Information Processing Letters 116 (2016) 71–74

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A filtration method for order-preserving matching

Tamanna Chhabra, Jorma Tarhio [*]

Department of Computer Science, Aalto University, P.O. Box 15400, FI-00076 Aalto, Finland

ARTICLE INFO

Article history:
Received 5 March 2015
Received in revised form 16 September 2015
Accepted 19 October 2015
Available online 2 November 2015
Communicated by Tsan-sheng Hsu

Keywords:
Algorithms
Combinatorial problems
Order-preserving matching
String searching

ABSTRACT

The problem of order-preserving matching has gained attention lately. The text and the pattern consist of numbers. The task is to find all the substrings in the text which have the same length and relative order as the pattern. The problem has applications in analysis of time series. We present a new sublinear solution based on filtration. Any algorithm for exact string matching can be used as a filtering method. If the filtration algorithm is sublinear, the total method is sublinear on average. We show by practical experiments that the new solution is more efficient than earlier algorithms.

1. Introduction

String matching [1] is a widely known problem in Computer Science. Given a text T of length n and a pattern P of length m, both being strings over a finite alphabet Σ, the task of string matching is to find all the occurrences of P in T. The problem of order-preserving matching [2–6] has gained attention lately. It considers strings of numbers. The task is to find all the substrings (also called factors) u in T which have the same relative order as P, and $|u| = |P|$. Suppose $P = (10, 22, 15, 30, 20, 18, 27)$ and $T = (22, 85, 79, 24, 42, 27, 62, 40, 32, 47, 69, 55, 25)$, then the relative order of P matches the substring $u = (24, 42, 27, 62, 40, 32, 47)$ of T, see Fig. 1.

Several online [7,5,3,4] and one offline solution [2] have been proposed for order-preserving matching. Kubica et al. [4] and Kim et al. [3] presented solutions based on the Knuth–Morris–Pratt algorithm (KMP) [8]. Later, Cho et al. [5,6] gave a sublinear solution based on the bad character heuristic of the Boyer–Moore algorithm [9]. Almost at the same time, Belazzougui et al. [7] derived an optimal sublinear solution. We will present a new practical solution based on filtration. We form a modified pattern and use an algorithm for exact string matching as a filtration method. Our approach is simpler and in practice more efficient than earlier solutions. We transform the original pattern P into a binary string P' expressing increases (1), equalities (0), and decreases (0) between subsequent pattern positions. Then we search for P' in the analogously transformed text T'. For example, $P' = 101001$ corresponds to $P = (10, 22, 15, 30, 20, 18, 27)$ and $T' = 100101001100$ to T above. Each occurrence is a match candidate which is verified following the numerical order of the positions of the original pattern P. Note that in this approach any algorithm for exact string matching can be used as a filtration method. If the filtration algorithm is sublinear and the text is transformed on line, the total method is sublinear on average.

We made experiments with two sublinear string matching algorithms and two linear string matching algorithms as the filtering method. Our approach with sublinear filters was considerably faster than the algorithm by Cho et al. [5], which is the first sublinear solution of the problem.

[*] Corresponding author.
E-mail addresses: tamanna.chhabra@aalto.fi (T. Chhabra), jorma.tarhio@aalto.fi (J. Tarhio).

http://dx.doi.org/10.1016/j.ipl.2015.10.005
0020-0190/© 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

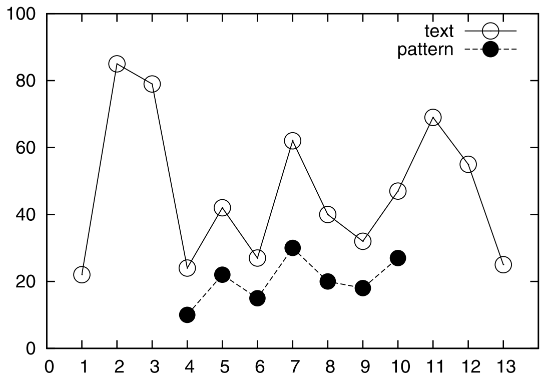

Fig. 1. Example of order-preserving matching.

The paper is organized as follows. Section 2 describes the previous solutions for order-preserving matching, Section 3 presents our solution based on filtration, Section 4 analyses the new approach, Section 5 presents and discusses the results of practical experiments, and Section 6 concludes the article.

2. Previous solutions

In the first KMP approach presented by Kubica et al. [4], the fail function in the KMP algorithm is modified to compute the order-borders table. This can be achieved in linear time. The KMP algorithm is mutated such that it determines if the text contains substring with the same relative order as that of the pattern using the order-borders table. This computation can be done in linear time. Hence, the total time complexity of the method is linear.

The second KMP approach by Kim et al. [3] is based on the prefix representation. The prefix representation is based on finding the rank of each number in the prefix. The time complexity of the method is $O(n \log m)$. This approach is further optimized using the nearest neighbor representation to overcome the overhead involved in computing the rank function. The time complexity of the improved version is $O(n + m \log m)$.

The BMH approach by Cho et al. [5] is based on the bad character rule applied to q-grams, i.e. strings of q characters. A q-gram is treated as a single character in order to make shifts longer. In this way, a large amount of text can be skipped for long patterns, and the algorithm is sublinear on average. The standard version works in $O(mn)$ in the worst case. Later, Cho et al. [6] introduced a linear version, which has been combined with KMP in order to guarantee linear behavior in the worst case.

3. Our solution

In Section 1 we gave an informal description of order-preserving matching. Let us define the problem formally.

Problem definition Two strings $u = u_1 u_2 \cdots u_m$ and $v = v_1 v_2 \cdots v_m$ of the same length over Σ are called *order-isomorphic* [3,4], written $u \approx v$, if

$$u_i \le u_j \Leftrightarrow v_i \le v_j \text{ for } 1 \le i, j \le m.$$

In the *order-preserving pattern matching problem*, the task is to find all the substrings of $T = t_1 t_2 \cdots t_n$ which are order-isomorphic with $P = p_1 p_2 \cdots p_m$.

Our solution for order-preserving matching consists of two phases: filtration and verification. First the text is transformed to a bit string which is filtered with some exact string matching algorithm. In the second phase the match candidates are verified using a checking routine.

Filtration For filtration, the consecutive numbers in the pattern $P = p_1 p_2 \cdots p_m$ are compared pairwise in the preprocessing phase and the result is encoded as a modified pattern $P' = b_1 b_2 \cdots b_{m-1}$ of binary numbers: b_i is 1 if $p_i < p_{i+1}$ holds, otherwise b_i is 0. In the search phase, some algorithm for exact string matching (let us call it A) is applied to filter out the text. When Algorithm A reads an alignment window of the original text, the text is encoded incrementally online in the same way as the pattern. Algorithm A is run as if the whole text would have been encoded. Because Algorithm A may recognize an occurrence of P' which does not correspond to an actual match of P in T, each occurrence of P' is only a match candidate which should be verified. It is clear that this filtration method cannot skip any occurrence of P in T.

Verification During preprocessing the pattern, the numbers of the pattern $P = p_1 p_2 \cdots p_m$ are sorted. The result is an auxiliary table r: $p_{r[i]} \le p_{r[j]}$ holds for each pair $i < j$ and $p_{r[1]}$ is the smallest number in P. In addition, we need a binary vector E representing the equalities: $E[i] = 1$ denotes that $p_{r[i]} = p_{r[i+1]}$ holds. The match candidates found by Algorithm A are traversed in accordance with the table r. If the candidate starts from t_j in T, the first comparison is done between $t_{j-1+r[1]}$ and $t_{j-1+r[2]}$. There is a mismatch when

$$t_{j-1+r[i]} > t_{j-1+r[i+1]} \text{ or}$$

$$(t_{j-1+r[i]} = t_{j-1+r[i+1]} \text{ and } E[i] = 0) \text{ or}$$

$$(t_{j-1+r[i]} < t_{j-1+r[i+1]} \text{ and } E[i] = 1)$$

is satisfied. The candidate is discarded when a mismatch is encountered. Verification is efficient because sorting is done only once during preprocessing.

Remark We use binary numbers in encoding. We also tried encoding of three numbers 0, 1, and 2 corresponding to '<', '=', and '>', but the binary approach was faster in practice, because testing of one condition is faster than testing of two conditions. Also the frequency of nearby equalities is low in real data.

4. Analysis

We will prove that our approach is sublinear in the average case, if the filtration algorithm is sublinear. Sublinearity means that on average all the characters in the text are not examined.

Let us assume that the numbers in P and T are integers and they are statistically independent of each other and the distribution of numbers is discrete uniform. Let

P' and T' be the transformed pattern and text. Let c be the count of the integer range (i.e. the alphabet size). The probability of one in a position of P' or T' (as a result of a comparison) is $p = (c^2/2 - c/2)/c^2 = (c-1)/2c$, because there are c^2 integer pairs and c equalities. So the probability of a character match q is

$$p^2 + (1-p)^2 = 2p(p-1) + 1 = 1 - \frac{c-1}{c} \cdot \frac{c+1}{2c}$$
$$= 1 - \frac{c^2-1}{2c^2} = \frac{1}{2} + \frac{1}{2c^2}.$$

Because adjacent positions in $P' = b_1 b_2 \cdots b_{m-1}$ and in T' are not independent, let us consider matching of a relaxed pattern $P'' = b_1\$b_3\$b_5 \cdots \$b_s$, which contains every other character of P' and where \$ matches both 0 and 1 and s is $2\lfloor m/2 \rfloor - 1$. The probability of a match of P'' at a certain position of T' is smaller than $q^{(m-1)/2}$, which approaches to zero, when m grows. This is true even for $c = 2$. The probability of a match of P' (i.e. a match candidate of P) is smaller than the probability of a match of P''. This means that the verification time approaches zero when m grows, and the filtration time dominates. If the filtration method is sublinear, the total algorithm is sublinear.

The preprocessing phase requires $O(m \log m)$ time due to sorting of the pattern positions. The space requirement is $O(m)$.

In the worst case, the total algorithm requires $O(nm)$ time if, for example, P' is 1^{m-1} and T' is 1^{n-1}. If the filtration method is linear in the worst case, the total algorithm can be modified to work in linear time by combining a linear solution [4,3] L with it. When the distance of starting positions of subsequent match candidates is less than $m/2$, next $2m$ positions are processed with L.

5. Experiments

We tested four string matching algorithms as filtration methods for order-preserving matching. Two of them, SBNDM2 and SBNDM4 [10] are based on the Backward Nondeterministic DAWG Matching (BNDM) algorithm [1]. In BNDM, each alignment window is processed from right to left like in the Boyer–Moore algorithm [9] by simulating the nondeterministic automaton of the reversed pattern with bitparallelism. SBNDMq starts the processing of each alignment window by reading a q-gram. The third algorithm is Fast Shift-Or (FSO) [11]. We utilized a version of FSO coded by B. Ďurian [10]. FSO was selected because it is fast on short binary patterns [10]. The fourth algorithm is the KMP algorithm [8]; together with verification it was supposed to approximate the two earlier methods [3,4] based on KMP. Of the algorithms, SBNDM2 and SBNDM4 are sublinear, whereas FSO and KMP are linear.

The tests were run on Intel 2.70 GHz i7 processor with 16 GB of memory running Ubuntu 12.10. All the algorithms were implemented in C in the 64-bit mode and run in the testing framework of Hume and Sunday [12]. Our solution based on filtration was compared with the BMH approach by Cho et al. [5] (the authors generously let us use their implementation). Because the BMH approach was clearly faster than the KMP-based algorithm [3] and slightly faster than the linear version of the BMH approach in the tests [6], we tested only the first mentioned algorithm.

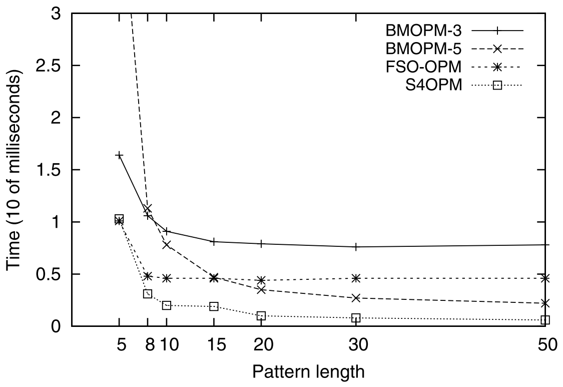

Fig. 2. Execution times of algorithms for the Dow Jones data.

For testing we used three texts: a random text and two real texts, which were time series of the Dow Jones index and Helsinki temperatures. The random data contains 1,000,000 random integers between 0 and 2^{30}. The Dow Jones data contains 15,248 integers pertaining to the daily values of the stock index in the years 1950–2011 and the Helsinki temperature data contains 6818 integers referring to the daily mean temperatures in Fahrenheit (multiplied by ten) in Helsinki in the years 1995–2005. From each text we picked randomly patterns of length 5, 8, 10, 15, 20, 30, and 50. Each set contains 1000 patterns for the random text and 200 patterns for the real texts. Table 1 shows the average execution times per pattern of all the algorithms. The unit is 10 milliseconds for real data and one second for random data. In addition, a graph on times for the Dow Jones data is shown in Fig. 2. The real texts were tested with 180 repeated runs and the random text was tested with 60 repeated runs. In Table 1, S2OPM represents the algorithm based on SBNDM2 filtration, S4OPM represents the algorithm based on SBNDM4 filtration, BMOPM-q represents the BMH approach [5] for $q = 3, 4, 5$, KOPM represents the algorithm based on KMP filtration and FSO-OPM represents the algorithm based on Fast Shift-Or.

From Table 1, it can be seen that in case of real data, S4OPM is a clear winner for most tested values of m, and FSO-OPM is the fastest for $m = 5$. With all the three data sets, S2OPM is mostly slower than S4OPM but its execution time approaches that of S4OPM as the value of m increases. For $m = 50$, the execution times of S2OPM and S4OPM are almost equal. Relatively, S2OPM and S4OPM perform better on the real data than on the random data. In the case of Helsinki daily temperatures, the execution times of S2OPM and S4OPM are comparable. In the case of random data, S2OPM and S4OPM are the best for $m = 5$ and FSO-OPM is the best for $m = 8, 10$.

6. Concluding remarks

We introduced a new practical solution based on filtration for order-preserving matching. Any exact string matching algorithm can be used as the filtration algorithm. In this paper, we utilized SBNDM2, SBNDM4, FSO,

Table 1
Execution times of algorithms (random: in seconds, others: in 10 milliseconds). The best time for each pattern set has been boxed.

Data	Algorithm	5	8	10	15	20	30	50
DOW	KOPM	2.02	1.94	1.94	2.00	1.94	1.96	1.95
	BMOPM-3	1.64	1.06	0.91	0.81	0.79	0.76	0.78
	BMOPM-4	2.16	0.96	0.72	0.50	0.41	0.34	0.31
	BMOPM-5	4.34	1.13	0.78	0.47	0.35	0.27	0.22
	FSO-OPM	$\boxed{1.01}$	0.48	0.46	0.46	0.44	0.46	0.46
	S2OPM	1.05	0.58	0.41	0.28	0.16	0.09	$\boxed{0.06}$
	S4OPM	1.03	$\boxed{0.31}$	$\boxed{0.20}$	$\boxed{0.19}$	$\boxed{0.10}$	$\boxed{0.08}$	$\boxed{0.06}$
Hel temp	KOPM	0.85	0.81	0.75	0.77	0.76	0.76	0.76
	BMOPM-3	0.70	0.46	0.46	0.40	0.39	0.39	0.44
	BMOPM-4	0.91	0.40	0.42	0.28	0.24	0.21	0.21
	BMOPM-5	1.87	0.49	0.52	0.31	0.23	0.18	0.17
	FSO-OPM	$\boxed{0.34}$	0.21	0.22	0.22	0.21	0.21	0.21
	S2OPM	0.40	0.18	0.13	0.08	0.06	$\boxed{0.03}$	$\boxed{0.03}$
	S4OPM	0.43	$\boxed{0.12}$	$\boxed{0.09}$	$\boxed{0.06}$	$\boxed{0.04}$	0.04	$\boxed{0.03}$
Random	KOPM	6.90	4.91	6.66	6.06	4.94	6.72	6.70
	BMOPM-3	8.08	4.01	4.69	4.17	4.08	4.10	4.07
	BMOPM-4	11.48	3.89	3.47	1.77	1.39	1.47	1.18
	BMOPM-5	22.83	5.97	4.70	2.67	1.88	1.22	0.79
	FSO-OPM	5.36	$\boxed{1.93}$	$\boxed{1.64}$	1.68	1.65	1.69	1.68
	S2OPM	$\boxed{3.90}$	2.47	1.89	1.17	1.26	0.79	$\boxed{0.35}$
	S4OPM	$\boxed{3.90}$	2.01	1.76	$\boxed{1.15}$	$\boxed{0.85}$	$\boxed{0.57}$	$\boxed{0.35}$

and KMP as the filtration method of our solution. The results of our practical experiments prove that the solutions based on SBNDM2 and SBNDM4 are faster than the earlier BMOPM algorithm. Moreover, the solution based on FSO is still faster for certain short pattern lengths. Research on filters and encodings for order-preserving matching is continuing. After submitting this paper, Chhabra et al. [13] improved filtering time with the SIMD technology. Chhabra et al. [14] applied the same encoding to approximate order-preserving matching. Cantone et al. [15] developed a more sophisticated encoding with a faster filter.

References

[1] G. Navarro, M. Raffinot, Flexible Pattern Matching in Strings – Practical On-line Search Algorithms for Texts and Biological Sequences, Cambridge University Press, 2002, http://www.dcc.uchile.cl/~gnavarro/FPMbook/.

[2] M. Crochemore, C.S. Iliopoulos, T. Kociumaka, M. Kubica, A. Langiu, S.P. Pissis, J. Radoszewski, W. Rytter, T. Walen, Order-preserving incomplete suffix trees and order-preserving indexes, in: 20th International Symposium on String Processing and Information Retrieval, SPIRE 2013, Jerusalem, Israel, October 7–9, 2013, 2013, pp. 84–95.

[3] J. Kim, P. Eades, R. Fleischer, S. Hong, C.S. Iliopoulos, K. Park, S.J. Puglisi, T. Tokuyama, Order-preserving matching, Theor. Comput. Sci. 525 (2014) 68–79, http://dx.doi.org/10.1016/j.tcs.2013.10.006.

[4] M. Kubica, T. Kulczynski, J. Radoszewski, W. Rytter, T. Walen, A linear time algorithm for consecutive permutation pattern matching, Inf. Process. Lett. 113 (12) (2013) 430–433, http://dx.doi.org/10.1016/j.ipl.2013.03.015.

[5] S. Cho, J.C. Na, K. Park, J.S. Sim, Fast order-preserving pattern matching, in: Proceedings of the 7th International Conference on Combinatorial Optimization and Applications, COCOA 2013, Chengdu, China, December 12–14, 2013, 2013, pp. 84–95.

[6] S. Cho, J.C. Na, K. Park, J.S. Sim, A fast algorithm for order-preserving pattern matching, Inf. Process. Lett. 115 (2) (2015) 397–402, http://dx.doi.org/10.1016/j.ipl.2014.10.018.

[7] D. Belazzougui, A. Pierrot, M. Raffinot, S. Vialette, Single and multiple consecutive permutation motif search, in: Proceedings of the 24th International Symposium on Algorithms and Computation, ISAAC 2013, Hong Kong, China, December 16–18, 2013, 2013, pp. 66–77.

[8] D.E. Knuth, J.H.M. Jr., V.R. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (2) (1977) 323–350, http://dx.doi.org/10.1137/0206024.

[9] R.S. Boyer, A fast string searching algorithm, Commun. ACM 20 (10) (1977) 762–772.

[10] B. Durian, J. Holub, H. Peltola, J. Tarhio, Improving practical exact string matching, Inf. Process. Lett. 110 (4) (2010) 148–152, http://dx.doi.org/10.1016/j.ipl.2009.11.010.

[11] K. Fredriksson, S. Grabowski, Practical and optimal string matching, in: Proceedings of the 12th International Conference on String Processing and Information Retrieval, SPIRE 2005, Buenos Aires, Argentina, November 2–4, 2005, 2005, pp. 376–387.

[12] A. Hume, D. Sunday, Fast string searching, Softw. Pract. Exp. 21 (11) (1991) 1221–1248.

[13] T. Chhabra, M.O. Külekci, J. Tarhio, Alternative algorithms for order-preserving matching, in: Proceedings of the Prague Stringology Conference 2015, Prague, Czech Republic, August 24–26, 2015, 2015, pp. 36–46, http://www.stringology.org/event/2015/p05.html.

[14] T. Chhabra, E. Giaquinta, J. Tarhio, Filtration algorithms for approximate order-preserving matching, in: Proceedings of the 22nd International Symposium on String Processing and Information Retrieval, SPIRE 2015, London, UK, September 1–4, 2015, 2015, pp. 177–187.

[15] D. Cantone, S. Faro, M.O. Külekci, An efficient skip-search approach to the order-preserving pattern matching problem, in: Proceedings of the Prague Stringology Conference, Prague, Czech Republic, August 24–26, 2015, 2015, pp. 22–35, http://www.stringology.org/event/2015/p04.html.