Theoretical Computer Science 557 (2014) 34–49

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

The swap matching problem revisited

Pritom Ahmed [a,*], Costas S. Iliopoulos [b], A.S.M. Sohidull Islam [a], M. Sohel Rahman [a]

[a] AlEDA Group, Department of Computer Science, BUET, Dhaka, Bangladesh
[b] Algorithm Design Group, Department of Computer Science, King's College London, University of London, United Kingdom

ARTICLE INFO

Article history:
Received 19 September 2013
Received in revised form 10 July 2014
Accepted 16 August 2014
Available online 23 August 2014
Communicated by G. Ausiello

Keywords:
Algorithms
Strings
Swap Matching
Graphs

ABSTRACT

In this paper, we revisit the much studied problem of Pattern Matching with Swaps (Swap Matching problem, for short). We first present a graph-theoretic model, which opens a new and so far unexplored avenue to solve the problem. Then, using the model, we devise two efficient algorithms to solve the swap matching problem. The resulting algorithms are adaptations of the classic shift-and algorithm. For patterns having length similar to the word-size of the target machine, both the algorithms run in linear time considering a fixed alphabet.

1. Introduction

The classical pattern matching problem is to find all the occurrences of a given pattern P of length m in a text T of length n, both being sequences of characters drawn from a finite character set Σ. This problem is interesting as a fundamental computer science problem and is a basic need of many practical applications such as text retrieval, music information retrieval, computational biology, data mining, network security, among many others. In this paper, we revisit the Pattern Matching with Swaps problem (the Swap Matching problem, for short), which is a well-studied variant of the classic pattern matching problem. In this problem, the pattern P is said to *swap match* the text T at a given location i, if adjacent pattern characters can be swapped, if necessary, so as to make the pattern identical to the substring of the text ending (or equivalently, starting) at location i. All the swaps are constrained to be disjoint, i.e., each character is involved in at most one swap.

Amir et al. [1] obtained the first non-trivial results for this problem. They showed how to solve the problem in time $O(nm^{1/3}\log m\log\sigma)$, where $\sigma = \min(|\Sigma|, m)$. Amir et al. [3] also studied certain special cases for which $O(n\log^2 m)$ time solution can be obtained. However, these cases are rather restrictive. Later, Amir et al. [2] solved the Swap Matching problem in time $O(n\log m\log\sigma)$. We remark that all the above solutions to swap matching depend on Fast Fourier Transformation (FFT) technique. Recently, Cantone and Faro [9] presented a dynamic programming approach to solve the swap matching problem which runs in linear time for finite character set Σ, when patterns are compatible with the word size of the target machine. Notably the work of [9] avoids the use of FFT technique. Cantone, Faro and Campanelli presented another

* Corresponding author.
E-mail addresses: pritom.11@gmail.com (P. Ahmed), csi@dcs.kcl.ac.uk (C.S. Iliopoulos), sohansayed@gmail.com (A.S.M.S. Islam), msrahman@cse.buet.ac.bd (M.S. Rahman).
URLs: http://www.dcs.kcl.ac.uk/staff/csi (C.S. Iliopoulos), http://teacher.buet.ac.bd/msrahman (M.S. Rahman).

http://dx.doi.org/10.1016/j.tcs.2014.08.009
0304-3975/

approach in [10] to solve the Swap matching problem. Though the algorithm of [10] runs in $O(nm)$ time for patterns compatible with the word size of the target machine, in practice it achieves quite good result. In fact as it turns out, the algorithm of [10] outperforms the algorithm of [9] most of the time. Finally, during the submission and review process of this manuscript we have come to know about a very recent work of Faro [12], where a reactive automata based approach has been presented to solve the swap matching problem. Notably, approximate swapped matching [4] and swap matching in weighted sequences [7] have also been studied in the literature.

1.1. Our contribution

The contribution of this paper is as follows. We first present a graph-theoretic approach to model the swap matching problem. Using this model, we devise two efficient algorithms to solve the swap matching problem. The resulting algorithms are adaptation of the classic shift-and algorithm [6] and runs in linear time if the pattern size is similar to the size of word in the target machine, assuming a fixed alphabet size. Notably, some preliminary results of this paper were presented in [8]. In [8], an algorithm running in $O(m/wn \log m)$ time was presented, where w is the machine word size. For short patterns, i.e., pattern size similar to machine word size, this runtime becomes $O(n \log m)$. Hence the result in this paper clearly improves the results of [8] and matches the result of [9]. Finally, we present experimental results to compare the non-FFT algorithms of [9,10] and our work.

1.2. RoadMap

The rest of the paper is organized as follows. In Section 2, we present some preliminary notations and definitions. In Section 3, we present our model to solve the swap matching problem. In Section 4, we present two different algorithms to solve the swap matching problem. Section 5, presents the experimental results. Finally, we briefly conclude in Section 6.

2. Preliminaries

A *string* is a sequence of zero or more symbols from an alphabet, Σ. A string X of length n is denoted by $X[1..n] = X_1X_2 \dots X_n$, where $X_i \in \Sigma$ for $1 \leq i \leq n$. The *length* of X is denoted by $|X| = n$. A string w is called a *factor* of X if $X = uwv$ for $u, v \in \Sigma^*$; in this case, the string w occurs at position $|u| + 1$ in X. The factor w is denoted by $X[|u| + 1..|u| + |w|]$. A *k-factor* is a factor of length k. A *prefix (or suffix)* of X is a factor $X[x..y]$ such that $x = 1$ $(y = n)$, $1 \leq y \leq n$ $(1 \leq x \leq n)$. We define the i-th prefix to be the prefix ending at position i, i.e., $X[1..i]$, $1 \leq i \leq n$. On the other hand, the i-th suffix is the suffix starting at position i, i.e., $X[i..n]$, $1 \leq i \leq n$.

Definition 1. (See [9,10].) A *swap permutation* for X is a permutation $\pi : \{1, \dots, n\} \to \{1, \dots, n\}$ such that:

1. if $\pi(i) = j$ then $\pi(j) = i$ (characters are swapped).
2. for all $i, \pi(i) \in \{i - 1, i, i + 1\}$ (only adjacent characters are swapped).
3. if $\pi(i) \neq i$ then $X_{\pi(i)} \neq X_i$ (identical characters are not swapped).

For a given string X and a swap permutation π for X, we use $\pi(X)$ to denote the *swapped version* of X, where $\pi(X) = X_{\pi(1)}X_{\pi(2)} \dots X_{\pi(n)}$.

Definition 2. (See [9,10].) Given a text $T = T_1T_2 \dots T_n$ and a pattern $P = P_1P_2 \dots P_m$, P is said to swap match at location i of T if there exists a swapped version P' of P that matches T at location[1] i, i.e. $P'_j = T_{i-m+j}$ for $j \in [1..m]$.

Problem "SM" (Pattern Matching with Swaps). *Given a text $T = T_1T_2 \dots T_n$ and a pattern $P = P_1P_2 \dots P_m$, we want to find each location $i \in [1..n]$ such that P swap matches with T at location i.*

Definition 3. A string X is said to be degenerate, if it is built over the potential $2^{|\Sigma|} - 1$ non-empty sets of letters belonging to Σ.

Example 1. Suppose we are considering DNA alphabet, i.e., $\Sigma = \Sigma_{DNA} = \{A, C, T, G\}$. Then we have 15 non-empty sets of letters belonging to Σ_{DNA}. In what follows, the set containing A and T will be denoted by $[AT]$ and the singleton $[C]$ will be simply denoted by C for ease of reading. The set containing all the letters, namely $[ACTG]$, is known as the do not care character in the literature.

Definition 4. Given two degenerate strings X and Y each of length n, we say $X[i]$ *matches* $Y[j]$, $1 \leq i, j \leq n$ if, and only if, $X[i] \cap Y[j] \neq \emptyset$.

[1] Note that, we are using the end position of the match to identify it.