# Filtration Algorithms for Approximate Order-Preserving Matching

Tamanna Chhabra, Emanuele Giaquinta, and Jorma Tarhio

Department of Computer Science
Aalto University
P.O. Box 15400, FI-00076 Aalto, Finland
firstname.lastname@aalto.fi

**Abstract.** The exact order-preserving matching problem is to find all the substrings of a text $T$ which have the same length and relative order as a pattern $P$. Like string maching, order-preserving matching can be generalized by allowing the match to be approximate. In approximate order-preserving matching two strings match if they have the same relative order after removing up to $k$ elements in the same positions in both strings. In this paper we present practical solutions for this problem. The methods are based on filtration, and one of them is the first sublinear solution on average. We show by practical experiments that the new solutions are fast and efficient.

## 1 Introduction

The exact string matching problem consists in finding all the occurrences of a pattern string $P$ of length $m$ in a text string $T$ of length $n$. A recent variant of this problem is the so called order-preserving matching problem [16, 14, 1, 4, 3]. In order-preserving matching, the task is to locate all the substrings of $T$ which have the same length and relative order as $P$. This problem has applications in time series studies such as in the analysis of development of share prices in a stock market. Formally, two strings $u$ and $v$ over an ordered alphabet are *order-isomorphic* if they have the same length and $u_i \leq u_j \Leftrightarrow v_i \leq v_j$, for any $1 \leq i, j \leq |u|$. The term relative order refers to the numerical order of the numbers in the string. In $P = (3, 13, 5, 8, 21)$, the number 3 is the smallest number in the pattern, 5 is the second smallest, 8 is the third smallest number and so on. Therefore, the relative order of $P$ is $1, 4, 2, 3, 5$. For instance, if $T = (6, 10, 55, 36, 45, 66, 6, 21, 28, 15, 36)$, then it can be observed that $P$ and the substring of $T$ starting at location 2 are order-isomorphic.

There exist various solutions for the exact order-preserving matching problem. Kubica et al. [16], Belazzougui et al. [1] and Kim et al. [14] presented generalizations of the Knuth–Morris–Pratt algorithm [15] which solve the problem in $O(n + m \log m)$ time, where $m$ is the length of $P$ and $n$ is the length of $T$. Belazzougui et al. also presented a sublinear algorithm which runs in $O(\frac{n \log m}{m \log \log m})$ optimal time in the average case. Cho et al. [4] introduced a different sublinear solution based on a generalization of the Boyer–Moore–Horspool algorithm [11]. Independently, Chhabra and Tarhio [3] presented another sublinear solution based on filtration, which was proved to be faster than the previous solutions in practice. Recently, Crochemore et al. [5] presented a generalization of the suffix tree to the order-preserving case.

A natural generalization of the string matching problem can be obtained by allowing the matching to be approximate, so as to search for the substrings of the text $T$ which are *similar* to the pattern $P$. One classical instance of this kind is the string matching with $k$ mismatches problem, where the task is to find all the substrings of $T$ that are at Hamming distance at most $k$ from $P$, i.e., that match $P$ with at most $k$ mismatches. With respect to applications of order-preserving matching, approximate search seems more meaningful than exact search. Recently, Gawrychowski and Uznanski proposed a generalization of the order-preserving matching problem to the approximate case [7]. In this model, two strings are $k$-isomorphic if they have the same relative order after removing up to $k$ elements in the same positions in both strings. In the previous example, for $k = 1$, we get two matches, at location 2 and 7. The algorithm presented by Gawrychowski and Uznanski [7] runs in $O(n(\log\log m + k \log\log k))$ time and it is the only existing solution for this problem to the best of our knowledge. The idea in their method is to quickly filter out positions in $T$ which are non-matching by comparing signatures of the pattern and of the text substrings. As also acknowledged by the authors, this algorithm is rather theoretical and has not been implemented to date.

In this paper, we introduce two practical solutions for the approximate order-preserving matching problem, also based on filtration. Their worst-case time complexities are $O(nm(\lceil m/w \rceil + \log m))$ and $O(n(\lceil m/w \rceil \log\log w + m \log m))$, respectively, where $w$ is the word size in bits, and the former is the first sublinear solution on average. We also present experimental results which show that the filtering is effective and the algorithms are considerably faster than the naive one where all the first $n - m + 1$ text positions are match candidates to be verified.

The paper is organized as follows. Section 2 contains the preliminaries, Section 3 outlines the previous solution for approximate order-preserving matching, Section 4 introduces our solutions based on filtration, Section 5 contains an analysis of the first solution, Section 6 presents the results of practical experiments, and Section 7 concludes the article.

## 2 Preliminaries

Let $\Sigma$ be a finite alphabet of symbols and let $\Sigma^*$ be the set of strings over $\Sigma$. Given a string $x$, we denote by $|x|$ the length of $x$ and by $x_i$ or $x[i]$ the $i$-th symbol of $x$, for $1 \leq i \leq |x|$. The concatenation of two strings $x$ and $y$ is denoted by $xy$. Given two strings $x$ and $y$, $y$ is a substring of $x$ if there are indices $1 \leq i, j \leq |x|$ such that $y = x_i \ldots x_j$. We denote by $x^r = x_{|x|}x_{|x|-1} \ldots x_1$ the reverse of the string $x$. Given a string $x$ and a permutation $\pi$ of $\{1, 2, \ldots, |x|\}$ we denote by $\pi(x)$ the string $x_{\pi(1)}x_{\pi(2)} \ldots x_{\pi(|x|)}$.

Given two strings $x$ and $y$ of length $m$, the Hamming distance between $x$ and $y$ is $d_h(x, y) = |\{1 \leq i \leq m \mid x_i \neq y_i\}|$, and the matching statistics $M(x, y)$ is an array of $|x|$ integers where $M(x, y)[i]$ denotes the length of the longest substring of $x$ starting at position $i$ that exactly matches a substring of $y$. A factorization of a string $x$ is a sequence $F_1, F_2, \ldots, F_r$ of strings such that $x = F_1 F_2 \ldots F_r$.

The RAM model is assumed, with words of size $w$ in bits. We use some bitwise operations following the standard notation as in C language: &, |, ^, ~, $\ll$, $\gg$ for `and`, `or`, `xor`, `not`, `left shift` and `right shift`, respectively.

*Problem definition.* Two strings $u$ and $v$ over $\Sigma$ are *order-isomorphic with k mismatches* [7] or $k$-isomorphic, written $u \approx_k v$, if they have the same length and there exists a subset $K$ of $\{1, 2, \ldots, |u|\}$ of size $k$ at most, such that

$$u_i \leq u_j \Leftrightarrow v_i \leq v_j \text{ for } i, j \in \{1, 2, \ldots, |u|\} \setminus K \, .$$

The *order-preserving pattern matching with k mismatches* problem is to locate all the substrings of a text $T$ which are $k$-isomorphic with a pattern $P$.

## 3 Previous Solution

This section describes the previous solution formulated for the approximate order-preserving matching problem. The method was proposed by Gawrychowski and Uznanski [7] and is based on the signature of a sequence. The signature $S(a_1, \ldots, a_m)$ of sequence $(a_1, \ldots, a_m)$ is $(1 - pred(1), \ldots, m - pred(m))$ where $pred(i)$ is the position where the predecessor of $a_i$ occurs in the sequence. Its computation takes $O(m \log\log m)$ time by sorting. The key result is that if $(a_1, \ldots, a_m) \approx_k (b_1, \ldots, b_m)$ then the Hamming distance between $S(a_1, \ldots, a_m)$ and $S(b_1, \ldots, b_m)$ is at most $3k$. The algorithm iterates over each substring $(T_i, \ldots, T_{i+m-1})$ in the text $T$, determining its signature S$(T_i, \ldots, T_{i+m-1})$ in $O(\log\log m)$ time per position. For each position $i$, it checks if the Hamming distance between $S(T_i, \ldots, T_{i+m-1})$ and $S(P_1, \ldots, P_m)$ is greater than $3k$. This step can be done in $O(k + \log\log m)$ time. If the test is true, the position is discarded. Otherwise, the algorithm checks if $(T_i, \ldots, T_{i+m-1}) \approx_k (P_1, \ldots, P_m)$ by reducing the problem to the one of computing a heaviest increasing subsequence spanning at most $3(k + 1)$ elements. This step can be assessed in $O(k \log\log k)$ time. Therefore, the total time complexity is $O(n(\log\log m + k \log\log k))$.

## 4 Our Solutions

Given a string $u$, we denote by $\phi(u)$ the binary string of length $|u| - 1$ such that $\phi(u)_i$ is equal to 1, if $u_i < u_{i+1}$, and to 0 otherwise. The function $\phi$ is a linear approximation of the order for fast filtration. Observe that any position $2 \leq i < |u|$ in $u$ covers two positions in $\phi(u)$, $i - 1$ and $i$. Let $u$ and $v$ be two strings and consider the mismatches between the strings $\phi(u)$ and $\phi(v)$. Each mismatch position $i$ identifies a different relative order, in $u$ and $v$, between the adjacent symbols at positions $i$ and $i + 1$.

As the following Lemma shows, if $u \approx_k v$, then the Hamming distance between $\phi(u)$ and $\phi(v)$ is at most $2k$:

**Lemma 1.** *For any two strings $u$ and $v$ such that $u \approx_k v$, $d_h(\phi(u), \phi(v)) \leq 2k$.*

*Proof.* Suppose by contradiction that $d_h(\phi(u), \phi(v)) > 2k$ and let $K$ be a subset of $\{1, 2, \ldots, |u|\}$ satisfying the definition of order-isomorphism with $k$ mismatches for $u$ and $v$. Observe that for any position $i$ such that $\phi(u)_i \neq \phi(v)_i$ we have $K \cap \{i, i + 1\} \neq \emptyset$, as $u_i < u_{i+1}$ and $v_i \geq v_{i+1}$ or *vice versa*. Hence, $|K| > k$, contradicting the hypothesis. □

For example, if $u = (4, 1, 2, 3)$ and $v = (4, 5, 2, 3)$ we have $u \approx_1 v$, $\phi(u) = (0, 1, 1)$, $\phi(v) = (1, 0, 1)$, and $d_h(\phi(u), \phi(v)) = 2$. The following Lemma defines a distance measure $d_o$, based on the Hamming distance, which satisfies $d_o(\phi(u), \phi(v)) \leq k$:

**Lemma 2.** *Given two strings $x$ and $y$ of the same length, let $z_0, z_1, \ldots, z_{|x|}$ be integers such that $z_0 = 0$ and*

$$z_i = \begin{cases} 1 & \text{if } x_i \neq y_i \wedge z_{i-1} = 0 \\ 0 & \text{otherwise} \end{cases}$$

*for $i = 1, \ldots, |x|$, and let also $H(x, y) = \{i : z_i = 1\}$. Then, for any two strings $u$ and $v$ such that $u \approx_k v$, $d_o(\phi(u), \phi(v)) = |H(\phi(u), \phi(v))| \leq k$.*

*Proof.* Suppose by contradiction that $|H(\phi(u), \phi(v))| > k$ and let $K$ be a subset of $\{1, 2, \ldots, |u|\}$ satisfying the definition of order-isomorphism with $k$ mismatches for $u$ and $v$. Observe that for any position $i \in H(\phi(u), \phi(v))$ we have $K \cap \{i, i + 1\} \neq \emptyset$ and $H(\phi(u), \phi(v)) \cap \{i - 1, i + 1\} = \emptyset$. Hence, $|K| > k$, contradicting the hypothesis. □

Informally, the set $H(x, y)$ is the largest subset of the mismatch positions between $x$ and $y$ such that no two positions are consecutive. Therefore, for any two strings $u$ and $v$, there is no overlap between the positions in $u$ and $v$ covered by any two mismatches in $H(\phi(u), \phi(v))$. Our solution for approximate order-preserving matching consists of two parts: filtration and verification. First the text is filtered with an algorithm so as to locate all the potential matching locations and then the match candidates are verified using a checking routine. Lemma 2 gives a necessary condition for two strings to be $k$-isomorphic. The idea is to use it in the first phase to quickly filter out non-matching positions in $T$.

*Filtration.* For filtration, the consecutive numbers in the pattern $P$ are compared pairwise in the preprocessing phase and transformed into the binary string $\phi(P)$ where a 1 bit means the successive element is greater than the current one and a 0 bit means the opposite. Thereafter, in the search phase, an algorithm is applied to filter the text $T$ and find all the positions $i$ in $T$ such that $d_o(\phi(T_{i,m}), \phi(P)) \leq k$, where $T_{i,m} = T_i T_{i+1} \dots T_{i+m-1}$ is the substring of $T$ of length $m$ starting at position $i$. The substrings $T_{i,m}$ are encoded into the binary string $\phi(T_{i,m})$ online in the same way as the pattern. The algorithm determines approximate matches of the transformed pattern $\phi(P)$ in the similarly transformed text $\phi(T)$. As these approximate matches are just the match candidates, they need to be verified using a checking routine.

*Verification.* For verification, we use the reduction, by Gawrychowski and Uznanski, of the problem of $k$-isomorphism to the one of computing an heaviest increasing subsequence (Lemma 8, [6]). To compute the heaviest increasing subsequence, we use the algorithm of Jacobson and Vo [13], which runs in $O(m \log m)$ time for a sequence of length $m$. If we use a sorting algorithm with $O(m \log m)$ worst-case time complexity, the total time complexity of the verification is also $O(m \log m)$. In theory, the time complexity can be reduced to $O(m \log \log m)$ by using Han's sorting algorithm [9] and plugging a data structure which supports predecessor search in $O(\log \log m)$ time, such as van Emde Boas trees, in Jacobson and Vo's algorithm. Observe that in the simpler case where there are no repeated elements in $u$ and $v$, deciding whether $u \approx_k v$ can be reduced to computing the longest increasing subsequence of $\pi(v)$, where $\pi$ is a sorting permutation of $u$.

We propose two filtration algorithms, which build on ideas from two algorithms for string matching with $k$ mismatches, namely approximate SBNDM [10] and the GGF algorithm [8], respectively.

The first filtration algorithm, named AOPF1, is based on the following Lemma, which is a generalization of the method used by approximate SBNDM and first proposed by Chang and Lawler [2]:

**Lemma 3.** *Given two strings $x$ and $y$ of the same length, let $F_1 g_1 F_2 g_2 \dots F_r g_r$ be the factorization of $x$ such that $|F_i| = M(x, y)[1 + \sum_{j=1}^{i-1} |F_j g_j|]$, i.e., $F_i \in \Sigma^*$ is the longest substring of $x$ starting at position $1 + \sum_{j=1}^{i-1} |F_j g_j|$ that matches a substring of $y$, $|g_i| = 2$, for $1 \leq i < r$, and $|g_r| \leq 2$. Then $r - 1 \leq d_o(x, y)$.*

*Proof.* Let $s_i = |F_1 g_1 \dots F_{i-1} g_{i-1} F_i| + 1$ be the position of symbol $g_i[1]$ in $x$, for $1 \leq i < r$. For a given $s_i$, let $j$ be a position in the interval $[s_i - |F_i|, s_i]$ such that $x_j \neq y_j$. Observe that such a position always exists, because $F_i g_i[1]$ is not a substring of $y$. Then, we have that either $z_j = 1$ or $z_{j-1} = 1$. In the latter case, observe that $j - 1 > s_{i-1}$, for $i > 1$, since $|g_{i-1}| = 2$ and $s_i - |F_i| = s_{i-1} + 2$. Hence, for each $s_i$ we can find a distinct integer $j$ such that $z_j = 1$. □

Informally, the idea is to factorize $x$ into substrings of $y$ which cannot be extended to the right and are separated by 2-grams (pairs of symbols). Let $\hat{m} = |\phi(P)|$. The AOPF1 algorithm slides a window of size $\hat{m}$ along $T$, starting at position 1. For a given position $i$ in $T$, the algorithm scans the substring $\phi(T_{i,m})$ from right to left and computes the factors $F_j$ of $\phi(P)^r$ until either it has found $k+2$ factors or it has scanned the whole substring. In the former case, by Lemma 3, the position is skipped. Otherwise the algorithm performs an additional filtration step, namely it computes $H(\psi(\pi(T_{i,m})), \psi(\pi(P)))$, where $\psi(u)$ is the the string of length $|u|-1$ such that

$$\psi(u)_i = \begin{cases} 1 & \text{if } u_i < u_{i+1} \\ 2 & \text{if } u_i = u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

and $\pi$ is a sorting permutation of $P$, computed in the preprocessing phase. The position is then verified only if $|H(\psi(\pi(T_{i,m})), \psi(\pi(P)))| \leq k$. Indeed, Lemma 2 can be easily proved to hold also when using $\psi(\pi(u))$ and $\psi(\pi(v))$ in place of $\phi(u)$ and $\phi(v)$ (observe that, if $u \approx_k v$, then $\pi(u) \approx_k \pi(v)$). We permute the strings with $\pi$ so as to obtain a permutation of $P$ where repeated elements are clustered, which allows us to perform a finer filtering using the $\psi$ function. Note that, in principle, this additional filtration works with any permutation and ordering of repeated elements. For example, if $u = (4, 1, 2, 4)$, $v = (4, 5, 2, 3)$ and $\pi$ is the sorting permutation of $u$ $2, 3, 1, 4$, we have $\pi(u) = (1, 2, 4, 4)$, $\pi(v) = (5, 2, 4, 3)$, $\psi(\pi(u)) = (1, 1, 2)$, $\psi(\pi(v)) = (0, 1, 0)$. Note that $d_o(\psi(\pi(u)), \psi(\pi(v))) = 2$, while $d_o(\phi(u), \phi(v)) = d_o(\psi(u), \psi(v)) = 1$, as $\phi(u) = \psi(u) = (0, 1, 1)$ and $\phi(v) = \psi(v) = (1, 0, 1)$.

The factors $F_j$ are computed using the nondeterministic factor automaton of $\phi(P)^r$, which is simulated using a modified version of the bit-parallel SBNDM algorithm [17, 19]. The SBNDM algorithm is a slightly faster version of BNDM (Backward Nondeterministic DAWG Matching) [18] without bookkeeping of prefixes. The next scanned position is then $i + (\hat{m} - l) + 1$, where $l$ is the length of the longest suffix of $\phi(T_{i,m})$ with at most $k+1$ factors. The worst-case time complexity of this algorithm is $O(nm(\lceil m/w \rceil + \log m))$.

The second filtration algorithm, named AOPF2, is based on the following Lemma:

**Lemma 4.** *Given two strings $x$ and $y$ of the same length, let $B_p = \{j : j \bmod 2 = p \wedge x_j \neq y_j\}$ and*

$$H'(x, y) = B_0 \cup B_1 \setminus (\{j-1 : j \in B_0\} \cup \{j+1 : j \in B_0\})$$

*Then $|H'(x, y)| \leq d_o(x, y)$.*

*Proof.* Let $i \in H'(x, y)$. Observe that, by definition, either $i$ is even or $x_{i-1} = y_{i-1}$. Indeed, if $i-1$ is even and $x_{i-1} \neq y_{i-1}$ then $i \in \{j+1 : j \in B_0\}$ and $i \notin H'(x, y)$. Since $x_i \neq y_i$, we have $z_i = 1$ or $z_{i-1} = 1$. If $z_{i-1} = 1$ then $i$ must be even and therefore $i-1 \in \{j-1 : j \in B_0\}$ and $i-1 \notin H'(x, y)$. Hence, for each $i \in H'(x, y)$ we can find a distinct integer $j$ such that $z_j = 1$. □

Informally, the set $H'(x, y)$ is the subset of the mismatch positions between $x$ and $y$ such that for each even position we exclude the two adjacent (odd) positions. For example, if $u = (4, 1, 2, 3)$ and $v = (4, 5, 3, 2)$ we have $u \approx_2 v$, $\phi(u) = (0, 1, 1)$, $\phi(v) = (1, 0, 0)$, $H(\phi(u), \phi(v)) = \{1, 3\}$, $H'(\phi(u), \phi(v)) = \{2\}$. In the preprocessing, the AOPF2 algorithm computes the bit-vector $X$ of $\hat{m}$ bits

```
AOPF1($P, T, k$)
1.  $\hat{m} \leftarrow |P| - 1$
2.  $B[0] \leftarrow B[1] \leftarrow 0^{\hat{m}}$
3.  $E \leftarrow 1^{\hat{m}}$
4.  for $i \leftarrow 1$ to $\hat{m}$ do
5.      $c \leftarrow 0$
6.      if $P_i < P_{i+1}$ then $c \leftarrow 1$
7.      $B[c] \leftarrow B[c] \mid (1 \ll (i-1))$
8.  $i \leftarrow \hat{m} + 1$
9.  while $i \leq |T|$ do
10.     $(e, j, D) \leftarrow (0, 0, E)$
11.     while $e \leq k$ and $j < \hat{m}$ do
12.         $j \leftarrow j + 1$
13.         $c \leftarrow 0$
14.         if $T_{i-j} < T_{i-j+1}$ then $c \leftarrow 1$
15.         $D \leftarrow (D \gg 1)$ & $B[c]$
16.         if $D = 0^{\hat{m}}$ then
17.             $(e, j, D) \leftarrow (e + 1, j + 1, E)$
18.     if $j \geq \hat{m}$ and $e \leq k$ then psi-filter($P, T, i$)
19.     $i \leftarrow i + (\hat{m} - \min(j, \hat{m})) + 1$
```

```
AOPF2($P, T, k$)
1.  $\hat{m} \leftarrow |P| - 1$
2.  $X \leftarrow Y \leftarrow 0^{\hat{m}}$
3.  $C[0] \leftarrow C[1] \leftarrow 0^{\hat{m}}$
4.  for $i \leftarrow 1$ to $\hat{m}$ do
5.      $j \leftarrow i \bmod 2$
6.      $C[j] \leftarrow C[j] \mid (1 \ll (i-1))$
7.      if $P_i < P_{i+1}$ then
8.          $X \leftarrow X \mid (1 \ll (i-1))$
9.      if $T_i < T_{i+1}$ then
10.         $Y \leftarrow Y \mid (1 \ll (i-1))$
11. for $i \leftarrow \hat{m}$ to $|T| - 1$ do
12.     if $T_i < T_{i+1}$ then
13.         $Y \leftarrow Y \mid (1 \ll (\hat{m}-1))$
14.     $B_0 \leftarrow (X \wedge Y)$ & $C[0]$
15.     $B_1 \leftarrow (X \wedge Y)$ & $C[1]$
16.     $W \leftarrow (B_0 \ll 1) \mid (B_0 \gg 1)$
17.     $e \leftarrow \textsc{sa}(B_0 \mid B_1$ & $\sim W)$
18.     if $e \leq k$ then verify($P, T, i$)
19.     $Y \leftarrow Y \gg 1$
```

**Fig. 1.** The AOPF1 and AOPF2 algorithms for the approximate order-preserving matching problem.

such that the $i$-th bit is set to 1 if $P_i < P_{i+1}$ and to 0 otherwise. In other words $X$ is the bit-vector encoding of $\phi(P)$. The algorithm then scans the text from left to right and maintains the bit-vector encoding $Y$ of $\phi(T_{i,m})$, for $i = 1, \ldots, |T|$. For a given position $i$ in $T$, the bit-vector encodings of $B_0$ and $B_1$ are computed as $(X \wedge Y)$ & $01\ldots01$ and $(X \wedge Y)$ & $10\ldots10$, respectively. Then, we have that the bit-vector encoding of $H'(\phi(P), \phi(T_{i,m}))$ is equal to

$$B_0 \mid B_1 \;\&\; {\sim}((B_0 \ll 1) \mid (B_0 \gg 1)) \,.$$

The size of $H'(\phi(P), \phi(T_{i,m}))$ is computed using the sideways addition operation SA on each word of the resulting bit-vector. Given a word $X$, the sideways addition of $X$ returns the number of bits set in $X$. This operation can be computed in $O(\log\log w)$ time in the word-RAM model [20] and is also available as a POPCNT instruction in recent processors of the x86 family. The worst-case time complexity of this algorithm is $O(n(\lceil m/w \rceil \log\log w + m \log m))$. The space complexity of both algorithms is $O(\lceil m/w \rceil)$. The pseudocode of the two algorithms is shown in Fig. 1. The **psi-filter** procedure called in AOPF1 at line 18 performs the additional filtration step based on the $\psi$ function and calls the verification procedure, if necessary.

## 5 Analysis

In this section we analyze the average-case running time of the AOPF1 algorithm, and show that it is sublinear on average if $k$ is not too large. Suppose that $T$ is a uniformly random string over an alphabet $\Sigma$ of size $\sigma$. The string $\phi(T)$ is not uniformly random in general as $Pr[\phi(T)_i = 1] = (\sigma + 1)/(2\sigma)$ and $Pr[\phi(T)_i = 0] = (\sigma - 1)/(2\sigma)$. We make the simplifying assumption that either

**Table 1.** Execution times of the algorithms (in 10 of milliseconds) for Dow Jones data and Helsinki temperature data.

Dow Jones

$k = 1$

| m | AOPF1 | AOPF1b | AOPF2 | AOPF2b | naive |
|---|---|---|---|---|---|
| 5 | 21.5 | 26.2 | $\boxed{16.9}$ | 22.8 | 24.4 |
| 10 | $\boxed{4.1}$ | 10.1 | 6.2 | 11.8 | 90.8 |
| 15 | $\boxed{1.7}$ | 5.1 | 3.0 | 3.9 | 172.2 |
| 20 | $\boxed{1.0}$ | 2.6 | 2.8 | 2.9 | 270.3 |
| 25 | $\boxed{0.7}$ | 1.7 | 2.8 | 2.9 | 374.0 |
| 30 | $\boxed{0.6}$ | 1.0 | 2.8 | 2.6 | 473.9 |
| 50 | $\boxed{0.3}$ | 0.5 | 2.8 | 2.5 | 1069.5 |

$k = 2$

| m | AOPF1 | AOPF1b | AOPF2 | AOPF2b | naive |
|---|---|---|---|---|---|
| 5 | 30.3 | 34.3 | 28.9 | 31.6 | $\boxed{27.4}$ |
| 10 | $\boxed{28.9}$ | 36.0 | 31.3 | 54.1 | 96.5 |
| 15 | 9.7 | 21.7 | $\boxed{7.3}$ | 19.4 | 172.0 |
| 20 | 3.8 | 19.6 | $\boxed{3.3}$ | 5.5 | 261.5 |
| 25 | $\boxed{2.0}$ | 11.9 | 3.0 | 3.2 | 372.3 |
| 30 | $\boxed{1.4}$ | 6.1 | 3.1 | 2.7 | 465.8 |
| 50 | $\boxed{0.6}$ | 1.5 | 3.1 | 2.7 | 1048.9 |

$k = 3$

| m | AOPF1 | AOPF1b | AOPF2 | AOPF2b | naive |
|---|---|---|---|---|---|
| 5 | 32.3 | 35.4 | 30.8 | 32.0 | $\boxed{28.0}$ |
| 10 | 88.4 | 96.3 | $\boxed{80.6}$ | 95.7 | 98.9 |
| 15 | $\boxed{31.0}$ | 34.2 | 31.7 | 76.3 | 174.4 |
| 20 | 18.2 | 30.7 | $\boxed{7.9}$ | 26.8 | 266.5 |
| 25 | 8.1 | 32.6 | $\boxed{3.6}$ | 7.5 | 380.6 |
| 30 | 3.8 | 27.7 | $\boxed{3.0}$ | 3.1 | 454.2 |
| 50 | $\boxed{1.1}$ | 4.5 | 3.0 | 2.7 | 963.2 |

Helsinki Temperatures

$k = 1$

| m | AOPF1 | AOPF1b | AOPF2 | AOPF2b | naive |
|---|---|---|---|---|---|
| 5 | 12.8 | 13.4 | $\boxed{9.5}$ | 11.3 | 13.1 |
| 10 | $\boxed{2.1}$ | 4.9 | 3.5 | 5.8 | 47.0 |
| 15 | $\boxed{0.9}$ | 2.5 | 1.7 | 1.8 | 84.2 |
| 20 | $\boxed{0.5}$ | 1.3 | 1.5 | 1.4 | 129.8 |
| 25 | $\boxed{0.3}$ | 0.8 | 1.5 | 1.4 | 182.9 |
| 30 | $\boxed{0.2}$ | 0.5 | 1.4 | 1.2 | 233.1 |
| 50 | $\boxed{0.1}$ | 0.2 | 1.4 | 1.2 | 522.6 |

$k = 2$

| m | AOPF1 | AOPF1b | AOPF2 | AOPF2b | naive |
|---|---|---|---|---|---|
| 5 | 16.8 | 17.3 | 15.6 | 15.7 | $\boxed{13.1}$ |
| 10 | $\boxed{14.9}$ | 16.8 | 16.2 | 27.5 | 46.7 |
| 15 | 5.0 | 10.4 | $\boxed{3.8}$ | 9.8 | 84.0 |
| 20 | 2.0 | 9.7 | $\boxed{1.7}$ | 2.9 | 129.8 |
| 25 | $\boxed{1.0}$ | 6.0 | 1.6 | 1.5 | 182.8 |
| 30 | $\boxed{0.6}$ | 2.8 | 1.4 | 1.3 | 225.2 |
| 50 | $\boxed{0.2}$ | 0.7 | 1.5 | 1.2 | 503.0 |

$k = 3$

| m | AOPF1 | AOPF1b | AOPF2 | AOPF2b | naive |
|---|---|---|---|---|---|
| 5 | 17.2 | 17.5 | 15.7 | 16.1 | $\boxed{13.0}$ |
| 10 | 46.1 | 47.9 | 41.2 | 48.2 | $\boxed{39.6}$ |
| 15 | $\boxed{14.5}$ | 15.5 | 16.1 | 38.6 | 70.5 |
| 20 | 9.2 | 14.9 | $\boxed{3.9}$ | 13.7 | 108.4 |
| 25 | 4.2 | 16.5 | $\boxed{1.8}$ | 3.8 | 152.2 |
| 30 | 1.7 | 12.7 | $\boxed{1.4}$ | $\boxed{1.4}$ | 226.5 |
| 50 | $\boxed{0.5}$ | 2.1 | 1.4 | 1.2 | 507.5 |

all the symbols of $T$ are distinct, in which case the distribution becomes uniform, or that the alphabet is large enough so that the distribution is arbitrarily close to uniform. Assume that $k < m/(\log_\sigma m + O(1))$ and let $X_j$ be the random variable corresponding to the length of factor $F_j$. By the "Main Lemma" of Chang and Lawler [2] we obtain that

1. the probability $Pr[X_1 + X_2 + \ldots + X_{k+1} \geq m]$ of a verification using Lemma 3 is less than $1/m^3$;
2. $E[X_j] < \log_\sigma m + 3$;

since skipping two symbols instead of one between each factor $F_j$ does not invalidate the assumption that the variables $X_j$ are independent and identically distributed. By (1), the total verification time is thus $O((n/m^3)m \log m)$. Instead, by (2), it follows that the average number of symbols scanned in a single window and the average shift length are equal to $(k+1)(\log_\sigma m + 3)$ and $m - (k+1)(\log_\sigma m + 3)$, respectively. From this we obtain that the average filtering time is $O((n/m)k \log_\sigma m)$ for the aformentioned choice of $k$. Hence, the running time of both phases is sublinear on average.

## 6 Experiments

We tested AOPF1 and AOPF2 against the following algorithms:

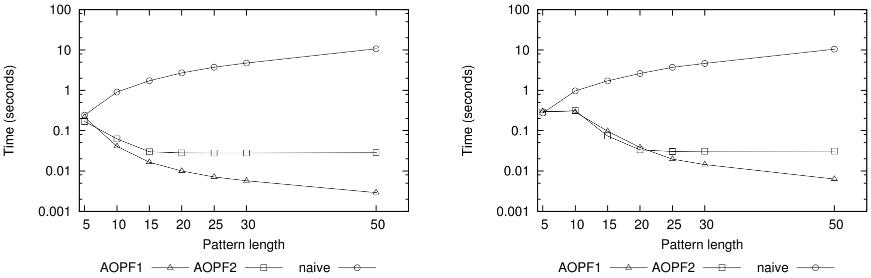

**Fig. 2.** Plot of the execution times of the algorithms for Dow Jones data with $k = 1$ (left) and $k = 2$ (right).

- AOPF1b: the filtration method based on the Hamming distance using Approximate SBNDM;
- AOPF2b: the filtration method based on the Hamming distance using the GGF algorithm;
- naive: the naive method where all the text positions are checked.

Note that the AOPF1b and AOPF2b algorithms must use $2k$ as bound on the number of mismatches. In the AOPF1b algorithm we employ the same additional filtration step used in AOPF1.

The tests were run in 64-bit mode on Intel 2.70 GHz i7 processor with 16 GB of memory running Ubuntu 12.10. All the algorithms were implemented in C and run in the testing framework of Hume and Sunday [12].

We performed the tests on two sequences of real data: the Dow Jones index and the Helsinki temperatures time series. The Dow Jones data contains $15,248$ integers corresponding to the daily values of the stock index in the years 1950–2011 and the Helsinki temperature data contains $6,818$ integers representing the daily mean temperatures in Fahrenheit (multiplied by ten) in Helsinki in the years 1995–2005. For each sequence we generated sets of 200 patterns, randomly extracted from the text, of fixed length $m \in \{5, 10, 15, 20, 25, 30, 50\}$. Table 1 shows the average execution times, over 99 repeated runs, of the algorithms for Dow Jones data and Helsinki temperature data in 10 of milliseconds for $k \in \{1, 2, 3\}$. In addition, a graph of the times for the Dow Jones data and $k = 1, 2$ (with logarithmic scale on the y axis) is shown in Fig 2. All the algorithms use the verification method described in Sect. 3.

From the results, we observe that i) AOPF1 and AOPF2 are significantly faster than the naive method, except for the cases $m = 5$ and also $m = 10$ for $k = 3$ where they are either faster or comparable; ii) AOPF1 is always faster than AOPF1b; iii) AOPF2 is either faster or comparable to AOPF2b. Consider the cases where the algorithms provide a significant speedup over the naive method, namely $m \geq 10$ for $k = 1, 2$ and $m \geq 15$ for $k = 3$. For $k = 1$, AOPF1 is the fastest algorithm, while for higher values of $k$ either AOPF1 or AOPF2 is the fastest depending on the value of $m$. In particular, there is a region of $m$, $\{15, 20\}$ for $k = 2$ and $\{20, 25, 30\}$ for $k = 3$, where AOPF2 obtains the best running time. Note that the execution time of the naive algorithm is proportional to $m$, as it verifies all the positions. In the case of AOPF1 and AOPF2, the execution time drops notably after a threshold value for $m$ which depends on $k$. In particular, for $k = 1$ the threshold value for $m$ is 5 while for $k = 2, 3$ it is 10. The AOPF1b and AOPF2b also shows this behaviour, although the drop in the running time is not as significant. Note that although the filtration phase of AOPF2 is linear, the total time of AOPF2 decreases with a fixed $k$ when $m$ grows. This is due to the fact that the verification probability and the verification time decrease on average when $m$ grows.

## 7 Concluding Remarks

In this paper we have presented two practical solutions, based on filtration, for the recently introduced approximate order-preserving matching problem. Both algorithms are effective in practice, as shown by experimental evaluation, and one of them is the first sublinear solution on average, provided that the number of errors is not too large.

## References

1. D. Belazzougui, A. Pierrot, M. Raffinot, and S. Vialette. Single and multiple consecutive permutation motif search. In *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, pages 66–77, 2013.
2. W. I. Chang and E. L. Lawler. Sublinear approximate string matching and biological applications. *Algorithmica*, 12(4/5):327–344, 1994.
3. T. Chhabra and J. Tarhio. Order-preserving matching with filtration. In *Experimental Algorithms - 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29 - July 1, 2014. Proceedings*, pages 307–314, 2014.
4. S. Cho, J. C. Na, K. Park, and J. S. Sim. A fast algorithm for order-preserving pattern matching. *Inf. Process. Lett.*, 115(2):397–402, 2015.
5. M. Crochemore, C. S. Iliopoulos, T. Kociumaka, M. Kubica, A. Langiu, S. P. Pissis, J. Radoszewski, W. Rytter, and T. Walen. Order-preserving incomplete suffix trees and order-preserving indexes. In *String Processing and Information Retrieval - 20th International Symposium, SPIRE 2013, Jerusalem, Israel, October 7-9, 2013, Proceedings*, pages 84–95, 2013.
6. P. Gawrychowski and P. Uznanski. Order-preserving pattern matching with k mismatches. *CoRR*, abs/1309.6453, 2013.
7. P. Gawrychowski and P. Uznanski. Order-preserving pattern matching with k mismatches. In *Combinatorial Pattern Matching - 25th Annual Symposium, CPM 2014, Moscow, Russia, June 16-18, 2014. Proceedings*, pages 130–139, 2014.
8. E. Giaquinta, S. Grabowski, and K. Fredriksson. Approximate pattern matching with k-mismatches in packed text. *Inf. Process. Lett.*, 113(19-21):693–697, 2013.
9. Y. Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. *J. Algorithms*, 50(1):96–105, 2004.

10. T. Hirvola and J. Tarhio. Approximate online matching of circular strings. In *Experimental Algorithms - 13th International Symposium, SEA 2014, Copenhagen, Denmark, June 29 - July 1, 2014. Proceedings*, pages 315–325, 2014.
11. R. N. Horspool. Practical fast searching in strings. *Softw., Pract. Exper.*, 10(6):501–506, 1980.
12. A. Hume and D. Sunday. Fast string searching. *Softw., Pract. Exper.*, 21(11):1221–1248, 1991.
13. G. Jacobson and K. Vo. Heaviest increasing/common subsequence problems. In *Combinatorial Pattern Matching, Third Annual Symposium, CPM 92, Tucson, Arizona, USA, April 29 - May 1, 1992, Proceedings*, pages 52–66, 1992.
14. J. Kim, P. Eades, R. Fleischer, S. Hong, C. S. Iliopoulos, K. Park, S. J. Puglisi, and T. Tokuyama. Order-preserving matching. *Theor. Comput. Sci.*, 525:68–79, 2014.
15. D. E. Knuth, J. H. M. Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.
16. M. Kubica, T. Kulczynski, J. Radoszewski, W. Rytter, and T. Walen. A linear time algorithm for consecutive permutation pattern matching. *Inf. Process. Lett.*, 113(12):430–433, 2013.
17. G. Navarro. Nr-grep: a fast and flexible pattern-matching tool. *Softw., Pract. Exper.*, 31(13):1265–1312, 2001.
18. G. Navarro and M. Raffinot. Fast and flexible string matching by combining bit-parallelism and suffix automata. *ACM Journal of Experimental Algorithmics*, 5:4, 2000.
19. H. Peltola and J. Tarhio. Alternative algorithms for bit-parallel string matching. In *String Processing and Information Retrieval, 10th International Symposium, SPIRE 2003, Manaus, Brazil, October 8-10, 2003, Proceedings*, pages 80–94, 2003.
20. S. Vigna. Broadword implementation of rank/select queries. In *Experimental Algorithms, 7th International Workshop, WEA 2008, Provincetown, MA, USA, May 30-June 1, 2008, Proceedings*, pages 154–168, 2008.