.: Improved Meet-LWE attack via ternary trees. Cryptology ePrint Archive, Report 2024/824 (2024)
36. Li, Z., Zheng, Q., Wang, Z., Song, S.: The Both-May algorithm over finite fields: q-ary information set decoding and wild McEliece parameter reassessment. In: Inscrypt 2026. LNCS, vol. 17226, pp. 410–431. Springer (2027)
37. Liu, H., Yu, Y.: A non-heuristic approach to time-space tradeoffs and optimizations for BKW. In: ASIACRYPT 2022. LNCS, vol. 13793, pp. 741–770 (2022)
38. May, A.: How to meet ternary LWE keys. In: CRYPTO 2021. LNCS, vol. 12826, pp. 701–731 (2021)
39. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{O}(2^{0.054n})$. In: ASIACRYPT 2011. LNCS, vol. 7073, pp. 107–124 (2011)
40. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: EUROCRYPT 2015. LNCS, vol. 9056, pp. 203–228 (2015)
41. Meurer, A.: A Coding-Theoretic Approach to Cryptanalysis. Ph.D. thesis, Ruhr-Universität Bochum (2012)
42. Minder, L., Sinclair, A.: The extended k-tree algorithm. Journal of Cryptology **25**(2), 349–382 (2012)
43. Narisada, S., Okada, H., Aikawa, Y., Fukushima, K.: A generalized ISD framework for large-weight syndrome decoding. Cryptology ePrint Archive, Report 2026/1947; to appear at CANS 2026 (2026)
44. Ozerov, I.: Combinatorial Algorithms for Subset Sum Problems. Ph.D. thesis, Ruhr-Universität Bochum (2016)
45. Peters, C.: Information-set decoding for linear codes over $\mathbb{F}_q$. In: PQCrypto 2010. LNCS, vol. 6061, pp. 81–94 (2010)
46. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Springer (2003)
47. Schroeppel, R., Shamir, A.: A $t = o(2^{n/2})$, $s = o(2^{n/4})$ algorithm for certain NP-complete problems. SIAM Journal on Computing **10**(3), 456–464 (1981)
48. Stern, J.: A method for finding codewords of small weight. In: Coding Theory and Applications. LNCS, vol. 388, pp. 106–113 (1989)
49. Wagner, D.: A generalized birthday problem. In: CRYPTO 2002. LNCS, vol. 2442, pp. 288–303 (2002)
50. Weger, V., Khathuria, K., Horlemann, A.L., Battaglioni, M., Persichetti, E., Santini, P.: On the hardness of the Lee syndrome decoding problem. Advances in Mathematics of Communications (2022), arXiv:2002.12785

A Relation to the Earlier Descriptions

Earlier works [33,8,11,28,38,29,27,39,10,41,3,26,14,17], [44, Lemma 41], [15, App. A] derive the number of representations by hand for their digit set. For every digit s of the sum they list its splits $s = i + j$, parametrize how often each split is used, multiply one multinomial coefficient per digit and maximize over the parameters [11, Lemma 10]. For the digits $\{-2, \ldots, 2\}$ May calls this "a quite

involved technical analysis" [38, Sect. 8]. Since every digit pair (i, j) belongs to exactly one digit $s = i + j$ of the sum, the product of these coefficients is

$$\prod_s \binom{p_s^+ n}{(J_{ij} n)_{i+j=s}} = \binom{n}{(J_{ij} n)_{i,j}} \Big/ \binom{n}{(p_s^+ n)_s} \simeq 2^{(H(J)-H(p^+))n}, \tag{27}$$

which is Lemma 1(a) for every digit set. The fraction of consistent pairs of [11, Sect. 2.3] and [28, Eq. (4)] is Lemma 1(b). Ozerov [44, Def. 9] already describes a pair by its joint distribution, and Bricout et al. [15, App. A] describe the pairs that sum to a ternary vector by a 3×3 table.

The earlier joins keep a pair whenever its sum has the type p^+ of the next level. Up to a polynomial factor, these are the pairs of the joint type J^* of largest entropy among those with row and column sums p and anti-diagonal sums p^+. Since the entropy is strictly concave and transposition preserves these sums, J^* is symmetric, so the earlier join is our join with $J = J^*$.

B Proof of Lemma 2

Proof. Only the occurring pairs (i, j), those with $J_{ij} > 0$, appear in $K_u(\theta) = h(u) - \sum_{i,j} J_{ij}\, h(\theta_{ij})$ and in Eq. (11). We therefore treat θ as a vector in $\mathbb{R}^{\mathrm{supp}(J)}$ with one entry per occurring pair, where $\mathrm{supp}(J)$ is the set of occurring pairs, and all sums over pairs (i, j) below run over the occurring pairs. For θ with entries strictly between 0 and 1 let $r_{ij}(\theta) := \log \frac{\theta_{ij}}{1-\theta_{ij}}$. Since $h'(t) = \log \frac{1-t}{t}$, the partial derivatives of K_u at such a point are

$$\frac{\partial K_u}{\partial \theta_{ij}} = -J_{ij}\, h'(\theta_{ij}) = J_{ij}\, r_{ij}(\theta). \tag{28}$$

We call $d \in \mathbb{R}^{\mathrm{supp}(J)}$ a *direction* if

$$\sum_j J_{ij} d_{ij} = 0 \quad \text{for every } i \qquad \text{and} \qquad \sum_i J_{ij} d_{ij} = 0 \quad \text{for every } j. \tag{29}$$

The difference of two admissible fractions is a direction, and adding a direction to an admissible fraction preserves the sums of Eq. (11), hence keeps it admissible. *Part (a), existence.* The admissible vectors form a nonempty, closed and bounded set, and K_u is continuous, so K_u attains its minimum at some admissible θ.

Part (a), no entry of θ is 0 or 1. Let θ° be admissible with all entries strictly between 0 and 1, and assume that some entry of θ is 0 or 1. The points $\theta(t) = \theta + t(\theta^\circ - \theta)$ with $0 \le t \le 1$ are admissible, for $t > 0$ all their entries lie strictly between 0 and 1, and by Eq. (28)

$$\frac{\mathrm{d}}{\mathrm{d}t} K_u\big(\theta(t)\big) = -\sum_{i,j} J_{ij}\, h'\big(\theta_{ij}(t)\big)\big(\theta^\circ_{ij} - \theta_{ij}\big).$$

Let t tend to 0. For $\theta_{ij} = 0$ we have $\theta^\circ_{ij} - \theta_{ij} > 0$ and $h'(\theta_{ij}(t)) \to +\infty$, and for $\theta_{ij} = 1$ we have $\theta^\circ_{ij} - \theta_{ij} < 0$ and $h'(\theta_{ij}(t)) \to -\infty$. In both cases the summand

tends to $-\infty$, while all other summands converge. The derivative is therefore negative for small $t > 0$, which gives $K_u(\theta(t)) < K_u(\theta)$ and contradicts the minimality of θ.

Part (a), form of θ. Let d be a direction. Since all entries of θ lie strictly between 0 and 1, the vectors $\theta \pm \varepsilon d$ are admissible for small $\varepsilon > 0$. The first-order Taylor expansion of K_u at θ, with the partial derivatives of Eq. (28), gives $K_u(\theta \pm \varepsilon d) = K_u(\theta) \pm \varepsilon \sum_{i,j} J_{ij} r_{ij}(\theta) d_{ij} + O(\varepsilon^2)$. By the minimality of θ the sum is zero for every direction d. We substitute $\eta_{ij} = J_{ij} d_{ij}$. Then Eq. (29) states that all row sums $\sum_j \eta_{ij}$ and all column sums $\sum_i \eta_{ij}$ are zero, and with $r = r(\theta)$ we obtain

$$\sum_{i,j} r_{ij} \eta_{ij} = 0 \qquad \text{for every } \eta \in \mathbb{R}^{\mathrm{supp}(J)} \text{ with zero row and column sums.} \quad (30)$$

Let $R_i \in \mathbb{R}^{\mathrm{supp}(J)}$ be 1 on the occurring pairs with first digit i and 0 elsewhere, and let $S_j \in \mathbb{R}^{\mathrm{supp}(J)}$ be 1 on the occurring pairs with second digit j and 0 elsewhere. The row and column sums of η are its inner products with the R_i and the S_j, so the vectors with zero row and column sums form the orthogonal complement $V^\perp$ of the span V of all R_i and S_j. Eq. (30) states that r is orthogonal to $V^\perp$, hence $r \in V$, that is, $r = \sum_i a_i R_i + \sum_j b_j S_j$ for real numbers a_i, b_j, and

$$r_{ij} = a_i + b_j \qquad \text{for all occurring pairs } (i, j).$$

For a digit that appears in no occurring pair we set the corresponding number to 0. Solving $\log \frac{\theta_{ij}}{1-\theta_{ij}} = a_i + b_j$ for θ_{ij} yields Eq. (16) with $w_i^x = 2^{a_i}$ and $w_j^y = 2^{b_j}$.

Part (b). All entries of θ lie strictly between 0 and 1, so $u := \sum_{i,j} J_{ij} \theta_{ij}$ satisfies $0 < u < 1$, and Eq. (11) defines tilts δ^x and δ^y, each of which sums to zero since the row sums and the column sums of $(J_{ij}\theta_{ij})$ both add up to u. For this filter θ is admissible, so part (a) gives a minimizer θ^* with $r_{ij}(\theta^*) = a_i^* + b_j^*$, and by Eq. (16) we have $r_{ij}(\theta) = a_i + b_j$ for $a_i = \log w_i^x$ and $b_j = \log w_j^y$. The vector $d = \theta - \theta^*$ is a direction, so for any numbers a_i, b_j Eq. (29) gives

$$\sum_{i,j} J_{ij}(a_i + b_j) d_{ij} = \sum_i a_i \sum_j J_{ij} d_{ij} + \sum_j b_j \sum_i J_{ij} d_{ij} = 0.$$

Applying this to $r(\theta)$ and to $r(\theta^*)$ and subtracting, we obtain

$$\sum_{i,j} J_{ij} \big(r_{ij}(\theta) - r_{ij}(\theta^*)\big)\big(\theta_{ij} - \theta_{ij}^*\big) = 0.$$

Since $t \mapsto \log \frac{t}{1-t}$ is strictly increasing on $(0, 1)$, every summand is non-negative and zero only if $\theta_{ij} = \theta_{ij}^*$. Hence $\theta = \theta^*$, and $\alpha = K_u(\theta)$. □

C The Filtered Join with Product Centers

For completeness, we prove the cost of the filtered join with product centers, in the notation of Section 4.2. The join permutes the positions at random and

cuts them into $k = \lceil \sqrt{n} \rceil$ blocks of n/k positions, with N random *block centers* on every block. A *product center* has one block center per block and accepts a vector if every block center accepts its part. The join is Algorithm 2 with product centers, where the accepting product centers of a vector are enumerated block by block, and the join is repeated for several random permutations.

Theorem 2 (Filtered Join with Product Centers). *Let (u, δ^x, δ^y) be a filter with list reductions β_x, β_y and finite center count α (Eqs. (10) and (14)). Under Heuristic 1, the filtered join with product centers on $k = \lceil \sqrt{n} \rceil$ blocks outputs exactly the consistent pairs in $\mathcal{L}_x \times \mathcal{L}_y$ whose labels match on the ℓn new bits, except with probability $2^{-\Omega(n)}$, in expected time $2^{Tn+o(n)}$ with T given by Eq. (15) and expected memory $2^{Ln+o(n)}$ besides the input and the output.*

Proof. Let θ^* attain the minimum in Eq. (14), and call a permutation *good* for a consistent pair if the pair has the joint type J in every block. A block center is a random set of un/k positions of its block, and it accepts the part of a left vector if the part has type p and its restriction to the block center has type $p + \delta^x$, and analogously on the right. We first assume that all counts $J_{ij}n/k$ and $\theta^*_{ij}J_{ij}n/k$ are integers, and treat the rounding at the end.

Acceptance. Within a block the counts of Section 4.1 hold with n/k in place of n. Up to polynomial factors, a random block center therefore accepts the part of a fixed vector with probability at most $2^{-\beta_x n/k}$ or $2^{-\beta_y n/k}$, and, for a good permutation, both parts of a consistent pair with a probability $\pi \geq 2^{-\alpha n/k}$. These probabilities multiply over the blocks, whose block centers are independent.

Correctness. By Eq. (1) for the blocks and the bound $\binom{n}{(J_{ij}n)_{i,j}} \leq 2^{H(J)n}$, a random permutation is good for a fixed consistent pair with probability $\iota \geq (n+1)^{-k|\mathcal{A}|^2}$, as in [40,44]. The join uses $\lceil \xi/\iota \rceil$ permutations and $N := \lceil \xi/\pi \rceil$ block centers per block, where $\xi := 2(L+1)n$. For a fixed consistent pair, each of the following $k+1$ events has probability at most $e^{-\xi}$: no permutation is good, and, for $s = 1, \ldots, k$, block s of the first good permutation contains no block center that accepts both parts of the pair. A union bound over these events and the 2^{2Ln} pairs shows that, except with probability $(k+1)2^{-2n}$, every consistent pair is accepted by some product center, so the join outputs exactly the consistent pairs with matching labels.

Time and memory. All counts below hold up to factors $2^{o(n)}$, since $k = \lceil \sqrt{n} \rceil$ gives $N = 2^{\alpha n/k} \cdot \mathrm{poly}(n) = 2^{o(n)}$ block centers per block and $\lceil \xi/\iota \rceil = 2^{o(n)}$ permutations. The join tests each of the $kN = 2^{o(n)}$ block centers against all vectors and keeps, for each vector, the indices of the centers that accept it, at cost $2^{Ln+o(n)} \leq 2^{Tn+o(n)}$ since $T \geq L$. As for random product codes [9], it then enumerates the accepting product centers of a vector block by block, depth first over the prefixes $(C_1, \ldots, C_t)$ of block centers that accept it. Sorting the stored indices gives the extensions of a prefix, so the cost is the number of pairs of a vector and an accepting prefix. A left vector has $2^{t(\alpha-\beta_x)n/k}$ accepting prefixes of length t in expectation, at most its number $2^{(\alpha-\beta_x)n}$ of accepting product centers since $\alpha \geq \max(\beta_x, \beta_y)$ (Section 4.2), and summed over both lists this gives the

bucketing terms of Eq. (15). Under Heuristic 1 all buckets together contain $2^{(2L-\ell+\alpha-\beta_x-\beta_y)n}$ label matches in expectation, the checking term. Besides the input and the output, the join stores the indices of the accepting block centers of every vector, in expectation $k \cdot 2^{(\alpha-\min(\beta_x,\beta_y))n/k} = 2^{o(n)}$ per vector and $2^{Ln+o(n)}$ in total, and the enumeration keeps no more along its current path.

Rounding. In general a block s has $n_s \in \{\lfloor n/k \rfloor, \lceil n/k \rceil\}$ positions. Integer tables b_s with $\sum_s b_s = Jn$, $\sum_{i,j} b_{s,ij} = n_s$ and $|b_{s,ij} - J_{ij} n_s| < 1$ exist. The point $(J_{ij} n_s)$ satisfies these equations and lies between $\lfloor J_{ij} n_s \rfloor$ and $\lceil J_{ij} n_s \rceil$, and the constraint matrix is totally unimodular, as for every transportation problem, so the polytope has integral vertices [46]. Replacing $J_{ij} n/k$ by $b_{s,ij}$ and $\theta^*_{ij} J_{ij} n/k$ by $\lfloor \theta^*_{ij} b_{s,ij} \rfloor$ in the block centers and good permutations changes each block count by $O(1)$, hence each multinomial coefficient by a factor $\mathrm{poly}(n)$ and each probability above by a factor $(n+1)^{O(k)} = 2^{O(\sqrt{n} \log n)} = 2^{o(n)}$, as do the numbers of permutations and block centers, each a product of $O(k)$ single-block factors. The time and memory therefore stay $2^{Tn+o(n)}$ and $2^{Ln+o(n)}$ and the failure probability stays $2^{-\Omega(n)}$. □

Joins with Several Parts. In a join whose positions are partitioned into parts, as in Section 4.4 or at the filtered top join of Section 7 with its window and syndrome part, each part has its own joint type and filter. Each part is cut into blocks with their own block centers, a center consists of one product center per part, and α, β_x and β_y are the sums over the parts, each multiplied by the fraction of the n positions that its part covers.

D Comparison with [32]

In this section we compare our work with the concurrent work of Howgrave-Graham [32]. It adds filters to every join of the subset sum tree and of the Meet-LWE tree of [38] for ternary keys. For these two trees, [32] also describes a join by the joint type of the two vectors that are added and uses the expressions of Lemma 1 for the number of representations and for the fraction of consistent pairs.

Scope of the Analysis. The two expressions are stated in [32] for integer digits only and without proof, and the cost formulas of the two trees are taken from [11] and [38] without their derivation. We prove the expressions for an arbitrary digit set with its addition (Lemma 1), give the tree as Algorithm 1 and derive its list sizes, its constraints and its running time in Section 3.3 in a general fashion. Every application only has to state its digits, labels and particular filter configurations. In this way the same model covers LWE keys with further distributions (Section 6), decoding over $\mathbb{F}_q$ (Section 7), the three decoding settings of Section 8 and time-memory trade-offs (Section 5).

Filters. The filters of [32] follow the bucketing codes of Dubiner [21]: a filter divides the positions into k sets and accepts a vector if it contains every digit a

	[32]		this work		
ω	no filters	with filters	no filters	with filters	partitioning
0.3	0.2972	0.2915	0.2884	0.2787	**0.2505**
0.375	0.3206	0.3076	0.3042	0.2889	**0.2596**
0.441	0.3369	0.3197	0.3139	0.2924	**0.2647**
0.5	0.3511	0.3343	0.3193	0.2935	**0.2684**
0.62	0.3877	0.3698	0.3287	0.2949	**0.2741**
0.667	0.4068	0.3843	0.3320	0.2955	**0.2759**

Table 6. Time exponents for ternary LWE keys in [32, Table 5] and in this work.

fixed number of times in every set. Our centers are the case $k = 2$, a center and its complement, and for $k = 2$ the cost of a filtered join in [32, Prop. 3] is that of Eq. (15). The filters of [32] have $k = 3$ sets. Their variables are the distribution over the k sets for every occurring pair of digits, that is, $k-1$ variables per pair, as in the cost formula of Ozerov [44, Theorem 17] for $k = 2$. This number grows quadratically with the number of digits and linearly with k. For $k = 2$, we show in Lemma 2 that one number per digit and list suffices and further reduce them by symmetry arguments. With these filters we optimize trees with digit sets up to $\{-5, \dots, 5\}$ (Section 6), while [32, Sect. 5] reports that its optimization already for smaller digit sets did not converge.

Subset Sum. For the digits $\{-1, 0, 1, 2\}$, the trees of [32] have the exponents 0.2793, 0.2727 and 0.2690 at depths 3, 4 and 5, and ours 0.2791, 0.2726 and 0.2724 (Table 2). The difference at depth 5 comes from the third set: with $k = 3$ sets in the filtered join, where the exponents of Eq. (15) are those of [32, Prop. 3], we obtain 0.2691 for these digits. The partitioning technique introduced in this work further decreases the subset sum exponent we obtain to 0.2620, outperforming the increased amount of filters from [32].

LWE Keys. Table 6 compares the exponents for ternary keys of weight ωn, the only keys that [32] treats. Our exponents with filters are lower by 0.0128 to 0.0888, and those with partitioning by 0.0410 to 0.1084. Our exponents without filters are below those of [32] with filters at every weight. The trees of [32] use the digits $\{-1, 0, 1\}$ on all levels, while ours use $\{-2, \dots, 2\}$ without filters and $\{-3, \dots, 3\}$ with filters below the top level. Moreover, [32] carries over from [38] the bound of the constraint size of every level by its number of representations, while our constraint sizes are free as in [27]. Restricted to the digits $\{-1, 0, 1\}$, our model gives 0.3511 without and 0.3341 with filters for $\omega = 0.5$, close to the 0.3511 and 0.3343 of [32], so the lower exponents of Table 3 require the digits ± 2 and, with filters, ± 3.

Heuristics. Both works rely on the heuristic that lists behave like random sets (Heuristic 1). For filtered joins, [32] supports it by a symmetry argument [32, Lemma 6] and by experiments for small n [32, Sect. 6].

	half distance			full distance	
q	Stern [45]	Hirose [31]	Stern-NN	Stern [45]	Stern-NN
2	0.0557	**0.0550**	**0.0550**	0.1166	**0.1138**
3	0.0827	0.0831	**0.0823**	0.1777	**0.1761**
4	0.0998	0.1007	**0.0995**	0.2179	**0.2171**
5	0.1119	0.1130	**0.1116**	0.2472	**0.2471**
7	0.1284	0.1296	**0.1282**	**0.2886**	0.2891
8	0.1344	0.1356	**0.1343**	**0.3041**	0.3048
11	**0.1476**	0.1488	**0.1476**	**0.3389**	0.3399
16	**0.1614**	0.1624	**0.1614**	**0.3763**	0.3774
27	**0.1779**	0.1786	**0.1779**	**0.4227**	0.4238

Table 7. Worst-case time exponents of Stern's algorithm over $\mathbb{F}_q$ with an exact join and with a nearest-neighbor join. Where [31,41] state an exponent, our evaluation agrees with it up to rounding.

Filters with Three Sets. The filtered join of Algorithm 2 also works with $k = 3$ sets, with the exponents of [32, Prop. 3] in Eq. (15) in a general sense. Furthermore, this technique is naturally compatible with the partitioning technique. A re-optimization in the subset sum case using both techniques with digits $-2, \ldots, 2$ yields slight gains at depth 6, improving the best exponent by 0.0014 to 0.2605.

E Stern's Algorithm with a Nearest-Neighbor Join

Table 7 compares three variants of Stern's algorithm over $\mathbb{F}_q$: with a join that matches the syndrome exactly on some coordinates [45] (Stern), and with a nearest-neighbor join on all $(1-R)n$ syndrome coordinates, once with the center count α_{H} of [31] (Hirose) and once with the center count α of Proposition 1 (Stern-NN). At half distance Stern-NN has a lower exponent than Stern for $q \leq 8$ and the same exponent for $q \geq 11$, while Hirose outperforms Stern only for $q = 2$. At full distance Stern-NN is lower than Stern for $q \leq 5$.

F Ternary Decoding with Large Weight

Over $\mathbb{F}_3$, decoding an error of relative weight $\omega > 2/3$ is the problem behind the signature scheme Wave [18,5], whose parameters lie at $\omega \approx 0.89$ to 0.95, where an instance has exponentially many solutions. Bricout, Chailloux, Debris-Alazard and Lequesne (BCDL) [15] showed that the hardest instances lie at $\omega = 1$ and rate $R = 1 - \log_3 2 \approx 0.369$, where the error has no zero entry and one solution in expectation.

Definition 5 (Ternary Decoding Problem with Large Weight). *Let* $\mathbf{H} \in \mathbb{F}_3^{(1-R)n \times n}$ *be drawn uniformly at random, and let* $2/3 < \omega \leq 1$. *For a random*

error $\mathbf{e} \in \mathbb{F}_3^n$ *of weight* ωn*, let* $\mathbf{s} := \mathbf{He}$ *be its* syndrome*. The* ternary decoding problem with large weight *is, given* $(\mathbf{H}, \mathbf{s})$ *and* ω*, to find any* $\mathbf{e}' \in \mathbb{F}_3^n$ *of weight* ωn *satisfying* $\mathbf{He}' = \mathbf{s}$*.*

We apply Algorithm 3 to this problem. As in [15,43], we state running times in base two.

Ternary Decoding with a Representation Tree Let us state where the decoder differs from Algorithm 3.

Shift. Algorithm 3 builds the error from sparse vectors of level 0 and is efficient when its relative weight is below $(q-1)/q = 2/3$, the weight of a uniformly random vector over $\mathbb{F}_3$. An error of relative weight $\omega > 2/3$ is denser, so we shift its part on the window, as done in previous works [15,43]. With $(1, \dots, 1)$ the all-one vector, after the permutation and the column scalings of Section 7.1 we set $\tilde{\mathbf{e}}_1 := \mathbf{e}_1 - (1, \dots, 1)$, $\tilde{\mathbf{s}}_A := \mathbf{s}_A - \mathbf{A}(1, \dots, 1)$ and $\tilde{\mathbf{s}}_B := \mathbf{s}_B - \mathbf{B}(1, \dots, 1)$, so that $\mathbf{A}\tilde{\mathbf{e}}_1 = \tilde{\mathbf{s}}_A$ and $\mathbf{B}\tilde{\mathbf{e}}_1 + \mathbf{e}_2 = \tilde{\mathbf{s}}_B$. Subtracting 1 sends $0, 1, 2$ to $2, 0, 1$, so the ones of $\mathbf{e}_1$ become the zeros of $\tilde{\mathbf{e}}_1$, and $\tilde{\mathbf{e}}_1$ is sparse if $\mathbf{e}_1$ contains many ones. The tree searches for $\tilde{\mathbf{e}}_1$ and returns $\mathbf{e}_1 = \tilde{\mathbf{e}}_1 + (1, \dots, 1)$, while $\mathbf{e}_2$ is not shifted.

Number of Runs. The decoder is Algorithm 3 with the shift above, so the tree builds $\tilde{\mathbf{e}}_1$, whose type $p^{(d)} = (p_0^{(d)}, p_1^{(d)}, p_2^{(d)})$ is a parameter. The digits 0, 1 and 2 of $\tilde{\mathbf{e}}_1$ are the ones, twos and zeros of $\mathbf{e}_1$, so $\mathbf{e}_1$ has weight $\omega_1 n$ with $\omega_1 = \kappa(p_0^{(d)} + p_1^{(d)})$. For a fixed solution $\mathbf{e}$, the vector $\tilde{\mathbf{e}}_1$ has the type $p^{(d)}$ if two events occur. First, the permutation places $\omega_1 n$ nonzero entries of $\mathbf{e}$ on the window, which has the probability $\mathcal{P}$ of Eq. (21). Second, $\kappa p_0^{(d)} n$ of these entries are ones and the other $\kappa p_1^{(d)} n$ are twos, which holds with probability

$$\binom{\kappa n}{\kappa p_0^{(d)} n,\, \kappa p_1^{(d)} n,\, \kappa p_2^{(d)} n} \Big/ \left(\binom{\kappa n}{\omega_1 n} 2^{\omega_1 n} \right) \simeq 2^{-\kappa(H_3(\omega_1/\kappa)\log 3 - H(p^{(d)}))n}$$

up to a polynomial factor, since by the column scalings every nonzero entry of $\mathbf{e}$ is 1 or 2 with probability $1/2$ (Section 7.1). With a single solution, the decoder therefore runs the tree $2^{\chi n}$ times with

$$\chi = \tfrac{1}{n} \log \mathcal{P}^{-1} + \kappa\big(H_3(\omega_1/\kappa)\log 3 - H(p^{(d)})\big). \tag{31}$$

Large-weight instances often have many solutions. Since an instance has at least one solution by construction, a random instance has $2^{\nu n}$ solutions in expectation, where $\nu := \max\big(0, (H_3(\omega) - (1-R))\log 3\big)$. With $2^{\nu n}$ solutions, the decoder therefore runs the tree $2^{\max(\chi-\nu,0)n}$ times and its running time is $2^{(\max(\chi-\nu,0)+\tau)n}$ with the time exponent τ of the tree (Eq. (8)). Note that for instances with many solutions we may have $\chi \leq \nu$, in which case a single run finds a solution in expectation.

If there are many solutions, a single permutation may yield many different vectors $\tilde{\mathbf{e}}_1$ of type $p^{(d)} = (p_0^{(d)}, p_1^{(d)}, p_2^{(d)})$ that combine to a solution with

$\mathbf{e}_2 = \tilde{\mathbf{s}}_B - \mathbf{B}\tilde{\mathbf{e}}_1$ (after shifting the error back and undoing the scalings and the permutation). In fact, we expect $2^{\max(\nu-\chi,0)n}$ solutions per execution of the tree. As recovering one of these solutions is enough to solve the problem, they play the same role as representations in Section 3. This relaxes the correctness constraint (5) by $\max(\nu - \chi, 0)$, that is, this term is added to its right-hand side and to the representations in the deficits of Eq. (6).

Setup Let us give the ternary decoding specifics on digits, the lists of level 0 and the filters used.

Digits and Types. The digits are $\{0, 1, 2\}$. Due to the scalings and the shift, the digits 1 and 2 of $\tilde{\mathbf{e}}_1$ are the twos and the zeros of $\mathbf{e}_1$. Their shares $p_1^{(d)}$ and $p_2^{(d)}$ are parameters and in general not the same, which is why we allow for arbitrary symmetric 3×3 joint types instead of those of Section 7.3, which treat all nonzero digits alike. Every digit has two splits into an unordered pair of digits,

$$0 = 0 + 0 = 1 + 2, \qquad 1 = 0 + 1 = 2 + 2, \qquad 2 = 0 + 2 = 1 + 1. \tag{32}$$

Given the type of the sum, a joint type is thus determined by three numbers: for each digit of the sum, the fraction of the positions formed by the first (or equivalently the second) split.

Split Joins and Lists of Level 0. A level may consist of *split joins*, the joins of Wagner's algorithm [49], in which $\mathbf{x}$ and $\mathbf{y}$ have support on disjoint halves of the positions of $\mathbf{x} + \mathbf{y}$ and carry the type of $\mathbf{x} + \mathbf{y}$ on their half. A split join has $r^{(i)} = c^{(i)} = 0$. Every level of split joins above a list halves the part of the window that the list covers, and the quantities $D^{(i)}$, $r^{(i)}$, $c^{(i)}$ of Section 3.3 and the window filter exponents are multiplied by the fraction of the n columns that a list of level i covers, in place of the factor κ of Section 7.1. The lists of level 0 of Algorithm 1 are constructed via a split join over all coordinates, which is a meet-in-the-middle. With intermediate split join levels the lists of level 0 are then built by a split join over the two halves of the columns they cover, as in Eq. (2). Fig. 6 shows a tree with one level of joins of Section 3 between two levels of split joins, together with the columns on which its vectors have support. Our trees for ternary decoding with large weight use split joins because the instances often have many solutions, so that fewer representations are needed to ensure that the tree recovers one of them.

Joins and Filters. The top join uses a window filter with opposite tilts and a syndrome filter. On the window the top join may also be a split join, which implies no filter there. The joint types are optimization variables, where the joint type on the syndrome part, i.e., on $\mathbf{Bx}$ and $\tilde{\mathbf{s}}_B - \mathbf{By}$, is J_γ from Section 7.2 with $\gamma = (\omega - \omega_1)/(1 - \kappa)$, ensuring weight $(\omega - \omega_1)n$ on $\mathbf{e}_2 = \tilde{\mathbf{s}}_B - \mathbf{B}(\mathbf{x} + \mathbf{y})$. Since $\mathbf{e}$ is dense, our trees have $\gamma > 2/3$, that is, the pairs are farther apart than random pairs. The filter of Proposition 1 requires $\gamma < 2/3$, so on the syndrome part we use the general filters with separate prices for the two lists (Lemma 2).

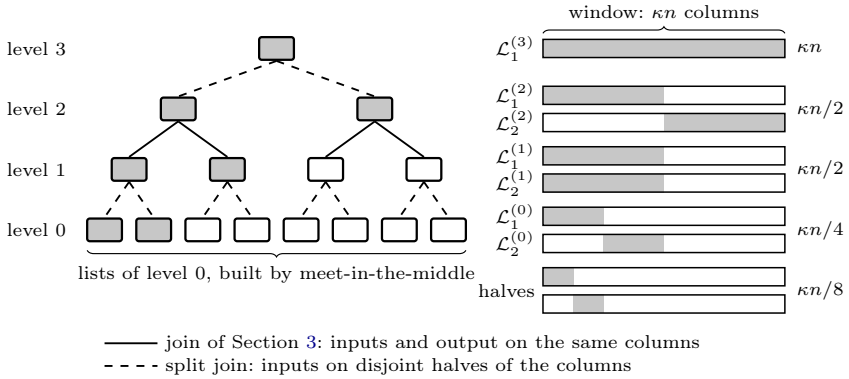


Fig. 6. A tree that mixes joins of Section 3 with split joins. Each bar is the window, shaded where a vector of the list named to its left (shaded in the tree) has support.

				this work		
instance	R	ω	published	no filters	with filters	two parts
hardest	0.369	1	0.243 [43]	0.2304	0.2179	**0.2148**
Wave	0.5	0.894	0.0151 [43]	0.0148	**0.0142**	**0.0142**
Wave	0.676	0.948366	0.0176 [15]	0.0176	**0.0167**	**0.0167**

Table 8. Time exponents for ternary decoding with large weight.

For each of the filters we obtain the center count and list reductions via Eq. (14) and Eq. (10), and as described in Appendix C we obtain $\alpha^{(d)}$, $\beta_x^{(d)}$ and $\beta_y^{(d)}$ as their respective sums, weighted by the fractions κ and $1-\kappa$ of the columns that the two parts cover. For the lower joins we use filtered joins with window filters or split joins, which by construction use no filters.

Numerical Results For a given assignment of split joins and filtered joins to the different levels, following the above description, we optimize the time exponent $\max(\chi-\nu,0)+\tau$. With filters all trees have their optimum at depth 6. In Table 8 we state the time exponents we obtained without and with filters, next to the best published exponents. At the hardest instance, the optimal tree has, from the bottom, one level of split joins followed by five levels of filtered joins. With free joint types on the levels of the tree of [43] we lower the exponent from 0.243 to 0.2304. This improvement comes from the free joint types, since [43] uses only one of the two splits of each digit in Eq. (32). The introduction of filters further improves to 0.2179 and partitioning further to 0.2148. At the Wave instances the best trees we found have, from the bottom: 1 split, 2 filtered and 3 split joins. The higher number of solutions favors split joins, since the solutions take the role of representations. Overall, we achieve 0.0142 (published: 0.0151)

and 0.0167 (published: 0.0176) for Wave parameters with no further gains from partitioning.

G Decoding with Restricted Errors

In the restricted syndrome decoding problem [2], which underlies the signature scheme CROSS [4], the error $\mathbf{e} \in \mathbb{F}_q^n$ has ωn nonzero entries, each of which lies in a fixed subgroup $E \subset \mathbb{F}_q^*$ of order z.

Definition 6 (Restricted Syndrome Decoding Problem). *Let $E \subset \mathbb{F}_q^*$ be a subgroup of order z, and let $\mathbf{H} \in \mathbb{F}_q^{(1-R)n \times n}$ be drawn uniformly at random. For a random* error $\mathbf{e} \in \mathbb{F}_q^n$ *with ωn entries in E and the others zero, let $\mathbf{s} := \mathbf{He}$ be its* syndrome. *The* restricted syndrome decoding problem *is, given $(\mathbf{H}, \mathbf{s})$, E and ω, to find any such $\mathbf{e}'$ satisfying $\mathbf{He}' = \mathbf{s}$.*

Baldi, Bitzer, Pavoni, Santini, Wachter-Zeh and Weger [3] adapted the BJMM decoder to this problem, with lists over the digits $E_0 := E \cup \{0\}$ and a small set of sums of two elements of E, and with the number of representations derived separately for $z = 2, 4, 6$. We apply Algorithm 3 to this problem, for their parameters $q = 157$ and $R = 0.5$. As in [3], we state running times in base two.

Restricted Decoding with a Representation Tree Let us state where the decoder differs from Algorithm 3.

Number of Runs. In place of the scalings of Section 7.1, the decoder multiplies every column of $\mathbf{H}$ by a fresh uniformly random element of E. Since E is a group, the new error is again restricted, and its nonzero entries are independent and uniformly random elements of E. For every restricted error, the new error thus has the type $\bar{p}$ with probability $1/\mathrm{poly}(n)$, where $\bar{p}_0 = 1-\omega$, $\bar{p}_j = \omega/z$ for $j \in E$ and $\bar{p}_j = 0$ for all other digits. The decoder searches for a part $\mathbf{e}_1$ of the error on the window with a predefined type $p^{(d)}$, where $p_j^{(d)} > 0$ exactly for $j \in E_0$. The error then has the type $\bar{p}' := (\bar{p} - \kappa p^{(d)})/(1-\kappa)$ on the remaining $(1-\kappa)n$ columns for every digit j. A random permutation gives $\mathbf{e}_1$ the type $p^{(d)}$ with probability

$$\mathcal{P} = \binom{\kappa n}{(\kappa p_j^{(d)} n)_j} \binom{(1-\kappa)n}{((1-\kappa)\bar{p}'_j n)_j} \Big/ \binom{n}{(\bar{p}_j n)_j},$$

the number of arrangements of the digits of $\mathbf{e}$ with type $p^{(d)}$ on the window divided by the number of all arrangements (compare to Eq. (21)), and the decoder runs the tree $\mathcal{P}^{-1}$ times, so its running time has the exponent of Eq. (22). Note that each of the $2^{H(\bar{p})n}$ vectors of type $\bar{p}$ other than $\mathbf{e}$ has the syndrome $\mathbf{s}$ with probability $q^{-(1-R)n}$. The expected number of these vectors that have the syndrome $\mathbf{s}$ is therefore $2^{(H(\bar{p})-(1-R)\log q)n}$. For $z \le 4$ only the $z+1 \le 5$ digits of E_0 have a nonzero share in $\bar{p}$, so $H(\bar{p}) \le \log 5 \approx 2.32$, which for $q = 157$ and $R = 0.5$ is smaller than $(1-R)\log q \approx 3.65$. This expected number is thus $2^{-\Omega(n)}$, so by Markov's inequality $\mathbf{e}$ is the only solution except with probability $2^{-\Omega(n)}$.

Setup Let us give the restricted decoding specifics on digits, labels and the filters used.

Digits and Types. The lists use the digits $E_0 \cup E_+ \cup E_-$, where $E_\circ = \{a \circ b \mid a, b \in E\}$. As in Section 3, the join into level i is given by a symmetric joint type $J^{(i)}$ over these digits whose anti-diagonal sums, taken in $\mathbb{F}_q$, give $p^{(i)}$, and all its entries are optimization variables. Lemma 1 gives $r^{(i)}$ and $c^{(i)}$, which are multiplied by κ as in Section 7.1.

Labels and Lists of Level 0. The labels and the lists of level 0 are those of Section 7.3.

Joins and Filters. The top join uses a window filter and a syndrome filter as in Appendix F. On the syndrome part, the vectors $\mathbf{Bx}$ and $\mathbf{s}_B - \mathbf{By}$ of the pair with $\mathbf{x} + \mathbf{y} = \mathbf{e}_1$ differ in each coordinate by the entry of $\mathbf{e}_2 = \mathbf{s}_B - \mathbf{B}(\mathbf{x} + \mathbf{y})$, which has the type $\bar{p}'$ over E_0. As in Section 7.2, the digits of $\mathbf{Bx}$ are uniformly random and independent of $\mathbf{e}_2$, so the top join filters for the joint type $J_{\bar{p}'}$ of $\mathbf{Bx}$ and $\mathbf{s}_B - \mathbf{By}$, with $(J_{\bar{p}'})_{i,i+a} = \bar{p}'_a/q$ for $i \in \mathbb{F}_q$ and $a \in E_0$, and with all other entries 0. Put differently, the joint type only includes pairs whose difference lies in E_0, with the shares given by the type $\bar{p}'$ of $\mathbf{e}_2$. Note that $J_{\bar{p}'}$ is symmetric only if $\bar{p}'_a = \bar{p}'_{-a}$ for every a. However, its row sums and its column sums are all $1/q$, so Lemma 2 still applies. Lower joins are filtered joins as well, using window filters with opposite tilts and one price per digit.

Numerical Results The variables of the optimization problem (9) with the correctness constraint (5) are the joint types, the constraint sizes, ℓ, $p^{(d)}$ and the prices and center sizes of the filters. Since [3] does not state the exponents explicitly, we recomputed their exponents with the code provided. In Table 9 we state the exponents of our trees without filters, with filters and with two parts (Section 4.4) next to them, for $z = 2$, where $E = \{\pm 1\}$ and the lists use the digits 0, ± 1 and ± 2, and for $z = 4$, which we discuss below. The columns d give the depth of each tree.

For $z = 2$, our tree with depth 3 and no filters agrees with [3] to within 10^{-3} at every weight, so for $\omega \geq 0.4$ the difference between the published exponents and those without filters comes from the additional join levels. Filters lower the exponent at every weight, while partitioning helps in denser regimes.

For $z = 4$ we have $E = \{\pm 1, \pm\eta\}$ with $\eta^2 = -1$. Here [3] uses the sums $\pm(\eta + 1), \pm(\eta - 1)$, whereas our lists use all eight nonzero sums of two elements of E, which gives 13 digits. Most of the improvement without filters comes from the increased digit set, while filters again improve in all cases with partitioning improving for $\omega = 0.6$.

H Decoding in the Lee Metric

Over $\mathbb{Z}_q$ the *Lee weight* of a digit $j \in \{0, \ldots, q-1\}$ is $\mathrm{wt}_\mathrm{L}(j) := \min(j, q - j)$, and the Lee weight of a vector is the sum of the Lee weights of its entries.

				this work					
z	ω	published [3]	d	no filters	d	with filters	d	two parts	d
2	0.2	0.1334	2	0.1334	2	**0.1294**	3	**0.1294**	3
	0.4	0.1893	3	0.1869	4	0.1725	5	**0.1678**	6
	0.6	0.2242	3	0.2099	5	0.1849	11	**0.1805**	11
	0.8	0.2517	3	0.2239	5	0.1962	8	**0.1931**	9
4	0.4	0.3053	2	0.3038	2	**0.2849**	3	**0.2849**	3
	0.6	0.4068	3	0.3990	3	0.3723	3	**0.3721**	3

Table 9. Time exponents for restricted errors with $q = 157$ and $R = 0.5$, where d is the number of join levels.

Definition 7 (Syndrome Decoding Problem in the Lee Metric). *Let* $\mathbf{H} \in \mathbb{Z}_q^{(1-R)n \times n}$ *be drawn uniformly at random. For a random* error $\mathbf{e} \in \mathbb{Z}_q^n$ *of Lee weight* δn*, let* $\mathbf{s} := \mathbf{He}$ *be its* syndrome*. The* syndrome decoding problem in the Lee metric *is, given* $(\mathbf{H}, \mathbf{s})$ *and* δ*, to find any* $\mathbf{e}' \in \mathbb{Z}_q^n$ *of Lee weight* δn *satisfying* $\mathbf{He}' = \mathbf{s}$.

Syndrome decoding in the Lee metric was proposed as a basis for code-based cryptography because the generic decoders are slower than in the Hamming metric [50]. Weger et al. [50] translate Stern and BJMM to the Lee weight, and Bariffi, Khathuria and Weger [7] restrict the lists to digits of small Lee weight. We apply Algorithm 3 to this problem. As in Section 7, the running times q^{Fn} stated in [50,7] correspond to the exponent $F \log q$ in this work.

Lee-Metric Decoding with a Representation Tree Let us state where the decoder differs from Algorithm 3 when the Lee weight takes the place of the Hamming weight.

Type of the Error. Over $\mathbb{F}_q$ the decoder works with an error in which all nonzero digits are equally frequent (Section 7.1). Among all types of a given Hamming weight this is the type with the most vectors. The Lee metric has a type with the same property. All vectors of one type have the same Lee weight: a vector of type p has $p_j n$ entries equal to j, so its Lee weight is $n \sum_j p_j \mathrm{wt_L}(j)$. The vectors of Lee weight δn are thus the vectors of the types with relative Lee weight $\sum_j p_j \mathrm{wt_L}(j) = \delta$. Each such type has $2^{H(p)n}$ vectors up to polynomial factors (Eq. (1)), and there are only polynomially many types. Up to polynomial factors, the number of vectors of Lee weight δn is therefore the number of vectors of the type of largest entropy among them. For the Lee weight this type is known as the Boltzmann distribution [6] defined as

$$p_j = \frac{2^{-\zeta \,\mathrm{wt_L}(j)}}{\sum_{k \in \mathbb{Z}_q} 2^{-\zeta \,\mathrm{wt_L}(k)}}, \tag{33}$$

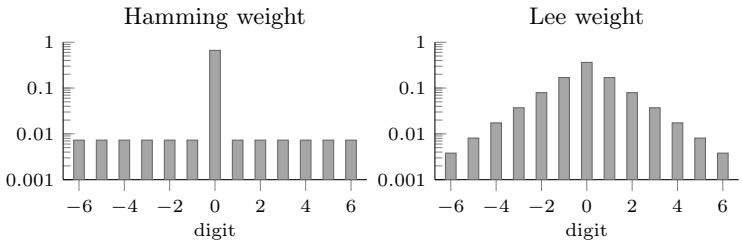


Fig. 7. Shares of the digits $-6, \ldots, 6$ in the error for $q = 47$ and $R = 0.5$ at full distance, on a logarithmic scale: all nonzero digits are equally frequent for the Hamming weight (left), and the share falls with the Lee weight of the digit in the Lee metric (right).

where ζ is the unique number for which the relative Lee weight of p is δ [7, Lemma 11]. For the weights that we consider ζ is positive, so the digit 0 is the most frequent one, followed by ± 1, ± 2 and so on. With the Hamming weight in place of wt_L, Eq. (33) is the type of Section 7.3, in which all nonzero digits have the same share. As there, a type of the form of Eq. (33) is determined by its weight. Fig. 7 shows both types for $q = 47$ and $R = 0.5$ at full distance. In the Lee metric the digit 0 has the share 0.36, the digits ± 1 have the share 0.17 each, and every further unit of Lee weight multiplies the share by 0.47.

Let $\bar{p}$ be the type of Eq. (33) with relative Lee weight δ. Unlike over $\mathbb{F}_q$ (Section 7.1), column scalings would change the Lee weight, so the decoder draws random permutations only, which keep the type of the error. A random error has the type $\bar{p}$ with probability $1/\mathrm{poly}(n)$, and we compute the exponents for this type. Up to a term that vanishes as n grows, they also hold for errors of a type close to $\bar{p}$, which are almost all errors of Lee weight δn (Eq. (1)).

Number of Runs. The decoder searches for a part $\mathbf{e}_1$ of the error on the window whose type $p^{(d)}$ has the form of Eq. (33). Its relative Lee weight is a parameter, like ω_1 in Section 7.1. The error then has the type $\bar{p}' := (\bar{p} - \kappa p^{(d)})/(1-\kappa)$ on the remaining $(1-\kappa)n$ columns, and a random permutation gives $\mathbf{e}_1$ the type $p^{(d)}$ with the probability $\mathcal{P}$ of Appendix G. The decoder thus runs the tree $\mathcal{P}^{-1}$ times. In *full-distance* decoding δ is the Lee weight at which the expected number $2^{(H(\bar{p})-(1-R)\log q)n}$ of solutions is 1, and in *half-distance* decoding it is half of this value, as in Section 7.

Setup Let us give the Lee-metric decoding specifics on digits, labels and the filters used.

Digits and Types. The digits are the elements of $\mathbb{Z}_q$, and all of them occur in the lists. Over $\mathbb{F}_q$ the type of every level is determined by its weight (Section 7.3). We proceed in the same way: the type $p^{(i)}$ of every level has the form of Eq. (33), which leaves one parameter per level, its relative Lee weight. At every join we

use the joint type J^* of largest entropy that satisfies the types of the current and the next level. Lemma 1 then gives $r^{(i)}$ and $c^{(i)}$.

Labels and Lists of Level 0. The labels and the lists of level 0 are those of Section 7.3.

Joins and Filters. As in Section 7.3, the top join matches the remaining $\ell^{(d)}n$ label bits and uses a syndrome filter. Every join, the top join included, also uses a window filter. The top join has to find the pairs for which $(\mathbf{s}_B - \mathbf{By}) - \mathbf{Bx}$ has the type $\bar{p}'$, which takes the place of the Hamming distance of Section 7.2. Since $\mathbf{Bx}$ and $\mathbf{s}_B - \mathbf{By}$ are treated as uniformly random vectors, such a pair has the joint type $J_{\bar{p}'}$ with

$$(J_{\bar{p}'})_{jj'} = \frac{\bar{p}'_{j'-j}}{q}, \tag{34}$$

where j is a digit of $\mathbf{Bx}$, j' is a digit of $\mathbf{s}_B - \mathbf{By}$, and the difference is taken in $\mathbb{Z}_q$. For an error with equally frequent nonzero digits this is the joint type J_γ of Eq. (23).

A general filter has $2q-1$ parameters. We use the filter of Section 7.2 with the Lee weight in place of the Hamming weight. That filter gives the price z_0 to the digit 0 and 0 to every other digit, which is the same filter as the prices $z_j = -z_0 \operatorname{wt}(j)$, since adding the same number (here $-z_0$) to all prices simply corresponds to a change in g. We therefore use the prices $z_j = -\sigma \operatorname{wt}_{\mathrm{L}}(j)$ with $\sigma \geq 0$. By Eq. (18) a center then contains the fraction

$$\theta_{jj'} = \frac{g\, 2^{-\sigma(\operatorname{wt}_{\mathrm{L}}(j)+\operatorname{wt}_{\mathrm{L}}(j'))}}{1 + g\, 2^{-\sigma(\operatorname{wt}_{\mathrm{L}}(j)+\operatorname{wt}_{\mathrm{L}}(j'))}}$$

of the positions in which the two vectors have the digits j and j'. Put differently, the positions in which both digits have a small Lee weight are the most likely to lie inside a center. The filter thus has the two parameters σ and g, like the filter of Section 7.2, and its exponents α and β follow from Eqs. (13) and (17) for the joint type $J_{\bar{p}'}$. The top join costs $T^{(d)}$ of Eq. (26) with these exponents. For comparison we also state exponents for a top join without filter, which checks every consistent label match for the type $\bar{p}'$.

Numerical Results Exponents have been published for three parameter sets: $q = 47$ at full distance [7], and $q = 49$ and $q = 4$ at half distance [50]. For a given depth d and a given rate, the variables of the optimization problem (9) are the relative Lee weights of the types $p^{(0)}, \ldots, p^{(d)}$, the constraint sizes, ℓ, the parameters σ and g of the syndrome filter and the prices and center sizes of the window filters, and we require the correctness constraint (5). As for decoding over $\mathbb{F}_q$ (Table 5), decoders are compared by their worst-case exponent, the maximum over the rate [7, Table 1], where every rate uses the depth with the lowest exponent.

In Table 10 we state the worst-case exponents, rounded up, for a top join without filter and for Algorithm 3 with all filters, next to the published values.

					this work			
q	weight	published		d	no filters	d	with filters	d
47	full distance	0.660	[7]	1	0.4058	6	**0.3580**	7
49	half distance	0.418	[50]	1	0.2851	4	**0.2582**	4
4	half distance	0.1053	[50]	1	0.0991	2	**0.0950**	2

Table 10. Worst-case time exponents for decoding in the Lee metric, where d is the number of join levels at the worst-case rate.

For $q = 47$ Algorithm 3 has the exponent 0.3580, attained at $R = 0.55$ with depth 7, against the best published value 0.660 at $R = 0.406$ [7]. The difference comes mostly from the number of representations (which was approximated in earlier works) and the depth. The filters lower the exponent from 0.4058 to 0.3580. Two parts (Section 4.4) give no lower exponent. At half distance the exponent gains behave similarly, while the overall improvement is smaller.

I Parameters of the Tree of Theorem 1

In this appendix we give the parameters of the tree that reaches the time exponent 0.2620 of Theorem 1 with the smallest memory that we found, a memory exponent of 0.212. The tree has depth 7, uses the digits $\{-3, \dots, 3\}$ and the filters of Section 5, and partitions the positions into two parts as in Section 4.4. The lists of level 0 are built by the algorithm of Schroeppel and Shamir [47] in time $2^{0.2572n}$ and memory $2^{0.1286n}$, and no part of the tree is repeated. With depth 6 the same memory gives the exponent 0.2637, and with memory exponent 0.211 depth 8 gives 0.2621.

Table 11 lists the parameters, with all quantities as exponents per n. The first block holds the quantities that are sums over both parts: the list size $L^{(i)}$ of level i, the constraint size $\ell^{(i)}$ and the number of representations $r^{(i)}$ of the join into level i, where $\ell^{(0)}$ is the constraint size of the lists of level 0. The other two blocks give, for each part, the type $p^{(i)}$ of level i by the shares $p_j^{(i)}$ of the digits $j = -3, \dots, 3$, and the exponent $c^{(i)}$ of the fraction of consistent pairs, the center size $u^{(i)}$ and the filter exponents $\alpha^{(i)}$, $\beta_x^{(i)}$ and $\beta_y^{(i)}$ of the join into level i. The values of a part are stated per position of that part, so a quantity of the tree is the average of the two parts weighted by their sizes. All values are rounded to nearest. The witness file, provided with the paper, contains all parameters in full precision, including the joint types $J^{(i)}$.

The first part covers 56% of the positions. Below the top level its vectors have at most 31% nonzero entries, and it uses filters only at the joins into levels 6 and 7, where the reduction $\beta_x^{(i)} + \beta_y^{(i)} - \alpha^{(i)}$ of the filter (Section 4.2) is 7% and 12% of the loss $c^{(i)}$. The second part has 24% to 58% nonzero entries below the top level. Although it covers 44% of the positions, it accounts for 64% to 94% of the loss $c^{(i)}$ of the tree at every join, and at the joins into levels 3 to 7 its filters reduce 36% to 82% of its loss.

	level i							
	0	1	2	3	4	5	6	7
both parts (weighted averages)								
$L^{(i)}$	0.2040	0.2070	0.2086	0.2099	0.2120	0.1926	0.2120	–
$\ell^{(i)}$	0.3103	0.1957	0.1808	0.1524	0.1004	0.0604	0	–
$r^{(i)}$	–	0.2990	0.5202	0.7004	0.8449	0.9396	0.9467	1.1065
first part, 56.01% of the positions								
$p_{-3}^{(i)}$	0.00002	0.00002	0.00001	0	0	0	0	0
$p_{-2}^{(i)}$	0.00004	0.00004	0.00003	0.00002	0.00001	0.00003	0.00010	0
$p_{-1}^{(i)}$	0.00277	0.00535	0.00982	0.01689	0.02551	0.02941	0.03132	0
$p_0^{(i)}$	0.99042	0.98137	0.96465	0.93491	0.88646	0.81621	0.69054	0.50000
$p_1^{(i)}$	0.00669	0.01317	0.02545	0.04815	0.08802	0.15423	0.27456	0.50000
$p_2^{(i)}$	0.00004	0.00004	0.00003	0.00002	0	0.00012	0.00348	0
$p_3^{(i)}$	0.00002	0.00002	0.00001	0	0	0	0	0
$c^{(i)}$	–	0.0013	0.0024	0.0054	0.0151	0.0400	0.0446	0.2714
$u^{(i)}$	–	0.5000	0.5000	0.5000	0.5000	0.5000	0.0031	0.9785
$\alpha^{(i)}$	–	0.0000	0.0000	0.0000	0.0000	0.0000	0.0110	0.0592
$\beta_x^{(i)}$	–	0.0000	0.0000	0.0000	0.0000	0.0000	0.0073	0.0472
$\beta_y^{(i)}$	–	0.0000	0.0000	0.0000	0.0000	0.0000	0.0067	0.0439
second part, 43.99% of the positions								
$p_{-3}^{(i)}$	0.00011	0.00017	0.00060	0.00103	0.00020	0.00001	0	0
$p_{-2}^{(i)}$	0.00070	0.00836	0.02205	0.03325	0.02668	0.00793	0.00212	0
$p_{-1}^{(i)}$	0.11867	0.18088	0.22024	0.23261	0.23606	0.20365	0.11819	0
$p_0^{(i)}$	0.75714	0.61391	0.50093	0.44134	0.42399	0.46731	0.52905	0.50000
$p_1^{(i)}$	0.12256	0.18758	0.23129	0.25142	0.27403	0.29775	0.32897	0.50000
$p_2^{(i)}$	0.00071	0.00891	0.02423	0.03904	0.03859	0.02324	0.02154	0
$p_3^{(i)}$	0.00011	0.00018	0.00067	0.00132	0.00044	0.00011	0.00013	0
$c^{(i)}$	–	0.0105	0.0526	0.1180	0.2251	0.3377	0.3370	0.6183
$u^{(i)}$	–	0.5000	0.5000	0.5000	0.5000	0.5000	0.5030	0.5065
$\alpha^{(i)}$	–	0.0000	0.0000	0.1209	0.3334	0.4582	0.5815	0.5202
$\beta_x^{(i)}$	–	0.0000	0.0000	0.0816	0.2367	0.3446	0.4286	0.4217
$\beta_y^{(i)}$	–	0.0000	0.0000	0.0816	0.2367	0.3446	0.4293	0.4260

Table 11. Parameters of the subset sum tree with time exponent 0.2620 and memory exponent 0.212.