Theoretical Computer Science 548 (2014) 1–13

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Motif matching using gapped patterns ☆

Emanuele Giaquinta [a,1,*], Kimmo Fredriksson [c], Szymon Grabowski [b], Alexandru I. Tomescu [a,d], Esko Ukkonen [a]

[a] Department of Computer Science, University of Helsinki, Finland
[b] Institute of Applied Computer Science, Lodz University of Technology, Al. Politechniki 11, 90-924 Łódź, Poland
[c] School of Computing, University of Eastern Finland, P.O. Box 1627, FI-70211 Kuopio, Finland
[d] Helsinki Institute for Information Technology HIIT, Finland

ARTICLE INFO

Article history:
Received 11 November 2013
Received in revised form 13 June 2014
Accepted 25 June 2014
Available online 30 June 2014
Communicated by M. Crochemore

Keywords:
Combinatorial problems
String algorithms
Multiple pattern matching
Pattern matching with gaps
Word-level parallelism

ABSTRACT

We present new algorithms for the problem of *multiple string matching* of gapped patterns, where a gapped pattern is a sequence of strings such that there is a gap of fixed length between each two consecutive strings. The problem has applications in the discovery of transcription factor binding sites in DNA sequences when using generalized versions of the Position Weight Matrix model to describe transcription factor specificities. In these models a motif can be matched as a set of gapped patterns with unit-length keywords. The existing algorithms for matching a set of gapped patterns are worst-case efficient but not practical, or *vice versa*, in this particular case. The novel algorithms that we present are based on dynamic programming and bit-parallelism, and lie in a middle-ground among the existing algorithms. In fact, their time complexity is close to the best existing bound and, yet, they are also practical. We also provide experimental results which show that the presented algorithms are fast in practice, and preferable if all the strings in the patterns have unit-length.

1. Introduction

We consider the problem of matching a set $\mathcal{P}$ of gapped patterns against a given text of length n, where a gapped pattern is a sequence of strings, over a finite alphabet Σ of size σ, such that there is a gap of fixed length between each two consecutive strings. We are interested in computing the list of matching patterns for each position in the text. This problem is a specific instance of the *Variable Length Gaps problem* [3] (VLG problem) for multiple patterns and has applications in the discovery of transcription factor (TF) binding sites in DNA sequences when using generalized versions of the Position Weight Matrix (PWM) model to represent TF binding specificities. The paper [8] describes how a motif represented as a generalized PWM can be matched as a set of gapped patterns with unit-length keywords, and presents algorithms for the restricted case of patterns with two unit-length keywords.

In the VLG problem a pattern is a concatenation of strings and of variable-length gaps. An efficient approach to solve the problem for a single pattern is based on the simulation of nondeterministic finite automata [12,6]. A method to solve the case of one or more patterns is to translate the patterns into a regular expression [13,4]. The best time bound for

☆ A preliminary version of this paper appeared in the Proceedings of the 24th International Workshop on Combinatorial Algorithms.

* Corresponding author.
E-mail address: emanuele.giaquinta@cs.helsinki.fi (E. Giaquinta).

[1] Supported by the Academy of Finland, grant 118653 (ALGODAN).

http://dx.doi.org/10.1016/j.tcs.2014.06.032
0304-3975/

Table 1
Comparison of different algorithms for the multiple string matching with gapped patterns problem. k-len($\mathcal{P}$) and len($\mathcal{P}$) are the total number of keywords and symbols in the patterns, respectively. $g_{\text{size}}(\mathcal{P})$ is the size of the variation range of the gap lengths. $\alpha \leq n\text{k-len}(\mathcal{P})$ and $\alpha' \leq n\text{k-len}(\mathcal{P})$ are the total number of occurrences in the text of keywords and pattern prefixes, respectively. $K \leq \text{k-len}(\mathcal{P})$ is the maximum number of suffixes of a keyword that are also keywords.

Time	Reference
$O(n \log \sigma + \alpha)$	Bille et al. [3]
$O(n(\log \sigma + K) + \alpha')$	Haapasalo et al. [9]
$O(n(\log \sigma + \log w \lceil \text{k-len}(\mathcal{P})/w \rceil) + occ)$	Bille and Thorup [4]
$O(n(\log \sigma + \log^2 g_{\text{size}}(\mathcal{P}) \lceil \text{k-len}(\mathcal{P})/w \rceil) + occ)$	This paper
$O(\lceil n/w \rceil \text{len}(\mathcal{P}) + n + occ)$	This paper

a regular expression is $O(n(k\frac{\log w}{w} + \log \sigma))$ [4], where k is the number of the strings and gaps in the pattern and w is the machine word size in bits. Observe that in the case of unit-length keywords $k = \Theta(\text{len}(\mathcal{P}))$, where len($\mathcal{P}$) is the total number of alphabet symbols in the patterns. There are also algorithms efficient in terms of the total number α of occurrences of the strings in the patterns (keywords) within the text [10,15,3].[2] The best bound obtained for a single pattern is $O(n \log \sigma + \alpha)$ [3]. This method can also be extended to multiple patterns. However, if all the keywords have unit length this result is not ideal, because in this case α is $\Omega(n\frac{\text{len}(\mathcal{P})}{\sigma})$ on average if we assume that the symbols in the patterns are sampled from Σ according to a uniform distribution. A similar approach for multiple patterns [9] leads to $O(n(\log \sigma + K) + \alpha')$ time, where K is the maximum number of suffixes of a keyword that are also keywords and α' is the number of text occurrences of pattern prefixes that end with a keyword. This result may be preferable in general when $\alpha' < \alpha$. In the case of unit-length keywords, however, a lower bound similar to the one on α holds also for α', as the prefixes of unit length have on average $\Omega(n\frac{|\mathcal{P}|}{\sigma})$ occurrences in the text. Recently, a variant of this algorithm based on word-level parallelism was presented in [18]. This algorithm works in time $O(n(\log \sigma + (\log |\mathcal{P}| + \frac{k}{w})\alpha_m))$, where k in this case is the maximum number of keywords in a single pattern and $\alpha_m \geq \lceil \alpha/n \rceil$ is the maximum number of occurrences of keywords at a single text position. When α or α' is large, the bound of [4] may be preferable. The drawback of this algorithm is that, to our knowledge, the method used to implement fixed-length gaps, based on maintaining multiple bit queues using word-level parallelism, is not practical.

Note that the above bounds do not include preprocessing time and the $\log \sigma$ term in them is due to the simulation of the Aho–Corasick automaton for the strings in the patterns.

In this paper we present two new algorithms, based on dynamic programming and bit-parallelism, for the problem of matching a set of gapped patterns. The first algorithm has $O(n(\log \sigma + g_{\text{w-span}} \lceil \text{k-len}(\mathcal{P})/w \rceil) + occ)$-time complexity, where k-len($\mathcal{P}$) is the total number of keywords in the patterns and $1 \leq g_{\text{w-span}} \leq w$ is the maximum number of distinct gap lengths that span a single word in our encoding. This algorithm is preferable only when $g_{\text{w-span}} \ll w$. We then show how to improve the time bound to $O(n(\log \sigma + \log^2 g_{\text{size}}(\mathcal{P}) \lceil \text{k-len}(\mathcal{P})/w \rceil) + occ)$, where $g_{\text{size}}(\mathcal{P})$ is the size of the variation range of the gap lengths. Note that in the case of unit-length keywords we have k-len($\mathcal{P}$) = len($\mathcal{P}$). This bound is a moderate improvement over the more general bound for regular expressions by Bille and Thorup [4] for $\log g_{\text{size}}(\mathcal{P}) = o(\sqrt{\log w})$. This algorithm can also be extended to support character classes with no overhead. The second algorithm is based on a different parallelization of the dynamic programming matrix and has $O(\lceil n/w \rceil \text{len}(\mathcal{P}) + n + occ)$-time complexity. The advantage of this bound is that it does not depend on the number of distinct gap lengths. However, it is not strictly on-line, because it processes the text w characters at a time and it also depends on len($\mathcal{P}$) rather than on k-len($\mathcal{P}$). Moreover, it cannot support character classes without overhead. The proposed algorithms obtain a bound similar to the one of [4], in the restricted case of fixed-length gaps, while being also practical. For this reason, they provide an effective alternative when α or α' is large. They are also fast in practice, as shown by experimental evaluation. A comparison of our algorithms with the existing ones is summarized in Table 1.

The rest of the paper is organized as follows. In Section 2 we recall some preliminary notions and elementary facts. In Section 3 we discuss the motivation for our work. In Section 4 we describe the method based on dynamic programming for matching a set of gapped patterns and then in Sections 5 and 6 we present the new algorithms based on it. Finally, in Section 7 we present experimental results to evaluate the performance of our algorithms.

2. Basic notions and definitions

Let Σ denote an integer alphabet of size σ and Σ^* the Kleene star of Σ, i.e., the set of all possible sequences over Σ. $|S|$ is the length of string S, $S[i], i \geq 0$, denotes its $(i+1)$-st character, and $S[i \dots j]$ denotes its substring between the $(i+1)$-st and the $(j+1)$-st characters (inclusive). For any two strings S and S', we say that S' is a suffix of S (in symbols, $S' \sqsupseteq S$) if $S' = S[i \dots |S| - 1]$, for some $0 \leq i < |S|$.

[2] Note that the number of occurrences of a keyword that occurs in r patterns and in l positions in the text is equal to $r \times l$.