Detecting k-(Sub-)Cadences and Equidistant Subsequence Occurrences

Mitsuru Funakoshi
Department of Informatics, Kyushu University, Fukuoka, Japan
mitsuru.funakoshi@inf.kyushu-u.ac.jp

Yuto Nakashima
Department of Informatics, Kyushu University, Fukuoka, Japan
yuto.nakashima@inf.kyushu-u.ac.jp

Shunsuke Inenaga
Department of Informatics, Kyushu University, Fukuoka, Japan
PRESTO, Japan Science and Technology Agency, Kawaguchi, Japan
inenaga@inf.kyushu-u.ac.jp

Hideo Bannai
M&D Data Science Center, Tokyo Medical and Dental University, Tokyo, Japan
hdbn.dsc@tmd.ac.jp

Masayuki Takeda
Department of Informatics, Kyushu University, Fukuoka, Japan
takeda@inf.kyushu-u.ac.jp

Ayumi Shinohara
Graduate School of Information Sciences, Tohoku University, Sendai, Japan
ayumis@tohoku.ac.jp

Abstract

The equidistant subsequence pattern matching problem is considered. Given a pattern string P and a text string T, we say that P is an *equidistant subsequence* of T if P is a subsequence of the text such that consecutive symbols of P in the occurrence are equally spaced. We can consider the problem of equidistant subsequences as generalizations of (sub-)cadences. We give bit-parallel algorithms that yield $o(n^2)$ time algorithms for finding k-(sub-)cadences and equidistant subsequences. Furthermore, $O(n \log^2 n)$ and $O(n \log n)$ time algorithms, respectively for equidistant and Abelian equidistant matching for the case $|P| = 3$, are shown. The algorithms make use of a technique that was recently introduced which can efficiently compute convolutions with linear constraints.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms

Keywords and phrases string algorithms, pattern matching, bit parallelism, subsequences, cadences

Digital Object Identifier 10.4230/LIPIcs.CPM.2020.12

Funding *Mitsuru Funakoshi*: JSPS KAKENHI Grant Number JP20J21147.
Yuto Nakashima: JSPS KAKENHI Grant Number JP18K18002.
Shunsuke Inenaga: JSPS KAKENHI Grant Number JP17H01697, JST PRESTO Grant Number JPMJPR1922.
Hideo Bannai: JSPS KAKENHI Grant Numbers JP16H02783, JP20H04141.
Masayuki Takeda: JSPS KAKENHI Grant Number JP18H04098.
Ayumi Shinohara: JSPS KAKENHI Grant Number JP15H05706.

31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020).
Editors: Inge Li Gørtz and Oren Weimann; Article No. 12; pp. 12:1–12:11
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Pattern matching on strings is a very important topic in string processing. Usually, strings are regarded and stored as one dimensional sequences and many pattern matching algorithms have been proposed to efficiently find particular substrings occurring in them [9, 2, 4, 8, 6, 3]. However, when one is to view the string/text data on paper or on a screen, it is usually shown in two dimensions: the single dimensional sequence is displayed in several lines folded by some length. It is known that the two dimensional arrangement can be used to embed hidden messages, and/or cause occurrences of unexpected or unintentional messages in the text. A common form for such an embedding is to consider the occurrence of a pattern in a linear layout: vertically or possibly diagonally along the two dimensional display.

For example, there was a (rather controversial) paper [12] on the so called Bible Code, claiming that the Bible contains statistically significant occurrences of various related words, occurring vertically and/or diagonally, in close proximity. Furthermore, there was an incident with a veto letter by the California State Governor [11]; Although it was considered a "weird coincidence", the first character on each line of the letter could be connected and interpreted as a very provocative message. In Japanese internet forums, there was a culture of actively using these techniques, referred to as "tate-yomi"(vertical reading) and "naname-yomi" (diagonal reading), where the author of a message purposely embeds a hidden message in his/her post. Most commonly, the author will write a message that praises some object or opinion in question, but embed a message with a completely opposite meaning bearing the author's true intention. The hidden message can be recovered by reading the text message vertically or diagonally from some position, and is used as form of sarcasm, as well as a clever method to mock those who were unable to *get* it.

Assuming that the text is folded into lines of equal length, vertical or diagonal occurrences of the pattern in two dimensions can be regarded as a subsequence of the original text, where the distance between each character is equal. We call the problem of detecting such occurrences of the pattern as the *equidistant subsequence matching* problem. To the best of the authors' knowledge, there exist only publications concerning the statistical properties of the occurrence of equidistant subsequence patterns, mainly with the so called Bible Code.

Recently, a notion of regularities in strings called *(sub)-cadences*, defined by equidistant occurrences of the same character, was considered by Amir et al. [1]. A k-sub-cadence of a string can be viewed as an occurrence of an equidistant subsequence of length k that consists of the same character. A k-sub-cadence is a k-cadence, if the starting position is less than or equal to d and the ending position is greater than $n - d$, where d is the distance between each consecutive character occurrence and n is the length of the string. To date, algorithms for detecting anchored cadences (cadences whose starting position is equal to d), 3-(sub-)cadences, and (π_1, π_2, π_3)-partial-3-cadences (an occurrence of an equidistant subsequence that can become a cadence by changing a character at most all but three positions $i + \pi_1 d$, $i + \pi_2 d$, and $i + \pi_3 d$, where i is the starting position of the equidistant subsequence.) have been proposed [1, 5]. However, no efficient algorithm for detecting k-(sub)-cadences for arbitrary k ($1 \leq k \leq n$) is known so far.

In this paper, we present counting algorithms for k-sub-cadences, k-cadences, equidistant subsequence patterns of length m and length 3, and equidistant Abelian subsequence patterns of length 3. Table 1 shows a summary of the results. All algorithms run in $O(n)$ space. Furthermore, we present locating algorithms for k-sub-cadences, k-cadences, and equidistant subsequence patterns of length m. The time complexities of these algorithms can be obtained by adding *occ* to the second term inside the minimum function of each time complexity of the counting algorithm. To the best of the authors' knowledge, these are the first $o(n^2)$ time algorithms for k-(sub)-cadences and equidistant subsequence patterns. In this paper, we assume a word RAM model with word size $\Theta(\log n)$. Also, unless otherwise noted, we assume that strings over a general ordered alphabet.

Table 1 Summary of results. Note that an equidistant Abelian subsequence pattern is an equidistant subsequence of any permutation of a given pattern.

Counting time	For a constant size alphabet	For a general ordered alphabet
k-sub-cadences	$O\left(\min\left\{\frac{n^2}{k}, \frac{n^2}{\log n}\right\}\right)$	$O\left(\min\left\{\frac{n^2}{k}, \frac{n^2\sqrt{k}}{\sqrt{\log n}}\right\}\right)$
k-cadences	$O\left(\frac{n^2}{k^2 \log n}\right)$	$O\left(\min\left\{\frac{n^2}{k^2}, \frac{n^2}{\sqrt{k}\sqrt{\log n}}\right\}\right)$

Counting time	For a general ordered alphabet
Equidistant subsequence pattern	$O\left(\min\left\{\frac{n^2}{m}, \frac{n^2}{\log n}\right\}\right)$
Equidistant subsequence pattern of length three	$O(n \log^2 n)$
Equidistant Abelian subsequence pattern of length three	$O(n \log n)$

2 Preliminaries

Let Σ be the *alphabet*. An element of Σ^* is called a *string*. The length of a string T is denoted by $|T|$. String $s \in \Sigma^*$ is said to be a *subsequence* of string $T \in \Sigma^*$ if s can be obtained by removing zero or more characters from T.

For a string T and an integer $1 \le i \le |T|$, $T[i]$ denotes the i-th character of T. For two integers $1 \le i \le j \le |T|$, $T[i..j]$ denotes the substring of T that begins at position i and ends at position j. For convenience, let $T[i..j] = \varepsilon$ when $i > j$.

2.1 k-(Sub-)Cadences

The term "cadence" has been used in slightly different ways in the literature (e.g., see [7, 10, 1]). In this paper, we use the definitions of cadences and sub-cadences which are used in [1] and [5].

For integers i and d, the pair (i, d) is called a k-*sub-cadence* of $T \in \Sigma^n$ if $T[i] = T[i+d] = T[i+2d] = \cdots = T[i+(k-1)d]$, where $1 \le i \le n$ and $1 \le d \le \lfloor \frac{n-i}{k-1} \rfloor$. The set of k-sub-cadences of T can be defined as follows:

▶ **Definition 1.** *For $T \in \Sigma^n$, $n \in \mathcal{N}$, and $k \in [1..n]$,*

$$KSC(T,k) = \left\{ (i,d) \;\middle|\; \begin{array}{l} T[i] = T[i+d] = T[i+2d] = \cdots = T[i+(k-1)d] \\ 1 \le i \le n, 1 \le d \le \lfloor \frac{n-i}{k-1} \rfloor \end{array} \right\}.$$

For integers i and d, the pair (i, d) is called a k-*cadence* of $T \in \Sigma^n$ if (i, d) is a k-sub-cadence and satisfies the inequalities $i - d \le 0$ and $n < i + kd$. The set of k-cadences of T can be defined as follows:

▶ **Definition 2.** *For $T \in \Sigma^n$, $n \in \mathcal{N}$, and $k \in [1..n]$,*

$$KC(T,k) = \left\{ (i,d) \;\middle|\; \begin{array}{l} T[i] = T[i+d] = T[i+2d] = \cdots = T[i+(k-1)d] \\ 1 \le i \le d, \frac{n-i}{k} < d \le \lfloor \frac{n-i}{k-1} \rfloor \end{array} \right\}.$$

2.2 Equidistant Subsequence Occurrences

For integers i and d, we say that pair (i, d) is an *equidistant subsequence occurrence* of $P \in \Sigma^m$ in $T \in \Sigma^n$ if $P = T[i] \cdot T[i+d] \cdot T[i+2d] \cdots T[i+(m-1)d]$, where $1 \leq i \leq n$ and $1 \leq d \leq \lfloor \frac{n-i}{m-1} \rfloor$. The set of equidistant subsequence occurrences of P in T can be defined as follows:

▶ **Definition 3.** *For $T \in \Sigma^n, P \in \Sigma^m$ and $n, m \in \mathcal{N}$,*

$$ESP(T,P) = \left\{ (i,d) \;\middle|\; \begin{array}{l} P = T[i] \cdot T[i+d] \cdot T[i+2d] \cdots T[i+(m-1)d] \\ 1 \leq i \leq n, 1 \leq d \leq \lfloor \frac{n-i}{m-1} \rfloor \end{array} \right\}.$$

2.3 Equidistant Abelian Subsequence Occurrences

Two strings S_1 and S_2 are said to be *Abelian equivalent* if S_1 is a permutation of S_2, or vice versa. Now for integers i and d, we say that pair (i, d) is an *equidistant Abelian subsequence occurrence* of $P \in \Sigma^m$ in $T \in \Sigma^n$ if $T[i] \cdot T[i+d] \cdot T[i+2d] \cdots T[i+(m-1)d]$ and P are Abelian equivalent, where $1 \leq i \leq n$ and $1 \leq d \leq \lfloor \frac{n-i}{m-1} \rfloor$. The set of equidistant Abelian subsequence occurrences of P in T can be defined as follows:

▶ **Definition 4.** *For $T \in \Sigma^n, P \in \Sigma^m$ and $n, m \in \mathcal{N}$,*

$$EASP(T,P) = \left\{ (i,d) \;\middle|\; \begin{array}{l} T[i] \cdot T[i+d] \cdots T[i+(m-1)d] \text{ and } P \text{ are Abelian equivalent} \\ 1 \leq i \leq n, 1 \leq d \leq \lfloor \frac{n-i}{m-1} \rfloor \end{array} \right\}.$$

When it is clear from the context, we denote $KSC(T,k)$ as KSC, $KC(T,k)$ as KC, and $ESP(T,P)$ as ESP.

3 Detecting k-Sub-Cadences

In this section, we consider algorithms for detecting k-sub-cadences.

3.1 Algorithm 1

One of the simplest methods is as follows: For each distance d with $1 \leq d \leq \lfloor \frac{n-1}{k-1} \rfloor$, we construct text $ST_d = T[1] \cdot T[1+d] \cdots T[1+d\lfloor \frac{n-1}{d} \rfloor] \cdot \$ \cdot T[2] \cdot T[2+d] \cdots T[2+d\lfloor \frac{n-2}{d} \rfloor] \cdot \$ \cdots T[d] \cdot T[2d] \cdots T[d\lfloor \frac{n}{d} \rfloor]$ of length $d\lfloor \frac{n}{d} \rfloor + d - 1$. Then, the strings $T[1] \cdot T[1+d] \cdots T[1+d\lfloor \frac{n-1}{d} \rfloor], T[2] \cdot T[2+d] \cdots T[2+d\lfloor \frac{n-2}{d} \rfloor], ..., T[d] \cdot T[2d] \cdots T[d\lfloor \frac{n}{d} \rfloor]$ are called *d-skip-strings*, and the ST_d is called *d-split text*. If we would like to find k-sub-cadences with distance d in text T, we find concatenations of the same character of length k as substrings in ST_d.

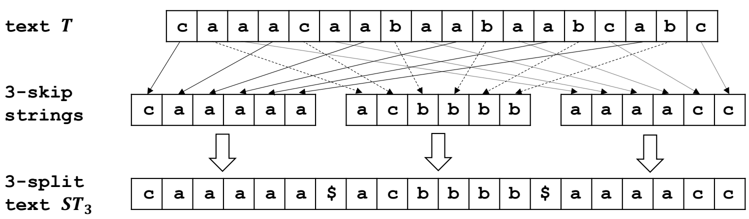

Figure 1 Preprocessing for Algorithm 1.

Fig. 1 is an example of the 3-split text ST_3. In ST_d, we use a symbol $\$ \notin \Sigma$ in order to prevent detecting false occurrences of concatenation of same character of the length k across the ends of d-skip strings as a k-sub-cadence. The text obtained by concatenating all ST_d for all $1 \leq d \leq \lfloor \frac{n-1}{k-1} \rfloor$ and \$ is called the *split text*. If we prepare the split text, we can compute KSC simply by checking that the same character is repeated k times.

The length of ST_d is at most $n+d$ including \$. The maximum value of d is $\lfloor \frac{n-1}{k-1} \rfloor$, and therefore, the number of ST_d is at most $\lfloor \frac{n-1}{k-1} \rfloor$. Hence, the length of the split text of T is $O(\frac{n^2}{k})$. We can check that the same character is repeated k times in the split text in $O(\frac{n^2}{k})$ time. Although we have presented the split text to ease the description, it does not have to be constructed explicitly.

From the above, we can get the following result.

▶ **Theorem 5.** *There is an algorithm for locating all k-sub-cadences for given k $(1 \leq k \leq n)$ which uses $O\left(\frac{n^2}{k}\right)$ time and $O(n)$ space.*

As can be seen from the example of $T = \mathtt{a}^n$, $|KSC|$ can be $\Omega(\frac{n^2}{k})$. Therefore, when we locate all $(i, d) \in KSC$, this algorithm is optimal in the worst case. In the next subsection, we show a counting algorithm that is efficient when the value of k is small. Moreover, we show a locating algorithm that is efficient when both the value of k and $|KSC|$ is small.

3.2 Algorithm 2

In this subsection, we will show the following result:

▶ **Theorem 6.** *For a constant size alphabet, there is an $O\left(\frac{n^2}{\log n}\right)$ time algorithm for counting all k-sub-cadences for given k. We can also locate these occurrences in $O\left(\frac{n^2}{\log n} + occ\right)$ time, where occ is the number of the outputs. For a general ordered alphabet, there is an $O\left(\frac{n^2\sqrt{k}}{\sqrt{\log n}}\right)$ time algorithm for counting all k-sub-cadences for given k. We can also locate these occurrences in $O\left(\frac{n^2\sqrt{k}}{\sqrt{\log n}} + occ\right)$ time. These algorithms run in $O(n)$ space.*

Note that for counting all k-sub-cadences, for a constant size alphabet (resp. for a general ordered alphabet), this algorithm is faster than Algorithm 1 if k is $o(\log n)$ (resp. $o\left(\sqrt[3]{\log n}\right)$). For locating all k-sub-cadences, for a constant size alphabet (resp. for a general ordered alphabet), if $|KSC|$ is $o(\frac{n^2}{k})$ and k is $o(\log n)$ (resp. $o\left(\sqrt[3]{\log n}\right)$), then this algorithm is faster.

Now we will show how to count all k-sub-cadences of character $c \in \Sigma$. Let $\delta_c[1..n]$ be a binary sequence given by the indicator function for the character c:

$$\delta_c[i] := \begin{cases} 1 & \text{if } T[i] = c, \\ 0 & \text{if } T[i] \neq c. \end{cases}$$

If (i, d) is a k-sub-cadence with character c, $\delta_c[i] = \delta_c[i+d] = \cdots = \delta_c[i+(k-1)d] = 1$. Therefore, we can check whether (i, d) is a k-sub-cadence or not by computing $\delta_c[i] \cdot \delta_c[i+d] \cdots \delta_c[i+(k-1)d]$. To compute this, we use bit-parallelism, i.e, the bit-wise operations AND and SHIFT_LEFT, denoted by `&` and `<<`, respectively, as in the C language. For each d with $1 \leq d \leq \lfloor \frac{n-1}{k-1} \rfloor$, let $Q_d = \delta_c \mathbin{\&} (\delta_c \ll d) \mathbin{\&} (\delta_c \ll 2d) \mathbin{\&} \cdots \mathbin{\&} (\delta_c \ll (k-1)d)$. If $Q_d[i] = 1$, then (i, d) is a k-sub-cadence. See Figure 2 for a concrete example.

If we want to count all k-sub-cadences with d, we only have to count the number of 1's in Q_d. If we want to locate all k-sub-cadences with d, we have to locate all 1's in Q_d.

In the word RAM model, SHIFT_LEFT and AND operations can be done in constant time per operation on bit sequences of length $O(\log n)$. Since δ_c is a binary sequence of length n, one SHIFT_LEFT or AND operation can be done in $O(\frac{n}{\log n})$ time. Therefore, Q_d

$$
\begin{array}{rcl}
\boldsymbol{T} & = & \texttt{c a a a c a a b a a b a a b c a b c} \\
\boldsymbol{\delta}_{\mathbf{a}} & = & \texttt{0 1 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 0} \\
\\
\boldsymbol{\delta}_{\mathbf{a}} & = & \texttt{0 1 1 1 0 1 1 0 1 1 0 1 1 0 0 1 0 0} \\
(\boldsymbol{\delta}_{\mathbf{a}} \ll \mathbf{3}) & = & \texttt{1 0 1 1 0 1 1 0 1 1 0 0 1 0 0 0 0 0} \\
(\boldsymbol{\delta}_{\mathbf{a}} \ll \mathbf{6}) & = & \texttt{1 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0} \\
(\boldsymbol{\delta}_{\mathbf{a}} \ll \mathbf{9}) & = & \texttt{1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0} \\
\hline
\boldsymbol{Q}_{\mathbf{3}} & = & \texttt{0 0 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0}
\end{array}
$$

Figure 2 Let $T = \texttt{caaacaabaabaabcabc}$. $(3,3)$, $(4,3)$, and $(7,3)$ are 4-sub-cadences of character 'a' with $d = 3$.

can be obtained in $O(k\frac{n}{\log n})$ time. Since it is known that the number of 1's in a bit sequence of length $O(\log n)$ can be obtained in $O(1)$ time by using the "popcnt" operation which is a standard operation on the word RAM model, the number of 1's in Q_d can be counted in $O(\frac{n}{\log n})$ time. Hence, for all $1 \leq d \leq \lfloor\frac{n-1}{k-1}\rfloor$, we can count all k-sub-cadences of character c in $O\left(k\frac{n}{\log n}\lfloor\frac{n-1}{k-1}\rfloor + \frac{n}{\log n}\lfloor\frac{n-1}{k-1}\rfloor\right) \subseteq O(\frac{n^2}{\log n})$ time. Also, it is known that the position of the rightmost 1 (the least significant set bit) in a bit sequence of length $O(\log n)$ can be answered in constant time by using bit-wise operations. We split Q_d into $O(\frac{n}{\log n})$ blocks of length $O(\log n)$. For each block, the least significant set bit can be found in $O(1)$ time if the block contains at least one 1. After finding the least significant set bit, we mask this bit to 0 and do the above operation again. Bit mask operation can be done in $O(1)$ time. Hence, we can answer all the positions of 1's in Q_d in $O(\frac{n}{\log n} + occ)$ time. Therefore, we can locate all k-sub-cadences of character c in $O(\frac{n^2}{\log n} + occ)$ time.

We showed how to detect all k-sub-cadences of character c, so we can detect all k-sub-cadences by doing the above operations for each character in Σ. For a constant size alphabet, since we only do the above operations a constant number of times, we can count all k-sub-cadences in $O(\frac{n^2}{\log n})$ time. We can also locate these occurrences in $O(\frac{n^2}{\log n} + occ)$ time. However, for a general ordered alphabet, we have to do the above operations $|\Sigma|$ times.

For a general ordered alphabet, if the number of occurrences of the character is small, we use another algorithm that generalizes Amir et al.'s algorithm [1] for detecting 3-cadences to k-sub-cadences: Let N_c be the set of positions which are occurrences of a character c. If we pick two positions in N_c and regard the smaller one as the starting position i of k-sub-cadences and the larger one as the second position $i + d$ of a k-sub-cadence, then the distance d is uniquely determined. We can check whether the pair (i, d) is a k-sub-cadence or not in $O(k)$ time. Since the number of pairs is at most $|N_c|^2$, we can count or locate k-sub-cadences of character c in $O(k|N_c|^2)$ time.

Thus, for a general ordered alphabet, all k-sub-cadences can be counted in $O(\sum_{c\in\Sigma}\min\{k|N_c|^2, \frac{n^2}{\log n}\})$ time. Since $O(\sum_{c\in\Sigma}\min\{k|N_c|^2, \frac{n^2}{\log n}\})$ is maximized when $k|N_c|^2 = \frac{n^2}{\log n}$, then $O(\sum_{c\in\Sigma}\min\{k|N_c|^2, \frac{n^2}{\log n}\}) \subseteq O((\sum_{c\in\Sigma}|N_c|)\frac{n\sqrt{k}}{\sqrt{\log n}}) \subseteq O(\frac{n^2\sqrt{k}}{\sqrt{\log n}})$. Therefore we can count in $O(\frac{n^2\sqrt{k}}{\sqrt{\log n}})$ time by using Algorithm 2 and the above algorithm that generalizes Amir et al.'s algorithm. Also, all k-sub-cadences can be located in $O(\sum_{c\in\Sigma}\min\{k|N_c|^2, \frac{n^2}{\log n} + occ_c\})$ time where occ_c is the number of k-sub-cadences of character c. Since $O(\sum_{c\in\Sigma}\min\{k|N_c|^2, \frac{n^2}{\log n} + occ_c\}) \subseteq O((\sum_{c\in\Sigma}\min\{k|N_c|^2, \frac{n^2}{\log n}\}) + occ) \subseteq O(\frac{n^2\sqrt{k}}{\sqrt{\log n}} + occ)$, we can locate all k-sub-cadences in $O(\frac{n^2\sqrt{k}}{\sqrt{\log n}} + occ)$ time.

From the above, we obtain the following result:

▶ **Theorem 7.** *For a constant size alphabet (resp. for a general ordered alphabet), all k-sub-cadences with given k can be counted in $O\left(\min\left\{\frac{n^2}{k}, \frac{n^2}{\log n}\right\}\right)$ time (resp. $O\left(\min\left\{\frac{n^2}{k}, \frac{n^2\sqrt{k}}{\sqrt{\log n}}\right\}\right)$ time) and $O(n)$ space, and can be located in $O\left(\min\left\{\frac{n^2}{k}, \frac{n^2}{\log n} + occ\right\}\right)$ time (resp. $O\left(\min\left\{\frac{n^2}{k}, \frac{n^2\sqrt{k}}{\sqrt{\log n}} + occ\right\}\right)$ time) and $O(n)$ space.*

4 Detecting k-Cadences

In this section, we consider algorithms for detecting k-cadences.

4.1 Algorithm 3

Again, each (i, d) has to satisfy the following formulas: $1 \le i \le d$ and $\frac{n-i}{k} < d \le \lfloor\frac{n-i}{k-1}\rfloor$. Then, each distance d satisfies $\frac{n}{k+1} < d < \frac{n}{k-1}$. We use same techniques of Algorithm 1 for each d with $\frac{n}{k+1} < d < \frac{n}{k-1}$. Since the number of possible values for d is $O(\frac{n}{k^2})$, we can check that the same character is repeated k times in the split text in $O(\frac{n^2}{k^2})$ time. Therefore, we can obtain the following result:

▶ **Theorem 8.** *There is an algorithm for locating all k-cadences for given k which uses $O\left(\frac{n^2}{k^2}\right)$ time and $O(n)$ space.*

4.2 Algorithm 4

Now, we will show the following result:

▶ **Theorem 9.** *For a constant size alphabet, there is an $O\left(\frac{n^2}{k^2\log n}\right)$ time algorithm for counting all k-cadences for given k. We can also locate these occurrences in $O\left(\frac{n^2}{k^2\log n} + occ\right)$ time. For a general ordered alphabet, there is an $O\left(\frac{n^2}{\sqrt{k}\sqrt{\log n}}\right)$ time algorithm for counting all k-sub-cadences for given k. We can also locate these occurrences in $O\left(\frac{n^2}{\sqrt{k}\sqrt{\log n}} + occ\right)$ time. These algorithms run in $O(n)$ space.*

Note that when we count all k-sub-cadences, for a constant size alphabet, this algorithm is faster than Algorithm 3. Also, for a general ordered alphabet, this algorithm is faster if k is $o(\sqrt[3]{\log n})$. (This is because if $\frac{n^2}{\sqrt{k}\sqrt{\log n}}$ is less than $\frac{n^2}{k^2}$, then $k\sqrt{k} \le \sqrt{\log n}$.) When we locate all k-sub-cadences, for a constant size alphabet (resp. for a general ordered alphabet), if $|KC|$ is $o(\frac{n^2}{k^2})$ (resp. $|KC|$ is $o(\frac{n^2}{k^2})$ and k is $o(\sqrt[3]{\log n})$) then this algorithm is faster.

Now we will show how to count all k-cadences of character $c \in \Sigma$. If (i, d) is a k-cadence with character c, then $\delta_c[i] = \delta_c[i+d] = \cdots = \delta_c[i+(k-1)d] = 1$, $1 \le i \le d$, and $\frac{n-i}{k} < d \le \lfloor\frac{n-i}{k-1}\rfloor$. Therefore, to calculate k-cadences, we need only the range $[1..d]$ of i for d with $\frac{n-i}{k} < d \le \lfloor\frac{n-i}{k-1}\rfloor$. For each d with $\frac{n-i}{k} < d \le \lfloor\frac{n-i}{k-1}\rfloor$, let $Q'_d = \delta_c[1..d]\ \&\ \delta_c[d+1..2d]\ \&\ \delta_c[3d+1..4d]\ \&\ \cdots\ \&\ \delta_c[(k-1)d+1..kd]$. If $Q'_d[i] = 1$, (i, d) is a k-cadence. Q'_d can be obtained by the following operation: $Q'_d = \delta_c[1..d]\ \&\ (\delta_c \ll d)[1..d]\ \&\ (\delta_c \ll 2d)[1..d]\ \&\ \cdots\ \&\ (\delta_c \ll (k-1)d)[1..d]$. By using the same techniques of Algorithm 2, we can compute Q'_d in $O(k\frac{d}{\log n})$ time. Hence, for all $\frac{n}{k+1} < d < \frac{n}{k-1}$, we can count all k-cadences of a character in $O(k\frac{d}{\log n}\frac{n}{k^2}) \subseteq O(\log n \frac{n^2}{k^2})$ time.

For a locating algorithm and for a general ordered alphabet, we can use same techniques of the above section. Then we can locate all k-cadences of a character in $O(\frac{n^2}{k^2 \log n} + occ)$ time. For a general ordered alphabet, all k-cadences can be counted in $O(\sum_{c\in\Sigma} \min\{k|N_c|^2, \frac{n^2}{k^2 \log n}\})$ time. Since $O(\sum_{c\in\Sigma} \min\{k|N_c|^2, \frac{n^2}{k^2 \log n}\})$ is maximized when $k|N_c|^2 = \frac{n^2}{k^2 \log n}$, then $O(\sum_{c\in\Sigma} \min\{k|N_c|^2, \frac{n^2}{k^2 \log n}\}) \subseteq O((\sum_{c\in\Sigma} |N_c|)\frac{n}{\sqrt{k}\sqrt{\log n}})$ $\subseteq O(\frac{n^2}{\sqrt{k}\sqrt{\log n}})$. Therefore we can count in $O(\frac{n^2}{\sqrt{k}\sqrt{\log n}})$ time. Also, all k-sub-cadences can be located in $O(\frac{n^2}{\sqrt{k}\sqrt{\log n}} + occ)$ time. From the above, we obtain the following result:

▶ **Theorem 10.** *For a constant size alphabet (resp. for a general ordered alphabet), all k-cadences with given k can be counted in $O\left(\frac{n^2}{k^2 \log n}\right)$ time (resp. $O\left(\min\left\{\frac{n^2}{k^2}, \frac{n^2}{\sqrt{k}\sqrt{\log n}}\right\}\right)$ time) and $O(n)$ space, and can be located in $O\left(\frac{n^2}{k^2 \log n} + occ\right)$ time (resp. $O\left(\min\left\{\frac{n^2}{k^2}, \frac{n^2}{\sqrt{k}\sqrt{\log n}} + occ\right\}\right)$ time) and $O(n)$ space.*

5 Detecting Equidistant Subsequence Pattern

In this section, we consider algorithms for detecting equidistant subsequence pattern.

5.1 Algorithm 5

We use similar techniques as of Algorithm 1. For each distance d with $1 \leq d \leq \lfloor\frac{n-1}{m-1}\rfloor$, we construct text ST_d. After preparing the split text, we can compute ESP using existing substring pattern matching algorithms. Since Knuth-Morris-Pratt algorithm [9] runs in $O(n)$ time for a text of length n, we obtain the following result:

▶ **Theorem 11.** *There is an algorithm for locating all equidistant subsequence occurrences for given pattern P of length m which uses $O\left(\frac{n^2}{m}\right)$ time and $O(n)$ space.*

Like KSC, for text $T = \mathtt{a}^n$ and pattern $P = \mathtt{a}^m$, $|ESP|$ can be $\Omega(\frac{n^2}{m})$. Therefore, when we locate all $(i, d) \in ESP$, this algorithm is optimal in the worst case. In the next subsection, we show a counting algorithm that is efficient when the value of m is small. And we show a locating algorithm that is efficient when the value of m and $|ESP|$ is small.

5.2 Algorithm 6

Now we will show the following results:

▶ **Theorem 12.** *There is an algorithm for counting all equidistant subsequence occurrences which uses $O\left(\frac{n^2}{\log n}\right)$ time and $O\left(\frac{|\Sigma_P|n}{\log n}\right)$ space, where Σ_P is the set of distinct characters in the given pattern P. We can also locate these occurrences in $O\left(\frac{n^2}{\log n} + occ\right)$ time and $O\left(\frac{|\Sigma_P|n}{\log n}\right)$ space.*

First, we construct δ_c for all $c \in \Sigma_P$. For each d with $1 \leq d \leq \lfloor\frac{n-1}{m-1}\rfloor$, let $Q''_d = \delta_{P[1]}$ `&` $(\delta_{P[2]} \ll d)$ `&` $(\delta_{P[3]} \ll 2d)$ `&` $\cdots$ `&` $(\delta_{P[m]} \ll (m-1)d)$. If $Q''_d[i] = 1$, (i, d) is an occurrence of equidistant subsequence pattern P. See Figure 3 for a concrete example.

All of the elements of ESP can be counted / located by using a method similar to Algorithm 2 for Q''_d. After constructing δ_c for all $c \in \Sigma$, all occurrences of equidistant

$\boldsymbol{T}$	=	c	a	a	a	c	a	a	b	a	a	b	a	a	b	c	a	b	c
$\boldsymbol{P}$	=	a	a	c	c														
$\boldsymbol{\delta}_{\mathbf{a}}$	=	0	1	1	1	0	1	1	0	1	1	0	1	1	0	0	1	0	0
$\boldsymbol{\delta}_{\mathbf{c}}$	=	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1
$\boldsymbol{\delta}_{\mathbf{a}}$	=	0	0	1	1	0	1	1	0	1	1	0	1	1	0	0	1	0	0
$(\boldsymbol{\delta}_{\mathbf{a}} \ll 3)$	=	1	0	1	1	0	1	1	0	1	1	0	0	1	0	0	0	0	0
$(\boldsymbol{\delta}_{\mathbf{c}} \ll 6)$	=	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0
$(\boldsymbol{\delta}_{\mathbf{c}} \ll 9)$	=	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
$\boldsymbol{Q}''_{\mathbf{3}}$	=	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Figure 3 Let $T =$ caaacaabaabaabcabc and $P =$ aacc. $(9, 3)$ is an occurrence of equidistant subsequence pattern with $d = 3$.

subsequence pattern can be counted in $O(\frac{n^2}{\log n})$ time and $O(n)$ space and can be located in $O(\frac{n^2}{\log n} + occ)$ time and $O(n)$ space. Constructing δ_c for all $c \in \Sigma_P$ needs $O(\frac{|\Sigma_P|n}{\log n})$ time and space. Since $\frac{|\Sigma_P|n}{\log n}$ is at most $O(\frac{n^2}{\log n})$, we get Theorem 12.

If m is $o(\log n)$, Algorithm 6 is faster than Algorithm 5 and $O(\frac{|\Sigma_P|n}{\log n}) \subseteq O(n)$. From the above, we obtain the following result:

▶ **Theorem 13.** *All occurrences of equidistant subsequence pattern can be counted in* $O\left(\min\left\{\frac{n^2}{m}, \frac{n^2}{\log n}\right\}\right)$ *time and* $O(n)$ *space and can be located in* $O\left(\min\left\{\frac{n^2}{m}, \frac{n^2}{\log n} + occ\right\}\right)$ *time and* $O(n)$ *space.*

6 Detecting Equidistant Subsequence Pattern of Length Three

In this section, we show more efficient algorithms that count all occurrences of an equidistant subsequence pattern for the case where the length of the pattern is three. In addition, we show an algorithm for counting all occurrences of equidistant Abelian subsequence patterns of length three. Since we heavily use the techniques of [5] for 3-sub-cadences, we first show their algorithm for 3-sub-cadences and then generalize it for solving the equidistant subsequence pattern matching problem.

6.1 Counting 3-sub-cadences [5]

Let $a[0..n]$ and $b[0..n]$ be two sequences. The sequence $h[1..2n]$ can be computed by the discrete acyclic convolution $h[z] = \sum_{\substack{x+y=z \\ (x,y)\in[0,1,2,\ldots,n]^2}} a[x]b[y]$. The discrete acyclic convolution can be computed in $O(n \log n)$ time by using the fast Fourier transform. This convolution can be interpreted geometrically as follows: $h[z] = \sum_{\substack{x+y=z \\ (x,y)\in G\cap\mathbb{Z}^2}} a[x]b[y]$, where G is the square given by $\{(x, y) : 0 \le x, y \le n\}$.

Funakoshi and Pape-Lange [5] showed that 3-sub-cadences can be counted by using the discrete acyclic convolution. If (i, d) is a 3-sub-cadence with a character c, then $\delta_c[i]\cdot\delta_c[i+2d] = 1$ and $T[i + d] = c$. Let $h[2z] = \sum_{\substack{x+y=2z \\ (x,y)\in[0,1,2,\ldots,n]^2}} \delta_c[x]\delta_c[y]$, then $h[2z]$ counts how many pairs x and y there are that satisfies $x + y = 2z$ and $T[x] = T[y] = c$ for the index z. Since

$z+z=2z$ and $\delta_c[z]\cdot\delta_c[z]=1$ if $T[z]=1$, then $h[2z]$ counts one false positive. In addition, $x+y=2z$ and $\delta_c[x]\cdot\delta_c[y]=1$ if $x\neq y$, then $h[2z]$ counts twice for same x and y. Let $f[z]$ be the number of all 3-sub-cadences with a character c such that the middle index of 3-sub-cadences is z. $f[z]$ can be computed in $O(n\log n)$ time as follows:

$$f[z] := \begin{cases} \frac{h[2z]-1}{2} & \text{if } T[z]=c, \\ 0 & \text{if } T[z]\neq c. \end{cases}$$

Furthermore, they extended the geometric interpretation of convolution and showed that if G is a triangle with perimeter p, the sequence c can be computed in $O(p\log^2 p)$ time.

6.2 Counting Equidistant Subsequence Patterns of Length Three

Now we show an algorithm for counting all occurrences of equidistant subsequence patterns whose length is three. Let $g[z]$ be the number of all occurrences of the equidistant subsequence pattern such that the middle index of P is z. If $P=\alpha\alpha\alpha$, this problem is equal to the counting all 3-sub-cadences problem. Therefore, $g[z]$ can be computed in $O(n\log n)$ time as follows:

$$g[z] := \begin{cases} \frac{h[2z]-1}{2} & \text{if } T[z]=\alpha \\ 0 & \text{if } T[z]\neq \alpha \end{cases}$$

where $h[2z]=\sum_{\substack{x+y=2z \\ (x,y)\in[0,1,2,\ldots,n]^2}} \delta_\alpha[x]\delta_\alpha[y]$.

If $P=\alpha\beta\alpha$, since the pattern is symmetrical, $g[z]$ can be computed in $O(n\log n)$ time as follows, by using almost the same technique as above:

$$g[z] := \begin{cases} \frac{h[2z]}{2} & \text{if } T[z]=\beta \\ 0 & \text{if } T[z]\neq \beta \end{cases}$$

where $h[2z]=\sum_{\substack{x+y=2z \\ (x,y)\in[0,1,2,\ldots,n]^2}} \delta_\alpha[x]\delta_\alpha[y]$.

However, if $P=\alpha\beta\gamma$, then $h[2z]=\sum_{\substack{x+y=2z \\ (x,y)\in[0,1,2,\ldots,n]^2}} \delta_\alpha[x]\delta_\gamma[y]$ would also include occurrences of the equidistant subsequence pattern $\gamma\beta\alpha$. Thus, in order to compute $g[z]$, we further add the condition $x<y$. By using triangle convolution of [5], $g[z]$ can be computed in $O(n\log^2 n)$ time as follows:

$$g[z] := \begin{cases} h[2z] & \text{if } T[z]=\beta \\ 0 & \text{if } T[z]\neq \beta \end{cases}$$

where $h[2z]=\sum_{\substack{x+y=2z \\ (x,y)\in G\cap\mathbb{Z}^2}} \delta_\alpha[x]\delta_\gamma[y]$, where G is the triangle as in the following Figure 4.

If $P=\alpha\alpha\gamma$ or $P=\alpha\gamma\gamma$, we can compute $g[z]$ by using the same technique as for the case of $P=\alpha\beta\gamma$. Therefore, we get the following result:

▶ **Theorem 14.** *All occurrences of equidistant subsequence pattern of length three can be counted in $O(n\log^2 n)$ time and $O(n)$ space.*

6.3 Counting Equidistant Abelian Subsequence Patterns of Length Three

Now we show the algorithm for counting all occurrences of equidistant Abelian subsequence pattern whose length is three. In this subsection we consider the case where all of the three characters are distinct, namely, $P=\alpha\beta\gamma$. The other cases can be computed similarly.

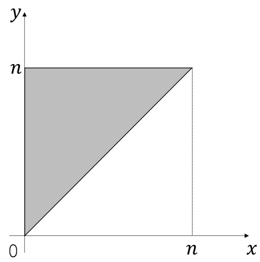

Figure 4 The triangle G.

In the previous subsection, we showed that if $P = \alpha\beta\gamma$, then $h[2z] = \sum_{\substack{x+y=2z \\ (x,y)\in[0,1,2,\ldots,n]^2}} \delta_\alpha[x]\delta_\gamma[y]$ includes the occurrences of equidistant subsequence pattern $\gamma\beta\alpha$. Therefore, we can compute all occurrences of equidistant subsequence pattern $\alpha\beta\gamma$, $\gamma\beta\alpha$, $\beta\gamma\alpha$, $\alpha\gamma\beta$, $\gamma\alpha\beta$, and $\beta\alpha\gamma$ by using discrete acyclic convolution for $P = \alpha\beta\gamma$, $P = \beta\gamma\alpha$, and $P = \gamma\alpha\beta$. Hence, we can get following result:

▶ **Theorem 15.** *All occurrences of equidistant Abelian subsequence pattern of length three can be counted in $O(n \log n)$ time and $O(n)$ space.*

References

1 Amihood Amir, Alberto Apostolico, Travis Gagie, and Gad M. Landau. String cadences. *Theoretical Computer Science*, 698:4–8, 2017. Algorithms, Strings and Theoretical Approaches in the Big Data Era (In Honor of the 60th Birthday of Professor Raffaele Giancarlo).

2 Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.

3 Maxime Crochemore and Dominique Perrin. Two-way string-matching. *J. ACM*, 38(3):650–674, 1991.

4 Simone Faro, Thierry Lecroq, Stefano Borzi, Simone Di Mauro, and Alessandro Maggio. The string matching algorithms research tool. In *Proceedings of the Prague Stringology Conference 2016, Prague, Czech Republic, August 29-31, 2016*, pages 99–111. Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, 2016.

5 Mitsuru Funakoshi and Julian Pape-Lange. Non-rectangular convolutions and (sub-)cadences with three elements. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 30:1–30:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

6 Zvi Galil and Joel Seiferas. Time-space-optimal string matching. *Journal of Computer and System Sciences*, 26(3):280–294, 1983.

7 J. Gardelle. Cadences. *Mathématiques et Sciences humaines*, 9:31–38, 1964.

8 R. Nigel Horspool. Practical fast searching in strings. *Software: Practice and Experience*, 10(6):501–506, 1980.

9 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

10 M. Lothaire. *Combinatorics on Words*. Cambridge Mathematical Library. Cambridge University Press, 1997.

11 Phillip Matier and Andrew Ross. Did Schwarzenegger drop 4-letter bomb in veto? San Francisco Chronicle, 2009. URL: `http://www.sfgate.com/cgi-bin/article.cgi?f=/c/a/2009/10/28/MNBN1ABKB8.DTL`.

12 Doron Witztum, Eliyahu Rips, and Yoav Rosenberg. Equidistant letter sequences in the book of genesis. *Statistical Science*, 9(3):429–438, 1994.