Enhancing Characters Distance Text Sampling by Condensed Alphabets

Simone Faro[1], Francesco Pio Marino[1], Arianna Pavone[2]

[1] Dipartimento di Matematica e Informatica,
Università di Catania, viale A.Doria n.6, 95125, Catania, Italia
faro@dmi.unict.it
[2] Dipartimento di Scienze Cognitive,
Università di Messina, via Concezione n.6, 98122, Messina, Italia
apavone@unime.it

Abstract. *Sampled string matching* is an efficient approach to the string matching problem introduced in order to overcome the prohibitive space requirements of indexed matching, on the one hand, and drastically reduce searching time for the online solutions, on the other hand. Known solutions to sampled string matching are able to speed up the searching up to 9 times while using less than 11% of the text size. However, they appear to work efficiently only in the case of natural language texts or, in general, when searching on input sequences over large alphabets.
In this paper we extend sampled-string matching to the case of small alphabets obtaining a new efficient solution which turns out to be feasible also for searching biological data like genome or protein sequences. Our solution extends a recent approach called Character Distance Sampling by using condensed characters in order to enlarge the size of the alphabet and speed up the searching process. From our experimental results it turns out that our solution significantly reduces the searching time when compared against previous sampled string matching algorithms. In particular on biological data it leads to reduce the space consumption up to 80% and to speed up the searching up to 96%, thus improving the standard online string matching algorithm up to 99.6% using less than 1% of the original text size.

1 Introduction

Exact string matching is a fundamental problem in computer science and in the wide domain of text processing. It consists in finding all the occurrences of a given pattern x, of length m, in a large text y, of length n, where both sequences are composed by characters drawn from an alphabet Σ of size σ.

It is a critical problem in computational molecular biology and plays a very important role in biological sequences analysis, mainly due to the constantly growing amount of molecular data extracted from living organisms. For this reason sequence matching techniques play a very important role in various applications in computational biology for data analysis.

As the size of data increases the space needed to store it is constantly increasing too, for this reasons the need for new efficient approaches to the problem.

Applications require two kinds of solutions: *online* and *offline* string matching. Solutions based on the first approach assume that the text is not preprocessed and thus they need to scan the input sequence *online*, when searching. Their worst case time complexity is $\Theta(n)$, and was achieved for the first time by the well known Knuth-Morris-Pratt (KMP) algorithm [13], while the optimal average time complexity of the problem is $\Theta(n \log_\sigma m/m)$ [16], achieved for example by the Backward-Dawg-Matching (BDM) algorithm [5].

Many string matching solutions have been also developed in order to obtain sub-linear performance in practice [7]. Among them the Boyer-Moore-Horspool (BMH) algorithm [1,12] deserves a special mention, since it has inspired much work.

Memory requirements of this class of algorithms are very low and generally limited to a precomputed table of size $O(m\sigma)$ or $O(\sigma^2)$ [7]. However their searching time is always proportional to the length of the text and thus their performances may stay poor in many practical cases, especially for huge texts and short patterns.

Solutions based on the second approach try to drastically speed up searching by preprocessing the text and building a data structure that allows searching in time proportional to the length of the pattern. This method is called *indexed searching* [7,1,11]. However, despite their optimal time performances, space requirements of such data structures are from 4 to 20 times the size of the text, which may be too large for many practical applications.

Leaving aside other different approaches, like those based on *compressed string matching* [14,2], which turn out to be theoretically efficient and are hardly feasible in practical cases, an alternative solution to the problem is *sampled string matching*, introduced in 1991 by Vishkin [15], which consists in the construction of a succinct sampled version of the text (which must be maintained together with the original text) and in the application of any online string matching algorithm directly on the sampled sequence. Although any candidate occurrence of the pattern may be found more efficiently, the drawback of this approach is that any occurrence reported in the sampled-text requires to be verified in the original text. Apart from this point a sampled-text approach may have a lot of good features: it may be easy to implement if compared with other succint matching approaches, it may require very small extra space and may allow fast searching. Additionally it may also allow fast updates of the data structure.

Apart the theoretical result of Vishkin, the first practical solution to sampled string matching has been introduced by Claude *et al.* [4] and is based on an alphabet reduction. Their solution has an extra space requirement which is only 14% of text size and turns out to be up to 5 times faster than standard online string matching on English texts. In this paper we refer to this algorithm as Occurrence Text Sampling (OTS).

More recently Faro *et al.* presented a more effective sampling approach based on *character distance sampling* (CDS) [10,9], obtaining in practice a speed up by a factor of up to 9 on English texts, using limited additional space whose amount goes from 11% to 2.8% of the text size, with a gain in searching time up to 50% if compared against the previous solution. However the previous sampling approaches to exact string matching prove to work efficiently only in the case of natural language texts or, in general, when searching on input sequences over large alphabets, while their performances degrade when the size of the underlying alphabets decreases.

In this paper, we present an extension of the approach proposed by Faro *et al.* [10] to small alphabets which turns out to be much more feasible in the case of biological data like genome or protein sequences and, in general, in the case of small alphabets. Our proposed approach makes use of condensed characters in order to enlarge the size of the underlying alphabet and, as a result, speed up the searching process and reduce the space consumption of the resulting sampled text. From our experimental results it turns out that the use of condensed alphabets leads to reduce the space consumption up to 80% and to speed up the searching process up to 95%, significantly improving the results obtained by the original text sampling technique. In addition, the new approach, while designed to work well with text on small alphabets, is also particularly effective on large alphabets. Our experimental results prove how the approach based on condensed characters significantly improves performance even in the case of natural language texts.

The paper is organized as follows. In Section 2 we present the Characters Distance Sampling approach introduced by Faro *et al.* [10] and we extend it to condensed alphabets in Section 2.1. In Section 3 we describe the three procedures used for searching a pattern in a text using the sampled text extended with condensed alphabets. Then in Section 4 we present experimental results and draw our conclusions in Section 5.

2 CDS and Condensed Alphabets

Let y be the input text, of length n, and let x be the input pattern, of length m, both over an alphabet Σ of size σ. We assume that all strings can be treated as vectors starting at position 1. Thus we refer to $x[i]$ as the i-th character of the string x, for $1 \leq i \leq m$, where m is the size of x.

We elect a set $C \subseteq \Sigma$ to be the *set of pivot characters*. Given this set of pivot characters we sample the text y by taking into account the distances between consecutive positions of any pivot characters $c \in C$ in y. More formally our sampling approach is based on the following definition of *position sampling* of a text.

Definition 1 (Position Sampling). *Let y be a text of length n, let $C \subseteq \Sigma$ be the set of pivot characters and let n_C be the number of occurrences of any $c \in C$ in the input text y.*

First we define the position function, $\delta : \{1, .., n_C\} \to \{1, .., n\}$, *where* $\delta(i)$ *is the position of the* i*-th occurrence of any character of* C *in* y*. Formally we have*

$$\begin{array}{ll} (i)\ 1 \leq \delta(i) < \delta(i+1) \leq n & \text{for each } 1 \leq i \leq n_C - 1 \\ (ii)\ y[\delta(i)] \in C & \text{for each } 1 \leq i \leq n_C \\ (iii)\ y[\delta(i)+1..\delta(i+1)-1] \text{ contains no chars of } C & \text{for each } 0 \leq i \leq n_C \end{array}$$

where in (iii) *we assume that* $\delta(0) = 0$ *and* $\delta(n_C + 1) = n + 1$*.*

Then the position sampled version of y*, indicated by* $\dot{y}$*, is a numeric sequence, of length* n_C*, defined as*

$$\dot{y} = \langle \delta(1), \delta(2), .., \delta(n_C) \rangle. \tag{1}$$

Example 1. Suppose $y =$ "agaacgcagtata" is a DNA sequence of length 13, over the alphabet $\Sigma = \{\text{a,c,g,t}\}$. Let C = {a} be the set of pivot characters. Thus the position sampled version of y is $\dot{y} = \langle 1, 3, 4, 8, 11, 13 \rangle$. Specifically the first occurrence of character $c \in C$ is at position 1 ($y[1] =$ a), its second occurrence is at position 3 ($y[3] =$ a), and so on.

Definition 2 (Characters Distance Sampling). *Let* $C \subseteq \Sigma$ *be the set of pivot characters, let* $n_C \leq n$ *be the number of occurrences of any pivot character in the text* y *and let* δ *be the position function of* y*. We define the* characters distance function $\Delta(i) = \delta(i+1) - \delta(i)$*, for* $1 \leq i \leq n_C - 1$*, as the distance between two consecutive occurrences of any pivot character in* y*.*

Then the characters-distance sampled version *of the text* y *is a numeric sequence, indicated by* $\bar{y}$*, of length* $n_C - 1$ *defined as*

$$\begin{aligned} \bar{y} &= \langle \Delta(1), \Delta(2), .., \Delta(n_C - 1) \rangle \\ &= \langle \delta(2) - \delta(1), .., \delta(n_C) - \delta(n_C - 1) \rangle \end{aligned} \tag{2}$$

Plainly we have

$$\sum_{i=1}^{n_C - 1} \Delta(i) \leq n - 1.$$

Example 2. Let $y =$ "agaacgcagtata" be a text of length 13, over the alphabet $\Sigma = \{\text{a,c,g,t}\}$. Let $C = \{a\}$ be the set of pivot characters. Thus the character distance sampling version of y is $\bar{y} = \langle 2, 1, 4, 3, 2 \rangle$. Specifically $\bar{y}[1] = \Delta(1) = \delta(2) - \delta(1) = 3 - 1 = 2$, while $\bar{y}[3] = \Delta(3) = \delta(4) - \delta(3) = 8 - 4 = 4$, and so on.

Definition 3 (Rank of a character). *Let* x *be a pattern of length* m*, and let* $c \in \Sigma$*. We define* $\phi : \Sigma \to \{0..m\}$ *as the function which associates any character of the text with the number of its occurrences in* x*. The rank of the character* c *is the position of* c *in the alphabet* Σ*, if we assume that all characters are sorted by their* $\phi(c)$ *values in decreasing order. More formally the rank of* c *is given by the cardinality of the set* $\{k \in \Sigma \mid \phi(k) > \phi(c)\} + 1$

2.1 Extension to Condensed Alphabets

Let y be an input sequence, of length n, over an alphabet Σ of size σ. Given a constant parameter q, with $1 \leq q < n$, we define the condensed alphabet $\Sigma_y^{(q)}$, related to y, as

$$\{c \in \Sigma^q \mid c = y[i..i+q-1] \text{ for some } 1 \leq i \leq n-q+1\}$$

Roughly speaking $\Sigma_y^{(q)}$ is the set of all different subsequences of length q (or q-grams) appearing in y. We define the q-condensed version of y as follows.

Definition 4 (q-Condensed Sequence). *Let y be a text of length n over an alphabet Σ of size σ and let $\Sigma_y^{(q)}$ be the condensed alphabet, related to y, for a given constant parameter q. We define the q-condensed version of the sequence y as the sequence, of length $n-q+1$, of all consecutive (and overlapping) substrings of length q appearing in y. More formally*

$$y^{(q)} = \langle y[1..q], y[2..q+1], y[3..q+2], .., y[n-q+1..n] \rangle$$

Example 3. Assume $y =$ "agtagcgcagt" is a DNA sequence of length 11, over the alphabet $\Sigma = \{$a,c,g,t$\}$. Then we have

$$\begin{aligned} y^{(2)} &= \langle ag, gt, ta, ag, gc, cg, gc, ca, ag, gt \rangle \\ y^{(3)} &= \langle agt, gta, tag, agc, gcg, cgc, gca, cag, agt \rangle \\ y^{(4)} &= \langle agta, gtag, tagc, agcg, gcgc, cgca, gcag, cagt \rangle \end{aligned}$$

Definition 5 (q-Characters Distance Sampling). *Let $C \subseteq \Sigma_y^{(q)}$ be the set of pivot characters, let $n_C \leq n$ be the number of occurrences of any pivot character in the text $y^{(q)}$ and let δ be the position function of $y^{(q)}$. We define the* q-characters distance function *$\Delta^{(q)}$ as the distance between two consecutive occurrences of any pivot character in $y^{(q)}$, where $\Delta^{(q)}(i)$, for $1 \leq i \leq n_C - 1$, is the distance between the $(i+1)$-th and the i-th occurrence of any occurrences of any pivot character in $y^{(q)}$.*

Then the q-characters-distance sampled version *of the sequence y is a numeric sequence of length $n_C - 1$, indicated by $\bar{y}^{(q)}$ and defined as*

$$\bar{y}^{(q)} = \langle \Delta^{(q)}(1), \Delta^{(q)}(2), .., \Delta^{(q)}(n_C - 1) \rangle. \tag{3}$$

Example 4. As in the previous Example 3 assume $y =$ "agtagcgcagtagta" is a DNA sequence of length 15, over the alphabet $\Sigma = \{$a,c,g,t$\}$. If we suppose $q = 2$ and $C =$ "ag'' is the set of pivot characters, then we have

$$\begin{aligned} \dot{y}^{(2)} &= \langle 1, 4, 9, 12 \rangle \\ \bar{y}^{(2)} &= \langle 3, 5, 3 \rangle. \end{aligned}$$

Similarly, if we suppose $q = 3$ and $C = \{$"agt" $\}$ is the set of pivot characters, then we have

$$\begin{aligned} \dot{y}^{(3)} &= \langle 1, 9, 12 \rangle \\ \bar{y}^{(3)} &= \langle 8, 3 \rangle. \end{aligned}$$

3 The Sampled Text Searching Algorithm

Let y be an input text of length n over an alphabet Σ of size σ, let $q > 1$ and let $\Sigma_y^{(q)}$ be the condensed alphabet over Σ. In addition let $C \subseteq \Sigma^{(q)}$ be the set of pivot characters.

In this paper we do not go into the way for a correct selection of the set of pivot characters, and even we leave the details of an analysis about what is the best subset to be chosen. However in our experimental evaluation (see Section 4) we will show how it is enough to put a single character in the set of pivot characters. Such character is selected on the basis of its *rank value*, where we remember that the rank of a character c corresponds to its position in the alphabet Σ when we assume that all characters are sorted by their frequencies inside the text (see Definition 3).

During the preprocessing phase the algorithm performs a scanning of the text y and builds the corresponding position sampled text $\dot{y}^{(q)}$. Assuming that the maximum distance between two consecutive occurrences of the pivot character is bounded by γ, by Definition 5, the sequences $\dot{y}^{(q)}$ and $\bar{y}^{(q)}$ require $(n_C)\log(n)$ and $(n_C)\log(\gamma)$ bits, respectively, to be maintained.

Let now x be an input pattern of length m and let m_C be the number of occurrences of any pivot character in x. The searching phase can be then divided in three different subroutines, depending on the value of m_C. All searching procedures work using a filtering approach. The idea behind such searching procedures is to take advantage of the sampled text $\dot{y}^{(q)}$ computed during the preprocessing phase in order to quickly locate any candidate substring s of the original text which may include an occurrence of the pattern.

If such candidate substring s has length m then the algorithm simply performs a character-by-character comparison between the pattern and the substring. Otherwise if the candidate substring s has length greater than m, then a searching procedure is called, based on a standard exact online string matching algorithm, for searching the pattern x in s.

We can prove that if we suppose the underlying algorithm to be characterized by a linear worst case time complexity, as in the case of the KMP algorithm [13], then our solution achieves the same complexity in the worst case. Similarly if we implement the underlying searching procedure using an optimal average string matching algorithm, like the BDM algorithm [12], the resulting solution achieves an optimal $O(n \log_\sigma m/m)$ average time complexity.

In what follows we describe in details the three different searching procedures which are applied when $m_C = 0$, $m_C = 1$ and $m_C > 1$, respectively. We will refer to the pseudo-codes of procedures SEARCH-0, SEARCH-1 and SEARCH-2^+ as shown in Figure 1.

Case 1: $m_C = 0$
If the pattern contains no occurrence of any pivot characters, we have that m_C is equal to 0. Under this assumption the algorithm searches for the pattern x in all substrings of the original text which do not contain the pivot characters.

```
SEARCH-0(x, ẏ^(q), y, q)
1. m ← len(x)
2. n_C ← len(ẏ)
3. ẏ^(q)[0] ← 0
3. ẏ^(q)[n_C + 1] ← n − q + 2
4. for i ← 1 to n_C + 1 do
5.       if (ẏ^(q)[i] − ẏ^(q)[i − 1] + q − 2 ≥ m) then
6.             l ← ẏ^(q)[i − 1] + 1
7.             r ← ẏ^(q)[i] + q − 2
8.             search for x in y[l..r]

SEARCH-1(x, ẏ^(q), y, q)
1.  m ← len(x)
2.  n_C ← len(ẏ)
3.  α ← min{i : x^(q)[i] ∈ C}
4.  ẏ^(q)[0] ← 0
5.  ẏ^(q)[n_C + 1] ← n − q + 2
6.  for i ← 1 to n_C + 1 do
7.        if (ẏ^(q)[i − 1] − ẏ^(q)[i − 2] > α − 1 and
              ẏ^(q)[i] − ẏ^(q)[i − 1] > m − α) then
8.             l ← ẏ^(q)[i − 1] − α + 1
9.             r ← ẏ^(q)[i − 1] − α + m
10.            compare x and y[l..r]

SEARCH-2^+(x, ẏ^(q), y, q)
1. m ← len(x)
2. (x̄^(q), m̄) ← COMPUTE-DISTANCE-SAMPLING(x, m, C)
3. search for x̄^(q) in ȳ^(q) :
4.       for each i such that x̄^(q) = ȳ^(q)[i..i + m̄ − 1] do
5.             l ← ẏ^(q)[i] − ẏ^(q)[0]
6.             r ← ẏ^(q)[i] + m − 1
7.             compare x and y[l..r]
```

Fig. 1. The pseudocode of procedures SEARCH-0, SEARCH-1 and SEARCH-2^+, respectively, for the sampled string matching problem.

Specifically such substrings are identified in the original text by the intervals $[\delta^{(q)}(i) + 1..\delta^{(q)}(i + 1) + q - 2]$, for each $0 \leq i \leq n_C$, assuming $\delta^{(q)}(0) = 0$ and $\delta^{(q)}(n_C + 1) = n - q + 2$.

Specifically, for each $1 \leq i \leq n_C + 1$, the algorithm checks if the value $\dot{y}^{(q)}(i) - \dot{y}^{(q)}(i-1) + q - 2$ is greater or equal to m. In such a case the algorithm searches for x in the substring of the text $y[\dot{y}^{(q)}[i-1]+1..\dot{y}^{(q)}[i]+q-2]$ using any standard string matching algorithm. Otherwise the substring is skipped, since no occurrence of the pattern could be found at such position.

Case 2: $m_C = 1$
If the pattern x contains a single occurrence of any character of the set C, then the length of the sampled version of the pattern is still equal to 0. However also in this case the algorithm is able to efficiently take advantage of the information precomputed in $\dot{y}^{(q)}$ using the positions of the pivot character in $y^{(q)}$ as an anchor to locate all candidate occurrences of x.

Specifically, let α be the unique position in x which contains the pivot character, i.e. we assume that $x[\alpha..\alpha+q-1] = c$ and that both $x[1..\alpha-1]$ and $x[\alpha+1..m]$ do not contain any pivot character. Then, for each $0 \leq i \leq n_C - 1$, the algorithm checks if the value $\dot{y}^{(q)}(i-1) - \dot{y}^{(q)}(i-2)$ is greater than $\alpha - 1$ and if the value $\dot{y}^{(q)}(i) - \dot{y}^{(q)}(i-1)$ is greater than $m - \alpha$.

In such a case the algorithm merely checks if the substring of the text $y[\dot{y}^{(q)}[i-1] - \alpha + 1..\dot{y}^{(q)}[i-1] - \alpha + m]$ is equal to the pattern. Otherwise the substring is skipped. As before we assume that $\dot{y}(0) = 0$ and $\dot{y}(n_C + 1) = n + 1$.

Case 3: $m_C \geq 2$
If the number of occurrences of any pivot character in C is greater than 1 then the algorithm uses the sampled text $\dot{y}^{(q)}$ to compute on the fly the sampled version $\bar{y}^{(q)}$ of $y^{(q)}$ and use it to search for any occurrence of $\bar{x}^{(q)}$. This is used as a filtering phase for locating in y any candidate occurrence of x.

First the character distance sampled version $\bar{x}$ of x is computed. Then the algorithm searches for $\bar{x}$ in $\bar{y}$ using any exact online string matching algorithm. Notice that $\bar{y}$ can be efficiently retrieved online from the sampled text $\dot{y}$, using relation given in (2).

For each candidate occurrence i of $\bar{x}$ located in $\bar{y}$, an additional procedure must be run to check if such occurrence corresponds to a match of the whole pattern x in y. For this purpose the algorithm checks if the substring of the text $y[\dot{y}^{(q)}[i] - \dot{y}^{(q)}[0]..\dot{y}^{(q)}[i] + m - 1]$ is equal to x, where $\dot{y}^{(q)}[0]$ is the position of the first occurrence of the pivot character into the pattern.

Regarding the time complexity we can prove that, assuming an underlying auxiliary string matching algorithm with a linear worst case and a $O(n \log m/m))$ average case time complexity, the resulting algorithm based on Character Distance Sampling achieves an optimal $O(n)$ time complexity in the worst-case and a $O(n \log m/m))$ time complexity in the average case.

4 Experimental Results

In this section we briefly present experimental results obtained by comparing the new proposed algorithms (with values of q ranging between 2 and 4) against the standard distance sampling algorithm (obtained with q set to 1). We also compare our algorithms against the Occurrence Text Sampling algorithms implemented using q-grams. Also in this case we used values of q ranging between 2 and 4.

Following the same lines of previous papers on sampled string matching [4,10] we tested all sampling solutions in combination with the Boyer-Moore-Horspool

algorithm [12] for the implementation of the underlying standard searching procedure. As a consequence, in our comparison we also included the Boyer-Moore-Horspool string matching algorithm (in its standard implementation) in order to understand how much the proposed sampling approach contributes to speed-up a standard online string matching solution.[3] Results are compared in terms of space consumption and searching speed.

All algorithms have been implemented using the C programming language, and have been tested using a variant of the SMART[4] tool [8] properly tuned for testing string matching algorithms based on a text-sampling. Tests have been executed on a MacBook Pro with 4 Cores, a 2.7 GHz Intel Core i7 processor, 16 GB RAM 2133 MHz LPDDR3, 256 KB of L2 Cache and 8 MB of Cache L3.

The algorithms have been tested on two 5MB text buffers containing real biological sequences. Specifically we used a genome sequence containing base pairs of Escherichia coli (with $\sigma = 4$). from the Large Canterbury Corpus (`http://www.data-compression.info/Corpora/CanterburyCorpus/`). The sequence is also available within the SMART tool.

In our implementation we selected the pivot character on the basis of its *rank value*, where we remember that the rank of a character c is the position of c in the alphabet Σ, if we assume that all characters are sorted by their frequencies inside the text (see Definition 3).

Then we evaluated the behaviour of our algorithms for different values of the rank r of the selected pivot character and specifically for r ranging between 1 (the most frequent character) and 16. Observe that if σ is the size of the original alphabet Σ, then σ^q is the size of the condensed alphabet $\Sigma^{(q)}$. As a consequence, in the case of experimental tests on genome sequences and $q = 1$, the value of the rank r is limited in the range between 1 and 4, since 4 is the size of the alphabet.

4.1 Space Requirements

In the context of text-sampling string-matching space requirement is one of the most significant parameter to take into account. It indicates how much additional space, with regard to the size of the original input sequences, is required by a given solution to solve the problem.

Text-sampling algorithms require to store the whole text together with the additional sampled-text which is used to speed-up the searching phase. In this context they are much more similar to online string-matching solutions and, although they have the additional good property to allow a direct access to the

[3] Although there exists many other searching algorithms able to show better practical performances on biological data (see for instance [3,6]) this kind of comparison goes beyond the objectives of this paper. We expect that the proposed approach is able to enhance the performances of different string matching algorithms with different, though similar, rates.

[4] The SMART tool is available online for download at `http://www.dmi.unict.it/~faro/smart/` or at `https://github.com/smart-tool/smart`.

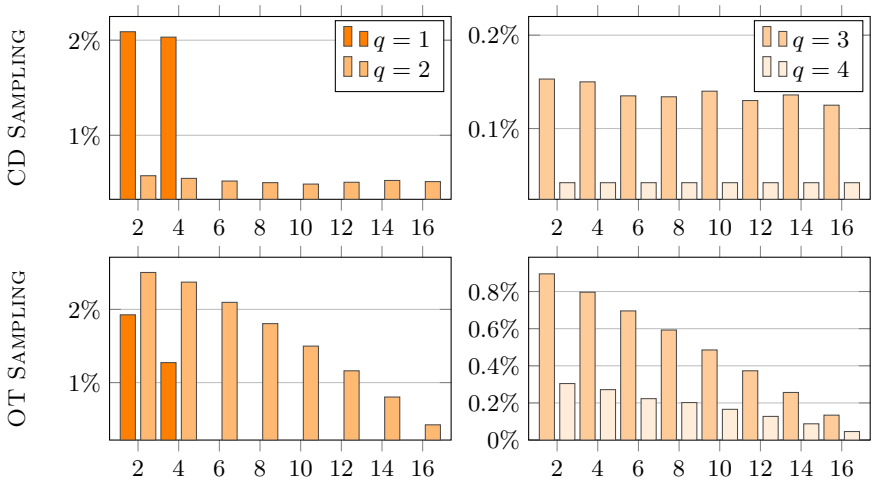

Fig. 2. Space consumption of the text sampled algorithms for different pivot characters with rank ranging from 1 to 16 and for different values of the parameter q, ranging from 1 to 4. Data are reported in terms of percentage of memory used relative to the original text size. On the top: memory space required in the case of Character Distance Sampling using q-grams. On the bottom: memory space required in the case of Occurrence Text Sampling using q-grams.

input text, to be of any practical interest they should require as little extra space as possible.

Fig. 2 shows the space consumption of the newly proposed text-sampling algorithms for different values of q in terms of percentage of memory used relative to the original text size. In the case of CDS memory space consumption is plotted on variations of the rank of the pivot character, while in the case of OTS it is plotted on variations of the size of the set of sampled characters. In both cases such value ranges from 2 to 16. As expected, the function which describes memory requirements follows a decreasing trend while the rank of the pivot character increases. Similarly space consumption drastically decreases when the size of q increases.

Data reported in Fig. 2 show that, when compared against the standard sampling algorithm (obtained with $q = 1$), the benefit in space consumption obtained by the algorithms based on condensed alphabets is impressive. Specifically the gain ranges from 72% (for $r = 1$ and $q = 2$) to 95% (for $r = 16$ and $q = 4$). In addition we can observe a sensible gain in the space consumption also in comparison with the OTS algorithms implemented using condensed alphabets.

For the sake of completeness we would like to point out that standard algorithms for the online string matching problem require an amount of space which is, in general, proportional to the length of the pattern and/or to the size of the alphabet. In this particular case (a 5MB text buffer) the Boyer-Moore-Horspool

algorithm requires only 1.24 KB of memory for implementing the occurrence heuristic (equivalent to a $O(\sigma)$-space complexity), while some among the most effective algorithms (for instance WFRq [3], SKIPq [6]) are implemented by means of a hash table of size 65536, requiring 0.2 MB of additional space. Thus it turns out that, under particular conditions (texts of moderate lengths), the practical space requirements of our proposed sampling algorithms are comparable with those of standard online string matching solutions.

4.2 Searching Time

In this section we compare the different text-sampling string matching algorithms in terms of searching times. In this context we refer to the *searching time* as the time needed to perform the searching of the pattern on both sampled and original texts, including any preprocessing of the underlying searching algorithm. In our analysis the searching time doesn't include the preprocessing time needed to construct the partial index.

Fig. 3 and Fig. 4 shows the resulting searching times of all tested algorithms when they were used for searching on a genome sequence and on an English text, respectively. As stated at the beginning of this section we also included the standard Boyer-Moore-Horspool string matching algorithm (identified by a dashed gray line), in order to put on evidence the speed-up proposed by our sampling algorithms.

From experimental results it turns out that in almost all cases the best results are obtained by the variants based on condensed alphabets and specifically for $q = 4$. When using a value of q greater than 1, the speed up obtained by CDS is always greater than 50% and reaches the value of 90% under suitable conditions, i.e. for $q = 4$ and long patterns.

In general the behaviour of CDS algorithms follow an increasing trend for increasing rank values. Thus in most cases the better choice is to use the most frequent element as the pivot character. Observe indeed that, when the rank of the pivot character is greater than a given threshold, the performances of the algorithms based on q-grams sensibly degrades. Specifically this threshold is approximately equal to 6 for short pattens ($m = 8$), while it increases for moderate patterns.

Going into details of the improvement in terms of running times we observe that the original CDS algorithm ($q = 1$) leads to improvements which are in percentage between 74% (in the case of short patterns) and 77% (in the case of long patterns) if compared with the underlying standard string matching algorithm. The new CDS algorithms based on condensed alphabets give instead much more evident improvements which range from 96% (for short patterns) and 99.6% (in the case of long patterns) compared with the same algorithm. This improvements translate into a gain up to 70% for short patterns and up to 96% in the case of long patterns, if compared with sampling solutions with no condensed alphabets.

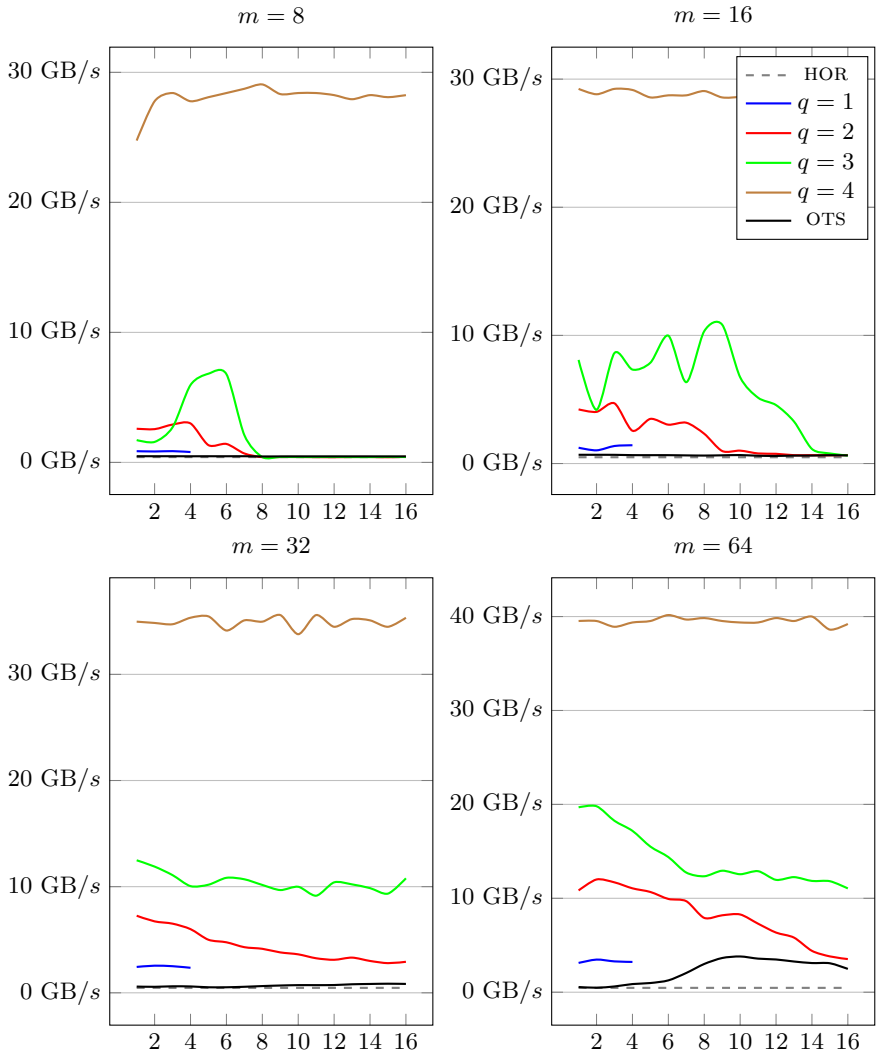

Fig. 3. Searching speed of the text sampling algorithms based on condensed alphabets for searching on a genome sequence. We used values q, with $1 \leq q \leq 4$, and pattern lengths m, with $8 \leq m \leq 256$. The dashed gray line represents the searching time of the standard Boyer-Moore-Horspool algorithm on the original text while the black solid line represents the best searching time of the OTS solution implemented with q-grams and $q = 4$. Running times (in the y axis) are represented in GB/s. The x axis represents the rank r of the pivot character in the case of the sampling algorithms, with $1 \leq r \leq 16$.

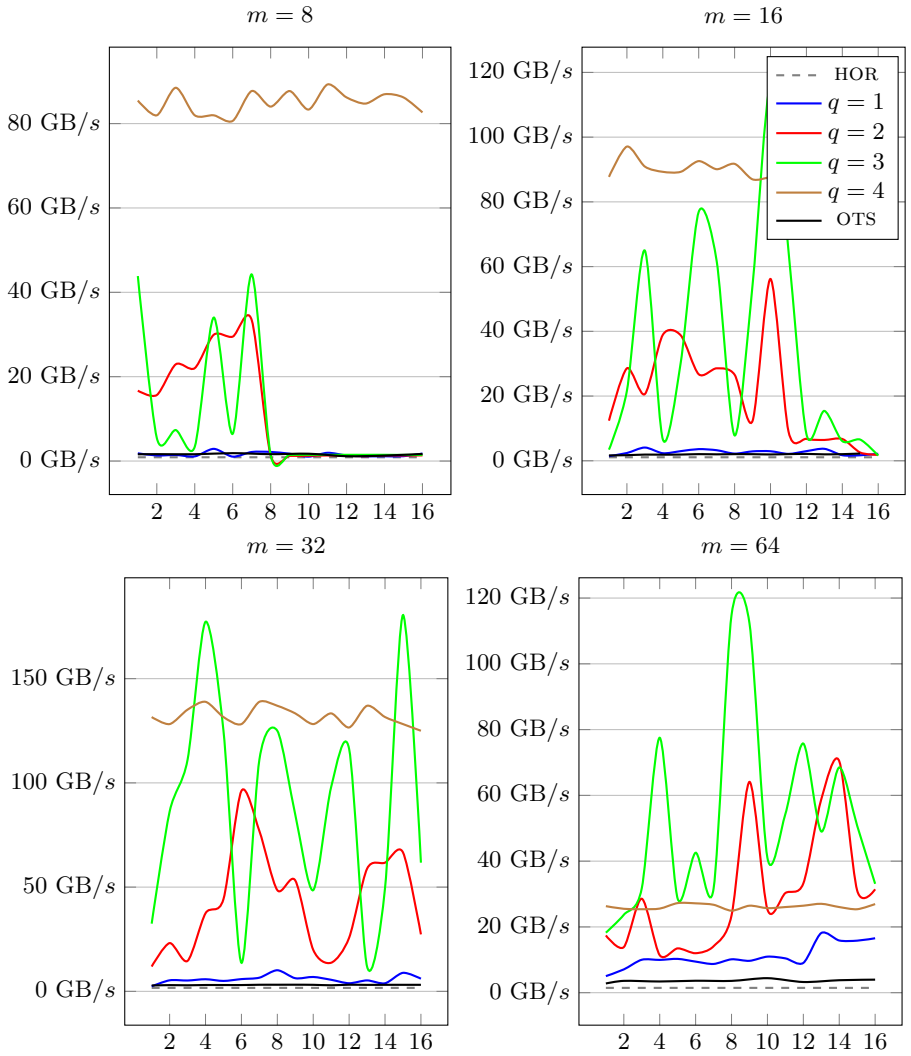

Fig. 4. Searching speed of the CDS and the best version of OTS algorithms based on condensed alphabets for searching on an English text. We used values q, with $1 \leq q \leq 4$, and short patterns of lengths m, with $m = 8$ and 16. The dashed gray line represents the searching time of the standard Boyer-Moore-Horspool algorithm on the original text. Running times (in the y axis) are represented in GB per second. The x axis represents the rank r of the pivot character (for CDS) and the size of the set of sampled characters (for OTS), with $1 \leq r \leq 16$.

5 Conclusions

In this paper we have presented an extension of a text sampling approach, called Character Distance, to the case of texts over small alphabets, as in the case of biological sequences. This extension was carried out using condensed alphabets in which consecutive groups of q characters are assimilated to a single element of the alphabet, significantly extending the size of it. The result obtained by this new approach was to significantly lower the execution time in the search phase while keeping the space used by the index below the space used by the previous approaches. Although our tests were limited to the exact string matching problem, obtaining excellent results, we believe that the approach can be effectively generalized even to non-standard string matching. Our future studies will focus in this direction in order to apply sampled string matching to other problems related to text processing.

References

1. Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977. doi:10.1145/359842.359859.
2. Domenico Cantone, Simone Faro, and Emanuele Giaquinta. Adapting boyer-moore-like algorithms for searching huffman encoded texts. *Int. J. Found. Comput. Sci.*, 23(2):343–356, 2012. doi:10.1142/S0129054112400163.
3. Domenico Cantone, Simone Faro, and Arianna Pavone. Linear and efficient string matching algorithms based on weak factor recognition. *ACM J. Exp. Algorithmics*, 24(1):1.8:1–1.8:20, 2019. doi:10.1145/3301295.
4. Francisco Claude, Gonzalo Navarro, Hannu Peltola, Leena Salmela, and Jorma Tarhio. String matching with alphabet sampling. *J. Discrete Algorithms*, 11:37–50, 2012. doi:10.1016/j.jda.2010.09.004.
5. Maxime Crochemore, Artur Czumaj, Leszek Gasieniec, Stefan Jarominek, Thierry Lecroq, Wojciech Plandowski, and Wojciech Rytter. Speeding up two string-matching algorithms. *Algorithmica*, 12(4/5):247–267, 1994. doi:10.1007/BF01185427.
6. Simone Faro. A very fast string matching algorithm based on condensed alphabets. In Riccardo Dondi, Guillaume Fertin, and Giancarlo Mauri, editors, *Algorithmic Aspects in Information and Management - 11th International Conference, AAIM 2016, Bergamo, Italy, July 18-20, 2016, Proceedings*, volume 9778 of *Lecture Notes in Computer Science*, pages 65–76. Springer, 2016. doi:10.1007/978-3-319-41168-2_6.
7. Simone Faro and Thierry Lecroq. The exact online string matching problem: A review of the most recent results. *ACM Comput. Surv.*, 45(2):13:1–13:42, 2013. doi:10.1145/2431211.2431212.
8. Simone Faro, Thierry Lecroq, Stefano Borzi, Simone Di Mauro, and Alessandro Maggio. The string matching algorithms research tool. In Jan Holub and Jan Zdárek, editors, *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. URL: http://www.stringology.org/event/2016/p09.html.

9. Simone Faro and Francesco Pio Marino. Reducing time and space in indexed string matching by characters distance text sampling. In Jan Holub and Jan Zdárek, editors, *Prague Stringology Conference 2020, Prague, Czech Republic, August 31 - September 2, 2020*, pages 148–159. Czech Technical University in Prague, Faculty of Information Technology, Department of Theoretical Computer Science, 2020. URL: http://www.stringology.org/event/2020/p13.html.
10. Simone Faro, Francesco Pio Marino, and Arianna Pavone. Efficient online string matching based on characters distance text sampling. *Algorithmica*, 82(11):3390–3412, 2020. doi:10.1007/s00453-020-00732-4.
11. Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.
12. R. Nigel Horspool. Practical fast searching in strings. *Softw. Pract. Exp.*, 10(6):501–506, 1980. doi:10.1002/spe.4380100608.
13. Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.
14. Gonzalo Navarro and Jorma Tarhio. Lzgrep: a boyer-moore string matching tool for ziv-lempel compressed text. *Softw. Pract. Exp.*, 35(12):1107–1130, 2005. doi:10.1002/spe.663.
15. Uzi Vishkin. Deterministic sampling - A new technique for fast pattern matching. *SIAM J. Comput.*, 20(1):22–40, 1991. doi:10.1137/0220002.
16. Andrew Chi-Chih Yao. The complexity of pattern matching for a random string. *SIAM J. Comput.*, 8(3):368–387, 1979. doi:10.1137/0208029.