Journal of Discrete Algorithms 11 (2012) 25–36

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

On the bit-parallel simulation of the nondeterministic Aho–Corasick and suffix automata for a set of patterns

Domenico Cantone, Simone Faro *, Emanuele Giaquinta

Università di Catania, Dipartimento di Matematica e Informatica, Viale Andrea Doria 6, I-95125 Catania, Italy

ARTICLE INFO

Article history:
Available online 2 March 2011

Keywords:
Multiple pattern matching
Text processing
Automata
Bit-parallelism

ABSTRACT

In this paper we present a method to simulate, using the bit-parallelism technique, the nondeterministic Aho–Corasick automaton and the nondeterministic suffix automaton induced by the trie and by the Directed Acyclic Word Graph for a set of patterns, respectively. When the prefix redundancy is nonnegligible, this method yields—if compared to the original bit-parallel encoding with no prefix factorization—a representation that requires smaller bit-vectors and, correspondingly, less words. In particular, if we restrict to single-word bit-vectors, more patterns can be packed into a word.

We also present two simple algorithms, based on such a technique, for searching a set $\mathcal{P}$ of patterns in a text T of length n over an alphabet Σ of size σ. Our algorithms, named Log-And and Backward-Log-And, require $\mathcal{O}((m+\sigma)\lceil m/w\rceil)$-space, and work in $\mathcal{O}(n\lceil m/w\rceil)$ and $\mathcal{O}(n\lceil m/w\rceil l_{min})$ worst-case searching time, respectively, where w is the number of bits in a computer word, m is the number of states of the automaton, and l_{min} is the length of the shortest pattern in $\mathcal{P}$.

1. Introduction

Given a set $\mathcal{P}$ of r patterns and a text T of length n, all strings over a common finite alphabet Σ of size σ, the *multiple pattern matching problem* is to determine all the occurrences in T of the patterns in $\mathcal{P}$. In this paper we focus on automata based solutions of such problem and, in particular, on the efficient simulation of the nondeterministic finite automaton (NFA) for the language $\bigcup_{P\in\mathcal{P}} \Sigma^* P$ induced by the *trie* data structure for $\mathcal{P}$ and the nondeterministic automaton for the language $\bigcup_{P\in\mathcal{P}} \mathit{Suff}(P)$ of all the suffixes of the strings in $\mathcal{P}$ induced by the Directed Acyclic Word Graph (DAWG) data structure for $\mathcal{P}$. We shall refer to such two automata as Aho–Corasick NFA and suffix NFA, respectively.

The first linear solution for the multiple pattern matching problem based on finite automata is due to Aho and Corasick in [1]. The Aho–Corasick algorithm uses a deterministic incomplete finite automaton based on the *trie* for the input patterns and on the *failure function*, a generalization of the border function of the Knuth–Morris–Pratt algorithm [9]. The optimal average complexity of the problem is $\mathcal{O}(n\log_\sigma(rl_{min})/l_{min})$ [11], where l_{min} is the length of the shortest pattern in the set $\mathcal{P}$; this bound has been achieved by algorithms based on the suffix automaton induced by the DAWG data structure, namely the Backward-DAWG-Matching (BDM) and Set-Backward-DAWG-Matching (SBDM) algorithms [7,10]. Later, Baeza-Yates and Gonnet introduced in [3] the bit-parallelism technique to simulate efficiently simple nondeterministic finite automata (NFAs, for short) for the single pattern case. Their Shift-And algorithm is one of the most efficient and elegant simulations of this kind of NFAs. Navarro and Raffinot used this technique to simulate the BDM algorithm; specifically, their algorithm,

* Corresponding author.
E-mail addresses: cantone@dmi.unict.it (D. Cantone), faro@dmi.unict.it (S. Faro), giaquinta@dmi.unict.it (E. Giaquinta).

1570-8667/$ – see front matter
doi:10.1016/j.jda.2011.02.001

named Backward-Nondeterministic-DAWG-Matching (BNDM), is based on a bit-parallel encoding of the nondeterministic suffix automaton induced by the DAWG of a single pattern [12].

In the bit-parallel simulation, the automaton current configuration is represented as an array of ℓ bits, where ℓ is the number of states in the automaton. Bits corresponding to active states are set to `1`, whereas bits corresponding to inactive states are set to `0`. Such representation allows one to take advantage of the intrinsic parallelism of the bit operations inside a computer word, thus cutting down the number of operations up to a factor equal to the number of bits in a computer word.

However, to simulate efficiently an NFA with the bit-parallelism technique, the states of the automaton must be mapped into the positions of a bit-vector by a suitable topological ordering of the NFA.[1] There are known bit-parallel simulations for the trie of a single pattern and for the maximal trie of a set of patterns. In the case of a single pattern, the construction of the topological ordering is quite simple, since it is unique [3]. Appropriate topological orderings can be obtained also for the maximal trie of a set of patterns, by interleaving the tries of the single patterns in either a parallel fashion, under the restriction that all the patterns have the same length [14], or in a sequential fashion [12]. The Shift-And and BNDM algorithms can be easily extended to the multiple patterns case by deriving the corresponding automaton from the maximal trie of the set of patterns. The resulting algorithms have an $\mathcal{O}(\sigma\lceil size(\mathcal{P})/w\rceil)$-space complexity and work in $\mathcal{O}(n\lceil size(\mathcal{P})/w\rceil)$ and $\mathcal{O}(n\lceil size(\mathcal{P})/w\rceil l_{min})$ worst-case searching time complexity, respectively, where $size(\mathcal{P}) = \sum_{P\in\mathcal{P}} |P|$ is the sum of the lengths of the strings in $\mathcal{P}$ and w is the size of a computer word.

In both cases, the bit-parallel simulation is based on the following property of the topological ordering π associated to the trie which allows to encode the transitions using a shift of k bits and a bitwise `and`: for each edge (p, q), the distance $\pi(q) - \pi(p)$ is equal to a constant k. For an in-depth survey on the topic, the reader is referred to [6].

The problem which arises when trying to bit-parallel simulate the Aho–Corasick NFA and the suffix NFA is that, in general, there might be no topological ordering π such that, for each edge (p, q), the distance $\pi(q) - \pi(p)$ is fixed. Cantone and Faro presented in [6] a bit-parallel simulation of the Aho–Corasick NFA that encodes variable length shifts using the carry property of addition and based on a particular topological ordering; however, such topological orderings do not always exist. Their algorithm has an $\mathcal{O}(\sigma\lceil m/w\rceil)$-space and $\mathcal{O}(n\lceil m/w\rceil)$-searching time complexity, where m is the number of nodes in the trie.

As explained above, the current technique used to extend string matching algorithms based on bit-parallelism to the multiple string matching problem consists, on a conceptual basis, in sequentially concatenating the automata for each pattern. The drawback of this method is that it is not possible to exploit the prefix redundancy in the patterns, a property which can be significant in the case of small alphabets. The trie and the DAWG data structures make it possible to factor common prefixes in the patterns. However, because of the lack of regularity in such structures, it is not possible to devise a simulation of the corresponding automata using the original bit-parallel encoding. In this paper we present a new more general approach to the efficient bit-parallel simulation of the Aho–Corasick NFAs and suffix NFAs. When the prefix redundancy is nonnegligible, this method yields—if compared to the original encoding with no prefix factorization—a representation that requires smaller bit-vectors and, correspondingly, less words. Therefore, if we restrict to single-word bit-vectors, it results that more patterns can be packed into a word. Our construction is based on a result for the Glushkov automaton [13], which however requires exponential space in the number of states in the NFA to encode the transition function. We show that, by exploiting the relation between active states of the NFA and its associated failure function, it is possible to represent the transition function in polynomial space using a similar encoding.

The rest of the paper is organized as follows. In Section 2 we recall some preliminary notions and elementary facts. In Section 3 we present a general technique to simulate NFAs for a set of patterns. Then in Sections 4 and 5 we devise a bit-parallel encoding of the Aho–Corasick NFA and of the suffix NFA, respectively, and describe also two bit-parallel algorithms for the *multiple pattern matching problem* based on such encodings. Finally, we briefly draw our conclusions in Section 6.

2. Basic notions and definitions

A string P of length $|P| = m$ over a given finite alphabet Σ is any sequence of m characters of Σ. For $m = 0$, we obtain the empty string ε. Σ^* is the collection of all finite strings over Σ. We denote by $P[i]$ the $(i+1)$-st character of P, for $0 \leqslant i < m$. Likewise, the substring of P contained between the $(i+1)$-st and the $(j+1)$-st characters of P is denoted by $P[i\,..\,j]$, for $0 \leqslant i \leqslant j < m$. We also put $P_i =_{\text{Def}} P[0\,..\,i]$, for $0 \leqslant i < m$, and make the convention that P_{-1} denotes the empty string ε. It is common to identify a string of length 1 with the character occurring in it. For any two strings P and P', we write $P.P'$ to denote the concatenation of P' to P, and $P' \sqsupset P$ to express that P' is a *proper* suffix of P, i.e., $P = P''.P'$ for some nonempty string P''. The notation $P' \sqsupseteq P$ will be used with the obvious meaning. Analogously, $P' \sqsubseteq P$ ($P' \sqsubset P$) expresses that P' is a (proper) prefix of P, i.e., $P = P'.P''$ for some (nonempty) string P''. We say that P' is a factor of P if $P = P''.P'.P'''$, for some strings P'', $P''' \in \Sigma^*$, and we denote by $Fact(P)$ the set of the factors of P. Likewise, we denote by $Suff(P)$ the set of the suffixes of P. We write P^r to denote the reverse of the string P, i.e., $P^r = P[m-1]P[m-2]\ldots P[0]$. Given a finite set of patterns $\mathcal{P}$, we put $\mathcal{P}^r =_{\text{Def}} \{P^r \mid P \in \mathcal{P}\}$ and $\mathcal{P}_l =_{\text{Def}} \{P[0\,..\,l-1] \mid P \in \mathcal{P}\}$.

[1] We recall that a *topological ordering* of an NFA is any total ordering $<$ of the set of its states such that $p < q$, for each edge (p, q) of the NFA.