Checking whether a word is Hamming-isometric in linear time

Marie-Pierre Béal[iD] and Maxime Crochemore[iD]

Univ. Gustave Eiffel, CNRS, LIGM

F-77454 Marne-la-Vallée, France,

August 9, 2022

Abstract

A finite word f is Hamming-isometric if for any two words u and v of the same length avoiding f, u can be transformed into v by changing one by one all the letters on which u differs from v, in such a way that all of the new words obtained in this process also avoid f. Words which are not Hamming-isometric have been characterized as words having a border with two mismatches. We derive from this characterization a linear-time algorithm to check whether a word is Hamming-isometric. It is based on pattern matching algorithms with k mismatches. Lee-isometric words over a four-letter alphabet have been characterized as words having a border with two Lee-errors. We derive from this characterization a linear-time algorithm to check whether a word over an alphabet of size four is Lee-isometric.

Keywords: Isometric words; Pattern matching with mismatches.

1   Introduction

Many parallel processing applications have communication patterns that can be viewed as graphs called d-ary n-cubes. A d-ary n cube is a graph Q_n^d whose nodes are the words of length n over the alphabet $\mathbb{Z}_d = \{0, 1, \ldots, d-1\}$. Two nodes are linked if and only if they differ in exactly one position, and the mismatch is given by two symbols a and b that verify $a = b \pm 1 \mod d$. In order to obtain some variants of hypercubes for which the number of vertices increases slower than in a hypercube, Hsu [9] introduced Fibonacci cubes in which nodes are on a binary alphabet and avoid the factor 11. The notion of d-ary n-cubes has subsequently be extended to define the generalized Fibonacci cube [10, 11, 19] ; it is the subgraph $Q_n^2(f)$ of a 2-ary n-cube whose nodes avoid some factor f. In this framework, a binary word f is said to be Lee-isometric when, for any $n \geq 1$, $Q_n^2(f)$ can be isometrically embedded into Q_n^2, that is, the distance between two words u and v vertices of $Q_n^2(f)$ is the same in $Q_n^2(f)$ and in Q_n^2.

1

On a binary alphabet, the definition of a Lee-isometric word can be equivalently given by ignoring hypercubes and adopting a point of view closer to combinatorics on words. A binary word f is n-Hamming-isometric if for any pair of words u and v of length n avoiding f, u can be transformed into v by exchanging one by one the bits on which they differ meanwhile generating only words avoiding f. The word f is Hamming-isometric if it is n-Hamming-isometric for all n. The structure of binary non-Hamming-isometric words has been characterized in [12, 17, 18] and extended to general alphabets in [2]. In particular, a binary word is Hamming-isometric if and only if it is Lee-isometric. A word is not Hamming-isometric if and only if it has a 2-error border, that is if it has a suffix that mismatches with the prefix of the same length in exactly two positions. In [12, 17, 18] and [2], 2-error border are called 2-error overlap.

In the case of an alphabet of size 4, non-Lee-isometric words have been characterized in [2] as words having a suffix and a prefix of the same length which are at distance 2 according to the Lee distance.

Binary Hamming-isometric words have also been considered in the two-dimensional setting, and non-Hamming-isometric pictures are investigated in [3], where they are called bad pictures.

In this paper we study the algorithmic complexity of checking whether a word is not Hamming-isometric. Our approach is based on the characterization of 2-error borders of such words. The naive algorithm runs clearly in quadratic time. We show that known algorithms for matching patterns with mismatches can be used to solve this problem efficiently. Pattern matching with k mismatches can be solved by algorithms running in time $O(nk)$ (see [13] and [7]). These algorithms are mostly based on a technique called the Kangaroo method. This method computes the Hamming distance for every alignment in time $O(k)$ by "jumping" from one error to the next error. A faster algorithm for pattern matching with k mismatches runs in $O(n\sqrt{k \log k})$ [1]. A simpler version of this algorithm is given in [15].

We show two methods to check whether a word is not Hamming-isometric. The first one uses the Kangaroo method which allows to derive an algorithm running in time $O(kn)$ and using $O(n)$ space to check whether a word of length n has a k-error border. The method has a preprocessing of linear time and space for computing the suffix tree of the word and to enhance it in order to answer lowest common ancestor queries in constant time. This overall leads to a linear-time and linear-space algorithm to check whether a word is not Hamming-isometric, and hence also to check whether a binary word is Lee-isometric. The second method uses the computation of a k-prefix table that gives, for some word u and every position on u, the length of the longest proper factor of u at this position that matches its prefix of the same length with at most k differences [5]. The computation of this k-prefix table is done in time $O(kn)$ using $O(n)$ space.

We also use the Kangaroo method to derive an algorithm running in time $O(kn)$ and using $O(n)$ space on a constant size alphabet to check whether a word of size n has a k-Lee-error border and thus check in linear time whether a word over an alphabet of size 4 is Lee-isometric.

2 Definitions and background

Let A be a finite alphabet. A word u in $A^\star$ is a finite sequence $u[0]u[1]\cdots u[n-1]$ of letters in A, where n is the length of u and $u[i]$ are its letters. The suffix of index i on u, denoted by u_i, is the word $u[i]\cdots u[n-1]$ of length $n-i$. A suffix (or prefix) of it is *proper* if it is distinct from u itself.

Let k be a non-negative integer. We say that a word u has a *k-error border* if u has a proper suffix s that matches its prefix of the same length with exactly k differences. In other words, the Hamming distance between the suffix and the prefix is k.

Example 1 The word 1010011 has a 2-error border. Indeed, it has the prefix 101 and the suffix 011 and the Hamming distance between 101 and 011 is 2.

Let f be a finite word and n be a positive integer. Then a word u is called *f-free* if it does not contain f as a factor, and f is called *n-Hamming-isometric* if for every f-free words u and v of length n, the following holds: u can be transformed into v by changing one by one all the letters on which u differs from v, in such a way that all of the new words obtained during this process are also f-free. Such a transformation is called an *f-free transformation* from u to v. Eventually, a word f is said to be *Hamming-isometric* if it is *n-Hamming-isometric* for every positive integer n.

The *d-ary n-cube*, denoted by Q_n^d, is the graph whose vertices are the words of length n over the alphabet $\mathbb{Z}_d = \{0, 1, \ldots, d-1\}$, and for which any two words u and v are adjacent if and only if u and v differ by one unit at exactly one position, say i, that is, $u[i] = v[i] \pm 1 \mod d$. The *d-ary n-cube avoiding f*, where f is a word over the alphabet $\mathbb{Z}_d$ is the graph $Q_n^d(f)$ obtained from Q_n^d by deleting the vertices containing f as a factor [2].

A word f over $\mathbb{Z}_d$ is said to be *Lee-isometric* if for all $n \geq 1$, $Q_n^d(f)$ is an isometric subgraph of Q_n^d.

Example 2 The word 0301 on the alphabet $\mathbb{Z}_4 = \{0, 1, 2, 3\}$ is non-Lee-isometric. Indeed, the words $u = 030001$ and $v = 030201$, which do not contain the factor 0301, are at distance 2 but there is no path of length 2 from u to v in $Q_6^4(0301)$ since any path of length 1 changing the symbol of index 3 of u goes from u to 030101 or to 030301 and these two words both have the word 0301 as factor.

It is shown in [2] that non-Hamming-isometric and non-Lee-isometric words coincide for words on an alphabet of size at most three. But this property is no more true for larger alphabets.

Hamming-isometric words have the following characterization obtained in [12, 17] for binary alphabets and in [2] for general alphabets.

Proposition 3 *A word is not Hamming-isometric if and only if it has a 2-error border.*

Example 4 For instance the words 11, 1^n for $n \geq 1$ are Hamming-isometric. The word 1010011 is not Hamming-isometric.

3 Algorithms for checking whether a word is Hamming-isometric

In this section, we use the characterization of non-Hamming-isometric words in terms of 2-error border (Proposition 3) and assume that the alphabet A has a constant size. Observe that a quadratic-time naive algorithm can be obtained to check whether a word is non-Hamming-isometric by computing the Hamming distance between each suffix of index i and the prefix of the same length. We show that checking if a word of length n has a k-error border can be done in time $O(kn)$ and space $O(n)$.

Proposition 5 *It can be checked in time $O(kn)$ and space $O(n)$ whether a word of length n has a k-error border.*

Proof. We give two algorithms for solving this problem. The first one is based on a technique called the Kangaroo method used for pattern matching with k mismatches in $O(nk)$ time (see [13], [7] and [14]). These algorithms compute the Hamming distance for every alignment in $O(k)$ time by "jumping" from one error to the next. We use the Kangaroo method to check for each index i on a word u of length n whether it has a k-error border of length $n - i$ in time $O(k)$.

To do so, we first compute in time and space $O(n)$ the suffix tree of the word u. The suffix tree is a compacted trie containing all the suffixes of u by their keys and positions on u as their values [6], [4]. The tree has a linear number of nodes and edges, each edge containing a pair of integers identifying a factor of u, e.g. (position, length), hence the linear space complexity. Suffix arrays can also be used for this problem. They contain essentially the starting positions of suffixes of u sorted in lexicographic order.

To get the overall running time, we need to answer Lowest Common Ancestor (LCA) queries in constant time [8], [16]. LCA queries give us the longest common prefix between two suffixes of u, essentially telling us where the first mismatch appears between a suffix of u and its prefix of the same length. This can be performed by first constructing a Longest Common Prefix (LCP) array. The LCP array stores the length of the longest common prefix between two consecutive suffixes in the suffix array (lexicographic consecutive suffixes). This array can also be constructed in linear time. To compute the length of the longest prefix common to any two suffixes in the suffix tree (instead of consecutive suffixes), we need to use some range minimum query data structure.

Thus, we assume that our suffix tree is enhanced to answer LCA queries in constant time. This can be done in linear time and space. We denote by $\mathrm{LCA}(i,j)$ the query that returns in $O(1)$ time the length of the common prefix between the suffix u_i and the suffix u_j of u.

For every index i, we try to find if the suffix of index i of the word u has k mismatches with its prefix of the same length. We first compute $\mathrm{LCA}(0, i)$. Let this length be ℓ_0. We skip the mismatching character in u_0 and u_i and try to find $\mathrm{LCA}(\ell_0 + 1, i + \ell_0 + 1)$. We repeat this to obtain k mismatches between u_i and $u[0] \cdots u[n - i - 1]$ or fail to obtain this condition.

The pseudo code of the technique is given in Algorithm 1. We maintain a variable ℓ which gives, after the line 4 of Algorithm 1, the index of the current mismatch between u_i and u_0. A variable d contains the current Hamming distance between $u[i]\cdots u[i+\ell-1]$ and $u[0]\cdots u[\ell-1]$. It is increased by 1 at the line 8 since a mismatch has been found.

Since there are at most $O(k)$ LCA queries for each index i, this can be done in $O(k)$ time. The overall time complexity is thus $O(kn)$ and the space complexity is $O(n)$.

We now show a second method to check whether a word of length n has a k-error border. We use the computation of a *k-prefix table* as done in [5]. For each position i, we compute a table π_k for which $\pi_k(i)$ is the length ℓ of the longest word $u[i]\ldots u[i+\ell-1]$ such that the Hamming distance between $u[0]\ldots u[\ell-1]$ and $u[i]\ldots u[i+\ell-1]$ is at most k and $u[0]\ldots u[\ell-1]$ is proper prefix of u.

This computation can be done in time $O(kn)$ and space $O(n)$ (see [5, Theorem 5]). It needs the computations of the prefix array of u and the longest common prefix array preprocessed for range minimum queries. The longest common prefix array gives for each index r the length of the longest common prefix of the rth suffix and the $(r-1)$th suffix in lexicographic order.

The existence of a k-error border is then obtained as follows. For $k \geq 1$, a word u has a k-error border if and only if there is a position i, $1 \leq i < n$, for which $\pi_k[i] = n - i$ and $\pi_{k-1}[i] < n - i$. Indeed such a position i exists if and only if there is a proper suffix $u[i]\ldots u[n-1]$ of u whose Hamming distance with $u[0]\ldots u[n-i-1]$ is exactly k. The existence of a k-error border is thus obtained with Algorithm 2 which is in $O(n)$ time. The overall time complexity is again in $O(kn)$ and the space complexity is $O(n)$. ∎

Algorithm 1: WORD WITH A k-ERROR BORDER(u)

Input: A non empty word u of length n, a non-negative integer k
Output: true if u has a k-error border

```
1  for i ← 1 to n − 1 do
2      (ℓ, d) ← (0, 0);
3      while d ≤ k do
4          ℓ ← ℓ + LCA(ℓ, i + ℓ);
5          if d = k and ℓ = n − i then
6              return TRUE;
7          if d < k and ℓ < n − i then
8              (ℓ, d) ← (ℓ + 1, d + 1);
9          else
10             BREAK
11 return FALSE;
```

Example 6 Let $u = 101011$. Let us check with Algorithm 1 whether u has a

2-error border. For $i = 1$, at the first step of the loop of the line 3 we obtain at the line 4 $\ell = \mathrm{LCA}(0, 1) = 0$; we set ℓ to 1 (the jump) and d to 1 at the line 8. At the second step of the loop of the line 3 we obtain at the line 4 $\ell = \ell + \mathrm{LCA}(1, 2) = 1$; we set ℓ to 2 and d to 2 at the line 8. At the third step of the loop of the line 3, we obtain at the line 4 $\ell = \ell + \mathrm{LCA}(2, 3) = 2$ and break at the line 10. For $i = 2$ the loop of the line 3 fails to return true. For $i = 3$, at the first step of the loop of the line 3 we obtain at the line 4 $\ell = \mathrm{LCA}(0, 3) = 0$; we set ℓ to 1 and d to 1 at the line 8. At the second step of the loop of the line 3 we obtain at the line 4 $\ell = \ell + \mathrm{LCA}(1, 4) = 1$; we set ℓ to 2 and d to 2 at the line 8. At the third step of the loop of the line 3, we obtain the at line 4 $\ell = \ell + \mathrm{LCA}(2, 5) = 3$ and, since $d = 2$ and $\ell = n - i = 3$, the algorithm returns true at the line 6. The algorithm has thus detected the 2-error border of length 3.

Algorithm 2: WORD WITH A k-ERROR BORDER(u)

Input: A non empty word u of length n, a non-negative integer k, the
k-prefix table π_k and the $(k-1)$-prefix table π_{k-1}
Output: true if u has a k-error border

1 for $i \leftarrow 1$ **to** $n - 1$ **do**
2   **if** $\pi_k[i] = n - i$ *and* $\pi_{k-1}[i] < n - i$ **then**
3   **return** TRUE;
4 return FALSE;

The following corollary follows then directly from Proposition 3 and the analysis of Algorithm 1 in Proposition 5.

Corollary 7 *It can be checked in linear time and space whether a word is Hamming-isometric.*

4   Algorithm for checking whether a word over an alphabet of size 4 is Lee-isometric

A combinatorial characterization of Lee-isometric words over an alphabet of size 4 has been obtained in [2]. It uses the notion of Lee distance which is defined as follows. The *Lee distance*, denoted by d_L, between two letters of the alphabet $\mathbb{Z}_d = \{0, 1, \ldots, d - 1\}$ is

$$d_L(a, b) = \min(|a - b|, d - |a - b|).$$

The Lee distance between two words u and v of length n over $\mathbb{Z}_d$ is

$$d_L(u, v) = \sum_{i=0}^{n-1} d_L(u[i], v[i]).$$

A word has a *k-Lee-error border* if it has a suffix u and a prefix v of same length satisfying $d_L(u,v) = k$.

For words over $\mathbb{Z}_4$, the Lee-isometric words are characterized as follows in [2].

Proposition 8 *A word over a 4-letter alphabet is non-Lee-isometric if and only if it has a 2-Lee-error border.*

In this section we show that checking if a word of length n has a k-Lee-error border can be done in time $O(kn)$ and space $O(n)$. The algorithm is Algorithm 3.

Algorithm 3: WORD WITH A k-LEE-ERROR BORDER(u)

Input: A non empty word u of length n, a non-negative integer k
Output: true if u has a k-Lee-error border

1 **for** $i \leftarrow 1$ **to** $n - 1$ **do**
2      $(\ell, d) \leftarrow (0, 0)$;
3   **while** $d \leq k$ **do**
4          $\ell \leftarrow \ell + \text{LCA}(\ell, i + \ell)$;
5   **if** $d = k$ *and* $\ell = n - i$ **then**
6   **return** TRUE;
7   **if** $d = k$ *and* $\ell < n - i$ **then**
8   BREAK
9   **if** $d < k$ *and* $\ell = n - i$ **then**
10   BREAK
11          $d \leftarrow d + d_L(u[\ell], u[i + \ell])$;
12          $\ell \leftarrow \ell + 1$;
13 **return** FALSE;

Proposition 9 *It can be checked in time $O(kn)$ and space $O(n)$ whether a word of length n has a k-Lee-error border.*

Proof. The algorithm is almost the same as Algorithm 1 and the proof is similar to the proof of Proposition 5. Therefore we only discuss the differences.

For every index i, we try to find if the suffix of index i is at Lee distance k from its prefix of the same length.

The pseudo code of the technique is given in Algorithm 3. A variable d contains, after the line 11, the current Lee distance between $u[i] \cdots u[i + \ell]$ and $u[0] \cdots u[\ell]$.

The difference with Algorithm 1 appears when there is mismatch between the suffix u_i and the prefix $u[0] \cdots u[n - i - 1]$ at positions ℓ on u_i and $i + \ell$ on $u[0] \cdots u[n - i - 1]$. The current Lee distance between $u[0] \cdots u[\ell]$ and $u[i] \cdots u[i + \ell]$ is augmented this time by the value of $d_L(u[\ell], u[i + \ell])$.

The case $d < k$ and $\ell = n - i$ of the line 9 of Algorithm 3 corresponds to the case where the suffix of u at position i is a d-Lee-error border with $d < k$ and is thus not a solution. The algorithms continues then to check the position $i + 1$.

Since there are at most $O(k)$ LCA queries for each index i, this can be done in $O(k)$ time. The overall time complexity is thus $O(kn)$ and the space complexity is $O(n)$.

∎

The following corollary follows then directly from Proposition 8 and the analysis of Algorithm 2 in Proposition 9.

Corollary 10 *It can be checked in linear time and space whether a word over an alphabet of size 4 is Lee-isometric.*

5   Acknowledgment

References

[1] Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004.

[2] Marcella Anselmo, Manuela Flores, and Maria Madonia. Quaternary n-cubes and isometric words. In *Combinatorics on Words - 13th International Conference, WORDS 2021, Rouen, France, September 13-17, 2021, Proceedings*, volume 12847 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2021.

[3] Marcella Anselmo, Dora Giammarresi, Maria Madonia, and Carla Selmi. Bad pictures: Some structural properties related to overlaps. In Galina Jirásková and Giovanni Pighizzini, editors, *Descriptional Complexity of Formal Systems - 22nd International Conference, DCFS 2020, Vienna, Austria, August 24-26, 2020, Proceedings*, volume 12442 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 2020.

[4] Alberto Apostolico, Maxime Crochemore, Martin Farach-Colton, Zvi Galil, and S. Muthukrishnan. 40 years of suffix trees. *Commun. ACM*, 59(4):66–73, 2016.

[5] Carl Barton, Costas S. Iliopoulos, Solon P. Pissis, and William F. Smyth. Fast and simple computations using prefix tables under Hamming and edit distance. In Jan Kratochvíl, Mirka Miller, and Dalibor Froncek, editors, *Combinatorial Algorithms - 25th International Workshop, IWOCA 2014, Duluth, MN, USA, October 15-17, 2014, Revised Selected Papers*, volume 8986 of *Lecture Notes in Computer Science*, pages 49–61. Springer, 2014.

[6] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on Sstrings*. Cambridge University Press, 2007.

[7] Z Galil and R Giancarlo. Improved string matching with k mismatches. *SIGACT News*, 17(4):52–54, March 1986.

[8] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

[9] W.-J. Hsu. Fibonacci cubes-a new interconnection topology. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):3–12, 1993.

[10] Aleksandar Ilic, Sandi Klavzar, and Yoomi Rho. Generalized Fibonacci cubes. *Discret. Math.*, 312(1):2–11, 2012.

[11] Sandi Klavzar. Structure of Fibonacci cubes: a survey. *J. Comb. Optim.*, 25(4):505–522, 2013.

[12] Sandi Klavzar and Sergey V. Shpectorov. Asymptotic number of isometric generalized Fibonacci cubes. *Eur. J. Comb.*, 33(2):220–226, 2012.

[13] Gad M. Landau and Uzi Vishkin. Efficient string matching in the presence of errors. In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 126–136. IEEE Computer Society, 1985.

[14] Gonzalo Navarro and Mathieu Raffinot. *Flexible Pattern Matching in Strings - practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002.

[15] Marius Nicolae and Sanguthevar Rajasekaran. On pattern matching with k mismatches and few don't cares. *Inf. Process. Lett.*, 118:78–82, 2017.

[16] Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. Comput.*, 17(6):1253–1262, 1988.

[17] Jianxin Wei. The structures of bad words. *Eur. J. Comb.*, 59:204–214, 2017.

[18] Jianxin Wei, Yujun Yang, and Xuena Zhu. A characterization of non-isometric binary words. *Eur. J. Comb.*, 78:121–133, 2019.

[19] Jianxin Wei and Heping Zhang. Proofs of two conjectures on generalized Fibonacci cubes. *Eur. J. Comb.*, 51:419–432, 2016.