UNIVERSITÀ POLITECNICA DELLE MARCHE
Repository ISTITUZIONALE

Computational Hardness of the Permuted Kernel and Subcode Equivalence Problems

This is the peer reviewd version of the followng article:

Original
Computational Hardness of the Permuted Kernel and Subcode Equivalence Problems / Santini, Paolo; Baldi, Marco; Chiaraluce, Franco. - In: IEEE TRANSACTIONS ON INFORMATION THEORY. - ISSN 0018-9448. - ELETTRONICO. - 70:3(2024), pp. 2254-2270. [10.1109/TIT.2023.3323068]

Availability:
This version is available at: 11566/325875 since: 2024-05-19T17:17:07Z

Publisher:

Published
DOI:10.1109/TIT.2023.3323068

Terms of use:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. The use of copyrighted works requires the consent of the rights' holder (author or publisher). Works made available under a Creative Commons license or a Publisher's custom-made license can be used according to the terms and conditions contained therein. See editor's website for further information and terms and conditions.
This item was downloaded from IRIS Università Politecnica delle Marche (https://iris.univpm.it). When citing, please refer to the published version.

note finali coverpage

(Article begins on next page)

18 July 2024

© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Computational Hardness of the Permuted Kernel and Subcode Equivalence Problems[⋆]

Paolo Santini, Marco Baldi, and Franco Chiaraluce

Polytechnic University of Marche, Ancona, Italy
{p.santini, m.baldi, f.chiaraluce}@univpm.it

Abstract. The Permuted Kernel Problem (PKP) asks to find a permutation which maps an input matrix into the kernel of some given vector space. The literature exhibits several works studying its hardness in the case of the input matrix being mono-dimensional (i.e., a vector), while the multi-dimensional case has received much less attention and, de facto, only the case of a binary ambient finite field has been studied. The Subcode Equivalence Problem (SEP), instead, asks to find a permutation so that a given linear code becomes a subcode of another given code. At the best of our knowledge, no algorithm to solve the SEP has ever been proposed. In this paper we study the computational hardness of solving these problems. We first show that, despite going by different names, PKP and SEP are exactly the same problem. Then we consider the state-of-the-art solver for the mono-dimensional PKP (namely, the KMP algorithm, proposed by Koussa, Macario-Rat and Patarin), generalize it to the multi-dimensional case and analyze both the finite and the asymptotic regimes. We further propose a new algorithm, which can be thought of as a refinement of KMP. In the asymptotic regime our algorithm does not improve on KMP but, in the finite regime (and for parameters of practical interest), we achieve significant improvements, especially for the multi-dimensional version of PKP. As an evidence, we show that it is the fastest algorithm to attack several recommended instances of cryptosystems based on PKP. As a side-effect, given the mentioned equivalence between PKP and SEP, all the algorithms we analyze in this paper can be used to solve instances of the latter problem.

1 Introduction

The Permuted Kernel Problem (PKP) is an NP-complete problem [14] which, in its more general formulation, asks to find a permutation of a given $\ell \times n$ matrix $\mathbf{V}$ which belongs to the right kernel of another, given, $m \times n$ matrix $\mathbf{A}$. The most studied case for the PKP is the one in which the input matrix $\mathbf{V}$ has only one row (i.e., $\ell = 1$), hence it is a vector; we will refer to this case as the mono-dimensional PKP.

The use of the mono-dimensional PKP in cryptography has been introduced by Shamir in 1989 [28]; recently, the problem has been employed to build post-quantum signature schemes [1, 7–9, 13]. These schemes have some very good features, such as competitive signature sizes (8.9 kB for 128 bits of security [9]) and compatibility with the MPC-in-the-Head paradigm [13], where MPC stays for multiparty computation. As it is well known, the construction of a satisfying post-quantum signature scheme is still an open problem, which is the reason why the National Institute of Standards and Technology (NIST) issued an additional round, specifically tailored to signatures. To contextualize: SPHINCS+, which has been identified for standardization during the third round of the NIST PQC Standardization process, has signatures of approximately 17 kB (for the fast version) or 8 kB (for the short version). The schemes in [9] and [13] already compare favourably with SPHINCS+ as well as other state-of-the-art post-quantum schemes. Consequently, PKP-based signatures are likely to be viable and competitive solutions.

Given this state of affairs, improving and consolidating our understanding about the practical hardness to solve PKP is definitely an important goal. The mono-dimensional variant has been extensively studied over time [4, 15, 18, 19, 22, 24, 25]. Prior to [25], the solver with the lowest time complexity was the Koussa-Macario-Rat-Patarin (KMP) algorithm [19], which consists in a brute-force search plus standard optimizations, such as reducing to a small instance and employing a meet-in-the-middle approach. In our recent work [25], we have proposed an improvement of the

[⋆] Part of the material in this paper has been presented at the 2022 IEEE International Symposium on Information Theory (ISIT), Espoo, Finland [25].

KMP algorithm, describing a strategy based on kernel subspaces with small support, i.e., containing vectors that have always zero entries in several positions[1]. These subspaces correspond to kernel equations involving an abnormally small number of coordinates and, consequently, are somewhat easier to solve. Such equations can be used to implement an initial filtering stage, after which the KMP algorithm is executed: this allows excluding some of the candidates which the KMP algorithm would consider and, in several cases, leads to a slight but nontrivial speed-up.

Perhaps surprisingly, the multi-dimensional PKP (i.e., the case in which the input $\mathbf{V}$ has $\ell > 1$ rows) has received much less attention. Only recently a digital signature based on such a problem has appeared [1]: the scheme, called PERK, exploits the increased value of ℓ to amplify the challenge space which, in turn, results in a lower soundness error. Prior to [1], the multi-dimensional PKP has been considered only in [20]. The recommended instances are for the binary case (i.e., the ambient finite field is binary) and have been cryptanalyzed in [21] using a coding theory approach: roughly speaking, the attack first searches for low-weight codewords and then uses them to efficiently reconstruct the target permutation. Note that the algorithm in [21] is specific for the binary case.

Another interesting and quite recent problem is the Subcode Equivalence Problem (SEP). On input two linear codes, the SEP asks to find a permutation mapping one into a subcode of the other, and has been proven to be NP-complete in [5]. There already exist some cryptographic proposals based on SEP: for instance, [16] and the quasi-dyadic specialization in [10]. At the best of our knowledge, currently there is no proposed algorithm to solve SEP; hence, the practical hardness of this problem is currently unknown.

Our contribution As highlighted above, there exist some connections between the PKP and problems from coding theory. Indeed, both [21] and [25] solve PKP employing coding concepts and algorithms. In this paper, we provide some further contributions along this line of research. We first show that PKP and SEP are actually the very same problem: this somehow justifies the fact that, to solve the PKP, one can borrow concepts from coding theory. As a consequence, all the algorithms we discuss in this paper (which we refer to as PKP solvers) can also be employed to solve SEP.

Then, we delve into the study of techniques for solving the PKP. We first consider the KMP algorithm and study its performance in the asymptotic regime. We show that, for growing n, the time complexity of KMP is a super-exponential function of n and is constant in ℓ. This holds only when the finite field size q is chosen adaptively as a function of n and ℓ, which implies that it is the smallest value for which PKP has on average a unique solution. In such a regime, q decays exponentially with ℓ, but the number of equations we dispose of increases with ℓ: these two effects compensate one each other, so that there is no dependency on ℓ. Notice that the asymptotic regime predominates only for very large values of n and, for parameters of practical interest, the time complexity can still vary considerably with ℓ. We also show that the instances with q chosen adpaptively with n and ℓ are the hardest to solve, since for different regimes (e.g., fixed q and growing ℓ) the time complexity of the KMP algorithm becomes lower.

We then improve the analysis of the solver in [25] and generalize it to the multi-dimensional case. We show that such an algorithm is not better than KMP in the asymptotic regime but, in the finite regime, it can achieve non trivial speed-ups with respect to KMP. As concrete examples, we show that for some recommended PKP instances this algorithm has a time complexity which is lower than that of KMP and [21]. Namely, for the mono-dimensional, 128-bits and 192-bits instances of PKP-DSS [8], our algorithm is slightly faster than KMP. For the multi-dimensional instances recommended in [20], our algorithm is faster than both KMP and [21]. In addition, our approach is more general since it works regardless of the finite field size. The algorithms we discuss in this paper has already been used as a basis for parameter selection for PERK [1].

We also provide an open source proof-of-concept implementation of our algorithm[2], which can be used to validate the theoretical analysis (and some of the employed heuristics) for small parameters, along with an open source software implementation of all the expressions we have used for deriving numerical results.

[1] Note that a similar idea was already briefly mentioned in [19], but the authors concluded that finding kernel equations with the desired properties was not feasible. In [25], we have shown that such equations can instead be efficiently found and employed.

[2] Source code available at `https://github.com/secomms/pkpattack/`

Paper organization In Section 2 we establish the notation and recall some background notions about linear codes. In Section 3 we state useful facts about subcodes, recalling and enriching some notions from [25]. The equivalence between PKP and SEP is presented and discussed in Section 4. The KMP algorithm and its asymptotic behavior are analyzed in Section 5. The generalization of the algorithm in [25] is presented and discussed in Section 6, which also provides a comparison of the considered PKP solvers. Finally, in Section 7, we briefly comment about how PKP and SEP behave in terms of input sizes, and compare them with other well known problems (say, the Syndrome Decoding Problem (SDP)). Finally, Section 8 concludes the paper.

2 Notation and background

In this section we settle the notation we use throughout the paper and recall some background concepts about linear codes.

2.1 Notation

As usual, $\mathbb{F}_q$ denotes the finite field with q elements; in this paper we focus only on the case of q being a prime. Bold lowercase (resp., uppercase) letters indicate vectors (resp., matrices). Given $\mathbf{a}$ (resp., $\mathbf{A}$), a_i (resp., $a_{i,j}$) denotes the entry in position i (resp., the entry in the i-th row and j-th column). With some abuse of notation, we use $\mathrm{GL}_{m,n}$ to indicate the set of $m \times n$ matrices over $\mathbb{F}_q$, with $m \leqslant n$, having full rank m. The identity matrix of size n is denoted as $\mathbf{I}_n$, while $\mathbf{0}$ denotes the all-zero matrix (the sizes will always be clear from the context). Given a set (or a list) A, $|A|$ denotes its cardinality (i.e., the number of elements). If a is a random variable (or a quantity that depends on some random variables), we write $a \doteq x$ if it has average value x. Given a matrix $\mathbf{A}$ and a set J, $\mathbf{A}_J$ is the matrix formed by the columns of $\mathbf{A}$ that are indexed by J; analogous notation, but referred to elements instead of columns, is used for vectors. We denote by $\mathsf{RREF}(\mathbf{A}, J)$ the algorithm that computes the Row Reduced Echelon Form (RREF) of $\mathbf{A}$ with respect to the set J, i.e., outputs $\mathbf{A}_J^{-1}\mathbf{A}$ if $\mathbf{A}_J$ is square and non singular, and otherwise returns a failure. We use S_n to denote the group of length-n permutations. Given $\mathbf{a} = (a_1, \cdots, a_n)$ and $\pi \in S_n$, we write $\pi(\mathbf{a}) = \big(a_{\pi(1)}, \cdots, a_{\pi(n)}\big)$. Analogous notation is employed for matrices, so that $\pi(\mathbf{A})$ is the matrix obtained by permuting the columns of $\mathbf{A}$ according to π. Given $\mathbf{a}$, $\mathbf{b}$, with some abuse of notation, we define $\mathbf{a} \cap \mathbf{b}$ as the set of entries appearing in both $\mathbf{a}$ and $\mathbf{b}$. We extend the notation to matrices, i.e., $\mathbf{A} \cap \mathbf{B}$ is the set of columns which are in both $\mathbf{A}$ and $\mathbf{B}$. For a matrix $\mathbf{A}$ with n columns with no repeated column, we define $\mathscr{S}_u(\mathbf{A})$ as the set of length-u matrices with columns picked from those of $\mathbf{A}$. Notice that $|\mathscr{S}_u(\mathbf{A})| = \frac{n!}{(n-u)!}$.

2.2 Linear codes

A linear code $\mathcal{C} \subseteq \mathbb{F}_q^n$ with dimension k, redundancy $r = n - k$ and rate $R = k/n$ is a linear k-dimensional subspace of $\mathbb{F}_q^n$. Any code admits two equivalent representations: a *generator matrix*, that is, any $\mathbf{G} \in \mathrm{GL}_{k,n}$ such that $\mathcal{C} = \{\mathbf{uG} \mid \mathbf{u} \in \mathbb{F}_q^k\}$, and a *parity-check matrix*, that is, any $\mathbf{H} \in \mathrm{GL}_{r,n}$ such that $\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_q^n \mid \mathbf{cH}^\top = \mathbf{0}\}$ (where $\top$ denotes transposition). If $\mathbf{G}$ generates $\mathcal{C}$, then any $\mathbf{SG}$, with $\mathbf{S} \in \mathrm{GL}_{k,k}$, is a generator for $\mathcal{C}$; the same holds for parity-check matrices, i.e., $\mathbf{H}$ and $\mathbf{SH}$, with $\mathbf{S} \in \mathrm{GL}_{r,r}$, are parity-check matrices for the same code. Given a vector $\mathbf{x} \in \mathbb{F}_q^n$, its *syndrome* is $\mathbf{s} = \mathbf{xH}^\top$: codewords are vectors having null syndrome. The dual of $\mathcal{C}$, that we denote by $\mathcal{C}^\perp$, is the space generated by any parity-check matrix $\mathbf{H}$. For any codeword $\mathbf{c} \in \mathcal{C}$ and any $\mathbf{b} \in \mathcal{C}^\perp$, we have $\mathbf{cb}^\top = 0$. The support of $\mathcal{C}$, that we indicate as $\mathrm{Supp}(\mathcal{C})$, is the set of indexes i such that there is at least one codeword $\mathbf{c} \in \mathcal{C}$ with $c_i \neq 0$. A subcode $\mathcal{B} \subseteq \mathcal{C}$ with dimension k' is a k'-dimensional linear subspace of $\mathcal{C}$. The number of such subcodes is counted by $\left[\begin{smallmatrix} k \\ k' \end{smallmatrix}\right]_q = \prod_{i=0}^{k'-1} \frac{1-q^{k-i}}{1-q^{i+1}}$. Any subspace with dimension 1 is the orbit of a codeword, under scalar multiplications by the elements in $\mathbb{F}_q$: the number of such subcodes is given by $\left[\begin{smallmatrix} k \\ 1 \end{smallmatrix}\right]_q = \frac{q^k-1}{q-1}$.

2.3 Useful approximations and asymptotics

In this section we recall some well known asymptotics and approximations for quantities that will arise naturally in our treatment (for more details about these estimates see, for instance, [17]).

For two functions $f(n)$ and $g(n)$, we write $f(n) \sim g(n)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$. From Stirling's approximation for factorials, we have

$$n! \sim \frac{1}{\sqrt{2\pi n}} \left(\frac{n}{e}\right)^n = 2^{n\left(\log_2(n) - \log_2(e)\right)\left(1+o(1)\right)}. \tag{1}$$

For any integer $q \geqslant 2$ and $x \in [0;1]$, the q-ary entropy function is defined as

$$h_q(x) = x\log_q(q-1) - x\log_q(x) - (1-x)\log_q(1-x).$$

Let $\omega \in [0;1]$ be a constant; then

$$\binom{n}{\omega n} = 2^{nh_2(\omega)\left(1+o(1)\right)}, \tag{2}$$

$$\frac{n!}{(n-\omega n)!} = 2^{n\left(h_2(\omega)+\omega\log_2(\omega n/e)\right)\left(1+o(1)\right)}. \tag{3}$$

The number of vectors with length n, values over $\mathbb{F}_q$ and Hamming weight ωn is then

$$\binom{n}{\omega n}(q-1)^{\omega n} = 2^{nh_q(\omega)\log_2(q)\left(1+o(1)\right)}. \tag{4}$$

Finally, for constant $d \in \mathbb{N}$ and $\omega \in \mathbb{R}$, $\omega \in [0;1]$, we have

$$\frac{\left[\begin{smallmatrix}\omega n\\ d\end{smallmatrix}\right]_q}{\left[\begin{smallmatrix}n\\ d\end{smallmatrix}\right]_q} \sim q^{-d(1-\omega)n}. \tag{5}$$

3 Subcodes with small support

In this section we discuss the properties of small support subcodes, which will have a fundamental role in the analysis of the algorithms we will describe in the following.

3.1 Number of subcodes with small support

For a random code, the average number of subcodes with dimension d and support size w can be bounded thanks to the following theorem [25].

Theorem 1 *For a code $\mathcal{C} \subseteq \mathbb{F}_q^n$, we define $\mathcal{A}_{w,d}(\mathcal{C})$ as the set of subcodes of $\mathcal{C}$ with dimension d and support size w. Let $N_k(w,d)$ be the average value of $|\mathcal{A}_{w,d}(\mathcal{C})|$, when $\mathcal{C}$ is picked at random among all codes with dimension k. Then $\widecheck{N}_k(w,d) \leqslant N_k(w,d) \leqslant \widehat{N}_k(w,d)$, with*

$$\widehat{N}_k(w,d) = \binom{n}{w}\frac{(q^d-1)^w}{\prod_{i=0}^{d-1}(q^d-q^i)}\frac{\left[\begin{smallmatrix}k\\ d\end{smallmatrix}\right]_q}{\left[\begin{smallmatrix}n\\ d\end{smallmatrix}\right]_q},$$

$$\widecheck{N}_k(w,d) = \binom{n}{w}(q^d-1)^{w-d}\frac{\left[\begin{smallmatrix}k\\ d\end{smallmatrix}\right]_q}{\left[\begin{smallmatrix}n\\ d\end{smallmatrix}\right]_q}.$$

Proof. See [25]. □

Remark 1. The bounds in Theorem 1 are tight, up to a factor which tends asymptotically to 1 as q grows. Indeed, consider that

$$\frac{\widehat{N}_k(w,d)}{\widecheck{N}_k(w,d)} = \frac{(q^d-1)^d}{\prod_{i=0}^{d-1} q^d - q^i} = \frac{(q^d-1)^d}{q^{d^2}\prod_{i=1}^{d}(1-q^{-i})}.$$

Since $e^{-\frac{1}{x}} \approx 1 - \frac{1}{x}$ if x is sufficiently large, we have $1 - q^{-i} \approx e^{-\frac{1}{q^i}}$ and

$$\begin{aligned}\frac{\widehat{N}_k(w,d)}{\widecheck{N}_k(w,d)} &\approx \frac{(q^d-1)^d}{q^{d^2}\prod_{i=1}^{d} e^{-\frac{1}{q^i}}} = (q^d-1)^d q^{-d^2} e^{\sum_{i=1}^{d}\frac{1}{q^i}} \\ &\leqslant e^{\sum_{i=1}^{d}\frac{1}{q^i}} = e^{\left(\frac{1-q^{-d-1}}{1-q^{-1}}-1\right)} = e^{\frac{q^d-1}{(q-1)q^d}} \leqslant e^{\frac{1}{q-1}}.\end{aligned}$$

The above quantity is independent of the code length and the desired support size, and tends to 1 as q increases.

Since $\widecheck{N}_k(w,d) \leqslant N_k(w,d) \leqslant \widehat{N}_k(w,d)$, the value of $\widecheck{N}_k(w,d)$ constitutes a tight estimate for $N_k(w,d)$, with an error that depends only on the finite field size and is never greater than $e^1 \approx 2.718$. Indeed, it holds that

$$\frac{N_k(w,d)}{\widecheck{N}_k(w,d)} \leqslant \frac{\widehat{N}_k(w,d)}{\widecheck{N}_k(w,d)} \leqslant e^{\frac{1}{q-1}} \leqslant e^1 \approx 2.718.$$

Notice that the estimate gets better when q increases: for instance, if $q = 251$, we have $e^{\frac{1}{q-1}} \approx 1.004$.

Starting from Theorem 1, one can derive the minimum expected support size for d-dimensional subcodes, when dealing with random codes. To this end, we consider the following proposition.

Proposition 1 *Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be a random code with rate R. Let $d \in \mathbb{N}$ be constant, and $\omega^* \in [0;1]$ such that $\omega^* = \min\{\omega \in [0;1] \mid N_k(\omega n, d) \geqslant 1\}$. Then*

$$\omega^* \approx h_{q^d}^{-1}\left(1 - R + \frac{d}{n}\right).$$

Proof. See Appendix A. □

3.2 Finding subcodes with small support

In [25], we considered an adaptation of Prange's Information Set Decoding (ISD) to find subcodes with small support[3]. In principle, the algorithm may be improved by considering techniques which are normally employed for standard ISD algorithms (e.g., partial Gaussian elimination and a meet-in-the-middle search). Yet, in this paper we do not focus on such aspects, hence we refer to the algorithm in [25], whose time complexity is recalled in the next proposition. For the sake of completeness, the algorithmic procedure of ISD, as well as the proof of the proposition, are shown in Appendix B.

Proposition 2 ***Time complexity of ISD***
We consider ISD as an algorithm that, on input $\mathcal{C} \subseteq \mathbb{F}_q^n$ with dimension k and integers $w, d \in \mathbb{N}$ such that $w \leqslant n + d - k$, returns a random element from $\mathcal{A}_{w,d}(\mathcal{C})$ with average running time

$$T_{\mathsf{ISD}}^{(d)}(n,k,w) = O\left(\frac{k^3 + \binom{k}{d}}{p^{(d)}(n,k,w)}\right),$$

with $p^{(d)}(n,k,w) = \min\left\{\frac{\binom{w}{d}\binom{n-w}{k-d}}{\binom{n}{k}}\widecheck{N}_k(w,d)\ ;\ 1\right\}$.

Proof. See Appendix B. □

4 The equivalence between PKP and SEP

In this section we recall the PKP and the SEP, and show the connections between this latter problem and another code-based problem, namely, the Permutation Equivalence Problem (PEP). We proceed by describing a reduction from SEP to PEP, which gives us a first way to solve SEP. This reduction is interesting since it bridges two different code-based problems but, in practice, is not efficient since it runs in exponential time. Finally, we show that PKP and SEP are equivalent, namely, they are different formulations of the very same problem. This unlocks the possibility of solving SEP with solvers for PKP.

[3] The use of ISD to find subcodes has been firstly proposed in [6]. In this paper (and in [25]) we consider the very same algorithm, with the only (minor) difference that we do not fix the subcode dimension to be equal to 2.

Algorithm 1: Reduction of SEP to PEP

```
Input: $\mathbf{G} \in \mathbb{F}_q^{k \times n}$, $\mathbf{G}' \in \mathbb{F}_q^{k' \times n}$
Output: permutation $\pi \in S_n$ and matrix $\mathbf{S} \in \mathrm{GL}_{k',k}$ such that $\mathbf{G}' = \mathbf{SGP}$

1 Choose uniformly at random $\mathbf{S} \in \mathrm{GL}_{k',k}$;
2 Set $\widetilde{\mathbf{G}} = \mathbf{SG}$;
3 Try to solve the PEP on input $\{\widetilde{\mathbf{G}}, \mathbf{G}'\}$;
4 If a solution $\pi \in S_n$ is found, return $\{\mathbf{S}; \pi\}$, else restart from step 1.
```

4.1 PKP, SEP and PEP

We start by introducing the PKP in its most general formulation.

Problem 1 Permuted Kernel Problem (PKP)
Given $\mathbf{A} \in \mathbb{F}_q^{m \times n}$ and $\mathbf{V} \in \mathbb{F}_q^{\ell \times n}$, with $m, \ell \leqslant n$, find $\pi \in S_n$ such that $\pi(\mathbf{V})\mathbf{A}^\top = \mathbf{0}$.

This formulation is sometimes called *homogeneous PKP*; in Section 5, we will also make use of the inhomogenous version (we require that $\pi(\mathbf{V})\mathbf{A}^\top$ is equal to a given, non necessarily null, $\ell \times m$ matrix). When $\ell = 1$ (which, arguably, is the most considered PKP formulation), the above problem corresponds to the one employed, for instance, in [8, 28]; we will refer to this case as *mono-dimensional PKP*. In [20], instead, the authors consider the case $\ell > 1$; we will refer to this case as *multi-dimensional PKP*.

The subcode equivalence problem is another NP-complete problem introduced in [5] which reads as follows.

Problem 2 Subcode Equivalence Problem (SEP)
Given $\mathcal{C} \subseteq \mathbb{F}_q^n$ with dimension k and $\mathcal{C}' \subseteq \mathbb{F}_q^n$ with dimension $k' < k$, find $\pi \in S_n$ such that $\pi(\mathcal{C}') \subseteq \mathcal{C}$. Equivalently, given $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ and $\mathbf{G}' \in \mathbb{F}_q^{k' \times n}$, find $\mathbf{S} \in \mathrm{GL}_{k',k}$ and $\pi \in S_n$ such that $\mathbf{G}' = \mathbf{S}\pi(\mathbf{G})$.

Considering the above formulation but requiring that $k' = k$ would correspond to the PEP.

Problem 3 Permutation Equivalence Problem (PEP)
Given $\mathcal{C}, \mathcal{C}' \subseteq \mathbb{F}_q^n$, both with dimension k, find $\pi \in S_n$ such that $\pi(\mathcal{C}') = \mathcal{C}$. Equivalently, given $\mathbf{G}, \mathbf{G}' \in \mathbb{F}_q^{k \times n}$, find $\mathbf{S} \in \mathrm{GL}_{k,k}$ and $\pi \in S_n$ such that $\mathbf{G}' = \mathbf{S}\pi(\mathbf{G})$.

We remember that SEP is NP-complete [5], while a well-known result establishes that the NP-completeness of PEP would imply collapse of the polynomial hierarchy [23]. Also, for codes with small hull (that is, the intersection between a code and its dual), efficient solvers for PEP exist [2, 27]. These algorithms essentially require some linear algebra computations (e.g., intersecting vector spaces) plus auxiliary operations, such as hull enumeration [27] or reducing to graph isomorphism [2]. The cost of these auxiliary operations grows exponentially with the hull dimension and, when the dimension is large, actually becomes the predominant term. In such a case, currently known best solvers are radically different algorithms [3, 6], which require to find low weight codewords and consequently take exponential time.

Notice that, for random codes, the hull dimension is a small constant with high probability [26]. Hence, for random codes, with overwhelming probability the algorithms in [2, 27] solve PEP in polynomial time: for this reason, PEP is considered easy on average.

4.2 Reducing SEP to PEP

We start by describing a trivial way to solve SEP, which highlights its connection with PEP. Consider the procedure in Algorithm 1 which, basically, performs a brute force search over all subcodes of $\mathcal{C}$ and, for each candidate, tries to solve the associated PEP instance. The running time of the algorithm is analyzed next.

PKP	Equivalent problem	Parameters
$\ell < n-m$	SEP	$k = n-m,\ k' = \ell$
$\ell = n-m$	PEP	$k = k' = n-m$
$\ell > n-m$	SEP	$k = \ell,\ k' = n-m$

Table 1: Relations between PKP, SEP and PEP

Proposition 3 *Algorithm 1 requires average time $O\left(\left[{k \atop k'}\right]_q T_{\mathsf{PEP}}(q,n,k)\right)$, where $T_{\mathsf{PEP}}(q,n,k)$ is the running time of an algorithm that solves the PEP on codes with length n and dimension k, defined over a finite field with q elements.*

Proof. The probability that a choice for $\mathbf{S}$ is valid is given by the reciprocal of $\left[{k \atop k'}\right]_q$. Hence, $\left[{k \atop k'}\right]_q$ corresponds to the number of times, on average, we call the PEP solver. □

Remark 2. As a simple approximation, we can use $\left[{k \atop k'}\right]_q \approx q^{k'(k-k')}$: regardless of the cost of solving the PEP, the above reduction takes exponential time.

The considered reduction is interesting from a theoretical point of view, since it creates a connection between SEP and PEP. However, its running time is not satisfying: even when the obtained PEP instance is easy (i.e., $T_{\mathsf{PEP}}(q,n,k)$ is polynomial), the reduction itself takes exponential time since $q^{k'(k-k')}$ grows with $k = Rn$. In the next section, we will see that SEP and PKP are exactly the same problem: this allows solving SEP through any known algorithm for solving PKP.

4.3 Equivalence between PKP and SEP

In the following proposition we show that PKP and SEP are equivalent.

Proposition 4 *If $\ell \neq n-m$, PKP is equivalent to SEP; if $\ell = n-m$, PKP is equivalent to PEP.*

Proof. In the following, we will denote by $\{\mathbf{A} \in \mathbb{F}_q^{m\times n}, \mathbf{V} \in \mathbb{F}_q^{\ell\times n}\}$ an instance of PKP, and by $\{\mathbf{G} \in \mathbb{F}_q^{k\times n}, \mathbf{G}' \in \mathbb{F}_q^{k'\times n}\}$ an instance of SEP. Also, we denote by $\mathcal{C}$ and $\mathcal{C}'$ the codes generated, respectively, by $\mathbf{G}$ and $\mathbf{G}'$. To avoid burdening the notation and the treatment, we carry out the proof considering that matrices have full rank; in case this is not true, the proof still holds, with the only precaution that one first needs to remove redundant rows (and update parameters accordingly). We start by considering that any SEP instance can be transformed into a PKP instance with some simple linear algebra. Indeed, to solve the SEP, we want to find a permutation π and a non-singular $\mathbf{S} \in \mathrm{GL}_{k',k}$ such that $\mathbf{G}' = \mathbf{S}\pi(\mathbf{G})$. Equivalently, it must be $\pi^{-1}(\mathbf{G}') = \mathbf{S}\mathbf{G}$. Let $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ be a parity-check matrix for $\mathcal{C}$: then, π solves SEP if and only if

$$\pi^{-1}(\mathbf{G}')\mathbf{H}^\top = \mathbf{0}.$$

The above corresponds exactly to the requirement for a PKP instance with $m = n-k$ and $\ell = k'$.

The same procedure works in the other way, i.e., in showing how PKP can be seen either as SEP or PEP. We start by considering the case of $\ell < n-m$, and interpret $\mathbf{A}$ as a parity-check matrix for $\mathcal{C}$ and $\mathbf{V}$ as a generator for $\mathcal{C}'$. Repeating the above reasoning, it is immediately seen that the problem corresponds to a SEP instance. If instead $\ell = n-m$, we have that $\mathcal{C}$ and $\mathcal{C}'$ have the same dimension $k = k' = \ell = n-m$, so we end up with a PEP instance. Finally, we notice that if $\ell > n-m$, the code $\mathcal{C}'$ has a dimension that is larger than that of $\mathcal{C}$. So, PKP still corresponds to a SEP instance, but the role of the codes are exchanged; this means that $\mathcal{C}$ is the permutation of a subcode of $\mathcal{C}'$. □

A summary of the relations between PKP and SEP in shown in Table 1. As we have already said, the PEP has been extensively studied and is considered easy when the input codes are random. Consequently, the PKP can be considered easy, on average, if $\ell = n-m$; for such reason, in the present paper we will not study this case. Also, from now on we set $1 \leqslant \ell < n-m$: indeed, the case of $\ell > n-m$ can be tackled identically, apart from a mere swap in the roles of the parameters ℓ and $n-m$.

5 General considerations on PKP and asymptotics for the KMP algorithm

In this section we recall general properties about hard PKP instances. Also, we recall the algorithm in [19], which we refer to as KMP (the acronym is formed by the authors' initials). We generalize all our considerations, as well as the KMP algorithm, to the multi-dimensional version of the PKP. Finally, we derive its running time in the asymptotic regime.

5.1 Hardest instances

Following the literature regarding the PKP, we study the constrains under which hardest to solve instances are obtained; in doing this, we generalize the considerations holding for the mono-dimensional PKP to the multi-dimensional case. Namely, we assume $\mathbf{A} \in \mathbb{F}_q^{m\times n}$ and $\mathbf{V} \in \mathbb{F}_q^{\ell\times n}$ both with full rank and no repeated columns, and also set the parameters so that, on average, only one solution exists. Formally, these conditions correspond to the following constrains:

i) $\text{Rank}(\mathbf{A}) = m$, $\text{Rank}(\mathbf{V}) = \ell$;
ii) $q^\ell \geqslant n$, $q^m \geqslant n$;
iii) $\mathbf{A}$ and $\mathbf{V}$ with no repeated columns;
iv) $n!\frac{\left[{n-m \atop \ell}\right]_q}{\left[{n \atop \ell}\right]_q} < 1$.

Notice that condition ii) is simply obtained considering that, if $q^\ell < n$, then for sure $\mathbf{V}$ contains at least two identical columns; the same holds for the condition $q^m < n$. Finally, condition iv) is about the uniqueness of the solution. Consider that any permutation π of $\mathbf{V}$ yields a vector space with dimension ℓ. The number of ℓ-dimensional spaces that are orthogonal to the space generated by $\mathbf{A}$ is counted by $\left[{n-m \atop \ell}\right]_q$. Then, $\frac{\left[{n-m \atop \ell}\right]_q}{\left[{n \atop \ell}\right]_q}$ is the probability that a random ℓ-dimensional subspace of $\mathbb{F}_q^n$ is orthogonal to $\mathbf{A}$. Since $\mathbf{V}$ has all distinct columns, we have that the number of distinct $\pi(\mathbf{V})$ is $|\mathscr{S}_n(\mathbf{V})| = n!$. Assuming that each $\pi(\mathbf{V})$ behaves as a random subspace of $\mathbb{F}_q^n$, then we can use $n!\frac{\left[{n-m \atop \ell}\right]_q}{\left[{n \atop \ell}\right]_q} < 1$ as an estimate of the average number of solutions: setting this number to be smaller than 1, we guarantee that, on average, we expect to have no more than one solution.

Remark 3. It holds that $\frac{\left[{n-m \atop \ell}\right]_q}{\left[{n \atop \ell}\right]_q} \approx q^{-\ell m}$, so that condition iv) becomes $n!q^{-\ell m} < 1$.

Another common criterion consists in studying instances which are not vacuously hard, i.e., such that at least one solution exists. To do this, we first choose $\widetilde{\mathbf{V}}$ as a random kernel element with full rank and no repeated columns; then, we obtain $\mathbf{V}$ by permuting $\widetilde{\mathbf{V}}$ according to a randomly selected permutation.

Adding one extra equation and inhomogenous PKP We consider that $\mathbf{A}$ can be enriched with an additional linear equation. Indeed, let $\mathbf{H} = \begin{pmatrix} \mathbf{A} \\ 1, 1, \cdots, 1 \end{pmatrix} \in \mathbb{F}_q^{r\times n}$, with $r = m + 1$. Let $\widetilde{\mathbf{V}} = \pi(\mathbf{V})$, where π is the solution for PKP. We consider that

$$\begin{aligned} \mathbf{E} = \widetilde{\mathbf{V}}\mathbf{H}^\top &= \left(\widetilde{\mathbf{V}}\mathbf{A}^\top, \widetilde{\mathbf{V}} \cdot (1, 1, \cdots, 1)^\top\right) \\ &= \left(\mathbf{0}_{\ell\times m}, \sum_{i=1}^n \widetilde{\mathbf{v}}_i\right) \\ &= \left(\mathbf{0}_{\ell\times m}, \sum_{i=1}^n \mathbf{v}_i\right) = (\mathbf{0}_{\ell\times m}, \mathbf{e}) \in \mathbb{F}_q^{\ell\times r}, \end{aligned} \tag{6}$$

where $\widetilde{\mathbf{v}}_i$ is the i-th column of $\widetilde{\mathbf{V}}$ and $\mathbf{v}_i$ that of $\mathbf{V}$. Notice that $\mathbf{e}$ is a length-ℓ column vector. The solution is, now, a permutation $\pi \in S_n$ such that $\pi(\mathbf{V})\mathbf{H}^\top = \mathbf{E}$ where, in general, $\mathbf{E} \neq \mathbf{0}$.

This formulation for the problem is sometimes called *Inhomogenous PKP* since, differently from what stated in Problem 1, the linear equations that define the problem may be not homogenous. In practice, transforming to the inhomogenous version allows to gain one extra equation which, in turns, reduces the time complexity of solvers. From now on, we always consider the inhomogenous formulation.

Guessing the action of π only on $n-r$ coordinates We now show that, to solve PKP, it is enough to guess how π acts on a subset of $n-r$ coordinates. To this end, let

$$\mathbf{S}\mathbf{H} = \mathsf{RREF}(\mathbf{H}, \{n-r+1, n-r+2, \cdots, n\}) = (\mathbf{U}, \mathbf{I}_r), \tag{7}$$

with $\mathbf{S} \in \mathrm{GL}_{r,r}$ and $\mathbf{U} \in \mathbb{F}_q^{r\times(n-r)}$. Let $\widetilde{\mathbf{E}} = \mathbf{E}\mathbf{S}^\top$, write $\widetilde{\mathbf{V}} = \left(\widetilde{\mathbf{V}}_1, \widetilde{\mathbf{V}}_2\right)$ and consider that

$$\widetilde{\mathbf{E}} = \widetilde{\mathbf{V}}_1\mathbf{U}^\top + \widetilde{\mathbf{V}}_2,$$

from which

$$\widetilde{\mathbf{V}}_2 = \widetilde{\mathbf{E}} - \widetilde{\mathbf{V}}_1\mathbf{U}^\top. \tag{8}$$

Notice that, if $\mathbf{H}$ does not admit a change of basis as in (7), it is enough to row reduce with respect to a different set of r indices. In other words, let $J \subset \{1, 2, \cdots, n\}$ of size $n-r$ and $\neg J = \{1, 2, \cdots, n\}\backslash J$. Also, let J be such that $\mathbf{H}_{\neg J}$ is non singular. Then, it holds that

$$\mathbf{V}_{\neg J} = \left(\mathbf{E} - \mathbf{V}_J\mathbf{H}_J\right)\mathbf{H}_{\neg J}^{-\top}, \tag{9}$$

where the $^{-\top}$ operator denotes the inverse transposal. In other words, we are always able to reconstruct the values of $\widetilde{\mathbf{V}}$ outside of the set J, since they depend only on $\widetilde{\mathbf{V}}_J$. Hence, we simply need to correctly guess the action of π for $n-r$ coordinates, that is, the ones indexed by J.

A coding theory perspective We now explicitly reformulate the PKP as a code-based problem. This has partially already been shown since we described that PKP and SEP are exactly the same problem. Yet, rephrasing everything in terms of codes also for the inhomogenous PKP will be helpful in analyzing the attacks we discuss in this paper.

Problem 4 ***Inhomogenous PKP as a Decoding Problem***
Given $\mathbf{H} \in \mathbb{F}_q^{r\times n}$, the parity-check matrix of $\mathcal{C} \subseteq \mathbb{F}_q^n$ with code rate $R = 1 - \frac{r}{n}$, $\mathbf{V} \in \mathbb{F}_q^{\ell\times n}$ and $\mathbf{E} \in \mathbb{F}_q^{\ell\times r}$, find $\pi \in S_n$ such that $\mathbf{E} = \pi(\mathbf{V})\mathbf{H}^\top$.

This formulation highlights the fact that the inhomogenous PKP is, in the end, a decoding problem. Namely, we are given a set of ℓ syndromes $\mathbf{e}_i$, where $\mathbf{e}_i$ is the i-th row of $\mathbf{E}$, and a set of ℓ vectors $\mathbf{v}_i$, where $\mathbf{v}_i$ is the i-th row of $\mathbf{V}$. We want to find a permutation π so that

$$\mathbf{e}_i = \pi(\mathbf{v}_i)\mathbf{H}^\top, \ \ \forall i \in \{1, \cdots, \ell\}.$$

To decode a syndrome into a vector, it is enough to guess the values of the vector in an information set, that is, a set $J \subset \{1, 2, \cdots, n\}$ of size k and such that

$$\{\mathbf{c}_J \neq \mathbf{c}'_J \mid \mathbf{c}, \mathbf{c}' \in \mathcal{C}, \ \mathbf{c} \neq \mathbf{c}'\}.$$

It is easy to see that, if J is an information set for $\mathcal{C}$, then $\mathbf{H}_{\neg J}$ is non singular, hence, it can be used in (9). In other words, to solve the problem, it is enough to determine the values of $\mathbf{V}$ in the information set J: the values outside of the information set can be computed using (9).

5.2 KMP algorithm

We here recall the KMP algorithm [19], originally proposed for the mono-dimensional PKP, and generalize it to the multi-dimensional case. The algorithm works by producing a list $\mathcal{L}$ with candidates for the values that the solution $\widetilde{\mathbf{V}} = \pi(\mathbf{V})$ assumes in an information set. Each candidate is then checked using (9). To produce $\mathcal{L}$, the algorithm relies on a collisions search approach by

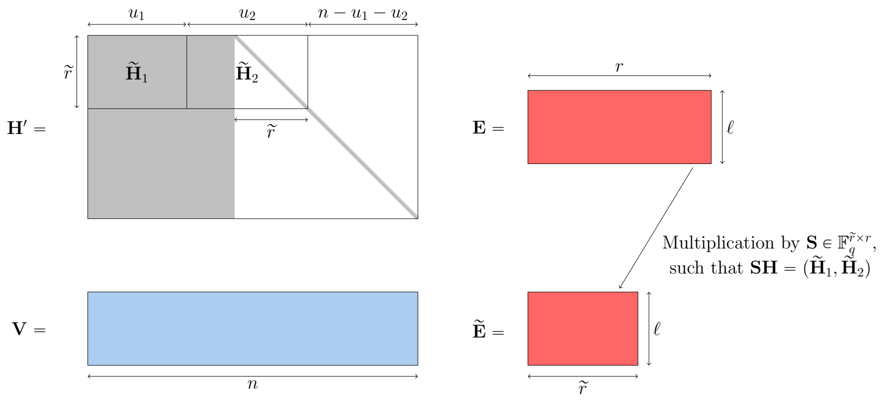

u_1
u_2
$n-u_1-u_2$
$\widetilde{r}$
$\widetilde{\mathbf{H}}_1$
$\widetilde{\mathbf{H}}_2$
$\mathbf{H}' =$
$\widetilde{r}$
r
$\mathbf{E} =$
ℓ
Multiplication by $\mathbf{S} \in \mathbb{F}_q^{\widetilde{r}\times r}$, such that $\mathbf{SH} = (\widetilde{\mathbf{H}}_1, \widetilde{\mathbf{H}}_2)$
$\mathbf{V} =$
n
$\widetilde{\mathbf{E}} =$
ℓ
$\widetilde{r}$

Fig. 1: Representation of the matrices employed in the KMP algorithm

creating and merging two lists $\mathcal{L}_1$ and $\mathcal{L}_2$. This approach works only if $\mathcal{L}_1$ and $\mathcal{L}_2$ can be constructed with relatively small effort, say, smaller than that of enumerating all the possible values in an information set. This becomes possible only if *sparse* equations are employed, that is, equations that involve a small number of variables. To achieve this, the KMP algorithm brings $\mathbf{H}$ into row reduced echelon form and then selects a subset of parity-check equations: by construction, there will be variables that never participate in the chosen equations.

Technically, the algorithm works with two parameters $u_1, u_2 \in \mathbb{N}$ such that $n-r+1 \leqslant u_1+u_2 \leqslant n$. Let $\widetilde{r} = u_1+u_2-(n-r)$ be the number of considered sparse equations. The procedure is initialized by first computing $\mathbf{H}' = \mathsf{RREF}(\mathbf{H}, \{n-r+1, \cdots, n\})$ and then setting $\widetilde{\mathbf{H}}$ as the sub-matrix formed by the entries of $\mathbf{H}'$ in the first $\widetilde{r}$ rows and the first u_1+u_2 columns. Let $\mathbf{S} \in \mathbb{F}_q^{\widetilde{r}\times r}$ with rank $\widetilde{r}$ and such that $\widetilde{\mathbf{H}}$ corresponds to the submatrix formed by the first u_1+u_2 columns of $\mathbf{SH}$[4]. This transformation is applied to $\mathbf{E}$, obtaining $\widetilde{\mathbf{E}} = \mathbf{ES}^\top \in \mathbb{F}_q^{\ell\times\widetilde{r}}$. Then, we partition $\widetilde{\mathbf{H}}$ as $(\widetilde{\mathbf{H}}_1, \widetilde{\mathbf{H}}_2)$, where $\widetilde{\mathbf{H}}_1 \in \mathbb{F}_q^{\widetilde{r}\times u_1}$ and $\widetilde{\mathbf{H}}_2 \in \mathbb{F}_q^{\widetilde{r}\times u_2}$. To visualize these matrices, see Figure 1.

After this preparatory step, the algorithm proceeds by constructing the two lists

$$\mathcal{L}_1 = \left\{ (\mathbf{X}, \mathbf{X}\widetilde{\mathbf{H}}_1^\top) \middle| \mathbf{X} \in \mathscr{S}_{u_1}(\mathbf{V}) \right\},$$

$$\mathcal{L}_2 = \left\{ (\mathbf{X}, \widetilde{\mathbf{E}} - \mathbf{X}\widetilde{\mathbf{H}}_2^\top) \middle| \mathbf{X} \in \mathscr{S}_{u_2}(\mathbf{V}) \right\}.$$

Let $\mathcal{L} = \mathcal{L}_1 \bowtie \mathcal{L}_2$, where $\bowtie$ is computed as follows:

1. use an efficient search algorithm (e.g., permutation plus binary search) to find collisions, i.e., pairs $(\mathbf{X}, \mathbf{Y}) \in \mathcal{L}_1$ and $(\mathbf{X}', \mathbf{Y}') \in \mathcal{L}_2$ such that $\mathbf{Y} = \mathbf{Y}'$;
2. keep only the collisions for which $\mathbf{X}$ and $\mathbf{X}'$ have no common columns.

By construction, it holds that

$$\mathcal{L} = \left\{ (\mathbf{X}, \mathbf{X}') \in \mathscr{S}_{u_1+u_2}(\mathbf{V}) \mid (\mathbf{X}, \mathbf{X}')\widetilde{\mathbf{H}}^\top = \widetilde{\mathbf{E}} \right\}.$$

Then, we find $J \subseteq \{1, \cdots, n-r+\widetilde{r}\}$ with size $n-r$ such that $\mathbf{H}_{\{1,\cdots,n\}\backslash J}$ is non-singular, compute $\widetilde{\mathbf{H}} = \mathsf{RREF}(\mathbf{H}, \{1, \cdots, n\}\backslash J)$ and use (8) to test each element in $\mathcal{L}$. Namely, for each $\mathbf{X} \in \mathcal{L}$, we use the entries of $\mathbf{X}_J$ as $\widetilde{\mathbf{V}}_J$ and see if the resulting $\widetilde{\mathbf{V}}$ belongs to $\mathscr{S}_n(\mathbf{V})$.

We observe that u_1+u_2 should be not lower than $n-r$, otherwise we would not be guessing enough values for an information set. Actually, we require $u_1+u_2 \geqslant n-r+1$. Indeed, collisions between $\mathcal{L}_1$ and $\mathcal{L}_2$ are searched using matrices of size $\ell\times\widetilde{r}$, so it must be $\widetilde{r} \geqslant 1$ (colliding matrices must have at least one column): since $\widetilde{r} = u_1+u_2-(n-r)$, we get $u_1+u_2 \geqslant n-r+1$.

Coherent with the literature about the PKP, we measure the running time of the KMP algorithm as the number of matrix-matrix multiplications and list operations.

[4] The matrix $\mathbf{S}$ corresponds to the submatrix formed by the first $\widetilde{r}$ rows of the full rank, $r\times r$ matrix that brings $\mathbf{H}$ into row reduced echelon form.

Proposition 5 ***Running time of the KMP algorithm***
Let $u_1, u_2 \in \mathbb{N}$ such that $n - r + 1 \leqslant u_1 + u_2 \leqslant n$. Then, the KMP algorithm runs in time

$$T_{\mathsf{KMP}}(u_1, u_2) = |\mathcal{L}_1| + |\mathcal{L}_2| + N_{\mathcal{L}_1 \bowtie \mathcal{L}_2} + |\mathcal{L}|,$$

where $|\mathcal{L}_i| = \frac{n!}{(n-u_i)!}$, $N_{\mathcal{L}_1 \bowtie \mathcal{L}_2} \doteq \frac{(n!)^2 q^{\ell(n-r-u_1-u_2)}}{(n-u_1)!(n-u_2)!}$ and $|\mathcal{L}| \doteq \frac{n!}{(n-u_1-u_2)!} q^{\ell(n-r-u_1-u_2)}$.

Proof. Each list $\mathcal{L}_i$ has size $|\mathscr{S}_{u_i}(\mathbf{V})|$: given that $\mathbf{V}$ does not have repeated columns, we have $|\mathcal{L}_i| = \frac{n!}{(n-u_i)!}$. We now estimate the average number of collisions $N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}$. We consider that each collision is due to two equal matrices with sizes $\ell \times \tilde{r}$. Since we are dealing with random PKP instances, we can consider that any pair of elements $(\mathbf{X}, \mathbf{Y}) \in \mathcal{L}_1$ and $(\mathbf{X}', \mathbf{Y}') \in \mathcal{L}_2$ collides with probability

$$q^{-\ell \tilde{r}} = q^{-\ell(u_1+u_2+r-n)} = q^{\ell(n-r-u_1-u_2)}.$$

Then, the average value of $N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}$ can be set as

$$|\mathcal{L}_1| \cdot |\mathcal{L}_2| \cdot q^{\ell(n-r-u_1-u_2)}.$$

Finally, we consider that $\mathcal{L}$ contains all the matrices in $\mathscr{S}_{u_1+u_2}(\mathbf{V})$ such that their product by $\widetilde{\mathbf{H}}$ returns $\widetilde{\mathbf{E}}$, having size $\ell \times \tilde{r}$. Hence, the average size of $\mathcal{L}$ can be estimated as

$$|\mathscr{S}_{u_1+u_2}(\mathbf{V})| \cdot q^{\ell(n-r-u_1-u_2)} = \frac{n! q^{\ell(n-r-u_1-u_2)}}{(n-u_1-u_2)!}.$$

□

Observe that we always have $|\mathcal{L}| \leqslant N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}$, so that in the end the time complexity of the KMP algorithm is dominated by $\max\{|\mathcal{L}_1| \,;\, |\mathcal{L}_2| \,;\, N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}\}$. In practice, the algorithm is optimized when these three terms are balanced, that is,

$$|\mathcal{L}_1| \approx |\mathcal{L}_2| \approx N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}.$$

In such a case, the time complexity is $\approx |\mathcal{L}_1| \approx |\mathcal{L}_2|$. Notice that the same would hold if, more generally, we require $|\mathcal{L}_1|, |\mathcal{L}_2| \geqslant N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}$. In any case, having $|\mathcal{L}_1| \approx |\mathcal{L}_2|$ implies $u_1 \approx u_2$ (that is, either $u_1 = u_2$ or $u_1 = u_2 \pm 1$). For the sake of simplicity, in the following we consider $u_1 = u_2 = u$, which leads to[5]

$$|\mathcal{L}_1| = |\mathcal{L}_2| = L = \frac{n!}{(n-u)!}.$$

$$N_{\mathcal{L}_1 \bowtie \mathcal{L}_2} = L^2 q^{\ell(n-r-2u)}.$$

Since we want $N_{\mathcal{L}_1 \bowtie \mathcal{L}_2} \leqslant L$, this results in $L q^{\ell(n-r-2u)} \leqslant 1$, which can be rewritten as

$$\frac{n!}{(n-u)!} \leqslant q^{\ell(2u+r-n)}. \tag{10}$$

The optimal value for u is the smallest one for which the above inequality is satisfied.

5.3 Asymptotic analysis of KMP

We now derive the asymptotic running time of the KMP algorithm. Let us first consider the following proposition.

Proposition 6 ***Asymptotic running time for KMP***
Let $\mu \in \mathbb{R}_+$ such that $\frac{R}{2} < \mu \leqslant \frac{1}{2}$ and

$$h_2(\mu) + \mu \log_2\left(\frac{\mu n}{e}\right) + \ell \log_2(q)\big(R - 2\mu\big) = 0. \tag{11}$$

Then, asymptotically, the running time of the KMP algorithm is minimized with $u_1 = u_2 = \mu n$ and is given by $2^{n \cdot c_{\mathsf{KMP}}(\mu)\left(1+o(1)\right)}$, where

$$c_{\mathsf{KMP}}(\mu) = h_2(\mu) + \mu \log_2\left(\frac{\mu n}{e}\right).$$

[5] There are cases in which requiring $u_1 = u_2$ results in slightly worse running times. For this reason, we presented the algorithm, as well as the time complexity analysis, with the more general description in which u_1 is not necessarily equal to u_2.

Proof. See Appendix C. □

Notice that the KMP algorithm runs in time which is super-exponential in the code length n, since the dominant term grows exponentially with $n\log_2(n)$. However, to give a rigorous proof of this fact, we need some further considerations.

First, we express q as a function of the code length. We choose q as the smallest integer such that the condition of having no more than one solution holds, that is, as the smallest integer such that $n!\frac{\begin{bmatrix} n-r \\ \ell \end{bmatrix}_q}{\begin{bmatrix} n \\ \ell \end{bmatrix}_q} \leqslant 1$. As we will see in the following, this corresponds to considering the hardest PKP instances with density 1 (i.e., having on average at most one solution). Then, taking (1) and (5) into account, we find

$$\begin{aligned} q &\sim \left(\frac{n}{e}\right)^{\frac{1}{\ell(1-R)}} \left(1 + \frac{1}{2n\ell(1-R)}\log_2(\pi n)\right) \\ &= \left(\frac{n}{e}\right)^{\frac{1}{\ell(1-R)}} \big(1 + o(1)\big). \end{aligned} \tag{12}$$

In this regime, we can find an easy, closed-form formula for the running time of KMP. To this end, let us consider the following proposition.

Proposition 7 ***Running time of KMP in the asymptotic regime: closed form formula*** *Let $q \sim \left(\frac{n}{e}\right)^{\frac{1}{\ell(1-R)}}$. Let $\mu \in (R/2; 1/2]$ be the value that optimizes the KMP algorithm, as in Proposition 6. Then, $\lim_{n\to\infty}\mu = \mu^* = \frac{R}{1+R}$ and the KMP asymptotically runs in time $2^{n\cdot c^*_{\mathsf{KMP}}}$, where*

$$\begin{aligned} c^*_{\mathsf{KMP}} &= \lim_{n\to\infty} c_{\mathsf{KMP}}(\mu) = c_{\mathsf{KMP}}\left(\frac{R}{1+R}\right) \\ &= \frac{1}{1+R}\left(\log_2(1+R) + R\log_2\left(\frac{n}{e}\right)\right). \end{aligned}$$

Proof. See Appendix C. □

The above propositions lead to the following three interesting observations, which hold in the asymptotic regime:

- the optimal value of μ is independent of ℓ;
- the optimal time complexity is independent of ℓ;
- the KMP algorithm runs in time which is super-exponential in the code length.

Remark 4. The fact that the running time of the KMP algorithm does not depend on ℓ can be quickly justified by considering how the relevant terms in the time complexity behave, when ℓ varies. First, the attack is optimized when $u_1 = u_2 = u$; notice that $|\mathcal{L}_1| = |\mathcal{L}_2| = L = \frac{n!}{(n-u)!}$, hence, there is no dependency on ℓ. Now, the average number of collisions is $N_{\mathcal{L}_1 \bowtie \mathcal{L}_2} = L^2 \cdot q^{\ell(n-r-2u)}$. Since we are considering $q \approx \left(\frac{n}{e}\right)^{\frac{1}{\ell(1-R)}}$, we get

$$N_{\mathcal{L}_1 \bowtie \mathcal{L}_2} \approx L^2 \left(\frac{n}{e}\right)^{\frac{n-r-2u}{1-R}}. \tag{13}$$

Finally, we choose u such that $N_{\mathcal{L}_1 \bowtie \mathcal{L}_2} \approx L$: both sides of the relation have no dependency on ℓ.

Let us now provide some more insight in the behavior of KMP, with the help of some numerical experiments. Let us consider several triplets (ℓ, R, n) and, for each one, look for the smallest prime q for which the average number of solutions to PKP is not greater than 1. Using these parameters, we study the performance of KMP. In Figure 2 we report an example of how the optimal value of μ behaves, for the case of $\ell = 1$; as we can see from the figure, the optimal value for μ is already well approximated by μ^* for $n \leqslant 1,000$. Figure 3 reports the optimal values of the complexity coefficient c_{KMP}, for growing n, and a comparison with c^*_{KMP}. We notice that there is a small, but non negligible, difference between the values of c_{KMP} and c^*_{KMP}. This is due to the fact that c_{KMP} tends to c^*_{KMP} very slowly. In any case, as predicted by Proposition 7, the values of c_{KMP} exhibit

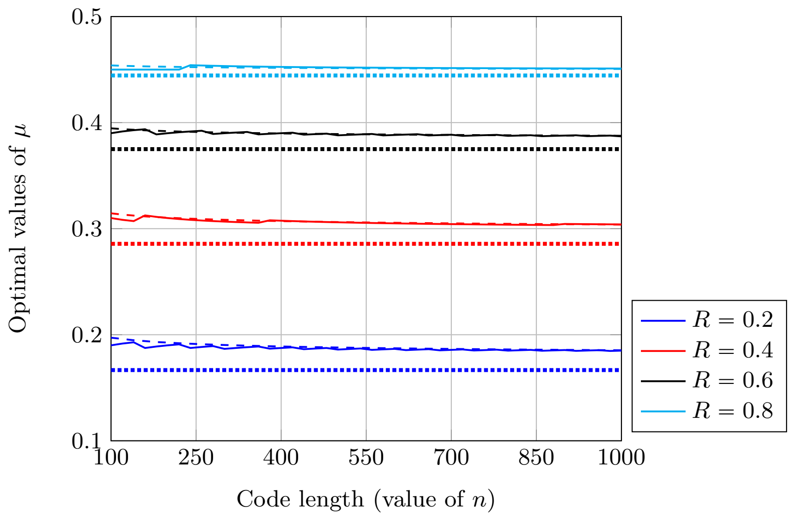


Fig. 2: Values of μ that optimize the KMP algorithm, for $\ell = 1$ and several code rates (indicated by R). The continuous line has been obtained by selecting, among all possible choices for (u_1, u_2), the ones yielding the smallest value for $T_{\mathsf{KMP}}(u_1, u_2)$. Dashed lines report the optimal values of μ which we have found by numerically solving (11). Dotted lines correspond to $\mu^* = \frac{R}{1+R}$.

a logarithmic dependence in n. In Figure 4 we show how the complexity exponent behaves, for the case of $R = 0.5$, several values of ℓ and much greater values of n (informally, we are closer to the asymptotic regime). We see that the resulting complexity exponents have essentially the same values, apart from some fluctuations which are due to the fact that we require q to be a prime integer, hence, we may need to change the value resulting from (12). Yet, as we expected, the value of ℓ does not affect significantly the complexity exponent.

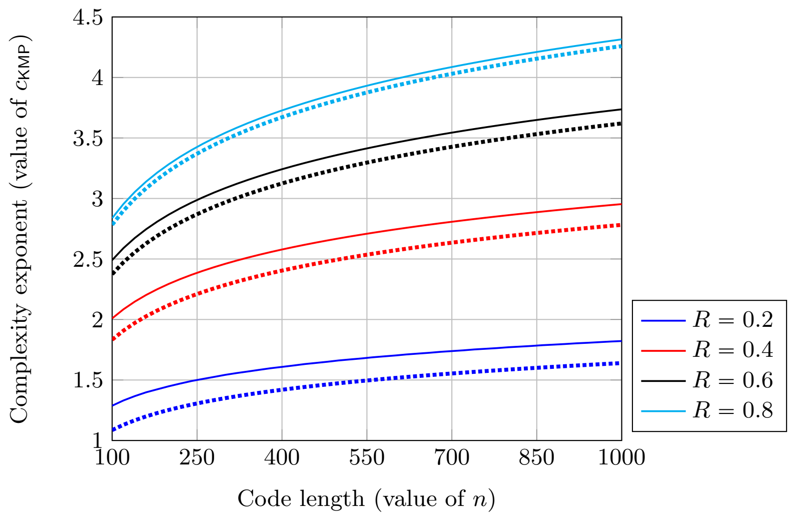


Fig. 3: Complexity exponents for the KMP algorithm, as a function of n, for the case of $\ell = 1$ and several code rates (indicated by R). Continuous lines report the values of c_{KMP} arising from Proposition 6, while dotted lines report the values of c^*_{KMP}.

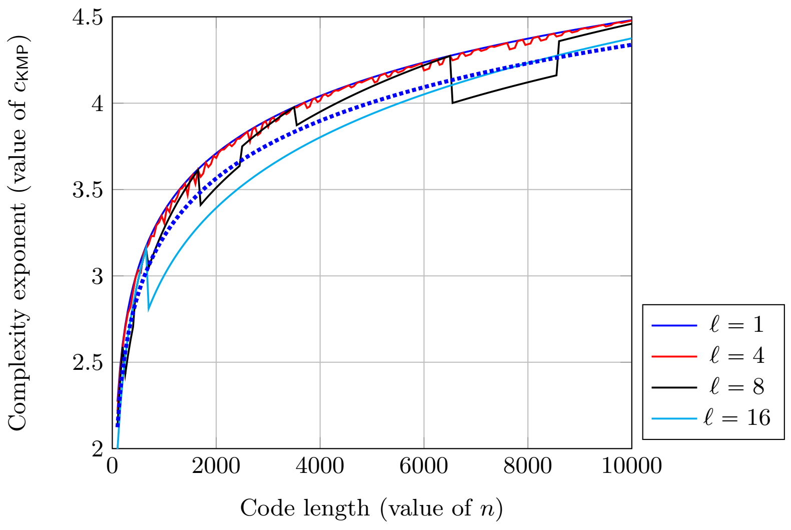


Fig. 4: Complexity exponents for the KMP algorithm, as a function of n, for the case of $\ell = 1$. Continuous lines report the values of c_{KMP} arising from Proposition 6, while dotted lines report the values of c^*_{KMP}.

5.4 KMP algorithm with constant q

We stress that the running time of the KMP algorithm does not depend on ℓ only when q is chosen according to (12), that is, as the smallest integer such that the average number of solutions is not greater than 1. If we choose larger values for q, then the running time gets lower. Consequently, PKP instances in which q is chosen adaptively as a function of n, R and ℓ (according to (12)) are the hardest to solve.

We give a brief explanation about this fact, by considering the situation where we fix q, n and R, and let ℓ grow. We set q as the value that guarantees uniqueness of the PKP solution for the mono-dimensional case, that is, $q \approx \left(\frac{n}{e}\right)^{\frac{1}{1-R}}$. Let us consider, for simplicity, but without loss of generality, $u_1 = u_2 = u = \mu n$, so that $L = |\mathcal{L}_1| = |\mathcal{L}_2|$. For any value of ℓ, we have

$$\begin{aligned}
N_{\mathcal{L}_1 \bowtie \mathcal{L}_2} &= L^2 q^{\ell(n-r-2u)} \\
&= L^2 \left(\frac{n}{e}\right)^{\frac{\ell(n-r-2u)}{1-R}} \\
&= L^2 \left(\frac{n}{e}\right)^{\frac{(n-r-2u)}{1-R} + \frac{(\ell-1)(n-r-2u)}{1-R}} \\
&= \underbrace{L^2 \left(\frac{n}{e}\right)^{\frac{n-r-2u}{1-R}}}_{\substack{\text{Number of collisions} \\ \text{with adapted } q}} \cdot \underbrace{\left(\frac{n}{e}\right)^{\frac{(\ell-1)(n-r-2u)}{1-R}}}_{\gamma(\ell) \leqslant 1} \\
&= N'_{\mathcal{L}_1 \bowtie \mathcal{L}_2} \cdot \gamma(\ell),
\end{aligned}$$

where $N'_{\mathcal{L}_1 \bowtie \mathcal{L}_2}$ is the average number of collisions one would have considering an adapted q. Now, because of $\gamma(\ell)$ (which is always not greater than 1 and decays exponentially with ℓ), we can choose values of u that are smaller than the ones we would have, when q is chosen adaptively through (12). This reduces the overall complexity of the attack, since we can use initial lists with smaller size.

To have a confirmation on this, we have considered how the time complexity of the KMP algorithm behaves, for the case of $n = 100$ and several code rates, when ℓ increases and q stays constant. We have chosen q as the smallest prime that guarantees uniqueness of the PKP solution for $\ell = 1$ (i.e., for the mono-dimensional case). In Figure 5 we report both the complexity exponent as well as the optimal choice for $\mu = \frac{u}{n}$. As ℓ increases, the complexity of the algorithm initially decreases and ultimately converges to a minimum value; the optimal value for μ exhibits the same

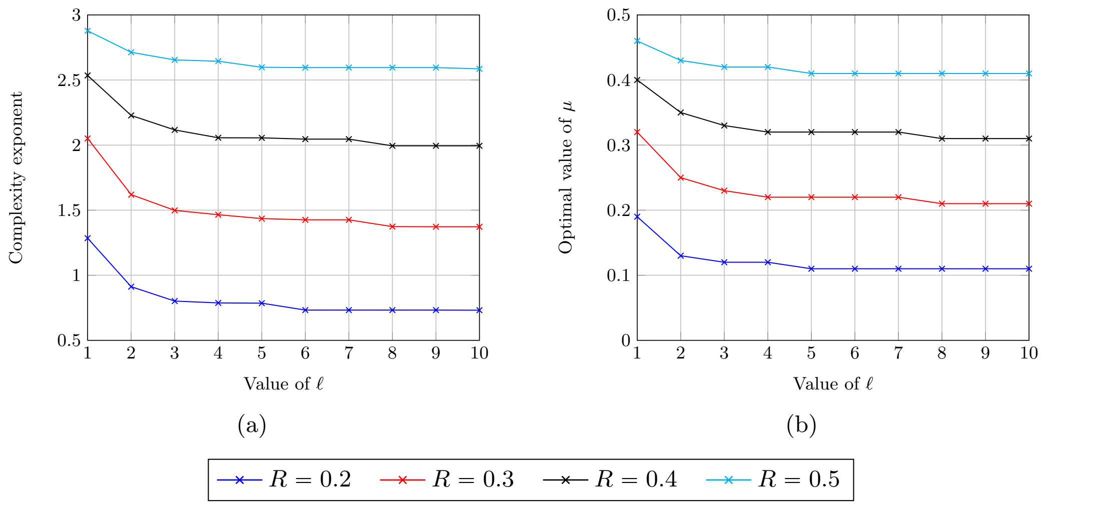


Fig. 5: Performances of the KMP algorithm, as a function of ℓ, for $n = 100$ and several code rates. q has been fixed as $\approx \left(\frac{n}{e}\right)^{\frac{1}{1-R}}$; (a) reports the complexity exponent, while (b) reports the optimal value of $\mu = \frac{u}{n}$.

behaviour. In particular, we see that μ tends to $\frac{R}{2}$: this means that we are choosing u as small as possible (which implies either $2u = k + 1$ or $2u = k + 2$). Indeed, when ℓ is large enough, the average number of collisions is guaranteed to be much smaller than L (the size of the initial lists), regardless of u. The running time of the algorithm is dominated by L, which is minimized by choosing u as small as possible. For growing ℓ, the time complexity converges from above to $L = \frac{n!}{(n-u)!} = \frac{n!}{(n-\frac{nR}{2})!}$ which, asymptotically, yields the complexity exponent

$$h_2\left(\frac{R}{2}\right) + \frac{R}{2}\log_2\left(\frac{Rn}{2e}\right).$$

6 Improving KMP in the finite regime

In [25] we have introduced a novel algorithm for the mono-dimensional case of PKP. As we have already said, it comes as a refinement of KMP and it has been shown that, in several cases and in the finite regime, it exhibits a better running time than KMP. In this section we further improve the analysis of such an algorithm: we first extend it to the multi-dimensional case, provide a more rigorous analysis and finally compare it with other PKP solvers.

6.1 New solver

We first describe, at a high level, the main idea behind [25]. In the following, we will denote by $\widetilde{\mathbf{V}} = \pi(\mathbf{V})$ the solution to PKP. Remember that the time complexity of the KMP algorithm grows with the size of the initial lists $\mathcal{L}_1$ and $\mathcal{L}_2$. These lists are built considering all possible values of $\widetilde{\mathbf{V}}_{\{1,\cdots,u_1\}}$ and $\widetilde{\mathbf{V}}_{\{u_1+1,\cdots,u_2\}}$, that is, by enumeration of $\mathscr{S}_{u_1}(\mathbf{V})$ and $\mathscr{S}_{u_2}(\mathbf{V})$. To get linear equations with few variables, the KMP algorithm brings $\mathbf{H}$ into systematic form. This creates $\widetilde{r} = u_1 + u_2 - (n - r)$ equations involving at most $u_1 + u_2$ variables, that is, the columns of $\widetilde{\mathbf{V}}$ in positions $\{1, \cdots, u_1 + u_2\}$.

The algorithm in [25] starts by finding a set of *abnormally sparse* linear equations, say, sparser than the ones we would obtain with the systematic form. Namely, the goal is to find d independent equations involving w variables and have $w < d+n-r$. This corresponds to finding a rank-d matrix $\mathbf{S} \in \mathrm{GL}_{d,r}$ such that $\mathbf{H}^* = \mathbf{SH} \in \mathbb{F}_q^{d\times n}$ has support $J \subset \{1, \cdots, n\}$ with size w. The equations given by $\mathbf{H}^*$ can be solved with a meet-in-the-middle approach, using two lists $\mathcal{K}_1$ and $\mathcal{K}_2$ obtained by

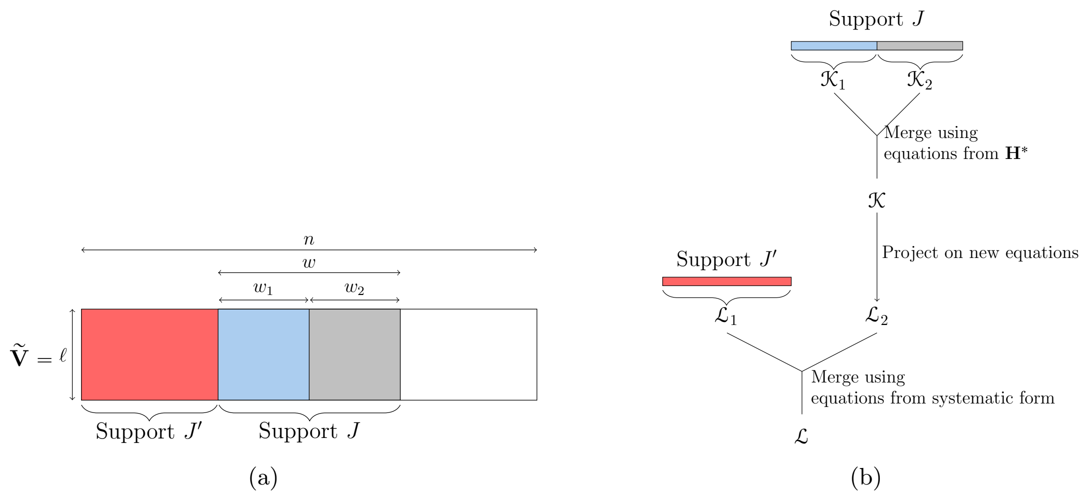


Fig. 6: Graphical representation of the algorithm in [25]. Figure (a) describes how the coordinates of the solution $\widetilde{\mathbf{V}}$ are partitioned; figure (b) shows lists operations. The use of colors in both figures is coherent, for instance: list $\mathcal{K}_2$ in figure (b) contains candidates for the submatrix of $\widetilde{\mathbf{V}}$ colored in grey.

enumerating $\mathscr{S}_{w_1}(\mathbf{V})$ and $\mathscr{S}_{w_2}(\mathbf{V})$ (obviously, it must be $w = w_1 + w_2$). This way, we produce a list

$$\mathcal{K} = \{\mathbf{X} \in \mathscr{S}_w(\mathbf{V}) \mid \mathbf{X}\mathbf{H}^{*\top} = \mathbf{E}\mathbf{S}^\top\}.$$

Each element in $\mathcal{K}$ is a candidate for $\widetilde{\mathbf{V}}_J$.

Notice that if $w < n - r$ we cannot use the elements of $\mathcal{K}$ for (9), since we do not have enough variables to fill an information set. Hence, we need to consider some new sparse equations involving new variables. Namely, we want to use linear equations that involve only coordinates of $\widetilde{\mathbf{V}}$ that are indexed by $J' \cup J$, where J' is disjoint with J. These equations can be found using the systematic form of $\mathbf{H}$. As in the KMP algorithm, we then build two lists $\mathcal{L}_1$ and $\mathcal{L}_2$, with candidates for $\widetilde{\mathbf{V}}_{J'}$ and $\widetilde{\mathbf{V}}_J$, respectively. The elements in $\mathcal{L}_1$ are obtained by enumeration, while the ones in $\mathcal{L}_2$ are picked from $\mathcal{K}$. This is the main difference with the KMP algorithm: the list $\mathcal{L}_2$ actually contains a subset of $\mathscr{S}_w(\mathbf{V})$, namely, all the matrices that satisfy the equations imposed by $\mathbf{H}^*$. Because of this difference, we say that our algorithm can be thought of as the KMP algorithm plus an initial filtering step. In other words, the equations represented by $\mathbf{H}^*$ are used to filter the elements that we would have used to populate the list $\mathcal{L}_2$ for the KMP algorithm.

In Figure 6 we show a graphical representation of how the algorithm operates. We point out that, in practice, the procedure is slightly more involved because of some technical caveats that are needed, in order to avoid a rank deficiency in the equations which are used to merge $\mathcal{L}_1$ and $\mathcal{L}_2$. Indeed, we want to find sparse equations that contain the coordinates indexed by J, but we already know that J is the (small) support of a subcode with dimension d. This affects the rank of such equations. As we detail in the following, we will need to row-reduce with respect to a specific set and, no matter what, will always have some rank-deficiency: we will use $\widetilde{r}$ equations from the systematic form, but these will have rank $\widetilde{r} - d$.

Our algorithm can improve with respect to KMP since the initial sparse equations can be obtained essentially for free. Indeed, the cost of finding such equations if much smaller than the cost of all the other operations (i.e., creating and merging lists). Improvements on KMP can be obtained only when the algorithm uses subcodes such that $w < d + n - r$. If the only subcodes we are able to find have $w = d + n - r$, then the equations we use for the initial filtering are as sparse as those that we would obtain with the systematic form: our algorithm is not different than KMP.

The algorithmic description of our new approach is reported in Algorithm 2. To visualize the matrices which are used in the algorithm, we refer to Figure 7. In the next proposition we show that the algorithm is correct, i.e., that its execution always results in a solution for the PKP instance $\{\mathbf{H}, \mathbf{V}, \mathbf{E}\}$.

Algorithm 2: Combinatorial algorithm to solve PKP

```
   Data: $w_1, w_2, d, \widetilde{r} \in \mathbb{N}$, such that $w = w_1 + w_2 \leqslant n$, $d \leqslant \widetilde{r} \leqslant r$
   Input: $\mathbf{H} \in \mathbb{F}_q^{r\times n}$, $\mathbf{E} \in \mathbb{F}_q^{\ell\times r}$, $\mathbf{V} \in \mathbb{F}_q^{\ell\times n}$
   Output: $\pi \in S_n$ such that $\pi(\mathbf{V})\mathbf{H}^\top = \mathbf{E}$

   /* Find subcode with support size $w$ and dimension $d$ */
1  Use ISD to find $\mathbf{H}^*$, generator matrix of $\mathcal{B} \subseteq \mathcal{C}^\perp$, with dimension $d$ and support size $w = w_1 + w_2$;

   /* Solve equations represented by $\mathbf{H}^*$ with meet-in-the-middle */
2  Compute $\mathbf{S} \in \mathrm{GL}_{d,r}$ such that $\mathbf{H}^* = \mathbf{SH}$;
3  Compute $\sigma \in S_n$ such that $\mathrm{Supp}\big(\sigma(\mathbf{H}^*)\big) = \{n - r + \widetilde{r} - w + 1, \cdots, n - r + \widetilde{r}\}$;
4  Set $J_1 = \{n - r + \widetilde{r} - w + 1, \cdots, n - r + \widetilde{r} - w_2\}$, $J_2 = \{n - r + \widetilde{r} - w_2, \cdots, n - r + \widetilde{r}\}$;
5  Set $\mathbf{E}^* = \mathbf{ES}^\top$, $\mathbf{H}_1^* = \big(\sigma(\mathbf{H}^*)\big)_{J_1}$, $\mathbf{H}_2^* = \big(\sigma(\mathbf{H}^*)\big)_{J_2}$;
6  Prepare $\mathcal{K}_1 = \left\{\left(\mathbf{Y}_1, \mathbf{Y}_1\mathbf{H}_1^{*\top}\right) \middle| \mathbf{Y}_1 \in \mathscr{S}_{w_1}(\mathbf{V})\right\}$, $\mathcal{K}_2 = \left\{\left(\mathbf{Y}_2, \mathbf{E}^* - \mathbf{Y}_2\mathbf{H}_2^{*\top}\right) \middle| \mathbf{Y}_2 \in \mathscr{S}_{w_2}(\mathbf{V})\right\}$;
7  Compute $\mathcal{K} = \mathcal{K}_1 \bowtie \mathcal{K}_2$;

   /* Solve equations represented by $\mathbf{H}^*$ with meet-in-the-middle */
8  Compute $\mathbf{M} \in \mathrm{GL}_{r,r}$ such that $\mathbf{M}\sigma(\mathbf{H}) = (\mathbf{U}, \mathbf{I}_r)$;
9  Set $\widetilde{\mathbf{H}}$ as the matrix formed by the rows of $(\mathbf{U}, \mathbf{I}_r)$ in positions $\{d + 1, \cdots, u\}$;
10 Set $\widetilde{\mathbf{E}}$ as the matrix formed by the columns of $\mathbf{EM}^\top$ in positions $\{d + 1, \cdots, u\}$;
11 Set $J' = \{1, \cdots, n - r + \widetilde{r} - w\}$ and $J = \{n - r + \widetilde{r} - w + 1, \cdots, n - r + \widetilde{r}\}$;
12 Set $\widetilde{\mathbf{H}}_1 = \widetilde{\mathbf{H}}_{J'}$, $\widetilde{\mathbf{H}}_2 = \widetilde{\mathbf{H}}_J$;
13 Prepare $\mathcal{L}_1 = \left\{\left(\mathbf{X}_1, \mathbf{X}_1\widetilde{\mathbf{H}}_1^\top\right) \middle| \mathbf{X}_1 \in \mathscr{S}_{n-r+u-w}(\mathbf{V})\right\}$, $\mathcal{L}_2 = \left\{\left(\mathbf{X}_2, \widetilde{\mathbf{E}} - \mathbf{X}_2\widetilde{\mathbf{H}}_2^\top\right) \middle| \mathbf{X}_2 \in \mathcal{K}\right\}$;
14 Compute $\mathcal{L} = \mathcal{L}_1 \bowtie \mathcal{L}_2$;

   /* Test each produced candidate */
15 Set $\widetilde{\mathbf{U}}$ as the matrix formed by the last $r - \widetilde{r}$ rows of $\mathbf{U}$;
16 Set $\widetilde{\mathbf{E}}$ as the matrix formed by the last $r - \widetilde{r}$ rows of $\mathbf{eM}^\top$;
17 for $\mathbf{X} \in \mathcal{L}$ do
18     Set $\mathbf{X}' = \widetilde{\mathbf{E}} - \mathbf{X}_{\{1,\cdots,n-r\}}\widetilde{\mathbf{U}}^\top$;
19     if $(\mathbf{X}, \mathbf{X}') \in \mathscr{S}_n(\mathbf{V})$ then
20         Compute $\pi$ such that $\sigma\big(\pi(\mathbf{X}, \mathbf{X}')\big) = \mathbf{V}$;
21         Return $\pi$.
```

Proposition 8 *Algorithm 2 is correct, i.e., it always returns a solution for the input PKP instance.*

Proof. In the following, we will indicate by π the solution to PKP and $\widetilde{\mathbf{V}} = \pi(\mathbf{V})$. Since π is a solution, i.e., $\pi(\mathbf{V})\mathbf{H}^\top = \mathbf{E}$, then

$$\mathbf{ES}^\top = \sigma\big(\widetilde{\mathbf{V}}\big)\big(\mathbf{S}\sigma(\mathbf{H})\big)^\top, \quad \forall \mathbf{S} \in \mathbb{F}_q^{r'\times r}, \ \forall r' \leqslant r, \ \forall \sigma \in S_n. \tag{14}$$

In line 1 of the algorithm, a subcode of the dual code $\mathcal{C}^\perp$ (i.e., the code generated by $\mathbf{H}$) is found. We denote by $\mathbf{H}^* \in \mathbb{F}_q^{d\times n}$ a generator matrix for such a subcode. Observe that $\mathbf{H}^* = \mathbf{SH}$ for some $\mathbf{S} \in \mathrm{GL}_{d,r}$. Thanks to the equality in (14), we have

$$\mathbf{E}^* = \mathbf{ES}^\top = \widetilde{\mathbf{V}}(\mathbf{SH})^\top = \widetilde{\mathbf{V}}\mathbf{H}^{*\top}.$$

The permutation σ we compute in line 3 is only considered to simplify the description, since it is such that $\sigma(\mathbf{H}^*)$ has the only non-null columns at positions $\{n - r + \widetilde{r} - w + 1, \cdots, n - r + \widetilde{r}\}$. Again, thanks to, (14), we are guaranteed that

$$\mathbf{E}^* = \sigma(\widetilde{\mathbf{V}})\sigma(\mathbf{H}^*)^\top.$$

Since $\sigma(\mathbf{H}^*)$ has only w non null columns (the ones indexed by $J = J_1 \cup J_2 = \{n - r + \widetilde{r} - w + 1, \cdots, n - r + \widetilde{r}\}$), solving the above equation allows to find all candidates for $\sigma(\widetilde{\mathbf{V}})_J$. Indeed, given that the only non-null columns of $\sigma(\mathbf{H}^*)$ are at the positions indexed by J, we can write

$$(\mathbf{0}_{d\times(n-r+\widetilde{r}-w)}, \sigma(\widetilde{\mathbf{V}})_J, \mathbf{0}_{d\times(r-\widetilde{r})})\sigma(\mathbf{H}^*)^\top = \mathbf{E}^*.$$

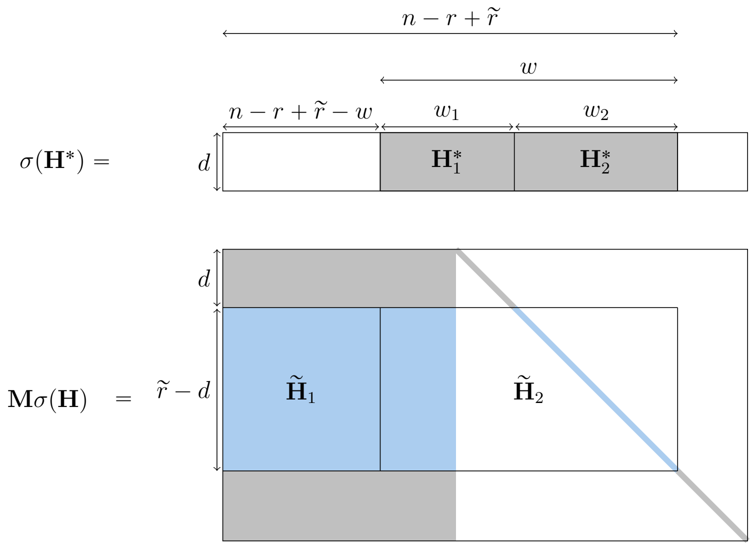


Fig. 7: Matrices employed in Algorithm 2 and respective sizes. The sub-matrix highlighted in light blue is $\widetilde{\mathbf{H}}$

The list $\mathcal{K}$ contains all matrices $(\mathbf{Y}_1, \mathbf{Y}_2) \in \mathscr{S}_w(\widetilde{\mathbf{V}})$ such that $(\mathbf{0}_{d\times(n-r+\widetilde{r}-w)}, \mathbf{Y}_1, \mathbf{Y}_2, \mathbf{0}_{d\times(r-\widetilde{r})})\sigma(\mathbf{H}^*)^\top = \mathbf{E}^*$, hence it also contains $\sigma(\widetilde{\mathbf{V}})_J$.

The matrices $\widetilde{\mathbf{H}}$ and $\widetilde{\mathbf{E}}$ which are computed in lines 9 and 10 of the algorithm are obtained by puncturing[6] $\widetilde{\mathbf{M}}\sigma(\mathbf{H})$ and $\widetilde{\mathbf{E}} = \mathbf{E}\widetilde{\mathbf{M}}^\top$, where $\widetilde{\mathbf{M}} \in \mathrm{GL}_{\widetilde{r}-d,r}$ is such that $\widetilde{\mathbf{M}}\sigma(\mathbf{H})^\top$ has, on the rightmost part, $r-\widetilde{r}$ null columns. Hence, (14) continues to hold. Notice that $J' \cup J = \{1, \cdots, n-r+\widetilde{r}\}$ and that $\mathcal{L}$ contains candidates for $\widetilde{\mathbf{V}}_{\{1,\cdots,n-r+\widetilde{r}\}}$. In particular, we consider all elements of $\mathscr{S}_{n-r+\widetilde{r}-w}(\mathbf{V})$ to build the list $\mathcal{L}_1$, while to build $\mathcal{L}_2$ we use the elements in $\mathcal{K}$. We populate $\mathcal{L}_1$ with all possible candidates for $\sigma(\widetilde{\mathbf{V}})_{J'}$ while, as we have already seen, $\mathcal{K}$ contains $\sigma(\widetilde{\mathbf{V}})_J$. So, the merge between these two lists is guaranteed to contain $\sigma(\widetilde{\mathbf{V}})_{\{1,\cdots,n-r+\widetilde{r}\}}$.

Notice that each $\mathbf{X} \in \mathcal{L}$ is a matrix with $n-r+\widetilde{r}$ columns. To compute the remaining $r-\widetilde{r}$ columns, which are denoted as $\mathbf{X}'$ in the algorithm, we exploit the systematic form $(\mathbf{U}, \mathbf{I}_r)$. Notice that, if $(\mathbf{X}, \mathbf{X}') \in \mathscr{S}_n(\mathbf{V})$, then this implies that $\sigma(\widetilde{\mathbf{V}}) = (\mathbf{X}, \mathbf{X}') = \sigma\big(\pi(\mathbf{V})\big)$. From this relation, we can easily find π. □

Remark 5. The additional permutation σ has been used only to simplify the description of the algorithm, namely, to allow having the pivoted columns on the right (after RREF). We choose σ so that the support of $\mathbf{H}^*$ is moved at the bottom of $\{1, \cdots, n-r+\widetilde{r}\}$. We argue that, without doing this, computing the desired systematic form for $\mathbf{H}$ would not be possible. For instance, consider a permutation σ that, instead, moves J in the first w positions: the resulting $\sigma(\mathbf{H})$ cannot be brought in systematic form. To see this, let $\mathbf{S} \in \mathrm{GL}_{d,r}$ such that $\mathbf{S}\sigma(\mathbf{H}) = \sigma(\mathbf{H}^*)$ has, on the rightmost part, $n-w$ null columns. Let $\mathbf{M} = \begin{pmatrix}\mathbf{S}' \\ \mathbf{S}\end{pmatrix}$, where $\mathbf{S}' \in \mathbb{F}_q^{(r-d)\times r}$ is such that $\mathbf{M}$ has full rank. Notice that $\mathbf{M}\sigma(\mathbf{H})$ is a matrix having, on the bottom rightmost part, a null matrix of size $d \times (n-w)$. This means that the rightmost $r \times r$ matrix of $\mathbf{M}\sigma(\mathbf{H})$ cannot have full rank (it contains d null rows) and, consequently, $\sigma(\mathbf{H})$ cannot be brought in systematic form.

In the following proposition we derive the time complexity of Algorithm 2. As for KMP, we express the cost in terms of linear algebra and lists operations.

Proposition 9 *Let d, w_1, w_2 such that $w = w_1 + w_2 \leqslant n$, $d \leqslant \widetilde{r} \leqslant r$ and $\breve{N}_{w,d} > 1$. Then, Algorithm 2 runs in average time*

$$T_{\mathsf{ISD}}^{(d)}(n, r, w) + T_{\mathcal{K}} + T_{\mathcal{L}} + |\mathcal{L}|,$$

[6] We discard the last $r-\widetilde{r}$ columns.

where

$$T_{\mathcal{K}} \doteq \frac{n!}{(n-w_1)!} + \frac{n!}{(n-w_2)!} + \frac{(n!)^2 q^{-d\ell}}{(n-w_1)!(n-w_2)!},$$

$$T_{\mathcal{L}} \doteq \frac{n!}{(r+w-\widetilde{r})!} + |\mathcal{K}| + \frac{n!}{(r+w-\widetilde{r})!} \cdot |\mathcal{K}| \cdot q^{-\ell(\widetilde{r}-d)},$$

$$|\mathcal{L}| \doteq \frac{(n-w)!q^{-(\widetilde{r}-d)\ell}}{(r-\widetilde{r})!} \cdot |\mathcal{K}|,$$

and $|\mathcal{K}| \doteq \max\left\{\frac{n!}{(n-w)!}q^{-d\ell}\ ,\ 1\right\}$.

Proof. Since $\breve{N}_k(w,d) > 1$, we expect that $\mathcal{C}^\perp$ contains at least a subcode with dimension d and support size w. To find such a subcode, we must face a cost given by $T^{(d)}_{\mathsf{ISD}}(n,r,w)$.

Steps 2–5 are obtained with some basic linear algebra, so we omit their cost. The cost of building the lists $\mathcal{K}_1$ and $\mathcal{K}_2$ is given by

$$|\mathcal{K}_1| + |\mathcal{K}_2| = \frac{n!}{(n-w_1)!} + \frac{n!}{(n-w_2)!}.$$

Every time we find a collision between two elements $(\mathbf{Y}_1, \mathbf{Y}_2)$, we need to check if there are no repeated columns. This can be done very efficiently: for instance, the schoolbook approach (i.e., scanning all pairs of columns) would take time $O(\ell w^2)$. Consequently, we consider that each collision is checked with one elementary operation. Assuming that each term $\mathbf{Y}_1\mathbf{H}_1^{*\top}$ and $\mathbf{E}^* - \mathbf{Y}_2\mathbf{H}_2^{*\top}$ is a random $\ell \times d$ matrix over $\mathbb{F}_q$, we can estimate the average number of collisions as

$$\frac{|\mathcal{K}_1| \cdot |\mathcal{K}_2|}{q^{d\ell}} = \frac{(n!)^2 q^{-d\ell}}{(n-w_1)!(n-w_2)!}.$$

Consequently, we estimate the cost of instructions 2–7 as

$$T_{\mathcal{K}} \doteq \frac{n!}{(n-w_1)!} + \frac{n!}{(n-w_2)!} + \frac{(n!)^2 q^{-d\ell}}{(n-w_1)!(n-w_2)!}.$$

After collisions are filtered, $\mathcal{K}$ contains, on average, $\max\left\{\frac{n!}{(n-w)!}q^{-d\ell}\ ,\ 1\right\}$ elements. Notice that we need to use the max operator since, knowing that the algorithm is correct, existence of at least one solution is guaranteed. Hence, we set $|\mathcal{K}| = \max\left\{\frac{n!}{(n-w)!}q^{-d\ell}\ ,\ 1\right\}$.

Steps 8–11 involve only linear algebra, hence their cost will be neglected. The cost to build $\mathcal{L}_1$ and $\mathcal{L}_2$ is, again, estimated by counting the number of elements. Notice that $|\mathcal{L}_1| = \frac{n!}{n-(n-r+\widetilde{r}-w)!} = \frac{n!}{(r+w-\widetilde{r})!}$ while $\mathcal{L}_2$ has an average number of elements given by $|\mathcal{K}|$. Given that $\mathbf{X}_1\widetilde{\mathbf{H}}_1^\top$ and $\widetilde{\mathbf{E}} - \mathbf{X}_2\widetilde{\mathbf{H}}_2^\top$ are $\ell \times (\widetilde{r}-d)$ matrices, we assume that each pair of list elements collides with probability $q^{-\ell(\widetilde{r}-d)}$. Hence, the average number of collisions is

$$\begin{aligned}\frac{|\mathcal{L}_1| \cdot |\mathcal{L}_2|}{q^{\ell(\widetilde{r}-d)}} &= \frac{n!}{(r+w-\widetilde{r})!} \cdot |\mathcal{K}| \cdot q^{-\ell(\widetilde{r}-d)} \\ &= \max\left\{\frac{(n!)^2 q^{-\widetilde{r}\ell}}{(r+w-\widetilde{r})!(n-w)!}\ ,\ \frac{n! q^{-\ell(\widetilde{r}-d)}}{(r+w-\widetilde{r})!}\right\}.\end{aligned}$$

Consequently, the cost of executing steps 13–14 is

$$T_{\mathcal{L}} \doteq \frac{n!}{(r+w-\widetilde{r})!} + |\mathcal{K}| + \frac{n!}{(r+w-\widetilde{r})!} \cdot |\mathcal{K}| \cdot q^{-\ell(\widetilde{r}-d)}.$$

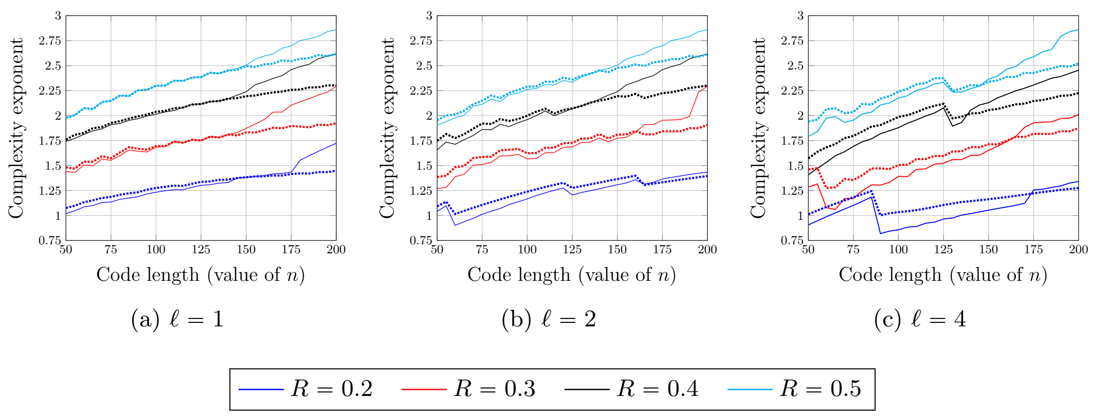
To conclude the proof, we need to estimate the cost of going through instructions 15–20, i.e., to test each element of $\mathcal{L}$. Each test requires basic matrix operations, hence we can use $|\mathcal{L}|$ as an estimate for the running time. We consider that for each matrix $\mathbf{X}_2$ in $\mathcal{L}_2$, the number of matrices $\mathbf{X}_1$ that i) do not have columns identical to those of $\mathbf{X}_2$, and ii) provide a collision can be obtained as $\frac{(n-w)!q^{-(\widetilde{r}-d)\ell}}{\big(n-w-(n-r+\widetilde{r}-w)\big)!} = \frac{(n-w)!q^{-(\widetilde{r}-d)\ell}}{(r-\widetilde{r})!}$. Multiplying this quantity by the number of elements in $\mathcal{L}_2$, we obtain that the average size of $\mathcal{L}$ is $O\left(\frac{(n-w)!q^{-(\widetilde{r}-d)\ell}}{(r-\widetilde{r})!} \cdot |\mathcal{K}|\right)$. □

6.2 Comparison with KMP

In this section we compare the performances of our algorithm with those of KMP. For both algorithms, we consider the finite regime, that is: for KMP, we use the estimate of T_{KMP} resulting from Proposition 5, while for our algorithm we refer to Proposition 9. In Figures 8a, 8b, 8c we report the complexity exponents of the two approaches for different code rates R, several code lengths n and the cases of $\ell = 1$, 2 and 4. The figures report the complexity exponent, that is: for a running time equal to T, we plot $\frac{1}{n}\log_2\left(T\right)$. As it can be seen, our algorithm can indeed be faster than KMP in several occasions. Actually, for any tested value of ℓ and unless n gets too large, our algorithm performs better than KMP. We also notice that, for larger value of ℓ, the improvement becomes more significant. Instead, when we approach the asymptotic regime (i.e., when n gets larger), our algorithm becomes slower; in the following, we provide a justification of this behavior. Recalling Proposition 1, for fixed d and increasing q, the value of ω^* (i.e., the ratio between the code length and the minimum support size for d-dimensional subcodes) tends to $1 - R - \frac{d}{n}$. When n increases, q increases as well (recall (12)), consequently ω^* tends to $1 - R + \frac{d}{n}$. Since here we are targeting subcodes in the code generated by $\mathbf{H}$, we need to replace R with $R^\perp = \frac{r}{n}$. Consequently, for our interest case, we have that ω^* tends to $1 - R^\perp + \frac{d}{n}$. This implies that the minimum support size we can hope to find, when n gets larger, tends to

$$w^* = n\omega^* = n - r + d.$$

This value of w^* is exactly the same support size we would obtain with any RREF: subcodes with these properties are already, implicitly, employed in the KMP algorithm. Remember that our algorithm can improve upon KMP only w and d can be chosen such that $w < n - r + d$. As we have just shown, for large values of n such parameters cannot be chosen because subcodes with the desired properties do not exist: this explains why, for increasing n, our algorithm becomes slower than KMP.



(a) $\ell = 1$ (b) $\ell = 2$ (c) $\ell = 4$

Fig. 8: Complexity exponents for our algorithm and KMP, for $\ell = 1$ and several code rates (indicated by R). Continuous lines are referred to our algorithm, dotted lines to the KMP algorithm.

To have more insight on how our algorithm compares with KMP, in Table 2 we consider some other examples. To analyze parameters with practical interest, we focus on the PKP-DSS instances recommended in [8]; the rows associated to these parameters are highlighted in grey in Table 2. For each instance, we fix the values of n and r, increase the values of ℓ and, consequently, recompute q. For our algorithm, the table also contains the parameters which optimize the attack. We can see that, in several cases, our algorithm is significantly faster than KMP.

In particular, the speed-up gets larger for increasing values of ℓ. This is due to the fact that, when ℓ increases, the required value of q gets lower. Recalling again Proposition 1, this implies that we can run our algorithm with more aggressive parameters, that is, larger values of d. This makes the initial filtering stage more powerful since, when d gets larger, the elements in $\mathcal{K}_1$ and $\mathcal{K}_2$ must collide on a larger number of equations. Again, when we cannot choose w and d so that

$w < n - r + d$, our algorithm should not be faster than KMP. For example, consider the first two instances with $(n, r) = (106, 48)$. Our algorithm is optimized by choosing $d = 21$ and $w = 79$, but $n - r + d = 79$: as we can see from the table, our algorithm has essentially the same running time of KMP.

Finally, we observe that the relevant term in the complexity of our algorithm is never $T^{(d)}_{\mathsf{ISD}}(n, r, w)$: the cost of ISD is always much smaller than the cost of creating and merging the lists.

Table 2: Comparison between the running times (in $\log_2$ units) of KMP and Algorithm 2

(n, r, q, ℓ)	KMP	$(d, w, w_1, \tilde{r})$	Algorithm 2
$(69, 42, 251, 1)$	127.37	$(1, 22, 2, 16)$	125.47
$(69, 42, 17, 2)$	125.26	$(14, 40, 20, 27)$	121.34
$(69, 42, 7, 3)$	124.32	$(3, 24, 5, 16)$	118.45
$(69, 42, 5, 4)$	118.85	$(11, 36, 18, 22)$	110.87
$(69, 42, 3, 5)$	128.00	$(8, 30, 12, 22)$	115.16
$(69, 42, 3, 6)$	118.10	$(10, 33, 16, 20)$	101.72
$(69, 42, 3, 7)$	112.20	$(8, 30, 15, 16)$	91.04
$(69, 42, 2, 8)$	127.00	$(12, 34, 17, 24)$	105.77
$(69, 42, 2, 9)$	119.64	$(10, 31, 15, 20)$	96.52
$(94, 55, 509, 1)$	190.82	$(1, 31, 2, 22)$	189.77
$(94, 55, 23, 2)$	190.48	$(2, 32, 3, 22)$	186.43
$(94, 55, 11, 3)$	178.89	$(2, 30, 4, 18)$	173.75
$(94, 55, 5, 4)$	189.36	$(4, 34, 6, 23)$	179.85
$(94, 55, 5, 5)$	172.07	$(3, 31, 6, 17)$	163.40
$(94, 55, 3, 6)$	185.98	$(18, 54, 27, 34)$	172.52
$(94, 55, 3, 7)$	175.68	$(14, 49, 24, 27)$	159.03
$(94, 55, 3, 8)$	166.51	$(11, 45, 22, 22)$	148.03
$(94, 55, 2, 9)$	190.76	$(11, 42, 16, 29)$	169.04
$(106, 48, 4093, 1)$	257.40	$(21, 79, 39, 23)$	257.40
$(106, 48, 67, 2)$	256.97	$(21, 79, 39, 23)$	256.97
$(106, 48, 17, 3)$	256.41	$(1, 40, 2, 21)$	251.56
$(106, 48, 11, 18)$	245.74	$(1, 39, 3, 18)$	241.13
$(106, 48, 7, 5)$	245.71	$(1, 39, 3, 18)$	239.95
$(106, 48, 5, 6)$	245.72	$(2, 41, 5, 19)$	238.33
$(106, 48, 5, 7)$	237.49	$(2, 40, 6, 16)$	227.60
$(106, 48, 3, 8)$	251.90	$(3, 43, 6, 22)$	241.35
$(106, 48, 3, 9)$	244.82	$(3, 42, 7, 19)$	230.59

6.3 Comparison with the attack in [21]

To make another practical example, we consider the PKP instances recommended in [20], for which (n, r, q, ℓ) are set as $(38, 16, 2, 10)$ and $(42, 16, 2, 11)$. Notice that we are already referring to the non-homogeneous version, i.e., we are considering $r = m + 1$. In Table 3 we report the time complexities of Algorithm 2, and compare them with those of the attack in [21] and with KMP. There instances have originally been proposed for security levels of 79 and 89 bits, but [21] shows how they can be attacked in approximate times 2^{63} and 2^{77}, respectively. Coherently with the literature on PKP (and with the analysis we have performed in this paper), the estimates in [21] count the number of matrix by matrix multiplications. Notice that, interestingly, the attack in [21] has some similarities with our approach. Indeed, it starts by repeatedly calling ISD to find low-weight codewords in both codes (i.e., the one generated by $\mathbf{V}$ and the one having $\mathbf{H}$ as parity-check matrix). Then, the secret permutation is reconstructed by matching the locations of set entries. In our attack, we search only for low weight codewords (or subcodes) in one code (that is, the one having $\mathbf{H}$ as parity-check matrix), and then use them to speed-up the KMP algorithm.

As we can see from the table, the running time of our algorithm is lower than that of [21]. Namely, the gain is approximately 5 bits for the first instance, and 8 bits for the second one. In addition, our algorithm works regardless of the considered finite field, while the one in [21] is specific to the binary case.

For a completely fair comparison, we observe that our algorithm requires an exponential amount of memory, while the one in [21] has a much lower space complexity. In principle, an algorithm running in time T and using a large memory M should somehow be penalized, in the sense that its overall cost should be larger than T. This additional cost is normally neglected and only the running time is considered. Yet, for the sake of completeness, we briefly comment about this fact. Notice that establishing how a large memory usage affects the performance of an algorithm is a rather complex task. We stick to the analysis in [12], where the authors conclude that a logarithmic cost seems to be the most appropriate one, for the case of ISD algorithms. Given that the large space complexity of advanced ISD algorithms is essentially due to operations with lists, it makes sense to extrapolate this result and apply it also to our case. Consequently, for an algorithm with time complexity T and space complexity M, we consider an overall cost of $T \cdot \log_2(M)$. Given that, for Algorithm 2, we have $M \approx T$, we use $T\log_2(T)$ as an estimate of its cost.

Even if we assume no penalty for the attack in [21] (the authors claim that the employed memory is negligible), our algorithm remains competitive. Indeed, the costs for our algorithm would become approximately 63 (instead of 57.7) and 75 (instead of 69.2), which are in the same ballpark of [21].

Table 3: Comparison between the running times (in $\log_2$ units) of KMP, the attack in [21] and algorithm 2, for the instances recommended in [20]

(n, r, q, ℓ)	KMP	[21]	$(d, w, w_1, \widetilde{r})$	Algorithm 2
$(38, 16, 2, 10)$	74.9	63	$(5, 21, 10, 10)$	57.7
$(42, 16, 2, 11)$	87.4	77	$(6, 26, 13, 11)$	69.2

7 Further considerations

Arguably, the interest in PKP and SEP is mainly due to their cryptographic applications. To the best of our knowledge, these problems have been used only in the design of signature schemes in the Fiat-Shamir paradigm. Yet, we cannot exclude that, in the future, other applications may appear, e.g., signatures in the hash&sign paradigm or encryption schemes. Analogously, most of the attention has been dedicated to the mono-dimensional PKP. The use of the multi-dimensional PKP has only been considered in [20], but [21] and this paper show that the recommended instances have a security level which is significantly below the claimed one. Yet, this is not enough to conclude that the multi-dimensional PKP is less useful, with respect to the mono-dimensional PKP. Following this line of reasoning, some questions arise naturally; for instance:

- *Can a PKP-based encryption scheme, or a hash&sign signature scheme, be competitive?*
- *Can multi-dimensional PKP be preferable over mono-dimensional PKP, in some cases?*

In the remainder of this section, we argue why these questions may admit, in principle, a positive answer. Namely, we show that to achieve a running time of at least 2^{128} operations (corresponding to a security level of 128 bits) the required input size for PKP (equivalently, SEP) can be rather small, when compared to other problems. This implies that, potentially, PKP and SEP may be employed to design cryptographic schemes with competitive performance. Notice that we adopt a purely speculative point of view, that is, we do not propose any specific construction but only consider the performance that hypothetical such schemes may achieve. Yet, these results hint at the fact that these problems are worth looking into.

7.1 Relevant quantities and scenarios

As it is common when studying hard problems, we first focus on the input size. For PKP, the input is constituted by $\mathbf{H}$, of size $r \times n$, and $\mathbf{V}$, of size $\ell \times n$; both matrices take values in $\mathbb{F}_q$. Notice that, for both matrices and without loss of generality, we can employ a convenient representation and consider the RREF form. Indeed, this allows saving some space, since the identity matrices can be excluded from the input. Consequently, to represent such matrices, the number of bits we need is

$$\mathrm{Size}(\mathbf{H}) = r(n-r)\log_2(q),$$
$$\mathrm{Size}(\mathbf{V}) = \ell(n-\ell)\log_2(q).$$

For the overall input size, we can consequently consider $\mathrm{Size}(\mathbf{H}) + \mathrm{Size}(\mathbf{V})$. Notice that, under a cryptographic point of view, the input size can be interpreted as the public key plus ciphertext size of a hypotethical encryption scheme where $\mathbf{H}$ constitutes the public key and $\mathbf{V}$ is the ciphertext.

Notice that, when considering signature schemes in the Fiat-Shamir paradigm, $\mathbf{H}$ can be fixed or generated at random starting from some seed. Instead, the size of (some) exchanged messages is, more or less, equal to that of $\mathbf{V}$. Consequently, in this scenario, it makes sense to neglect the size of $\mathbf{H}$ and consider only that of $\mathbf{V}$.

7.2 Case study of 128-bit security

Let us focus on the common case of 128 bits of classical security, which is the minimum security level required, nowadays, for cryptographic schemes. We first consider the minimum input size we need, for PKP, to have that all known attacks have a running time not lower than 2^{128}. In other words, we consider several code rates and, for each value of R, find the values of n, q and ℓ such that all known attacks run in time $\geqslant 2^{128}$ and the value of $\mathrm{Size}(\mathbf{H}) + \mathrm{Size}(\mathbf{V})$ is minimized. We first focus on the mono-dimensional case. For the corresponding parameters, see Table 4, where we additionally compare with the results in [11], where the authors derive the minimum input size for the SDP with the low Hamming weight and the high Hamming weight requirements. As we can see from the Table, the PKP can achieve the same complexity with a much smaller input. We also consider how the problem behaves when switching to the multi-dimensional version. Interestingly, the input size can be reduced significantly. Indeed, when ℓ gets larger, we can use smaller values for q, and this has a positive impact on $\mathrm{Size}(\mathbf{V})$. For example, considering the same rate $R \approx 0.51$, we can choose $n = 70$, $r = 32$, $q = 2$ and $\ell = 10$, yielding to an input size of approximately 0.23 kB. Notice that this reduction is mostly due to the fact that representing $\mathbf{H}$ becomes significantly less costly: instead of the 1.12 kB required for the mono-dimensional case, we need only 0.15 kB.

Problem	Parameters	Min. Input Size
SDP, Low Weight	$q = 2$, $R = 0.326$	46.75
SDP, High Weight	$q = 3$, $R = 0.369$	12.38
PKP, $\ell = 1$	$n = 62$, $q = 653$, $r = 30$	1.19

Table 4: Minimum input size, in kB, and corresponding parameters, for different problems and 128 bits of complexity

We now focus on minimizing just the size of $\mathbf{V}$. For $\ell = 1$, we find that the optimal choice is $n = 69$, $q = 239$ and $r = 41$ (the code rate is approximately 0.4), yielding to $\mathrm{Size}(\mathbf{V}) \approx 67$ B. When switching to the multi-dimensional case, we can obtain approximately the same sizes with essentially the same n but a much smaller q. For instance, choosing $n = 68$, $q = 17$ and $\ell = 2$, we obtain, in practice, the same value for $\mathrm{Size}(\mathbf{V})$. We observe that, using the same n, the solution to PKP has the same size; however, we can use a much smaller value for q, and this should make the arithmetic faster.

8 Conclusions

We have studied the hardness to solve the PKP and the SEP. First, we have shown that the two problems are actually equivalent, hence, all solvers for the former can also be used to solve the latter. Despite that this result is based on a very simple reduction, to the best of our knowledge, this is the first time it is made explicit. Then, we have deeply studied the performance of state-of-the-art solvers. For what concerns the KMP algorithm, we have generalized it to the multi-dimensional case (i.e., when $\ell > 1$) and have derived its complexity in the asymptotic regime. Our analysis shows that, perhaps surprisingly, the running time does not depend on the value of ℓ. Also, the algorithm runs in time which is super-exponential in the code length. We have then thoroughly analyzed the algorithm we introduced in [25], extended it to the multi-dimensional case (regardless of the finite field size) and compared it with KMP and the attack in [21], which is specific to the binary field. Our analysis shows that, in the finite regime, our algorithm is in several cases faster than other approaches; instead, in the asymptotic regime, KMP has better performance. Finally, we have considered how PKP and SEP behave in terms of input size. We have shown that they can achieve practical security levels with rather compact inputs, when compared to other problems (say, SDP). Also, switching to the multi-dimensional version has no practical impact on the security, while it can lead to a significantly reduced input size. This analysis hints at the fact that, potentially, PKP and SEP can be used to design very promising quantum-safe cryptographic primitives. As a confirmation of this, the algorithms analyzed in this paper have already been used as a basis for parameters selection for PERK [1], a digital signature scheme which has been submitted to the NIST additional call for PQC signatures.

References

1. Aaraj, N.A., Bettaieb, S., Bidoux, L., Budroni, A., Dyseryn, V., Esser, A., Gaborit, P., Kulkarni, M., Mateu, V., Palumbi, M., Perin, L., Tillich, J.P.: CROSS: Cross & Restricted Objects Signature Scheme. Submission to NIST Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process (2023), `https://www.cross-crypto.com/`
2. Bardet, M., Otmani, A., Saeed-Taha, M.: Permutation code equivalence is not harder than graph isomorphism when hulls are trivial. In: 2019 IEEE International Symposium on Information Theory (ISIT). pp. 2464–2468. IEEE (2019)
3. Barenghi, A., Biasse, J.F., Persichetti, E., Santini, P.: On the computational hardness of the code equivalence problem in cryptography. Advances in Mathematics of Communications **0**, – (2022)
4. Baritaud, T., Campana, M., Chauvaud, P., Gilbert, H.: On the security of the permuted kernel identification scheme. In: Brickell, E.F. (ed.) Advances in Cryptology — CRYPTO' 92. Lecture Notes in Computer Science, vol. 740, pp. 305–311. Springer (1992)
5. Berger, T.P., Gueye, C.T., Klamti, J.B.: A NP-complete problem in coding theory with application to code based cryptography. In: International Conference on Codes, Cryptology, and Information Security. pp. 230–237. Springer (2017)
6. Beullens, W.: Not Enough LESS: An Improved Algorithm for Solving Code Equivalence Problems over $\mathbb{F}_q$. In: International Conference on Selected Areas in Cryptography. pp. 387–403. Springer (2020)
7. Beullens, W.: Sigma protocols for mq, pkp and sis, and fishy signature schemes. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 183–211. Springer (2020)
8. Beullens, W., Faugère, J.C., Koussa, E., Macario-Rat, G., Patarin, J., Perret, L.: PKP-based signature scheme. In: F. Hao, S.R., Gupta, S.S. (eds.) Progress in Cryptology – INDOCRYPT 2019. Lecture Notes in Computer Science, vol. 11898, pp. 3–22. Springer, Cham (2019)
9. Bidoux, L., Gaborit, P.: Shorter Signatures from Proofs of Knowledge for the SD, MQ, PKP and RSD Problems. arXiv preprint arXiv:2204.02915 (2022)
10. Boidje, B.O., Gueye, C.T., Dione, G.N., Klamti, J.B.: Quasi-Dyadic Girault Identification Scheme. In: International Conference on Codes, Cryptology, and Information Security. pp. 307–321. Springer (2019)
11. Bricout, R., Chailloux, A., Debris-Alazard, T., Lequesne, M.: Ternary syndrome decoding with large weight. In: International Conference on Selected Areas in Cryptography. pp. 437–466. Springer (2020)
12. Esser, A., May, A., Zweydinger, F.: McEliece needs a break–solving McEliece-1284 and quasi-cyclic-2918 with modern ISD. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 433–457. Springer (2022)
13. Feneuil, T.: Building MPCitH-based Signatures from MQ, MinRank, Rank SD and PKP. Cryptology ePrint Archive (2022)
14. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, San Francisco (1979)
15. Georgiades, J.: Some remarks on the security of the identification scheme based on permuted kernels. Journal of Cryptology **5**(2), 133–137 (1992)
16. Girault, M.: A (non-practical) three-pass identification protocol using coding theory. In: International Conference on Cryptology. pp. 265–272. Springer (1990)
17. Guruswami, V.: Introduction to coding theory, lecture 2: Gilbert-varshamov bound. University Lecture (2010)

18. Jaulmes, É., Joux, A.: Cryptanalysis of PKP: a new approach. In: K., K. (ed.) Public Key Cryptography. PKC 2001. Lecture Notes in Computer Science, vol. 1992, pp. 165–172. Springer, Berlin, Heidelberg (2001)
19. Koussa, E., Macario-Rat, G., Patarin, J.: On the complexity of the Permuted Kernel Problem. Cryptology ePrint Archive, Report 2019/412 (2019), `https://ia.cr/2019/412`
20. Lampe, R., Patarin, J.: Analysis of some natural variants of the PKP algorithm. Cryptology ePrint Archive (2011)
21. Paiva, T.B., Terada, R.: Cryptanalysis of the binary permuted kernel problem. In: International Conference on Applied Cryptography and Network Security. pp. 396–423. Springer (2021)
22. Patarin, J., Chauvaud, P.: Improved algorithms for the permuted kernel problem. In: Stinson, D.R. (ed.) Cryptology — CRYPTO' 93 Proceedings. CRYPTO 1993. Lecture Notes in Computer Science, vol. 773, pp. 391–402. Springer, Berlin, Heidelberg (1993)
23. Petrank, E., Roth, R.M.: Is code equivalence easy to decide? IEEE Transactions on Information Theory **43**(5), 1602–1604 (1997)
24. Poupard, G.: A realistic security analysis of identification schemes based on combinatorial problems. European Transactions on Telecommunications **8**(5), 471–480 (1997)
25. Santini, P., Baldi, M., Chiaraluce, F.: A Novel Attack to the Permuted Kernel Problem. In: 2022 IEEE International Symposium on Information Theory (ISIT). pp. 1441–1446 (2022). https://doi.org/10.1109/ISIT50566.2022.9834867
26. Sendrier, N.: On the dimension of the hull. SIAM Journal on Discrete Mathematics **10**(2), 282–293 (1997)
27. Sendrier, N.: Finding the permutation between equivalent linear codes: The support splitting algorithm. IEEE Transactions on Information Theory **46**(4), 1193–1203 (2000)
28. Shamir, A.: An efficient identification scheme based on permuted kernels. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89 Proceedings. CRYPTO 1989. Lecture Notes in Computer Science, vol. 435, pp. 606–609. Springer (1989)

Appendix A - Proof of Proposition 1

We have already observed that the bounds $\widecheck{N}_k(w,d)$ and $\widehat{N}_k(w,d)$ are tight, up to a factor which is not greater than $e^1 \approx 2.7183$: so, we can reliably use $\widecheck{N}_k(w,d)$ in place of $N_k(w,d)$. Let $k = Rn$, for a constant $R \in [0;1]$; since d is constant as well, from (5) we have

$$\frac{\left[\begin{smallmatrix}k\\d\end{smallmatrix}\right]_q}{\left[\begin{smallmatrix}n\\d\end{smallmatrix}\right]_q} = \frac{\left[\begin{smallmatrix}Rn\\d\end{smallmatrix}\right]_q}{\left[\begin{smallmatrix}n\\d\end{smallmatrix}\right]_q} \sim 2^{d(k-n)\log_2(q)} = 2^{dn(R-1)\log_2(q)}.$$

From (4), and neglecting the $o(1)$ (since they vanish for growing n), we have that

$$\begin{aligned}\binom{n}{w}(q^d-1)^{w-d} &= \binom{n}{\omega n}(q^d-1)^{w-d}\\ &= \binom{n}{\omega n}(q^d-1)^{w}(q^d-1)^{-d}\\ &= 2^{nh_{q^d}(\omega)\log_2(q^d)-d\log_2(q^d-1)}\\ &= 2^{d\left(nh_{q^d}(\omega)\log_2(q)-\log_2(q^d-1)\right)}.\end{aligned}$$

Then, we can write

$$\widecheck{N}_k(w,d) = 2^{nd\log_2(q)h_{q^d}(\omega)-d\log_2(q^d-1)-dn(1-R)\log_2(q)}.$$

Let ω^* be the minimum $\omega \in [0;1]$ so that $\widecheck{N}_k(\omega n,d) \geqslant 1$. From the above equation, we obtain

$$\omega^* = h_{q^d}^{-1}\left(1-R+\frac{\log_2(q^d-1)}{n\log_2(q)}\right).$$

We further notice that, whenever $q^d \gg 1$, we can further simplify $\log_2(q^d-1) \approx d\log_2(q)$, from which

$$\omega^* \approx h_{q^d}^{-1}\left(1-R+\frac{d}{n}\right).$$

Appendix B - Proof of Proposition 2

ISD is a randomized, iterative procedure in which the steps in Algorithm 3, corresponding to a single iteration, are continuously executed until the algorithm successes.

```
Algorithm 3: One iteration of ISD for $d > 1$
   Input: Code $\mathcal{C}$ generated by $\mathbf{G} \in \mathrm{GL}_{k,n}$, $w, d \in \mathbb{N}$ such that $w \leqslant n - k + d$
   Output: failure, or generator matrix for $\mathcal{B} \subseteq \mathcal{C}$ with dimension $d$ and support size $w$
 1 Choose uniformly at random $\sigma \in S_n$;
 2 if $\mathsf{RREF}\big(\sigma(\mathbf{G}), \{1, \cdots, k\}\big)$ fails then
 3     Report failure;
 4 else
 5     $(\mathbf{I}_d, \mathbf{M}) \leftarrow \mathsf{RREF}\big(\sigma(\mathbf{G}), \{1, \cdots, k\}\big)$
 6 for $U \subseteq \{1, \cdots, k\}$ with size $d$ do
 7     $\mathbf{B} \leftarrow$ matrix formed by rows of $\mathbf{M}$ indexed by $U$;
 8     if $\mathbf{B}$ has support size $w - d$ then
 9         Return $\sigma^{-1}\big((\mathbf{I}_d, \mathbf{B})\big)$
10 Report failure;
```

Consider the running time in Proposition 2. Notice that the numerator of the formula corresponds to the cost of each iteration, while the denominator is the success probability of each iteration. For this probability, we consider that the code contains $N_k(w, d)$ subcodes with dimension d and support size w. The probability that a chosen permutation is valid for one of them is given by $\frac{\binom{w}{d}\binom{n-w}{k-d}}{\binom{n}{k}}$ and, if we multiply such probability by the expected number of subcodes, we obtain the average number of subcodes each ISD iteration is able to find. Now, if this product is smaller than 1, then we can deem it as the success probability. Instead, if it is close to (or greater) than 1, then we can assume that every ISD iteration returns a subcode, so that the success probability of every iteration is practically 1. This reasoning explains why we can set the success probability of each iteration as $p^{(d)}(n, k, w) = \min\left\{\frac{\binom{w}{d}\binom{n-w}{k-d}}{\binom{n}{k}} N_k(w, d)\ ;\ 1\right\}$. Finally, we replace $N_k(w, d)$ with the (tight) lower bound $\underset{\smile}{N}_k(w, d)$.

Appendix C - Asymptotics of KMP algorithm

Proof of Proposition 6

We first notice that the number of elements in $\mathcal{L}$ is always not larger than the number of collisions between $\mathcal{L}_1$ and $\mathcal{L}_2$, that is, $N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}$. Hence, asymptotically, the cost of the algorithm can be assumed as $\max\{|\mathcal{L}_1|\ ,\ |\mathcal{L}_2|\ ,\ N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}\}$, and the algorithm is optimized when the three quantities are identical. To achieve this, we choose $u_1 = u_2 = u = \mu n$, where $\mu \in (R/2; 1]$. This guarantees that $|\mathcal{L}_1| = |\mathcal{L}_2|$ and, recalling the asymptotics in Section 2.3, we have[7]

$$L = |\mathcal{L}_1| = |\mathcal{L}_2| = 2^{n\left(h_2(\mu) + \mu \log_2\left(\frac{\mu n}{e}\right)\right)}.$$

Furthermore, it holds that

$$\begin{aligned} N_{\mathcal{L}_1 \bowtie \mathcal{L}_2} &= L^2 q^{\ell(n - r - 2\mu n)} \\ &= 2^{n\left(2h_2(\mu) + 2\mu \log_2\left(\frac{\mu n}{e}\right) + \ell \log_2(q)(R - 2\mu)\right)}. \end{aligned}$$

Then, it is easy to see that $L = N_{\mathcal{L}_1 \bowtie \mathcal{L}_2}$ happens when μ is such that

$$h_2(\mu) + \mu \log_2\left(\frac{\mu n}{e}\right) + \ell \log_2(q)(R - 2\mu) = 0. \tag{15}$$

These considerations and relationships prove the proposition.

[7] To ease notation, we here neglect the $o(1)$ terms.

Proof of Proposition 7

In the asymptotic regime we have $q \sim \left(\frac{n}{e}\right)^{\frac{1}{\ell(1-R)}}$. Consequently, we rewrite (15) as

$$h_2(\mu) + \mu \log_2\left(\frac{\mu n}{e}\right) + \frac{R-2\mu}{1-R}\log_2\left(\frac{n}{e}\right) = 0. \tag{16}$$

It is easy to see that, for any sufficiently large n, the above equation always admits a root μ in the range $\mu^* \in (R/2; 1/2]$. Indeed, the function on the left side of the equation is continuous, is positive for $\mu = R/2$ and negative for $\mu = 1/2$: consequently, it must have a root in the range $(R/2; 1/2]$. Let ω^* be the limit of the root of (16), for n going at infinity, and consider that

$$\begin{aligned}\lim_{n\to\infty} h_2(\mu^*) + \mu^* \log_2\left(\frac{\mu^* n}{e}\right) + \frac{R-2\mu^*}{1-R}\log_2\left(\frac{n}{e}\right)\\ = \mu^* \log_2\left(\frac{n}{e}\right) + \frac{R-2\mu^*}{1-R}\log_2\left(\frac{n}{e}\right).\end{aligned}$$

Requiring the above limit to be equal to 0, we find

$$\mu^* = \frac{R}{1+R}.$$

Then, we consider that

$$\begin{aligned}c_{\mathsf{KMP}}(\mu^*) &= h_2(\mu^*) + \mu^* \log_2\left(\frac{\mu^* n}{e}\right)\\ &= -(1-\mu^*)\log_2(1-\mu^*) + \mu^* \log_2\left(\frac{n}{e}\right)\\ &= -\frac{1}{1+R}\log_2\left(\frac{1}{1+R}\right) + \frac{R}{1+R}\log_2\left(\frac{n}{e}\right)\\ &= \frac{1}{1+R}\log_2(1+R) + \frac{R}{1+R}\log_2\left(\frac{n}{e}\right).\end{aligned}$$