Backward Pattern Matching on Elastic Degenerate Strings

Petr Procházka, Ondřej Cvacho, Luboš Krčál and Jan Holub
Dep. of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Thakurova 2700/9, Prague 6, Czech Republic

Keywords: Elastic Degenerate Strings, Genomic Sequences, Degenerate Pattern Matching, Pan-Genomics.

Abstract: Recently, the concept of Elastic Degenerate Strings (EDS) was introduced as a way of representing a sequenced population of the same species. Several on-line Elastic Degenerate String Matching (EDSM) algorithms were presented so far. Some of them provide practical implementation. We propose a new on-line EDSM algorithm `BNDM-EDS`. Our algorithm combines two traditional algorithms `BNDM` and the `Shift-And` that were adapted to the specifics needed by Elastic Degenerate Strings. `BNDM-EDS` is running in $O(Nm\lceil\frac{m}{w}\rceil)$ worst-case time. This implies $O(Nm)$ time for small patterns, where m is the length of the searched pattern, N is the size of EDS, and w is the size of the computer word. The algorithm uses $O(N+n)$ space, where n is the length of EDS. `BNDM-EDS` requires a simple preprocessing step with time and space $O(m)$. Experimental results on real genomic data show superiority of `BNDM-EDS` over state-of-the-art algorithms.

1 INTRODUCTION

Decreasing costs of DNA sequencing enabled a huge development of projects focused on sequencing whole populations of individuals of the same species. The projects cataloguing human genetic variations include the 1000 Genomes Projects (Consortium, The 1000 Genomes Project, 2011) and the UK10K project (Consortium, The UK10K, 2015). Sequenced populations pose a challenge to fast pattern matching algorithms on a set of similar strings.

A common approach to represent the sequenced population is to store genetic variations of individuals of the population with respect to the chosen 'reference sequence'. This approach is very space efficient since single human individuals differ only in 0.1% bases on average. In return, the pattern matching algorithms performed on this kind of data are confronted by a few obstacles given by the structure of the data.

During the last two decades, many different formats representing the sequenced population on different level of detail were developed. A consensus sequence drawn from an entire population in multiple sequence alignment (MSA) is one of the most concise forms that the sequenced population can take. The consensus sequence can be expressed as a degenerate string over a degenerate alphabet. IUPAC alphabet (on Biochemical Nomenclature (CBN), 1971) is used for genomic data. Another more recent format is EDS (Elastic Degenerate String) (Iliopoulos et al., 2017). EDS enables to express the variants of different lengths within one degenerate position in the sequence which provides more accuracy since insertions and deletions can be expressed precisely. Variation graph (Marschall, 2018; Church et al., 2015; Dilthey et al., 2015) data structure provides similar level of accuracy as EDS. It is a graph data structure where the backbone of the graph represents the reference genome and other alternatives (variants) across the population are stored as additional edges. The 1000 Genomes Projects (Consortium, The 1000 Genomes Project, 2011) stores the sequenced populations in `vcf` format (Danecek et al., 2011). The variant-call-format (`vcf`) provides even more information since the stored variations are addressed to single individuals of the population.

The idea of Pan-genome poses another challenge for efficient storage of the sequenced population because the Pan-genome represents a collection of genomic sequences that are analyzed together or used as a reference (Consortium, 2016).

A lot of work has been done on *off-line* searching in the sequenced populations, i.e. many index data structures solving this problem were proposed (Procházka and Holub, 2014; Maciuca et al., 2016; Na et al., 2016; Navarro and Pereira, 2016; Sirén, 2016). However, the authors of (Grossi et al., 2017) clearly state the following scenarios where *on-line* version is more appropriate: (i) efficient on-line solutions can be used in combination with partial

Procházka, P., Cvacho, O., Krčál, L. and Holub, J.
Backward Pattern Matching on Elastic Degenerate Strings.
DOI: 10.5220/0010243600500059
In *Proceedings of the 14th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2021) - Volume 3: BIOINFORMATICS*, pages 50-59
ISBN: 978-989-758-490-9

(block-oriented) indexes as practical trade-offs; (ii) efficient on-line solutions for exact pattern matching can be applied for fast average-case approximate pattern matching; (iii) on-line solutions can be useful when one wants to search for a set of patterns in elastic degenerate texts.

Elastic Degenerate Strings (EDS) provide an alternative to represent consensus/pan-genomic sequences. Implementation called `ElDeS` (the first on-line search algoritm on EDS) was provided later satisfying the presented upper bound $O(N+\alpha\gamma nm)$, where α is the maximum number of strings in any elastic-degenerate symbol of the text and γ is the maximum number of elastic-degenerate symbols spanned by any occurrence of the pattern in the text. Constants α and γ represent the limits of the algorithm. `ElDeS` combines KMP algorithm (Knuth et al., 1977) and searching in preprocessed suffix trees for potential occurrences spanning the degenerate positions in the text. The suffix trees overhead and their use in recursive extension of the potential occurrences is the possible reason of high search time.

Later, a practical implementation of a pattern matching algorithm in Elastic Degenerate Strings was given in (Grossi et al., 2017). The proposed `EDSM` algorithm works in $O(nm^2+N)$ time. Border tables (Crochemore et al., 2007) are used to reveal pattern prefixes and a suffix tree $\mathcal{ST}_P$ of the pattern to reveal the pattern factors. Potential occurrences are stored in lists and extended over segment borders in left-to-right fashion. KMP is used in elements longer than the pattern length m. The authors also present a bit-vector version of the algorithm where the potential occurrences are stored within a bit register. The bit-version of the algorithm `EDSM-BV` works in $O(N\lceil\frac{m}{w}\rceil)$ time, where w is a size of the computer word (basically, size of the computer registers, usually $w = 64$ bits for standard computers). The practical implementation of the bit-version dominates especially for longer patterns where the original version of the algorithm suffers from the $O(m^2)$ factor. Pissis and Retha (Pissis and Retha, 2018) presented their on-line algorithm for a set of patterns of total length M. Their algorithm achieves $O(N\lceil\frac{M}{w}\rceil)$ time with preprocessing time and space $O(M)$.

Recently, Cisłak et al. (Cisłak et al., 2018) presented `SOPanG` algorithm solving Elastic Degenerate String matching problem in the same upper bound $O(N\lceil\frac{m}{w}\rceil)$, however, achieving an order of magnitude better time than (Grossi et al., 2017) in practical experiments. The algorithm performs standard `Shift-Or` pattern matching on every element of a segment. Resulting search registers (after processing an element) are merged across all elements in each EDS segment. The register then stores all prefixes of the potential occurrences and the algorithm starts to process next EDS segment.

Aoyama et al. (Aoyama et al., 2018) proposed another EDSM on-line algorithm based on efficient sum set computation using Fast Fourier Transform (Cooley and Tukey, 1965). The proposed algorithm improved the time complexity to $O(nm\sqrt{m\log m}+N)$. Bernardini at al. (Bernardini et al., 2019) designed non-combinatorial $O(nm^{1.381}+N)$-time algorithm combining three basic ingredients: a string periodicity argument, Fast Fourier Transform and fast matrix multiplication. In addition, a variant of EDSM algorithm allowing errors was presented in (Bernardini et al., 2017). Unfortunately, we could not compare `BNDM-EDS` with the last three algorithms mentioned in this paragraph due to missing public implementation.

1.1 Our Contribution

We propose a novel on-line algorithm `BNDM-EDS` solving EDSM in $O(Nm\lceil\frac{m}{w}\rceil)$ worst-case time. To the knowledge of the authors, `BNDM-EDS` is the first backward pattern matching algorithm optimized for EDS. The algorithm exploits standard BNDM algorithm for single elements of EDS text. `Shift-And` (Dömölki, 1964) algorithm is used for crossing the borders between two consecutive EDS segments to reveal the occurrences starting in one EDS segment and continuing in the following EDS segment. `BNDM-EDS` is designed to be efficient on real-world data. For small patterns ($m \leq w$), an alphabet size σ, and variability v, the average time complexity is $O(\frac{N(1-v)\log_\sigma m}{m})+O(Nvm(1-v))+O(Nv\beta\delta)$, where $\beta = O(1)$ represents the average number of elements within a degenerate EDS segment and $\delta < m$ represents the average length of an element within a degenerate EDS segment. The assumptions about β and δ are made based on their values in real datasets. Moreover, `BNDM-EDS` beats all its competitors in experiments performed on real genomic data.

1.2 Roadmap

The rest of the paper is organized as follows. We define basic notions in Section 2. In Section 3, we provide exact definition and description of `BNDM-EDS` algorithm. Section 4 summarizes experimental results performed on synthetic and real genomic data. We give the conclusion and some ideas for future work in Section 5.

2 BASIC NOTIONS

Let $x = x_1x_2..x_n$ be a *string* composed of single symbols x_i of a finite ordered alphabet Σ. The length of the string x is $n = |x|$. The size of the alphabet Σ is $\sigma = |\Sigma| = O(1)$. The start position i and the length j define a *factor* (or *substring*) denoted by $x_{i,j} = x_i..x_{i+j-1}$. Factor with $i = 0$ is called *prefix* and a factor with $i + j - 1 = n$ is called *suffix* of the string x. The empty string (of length 0) is denoted by ε.

Definition 1. *Elastic Degenerate String (Grossi et al., 2017)*
An Elastic Degenerate String (EDS) $\tilde{X} = \tilde{X}[0]\tilde{X}[1]...\tilde{X}[n-1]$ of length n on an alphabet Σ is a finite sequence of n degenerate symbols. Every degenerate symbol $\tilde{X}[i]$, for all $0 \leq i < n$, is a non-empty set of strings $\tilde{X}[i][j]$, with $0 \leq j < |\tilde{X}[i]|$, where every $\tilde{X}[i][j]$ is a deterministic string on an alphabet Σ. The degenerate symbols are also called segments. The strings contained in the set corresponding to a segment are also called elements.

N is the total size of an EDS and is defined as

$$N = \sum_{i=0}^{n-1} \sum_{j=0}^{|\tilde{X}[i]|-1} |\tilde{X}[i][j]|$$

Example 1. *Elastic Degenerate String.*
Suppose the following multiple sequence alignment (MSA):

```
G C A A C G G G T A - - A C T
| | | | | | | | | | | | | | |
G C A A C G G G T A T A A C T
| | | × | × | | | | | | | | | |
G C A C C T G G - - - - A C T
```

The resulting Elastic Degenerate String (EDS) $\tilde{T}$ has the following properties:

$$\tilde{T} = \{GCA\}\left\{\begin{array}{c} A \\ C \end{array}\right\}\{C\}\left\{\begin{array}{c} G \\ T \end{array}\right\}\{GG\}\left\{\begin{array}{c} TA \\ TATA \\ \varepsilon \end{array}\right\}\{ACT\}$$

$$N = \sum_{i=0}^{6} \sum_{j=0}^{|\tilde{T}[i]|-1} |\tilde{T}[i][j]| = 3+2+1+2+2+6+3 = 19$$

The length of $\tilde{T}$ corresponds to the number of segments which is $n = 7$. The size of $\tilde{T}$ corresponds to the sum of the length of all elements which is $N = 19$.

A solid string Y is a finite sequence of solid symbols. It means that every position i of the solid string Y is represented just by one symbol of alphabet Σ. A solid string Y matches (Grossi et al., 2017) Elastic Degenerate String $\tilde{X}[0][1]...[n-1]$ of length $n > 1$ (denoted by $Y \approx \tilde{X}$), if and only if string Y can be decomposed into factors $y_0...y_{n-1}, y_i \in \Sigma^*$, such that:

1. there exists a string $s \in \tilde{X}[0]$ such that a suffix of s is $y_0 \neq \varepsilon$;
2. if $n > 2$, there exists $s \in \tilde{X}[i]$, for all $1 \leq i \leq n-2$, such that $s = y_i$;
3. there exists a string $s \in \tilde{X}[n-1]$ such that a prefix of s is $y_{n-1} \neq \varepsilon$.

A string Y has an occurrence in an Elastic Degenerate String $\tilde{T}$ ending at position j if there exists a position $i < j$ such that $Y \approx \tilde{T}[i]...T[j]$, or if there exists an element $s \in \tilde{T}[j]$ such that Y occurs in s. The string matching in Elastic Degenerate Strings is formally defined as Problem 1.

Problem 1. *Elastic Degenerate String Matching (EDSM) (Grossi et al., 2017) Given a string P of length m and an Elastic Degenerate String $\tilde{T}$ of length n and size $N \geq m$, find all positions j in $\tilde{T}$ where at least one occurrence of P ends.*

Example 2. *Elastic Degenerate String Matching. Consider pattern $P = AAC$ of length $m = 3$. Next, consider the following Elastic Degenerate String $\tilde{T}$ of length $n = 7$ and size $N = 19$.*

$$\tilde{T} = \{GCA\}\left\{\begin{array}{c} A \\ C \end{array}\right\}\{C\}\left\{\begin{array}{c} G \\ T \end{array}\right\}\{GG\}\left\{\begin{array}{c} TA \\ TATA \\ \varepsilon \end{array}\right\}\{ACT\}$$

The occurrences are marked with red color. The first occurrence of P starts at position 0 and it ends at position 2. In terms of the first occurrence the pattern P can be (according to the previous description) decomposed into the following three factors: $p_0 = A$, $p_1 = A$ and $p_2 = C$. The second occurrence of P starts at position 5 and it ends at position 6. The second occurrence can be decomposed into the two following factors: $p_0 = A$ and $p_1 = AC$. Notice that the pattern factor p_0 is a suffix of two elements $\tilde{T}[5][0]$ and $\tilde{T}[5][1]$. Despite of that, we account only one occurrence since by EDSM definition we accept only distinct positions in EDS .

3 ALGORITHM

The idea behind `BNDM-EDS` is simple: to exploit shifting capabilities of backward pattern matching algorithms, specifically "good prefix shift" heuristic (Melichar et al., 2005). We chose BNDM (Navarro and Raffinot, 1998) algorithm because of its simplicity and its bit-parallel approach. We adapted BNDM for EDS matching problem in the following way. The elements $\tilde{T}[i][j]$ of each segment $\tilde{T}[i]$ are searched independently using a standard variant of BNDM. Intermediate results (i.e. the discovered pattern prefixes occurring in the last $m-1$ bases of each element $\tilde{T}[i][j]$) must be merged at the end of

each segment $\tilde{T}[i]$. The registers storing the intermediate results are transformed to be suitable for the `Shift-And` algorithm that is used to process the borders of the two following segments $\tilde{T}[i]$ and $\tilde{T}[i+1]$. The `Shift-And` algorithm initialized with the intermediate results from the previous segment processes up to the first m bases of each element from the following segment $\tilde{T}[i+1]$. The point is to discover all potential occurrences spanning two consecutive segments. The `Shift-And` algorithm was chosen since BNDM (generally any backward algorithm) could be forced to process $O(\prod_{k=1}^{m-1} |\tilde{T}[i-k]|)$ different strings when crossing the borders of the segments right-to-left for pathological EDS (i.e. EDS containing many consecutive segments with one-base-long elements).

Algorithm 1 describes the idea of `BNDM-EDS` in more detail. The algorithm assumes pattern length smaller than computer word size ($m \leq w$) for simplicity. The PREPROCESS function performs preprocessing of the pattern P of length m. The first while cycle traverses the pattern from right to left and stores the corresponding byte values (with active bits set to 1) to byte vector $\mathcal{B}$ of length σ that is necessary for BNDM part of the algorithm. Byte vector $\mathcal{R}$ (of length m) stores active prefixes corresponding to BNDM *last* position. The second while cycle traverses the pattern from left to right and stores the corresponding byte values to byte vector $\mathcal{S}$ that is used for the `Shift-And` part of the algorithm. The active bits correspond to the positions of the indexed character in the pattern P in both cases (vector $\mathcal{B}$ and vector $\mathcal{S}$).

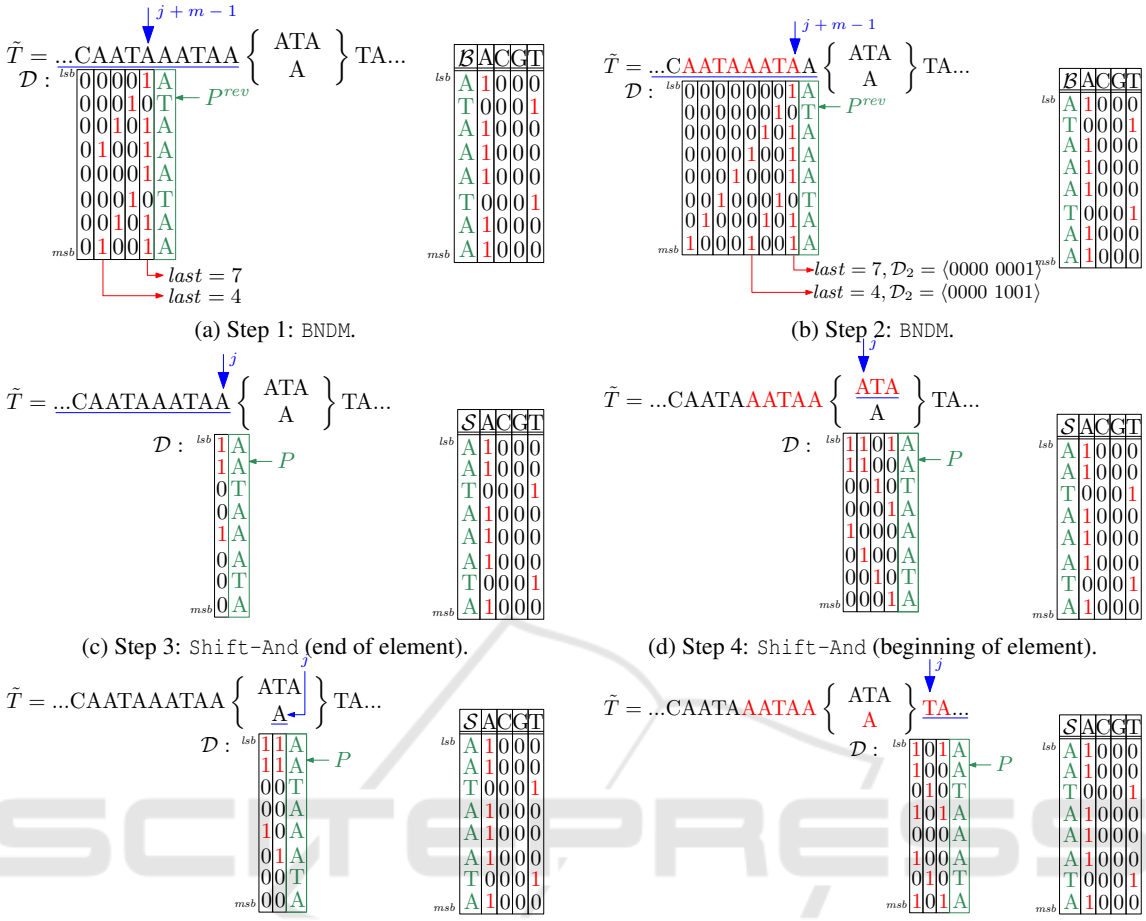
The SEARCH function iterates over individual EDS segments and their elements (see line 15 and line 17). At the beginning of each element e, `Shift-And` algorithm is performed for the first up to m bases (see lines 18 – 23). Elements shorter than m bases continue to line 45 where intermediate results (found pattern prefixes) are merged to the register $\mathcal{R}_2$. BNDM algorithm is performed on elements longer than m bases starting from the position m. The search register $\mathcal{D}$ is initialized with all bits set to 1 and it stores the active bits corresponding to the discovered pattern factors. The condition at line 31 defines that either pattern prefix (subcondition at line 32) or an occurrence (else statement at line 34) was found. Every discovered pattern prefix is also transformed to the corresponding bit position and stored to $\mathcal{D}_2$ register (see line 33) that is later used for the `Shift-And` algorithm performed at the end of each element e. BNDM part of the algorithm is finished when $j+m-1$ exceeds the element boundary (see the while condition at line 26). At last, the `Shift-And` algorithm starts at the position $j+m-last$ and the search register is initialized with the active prefixes discovered in the last iteration of BNDM (see line 38).

Algorithm 1: `BNDM-EDS` preprocessing and searching phase.

```
 1: function PREPROCESS(P, m)
 2:     B[1..σ] ← 0; S[1..σ] ← 0; R[1..m] ← 0;
 3:     i ← m; F ← 1;
 4:     while i > 0 do                          ▷ Building BNDM mask vector
 5:         B[P_i,1] ← B[P_i,1] | F;
 6:         R[i] ← F;
 7:         F ← F ≪ 1; i ← i − 1;
 8:     i ← 1; j ← 1;
 9:     while i ≤ m do                          ▷ Building Shift-And mask vector
10:         S[P_i,1] ← S[P_i,1] | j;
11:         i ← i + 1;
12:         j ← j ≪ 1;

13: function SEARCH(T̃, P, m)
14:     R1 ← 0; R2 ← 0; D2 ← 0; count ← 0; F ← 10^(m−1)
15:     for each segment T̃[i] of T̃ do
16:         R1 ← R2; R2 ← 0;
17:         for each element e ∈ T̃[i] do
18:             j ← 1; D ← R1;
19:             while j < m & j < |e| do    ▷ Shift-And part for the first up to m bases
20:                 D ← ((D ≪ 1) | 1) & S[e_j,1];
21:                 if D & F then
22:                     count ← count + 1;
23:                 j ← j + 1;
24:             if j < m then go to elementEnd;
25:             j ← 1;
26:             while j + m − 1 < |e| do                    ▷ BNDM part
27:                 D2 ← 0; last ← m; i ← m − 1; D ← ∼0;
28:                 while i ≥ 0 & D ≠ 0 do
29:                     D ← D & B[e_j+i,1];
30:                     i ← i − 1;
31:                     if (D & F) ≠ 0 then
32:                         if i ≥ 0 then
33:                             last ← i + 1; D2 ← D2 | R[last];
34:                         else
35:                             count ← count + 1;
36:                     D ← D ≪ 1;
37:                 j ← j + last;
38:             j ← j + m − last; D ← D2;
39:             while j < |e| & D ≠ 0 do    ▷ Shift-And part for the last up to m bases
40:                 D ← ((D ≪ 1) | 1) & S[e_j,1];
41:                 if D & F then
42:                     count ← count + 1;
43:                 j ← j + 1;
44:             elementEnd:
45:             R2 ← R2 | D;
46:     return count;
```

Example 3. *`BNDM-EDS` searching. Figure 1 demonstrates individual steps of `BNDM-EDS` algorithm. Consider EDS fragment $\tilde{T}$ and pattern $P =$ AATAAATA. Each step depicts: (i) the pattern (green color), (ii) the processed element of $\tilde{T}$ (blue underline), (iii) search register $\mathcal{D}$ together with active bits*



(a) Step 1: `BNDM`.

(b) Step 2: `BNDM`.

(c) Step 3: `Shift-And` (end of element).

(d) Step 4: `Shift-And` (beginning of element).

(e) Step 5: `Shift-And` (beginning of element).

(f) Step 6: `Shift-And` (beginning of element).

Figure 1: `BNDM-EDS`: Searching $P = AATAAATA$ in EDS.

(red color), and (iv) preprocessed byte vectors $\mathcal{B}$ and $\mathcal{S}$. Search register $\mathcal{D}$ is depicted for each step. In case of BNDM, the direction is right to left. In case of the `Shift-And`, the direction is left to right. The least significant bit (lsb) is on the top and the most significant bit (msb) is at the bottom. For the sake of clarity, the example starts inside of the first segment with BNDM part of the algorithm (line 25 of Algorithm 1). In the first step, BNDM discovers the longest prefix AATA which implies the shift by last = 4 bases. In the second step, BNDM reports an occurrence (see the diagonal of red bits in the search register $\mathcal{D}$). There are two other potential occurrences since two pattern prefixes A and AATA are discovered. These prefixes are also stored in register $\mathcal{D}_2 = \langle 0000\ 1001 \rangle$. The value of $\mathcal{D}_2$ register (shifted by one to the left) becomes the initial value of search register $\mathcal{D}$ in the next step. In the third step, the `Shift-And` algorithm processes the end of the element. In the fourth step, the `Shift-And` processes the element ATA and discovers the second occurrence. The initial value of search register $\mathcal{D}$ is inherited from the previous segment. In the fifth step, element A is processed by the `Shift-And`. Register $\mathcal{D}$ values from step 4 and 5 are merged and used as the initial value in the next step. The beginning of the last depicted element (TA...) is processed by the `Shift-And` in the last step and the last occurrence is reported.

`BNDM-EDS` time and space complexity can be deduced from Algorithm 1. The preprocessing step consists of two while cycles iterating over bases of the searched pattern P. This implies worst-case time $O(m)$ for preprocessing phase. The needed space comprises of vectors $\mathcal{B}$, $\mathcal{S}$ and $\mathcal{R}$. For patterns smaller that the size of the computer word ($m \leq w$), this implies $O(\sigma + m)$ space. This space can further be reduced to $O(m)$ for small alphabets where $\sigma \leq m$.

`BNDM-EDS` search function consists of three steps performed on all EDS elements. For each element, the first up to m bases and the last up to m bases are processed by the `Shift-And`. Assuming (based on values of real datasets) constant number of elements within one EDS segment, this implies processing up to $2\times$

$n \times m = O(nm)$ bases for small patterns ($m \leq w$) and $O(nm\lceil\frac{m}{w}\rceil)$ for large patterns. These two `Shift-And` steps are represented as while cycles at lines 19 and 39 of Algorithm 1. The third step is BNDM algorithm that needs to process all EDS elements (all bases). This implies processing $O(Nm)$ bases for small patterns ($m \leq w$) and $O(Nm\lceil\frac{m}{w}\rceil)$ for large patterns in the worst case. This step is represented as a while cycle at line 26 of Algorithm 1. Finally, the total worst-case time is $O(Nm+nm) = O(Nm)$ for small patterns and $O(Nm\lceil\frac{m}{w}\rceil+nm\lceil\frac{m}{w}\rceil) = O(Nm\lceil\frac{m}{w}\rceil)$ for large patterns.

We need to define a few parameters to derive the average time complexity for `BNDM-EDS`. Let $v \in [0,1]$ be the variability of a genomic sequence of length N bases. It implies that every $\lceil\frac{1}{v}\rceil$-th base is degenerate. Furthermore, let $\beta = O(1)$ represents the average number of elements within a degenerate EDS segment and let $\delta < m$ represents the average length of an element within a degenerate EDS segment. The number of EDS segments can be defined by the number of bases (which approximately equals the EDS size N) and the variability v: $n \approx 2Nv$. The probability of solid segments of length k bases is defined as: $p_k = v(1-v)^k$. Thus, we can state the number of k-base-long solid segments as: $\frac{n}{2}p_k = Nvp_k = Nv^2(1-v)^k$. BNDM part of the algorithm processes only solid segments (since $\delta < m$) that are longer than m bases. `Shift-And` part of the algorithm processes: (i) all solid segments, (ii) all degenerate segments and their elements.

BNDM average time complexity is claimed to be the same as the BDM average time complexity (Navarro and Raffinot, 1998). Crochemore and Rytter proved BDM average time complexity $O(\frac{n\log_\sigma m}{m})$ in (Crochemore and Rytter, 1994). BNDM part of the algorithm processes $Nv^2(1-v)^k$ solid segments for $k \in [m+1,\infty)$. This implies $Nv^2\sum_{k=m+1}^{\infty}k(1-v)^k$ processed bases. $\sum_{k=m+1}^{\infty}k(1-v)^k$ can by upper-bounded by $\sum_{k=1}^{\infty}k(1-v)^k = \frac{1-v}{v^2}$ which means that BNDM part needs to process $N(1-v)$ bases. Finally, BNDM part of the algorithm works in $O(\frac{N(1-v)\log_\sigma m}{m})$ time.

`Shift-And` part of the algorithm also processes $Nv^2(1-v)^k$ solid segments for $k \in [1,\infty)$. In every solid segment, $O(m)$ bases are processed which is $Nv^2m\sum_{k=1}^{\infty}(1-v)^k = Nvm(1-v)$ processed bases overall. Next, `Shift-And` processes Nv degenerate segments which is $Nv\beta\delta$ processed bases overall. Finally, `Shift-And` part of the algorithm works in $O(Nvm(1-v)) + O(Nv\beta\delta)$ time. We conclude that `BNDM-EDS` average time complexity is $O(\frac{N(1-v)\log_\sigma m}{m}) + O(Nvm(1-v)) + O(Nv\beta\delta)$.

4 EXPERIMENTS

We present experimental results comparing `BNDM-EDS` with other baseline algorithms performed on EDS data. Particularly, we compare our algorithm with `ElDeS` (Iliopoulos et al., 2017), `EDSM-BV` (Grossi et al., 2017) and `SOPanG` (Cisłak et al., 2018). All tested algorithms are optimized for EDS data and were implemented in C/C++ programming language. `BNDM-EDS` implementation is available at https://github.com/stringology-prague/eds_search.

We carried out our tests on Intel® Core™ i7-4740 3.40 GHz, 24 GB RAM, Linux Gentoo kernel 4.19.57. We used compiler gcc version 8.3.0 with compiler optimization -O3. The tested patterns were chosen randomly from the input text and their length m was ranging from 8 to 32. All experiments were run in loop 100 times and we report the mean running time in seconds. All reported times represent measured `user` time + `sys` time. The reported times always include any necessary pattern preprocessing and exclude the time necessary to read the data in memory.

We performed our tests on two different data sets. We include synthetic EDS data from (Grossi et al., 2017) to achieve better comparability. The synthetic data set includes 5 files with n ranging from 100 000 to 1 600 000. The percentage of degenerate positions is 10%. The number of elements within one segment is random and is bounded by 10. The length of each element at the degenerate position is random and it is bounded by 10 bases. Table 1 illustrates the results achieved by the algorithms performed on synthetic data set. The files of the data set are characterized by the following parameters: EDS length n, EDS size N and variability v. The measured times prove the dominance of `BNDM-EDS` and `SOPanG` that is given by their simplicity. `BNDM-EDS` achieved slightly better results than `SOPanG` and this difference is growing (up to 15%) with growing pattern length. This performance gain is caused by BNDM shifting across the searched text at increased rates for longer patterns. The measured results provide two observations: (i) all compared algorithms depend on the size of EDS by factor $O(N)$; (ii) `ElDeS` and `EDSM-BV` require large portion of time for their preprocessing phase. The second observation is analyzed in more detail in Figure 2. We measured ratio of preprocessing time and total time of single algorithms. The results are given in % of the total time. Figure 2 shows that `ElDeS` and `EDSM-BV` need significantly larger portion of the preprocessing time in comparison to `BNDM-EDS` and `SOPanG`. Moreover, the measured ratio of preprocessing time is not decreasing with the growing file size for `ElDeS`.

Table 1: Search time in seconds of BNDM-EDS, ElDeS, EDSM-BV and SOPanG for synthetic data set.

File	File params	Pattern length	BNDM-EDS	SOPanG	EDSM-BV	ELDeS
100000	$n = 100000$ $N = 361546$ $v = 0.095410$	$m = 8$	0.00118260	**0.00113000**	0.02016180	1.80522000
		$m = 16$	**0.00100830**	0.00121400	0.02511590	1.67540000
		$m = 24$	**0.00099149**	0.00113000	0.02863710	1.70129000
		$m = 32$	**0.00098746**	0.00113000	0.02898240	1.76652000
200000	$n = 200000$ $N = 725343$ $v = 0.095325$	$m = 8$	0.00237368	**0.00225000**	0.03969590	3.75465000
		$m = 16$	**0.00202329**	0.00224500	0.04990100	3.94397000
		$m = 24$	**0.00198232**	0.00225400	0.05714240	3.60240000
		$m = 32$	**0.00197887**	0.00225400	0.05706520	3.81179000
400000	$n = 400000$ $N = 1443453$ $v = 0.095015$	$m = 8$	0.00471225	**0.00465500**	0.07850180	8.18302000
		$m = 16$	**0.00402697**	0.00457400	0.09871310	7.81520000
		$m = 24$	**0.00394175**	0.00460800	0.11369500	7.90823000
		$m = 32$	**0.00393171**	0.00461000	0.11334600	7.92150000
800000	$n = 800003$ $N = 2887787$ $v = 0.095186$	$m = 8$	**0.00944679**	0.01000000	0.15683100	16.57840000
		$m = 16$	**0.00804698**	0.00915000	0.19761300	16.06050000
		$m = 24$	**0.00788773**	0.00916900	0.22609000	15.96310000
		$m = 32$	**0.00786342**	0.00915300	0.22549400	16.76730000
1600000	$n = 1600000$ $N = 5769390$ $v = 0.095174$	$m = 8$	0.01884609	**0.01831300**	0.31360500	33.98690000
		$m = 16$	**0.01611825**	0.01824100	0.39423000	35.46980000
		$m = 24$	**0.01581450**	0.01827100	0.45437200	33.81830000
		$m = 32$	**0.01576465**	0.01827700	0.45616800	33.92970000

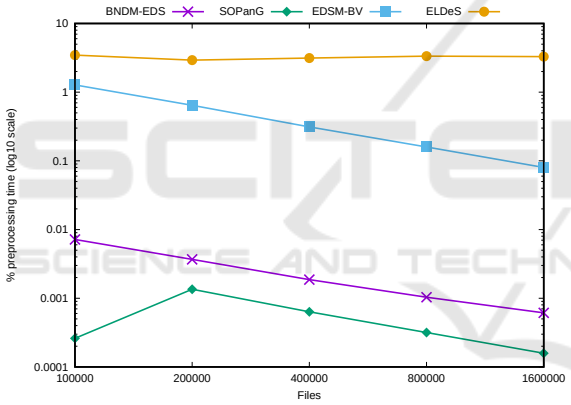


Figure 2: Preprocessing time / Total time ratio for synthetic data set.

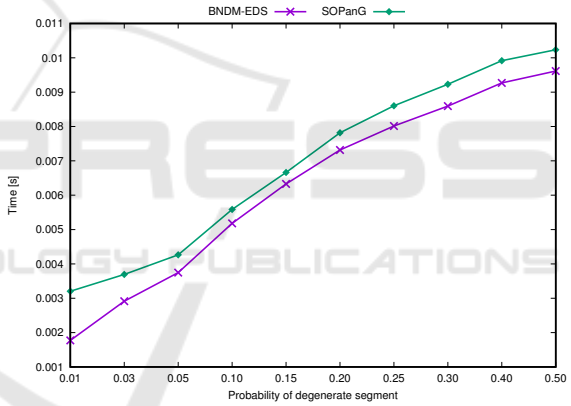


Figure 3: BNDM-EDS and SOPanG comparison on files with different variation.

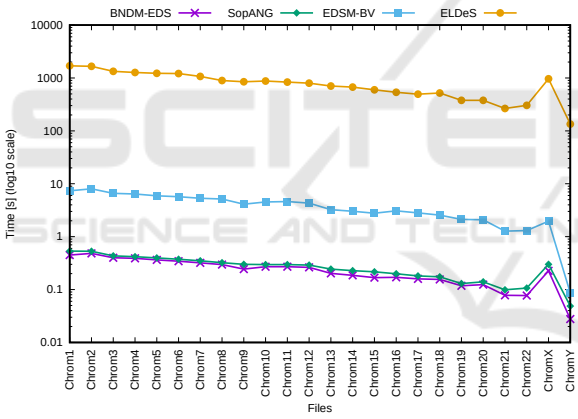
The other data set is composed of real-world human genomic data obtained from 1000 Genomes Projects (Consortium, The 1000 Genomes Project, 2011) (Phase 3 of the project). We downloaded 24 vcf files representing 24 human chromosomes (same population of 2504 individuals for all chromosomes) and transformed them into EDS. The percentage of degenerate positions is approximately 3% on average and the average length of one element (within a degenerate position) is 2 bases. We selected representative chromosomes/files in terms of their size and variabilty and we show the measured times achieved on these files in Table 2. Furthermore, Figure 4 provides the general comparison of individual algorithms for all tested chromosomes/files. Due to large difference of measured times, the vertical axis is in $\log_{10}$ scale.

We can observe stronger BNDM-EDS dominance. This is given by lower percentage of degenerate positions which implies that Shift-And part of the algorithm is not applied so often. This reduces the overhead of swapping between two algorithms and enables better utilization of BNDM shifting capability ("good prefix shift" heuristic). Particularly, this can be seen in case of chromosome Y where the variability among individuals is minimal and the time difference between BNDM-EDS and SopANG increased. BNDM-EDS achieves 43% time improvement in comparison to SopANG for chromosome Y and pattern length $m = 8$. However, for longer patterns $m = 32$, BNDM-EDS achieves 81% time improvement. Generally, the time achieved for single chromosomes is given especially by their size that corresponds to the N element in the time com-

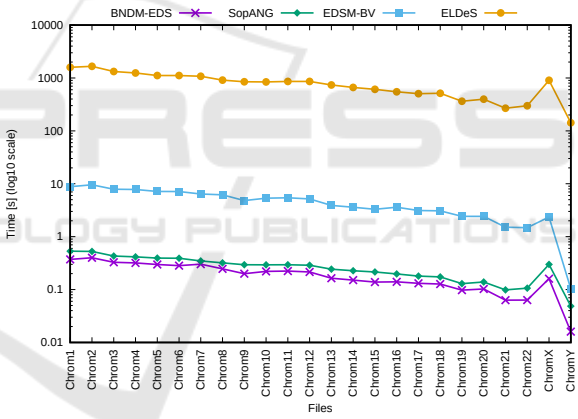
Table 2: Search time in seconds of BNDM-EDS, ElDeS, EDSM-BV and SOPanG for real data set.

File	File params	Pattern length	BNDM-EDS	SOPanG	EDSM-BV	ELDeS
Chrom1	$n = 248\,458\,108$ $N = 255\,872\,195$ $v = 0.025814$	$m = 8$	**0.44913196**	0.53431100	7.35913000	1695.14000000
		$m = 16$	**0.37187386**	0.53086700	8.80502000	1590.15000000
		$m = 24$	**0.35176826**	0.53060000	9.35136000	1590.61000000
		$m = 32$	**0.35678421**	0.53099200	9.68736000	1688.14000000
Chrom7	$n = 158\,984\,673$ $N = 164\,380\,710$ $v = 0.029508$	$m = 8$	**0.31989103**	0.34962400	5.32563000	1070.06000000
		$m = 16$	**0.30178564**	0.34705100	6.40293000	1077.33000000
		$m = 24$	**0.24784241**	0.34708700	6.84947000	1035.08000000
		$m = 32$	**0.25140232**	0.34661600	7.09477000	1068.89000000
Chrom22	$n = 50\,713\,670$ $N = 52\,004\,687$ $v = 0.021605$	$m = 8$	**0.07712649**	0.10663000	1.30624000	304.21900000
		$m = 16$	**0.06338260**	0.10615300	1.47754000	298.53900000
		$m = 24$	**0.05970330**	0.10614700	1.62679000	277.77800000
		$m = 32$	**0.05970250**	0.10603900	1.63156000	315.12800000
ChromX	$n = 155\,619\,941$ $N = 157\,232\,908$ $v = 0.009292$	$m = 8$	**0.22626265**	0.30025500	1.96430000	963.23200000
		$m = 16$	**0.16053621**	0.29899700	2.35361000	903.40400000
		$m = 24$	**0.14193202**	0.29896900	2.48494000	928.85500000
		$m = 32$	**0.14133996**	0.29900700	2.55960000	943.67600000
ChromY	$n = 26\,622\,695$ $N = 26\,629\,512$ $v = 0.000245$	$m = 8$	**0.02775240**	0.04873900	0.08626860	134.60100000
		$m = 16$	**0.01600782**	0.04860800	0.10365000	142.23600000
		$m = 24$	**0.01156795**	0.04859900	0.10476000	136.01800000
		$m = 32$	**0.00933492**	0.04861900	0.10436000	134.60100000

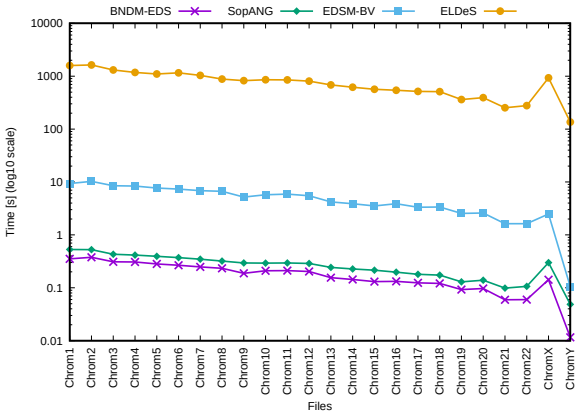
(a) $m = 8$.



(b) $m = 16$.



(c) $m = 24$.



(d) $m = 32$.

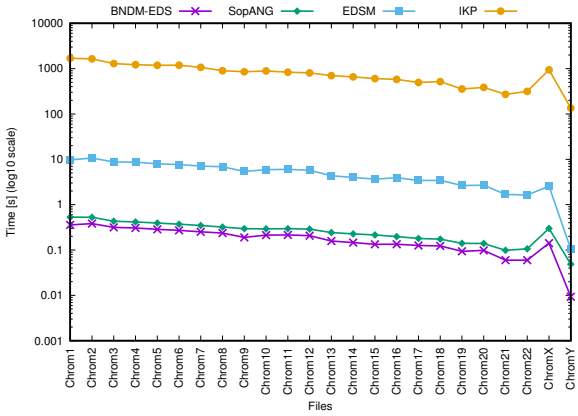


Figure 4: Search time of BNDM-EDS, ElDeS, EDSM-BV and SOPanG for real data set.

plexity of single algorithms.

Since BNDM-EDS gain in comparison to SopANG clearly depends on variability in the processed EDS, we included another experiment comparing both algorithms on synthetic data with different probability of degenerate EDS segment. Figure 3 depicts the achieved search time in seconds. The probability of degenerate EDS segment in single files starts from 0.01 and continues up to 0.5. The size of the files is 1 600 000 bases and we searched for a randomly chosen pattern of length $m = 16$. The results show that starting from 0.05 probability of degenerate segment, the length of processed elements (of the solid segments) is shorter than the pattern length $m = 16$. This implies that BNDM part of BNDM-EDS is not utilized for most of the processed elements.

5 CONCLUSION AND FUTURE WORK

We proposed BNDM-EDS algorithm which is the first backward pattern matching algorithm for Elastic Degenerate Strings. Its worst-case time is bounded by $O(Nm\lceil\frac{m}{w}\rceil)$. Moreover, for small patterns ($m \leq w$), BNDM-EDS achieves average time complexity $O(\frac{N(1-v)\log_\sigma m}{m}) + O(Nvm(1-v)) + O(Nv\beta\delta)$ which is optimal. Our goal was to design a simple (easy-to-implement) algorithm achieving fast search times on real-world data. This expectation was confirmed by experiments on real-world data set. BNDM-EDS proves its superiority especially on data with lower degenerate rate and for longer patterns.

We plan to extend BNDM-EDS for protein alphabet in future work. Furthermore, we want to focus on processing more general variants of Elastic Degenerate Strings, such as recursive Elastic Degenerate Strings or colored Elastic Degenerate Strings (allowing to map elements to individuals of the sequenced population).

ACKNOWLEDGEMENTS

The research was partially supported by the Czech Science Foundation as project No. 19-20759S.

REFERENCES

Aoyama, K., Nakashima, Y., I, T., Inenaga, S., Bannai, H., and Takeda, M. (2018). Faster Online Elastic Degenerate String Matching. In Navarro, G., Sankoff, D., and Zhu, B., editors, *Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*, volume 105 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:10, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Bernardini, G., Gawrychowski, P., Pisanti, N., Pissis, S. P., and Rosone, G. (2019). Even faster elastic-degenerate string matching via fast matrix multiplication. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece.*, pages 21:1–21:15.

Bernardini, G., Pisanti, N., Pissis, S. P., and Rosone, G. (2017). Pattern matching on elastic-degenerate text with errors. In Fici, G., Sciortino, M., and Venturini, R., editors, *String Processing and Information Retrieval*, pages 74–90, Cham. Springer International Publishing.

Church, D. M. et al. (2015). Extending reference assembly models. *Genome Biology*, 16(13).

Cisłak, A., Grabowski, S., and Holub, J. (2018). Sopang: online text searching over a pan-genome. *Bioinformatics*, page bty506.

Consortium, T. C. P.-G. (2016). Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135.

Consortium, The 1000 Genomes Project (2011). A map of human genome variation from population-scale sequencing. *Nature*, 473:544 EP –. Corrigendum.

Consortium, The UK10K (2015). The uk10k project identifies rare variants in health and disease. *Nature*, 526:82 EP –.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301.

Crochemore, M., Hancart, C., and Lecroq, T. (2007). *Algorithms on Strings*. Cambridge University Press, New York, NY, USA.

Crochemore, M. and Rytter, W. (1994). *Text algorithms*. Oxford University Press.

Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., McVean, G., and Durbin, R. (2011). The variant call format and vcftools. *Bioinformatics*, 27(15):2156–2158.

Dilthey, A. et al. (2015). Improved genome inference in the mhc using a population reference graph. *Nature Genetics*, 47:682.

Dömölki, B. (1964). An algorithm for syntactical analysis. *Computational Linguistics*, 3:29–46. Hungarian Academy of Science, Budapest.

Grossi, R., Iliopoulos, C. S., Liu, C., Pisanti, N., Pissis, S. P., Retha, A., Rosone, G., Vayani, F., and Versari, L. (2017). On-line pattern matching on similar texts. In *28th Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, pages 9:1–9:14.

Iliopoulos, C. S., Kundu, R., and Pissis, S. P. (2017). Efficient pattern matching in elastic-degenerate texts. In Drewes, F., Martín-Vide, C., and Truthe, B., editors, *Language and Automata Theory and Applica-*

tions, pages 131–142, Cham. Springer International Publishing.

Knuth, D. E., Morris, J. H., and Pratt, V. R. (1977). Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6(2):323–350.

Maciuca, S. et al. (2016). A natural encoding of genetic variation in a burrows-wheeler transform to enable mapping and genome inference. In *Algorithms in Bioinformatics*, pages 222–233, Cham. Springer International Publishing.

Marschall, T. (2018). Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135.

Melichar, B., Holub, J., and Polcar, J. (2005). *Text Searching Algorithms*.

Na, J. C., Kim, H., Park, H., Lecroq, T., Léonard, M., Mouchard, L., and Park, K. (2016). Fm-index of alignment: A compressed index for similar strings. *Theoretical Computer Science*, 638:159 – 170. Pattern Matching, Text Data Structures and Compression.

Navarro, G. and Pereira, A. O. n. (2016). Faster compressed suffix trees for repetitive collections. *J. Exp. Algorithmics*, 21:1.8:1–1.8:38.

Navarro, G. and Raffinot, M. (1998). A bit-parallel approach to suffix automata: Fast extended string matching. In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching*, CPM '98, pages 14–33, London, UK, UK. Springer-Verlag.

on Biochemical Nomenclature (CBN), I.-I. C. (1971). Abbreviations and symbols for nucleic acids, polynucleotides and their constituents. *Journal of Molecular Biology*, 55(3):299 – 310.

Pissis, S. P. and Retha, A. (2018). Dictionary Matching in Elastic-Degenerate Texts with Applications in Searching VCF Files On-line. In D'Angelo, G., editor, *17th International Symposium on Experimental Algorithms (SEA 2018)*, volume 103 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:14, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Procházka, P. and Holub, J. (2014). Compressing similar biological sequences using fm-index. In *Data Compression Conference, DCC 2014, Snowbird, UT, USA, 26-28 March, 2014*, pages 312–321.

Sirén, J. (2016). Indexing variation graphs. *CoRR*, abs/1604.06605.