Super-d-complexity of finite words

Zoltán Kása

Sapientia Hungarian University of Transylvania
Cluj–Tg. Mureş–M. Ciuc
Department of Mathematics and Informatics, Tg. Mureş/Marosvásárhely
kasa@ms.sapientia.ro

Abstract

For positive integer d special scattered subwords, named super-d-subwords, in which the gaps are of length at least $(d-1)$, are defined. The super-d-complexity (the number of super-d-subwords) is studied for rainbow words.

Subject Classifications: MSC2010: 68R15 **CCS1998:** G.2.1, F.2.2

1 Introduction

Sequences of characters called *words* or *strings* are widely studied in combinatorics, and used in various fields of sciences (e.g. biology, chemistry, physics, social sciences etc.) [2, 3, 4, 8]. The elements of a word are called *letters*. A contiguous part of a word (obtained by eliminating a prefix or/and a suffix) is a *subword* or *factor*. If we eliminate arbitrary letters from a word, what is obtained is a *scattered subword*. Special scattered subwords, in which the consecutive letters are at distance at most d ($d \geq 1$) in the original word, are called *d-subwords* [5, 6]. In this paper we define another kind of scattered subwords, in which the original distance between two letters which are consecutive in the subword, is at least d ($d \geq 1$), these will be called *super-d-subwords*.

The *complexity of a word* is defined as the number of all its different subwords. Similar definitions are for *d-complexity* and *super-d-complexity*.

Let Σ be an alphabet, Σ^n the set of all n-length words over Σ, Σ^* the set of all finite word over Σ, and d a positive integer.

Definition 1 *Let n, d and s be positive integers, and $u = x_1x_2\ldots x_n \in \Sigma^n$. A **super-$d$-subword** of length s of u is defined as $v = x_{i_1}x_{i_2}\ldots x_{i_s}$ where*
$i_1 \geq 1$,
$d \leq i_{j+1} - i_j < n$ *for* $j = 1, 2, \ldots, s-1$,
$i_s \leq n$.

Definition 2 *The **super-d-complexity** of a word is the number of all its different super-d-subwords.*

The super-2-subwords of the word *abcdef* are the following: *a, ac, ad, ae, af, ace, acf, adf, b, bd, be, bf, bdf, c, ce, cf, d, df, e, f,* therefore the super-2-complexity of this word is 20.

2 Super-d-complexity of rainbow words

Words with different letters are called *rainbow words*. The super-d-complexity of an n-length rainbow word does not depends on what letters it contains, and is denoted by $S(n, d)$.

Let us denote by $b_{n,d}(i)$ the number of super-d-subwords which begin in the position i in an n-length rainbow word. Using our previous example (*abcdef*), we can see that $b_{6,2}(1) = 8$, $b_{6,2}(2) = 5$, $b_{6,2}(3) = 3$, $b_{6,2}(4) = 2$, $b_{6,2}(5) = 1$, and $b_{6,2}(6) = 1$.

n \ d	1	2	3	4	5	6	7	8	9	10	11
1	1	1	1	1	1	1	1	1	1	1	1
2	3	2	2	2	2	2	2	2	2	2	2
3	7	4	3	3	3	3	3	3	3	3	3
4	15	7	5	4	4	4	4	4	4	4	4
5	31	12	8	6	5	5	5	5	5	5	5
6	63	20	12	9	7	6	6	6	6	6	6
7	127	33	18	13	10	8	7	7	7	7	7
8	255	54	27	18	14	11	9	8	8	8	8
9	511	88	40	25	19	15	12	10	9	9	9
10	1023	143	59	35	25	20	16	13	11	10	10
11	2047	232	87	49	33	26	21	17	14	12	11
12	4095	376	128	68	44	33	27	22	18	15	13

Table 1: Values of $S(n,d)$.

The following formula immediately results:

$$b_{n,d}(i) = 1 + b_{n,d}(i{+}d) + b_{n,d}(i{+}d{+}1) + \cdots + b_{n,d}(n), \tag{1}$$

for $n > d, 1 \le i \le n-d$,

$$b_{n,d}(1) = 1 \text{ for } n \le d.$$

The super-d-complexity of rainbow words can be computed by the formula:

$$S(n,d) = \sum_{i=1}^{n} b_{n,d}(i). \tag{2}$$

This can be expressed also as

$$S(n,d) = \sum_{k=1}^{n} b_{k,d}(1), \tag{3}$$

because of the formula

$$S(n+1,d) = S(n,d) + b_{n+1,d}(1).$$

In the case $d = 1$ the complexity $S(n,1)$ can be computed easily: $S(n,1) = 2^n - 1$. This is equal to the n-complexity of n-length rainbow words.

3 Computing super-d-complexity

In this section we will present different methods to compute the super-d-complexity of rainbow words.

3.1 Computing by recursive algorithm

From (1) for the computation of $b_{n,d}(i)$ the following algorithm results. The numbers $b_{n,d}(k)$ ($k = 1, 2, \ldots$) for a given n and d are obtained in the array $b = (b_1, b_2, \ldots)$. Initially all these elements are equal to -1. The call for the given n and d and the desired i is:

```
for k ← 1 to n
    do b_k ← −1
B(n, d, i)
```

The recursive algorithm is the following:

```
B(n, d, i)
p ← 1
for k ← i + d to n
    do if b_k = −1
           then B(n, d, k)
       p ← p + b_k
b_i ← p
return
```

If the call is $B(8, 2, 1)$, the elements will be obtained in the following order: $b_7 = 1$, $b_8 = 1$, $b_5 = 3$, $b_6 = 2$, $b_3 = 8$, $b_4 = 5$, and $b_1 = 21$.

Lemma 3 $b_{n,2}(1) = F_n$, *where* F_n *is the nth Fibonacci number.*

Proof. Let us consider a rainbow word $a_1a_2 \dots a_n$ and let us count all its super-2-subwords which begin with a_2. If we change a_2 in a_1 in each super-2-subword which begin with a_2, we obtain super-2-subwords too. If we add a_1 in front of each super-d-subword which begin with a_3, we obtain super-d-subwords too. Thus

$$b_{n,2}(1) = b_{n-1,2}(1) + b_{n-2,2}(1).$$

So $b_{n,2}(1)$ is a Fibonacci number, and because $b_{1,2}(1) = 1$, we obtain $b_{n,2}(1) = F_n$. □

Theorem 4 $S(n, 2) = F_{n+2} - 1$, *where* F_n *is the nth Fibonacci number.*

Proof. From (3) and Lemma 3:

$$\begin{aligned} S(n,2) &= b_{1,2}(1) + b_{2,2}(1) + b_{3,2}(1) + b_{4,2}(1) + \cdots + b_{n,2}(1) \\ &= F_1 + F_2 + \cdots + F_n \\ &= F_{n+2} - 1. \end{aligned}$$

□

If we denote by $M_{n,d} = b_{n,d}(1)$, because of the formula

$$b_{n,d}(1) = b_{n-1,d}(1) + b_{n-d,d}(1),$$

a generalized middle sequence [7] (see sequence A000930) will be obtained:

$$\begin{aligned} &M_{n,d} = M_{n-1,d} + M_{n-d,d}, \quad \text{for } n \geq d \geq 2, \\ &M_{0,d} = 0,\ M_{1,d} = 1,\ \dots,\ M_{d-1,d} = 1. \end{aligned} \tag{4}$$

Let us name this sequence *d-middle sequence*. Because of the $M_{n,2} = F_n$ equality, the d-middle sequence can be considered as a generalization of the Fibonacci sequence.

The d-middle sequence defined in (4) is a little different from the generalization of the sequence A000930 in [7] because of its initial values.

Then next algorithm computes $M_{n,d}$, by using an array $M_0, M_1, \dots, M_{d-1}$ to store the necessary previous elements:

```
Middle(n, d)
M_0 ← 0
for i ← 1 to d − 1
    do M_i ← 1
for i ← d to n
    do M_{i mod d} ← M_{(i−1) mod d} + M_{(i−d) mod d}
        print M_{i mod d}
return
```

Using the generating function $M_d(z) = \sum_{n\geq 0} M_{n,d} z^n$, the following closed formula results:

$$M_d(z) = \frac{z}{1-z-z^d}. \tag{5}$$

This can be used to compute the sum $s_{n,d} = \sum_{n=1}^{n} M_{i,d}$, which is the coefficient of z^{n+d} in the expansion of the function

$$\frac{z^d}{1-z-z^d} \cdot \frac{1}{1-z} = \frac{z^d}{1-z-z^d} + \frac{z}{1-z-z^d} - \frac{z}{1-z}.$$

So $s_{n.d} = M_{n+(d-1),d} + M_{n,d} - 1 = M_{n+d,d} - 1$. Therefore

$$\sum_{i=1}^{n} M_{i,d} = M_{n+d,d} - 1. \tag{6}$$

Theorem 5 *$S(n,d) = M_{n+d,d} - 1$, where $n > d$ and $M_{n,d}$ is the nth elements of d-middle sequence.*

Proof. The proof is similar to that in Theorem 4 taking into account formula (6). □

3.2 Computing by mathematical formulas

Theorem 6 $S(n,d) = \sum_{k\geq 0} \binom{n-(d-1)k}{k+1}$, *for $n \geq 2, d \geq 1$.*

Proof. Let us consider the generating function $G(z) = \frac{1}{1-z} = 1 + z + z^2 + \cdots$. Then, taking into account the formula (5) we obtain $M_d(z) = zG(z+z^d) = z + z(z+z^d) + z(z+z^d)^2 + \cdots + z(z+z^d)^i + \cdots$. The general term in this expansion is equal to

$$z^{i+1} \sum_{k=1}^{i} \binom{i}{k} z^{(d-1)k},$$

and the coefficient of z^{n+1} is equal to

$$\sum_{k\geq 0} \binom{n-(d-1)k}{k}.$$

The coeeficient of z^{n+d} is

$$M_{n+d,d} = \sum_{k\geq 0} \binom{n+d-1-(d-1)k}{k}. \tag{7}$$

By Theorem 5 $S(n,d) = M_{n+d,d} - 1$, and an easy computation yields

$$S(n,d) = \sum_{k\geq 0} \binom{n-(d-1)k}{k+1}.$$

□

Theorem 7 $b_{n+1,d}(1) = \sum_{k\geq 0} \binom{n-(d-1)k}{k}$, *for* $n \geq 1, d \geq 1$.

Proof. From $b_{n+1,d}(1) = M_{n+1,d}$ and (7):

$$b_{n+1,d} = \sum_{k\geq 0} \binom{n-(d-1)k}{k}.$$

□

3.3 Computing by graph algorithms

To compute the super-d-complexity of a rainbow word of length n, let us consider the word $a_1 a_2 \ldots a_n$ and the correspondig digraph $G = (V, E)$, with

$V = \{a_1, a_2, \ldots, a_n\}$,

$E = \{(a_i, a_j) \mid j - i \geq d,\ i = 1, 2, \ldots, n, j = 1, 2, \ldots, n\}$.

For $n = 6, d = 2$ see Fig. 1.

The adjacency matrix $A = \left(a_{ij}\right)_{\substack{i=\overline{1,n}\\ j=\overline{1,n}}}$ of the graph is defined by:

$$a_{ij} = \begin{cases} 1, & \text{if } j - i \geq d, \\ 0, & \text{otherwise,} \end{cases} \qquad \text{for } i = 1, 2, \ldots, n, j = 1, 2, \ldots, n.$$

Because the graph has no directed cycles, the element in row i and column j in A^k (where $A^k = A^{k-1}A$, with $A^1 = A$) will represent the number of k-length directed paths from a_i to a_j. If I is the identity matrix (with elements equal to 1 only on the first diagonal, and 0 otherwise), let us define the matrix $R = (r_{ij})$:

$$R = I + A + A^2 + \cdots + A^k, \text{ where } A^{k+1} = O \text{ (the null matrix).}$$

The super-d-complexity of a rainbow word is then

$$S(n,d) = \sum_{i=1}^{n} \sum_{j=1}^{n} r_{ij}.$$

Matrix R can be better computed using a variant of the well-known Warshall algorithm (see for example [1]):

```
Warshall(A, n)
W ← A
for k ← 1 to n
    do for i ← 1 to n
           do for j ← 1 to n
                  do w_ij ← w_ij + w_ik w_kj
return W
```

From W we obtain easily $R = I + W$.

For example let us consider the graph in Fig. 1. The corresponding adjacency matrix is:

$$A = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

After applying the Warshall algorithm:

$$W = \begin{pmatrix} 0 & 0 & 1 & 1 & 2 & 3 \\ 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \qquad R = \begin{pmatrix} 1 & 0 & 1 & 1 & 2 & 3 \\ 0 & 1 & 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

and then $S(6,2) = 20$, the sum of elements in R.

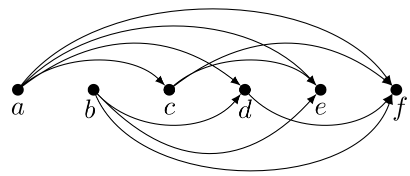

Figure 1: Graph for 2-subwords when $n = 6$.

The Warshall algorithm combined with the Latin square method can be used to obtain all nontrivial (with length at least 2) super-d-subwords of a given n-length rainbow word $a_1 a_2 \cdots a_n$. Let us consider a matrix $\mathcal{A}$ with the elements A_{ij} which are set of strings. Initially this matrix is defined as:

$$A_{ij} = \begin{cases} \{a_i a_j\}, & \text{if } j - i \geq d, \\ \emptyset, & \text{otherwise,} \end{cases} \qquad \text{for } i = 1, 2, \ldots, n, \; j = 1, 2, \ldots, n.$$

If $\mathcal{A}$ and $\mathcal{B}$ are sets of strings, $\mathcal{AB}$ will be formed by the set of concatenation of each string from $\mathcal{A}$ with each string from $\mathcal{B}$:

$$\mathcal{AB} = \{ ab \mid a \in \mathcal{A}, b \in \mathcal{B} \}.$$

If $s = s_1 s_2 \cdots s_p$ is a string, let us denote by $'s$ the string obtained from s by eliminating the first character: $'s = s_2 s_3 \cdots s_p$. Let us denote by $'A_{ij}$ the set A_{ij} in which we eliminate from each element the first character. In this case $'\mathcal{A}$ is a matrix with elements $'A_{ij}$.

Starting with the matrix $\mathcal{A}$ defined as before, the algorithm to obtain all nontrivial super-d-subwords is the following:

Warshall-Latin($\mathcal{A}, n$)

$\mathcal{W} \leftarrow \mathcal{A}$
for $k \leftarrow 1$ **to** n
  do for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
      do if $W_{ik} \neq \emptyset$ and $W_{kj} \neq \emptyset$
        then $W_{ij} \leftarrow W_{ij} \cup W_{ik}\,'W_{kj}$
return $\mathcal{W}$

The set of nontrivial super-d-subwords is $\bigcup_{i,j\in\{1,2,\ldots,n\}} W_{ij}$.

For $n = 8$, $d = 3$ the initial matrix is:

$$\begin{pmatrix}
\emptyset & \emptyset & \emptyset & \{ad\} & \{ae\} & \{af\} & \{ag\} & \{ah\} \\
\emptyset & \emptyset & \emptyset & \emptyset & \{be\} & \{bf\} & \{bg\} & \{bh\} \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{cf\} & \{cg\} & \{ch\} \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{dg\} & \{dh\} \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{eh\} \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset
\end{pmatrix}.$$

The result of the algorithm in this case is:

$$\begin{pmatrix}
\emptyset & \emptyset & \emptyset & \{ad\} & \{ae\} & \{af\} & \{ag, adg\} & \{ah, adh, aeh\} \\
\emptyset & \emptyset & \emptyset & \emptyset & \{be\} & \{bf\} & \{bg\} & \{bh, beh\} \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{cf\} & \{cg\} & \{ch\} \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{dg\} & \{dh\} \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{eh\} \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset
\end{pmatrix}.$$

4 The general case

In the general case for any word $w \in \Sigma^*$, let us denote the super-d-complexity by $S_w(d)$. We have

$$\left\lceil \frac{|w|}{d} \right\rceil \le S_w(d) \le S(|w|, d),$$

where $|w|$ is the length of w. The minimal value is obtained for a trivial word $w = a \ldots a$, and the maximal one for a rainbow word.

The algorithm Warshall-Latin can be used for nonrainbow words too, with the remark that repeating subwords must be eliminated. For the word *aabbbaaa* and $d = 3$ the result is: *aa*, *ab*, *aba*, *ba*.

n \ d	2	3	4	5	6	7	8	9	10	11
3	3	-	-	-	-	-	-	-	-	-
4	5	3	-	-	-	-	-	-	-	-
5	7	5	3	-	-	-	-	-	-	-
6	10	6	5	3	-	-	-	-	-	-
7	14	7	6	5	3	-	-	-	-	-
8	19	10	6	6	5	3	-	-	-	-
9	26	13	7	6	6	5	3	-	-	-
10	35	15	10	6	6	6	5	3	-	-
11	47	19	13	7	6	6	6	5	3	-
12	63	25	14	10	6	6	6	6	5	3

Table 2: Values of $f(2, n, d)$.

Let us denote by $f(m, n, d)$ the maximal value of the super-d-complexity of all words of length n over an alphabet of m letters:

$$f(m,n,d) = \max_{\substack{w \in \Sigma^n \\ m = |\Sigma|}} \Big(S_w(d) \Big).$$

For $f(2,n,d)$ the following are true, and can be easily proved.

- $f(2,n,n-1) = 3$ for $n \geq 3$.
- $f(2,n,n-2) = 5$ for $n \geq 4$.
- If $\left\lceil \dfrac{n}{2} \right\rceil \leq d \leq n-3$ then $f(2,n,d) = 6$ for $n \geq 6$.
- If n is even, then $f\left(2, n, \dfrac{n-2}{2}\right) = 10$ for $n \geq 6$.
- If n is odd, then $f\left(2, n, \dfrac{n-1}{2}\right) - 7$ for $n \geq 5$.

For $m = d = 2$ the following conjecture is stated.

Conjecture 8 $f(2,n,2) = f(2,n-1,2) + f(2,n-2,2) - f(2,n-4,2)$ *for* $n \geq 7$.

Acknowledgment

The European Union and the European Social Fund have provided financial support to the project of this work under the grant agreement no. TÁMOP-4.2.1./B-09/1/KMR-2010-0003.

References

[1] S. Baase, *Computer algorithms: Introduction to design and analysis,* Second edition, Addison–Wesley, 1988.

[2] W. Ebeling, R. Feistel, *Physik der Selbstorganisation und Evolution,* Akademie-Verlag, Berlin, 1982.

[3] C. Elzinga, S. Rahmann, H. Wung, Algorithms for subsequence combinatorics, *Theor. Comput. Sci.*, **409,** 3 (2008) 394–404.

[4] C. H. Elzinga, Complexity of categorial time series, *Sociological Methods & Research*, **38,** 3 (2010) 463–481.
http://home.fsw.vu.nl/ch.elzinga/Complexity%20Preliminary.pdf

[5] A. Iványi, On the d-complexity of words, *Annales Univ. Sci. Budapest., Sect. Computatorica,* **8** (1987) 69–90.

[6] Z. Kása, On the d-complexity of strings, *Pure Math. Appl.*, **9,** 1–2 (1998) 119–128.

[7] N. J. A. Sloane, The on-line encyclopedia of integer sequences,
`http://www.research.att.com/~njas/sequences/`.

[8] O. G. Troyanskaya, O. Arbell, Y. Koren, G. M. Landau, A. Bolshoy, Sequence complexity profiles of prokaryotic genomic sequences: A fast algorithm for calculating linguistic complexity, *Bioinformatics,* **18,** 5 (2002) 679–688.