Super-$d$-complexity of finite words

Zoltán Kása

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

Abstract

In this paper we introduce and study a new complexity measure for finite words. For positive integer $d$ special scattered subwords, called super-$d$-subwords, in which the gaps are of length at least $(d - 1)$, are defined. We give methods to compute super-$d$-complexity (the total number of different super-$d$-subwords) in the case of rainbow words (with pairwise different letters) by recursive algorithms, by mathematical formulas and by graph algorithms. In the case of general words, with letters from a given alphabet without any restriction, the problem of the maximum value of the super-$d$-complexity of all words of length $n$ is presented.

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

1 A new complexity measure: the super-$d$-complexity

Sequences of characters called words or strings are widely studied in combinatorics, and used in various fields of sciences (e.g. chemistry, physics, social sciences, biology [3, 4, 5, 9] etc.). The elements of a word are called letters. A contiguous part of a word (obtained by erasing a prefix or/and a suffix) is a subword or factor. If we erase 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 [6,7]. 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.

One can easily observe that in any given word, the 1-subwords are exactly the (ordinary) subwords, and the super-1-subwords are exactly the scattered subwords.

The complexity of a word is defined as the total number of its different subwords. The definitions of $d$-complexity and super-$d$-complexity are similar.

For a (finite) alphabet $\Sigma$, as usual, $\Sigma^n$ and $\Sigma^*$ are the sets of all words of length $n$, and of all finite words, respectively, over $\Sigma$.

In order to formalize the above, we introduce the following two definitions.
Definition 1 Let $n$, $d$ and $s$ be positive integers, and $u = x_1 x_2 \ldots x_n \in \Sigma^n$. A super-$d$-subword of length $s$ of $u$ is defined as $v = x_{i_1} x_{i_2} \ldots x_{i_s}$ where

\begin{align*}
i_1 &\geq 1, \\
d &\leq i_{j+1} - i_j < n \text{ for } j = 1, 2, \ldots, s - 1, \\
i_s &\leq n.
\end{align*}

Definition 2 The super-$d$-complexity of a word is the total number of 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 Computing the super-$d$-complexity of rainbow words

Words with pairwise different letters are called rainbow words. The super-$d$-complexity of a rainbow word of length $n$ 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 at the $i$-th position in a rainbow word of length $n$. 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 $b_{6,2}(6) = 1$.

We immediately get the following formula:

\begin{equation}
b_{n,d}(i) = 1 + b_{n,d}(i+d) + b_{n,d}(i+d+1) + \cdots + b_{n,d}(n),
\end{equation}

for $n > d$, $1 \leq i \leq n - d$, and

\begin{equation}
b_{n,d}(1) = 1 \text{ for } n \leq d.
\end{equation}

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

\begin{equation}
S(n, d) = \sum_{i=1}^{n} b_{n,d}(i).
\end{equation}

This can be expressed also as

\begin{equation}
S(n, d) = \sum_{k=1}^{n} b_{k,d}(1),
\end{equation}

because of the formula

\begin{equation}
S(n + 1, d) = S(n, d) + b_{n+1,d}(1).
\end{equation}
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 rainbow words of length $n$.

In the sequel we will present different methods to compute the super-$d$-complexity of the rainbow words. In the description of algorithms the pseudocode conventions from [2] are used.

### 2.1 Computing by recursive algorithm

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

**Input** $n, d, i$

**for** $k \leftarrow 1$ **to** $n$

**do** $b_k \leftarrow -1$

$B(n, d, i)$

**Output** $b_1, b_2, \ldots, b_n$

The recursive algorithm is the following:
B(n, d, i)
1  \( p \leftarrow 1 \)
2  \[\text{for } k \leftarrow i + d \text{ to } n \]
3  \[\text{do if } b_k = -1 \]
4  \[\text{then } B(n, d, k) \]
5  \( p \leftarrow p + b_k \)
6  \( b_i \leftarrow p \)
7  \text{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 \( n \)-th Fibonacci number.

**Proof.** Let us consider a rainbow word \( a_1a_2\ldots a_n \) and let us count all of its super-2-subwords which begin with \( a_2 \). If we change \( a_2 \) for \( a_1 \) in each super-2-subword which begin with \( a_2 \), we again obtain super-2-subwords. If we prefix an \( a_1 \) to each super-\( d \)-subword which begin with \( a_3 \), we again obtain super-\( d \)-subwords. 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 \( n \)-th Fibonacci number.

**Proof.** From (3) and Lemma 3

\[
S(n, 2) = b_{1,2}(1) + b_{2,2}(1) + b_{3,2}(1) + b_{4,2}(1) + \cdots + b_{n,2}(1)
= F_1 + F_2 + \cdots + F_n
= F_{n+2} - 1. \quad \Box
\]

Introducing the notation \( M_{n,d} = b_{n,d}(1) \), then by the formula

\[
b_{n,d}(1) = b_{n-1,d}(1) + b_{n-d,d}(1),
\]

a generalized middle sequence (see the sequence A000930 in [8]) will be obtained in the following, recursive way:

\begin{equation}
M_{n,d} = M_{n-1,d} + M_{n-d,d}, \quad \text{for } n \geq d \geq 2,
\end{equation}

\begin{equation}
M_{0,d} = 0, \ M_{1,d} = 1, \ldots, \ M_{d-1,d} = 1.
\end{equation}

\footnote{From [8]: \( a_0 = a_1 = a_2 = 1; \) thereafter \( a_n = a_{n-1} + a_{n-3}, \) Might be called the Middle Sequence, since it is a cross between the Fibonacci sequence (A000045) and the Padovan sequence (A000931).}
Let us call this sequence \( d \)-middle sequence. Because of the equality \( M_{n,2} = F_n \), 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 [8] because of its initial values.

The next algorithm computes \( M_{n,d} \), by using an array \( M_0, M_1, \ldots, M_{d-1} \) to store the necessary previous elements:

```plaintext
MIDDLE(n, d)
1    M_0 ← 0
2    for i ← 1 to d - 1
3        do M_i ← 1
4    for i ← d to n
5        do M_{i mod d} ← M_{(i-1) mod d} + M_{(i-d) mod d}
6    print M_{i mod d}
7    return
```

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

\[
M_d(z) = \frac{z}{1 - z - z^d}.
\]

(5)

This can be used to compute the sum \( s_{n,d} = \sum_{i=1}^{n} M_{i,d} \), 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

\[
\sum_{i=1}^{n} M_{i,d} = M_{n+d,d} - 1.
\]

(6)

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

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

### 2.2 Computing by mathematical formulas

**Theorem 6** \( S(n, d) = \sum_{k \geq 0} \left( \frac{n - (d - 1)k}{k + 1} \right) \), 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
\[
\sum_{k \geq 0} \binom{n - (d-1)k}{k+1}.
\]
The coefficient of \( z^{n+d} \) is
\[
M_{n+d,d} = \sum_{k \geq 0} \binom{n + d - 1 - (d-1)k}{k}.
\]

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

2.3 Computing by graph algorithms

To compute the super-\( d \)-complexity of a rainbow word of length \( n \), let us consider the word \( a_1a_2\ldots a_n \) and the corresponding 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 Figure [1].

The adjacency matrix \( A = (a_{ij})_{i,j=1,n} \) of the graph is defined by:
\[
a_{ij} = \begin{cases} 1, & \text{if } j - i \geq d, \\ 0, & \text{otherwise}, \end{cases} \text{ for } i = 1, 2, \ldots, n, j = 1, 2, \ldots, n.
\]
Figure 1: Graph for 2-subwords when \( n = 6 \).

Because the graph has no directed cycles, the entry 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

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

To compute matrix \( R \), we define a variant of the well-known Warshall algorithm (for this see for example [1]):

\begin{verbatim}
Warshall(A, n)
1 W ← A
2 for k ← 1 to n
3     do for i ← 1 to n
4         do for j ← 1 to n
5             do w_{ij} ← w_{ij} + w_{ik}w_{kj}
6 return W
\end{verbatim}

From \( W \) we obtain easily \( R = I + W \).

For example let us consider the graph in Figure1. 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 we obtain:

\[
W = \begin{pmatrix}
0 & 0 & 1 & 1 & 2 & 3 \\
0 & 0 & 0 & 1 & 1 & 2 \\
0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}, \quad R = \begin{pmatrix}
1 & 0 & 1 & 1 & 2 & 3 \\
0 & 1 & 0 & 1 & 1 & 2 \\
0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix},
\]

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

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 rainbow word \( a_1a_2\cdots a_n \). Let us consider a matrix \( A \) with entries \( A_{ij} \) which are set of words. Initially this matrix is defined as:

\[
A_{ij} = \begin{cases}
\{a_ia_j\}, & \text{if } j - i \geq d, \\
\emptyset, & \text{otherwise},
\end{cases}
\text{ for } i = 1, 2, \ldots, n, \ j = 1, 2, \ldots, n.
\]

If \( A \) and \( B \) are sets of words, \( AB \) will be formed by the set of concatenation of each word from \( A \) with each word from \( B \):

\[
AB = \{ab \mid a \in A, b \in B\}.
\]

If \( s = s_1s_2\cdots s_p \) is a word, let us denote by \( 's \) the word obtained from \( s \) by erasing its first character: \( 's = s_2s_3\cdots s_p \). Let us denote by \( 'A_{ij} \) the set \( A_{ij} \) in which we erase from each element the first character. In this case \( 'A \) is a matrix with entries \( 'A_{ij} \).

Starting with the matrix \( A \) defined as before, the algorithm to obtain all nontrivial super-\( d \)-subwords is the following:

\[
\text{Warshall-Latin}(A, n)
\]

\begin{verbatim}
1 \( W \leftarrow A \\
2 \text{ for } k \leftarrow 1 \text{ to } n \\
3 \quad \text{ do for } i \leftarrow 1 \text{ to } n \\
4 \quad \quad \text{ do for } j \leftarrow 1 \text{ to } n \\
5 \quad \quad \quad \text{ do if } W_{ik} \neq \emptyset \text{ and } W_{kj} \neq \emptyset \\
6 \quad \quad \quad \quad \text{ then } W_{ij} \leftarrow W_{ij} \cup W_{ik} 'W_{kj} \\
7 \text{ return } W
\end{verbatim}

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:
The result of the algorithm in this case is:

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

The result of the algorithm in this case is:

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

3 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

\[
\left\lceil \frac{|w|}{d} \right\rceil \leq S_w(d) \leq S(|w|, d),
\]

where \( |w| \) is the length of \( w \). The minimum value is obtained for a trivial word \( w = a \ldots a \), and the maximum 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: \( aa, ab, aba, ba \).

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

\[
f(m, n, d) = \max_{w \in \Sigma^n} \left( S_w(d) \right).
\]

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 \( \left\lfloor \frac{n}{2} \right\rfloor \leq d \leq n - 3 \) then \( f(2, n, d) = 6 \) for \( n \geq 6 \).
Table 2: Values of \( f(2, n, d) \).

<table>
<thead>
<tr>
<th>(n)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>26</td>
<td>13</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>35</td>
<td>15</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>47</td>
<td>19</td>
<td>13</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>63</td>
<td>25</td>
<td>14</td>
<td>10</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

• If \(n\) is even, then \( f\left(2, n, \frac{n-2}{2}\right) = 10 \) for \(n \geq 6\).

• If \(n\) is odd, then \( f\left(2, n, \frac{n-1}{2}\right) - 7 \) for \(n \geq 5\).

Conclusions

The super-\(d\)-complexity of the finite rainbow words can be obtained by recursive algorithms, by direct mathematical formulas, and by graph algorithms, all these being presented in this paper. The advantage of the graphs algorithm is that these can be easily altered for obtaining not only the complexity, but the all super-\(d\)-subwords too. This method can be adapted to obtain the super-\(d\)-subwords in the general case of the words too, when no restriction on the letters are given.

In the set of all words of a given length over a given alphabet the maximum super-\(d\)-complexity may be interesting. We present here only some easy to prove results, an extensive study remaining for the future.

Acknowledgment

This work was supported by the project TÁMOP-4.2.1/B-09/1/KMR-2010-0003 of Eötvös Loránd University Budapest.

The author is indebted for the useful remarks and suggestions of the unknown referees.
References


