The Approximate k-List Problem

Leif Both and Alexander May

Horst Görtz Institute for IT-Security
Ruhr-University Bochum, Germany
Faculty of Mathematics
leif.both@rub.de, alex.may@rub.de

Abstract. We study a generalization of the k-list problem, also known as the Generalized Birthday problem. In the k-list problem, one starts with k lists of binary vectors and has to find a set of vectors – one from each list – that sum to the all-zero target vector. In our generalized *Approximate k-list problem*, one has to find a set of vectors that sum to a vector of small Hamming weight ω. Thus, we relax the condition on the target vector and allow for some error positions.
This in turn helps us to significantly reduce the size of the starting lists, which determines the memory consumption, and the running time as a function of ω. For $\omega = 0$, our algorithm achieves the original k-list run-time/memory consumption, whereas for $\omega = \frac{n}{2}$ it has polynomial complexity.
As in the k-list case, our Approximate k-list algorithm is defined for all $k = 2^m, m > 1$. Surprisingly, we also find an Approximate 3-list algorithm that improves in the run-time exponent compared to its 2-list counterpart for all $0 < \omega < \frac{n}{2}$. To the best of our knowledge this is the first such improvement of some variant of the notoriously hard 3-list problem.
As an application of our algorithm we compute small weight multiples of a given polynomial with more flexible degree than with Wagner's algorithm from Crypto 2002 and with smaller time/memory consumption than with Minder and Sinclair's algorithm from SODA 2009.

Keywords: nearest neighbor problem · approximate matching · k-list problem · birthday problem · collision search · low weight polynomials

1 Introduction

Birthday-type attacks and their generalization by Wagner in 2002 [Wag02] to the so-called k-list algorithm are one of the most fundamental tools in cryptanalysis, and therefore of invaluable importance for assessing secure instantiations of cryptographic problems. Wagner's algorithm and its variations found numerous applications, e.g. for attacking problems like e.g. hash function constructions [CJ04], LPN [LF06, GJL14, ZJW16], codes [MO15, BJMM12, MMT11], lattices [AKS01, KS01] and subset sum [Lyu05, BCJ11, HGJ10]. Furthermore the famous BKW algorithm [BKW00, AFFP14] – that was proposed 2 years before Wagner's algorithm – can be considered as a special case of Wagner's algorithm with a limited number of lists.

In Wagner's original k-list problem one receives lists $L_1, \ldots, L_k$ with independent random vectors from $\mathbb{F}_2^n$. The goal is to find $(x_1, \ldots, x_k) \in L_1 \times \ldots L_k$ such that $x_1 + \ldots + x_n = 0^n$ over $\mathbb{F}_2^n$. Notice that one can freely choose the list sizes $|L_i|$. Information theoretically, a solution exists whenever $|L_i| \geq 2^{\frac{n}{k}}$. Wagner showed that one can also compute a (special type) solution, whenever $|L_i| \geq 2^{\frac{n}{\log k+1}}$ for all i, and k a power of two. The run-time and memory consumption of Wagner's algorithm is up to polynomial factors the input size of the k lists. For non-powers of two $k = 2^m + i, i < 2^m$ one randomly

chooses an element $(x_{2^m+1}, \ldots, x_k) \in L_{2^m+1} \times \ldots \times L_k$, defines $c := x_{2^m+1} + \ldots + x_{2^k}$ and runs the $k' = 2^m$-list algorithm on the lists $L_1, \ldots, L_{k'-1}, (L_{k'} + c)$ with $(L_{k'} + c) := \{x_{k'} + c | x_{k'} \in L_{k'}\}$. It is a long-standing open problem to improve the running time for the k-list problem between any two consecutive powers of 2.

If in practical applications one does not have sufficiently many elements to fill each list with $|L_i| = 2^{\frac{n}{\log k+1}}$ elements, then one can use Minder and Sinclair's variant [MS09] of the k-list algorithm. This variant first expands the lists in a tree-based fashion up until a certain stage, where Wagner's original algorithm can be applied. Moreover, there exist time-memory tradeoffs [Ber07, NS15] to reduce the large and often impractical memory consumption of the k-list algorithm.

Interestingly in some recent applications, the requirement that the sum of all x_i exactly matches to the target vector 0^n is too strong. In fact, in many recent advances in cryptanalysis – e.g. for the LPN/LWE [GJL14, GJS15, KF15] and decoding [MO15, BJMM12, MMT11] problem – the authors relax their algorithms to allow for sums that match to some low Hamming weight. A similar situation appears for the search of near collisions of a hash function [LR11, Leu12]. Thus, we belief that a more flexible treatment of the k-list problem, where one additionally allows for some error positions in the target, will find various further applications.

This motivates our *Approximate k-list problem*, parameterized by some error ω, in which one has to find $(x_1, \ldots, x_k) \in L_1 \times \ldots L_k$ such that $|x_1 + \ldots + x_n| \leq \omega$, where $|\cdot|$ denotes the Hamming weight. Intuitively, this problem should be easier to solve than the original k-list problem, since we obtain more potential solutions for growing error ω. Information theoretically, the Approximate k-list problem has a solution, whenever $|L_i| \geq (2^n \binom{n}{\omega})^{\frac{1}{k}} \approx 2^{\frac{(1-H(\omega))n}{k}}$, where $H(p) = -p\log(p) - (1-p)\log(1-p)$ with $p \in (0,1)$ is the binary entropy function.

Our results: In Section 2.1, we start with an algorithm for the Approximate 2-list problem. Notice that in this case one has to find $(x_1, x_2) \in L_1 \times L_2$ having a Hamming distance of at most ω. Therefore, we can directly apply the algorithm of May-Ozerov [MO15] from Eurocrypt 2015 to find *Nearest Neighbors* in the Hamming distance. This results in an algorithm whose run-time/memory is a strictly decreasing function for growing ω in the relevant interval $0 \leq \omega \leq \frac{n}{2}$. For $\omega = 0$, we receive as a special case the original complexity $2^{\frac{n}{2}}$ of the 2-list problem.

Next, we give an Approximate 3-list algorithm in Section 2.2. As in the original k-list algorithm, our algorithm starts to match the elements in the first two lists L_1, L_2 exactly on ℓ coordinates, for some parameter ℓ. In the list L_3, we only filter the candidates for small relative Hamming weight $\frac{\omega}{\ell}$ on these ℓ coordinates. This technique is somewhat similar to the approach from Nikolic, Sasaki [NS15], but in the approximate k-list scenario such a combination of matching and filtering achieves an exponential speedup in the run-time in comparison to the 2-list algorithm, whereas Nikolic and Sasaki gain only some polynomial speedup in the original k-list scenario. Eventually, we match the two resulting lists again via a Nearest Neighbor search.

Notice that it is widely believed that the 3-list problem cannot be solved faster than the 2-list problem. This belief is directly linked to the famous 3SUM-problem in complexity theory [KPP14], where it is conjectured that any algorithm for the 3-list problem requires running time at least quadratic in the list sizes. This means that every algorithm basically has to look at every pair of $L_1 \times L_2$ and compare with the elements in L_3.

In our Approximate 3-list setting we can work around this restriction, since Nearest Neighbor search already works in sub-quadratic time, i.e., without looking explicitly at every pair in two lists. To the best of our knowledge our approximate 3-list algorithm is the first exponential improvement to some variant of the 3-list problem.

In Section 2.3, we generalize our Approximate k-list algorithm for powers of two,

$k = 2^m, m > 2$. In a nutshell, our general $k = 2^m$-algorithm first applies exact matchings (like in Wagner's algorithm) and some filtering for small weight up until we reach two lists, and then applies a Nearest Neighbor search. We first give a non-optimized version of our algorithm that allows for explicitly stating the original lists sizes $|L_i|$ and the run-time/memory consumption as a closed formula of the parameters (n, k, ω). Such a closed formula is certainly useful for direct application by cryptanalysts – although our formula is somewhat less handy than Wagner's formula for the k-list problem.

In Section 2.4 we do a more refined analysis to minimize the running time or the memory consumption. This however requires some parameter optimization, which prevents us from obtaining a closed formula. Of course, we are still able to illustrate our further improvements from optimization in form of tables and run-time graphs as a function of ω.

In Section 2.5 we give algorithms for the cases $k = 2^m + 2^{m-1}, m > 2$ which are in some sense a combination of the 3-list and the 2^m-list algorithms. We would like to point out that we cannot get improved complexities for other choices of k, e.g. for $k = 5, 7, 9, 10, 11$.

In Section 3 we give a simpler and easy to implement variant of our algorithm, which we call Match-and-Filter, with only slightly larger running times. Our Match-and-Filter algorithm is inspired by near collision search for hash functions [Leu12].

As an application of our approximate k-list problem, we show how to compute small weight multiples of a given polynomial $P(x) \in \mathbb{F}[x]$ of degree n. This is a problem that directly arises in the context of fast correlation attacks [MS89, CT00, CJM02, LJ14, ZXM15]. If one wants to find a weight-d, $d = 2^m + i$, $i < 2^m$ multiple $Q(x)$ of $P(x)$, one usually has to run the non-optimal $k = 2^m$-algorithm. However in the approximate scenario, we can now put the remaining i-coordinates into the error ω, and thus allow for more flexible parameter settings. See Section 4.1 for further details.

2 Solving the Approximate k-list Problem

Throughout this section, we will omit polynomial factors in our exponential running times. I.e., when we sort a list of size 2^{cn} for some constant c, we will for simplicity of notation write that this can be done in time 2^{cn} instead of $\mathcal{O}(n2^{cn})$ or $2^{cn(1+o(1))}$. We will however take care of all polynomial factors in our theorem statements, where these factors contribute to some error term ε. Hence, in our notation sorting can be done in $2^{cn(1+\varepsilon)}$ for any $\varepsilon > 0$.

Furthermore, we will throughout this section assume that k is a constant and does not depend on n, which is the usual application scenario in crypto. We will also assume that $w = c_w n$ for some constant c_w, i.e., the target weight is linear in n.

For completeness, let us also formally define the approximate k-list problem.

Definition 1. In the *k-list problem with approximation* $c_w \in [0, \frac{1}{2}]$, one has to find in k lists $L_1, \ldots, L_k \subset \mathbb{F}_2^n$ an element $(x_1, \ldots, x_k) \in L_1 \times \ldots \times L_k$ such that $x_1 + \ldots + x_k$ has Hamming distance $|x_1 + \ldots + x_k| \leq c_w n$. Here the entries in each L_i are randomly and independently chosen from $\mathbb{F}_2^n$, the list sizes can be chosen freely. We say that an algorithm $\mathcal{A}$ in expectation solves the k-list problem with approximation c_w, if $\mathcal{A}$ finds in expectation a solution, where the probability space is taken over the k-list instance and $\mathcal{A}$'s internal coin tosses.

2.1 An Algorithm for the Approximate 2-list Problem

The basic idea of our approximate 2-list algorithm is to start with two lists L_1, L_2 of equal size such that there exists in expectation an element $(x_1, x_2) \in L_1 \times L_2$ with Hamming distance $|(x_1, x_2)| \leq c_w n$. Such an element can with overwhelming probability be found in subquadratic time in the list size $|L_i|$ with the May-Ozerov Nearest Neighbor algorithm.

Theorem 1 (May-Ozerov [MO15]). *Let $L_1, L_2 \subset \mathbb{F}_2^n$ be two lists of size 2^c with uniform and pairwise independent vectors. For any constant $\varepsilon > 0$, one can find some pair $(x_1, x_2) \in L_1 \times L_2$ with Hamming distance $|(x_1, x_2)| \leq c_w n \in \mathbb{N}$, $c_w \in [0, \frac{1}{2}]$ in time*

$$\widetilde{\mathcal{O}}\left(2^{(y+\varepsilon)n}\right) \quad \text{with} \quad y := (1-c_w)\left(1 - H\left(\frac{H^{-1}(1-\frac{c}{n}) - \frac{c_w}{2}}{1-c_w}\right)\right)$$

with overwhelming probability, provided that the restriction $\frac{c}{n} < 1 - H(\frac{c_w}{2})$ holds.

The restriction $1 - \frac{c}{n} > H(\frac{c_w}{2})$ guarantees that in the definition of y the argument of $H(\cdot)$ is positive. We would also like to remark that in [MO15], May and Ozerov originally showed Theorem 1 for the case $|(x_1, x_2)| = c_w n$, but it is not hard to see that it also holds for $|(x_1, x_2)| \leq c_w n$. Namely, y is for constant $\frac{c}{n}$ a strictly increasing function in c_w. Therefore, we can run the May-Ozerov algorithm for all $\mathcal{O}(n)$ integers in the interval $[0, c_w n]$ that are candidates for $|(x_1, x_2)|$. The largest running time appears for the maximum value, and thus we get an additional polynomial overhead of at most $\mathcal{O}(n)$, which can be subsumed in the run time's ε-term.

Together with the basic idea of our approximate 2-list algorithm described above, we get the following immediate result for the approximate 2-list problem, which will also serve as a building block for analyzing the approximate k-list problem for all $k > 2$.

Theorem 2. *For all $\varepsilon > 0$, the 2-list problem with approximation $c_w \in [0, \frac{1}{2}]$ can in expectation be solved in time*

$$T = 2^{(T^*(c_w)+\varepsilon)n} \quad \text{with} \quad T^*(c_w) := (1-c_w)\left(1 - H\left(\frac{H^{-1}(\frac{1+H(c_w)}{2}) - \frac{c_w}{2}}{1-c_w}\right)\right),$$

using memory $M = 2^{\frac{1-H(c_w)}{2} n(1+o(1))}$ and two lists of size M.

Proof. Let 2^c be the size of the two initial lists. We set $c = \frac{1-H(c_w)}{2} n$. The size of $L_1 \times L_2$ is 2^{2c}. Let $S_{i,j}$ be an indicator variable that takes value 1 iff the i-th element in L_1 and the j-th element in L_2 have Hamming distance at most $c_w n$. Then

$$\mathbb{E}[S_{i,j}] = \Pr(S_{i,j} = 1) = \frac{\sum_{i=0}^{c_w n} \binom{n}{i}}{2^n} \geq \binom{n}{c_w n} \cdot 2^{-n} \geq \frac{2^{(H(c_w)-1)n}}{n+1},$$

where $H(\cdot)$ is the binary entropy function and the last inequality can be found as Lemma 9.2 in [MU05]. Therefore, the expected number of solutions satisfies

$$\mathbb{E}[S] = \mathbb{E}\left[\sum_{i,j} S_{i,j}\right] = \sum_{i,j} \mathbb{E}[S_{i,j}] \geq \frac{2^{2c+(1-H(c_w))n}}{n+1} = \frac{1}{n+1}.$$

Thus if we increase the initial sizes $|L_1|$, $|L_2|$ by only a polynomial factor of $\theta(\sqrt{n})$, we get a solution in expectation.

Notice that the restriction of Theorem 1 is satisfied since

$$\frac{c}{n} = \frac{1-H(c_w)}{2} < 1 - H(c_w) < 1 - H(\frac{c_w}{2}).$$

Thus, by Theorem 1 our Nearest Neighbor search finds a solution with overwhelming probability in time $2^{(T^*(c_w)+\varepsilon)n}$.

□

Remark 1. We would like to point out that our analysis does *not* rely on a specific Nearest Neighbor algorithm but works for any Nearest Neighbor routine. We just choose May-Ozerov because it currently provides the best asymptotical running-time. In practical applications one might replace May-Ozerov with a simpler Nearest Neighbor routine, see e.g. Section 3.

2.2 An Algorithm for the Approximate 3-list Problem

In this section, we propose an algorithm for the approximate 3-list problem, which gives a (small, but remarkable) exponential improvement for all approximations $0 < c_w < \frac{1}{2}$.

Our algorithm, illustrated in Figure 1, works as follows. We start with two lists L_1, L_2 of size 2^{c_1} and one list L_3 of size 2^{c_2}. In the first step we do a usual 2-list join of L_1, L_2 by finding and adding elements that agree on ℓ bits. This gives us a new list L_{12} with ℓ bits cancelled out. For L_3 we filter out all but those elements, whose ℓ bits in the same positions have a Hamming weight of $\leq c'_w\ell$, resulting in a list L'_3.

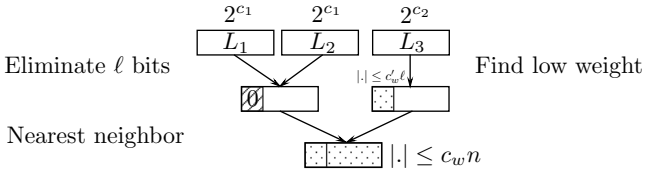

Figure 1: The Approximate 3-List Algorithm

Hence, if we add elements from L_{12} and L'_3, their sum will also have a relative Hamming weight of c'_w in these ℓ positions. We now start a Nearest Neighbor search on the remaining $(n-\ell)$ bits that guarantees the correct Hamming weight for the whole vector.

Theorem 3. *For all $\varepsilon > 0$, the 3-list problem with approximation $c_w \in [0, \frac{1}{2}]$ can in expectation be solved in time and memory*

$$T = M = 2^{(y \cdot T^*(c_w) + \varepsilon)n} \quad \textit{with} \quad y := \begin{cases} \frac{2-2H(c_w)}{2T^*(c_w)-H(c_w)+1} & c_w \leq c_w^* \\ \frac{2}{H(c_w)+4T^*(c_w)+1} & c_w \geq c_w^* \end{cases},$$

where $T^*(c_w)$ is as defined in Theorem 2, and $c_w^* \approx 0.241$ satisfies $T^*(c_w) = \frac{H(c_w)-H(c_w)^2}{4H(c_w)-2}$.

Proof. Let $c_w \leq c_w^*$. We use the previously described algorithm and set

$$c_1 := \tfrac{H(c_w)^2 - H(c_w) + 2T^*(c_w)}{4T^*(c_w) - 2H(c_w) + 2} n, \quad c_2 := y \cdot T^*(c_w) n,$$
$$\ell := \tfrac{H(c_w) + 2T^*(c_w) - 1}{2T^*(c_w) - H(c_w) + 1} n, \quad c'_w = c_w.$$

The lists L_{12}, L'_3 now satisfy

$$\log\left(\mathbb{E}[|L_{12}|]\right) = 2c_1 - \ell = \tfrac{1-H(c_w)}{2}(n-\ell)$$
$$\log\left(\mathbb{E}[|L'_3|]\right) = c_2 + (H(c'_w) - 1) \cdot \ell = \tfrac{1-H(c_w)}{2}(n-\ell),$$

i.e. they are of equal size. Let $T_1 = \max\{2^{c_1}, |L_{12}|\}, T_2 = 2^{c_2}$ be the running time to compute L_{12}, L'_3, respectively.

An application of the approximate 2-list algorithm from Theorem 2 on the $n-\ell$ right-most bits yields a solution in expectation in time $T_3 = 2^{T^*(c_w)(n-\ell)}$. Notice that by our definition of ℓ

$$T_3 = 2^{T^*(c_w)(n-\ell)} = 2^{\frac{2-2H(c_w)}{2T^*(c_w)-H(c_w)+1} T^*(c_w) n} = 2^{c_2} = T_2.$$

Since certainly $T_3 \geq |L_{12}|$, we can conclude that $T_1 \leq \max\{2^{c_1}, T_3\}$. Using $c_w \leq c_w^*$, we further conclude

$$\begin{aligned} & & T^*(c_w) &\leq \frac{H(c_w) - H(c_w)^2}{4H(c_w) - 2} \\ \Rightarrow & & \tfrac{2-2H(c_w)}{2T^*(c_w)-H(c_w)+1} T^*(c_w) &\geq \frac{H(c_w)^2 - H(c_w) + 2T^*(c_w)}{4T^*(c_w) - 2H(c_w) + 2} \\ \Rightarrow & & c_2 &\geq c_1 \end{aligned}$$

Therefore the overall run-time is

$$T = T_2 = 2^{c_2} = 2^{\frac{2-2H(c_w)}{2T^*(c_w)-H(c_w)+1}T^*(c_w)n}, \quad \text{if } c_w \leq c_w^*.$$

This already proofs the run-time for the first case of Theorem 3. Now assume $c_w \geq c_w^*$ and let us choose

$$c_1 := y \cdot T^*(c_w)n, \quad c_2 := \tfrac{(1-H(c_w))(H(c_w)+4T^*(c_w))}{H(c_w)+4T^*(c_w)+1}n,$$
$$\ell := \tfrac{H(c_w)+4T^*(c_w)-1}{H(c_w)+4T^*(c_w)+1}n, \quad c'_w = c_w.$$

Analogously to the reasoning before, we can show that this implies $c_1 \geq c_2$ and

$$T = T_1 = 2^{c_1} = 2^{\frac{2}{H(c_w)+4T^*(c_w)+1}T^*(c_w)n}, \quad \text{if } c_w \geq c_w^*.$$

For both cases the memory consumption is

$$M = \max(2^{c_1}, 2^{c_2}, \mathbb{E}[|L_{12}|], \mathbb{E}[|L_{3'}|]) = \max(T_1, T_2, T_2 \cdot \underbrace{2^{(H(c_w)-1)\ell)}}_{\leq 1}) = T.$$

□

Remark 2. It is not hard to calculate that both running times are equal for $c_w = c_w^*$, making the run-time exponent as a function of c_w a smooth curve, see the graph in Figure 2.

In the proof of Theorem 3 we chose $c'_w = c_w$. The only reason for this choice is to achieve a closed formula for runtime and memory. Via numerical optimizations of c'_w we can further minimize the running time, see the table in Figure 2.

Figure 2 shows the run-time improvement in the exponent of our approximate 3-list algorithm in comparison with the 2-list algorithm.

c_w	$k=2$	$k=3$	$c'_w = c_w$
0.1	0.3004	0.2818	0.2819
0.2	0.1774	0.1558	0.1559
0.3	0.0861	0.0721	0.0774
0.4	0.0244	0.0210	0.0236

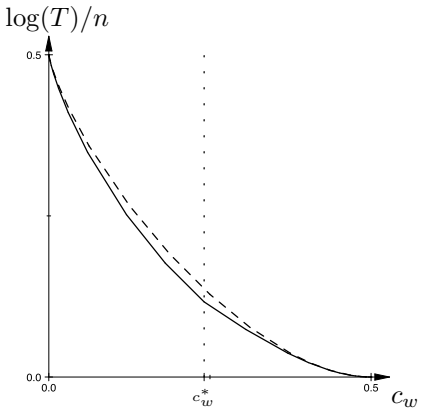

Figure 2: Running time exponent $\frac{\log T}{n}$ of the approximate 3-list algorithm (solid) compared to its approximate 2-list counterpart (dashed).

2.3 The Approximate k-List Algorithm

For generalizing our approximate k-list algorithm to arbitrary powers of two $k = 2^m$, let us first look at the easiest case $k = 4$ (see Figure 3).

2.3.1 The Case $k = 4$.

We start with L_1, L_2, L_3, L_4 of length 2^c. In the first phase we look for $x_1 \in L_1$, $x_2 \in L_2$ s.t. their $\ell_1 + \ell_2$ left-most bits sum to zero. We further look for $x_3 \in L_3$, $x_4 \in L_4$ s.t. their ℓ_1 left-most bits as well as the bits in position $\ell_1 + \ell_2 + 1, \ldots, \ell_1 + 2\ell_2$ sum to zero. The sums $x_1 + x_2$, $x_3 + x_4$ are stored in lists L_{12}, L_{34}.

Now we filter L_{12} (resp. L_{34}) for elements whose bits in positions $\ell_1+\ell_2+1, \ldots, \ell_1+2\ell_2$ (resp. $\ell_1+1, \ldots, \ell_1+\ell_2$) are of relative Hamming weight $\leq c_w$. Let L'_{12} (resp. L'_{34}) denote the resulting list. The sum of all elements $x \in L'_{12}$, $y \in L'_{34}$ is now zero on the ℓ_1 left-most bits and has relative Hamming weight $\leq c_w$ on the bits $\ell_1 + 1, \ldots, \ell_1 + 2\ell_2$. In the last phase we apply Nearest Neighbor search with target weight c'_w on the remaining bits $\ell_1 + 2\ell_2 + 1, \ldots, n$. It remains to determine c, ℓ_1, ℓ_2, c'_w in order to minimize the run-time.

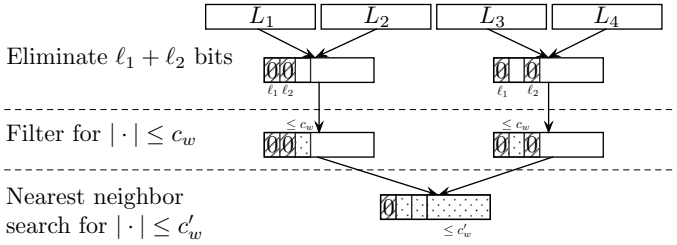

Figure 3: The Approximate k-List Algorithm, $k = 4$

2.3.2 The Case $k = 2^m$.

For bigger k the algorithm consists of three phases (see Figure 4) determined by parameters $\ell_1, \ell_2, \ell_3, \ell_4 \in \mathbb{N}$. We start with $k = 2^m$ lists of length 2^c each. In the first step of the first phase the algorithm pairwise merges the lists into new lists by combining elements $x \in L_{i-1}$, $y \in L_i$, $i = 2, 4, \ldots, k$ s.t. x+y is zero on all bits of the sets $A_{i,1}$, where

$$A_{i,j} = \begin{cases} \{1, \ldots, j\ell_1, (m-2)\ell_1 + \ell_2 + 1, \ldots (m-2)\ell_1 + \ell_2 + j\ell_3\} & i \leq k/2^j \\ \{1, \ldots, j\ell_1, (m-2)(\ell_1+\ell_3) + \ell_2 + \ell_4 + 1, \ldots (m-2)(\ell_1+\ell_3) + \ell_2 + \ell_4 + j\ell_3\} & i > k/2^j \end{cases},$$

$j = 1, \ldots, m-2$. Let $L_{i-1,i}$, $i = 2, 4, \ldots, k$ denote the resulting lists.

This step is repeated $m-2$ times, until we end up with four lists containing only elements whose $(m-2)(\ell_1 + \ell_3)$ bits determined by the sets $A_{i,m-2}$ are zero.

In the second phase we do one more k-list step eliminating $\ell_2+\ell_4$ bits which results in two lists L'_1 and L'_2. L'_1 contains elements with zeros on the bits $1, \ldots, (m-2)(\ell_1+\ell_3)+\ell_2+\ell_4$, while L'_2 contains elements with zeros on the bits $1, \ldots, (m-2)\ell_1 + \ell_2, (m-2)(\ell_1 + \ell_3) + \ell_2 + \ell_4 + 1, \ldots, (m-2)(\ell_1 + 2\ell_3) + \ell_2 + 2\ell_4$. Now we filter L'_1 for elements whose bits $(m-2)(\ell_1+\ell_3)+\ell_2+\ell_4+1, \ldots, (m-2)(\ell_1+2\ell_3)+\ell_2+2\ell_4$ have a relative Hamming weight $\leq c_w$ and do the same for L'_2 on the bits $(m-2)\ell_1 + \ell_2 + 1, \ldots (m-2)(\ell_1 + \ell_3) + \ell_2 + \ell_4$ to create two lists L''_1, L''_2. An application of Nearest Neighbor search with target weight c'_w on the remaining bits, for some c'_w that we will define later, results in a proper solution of total Hamming weight $\leq c_w n$.

Theorem 4. *For all $\varepsilon > 0$ the approximate k-list problem, $k = 2^m$, $m > 1$, with approximation $c_w \in [0, \frac{1}{2}]$ can in expectation be solved in time and memory*

$$T = M = 2^{(y \cdot T^*(c_w) + \varepsilon)n} \quad \textit{with}$$
$$y := \begin{cases} \frac{2-2H(c_w)}{2H(c_w)T^*(c_w) - H(c_w) + 2mT^*(c_w) - 2H(c_w)mT^*(c_w) + 1} & c_w \leq c^*_w \\ \frac{2-H(c_w)}{2mT^*(c_w)+1} & c_w \geq c^*_w \end{cases},$$

where c^*_w satisfies $H(c_w) = \frac{2mT^*(c_w) - 4T^*(c_w) + 1}{2mT^*(c_w) - 2T^*(c_w) + 1}$.

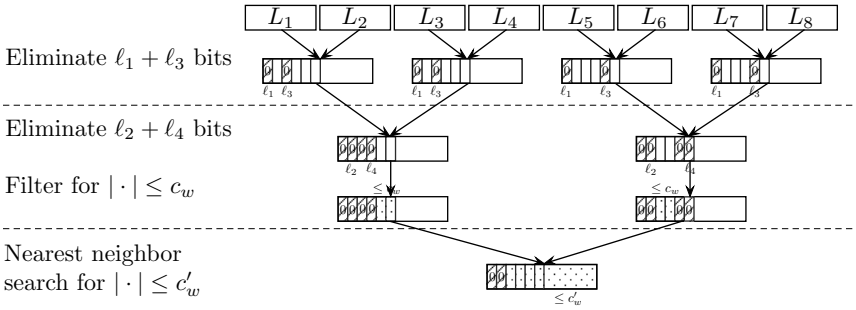

Figure 4: The Approximate k-List Algorithm, $k = 8$

Proof. Let $c_w \geq c_w^*$. We use the previously described algorithm and set

$$c = y \cdot T^*(c_w)n \quad \ell_1 = \ell_2 = 0 \quad \ell_3 = c \quad c'_w = c_w$$
$$\ell_4 = \tfrac{H(c_w)+8T^*(c_w)-4H(c_w)T^*(c_w)-2mT^*(c_w)+2H(c_w)mT^*(c_w)-1}{4mT^*(c_w)+2}n.$$

Thus we eliminate c bits per step in the first phase, i.e. the list sizes stay the same in expectation. We achieve the four resulting lists in time $T_1 = 2^c$. In the second phase ℓ_4 bits are eliminated in time $\max\{2^c, |L'_i|\}$. The two remaining lists L'_1 and L'_2 satisfy

$$\log(\mathbb{E}(|L'_i|)) = 2c - \ell_4 = \frac{1 - H(c_w)}{2}n.$$

The filtering step takes time $2^{\frac{1-H(c_w)}{2}n}$, resulting in two lists L''_1 and L''_2 satisfying

$$\begin{aligned}\log(\mathbb{E}(|L''_i|)) &= \frac{1 - H(c_w)}{2}n + (H(c_w) - 1)((m-2)\ell_3 + \ell_4) \\ &= \frac{1 - H(c_w)}{2}(n - (m-2)c - 2\ell_4).\end{aligned}$$

Thus the second phase overall takes time $T_2 = \max\{2^c, \frac{1-H(c_w)}{2}n\}$. Using the approximate 2-list algorithm of Theorem 2 on the remaining $n-(m-2)\cdot(\ell_1+2\ell_3)-\ell_2-2\ell_4 = n-2(m-2)c-2\ell_4$ bits yields one solution in expectation in time $T_3 = 2^{T^*(c_w)(n-2(m-2)c-2\ell_4)} = 2^c = T_1$. Note that all elements $x \in L''_1$, $y \in L''_2$ sum up to a vector whose left-most $2(m-2)c+2\ell_4$ bits are of relative Hamming weight $\leq c_w$. Therefore the found solution is of Hamming weight $\leq c_w n$. Using $c_w \geq c_w^*$ we conclude

$$\begin{aligned}
\Rightarrow \quad & H(c_w) && \geq \frac{2mT^*(c_w) - 4T^*(c_w) + 1}{2mT^*(c_w) - 2T^*(c_w) + 1} \\
\Rightarrow \quad & \tfrac{2-H(c_w)}{2mT^*(c_w)+1}T^*(c_w)n && \geq \frac{1 - H(c_w)}{2}n \\
\Rightarrow \quad & c && \geq \frac{1 - H(c_w)}{2}n \\
\Rightarrow \quad & T_1 && = T_2.
\end{aligned}$$

Therefore the overall run-time is

$$T = T_1 = 2^c = 2^{\frac{2-H(c_w)}{2mT^*(c_w)+1}T^*(c_w)n}, \quad \text{if } c_w \geq c_w^*$$

proving the second case. Assuming $c_w \leq c_w^*$ we choose

$$\begin{aligned}
c &= y \cdot T^*(c_w)n \\
\ell_1 = \ell_2 &= \tfrac{1-2H(c_w)mT^*(c_w)+2mT^*(c_w)+2H(c_w)T^*(c_w)-4T^*(c_w)-H(c_w)}{(m-1)\cdot(2H(c_w)T^*(c_w)-H(c_w)+2mT^*(c_w)-2H(c_w)mT^*(c_w)+1)}n \\
\ell_3 = \ell_4 &= \tfrac{H(c_w)+2T^*(c_w)-1}{(m-1)\cdot(2H(c_w)T^*(c_w)-H(c_w)+2mT^*(c_w)-2H(c_w)mT^*(c_w)+1)}n.
\end{aligned}$$

Note that $\ell_1 + \ell_3 = c$, i.e. we eliminate c bits per step in the first phase, again resulting in stable list sizes in expectation and a run-time of $T_1 = 2^c$ for this phase. In the second phase another $\ell_2 + \ell_4 = c$ bits are eliminated. The filtering step returns two lists L_1'' and L_2'' satisfying

$$\log(\mathbb{E}(|L_i''|)) = c + (H(c_w) - 1)((m-2)\ell_3 + \ell_4) = \frac{1 - H(c_w)}{2}(n - (m-1)(\ell_1 + 2\ell_3)).$$

This phase takes time $T_2 = 2^c = T_1$. Again Theorem 2 can be applied on the remaining $n - (m-1)(\ell_1 + 2\ell_3)$ bits. Thus, we obtain one solution in expectation in time $T_3 = 2^{T^*(c_w)(n-(m-1)(\ell_1+2\ell_3))} = 2^c = T_1$, i.e. all phases have the same running time.

Note that all elements $x \in L_1''$, $y \in L_2''$ sum up to a vector whose left-most $(m-1)\ell_1$ bits are zero, while the bits $(m-1)\ell_1 + 1, \ldots, (m-1)(\ell_1 + 2\ell_3)$ are of relative Hamming weight at most c_w. Therefore the found solution is of Hamming weight at most $c_w(2(m-2)\ell_3) + c_w(n - (m-1)(\ell_1 + 2\ell_3)) = c_w(n - (m-1)\ell_1) \leq c_w n$.

The algorithm returns a valid solution in time

$$T = 2^c = 2^{\frac{2-2H(c_w)}{2H(c_w)T^*(c_w) - H(c_w) + 2mT^*(c_w) - 2H(c_w)mT^*(c_w) + 1} T^*(c_w) n}$$

for the given parameters. Note that $\ell_1 \geq 0$ is equivalent to $c_w \leq c_w^*$. Thus this is a necessary condition for this choice of parameters. □

Figure 5 visualizes the running times of our approximate k-list algorithm.

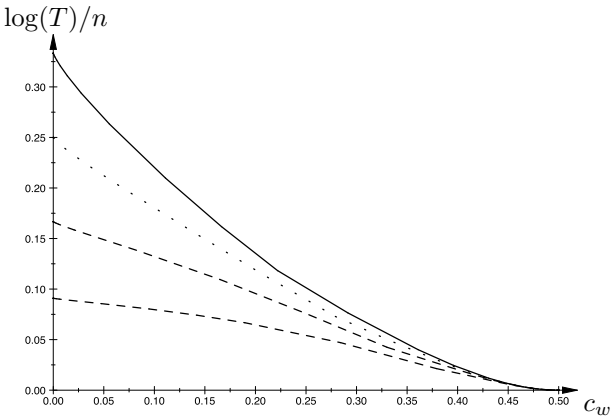

Figure 5: run-time exponent $\frac{\log T}{n}$ of the approximate k-list algorithm for $k = 4$ (solid), $k = 8$ (dotted), $k = 32$ (dashed, upper), $k = 1024$ (dashed, lower).

2.4 Practical Optimizations

In the case $c_w \leq c_w^*$ our approximate k-list algorithm from Section 2.3 returns a solution of weight strictly smaller than $c_w n$ for the given parameters. Obviously this is not optimal – but we sacrificed optimality in Section 2.3 for the sake of obtaining a closed run time formula in Theorem 4.

In this section we choose optimal parameters as follows

$$\ell_1 = c \quad \ell_2 = \ell \quad \ell_3 = \ell_4 = 0$$
$$c_w' = \tfrac{c_w n}{n - (m-2)c - \ell}.$$

Our resulting algorithm consists of several k-list steps and a Nearest Neighbor search at the end (see Figure 6 for the case $k = 8$), where the parameters c, ℓ remain to be specified.

As we eliminate c bits per step in the first phase, the four resulting lists are of expected size 2^c. After eliminating another ℓ bits, we have two lists L'_1, L'_2 satisfying

$$\log\left(\mathbb{E}[|L'_1|]\right) = \log\left(\mathbb{E}[|L'_2|]\right) = 2 \cdot c - \ell.$$

This takes time $T_1 = \max\{2^c, 2^{2c-\ell}\}$. An application of Nearest Neighbor search on L'_1, L'_2 takes time $T_2 = 2^{T_2^*(c'_w)(n-(m-2)c-\ell)}$. The resulting solution is of weight at most $c'_w(n-(m-2)c-\ell) = c_w n$. In order to make the algorithm return a solution in expectation and to optimize for run time, we need to solve

$$\log\left(\mathbb{E}[|L'_i|]\right) = \tfrac{1-H(c'_w)}{2}(n-(m-2)c-\ell)$$
$$T_1 = T_2$$

which gives us

$$c = \tfrac{2T_2^*(c'_w)}{H(c'_w)+2mT_2^*(c'_w)+1}n$$
$$\ell = \tfrac{H(c'_w)+4T_2^*(c'_w)-1}{H(c'_w)+2mT_2^*(c'_w)+1}n.$$

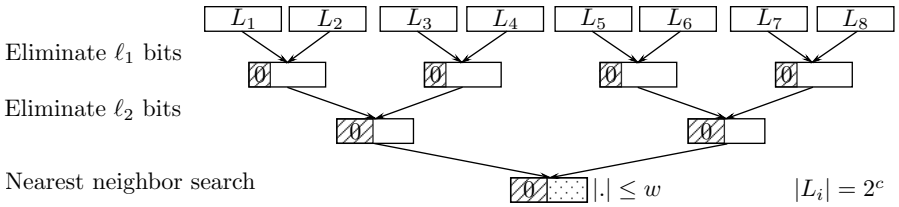

Figure 6: The Approximate k-List Algorithm, $k = 8$

Unfortunately c'_w is now a function of c and ℓ, which prevents us from deriving a closed formula for c, ℓ or T as in Section 2.3. Instead, we determine c, ℓ by numerical optimization to minimize the run time $T = 2^c$. Figures 7 and 8 give optimized run times for various k, c_w in comparison with the results of Section 2.3.

c_w	T. 4	c	ℓ
0.0	0.3333	**0.3333**	0.3333
0.1	0.2199	**0.2001**	0.2294
0.2	0.1349	**0.1204**	0.1504
0.3	0.0716	**0.0611**	0.0815
0.4	0.0228	**0.0190**	0.0268

(a) $k = 4$

c_w	T. 4	c	ℓ
0.0	0.2500	**0.2500**	0.2500
0.1	0.1803	**0.1564**	0.1862
0.2	0.1188	**0.0961**	0.1219
0.3	0.0635	**0.0506**	0.0680
0.4	0.0219	**0.0167**	0.0230

(b) $k = 8$

c_w	T. 4	c	ℓ
0.0	0.1667	**0.1667**	0.1667
0.1	0.1325	**0.1077**	0.1288
0.2	0.0960	**0.0692**	0.0893
0.3	0.0549	**0.0381**	0.0510
0.4	0.0202	**0.0133**	0.0189

(c) $k = 32$

c_w	T. 4	c	ℓ
0.0	0.0909	**0.0909**	0.0909
0.1	0.0797	**0.0619**	0.0755
0.2	0.0649	**0.0417**	0.0510
0.3	0.0430	**0.0240**	0.0313
0.4	0.0169	**0.0092**	0.0122

(d) $k = 1024$

Figure 7: Parameters for different k, c_w, compared to the run-time exponents $\frac{\log(T)}{n}$ given by Theorem 4 (last column).

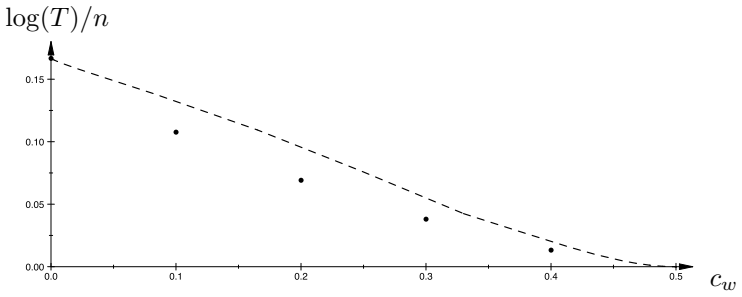

Figure 8: Run-time exponent $\frac{\log(T)}{n}$ of the approximate 32-list algorithm for minimal time (dots) and given by Theorem 4 (curve)).

The above choice of parameters minimizes the run-time. If we want to minimize the memory consumption, we can simply balance the list size over the whole algorithm through setting $\ell = c$. Figures 8 and 9 show the differences between the two optimizations.

	min. time	min. memory	
c_w	c/n	$\log(T)/n$	$\log(M)/n$
0.0	0.3333	0.3333	0.3333
0.1	0.2001	0.2182	0.1876
0.2	0.1204	0.1374	0.1047
0.3	0.0611	0.0712	0.0481
0.4	0.0190	0.0218	0.0129

(a) $k = 4$

	min. time	min. memory	
c_w	c/n	$\log(T)/n$	$\log(M)/n$
0.0	0.2500	0.2500	0.2500
0.1	0.1564	0.1735	0.1461
0.2	0.0961	0.1136	0.0847
0.3	0.0506	0.0612	0.0407
0.4	0.0167	0.0197	0.0117

(b) $k = 8$

	min. time	min. memory	
c_w	c/n	$\log(T)/n$	$\log(M)/n$
0.0	0.1667	0.1667	0.1667
0.1	0.1077	0.1252	0.1021
0.2	0.0692	0.0857	0.0620
0.3	0.0381	0.0484	0.0316
0.4	0.0133	0.0168	0.0098

(c) $k = 32$

	min. time	min. memory	
c_w	c/n	$\log(T)/n$	$\log(M)/n$
0.0	0.0909	0.0909	0.0909
0.1	0.0619	0.0764	0.0591
0.2	0.0417	0.0549	0.0379
0.3	0.0240	0.0327	0.0207
0.4	0.0092	0.0124	0.0072

(d) $k = 1024$

Figure 9: Comparisons of the run-time exponents $\frac{\log(T)}{n}$ and memory consumption exponents $\frac{\log(M)}{n}$ between the time-minimized ($c \neq \ell$, $T = M = 2^c$) and memory-minimized ($c = \ell$, $M = 2^c$, $T = T_2$) choice of parameters.

Remark 3. The analysis in Section 2 only holds when the initial list sizes can be freely chosen. Minder and Sinclair [MS09] showed how to adapt Wagner's k-list algorithm to the case, where one starts with smaller sizes. The same techniques can be adapted here.

2.5 Improvements for $k = 2^m + 2^{m-1}$

We are also able to provide improved algorithms for k of the form $k = 2^m + 2^{m-1}$. In a nutshell these algorithms are natural generalizations of the techniques used for $k = 2^m$ and $k = 3$. Figure 10 and Figure 11 show our algorithms for $k = 6 = 2^2 + 2$ resp. $k = 12 = 2^3 + 2^2$. Figure 12 compares their running-times to our approximate 4 and 8-list

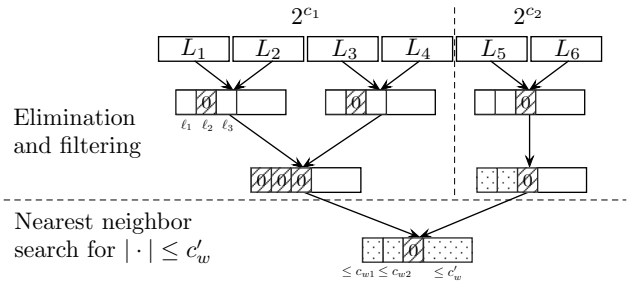

Figure 10: The Approximate k-List Algorithm, $k = 6$

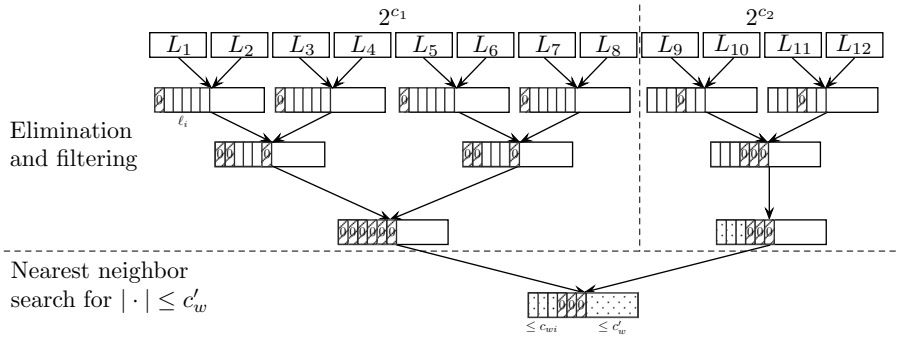

Figure 11: The Approximate k-List Algorithm, $k = 12$

algorithms. Unfortunately we did not find improved algorithms for other choices of k, i.e. we found no improvement for $k = 5$ over $k = 4$, no improvement for $k = 7$ over $k = 6$, and so on. We leave it as an open problem whether improvements are possible in these cases.

c_w	$k = 4$	$k = 6$	$k = 8$	$k = 12$
0.0	0.3333	0.3333	0.2500	0.2500
0.1	0.2001	0.1975	0.1564	0.1532
0.2	0.1204	0.1152	0.0961	0.0926
0.3	0.0611	0.0573	0.0506	0.0479
0.4	0.0190	0.0180	0.0167	0.0159

Figure 12: Running time exponent $\frac{\log T}{n}$ for $k = 4, 6, 8$ and 12.

3 Match-and-Filter algorithm

In this section we give an alternative algorithm for the approximate k-list problem, which we call Match-and-Filter. It has a simple description and is easy to implement. Moreover, its running time is for $k = 2, 3$ better than the results from Section 2.1 and 2.2 (but without closed formula for the run time), and for $k \ge 4$ only slightly worse than the results of Section 2.4.

The Match-and-Filter algorithm adapts ideas from Leurent's near-collision search for hash functions [Leu12] to our approximate k-list setting. Namely, we first match elements via exact collisions on a fraction of all coordinates and we eventually hope that the remaining (so far truncated) part contains vectors with small Hamming weight.

In more detail (see also Figure 13), Match-and-Filter starts with lists $L_1, \ldots, L_k$, $k = 2^m$ with $|L_i| = 2^c$, and merges them pairwise by finding elements that agree on the c

leftmost bits. This is repeated m times until only one list is left that contains elements with zeros on the $m \cdot c$ leftmost bits. Then the algorithm searches for an element of weight w on the remaining rightmost bits.

Note that this technique can also be adapted to the case $k = 2^m + 2^{m-1}$, similar to Section 2.5.

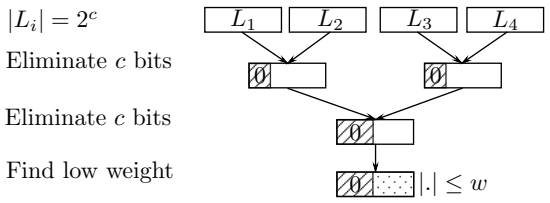

Figure 13: Match-and-Filter algorithm for $k = 4$

The expected size of the lists stays 2^c throughout all but the final step of Match-and-Filter. Therefore, in the final filtered list L' the expected number of solutions is

$$\mathbb{E}[|L'|] = 2^c \cdot \frac{\sum_{i=0}^{w} \binom{n-m\cdot c}{i}}{2^{n-m\cdot c}}.$$

Finding a value for c numerically that satisfies $\mathbb{E}[|L'|] = 1$ gives us the run time $T = 2^c$ and the memory consumption $M = 2^c$. Figure 14 provides the run times for $k = 2, 3, 4, 8, 32, 1024$ in comparison to the results of Sections 2.1, 2.2 and 2.4.

For the cases $k = 2, 3$ the Match-and-Filter algorithm gives better running times. But notice that Match-and-Filter is also slightly more restrictive in the sense that it fixes a larger region of all-zero bits in the target vector. This might be an issue for applications, where one searches for target vectors of a special form.

c_w	$\log(T)/n$	S. 2.1
0.0	0.5000	0.5000
0.1	0.2920	0.3004
0.2	0.1692	0.1774
0.3	0.0814	0.0861
0.4	0.0232	0.0244

(a) $k = 2$

c_w	$\log(T)/n$	S. 2.2
0.0	0.5000	0.5000
0.1	0.2769	0.2818
0.2	0.1590	0.1558
0.3	0.0778	0.0721
0.4	0.0221	0.0210

(b) $k = 3$

c_w	$\log(T)/n$	S. 2.4
0.0	0.3333	0.3333
0.1	0.2040	0.2001
0.2	0.1238	0.1204
0.3	0.0630	0.0611
0.4	0.0195	0.0190

(c) $k = 4$

c_w	$\log(T)/n$	S. 2.4
0.0	0.2500	0.2500
0.1	0.1576	0.1564
0.2	0.0984	0.0961
0.3	0.0518	0.0506
0.4	0.0170	0.0167

(d) $k = 8$

c_w	$\log(T)/n$	S. 2.4
0.0	0.1667	0.1667
0.1	0.1091	0.1077
0.2	0.0704	0.0692
0.3	0.0387	0.0381
0.4	0.0136	0.0133

(e) $k = 32$

c_w	$\log(T)/n$	S. 2.4
0.0	0.0909	0.0909
0.1	0.0623	0.0619
0.2	0.0418	0.0417
0.3	0.0241	0.0240
0.4	0.0092	0.0092

(f) $k = 1024$

Figure 14: Running time exponents $\frac{\log(T)}{n}$ for various k, c_w in comparison to Section 2.

4 Applications

Our work was motivated by [BJMM12, MMT11] which solve some approximate k-list problem (but with a unique solution) in the context of decoding linear codes.

Wagner [Wag02] showed that his original k-list algorithm finds collisions for hash functions that are based on sums, like e.g. AdHash or PCIHF hash. Our approximate k-list algorithm can be used to find near collisions for those hash functions.

Another application from Wagner [Wag02] is the so-called Parity Check Problem. We will show in the following that this problem is indeed rather an instance of the approximate k-list problem, and thus our algorithms improve over the results obtained via the original k-list algorithm.

4.1 Solving the Parity Check Problem

In the Parity Check Problem we are given an irreducible polynomial $P(X)$ of degree n over $\mathbb{F}_2$, and we want to find a multiple $Q(X)$ of $P(X)$ of weight smaller than $d \ll n$ and degree smaller than $N \gg n$. We show now how this problem relates to the approximate k-list problem. As we cannot easily compute the polynomial overhead of May-Ozerov's nearest neighbor, we use our simpler algorithm from Section 3.

Let $\mathbb{F} := \mathbb{F}_2[X]/P[X]$ be a finite field of size 2^n. $|P(X)|$ denotes the weight of a polynomial $P(X)$, i.e. the number of non-zero coefficients. The sum of two elements corresponds to the sum of their coefficient vectors. We choose $k = 2^m = 2^{\lfloor \log d \rfloor}$ as the number of lists and fill them with 2^c elements of the form

$$X^a \mod P(X) \in \mathbb{F}, \quad a \in \{1, 2, \ldots, N\}.$$

Now we run m k-list steps in order to eliminate c bits per step to eventually create one list L. This list contains elements of the form $X^{a_1} + \ldots + X^{a_k}$ whose $m \cdot c$ leftmost coefficients are zero. In our analysis in this section, we assume this takes time and memory (# number of lists $\cdot$ size of a list) per step.

In the last step, we filter L to find a small-weight polynomial

$$Q_1(X) := X^{a_1} \oplus \ldots \oplus X^{a_k} = Q_2(X) \mod P(X),$$

where $Q_2(X)$, $|Q_2(X)| \leq w := d - k$. The number of solutions in the filter list L' is on expectation

$$\mathbb{E}[|L'|] = \mathbb{E}[|L|] \cdot \frac{\sum_{i=0}^{w} \binom{n-mc}{i}}{2^{n-mc}} = 2^{(m+1)c-n} \cdot \sum_{i=0}^{w} \binom{n-mc}{i}.$$

Furthermore it follows that

$$Q(X) := Q_1(X) \oplus Q_2(X) = 0 \mod P(X),$$

i.e. $Q(X)$ is a multiple of $P(X)$ and $|Q(X)| \leq |Q_1(X)| + |Q_2(X)| \leq d$. We have $a_i \leq N$ and $\deg(Q_2(X)) \leq n$, because $Q_2(X)$ is reduced modulo $P(X)$. Hence, $Q(X)$ is of degree at most N, as required. Solving $\mathbb{E}[|L'|] = 1$ gives us the required initial list size 2^c and therefore the degree $N = \lceil 2^c \rceil$.

4.2 Comparison with [Wag02] and [MS09].

Wagner's k-list algorithm [Wag02] returns a polynomial of degree $N = 2^{\frac{n}{\log k+1}}$ ($\widehat{=}$ the initial list size) determined by the number of lists k in time and memory kN.

The extended k-list algorithm of Minder and Sinclair [MS09] works for smaller list sizes than Wagner's algorithm, automatically resulting in lower degree polynomials. Actually the effects of using Minder-Sinclair for the parity check problem have not been discussed in the cryptographic literature so far. We provide the results in Figure 15, where we also compare to [Wag02] and our Match-and-Filter algorithm.

For a fixed degree (to the value of Minder-Sinclair), our algorithm gives better run time/memory consumptions. When we fix the time/memory complexity (again to the value of Minder-Sinclair), our Match-and-Filter algorithm provides a better degree.

Furthermore, the Minder-Sinclair algorithm only achieves a minimum degree of $2^{n/k}$, while we can go below this bound for $d \neq 2^m$. The reason is that our search for approximations returns more solutions than an exact collision search, and therefore the initial lists are usually smaller.

		Wagner		Minder-S.		Match-and-Filter			
						fixed deg		fixed T/M	
d	k	deg	T/M	deg	T/M	deg	**T/M**	**deg**	T/M
2	2	60	61	60	61	60	**61**	**60**	61
3	2	60	61	60	61	57	**58**	**57**	58
4	4	40	42	40	42	40	**42**	**40**	42
5	4	40	42	39	43	39	**41**	**36**	43
6	4	40	42	37	47	37	**39**	**32**	47
7	4	40	42	36	49	36	**38**	**28**	49
8	8	30	33	30	33	30	**33**	**30**	33
9	8	30	33	29	33	29	**32**	**27**	33
15	8	30	33	24	38	24	**27**	**14**	38

Figure 15: Comparison of the bit complexities of the algorithms k-list [Wag02], extended k-list [MS09] and our Match-and-Find for finding a multiple of a polynomial of degree $n = 120$.

The main application of the parity check problem is fast correlation attacks on stream ciphers [MS89, CT00, CJM02, LJ14, ZXM15]. Here it is of major importance to obtain low degree polynomials, because the degree determines the number of output bits an attacker has to know. Thus investing more time for finding suitable polynomials usually pays off in total. For this task our algorithm improves over Wagner and Minder Sinclair. Currently most fast correlation attacks are restricted to weight 4 or 5, while our algorithm provides more flexibility for the choice of weights.

Acknowledgement

We would like to thank Gaëtan Laurent for pointing out the relation between near collisions of hash functions and the approximate k-list problem, which led to the Match-and-Filter algorithm in Section 3. We also thank the anonymous FSE reviewers for their detailed comments that improved and clarified our work.

References

[AFFP14] Martin R. Albrecht, Jean-Charles Faugère, Robert Fitzpatrick, and Ludovic Perret. Lazy modulus switching for the BKW algorithm on LWE. In Hugo Krawczyk, editor, *PKC 2014: 17th International Conference on Theory and Practice of Public Key Cryptography*, volume 8383 of *Lecture Notes in Computer Science*, pages 429–445, Buenos Aires, Argentina, March 26–28, 2014. Springer, Heidelberg, Germany.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *33rd Annual ACM Symposium on Theory of Computing*, pages 601–610, Crete, Greece, July 6–8, 2001. ACM Press.

[BCJ11] Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

[Ber07] Daniel J. Bernstein. Better price-performance ratios for generalized birthday attacks. In *Workshop Record of SHARCS'07: Special-purpose Hardware for Attacking Cryptographic Systems*, 2007. http://cr.yp.to/rumba20/genbday-20070719.pdf.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *32nd Annual ACM Symposium on Theory of Computing*, pages 435–440, Portland, Oregon, USA, May 21–23, 2000. ACM Press.

[CJ04] Jean-Sebastien Coron and Antoine Joux. Cryptanalysis of a provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2004/013, 2004. http://eprint.iacr.org/2004/013.

[CJM02] Philippe Chose, Antoine Joux, and Michel Mitton. Fast correlation attacks: An algorithmic point of view. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany.

[CT00] Anne Canteaut and Michaël Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588, Bruges, Belgium, May 14–18, 2000. Springer, Heidelberg, Germany.

[GJL14] Qian Guo, Thomas Johansson, and Carl Löndahl. Solving LPN using covering codes. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 1–20, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[GJS15] Qian Guo, Thomas Johansson, and Paul Stankovski. Coded-BKW: Solving LWE using lattice codes. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 23–42, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[HGJ10] Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[KF15] Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 43–62, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[KPP14] T. Kopelowitz, S. Pettie, and E. Porat. Higher Lower Bounds from the 3SUM Conjecture. *ArXiv e-prints*, July 2014.

[KS01] Ravi Kumar and D. Sivakumar. On polynomial approximation to the shortest lattice vector length. In S. Rao Kosaraju, editor, *12th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 126–127, Washington, DC, USA, January 7–9, 2001. ACM-SIAM.

[Leu12] Gaëtan Leurent. Time-memory trade-offs for near-collisions. Cryptology ePrint Archive, Report 2012/731, 2012. http://eprint.iacr.org/2012/731.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In Roberto De Prisco and Moti Yung, editors, *SCN 06: 5th International Conference on Security in Communication Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 348–359, Maiori, Italy, September 6–8, 2006. Springer, Heidelberg, Germany.

[LJ14] Carl Löndahl and Thomas Johansson. Improved algorithms for finding low-weight polynomial multiples in $f_2[x]$ and some cryptographic applications. *Des. Codes Cryptography*, 73(2):625–640, 2014.

[LR11] Mario Lamberger and Vincent Rijmen. Optimal covering codes for finding near-collisions. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *SAC 2010: 17th Annual International Workshop on Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 187–197, Waterloo, Ontario, Canada, August 12–13, 2011. Springer, Heidelberg, Germany.

[Lyu05] Vadim Lyubashevsky. On random high density subset sums. In *Proceedings of APPROX-RANDOM 2005*, pages 378–389. Springer-Verlag, 2005.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\wr(2^{0.054n})$. In *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 107–124, 2011.

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 203–228, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[MS89] Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989.

[MS09] Lorenz Minder and Alistair Sinclair. The extended k-tree algorithm. In Claire Mathieu, editor, *20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 586–595, New York, NY, USA, January 4–6, 2009. ACM-SIAM.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[NS15] Ivica Nikolic and Yu Sasaki. Refinements of the k-tree algorithm for the generalized birthday problem. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 683–703, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

[Wag02] David Wagner. A generalized birthday problem. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany.

[ZJW16] Bin Zhang, Lin Jiao, and Mingsheng Wang. Faster algorithms for solving LPN. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 168–195, 2016.

[ZXM15] Bin Zhang, Chao Xu, and Willi Meier. Fast correlation attacks over extension fields, large-unit linear approximation and cryptanalysis of SNOW 2.0. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.