Approximate String Matching Allowing for Inversions and Translocations

Domenico Cantone, Simone Faro, and Emanuele Giaquinta

Università di Catania, Dipartimento di Matematica e Informatica
Viale Andrea Doria 6, I-95125 Catania, Italy
{cantone | faro | giaquinta}@dmi.unict.it

Abstract. The approximate string matching problem consists in finding all locations at which a pattern P of length m matches a substring of a text T of length n, after a given finite number of edit operations.
In this paper we investigate such problem when the string distance involves translocations of equal length adjacent factors and inversions of factors. In particular, we devise a $\mathcal{O}(nm \max(\alpha, \beta))$-time and $\mathcal{O}(m^2)$-space algorithm, where α and β are respectively the maximum length of the factors involved in any translocation and inversion. Our algorithm is based on the dynamic-programming approach and makes use of the Directed Acyclic Word Graph of the pattern. Moreover we show that under the assumptions of equiprobability and independence of characters our algorithm has a $\mathcal{O}(n \log_\sigma m)$ average time complexity. Finally, we briefly sketch in an appendix an efficient implementation, based on bit-parallelism.

1 Introduction

Retrieving information and teasing out the meaning of biological sequences are central problems in modern biology. Generally, basic biological information is stored in strings of nucleic acids (DNA, RNA) or amino acids (proteins). Aligning sequences helps in revealing their shared characteristics, while matching sequences can infer useful information from them.

With the availability of large amounts of DNA data, matching of nucleotide sequences has become an important application and there is an increasing demand for fast computer methods for analysis and data retrieval. In recent years, much work has been devoted to the development of efficient methods for aligning strings and, despite sequence alignment seems to be a well-understood problem (especially in the edit-distance model), the same can not be said for the approximate string matching problem on biological sequences.

Approximate string matching is a fundamental problem in text processing and consists in finding approximate matches of a pattern in a string. The closeness of a match is measured in terms of the sum of the costs of the edit operations necessary to convert the string into an exact match.

Most biological string matching methods are based on the *edit distance* [7] (also called the *Levenshtein distance*) or on the *Damerau edit distance* [6]. The edit operations in the former edit distance are *insertion*, *deletion*, and *substitution* of characters, while the latter one allows *swaps* of characters, i.e., traspositions of two adjacent characters (for an in-depth survey on approximate string matching, see [8]). These distances assume that changes between strings occur locally, i.e., only a small portion of the string is involved in the mutation event. In contrast, evidence shows that large scale changes are possible. For example, large pieces of DNA can be moved from

Domenico Cantone, Simone Faro, Emanuele Giaquinta: Approximate String Matching Allowing for Inversions and Translocations, pp. 37–51 .
Proceedings of PSC 2010, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04597-8 © Czech Technical University in Prague, Czech Republic

one location to another (*translocations*), or replaced by their reversed complements (*inversions*).

In this paper we investigate the approximate string matching problem under a string distance whose edit operations are translocations of equal length adjacent factors and inversions of factors. In particular, we present a $\mathcal{O}(nm \max(\alpha, \beta))$-time and $\mathcal{O}(m^2)$-space algorithm, where α and β are the maximum length of the factors involved in a translocation and in an inversion, respectively. Our algorithm is based on a dynamic-programming approach and makes use of the Directed Acyclic Word Graph of the pattern. The DAWG data structure has already been used in algorithms for the approximate string matching problem [11,10], to keep track of the substrings of the pattern that match the text at every location. We show that under the assumption of equiprobability and independence of characters in the alphabet, on the average our algorithm has a $\mathcal{O}(n \log_\sigma m)$-time complexity. Finally, we present also an efficient implementation of our algorithm, based on bit-parallelism, which has $\mathcal{O}(n \max(\alpha, \beta))$-time and $\mathcal{O}(\sigma + m)$-space complexity, when the pattern length is comparable with the size of the computer word. To our knowledge there is no report in the literature of a similar formalization of the above problem.

The rest of the paper is organized as follows. In Section 2 we introduce some preliminary notions and definitions. Subsequently, in Section 3 we present a new automaton-based algorithm for the approximate string matching problem with translocations and inversions. Section 4 is devoted to the analysis of our algorithm both in the worst and in the average-case. In Section 5 we present experimental results which allow us to evaluate the practical performance of our newly proposed algorithm and its bit-parallel variant, briefly described in Appendix A. Finally, we draw our conclusions in Section 6.

2 Basic notions and definitions

Let P be a string of length $m \geq 0$, over an alphabet Σ. We represent it as a finite array $P[0 \,..\, m-1]$ of characters of Σ and write $|P| = m$. In particular, for $m = 0$ we obtain the empty string ε. We denote by $P[i]$ the $(i+1)$-st character of P, for $0 \leq i < m$. Likewise, the substring of P contained between the $(i+1)$-st and the $(j+1)$-st characters of P is indicated with $P[i \,..\, j]$, for $0 \leq i \leq j < m$. The set of substrings (also called *factors*) of P is denoted by $\mathit{Fact}(P)$. Given another string P', we say that P' is a suffix of P (in symbols, $P' \sqsupseteq P$) if $P' = P[i \,..\, m-1]$, for some $0 \leq i < m$, and indicate with $\mathit{Suff}(P)$ the set of the suffixes of P. Similarly, we say that P' is a prefix of P if $P' = P[0 \,..\, i]$, for some $0 \leq i < m$. We also put $P_i =_{\text{Def}} P[0 \,..\, i]$, for $0 \leq i < m$, and make the convention that P_{-1} denotes the empty string ε. In addition, we write PP' to denote the concatenation of P and P', and P^{r} for the reverse of the string P, i.e., $P^{\mathsf{r}} =_{\text{Def}} P[m-1]P[m-2]\ldots P[0]$.

A *distance* $d : \Sigma^* \times \Sigma^* \to \mathbb{R}$ is a function which associates to any pair of strings X and Y the minimal cost of any finite sequence of edit operations which transforms X into Y, if such a sequence exists, ∞ otherwise. Edit operations have the form $Z \to_t W$, with $Z, W \in \Sigma^*$ and t a nonnegative real number which represents the cost. If, for every operation $Z \to_t W$, there is also the symmetric operation $W \to_t Z$ (with the same cost), then the distance d is symmetric, i.e., $d(X,Y) = d(Y,X)$, for all $X, Y \in \Sigma^*$.

For $X \in Fact(P)$, we denote with $end\text{-}pos(X)$ the set of all positions in P where an occurrence of X ends; formally,

$$end\text{-}pos(X) =_{\text{Def}} \{i \mid |X| - 1 \leq i < m \text{ and } X \sqsupseteq P_i\}.$$

For any given pattern P, we define an equivalence relation $\mathcal{R}_P$ by putting

$$X \,\mathcal{R}_P\, Y \iff_{\text{Def}} end\text{-}pos(X) = end\text{-}pos(Y),$$

for all $X, Y \in \Sigma^*$, and denote with $\mathcal{R}_P(X)$ the equivalence class over Σ^* of the string X.

The Directed Acyclic Word Graph [3,4,5] of a pattern P (DAWG, for short) is the deterministic automaton $\mathcal{A}(P) = (Q, \Sigma, \delta, root, F)$ whose language is $Fact(P)$, where

- $Q = \{\mathcal{R}_P(X) : X \in Fact(P)\}$ is the set of states,
- Σ is the alphabet of the characters in P,
- $\delta : Q \times \Sigma \to Q$ is the transition function defined, for all $c \in \Sigma$ and $Yc \in Fact(P)$, by $\delta(\mathcal{R}_P(Y), c) =_{\text{Def}} \mathcal{R}_P(Yc)$,
- $root = \mathcal{R}_P(\varepsilon)$ is the initial state,
- $F = Q$ is the set of final states.

For each equivalence class q of $\mathcal{R}_P$, let $val(q)$ be the longest string X in the equivalence class q and put $length(q) =_{\text{Def}} length(val(q))$. In addition, we define a failure function, $s\ell : Fact(P) \setminus \{\varepsilon\} \to Fact(P)$, called *suffix link*, by putting, for any $X \in Fact(P) \setminus \{\varepsilon\}$,

$$s\ell(X) =_{\text{Def}} \text{longest } Y \in Suff(X) \text{ such that } Y \not\mathcal{R}_P X.$$

The function $s\ell$ has the following property:

$$X \,\mathcal{R}_P\, Y \Longrightarrow s\ell(X) = s\ell(Y).$$

We extend the functions $s\ell$ and $end\text{-}pos$ to Q by putting, for each $q \in Q$,

$$\begin{aligned} s\ell(q) &=_{\text{Def}} \mathcal{R}_P(s\ell(val(q))) \\ end\text{-}pos(q) &=_{\text{Def}} end\text{-}pos(val(q)). \end{aligned}$$

Definition 1. *Given two strings X and Y, the* mutation distance $md(X, Y)$ *is based on the following edit operations:*

(1) ***Translocation****: a factor of the form ZW is transformed into WZ, provided that $|Z| = |W| > 0$.*
(2) ***Inversion****: a factor Z is tranformed into Z^{r}.*

Both operations are assigned unit cost. □

Observe that, by definition, the maximum length of the factors involved in a translocation is $\lfloor |X|/2 \rfloor$, whereas the length of the factors involved in an inversion can be up to $|X|$. Note, moreover, that there are strings X, Y such that X can not be converted into Y by any sequence of translocations and inversions, in which case $md(X, Y) = \infty$. When $md(X, Y) < \infty$, we say that X and Y have an md-match. Additionally, if X has an md-match with a suffix of Y, we write $X \sqsupseteq_{md} Y$.

3 An automaton-based approach for the pattern matching problem with translocations and inversions

We present an efficient algorithm, called M-Sampling, which finds the md-matches of a given pattern P (of length m) in a text T (of length n). Our algorithm, based on the dynamic programming approach, has a $\mathcal{O}(nm\max(\alpha,\beta))$-time and $\mathcal{O}(m^2)$-space complexity, where $\alpha \leq \lfloor m/2 \rfloor$ is a bound on the length of the factors involved in any translocation and $\beta \leq m$ is a bound on the length of the factors involved in any inversion.

Given P, T, m, n, α, and β as above, the M-Sampling algorithm iteratively computes for $j = m-1, m, \ldots, n-1$ all the prefixes of P which have an md-match with a suffix of T_j, by exploiting information gathered at previous iterations. For this purpose, a set $\mathcal{S}_j$ is maintained, defined by

$$\mathcal{S}_j =_{\text{Def}} \{0 \leq i \leq m-1 \mid P_i \sqsupseteq_{md} T_j\} .$$

Thus, the pattern P has an md-match ending at position j of the text T if and only if $(m-1) \in \mathcal{S}_j$.

Since the allowed edit operations involve substrings of the pattern P, it is useful to introduce the set $\mathcal{F}_j^k$ of all the positions in P where an occurrence of the suffix of T_j of length k ends. More precisely, for $1 \leq k \leq \alpha$ and $k-1 \leq j < n$, we put

$$\mathcal{F}_j^k =_{\text{Def}} \{k-1 \leq i \leq m-1 \mid T[j-k+1\,..\,j] \sqsupseteq P_i\} .$$

Observe that $\mathcal{F}_j^k \subseteq \mathcal{F}_j^h$, for $1 \leq h \leq k \leq m$.

Similarly, to handle inversions, it is convenient to define the set $\mathcal{I}_j^k$ of the positions in P where an occurrence of the reverse of the suffix of T_j of length k ends. More precisely, for $1 \leq k \leq \beta$ and $k-1 \leq j < n$, we put

$$\mathcal{I}_j^k =_{\text{Def}} \{k-1 \leq i \leq m-1 \mid (T[j-k+1\,..\,j])^{\mathsf{r}} \sqsupseteq P_i\} .$$

The sets $\mathcal{S}_j$ can then be computed based on the following elementary recursion.

Lemma 2. *Let T and P be a text of length n and a pattern of length m, respectively. Then $i \in \mathcal{S}_j$, for $0 \leq i < m$ and $i \leq j < n$, if and only if one of the following three facts holds*

(a) $P[i] = T[j]$ and $(i-1) \in \mathcal{S}_{j-1} \cup \{-1\}$ (standard match);
(b) $(i-k) \in \mathcal{F}_j^k$, $i \in \mathcal{F}_{j-k}^k$, and $(i-2k) \in \mathcal{S}_{j-2k} \cup \{-1\}$, for some $1 \leq k \leq \lfloor \frac{i+1}{2} \rfloor$ (translocation);
(c) $i \in \mathcal{I}_j^k$ and $(i-k) \in \mathcal{S}_{j-k} \cup \{-1\}$, for some $1 \leq k \leq i+1$ (inversion). □

Conditions (b) and (c) refer to a translocation of adjacent factors of length k and an inversion of a factor of length k, respectively.

Likewise, the sets $\mathcal{F}_j^k$ and $\mathcal{I}_j^k$ can be computed according to the following lemma:

Lemma 3. *Let T and P be a text of length n and a pattern of length m, respectively. Then $i \in \mathcal{F}_j^k$, for $1 \leq k \leq \alpha$, $k-1 \leq i < m$, and $k-1 \leq j < n$, if and only if the following condition holds*

$$(k = 1 \text{ or } (i-1) \in \mathcal{F}_{j-1}^{k-1}) \text{ and } P[i] = T[j] .$$

Similarly, $i \in \mathcal{I}_j^k$, for $1 \leq k \leq \beta$, $k-1 \leq i < m$, and $k-1 \leq j < n$, if and only if the following condition holds

$$(k = 1 \text{ or } i \in \mathcal{I}_{j-1}^{k-1}) \text{ and } P[i-k+1] = T[j] .$$ □

Based on Lemmas 2 and 3, a general dynamic programming algorithm can be readily constructed, characterized by an overall $\mathcal{O}(nm \max(\alpha, \beta))$-time and $\mathcal{O}(m^2)$-space complexity. However, the overhead due to the computation of the sets $\mathcal{F}_j^k$ and $\mathcal{I}_j^k$ turns out to be quite large. By suitably preprocessing the pattern with the DAWG data structure, as will be described in the next section, the M-Sampling algorithm succeeds in reducing drastically such overhead (see Fig. 2). The code of the algorithm M-Sampling is shown in Fig. 1 (on the left).

3.1 Efficient computation of the sets $\mathcal{F}_j^k$ and $\mathcal{I}_j^k$

An efficient method for computing the sets $\mathcal{F}_j^k$ described above, for $1 \leq k \leq \alpha$ and $k-1 \leq j < n$, makes use of the DAWG of the pattern P and the function *end-pos*. Later we will also show how to compute efficiently the sets $\mathcal{I}_j^k$.

Let $\mathcal{A}(P) = (Q, \Sigma, \delta, root, F)$ be the DAWG of P. For each position j in T, let P' be the longest factor of P, of length at most α, which is a suffix of T_j, let q_j be the state of $\mathcal{A}(P)$ such that $\mathcal{R}_p(P') = q_j$, and let l_j be the length of P'. We call the pair (q_j, l_j) a T-configuration of $\mathcal{A}(P)$. The idea is then to compute the T-configuration (q_j, l_j) of $\mathcal{A}(P)$, for each position j of the text, while scanning the text. The set $\mathcal{F}_j^k$ computed at previous iterations are not maintained explicitly; rather, only T-configurations are maintained. These are then used to compute efficiently the set $\mathcal{F}_j^k$ only when needed.

The longest factor of P ending at position j of T is computed in the same way as in the Forward-Dawg-Matching algorithm for the exact pattern matching problem (cf. [5]). Since we are interested in factors of length at most α, we maintain the invariant that the current state of the automaton never corresponds to factors longer than α (we discovered that much the same idea was used in [10]).

Let (q_{j-1}, l_{j-1}) be the T-configuration of $\mathcal{A}(P)$ at step $(j-1)$. Two cases must be distinguished.

Case $l_{j-1} < \alpha$: The new T-configuration (q_j, l_j) is set to $(\delta(q, T[j]), length(q)+1)$, where q is the first node in the suffix path $(q_{j-1}, s\ell(q_{j-1}), s\ell^{(2)}(q_{j-1}), \ldots)$ of q_{j-1}, including q_{j-1}, having a transition on $T[j]$, if such a node exists; otherwise (q_j, l_j) is set to $(root, 0)$.[1]

Case $l_{j-1} = \alpha$: We first compute the T-configuration corresponding to the factor $T[j-\alpha+1 \,..\, j-1]$ of P of length $(\alpha - 1)$ ending at position $j-1$ in T, namely the T-configuration (q'_{j-1}, l'_{j-1}), where

$$(q'_{j-1}, l'_{j-1}) \;=_{\text{Def}} \begin{cases} (s\ell(q_{j-1}), l_{j-1}-1) & \text{if } length(s\ell(q_{j-1})) = l_{j-1}-1 \\ (q_{j-1}, l_{j-1}-1) & \text{otherwise.} \end{cases}$$

Then we compute the new T-configuration (q_j, l_j) starting from (q'_{j-1}, l'_{j-1}) as in the previous case, observing that $l'_{j-1} = \alpha - 1$. The algorithm to update the T-configuration of the DAWG $\mathcal{A}(P)$ is given in Fig. 1 (on the right), where $s\ell^*$ denotes the improved suffix link [5].

Before explaining how to compute the sets $\mathcal{F}_j^k$, it is convenient to introduce a partial function, $\phi : Q \times \mathbb{N} \rightarrow Q$, which given a node $q \in Q$ and a length $k \leq length(q)$ computes the node $\phi(q, k)$ whose corresponding set of factors contains the suffix of

[1] We recall that $s\ell^{(0)}(q) =_{\text{Def}} q$ and, recursively, $s\ell^{(h+1)}(q) =_{\text{Def}} s\ell(s\ell^{(h)}(q))$, for $h \geq 0$, provided that $s\ell^{(h)}(q) \neq root$.

$val(q)$ of length k. This is the same as saying, more formally, that $\phi(q,k)$ is the node $s\ell^{(i)}(q)$ such that

$$length(s\ell^{(i+1)}(q)) < k \leq length(s\ell^{(i)}(q)),$$

for each $q \in Q$ and each integer $k \leq length(q)$. Roughly speaking, $\phi(q,k)$ is the first node p in the suffix path of q such that $length(s\ell(p)) < k$.

In the preprocessing phase, the DAWG $\mathcal{A}(P) = (Q, \Sigma, \delta, root, F)$ together with the associated *end-pos* function is computed. Since for a pattern P of length m we have that $|Q| \leq 2m+1$ and $|end\text{-}pos(q)| \leq m$, for each $q \in Q$, we need only $\mathcal{O}(m^2)$ extra space (see [3,4]).

To compute the set $\mathcal{F}_j^k$, for $1 \leq k \leq l_j$, one can take advantage of the following relation

$$\mathcal{F}_j^k = end\text{-}pos(\phi(q_j,k)). \tag{1}$$

Notice that, in particular, we have $\mathcal{F}_j^{l_j} = end\text{-}pos(q_j)$.

The time complexity of the computation of $\phi(q,k)$ can be bounded by the length of the suffix path of node q. Specifically, since the sequence

$$(length(s\ell^{(0)}(q)), length(s\ell^{(1)}(q)), \ldots, 0)$$

of the lengths of the nodes in the suffix path from q is strictly deacreasing, we can do at most $length(q)$ iterations over the suffix link, obtaining a $\mathcal{O}(m)$-time complexity.

According to Lemma 2, a translocation of length $2k$ at position j of the text T is possible only if factors of P of length at least k have been recognized at both positions j and $j-k$, namely if $l_j \geq k$ and $l_{j-k} \geq k$.

Let $\langle k_1, k_2, \ldots, k_r \rangle$ be the increasing sequence of all values k such that $1 \leq k \leq \min(l_j, l_{j-k})$. For each $1 \leq i \leq r$, condition (b) of Lemma 2 requires member queries on the sets $\mathcal{F}_j^{k_i}$ and $\mathcal{F}_{j-k_i}^{k_i}$.

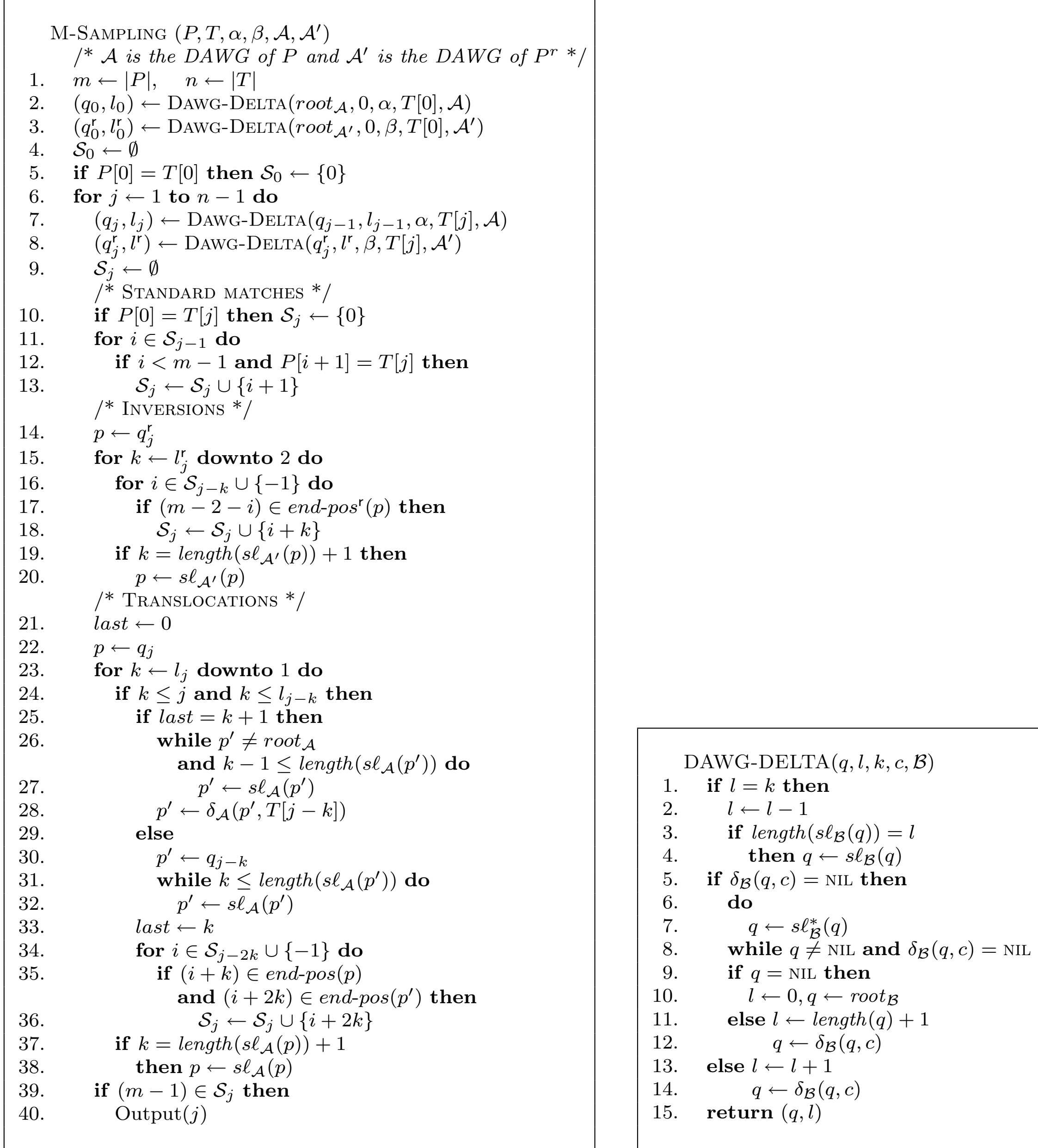
We notice that, if we proceed for decreasing values of k, the sets $\mathcal{F}_j^k$, for $1 \leq k \leq l_j$, can be computed in constant time. Specifically, the set $\mathcal{F}_j^k$ can be computed in constant time from $\mathcal{F}_j^{k+1}$, for $k = 1, \ldots, l_j - 1$, with at most one iteration over the suffix link of the state $\phi(q_j, k+1)$.

The computation of $\mathcal{F}_{j-k_r}^{k_r}$ has a $\mathcal{O}(\alpha)$-time complexity, since $length(q_{j-k_r}) \leq \alpha$. To compute $\mathcal{F}_{j-k_i}^{k_i}$, for $i = r-1, r-2, \ldots, 1$, we distinguish the following two cases:

Case $k_{i+1} = k_i + 1$: Let $q' = \phi(q_{j-k_{i+1}}, k_{i+1})$. Given the node q' computed in the previous iteration, the node $\phi(q_{j-k_i}, k_i)$ can be computed in two steps: first, we look up the node corresponding to the suffix of length $k_{i+1} - 2$ of the factor represented by q', with at most two iterations of the suffix link of q'; then, we perform a transition on $T[j-k_i]$ on the node so found. Formally:

$$\phi(q_{j-k_i}, k_i) = \delta(\phi(q', k_{i+1} - 2), T[j-k_i]).$$

Case $k_{i+1} > k_i + 1$: Observe that $l_{j-s} \leq s-1$ must hold, for each $s = k_{i+1}-1, \ldots, k_i + 1$. In particular, we have $l_{j-(k_i+1)} \leq k_i$ which implies that $l_{j-k_i} \leq k_i + 1$ since $l_j \leq l_{j-1} + 1$ always holds. Hence, the computation of $\phi(q_{j-k_i}, k_i)$ requires at most one iteration of the suffix link of q_{j-k_i}.



```
M-Sampling (P, T, α, β, A, A')
     /* A is the DAWG of P and A' is the DAWG of P^r */
1.   m ← |P|,   n ← |T|
2.   (q_0, l_0) ← Dawg-Delta(root_A, 0, α, T[0], A)
3.   (q^r_0, l^r_0) ← Dawg-Delta(root_A', 0, β, T[0], A')
4.   S_0 ← ∅
5.   if P[0] = T[0] then S_0 ← {0}
6.   for j ← 1 to n − 1 do
7.     (q_j, l_j) ← Dawg-Delta(q_{j−1}, l_{j−1}, α, T[j], A)
8.     (q^r_j, l^r) ← Dawg-Delta(q^r_j, l^r, β, T[j], A')
9.     S_j ← ∅
       /* Standard matches */
10.    if P[0] = T[j] then S_j ← {0}
11.    for i ∈ S_{j−1} do
12.      if i < m − 1 and P[i + 1] = T[j] then
13.        S_j ← S_j ∪ {i + 1}
       /* Inversions */
14.    p ← q^r_j
15.    for k ← l^r_j downto 2 do
16.      for i ∈ S_{j−k} ∪ {−1} do
17.        if (m − 2 − i) ∈ end-pos^r(p) then
18.          S_j ← S_j ∪ {i + k}
19.      if k = length(sℓ_A'(p)) + 1 then
20.        p ← sℓ_A'(p)
       /* Translocations */
21.    last ← 0
22.    p ← q_j
23.    for k ← l_j downto 1 do
24.      if k ≤ j and k ≤ l_{j−k} then
25.        if last = k + 1 then
26.          while p' ≠ root_A
               and k − 1 ≤ length(sℓ_A(p')) do
27.            p' ← sℓ_A(p')
28.          p' ← δ_A(p', T[j − k])
29.        else
30.          p' ← q_{j−k}
31.          while k ≤ length(sℓ_A(p')) do
32.            p' ← sℓ_A(p')
33.        last ← k
34.        for i ∈ S_{j−2k} ∪ {−1} do
35.          if (i + k) ∈ end-pos(p)
               and (i + 2k) ∈ end-pos(p') then
36.            S_j ← S_j ∪ {i + 2k}
37.      if k = length(sℓ_A(p)) + 1
38.        then p ← sℓ_A(p)
39.    if (m − 1) ∈ S_j then
40.      Output(j)
```

```
DAWG-DELTA(q, l, k, c, B)
1.   if l = k then
2.     l ← l − 1
3.     if length(sℓ_B(q)) = l
4.       then q ← sℓ_B(q)
5.   if δ_B(q, c) = nil then
6.     do
7.       q ← sℓ*_B(q)
8.     while q ≠ nil and δ_B(q, c) = nil
9.     if q = nil then
10.      l ← 0, q ← root_B
11.    else l ← length(q) + 1
12.          q ← δ_B(q, c)
13.  else l ← l + 1
14.        q ← δ_B(q, c)
15.  return (q, l)
```

Figure 1. On the left: the M-Sampling algorithm for solving the pattern matching problem with translocations and inversions. **On the right:** the DAWG state update algorithm.

Thus, in both cases, $\mathcal{F}^{k_i}_{j-k_i}$ can be computed in constant time, for $1 \leq i < r$. Therefore, the total complexity for computing all the sets $\mathcal{F}^{k_i}_{j-k_i}$, for $i = 1, \ldots, r$, is $\mathcal{O}(\alpha)$.

Next, to compute the sets $\mathcal{I}^k_j$ we use the DAWG $\mathcal{A}(P^{\mathsf{r}})$ of P^{r}. Specifically, we compute the longest reversed factor ending at j and maintain the invariant that the current state of the automaton never corresponds to factors longer than β, using algorithm given in Fig. 1 (on the right), as for the computation of the sets $\mathcal{F}^k_j$. Let

$(q^{\mathsf{r}}_j, l^{\mathsf{r}}_j)$ denote the T-configuration of $\mathcal{A}(P^{\mathsf{r}})$ after having read the character of T at position j, where l^{r}_j is the length of the longest reversed factor of P recognized. Then the sets $\mathcal{I}^k_j$ can be computed, for $2 \leq k \leq l^{\mathsf{r}}_j$, by

$$\mathcal{I}^k_j = \{i \mid (m - i + k - 2) \in \mathit{end\text{-}pos}(\phi(q^{\mathsf{r}}_j, k))\}\,. \tag{2}$$

Indeed, $i \in \mathcal{I}^k_j$ iff $P[i - k + 1 \,..\, i] = (T[j - k + 1 \,..\, j])^{\mathsf{r}}$ iff

$$P^{\mathsf{r}}[(m-1) - i \,..\, (m-1) - (i - k + 1)] = (P[i - k + 1 \,..\, i])^{\mathsf{r}} = T[j - k + 1 \,..\, j]\,.$$

Thus (2) follows, since the latter is equivalent to $(m - i + k - 2) \in \mathit{end\text{-}pos}(\phi(q^{\mathsf{r}}_j, k))$.

For each $k = 1, \ldots, l^{\mathsf{r}}_j$, condition (c) of Lemma 2 requires member queries on the sets $\mathcal{I}^k_j$. As in the case of the sets $\mathcal{F}^k_j$, the set $\mathit{end\text{-}pos}(\phi(q^{\mathsf{r}}_j, k))$ can be computed in constant time, in decreasing order of k, by iterating the suffix link on q^{r}_j. Although $\mathcal{I}^k_j$ is not equal to $\mathit{end\text{-}pos}(\phi(q^{\mathsf{r}}_j, k))$, a member query on $\mathcal{I}^k_j$ can still be done in constant time, using (2).

4 Complexity analysis

We first analyze the worst-case time complexity of the M-Sampling algorithm and then its average-case complexity. Our analysis assumes that sets are implemented as bit vectors so that any member query on a set takes constant time. We will also evaluate the space complexity of the M-Sampling algorithm.

4.1 Worst-case analysis

First of all, observe that the main **for**-loop at line 6 is always executed n times. Moreover, observe that $|\mathcal{S}_j| \leq m$, $l_j \leq \alpha$, and $l^{\mathsf{r}}_j \leq \beta$, for all $0 \leq j < n$. For each iteration of the **for**-loop at line 23, the amortized cost of the two **while**-loops at lines 26 and 31 is $\mathcal{O}(1)$. Thus, at each iteration of the main **for**-loop, the **for**-loop at line 11 takes at most $\mathcal{O}(m)$ time while the **for**-loops at lines 15 and 23 take at most $\mathcal{O}(m\beta)$ and $\mathcal{O}(m\alpha)$ time respectively. Summing up, the algorithm has a $\mathcal{O}(nm\max(\alpha,\beta))$ worst-case time complexity, which becomes $\mathcal{O}(nm^2)$-time when $\max(\alpha,\beta) = \Theta(m)$.

4.2 Average-case analysis

Next, we evaluate the average time complexity of the algorithm M-Sampling assuming the uniform distribution and independence of characters.

Given integers $1 \leq \alpha, \beta \leq m \leq n$ and an alphabet Σ of size $\sigma \geq 4$, for $j = 0, 1, \ldots, n-1$ we consider the following nonnegative random variables over the sample space of the pairs of strings $P, T \in \Sigma^*$ of length m and n, respectively:

- $X(j) =_{\text{Def}}$ the length $l_j \leq \alpha$ of the longest factor of P which is a suffix of T_j,
- $Y(j) =_{\text{Def}}$ the length $l^{\mathsf{r}}_j \leq \beta$ of the longest factor of P^{r} which is a suffix of T_j,
- $Z(j) =_{\text{Def}}$ $|\mathcal{S}_j|$, where we recall that $\mathcal{S}_j = \{0 \leq i \leq m-1 \mid P_i \sqsupseteq_{md} T_j\}$.

Then the run-time of a call to the M-Sampling algorithm with parameters (P, T, α, β) is proportional to

$$\sum_{j=1}^{n-1}\left(Z(j-1)+\sum_{k=2}^{Y(j)} Z(j-k)+\left(\sum_{k=1}^{X(j)} Z(j-2k)+X(j)\right)\right), \tag{3}$$

where the external summation refers to the main **for**-loop (at line 6), and the three terms within it take care of the internal **for**-loops at lines 11, 15, and 23, in that order.

The average-case complexity of the M-Sampling algorithm is thus the expectation of (3), which, in view of the linearity of expectation, is equal to

$$\sum_{j=1}^{n-1}\left(E(Z(j-1))+E\left(\sum_{k=2}^{Y(j)} Z(j-k)\right)+E\left(\sum_{k=1}^{X(j)} Z(j-2k)\right)+E(X(j))\right). \tag{4}$$

Since

$$\begin{aligned} E(X(j)) &\leq E(X(n-1)) \\ E(Y(j)) &\leq E(Y(n-1)) \\ E(Z(j)) &\leq E(Z(n-1)), \end{aligned}$$

for $0 \leq j \leq n-1$,[2] and also

$$E(X(n-1)) = E(Y(n-1)),$$

by putting

$$X =_{\text{Def}} X(n-1) \qquad \text{and} \qquad Z =_{\text{Def}} Z(n-1),$$

expression (4) gets bounded from above by

$$\sum_{j=1}^{n-1}\left(E(Z)+E\left(\sum_{k=2}^{X} Z\right)+E\left(\sum_{k=1}^{X} Z\right)+E(X)\right). \tag{5}$$

For $i = 0, \ldots, m-1$, let Z_i be the indicator variable

$$Z_i =_{\text{Def}} \begin{cases} 1 & \text{if } i \in \mathcal{S}_{n-1} \\ 0 & \text{otherwise}, \end{cases}$$

so that

$$Z = \sum_{i=0}^{m-1} Z_i \qquad \text{and} \qquad E(Z_i^2) = E(Z_i) = Pr\{P_i \sqsupseteq_{md} T\}.$$

Likewise, for $k = 1, \ldots, m$, let X_k be the indicator variable

$$X_k =_{\text{Def}} \begin{cases} 1 & \text{if } X \geq k \\ 0 & \text{otherwise}, \end{cases}$$

so that

$$X = \sum_{k=1}^{m} X_k \qquad \text{and} \qquad E(X_k^2) = E(X_k) = Pr\{X \geq k\}.$$

[2] In fact, for $j = m, \ldots, n-1$ all inequalities hold as equalities.

The we have

$$\sum_{k=1}^{X} Z = XZ = \left(\sum_{k=1}^{m} X_k\right) \cdot \left(\sum_{i=0}^{m-1} Z_i\right) = \sum_{k=1}^{m}\sum_{i=0}^{m-1} X_k Z_i \,.$$

Therefore

$$E\left(\sum_{k=2}^{X} Z\right) \leq E\left(\sum_{k=1}^{X} Z\right) = \sum_{k=1}^{m}\sum_{i=0}^{m-1} E(X_k Z_i) \ ,$$

yielding the following upper bound for (5):

$$\sum_{j=1}^{n-1}\left(E(Z) + 2 \cdot \sum_{k=1}^{m}\sum_{i=0}^{m-1} E(X_k Z_i) + E(X)\right) \ . \tag{6}$$

To estimate each of the terms $E(X_k Z_i)$ in (6), we use the well-known Cauchy-Schwarz inequality which in the context of expectations assumes the form

$$|E(UV)| \leq \sqrt{E(U^2)E(V^2)}\,,$$

for any two random variables U and V such that $E(U^2)$, $E(V^2)$ and $E(UV)$ are all finite.

Then, for $1 \leq k \leq m$ and $0 \leq i \leq m-1$, we have

$$E(X_k Z_i) \leq \sqrt{E(X_k^2)E(Z_i^2)} = \sqrt{E(X_k)E(Z_i)}\,. \tag{7}$$

From (7), it then follows that (6) is bounded from above by

$$\begin{aligned} &\sum_{j=1}^{n-1}\left(E(Z) + 2 \cdot \sum_{k=1}^{m}\sum_{i=0}^{m-1} \sqrt{E(X_k)E(Z_i)} + E(X)\right) \\ &\quad = \sum_{j=1}^{n-1}\left(E(Z) + 2 \cdot \left(\sum_{k=1}^{m} \sqrt{E(X_k)}\right) \cdot \left(\sum_{i=0}^{m-1} \sqrt{E(Z_i)}\right) + E(X)\right) \ . \end{aligned} \tag{8}$$

To better understand (8), we evaluate the expectations $E(X)$ and $E(Z)$ and the sums $\sum_{k=1}^{m} \sqrt{E(X_k)}$ and $\sum_{i=0}^{m-1} \sqrt{E(Z_i)}$. To this purpose, it will be useful to estimate also the expectations

- $E(X_k) = Pr\{X \geq k\}$, for $1 \leq k \leq m$, and
- $E(Z_i) = Pr\{P_i \sqsupseteq_{md} T\}$, for $0 \leq i \leq m-1$.

Concerning $E(X_k) = Pr\{X \geq k\}$, we reason as follows. Since $T[n-k\,..\,n-1]$ ranges uniformly over a collection of σ^k strings and there can be at most $\min(\sigma^k, m - k+1)$ distinct factors of length k in P, the probability $Pr\{X \geq k\}$ that one of them matches $T[n-k\,..\,n-1]$ is at most $\min\left(1, \frac{m-k+1}{\sigma^k}\right)$, so that, for $k = 1, \ldots, m$, we have

$$E(X_k) \leq \min\left(1, \frac{m-k+1}{\sigma^k}\right) \ . \tag{9}$$

Then, in view of (9), we have:

$$E(X) = \sum_{i=0}^{m} i \cdot Pr\{X = i\} = \sum_{i=1}^{m} Pr\{X \geq i\} \leq \sum_{i=1}^{m} \min\left(1, \frac{m-i+1}{\sigma^i}\right) \ . \tag{10}$$

Let $\overline{k}$ be the smallest integer $1 \leq k < m$ such that $\frac{m-k+1}{\sigma^k} < 1$. Then from (10) we have

$$
\begin{aligned}
E(X) &\leq \sum_{i=1}^{\overline{k}-1} 1 + \sum_{i=\overline{k}}^{m} \frac{m-i+1}{\sigma^i} \leq \overline{k} - 1 + (m - \overline{k} + 1) \sum_{i=\overline{k}}^{m} \frac{1}{\sigma^i} \\
&< \overline{k} - 1 + \frac{\sigma}{\sigma - 1} \cdot \frac{m - \overline{k} + 1}{\sigma^{\overline{k}}} < \overline{k} - 1 + \frac{\sigma}{\sigma - 1} < \overline{k} + 1\,.
\end{aligned}
\tag{11}
$$

Since $\frac{m-(\overline{k}+1)+1}{\sigma^{\overline{k}+1}} \geq 1$, then $\sigma^{\overline{k}+1} \leq m - (\overline{k}+1) + 1 \leq m - 1$, so that

$$
\overline{k} + 1 < \log_\sigma m\,. \tag{12}
$$

From (11) and (12), we obtain

$$
E(X) < \log_\sigma m\,. \tag{13}
$$

Likewise, from (9) and (12) we have

$$
\begin{aligned}
\sum_{k=1}^{m} \sqrt{E(X_k)} &\leq \sum_{k=1}^{m} \sqrt{\min\left(1, \frac{m-k+1}{\sigma^k}\right)} = \sum_{k=1}^{\overline{k}-1} 1 + \sum_{k=\overline{k}}^{m} \sqrt{\frac{m-k+1}{\sigma^k}} \\
&\leq \overline{k} - 1 + \sqrt{m - \overline{k} + 1} \cdot \sum_{k=\overline{k}}^{m} \frac{1}{\sqrt{\sigma^k}} < \overline{k} - 1 + \frac{\sqrt{\sigma}}{\sqrt{\sigma} - 1} \cdot \sqrt{\frac{m - \overline{k} + 1}{\sigma^{\overline{k}}}} \\
&< \overline{k} - 1 + \frac{\sqrt{\sigma}}{\sqrt{\sigma} - 1} \leq \overline{k} + 1 < \log_\sigma m\,,
\end{aligned}
\tag{14}
$$

where $\overline{k}$ is defined as above.

Next we estimate $E(Z_i) = Pr\{P_i \sqsupseteq_{md} T\}$, for $0 \leq i \leq m-1$.

Let us denote by $\mu(i)$ the number of distinct strings which have an *md*-match with a given string of length i and whose characters are pairwise distinct. Then

$$
Pr\{P_i \sqsupseteq_{md} T\} \leq \frac{\mu(i+1)}{\sigma^{i+1}}\,.
$$

From the recursion

$$
\begin{cases}
\mu(0) = 1 \\
\mu(k+1) = \sum_{h=0}^{k} \mu(h) + \sum_{h=1}^{\lfloor \frac{k-1}{2} \rfloor} \mu(k-2h-1) \qquad \text{(for } k \geq 0\text{)}\,,
\end{cases}
$$

it is not hard to see that $\mu(i+1) \leq 3^i$, for $i = 0, 1, \ldots, m-1$, so that we have

$$
E(Z_i) = Pr\{P_i \sqsupseteq_{md} T\} \leq \frac{3^i}{\sigma^{i+1}}\,. \tag{15}
$$

Then, concerning $E(Z)$, from (15) we have

$$
E(Z) = E\left(\sum_{i=0}^{m-1} Z_i\right) = \sum_{i=0}^{m-1} E(Z_i) \leq \sum_{i=0}^{m-1} \frac{3^i}{\sigma^{i+1}} < \frac{1}{\sigma} \cdot \frac{1}{1 - \frac{3}{\sigma}} = \frac{1}{\sigma - 3} \leq 1 \tag{16}
$$

(we recall that we have assumed $\sigma \geq 4$).

Likewise, from (15) we have

$$\sum_{i=0}^{m-1} \sqrt{E(Z_i)} \le \sum_{i=0}^{m-1} \sqrt{\frac{3^i}{\sigma^{i+1}}} < \frac{1}{\sqrt{\sigma}} \cdot \frac{1}{1-\sqrt{\frac{3}{\sigma}}} = \frac{1}{\sqrt{\sigma}-\sqrt{3}} < 4\,. \tag{17}$$

From (16), (13), (14), and (17), it then follows that (8) is bounded from above by

$$(n-1)\cdot(9\log_\sigma m + 1)\,,$$

yielding a $\mathcal{O}(n \log_\sigma m)$ average-time complexity for the M-Sampling algorithm.

4.3 Space complexity

In order to evaluate the space complexity of the M-Sampling algorithm, we observe that in the worst case, during the j-th iteration of its main **for**-loop, the sets $\mathcal{F}^k_{j-k}$ and $\mathcal{S}_{j-2k}$, for $1 \le k \le \alpha$, must be kept in memory to handle translocations, as well as the sets $\mathcal{S}_{j-k}$, for $2 \le k \le \beta$, to handle inversions. However, as explained before, we do not keep the values of $\mathcal{F}^k_{j-k}$ explicitly but rather we maintain only their corresponding T-configurations of the automaton $\mathcal{A}(P)$. Thus, we need $\mathcal{O}(\alpha)$-space for the last α configurations of the automaton and $\mathcal{O}(m \max(\alpha,\beta))$-space to keep the last $\max(2\alpha,\beta)$ values of the sets $\mathcal{S}_{j-k}$, considering the maximum cardinality of each set is m. Observe also that, although the size of the DAWG is linear in m, the *end–pos*$(\cdot)$ function can require $\mathcal{O}(m^2)$-space. Therefore, the total space complexity of the M-Sampling algorithm is $\mathcal{O}(m^2)$.

5 Experimental evaluation

In this section we present some experimental results which allow to compare in terms of running times the M-Sampling algorithm, based on the DAWG approach, against its direct dynamic programming implementation. We have also included in our comparison the variant BPM-Sampling of the M-Sampling algorithm, based on the bit-parallelism technique [2], which is briefly described in Appendix A.

We remark that sets have been implemented as bit vectors also in the first two algorithms, so that *member* and *insert* operations can be performed in constant time.

Iteration over the elements of a set represented as a bit vector can then be implemented efficiently in time proportional to its cardinality by repeatedly

(a) extracting the lowest bit set,
(b) computing its index, and
(c) masking it, until there are no more bits set.

Observe also that the index of the lowest bit set of a word x can be computed very efficiently by the operation

$$\lfloor \log_2(x \;\&\; (\sim x + 1)) \rfloor\,,$$

where & and $\sim$ stand respectively for the bitwise `and` and the bitwise complementation.

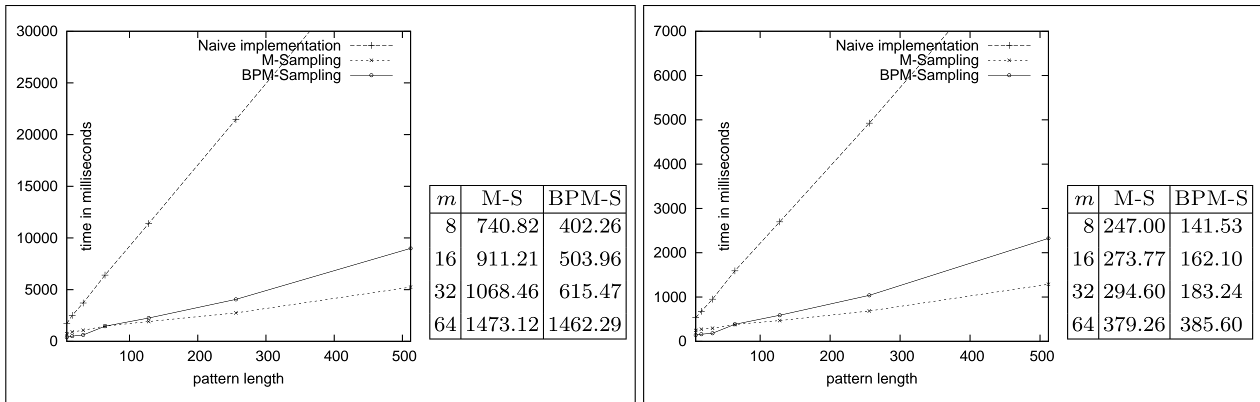
In the BPM-Sampling algorithm, bitwise operations have a $\Theta(\lceil m/w \rceil)$ complexity, since they have to update $\lceil m/w \rceil$ words. Instead, in the M-Sampling algorithm

the corresponding operations have a $\Theta(\lceil m/w \rceil + |\mathcal{S}_j|)$ complexity, because, for each word of the bit vector that encodes $\mathcal{S}_j$, it iterates over all the bits set ($|\mathcal{S}_j|$ in total). Since, on average, the sets $\mathcal{S}_j$ contain only a few elements, the average complexity of iterating over all the elements of a set is $\mathcal{O}(\lceil m/w \rceil)$.

All algorithms have been implemented in the `C` programming language and have been compiled with the `GNU C Compiler`, using the optimization options `-O2 -fno-guess-branch-probability`. The tests have been performed on a 1.5 GHz PowerPC G4 with a computer word of size 32 and running times have been measured with a hardware cycle counter, available on modern CPUs.

As input files, we used a genome sequence of 4,638,690 base pairs of *Escherichia coli* [1] and a protein sequence from the *Saccharomyces cerevisiae* genome [9].

For each input file, we have generated sets of 50 patterns of fixed length m, randomly extracted from the text, for m ranging in the set {8, 16, 32, 64, 128, 256, 512}. For each set of patterns, we have calculated the mean over the running times of the 50 runs.



m	M-S	BPM-S
8	740.82	402.26
16	911.21	503.96
32	1068.46	615.47
64	1473.12	1462.29

m	M-S	BPM-S
8	247.00	141.53
16	273.77	162.10
32	294.60	183.24
64	379.26	385.60

Figure 2. Experimental results relative to a genome sequence of *Escherichia coli* with $\sigma = 4$ (on the left) and to a protein sequence of the *Saccharomyces cerevisiae* genome with $\sigma = 20$ (on the right). To ease comparison of the M-Sampling algorithm (M-S) and the BPM-Sampling algorithm (BPM-S) for small values of m, we have also tabulated their running times.

As can be seen from the plots in Figure 2, the M-Sampling algorithm is considerably faster than its naive implementation. Indeed, even if their asymptotic time complexity is the same, the hidden constant in the naive implementation, due to the explicit computation of the sets $\mathcal{F}_j^k$ and $\mathcal{I}_j^k$, is quite large. In our experiment with a computer word of size 32, it turns out that the BPM-Sampling algorithm is faster than the M-Sampling algorithm only for $m \leq 32$, as can be observed by looking at the running times (in milliseconds) reported in the tables. As explained above, the complexity of the bitwise operations, on average, is the same for both algorithms. However, the M-Sampling algorithm scales better because it requires fewer bitwise operations. Finally, observe that the rate of growth of the M-Sampling and the BPM-Sampling aglorithm matches the average $\mathcal{O}(n \log_\sigma m)$-time complexity estimated in Section 4.2 under the assumptions of equiprobability and independence of characters.

6 Conclusions

In this paper we have presented an algorithm, based on the dynamic programming paradigm, to solve the pattern matching problem under a string distance which allows translocations of equal length adjacent factors and inversions of factors. Our algorithm, named M-Sampling, has a worst-case $\mathcal{O}(nm\max(\alpha,\beta))$-time and $\mathcal{O}(m^2)$-space complexity, where α and β are respectively the bounds on the maximum length of any factor involved in a translocation and in an inversion. Moreover, we have shown that under the assumption of equiprobability and independence of characters, the M-Sampling algorithm has a $\mathcal{O}(n\log_\sigma m)$ average-time complexity. Finally, in the appendix we have also briefly described an efficient implementation of the M-Sampling algorithm based on the bit-parallelism technique, which achieves a worst-case $\mathcal{O}(n\lceil m/w\rceil\max(\alpha,\beta))$-time and $\mathcal{O}(\sigma\lceil m/w\rceil + m\lceil m/w\rceil)$-space complexity.

We are currently investigating how to extend our approach to handle efficiently also translocations of factors which are not necessarily adjacent or of equal length and how to to compute the minimum cost, when the weights are either unitary or generic.

Acknowledgements

The authors wish to thank Paul Doukhan and Salvatore Ingrassia for helpful suggestions.

References

1. R. Arnold and T. Bell: *A corpus for the evaluation of lossless compression algorithms*, in DCC'97: Proceedings of the Conference on Data Compression, Washington, DC, USA, 1997, IEEE Computer Society, `http://corpus.canterbury.ac.nz/`.
2. R. Baeza-Yates and G. H. Gonnet: *A new approach to text searching.* Commun. ACM, 35(10) 1992.
3. A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M. T. Chen, and J. Seiferas: *The smallest automaton recognizing the subwords of a text.* Theor. Comput. Sci., 40(1) 1985, pp. 31–55.
4. M. Crochemore: *Transducers and repetitions.* Theor. Comput. Sci., 45(1) 1986, pp. 63–86.
5. M. Crochemore and W. Rytter: *Text algorithms*, Oxford University Press, 1994.
6. F. J. Damerau: *A technique for computer detection and correction of spelling errors.* Commun. ACM, 7(3) 1964, pp. 171–176.
7. V. I. Levenshtein: *Binary codes capable of correcting deletions, insertions and reversals.* Sov. Phys. Dokl., 10 1966, pp. 707–710.
8. G. Navarro: *A guided tour to approximate string matching.* ACM Comp. Surv., 33(1) 2001, pp. 31–88.
9. C. G. Nevill-Manning and I. H. Witten: *Protein is incompressible*, in DCC'99: Proceedings of the Conference on Data Compression, Washington, DC, USA, 1999, IEEE Computer Society, `http://data-compression.info/Corpora/ProteinCorpus/`.
10. E. Ukkonen: *Approximate string-matching with q-grams and maximal matches.* Theor. Comput. Sci., 92(1) 1992.
11. E. Ukkonen and D. Wood: *Approximate string matching with suffix automata.* Algorithmica, 10(5) 1993.

A A bit-parallel implementation

In this appendix we present an efficient simulation of the M-Sampling algorithm based on the bit-parallelism technique [2]. The bit-parallelism technique takes advantage of the intrinsic parallelism of the bit operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor of at most w, where w is the number of bits in the computer word. All sets are represented by vectors of m bits, where m is the length of the pattern. The i-th bit of a vector is set to 1 if the element i belongs to the corresponding set, 0 otherwise. Note that if $m \leq w$, a whole vector fits into a single computer word, whereas if $m > w$ then $\lceil m/w \rceil$ computer words are needed to represent each set.

In the following we denote with & the bitwise and, with | the bitwise or, and with $\ll$ the shift-to-left operator.

We associate to each node of the DAWG a bit vector pos. For each node q of the DAWG of P, $\mathsf{pos}(q)$ encodes the *end-pos* function, while, for each node q of the DAWG of P^{r}, $\mathsf{pos}(q)$ encodes the starting positions in P of the reversed factors represented by the node, i.e. $\{(m-1-i) \mid i \in \textit{end-pos}(q)\}$.

The bit-vectors F_j^k and I_j^k, corresponding to $\mathcal{F}_j^k$ and $\mathcal{I}_j^k$ respectively, can be computed by the following assignments:

$$\begin{aligned} \mathsf{F}_j^k &\leftarrow \mathsf{pos}(\phi(q_j, k)) \\ \mathsf{I}_j^k &\leftarrow \mathsf{pos}(\phi(q_j^{\mathsf{r}}, k)) \ll (k-1)\,. \end{aligned}$$

Each set $\mathcal{S}_j$ is mapped into a corresponding bit-vector S_j. Finally, for each character c of the alphabet Σ, a bit mask $\mathsf{B}[c]$, representing the positions of c in P, is maintained.

The algorithm scans T from left to right and, for each position $j \geq 0$, it computes the vector S_j in terms of S_{j-1}, of S_{j-2k}, F_{j-k}^k, and F_j^k, for $1 \leq k \leq l_j$, and of S_{j-k} and I_j^k for $1 \leq k \leq l_j^{\mathsf{r}}$, with the following bitwise operations:

$$\begin{aligned} &\mathsf{S}_j \leftarrow ((\mathsf{S}_{j-1} \ \ll\ 1) \mid 1) \ \&\ B[T[j]] \\ &\mathsf{S}_j \leftarrow \mathsf{S}_j \mid ((((\mathsf{S}_{j-2k} \ll k) \mid (1 \ll (k-1))) \ \&\ \mathsf{F}_j^k) \ll k) \ \&\ \mathsf{F}_{j-k}^k \\ &\mathsf{S}_j \leftarrow \mathsf{S}_j \mid (((\mathsf{S}_{j-k} \ll k) \mid (1 \ll (k-1))) \ \&\ \mathsf{I}_j^k)\,, \end{aligned}$$

corresponding respectively to the relations:

$$\begin{aligned} &\mathcal{S}_j = \{i+1 : i \in \mathcal{S}_{j-1} \cup \{-1\} \wedge P[i] = T[j]\} \\ &\mathcal{S}_j = \mathcal{S}_j \cup \{i+2k : i \in \mathcal{S}_{j-2k} \cup \{-1\} \wedge (i+k) \in \mathcal{F}_j^k \wedge (i+2k) \in \mathcal{F}_{j-k}^k\} \\ &\mathcal{S}_j = \mathcal{S}_j \cup \{i+k : i \in \mathcal{S}_{j-k} \cup \{-1\} \wedge (i+k) \in \mathcal{I}_j^k\}\,. \end{aligned}$$

During the j-th iteration, if the m-th bit of S_j is set to 1, i.e., if $\mathsf{S}_j \ \&\ 10^{m-1} \neq 0^m$, a match at position j is reported.

The resulting algorithm has a $\mathcal{O}(n \max(\alpha, \beta)\lceil m/w \rceil)$ worst-case time complexity and a $\mathcal{O}((m+\sigma)\lceil m/w \rceil)$-space complexity, where σ is the size of the alphabet. When the length of the pattern satisfies $m \leq w$, the worst-case time and space complexity become $\mathcal{O}(n \max(\alpha, \beta))$ and $\mathcal{O}(\sigma + m)$, respectively.