International Journal of Foundations of Computer Science
© World Scientific Publishing Company

A SPACE EFFICIENT BIT-PARALLEL ALGORITHM FOR THE MULTIPLE STRING MATCHING PROBLEM[*]

DOMENICO CANTONE [†]
and
SIMONE FARO [‡]
Dipartimento di Matematica e Informatica, Università di Catania
Viale A.Doria n.6, Catania, Italy, 95125

ABSTRACT

Finite (nondeterministic) automata are very useful building blocks in the field of string matching. This is particularly true in the case of multiple pattern matching, where the use of factor-based automata can reduce substantially the number of computational steps when the patterns have large common factors.
Direct simulation of nondeterministic automata can be performed very efficiently using the bit-parallelism technique, though this is not necessarily true for factor-based automata.
In this paper we present an algorithm for the multiple string matching problem, based on the bit-parallel simulation of nondeterministic factor-based automata which satisfy a particular ordering condition. We also show how to enforce such condition by suitably modifying a minimal initial automaton, through equivalence preserving transformations. The resulting automaton turns out to be smaller than the corresponding maximal automata used by existing bit-parallel algorithms, as they do not take any advantage of common factors in patterns.

Keywords: multiple string matching, bit-parallelism, text searching

1. Introduction

Given a set $\mathcal{P} = \{P_1, \ldots, P_r\}$ of patterns and a text T, all strings over a finite alphabet Σ of size σ, the *multiple pattern matching problem* is to determine all the positions where any of the patterns in $\mathcal{P}$ occurs in T. This problem arises naturally in many applications, and several algorithms exist to solve it. For example, the UNIX `fgrep` and `egrep` programs support multi-pattern matching through the `-f` option. The worst case complexity of multiple pattern matching is $\Omega(n)$ and it has been achieved by the Aho-Corasick algorithm [1]. From a practical point of view, the best average complexity bound for multi-pattern matching algorithms is $\mathcal{O}(n \log_\sigma(rm)/m)$, where m is the minimum length of any pattern in $\mathcal{P}$. Such bound has been reached, for instance, by the Dawg-Match algorithm [6] and by

[*] This paper is based on results preliminarily presented in [4]

[†] cantone@dmi.unict.it

[‡] faro@dmi.unict.it

the Multi-BDM algorithm [8]. We cite also that the Boyer-Moore strategy has been extended to multi-pattern matching, such as in the Commenz-Walter [5] and in the Wu-Manber [14] algorithms.

In this paper we are mainly interested on automata based solutions of the pattern matching problem, and on their implementation by bit-parallelism. In general, (nondeterministic) automata allow to handle classes of characters and multiple patterns in a simple, efficient, and flexible way, leading to algorithms which are asymptotically optimal both in space and time [10, 1].

The bit-parallelism technique [**?**] consists in exploiting the intrinsic parallelism of the bit operations inside a computer word. It can be profitably used for the simulation of finite automata even in their nondeterministic form.

The paper is organized as follows. After introducing in Section 2 the basic notations used in the paper, in Section 3 we survey the most significant algorithms for the single and multiple pattern matching problem which make use of factor-based deterministic finite automata. Then, in Section 4 we describe the bit-parallelism technique and discuss some of the single and multi-pattern matching algorithms based on it. Existing algorithms in the multi-pattern case do not take any particular advantage of the presence of large common factors in the patterns. Thus, in Section 5 we present a new solution for the multi-pattern matching problem which efficiently mixes the advantages in space obtained from factor-based automata with the simplicity and flexibility of bit-parallelism. Finally, we draw our conclusions and propose some hints for future work in Section 6.

2. Basic Definitions and Terminology

We introduce here the basic notations and terminology used in the paper. A string P of length m is represented as an array $P[0\,..\,m-1]$. Thus $P[i]$ will denote the $(i+1)$-st character of P, for $i = 0, \ldots, m-1$. We denote the length of P by $|P|$. In addition, if $\mathcal{P} = \{P_1, P_2, \ldots, P_r\}$ is a set of strings, we denote by $size(\mathcal{P})$ the sum of the lengths of its strings, namely $size(\mathcal{P}) = \sum_{i=1}^{r} |P_i|$.

For any two strings P and P', we write $P' \sqsupset P$ to indicate that P' is a *proper* suffix of P, $P' \sqsubset P$ to indicate that P' is a *proper* prefix of P, and $P.P'$ to denote the concatenation of P' to P. Given a set of patterns $\mathcal{P} = \{P_1, P_2, \ldots, P_r\}$ in an alphabet Σ, the *trie* $\mathcal{T}$ associated with $\mathcal{P}$ is a rooted directed tree, whose edges are labeled by single characters of Σ, such that (i) distinct edges out of a same node are labeled by distinct characters, (ii) all paths in $\mathcal{T}$ from the root are labeled by prefixes of the strings in $\mathcal{P}$, (iii) for each string P in $\mathcal{P}$ there exists a path in $\mathcal{T}$ from the root which is labeled by P.

If we do not insist on property (i) above, we obtain a more relaxed form of trie, which we call *nondeterministic trie*. Since all tries considered in this paper are nondeterministic, for the sake of simplicity we will refer to them just as "tries."

For any node p in a trie $\mathcal{T}$, we denote by $lbl(p)$ the string which labels the path from the root of $\mathcal{T}$ to the node p and put $len(p) = |lbl(p)|$, i.e., $len(p)$ is the length of the path from the root of $\mathcal{T}$ to p. Additionally, for any edge (p, q) in $\mathcal{T}$, we denote the label of (p, q) by $lbl(p, q)$. We also denote by $children_{\mathcal{T}}(p)$ the set of the children of p in the trie $\mathcal{T}$.

Given a (nondeterministic) trie $\mathcal{T}$ relative to a set of patterns $\mathcal{P} = \{P_1, P_2, \dots, P_r\}$ over an alphabet Σ, we can naturally associate with $\mathcal{T}$ the following *canonical nondeterministic finite automaton (NFA)* $\widehat{\mathcal{T}} = (Q_{\mathcal{T}}, q_0, F_{\mathcal{T}}, \delta_{\mathcal{T}})$, where:

- $Q_{\mathcal{T}}$ is the set of nodes of $\mathcal{T}$ (*set of states*);

- $q_0 \in Q_{\mathcal{T}}$ is the root of $\mathcal{T}$ (*initial state*);

- $F_{\mathcal{T}} =_{Def} \{q \in Q_{\mathcal{T}} \mid lbl(q) \in \mathcal{P}\}$ (*set of final or terminal states*);

- $\delta_{\mathcal{T}} : Q_{\mathcal{T}} \times \Sigma \rightarrow \mathscr{P}(Q_{\mathcal{T}})$, with

  $$\delta_{\mathcal{T}}(q, c) =_{Def} \begin{cases} \{p \in Q_{\mathcal{T}} \mid lbl(q).c = lbl(p)\} & \text{if } q \neq q_0 \\ \{p \in Q_{\mathcal{T}} \mid lbl(q).c = lbl(p)\} \cup \{q_0\} & \text{if } q = q_0\,, \end{cases}$$

  for $q \in Q_{\mathcal{T}}$, $c \in \Sigma$, and where $\mathscr{P}(\cdot)$ is the powerset operator (*transition function*).

Thus the words *node* and *state* will often be used interchangeably. Likewise, we will often identify a trie $\mathcal{T}$ with its corresponding NFA $\widehat{\mathcal{T}}$.

3. Automata Based String Matching Algorithms

Automata play a very important role in the design of efficient pattern matching algorithms. For instance the well known KNUTH-MORRIS-PRATT algorithm [10] uses a deterministic automaton that searches a pattern in a text by performing its transitions on the text characters. The main result relative to the KNUTH-MORRIS-PRATT algorithm is that its automaton can be constructed in $\mathcal{O}(m)$-time and -space, whereas pattern search takes $\mathcal{O}(n)$-time, thus reaching the best bound for a pattern matching algorithm (as usual, m and n denote the length of the pattern and text, respectively). In the case of multiple pattern matching, the AHO-CORASICK algorithm [1] has been the first having a linear behavior. It is also based on the automata approach and can be viewed much as a generalization of the KNUTH-MORRIS-PRATT algorithm to the multi-pattern case. In particular, the AHO-CORASICK automaton is a trie $\mathcal{T}$ for the set of patterns $\mathcal{P}$, with a failure function $f : Q_{\mathcal{T}} \rightarrow Q_{\mathcal{T}}$ which is followed when no transition is possible on a text character. The function f is defined on each node $u \in Q_{\mathcal{T}}$ in such way that:

- $lbl(f(u)) \sqsupset lbl(u)$, and

- $len(f(u)) \geq len(p)$, for each $p \in Q_{\mathcal{T}}$ such that $lbl(p) \sqsupset lbl(u)$.

The AHO-CORASICK automaton can be constructed in linear time and space [8].

Automata based solutions have been also developed to design algorithms which have optimal sublinear performances on average. For instance, several algorithms have been developed to extend to the multiple pattern matching case the efficient

Boyer-Moore strategy [3]. Among them, we cite the Commenz-Walter algorithm [5] which extends the Horspool algorithm [9] through a suffix based approach. The Commenz-Walter algorithm starts by reading the text backwards from position j, initially set to $\ell = \min\{|P_k| \; : \; P_k \in \mathcal{P}\}$. Then characters are matched against the labels of the trie $\mathcal{T}$ for the set $\mathcal{P}^r$ of the reverse patterns. When a final state is reached, an occurrence is reported. If no matching is possible with the current character, then position j is shifted by the minimum nonnull depth in $\mathcal{T}$ of an edge labeled by the previous read character $T[j]$. If no edge in $\mathcal{T}$ is labeled by $T[j]$, then j is increased by ℓ.

Another type of automaton, called *suffix automaton* (or Dawg, for Directed Acyclic Word Graph), has been introduced for the single pattern matching problem in [6, 7, 8, 13] and later generalized to the multi-pattern case. A suffix automaton for a set $\mathcal{P}$ of patterns is a trie for the set $\mathcal{P}^r$ that recognizes all the suffixes of the patterns in $\mathcal{P}$.

For instance, the Reverse-Factor algorithm [7], for the single pattern matching problem, computes shifts which match prefixes of the pattern, rather than suffixes, using the smallest suffix automaton of the reverse of the pattern. Despite its quadratic worst-case time complexity, the Reverse-Factor algorithm is very fast in practice. Other optimal sublinear algorithms on average, like Backward-DAWG-Match (BDM) and Turbo-BDM [7, 8], have been obtained with this approach, and have been also extended to multiple pattern matching in [6, 8, 13].

4. String Matching and Bit-Parallelism

In general, it is much easier to construct a nondeterministic automaton rather than a deterministic one, due to its simplicity and regularity. Thus, it would be desirable to be able to simulate efficiently the parallel computation of an NFA. This can be done using the bit-parallelism technique [**?**]. Such technique consists in exploiting the intrinsic parallelism of the bit operations inside a computer word. In favorable cases it allows to cut down the overall number of operations by a factor of ω, where ω is the number of bits in a computer word. For this reason, although string matching algorithms based on bit-parallelism are usually simple and have very low memory requirements, they generally work well only with patterns of moderate length.

In the context of string matching, such technique has been especially used to speed-up algorithms based on automata. The simulation is carried out by representing an automaton as an array of L bits, where $L + 1$ is the number of states of the automaton. The initial state does not need to be represented, because it is always active. Bits corresponding to active states are set to `1`, whereas bits corresponding to inactive states are set to `0`.

To simulate efficiently an NFA using the bit-parallelism technique, its states must be mapped into the positions of a bit-vector by a suitable bijection.

In the case of a trie (or better, the NFA associated with it), we succeded to simulate it efficiently provided that the bijection is a *weakly safe topological ordering*, in a sense which will be explained later.

For the time being, we just recall that a topological ordering of a trie $\mathcal{T}$ is a bijection $\pi : Q_{\mathcal{T}} \rightarrow \{0, \dots, |Q_{\mathcal{T}}| - 1\}$, which agrees with the edges of $\mathcal{T}$, namely such that $\pi(p) < \pi(q)$ whenever (p, q) is in $\mathcal{T}$. It is convenient to associate with π its inverse $\phi : \{0, \dots, |Q_{\mathcal{T}}| - 1\} \rightarrow Q_{\mathcal{T}}$, which is assumed to map each position of a bit-vector to the corresponding state of $\mathcal{T}$.

For later purposes, given a topological ordering π of $\mathcal{T}$, it is also convenient to associate to each edge (p, q) in $\mathcal{T}$ its π-*interval* $[\pi(p), \pi(q)[$, also denoted by $Int_\pi(p, q)$. The length $\pi(q) - \pi(p)$ of the π-interval $[\pi(p), \pi(q)[$ will be denoted by $|Int_\pi(p, q)|$. Notice that since π is a topological ordering of $\mathcal{T}$, then $|Int_\pi(p, q)| \geq 1$, for each edge (p, q) in $\mathcal{T}$.

4.1. Searching for a Single Pattern

In the case of single pattern matching, the trie $\mathcal{T}$ associated with a given pattern P of length m is linear. Thus, the corresponding NFA $\widehat{\mathcal{T}}$ is obtained from $\mathcal{T}$ just by adding a self-loop on its initial state, labeled by all symbols of the alphabet Σ, to allow the scan to begin at any position in the text. Plainly, in this case we have only one possible topological ordering of $\mathcal{T}$, whose inverse ϕ_1 is recursively defined by:

$$\phi_1(i) = \begin{cases} \delta_{\mathcal{T}}(q_0, P[0]) & \text{if } i = 0 \\ \delta_{\mathcal{T}}(\phi_1(i-1), P[i-1]) & \text{if } 1 \leq i \leq m-1 \,. \end{cases}$$

Thus, for $i = 0, 1, \dots, m-1$, state $\phi_1(i)$ is simulated by the i-th bit of a bit-vector. The initial state does not need to be represented, because it is always active. Figure 1(A) shows the nondeterministic finite automaton which recognizes the pattern $P = \mathtt{aababb}$.

The first result, concerning single pattern matching algorithms using the bit-parallelism technique, is due to Baeza-Yates and Gonnet [**?**]. Their algorithm, named Shift-And, maintains, for each symbol c of the alphabet Σ, a bit mask $B[c]$ whose i-th bit is set to $\mathtt{1}$, provided that $P[i] = c$, where P is the pattern. The current configuration of the automaton is maintained in a bit mask D, which is initialized to $\mathtt{0}^L$, since initially all (noninitial) states are inactive. Moreover a *final-state* bit-mask $M = \mathtt{1}\mathtt{0}^{L-1}$ maintains the position of the final state of the automaton, whereas an *initial-state* bit-mask $I = \mathtt{0}^{L-1}\mathtt{1}$ maintains the position of the node adjacent to the initial state.

While scanning a text T from left to right, the Shift-And algorithm simulates automaton transitions by the following basic shift-and operation, for each position j:

$$D = ((D \ll 1) \;\mid\; I) \;\&\; B[T[j]] \,.$$

If the final state is active, i.e. $D \;\&\; M \neq \mathtt{0}^L$, a matching is reported at position j. It turns out that the Shift-And algorithm has an $\mathcal{O}(\lceil mn/\omega \rceil)$ worst-case running time and requires $\mathcal{O}(\lceil L/\omega \rceil)$-space.

Other algorithms based on bit-parallelism use a Boyer-Moore strategy, to simulate a right to left scan of the pattern. For instance, the BNDM algorithm [**?**] is the bit-parallel implementation of the Reverse-Factor algorithm. It is based

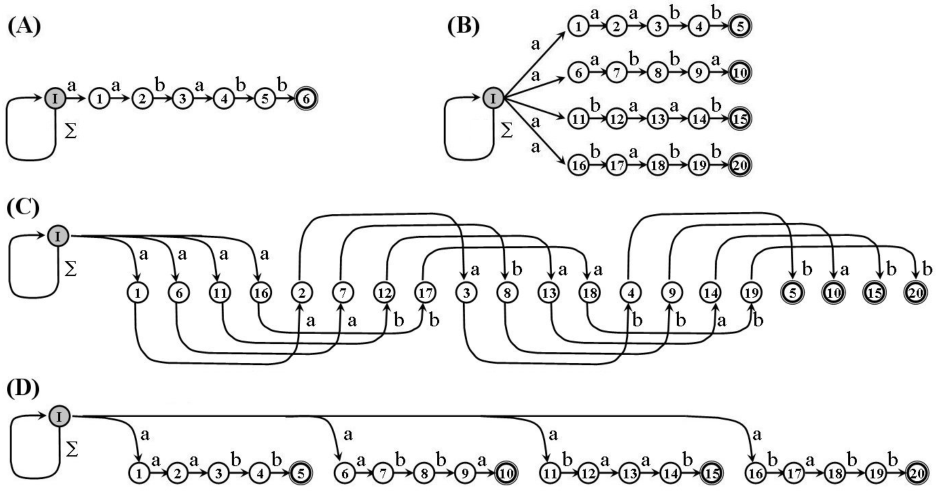

Figure 1: **(A)** An NFA which recognizes the pattern P = `aababb`. **(B)** An NFA obtained from the maximal trie $\mathcal{T}$ of the set of patterns $\mathcal{P}$ = $\{$`aaabb`, `aabba`, `abaab`, `ababb`$\}$. **(C)** The parallel topological ordering of $\mathcal{T}$. **(D)** The sequential topological ordering of $\mathcal{T}$.

on the nondeterministic version of the smallest suffix automaton of the reverse of the pattern P. Unlike the Shift-And algorithm, characters of text and pattern are compared from right to left until the entire pattern is read or no transition by the automaton is possible. Then the pattern is shifted by ℓ positions to the right, where ℓ is the length of the last matched prefix. Despite its quadratic worst-case running time, the BNDM algorithm performs well in practical cases.

4.2. Searching for Multiple Patterns

Existing algorithms that search for a set $\mathcal{P} = \{P_1, \ldots, P_r\}$ of patterns, using bit-parallelism, simulate the behavior of the *maximal trie* of $\mathcal{P}$. This is the trie $\mathcal{T}$ of $\mathcal{P}$ obtained from the linear tries $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_r$ for the patterns $P_1, P_2, \ldots, P_r$, respectively, by merging the roots of $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_r$ in a single node. Plainly, the number of states of $\mathcal{T}$ is given by $|\mathcal{T}| = \sum_{i=1}^{r} |\mathcal{T}_i| - r + 1 = \mathit{size}(\mathcal{P}) + 1$, so that it can be represented by a bit-vector of $L = \mathit{size}(\mathcal{P})$ bits. For instance, Figure 1(B) shows the maximal trie relative to the set of patterns $\mathcal{P} = \{$`aaabb`, `aabba`, `abaab`, `ababb`$\}$. Two different topological orderings have been used in literature to simulate a maximal trie of a set of pattern $\mathcal{P}$. A first arrangement, π^{par}, has been proposed in [14], under the restriction that all patterns in the set $\mathcal{P}$ have the same length. Given a set $\mathcal{P} = \{P_1, P_2, \ldots, P_r\}$ of r distinct patterns of the same length m, the topological ordering π^{par} of the trie $\mathcal{T}$ relative to $\mathcal{P}$ is obtained just by interleaving the NFAs of the patterns of $\mathcal{P}$ in a parallel fashion. More precisely, the inverse ϕ^{par} of π^{par}

can be recursively defined by

$$\phi^{\text{par}}(kr+j) = \begin{cases} \delta_{\mathcal{T}_{j+1}}(q_0, P_{j+1}[0]) & \text{if } k = 0 \\ \delta_{\mathcal{T}_{j+1}}(\phi^{\text{par}}((k-1)r+j), P_{j+1}[k]) & \text{if } 1 \leq k \leq m-1\,, \end{cases}$$

with $0 \leq j \leq r-1$. Figure 1(C) shows the parallel topological ordering of the NFA of Figure 1(B). Using such arrangement, it is possible to search for patterns in $\mathcal{P}$ just as in the case of a single pattern. The only difference with the single pattern case is that the shift is not by a single bit, but by r bits (since consecutive nodes are r bits apart in the parallel arrangement). Moreover, we need to use the new initial-state and final-state masks $I = \mathtt{0}^{r(m-1)}\mathtt{1}^r$ and $M = \mathtt{1}^r\mathtt{0}^{r(m-1)}$, respectively. Figure 2 (left side) shows the code of an implementation of the Shift-And algorithm, based on a parallel ordering of the maximal trie for a set $\mathcal{P}$ of patterns having the same length.

An alternative arrangement, π^{seq}, has been proposed in [12]. It consists in concatenating in a sequential fashion the different branches of the maximal trie of a set $\mathcal{P}$ of patterns. More precisely, given a set $\mathcal{P} = \{P_1, P_2, \ldots, P_r\}$ of patterns (not necessarily of the same length), the inverse ϕ^{seq} of the ordering π^{seq} relative to the maximal trie of $\mathcal{P}$ is recursively defined by

$$\phi^{\text{seq}}\left(\sum_{j=1}^{h-1}|P_j| + i\right) = \begin{cases} \delta_{\mathcal{T}_h}(q_0, P_h[0]) & \text{if } i = 0 \\ \delta_{\mathcal{T}_h}(\phi^{\text{seq}}(\sum_{j=1}^{h-1}|P_j| + i - 1), P_h[i-1]) & \text{if } 1 \leq i \leq |P_h| - 1\,, \end{cases}$$

with $1 \leq h \leq r$.

Figure 1(D) shows the sequential topological ordering of the NFA in Figure 1(B). In this case, we return to single bit shifts, whereas the initial-state and final-state masks are

$$\begin{aligned} I &= (\mathtt{0}^{|P_1|-1}\mathtt{1})(\mathtt{0}^{|P_2|-1}\mathtt{1})\cdots(\mathtt{0}^{|P_r|-1}\mathtt{1}) \\ M &= (\mathtt{1}\mathtt{0}^{|P_1|-1})(\mathtt{1}\mathtt{0}^{|P_2|-1})\cdots(\mathtt{1}\mathtt{0}^{|P_r|-1})\,. \end{aligned}$$

On some processors, shifts by a single position is faster than shift by $r > 1$ positions. In such cases the arrangement π^{seq} yields faster algorithms. Moreover, as already observed, such arrangement allows to deal with sets of patterns of different lengths.

Figure 2 (right side) shows the code of an implementation of the Shift-And algorithm, based on a sequential ordering of the maximal trie of a set $\mathcal{P}$. Though not necessary, for the sake of simplicity we have assumed that the patterns in $\mathcal{P}$ have the same length m.

5. A new space efficient approach

In this section we propose a new approach to bit-parallel multiple pattern matching. Unlike existing solutions, presented in the previous section, which make use of the maximal trie of a set $\mathcal{P}$ of patterns, here we propose a solution which simulates, using bit-parallelism, a factor-based automaton thus reducing the number of states and, accordingly, the number of bits needed for its representation.

Below we introduce the important notion of *(weakly) safe topological ordering* of a trie. Then, in Section 5.1 we present an efficient variant of the Shift-And

```
Sequential-Shift-And ($T$, $\mathcal{P} = \{P_1, \ldots, P_r\}$)
$n = \text{length}(T)$
$m = \text{length}(P_1)$
$L = m^r$
for $c \in \Sigma$ do $B[c] = 0^L$
$l = 0$
for $k = 1$ to $r$ do
    for $i = 0$ to $m - 1$ do
        $B[P_k[i]] = (B[P_k[i]] \mid (0^{L-1}1 \ll l + i))$
    $l = l + m$
$I = (0^{m-1}1)^r$
$M = (10^{m-1})^r$
$D = 0^L$
for $j = 0$ to $n - 1$ do
    if $D \;\&\; M \neq 0^L$ then print($j$)
    $D = ((D \ll 1) \mid I) \;\&\; B[T[j]]$
```

```
Parallel-Shift-And ($T$, $\mathcal{P} = \{P_1, \ldots, P_r\}$)
$n = \text{length}(T)$
$m = \text{length}(P_1)$
$L = m^r$
for $c \in \Sigma$ do $B[c] = 0^L$
$l = 0$
for $i = 0$ to $m - 1$ do
    for $k = 1$ to $r$ do
        $B[P_k[i]] = (B[P_k[i]] \mid (0^{L-1}1 \ll l + k))$
    $l = l + r$
$I = 0^{r(m-1)}1^r$
$M = 1^r0^{r(m-1)}$
$D = 0^L$
for $j = 0$ to $n - 1$ do
    if $D \;\&\; M \neq 0^L$ then print($j$)
    $D = ((D \ll r) \mid I) \;\&\; B[T[j]]$
```

Figure 2: The Parallel-Shift-And algorithm which uses a parallel ordering of the maximal trie $\mathcal{T}$ of the set $\mathcal{P}$, and the Sequential-Shift-And algorithm which uses a sequential ordering of the nodes of $\mathcal{T}$.

algorithm, based on a trie for $\mathcal{P}$ admitting a weakly safe topological ordering. Our proposed algorithm, called Multiple-Trie-Shift-And, searches a text T for any pattern in a set $\mathcal{P}$ in $\mathcal{O}(n\lceil L/\omega\rceil)$-time, where $n = |T|$, $L = \mathit{size}(\mathcal{P})$, and ω is the size of a computer word. Subsequently, in Section 5.2 we present an algorithm, named Construct-Safe-Topological-Ordering, which given a (minimal) trie $\mathcal{T}$ for a set $\mathcal{P}$ of patterns constructs another trie $\mathcal{T}'$ for $\mathcal{P}$ admitting a weakly safe topological ordering (in general, the size of $\mathcal{T}'$ may be larger than the size of $\mathcal{T}$). The Construct-Safe-Topological-Ordering algorithm is based on a DFS approach and runs in $\mathcal{O}(L)$-time and -space, under suitable hypotheses.

Let π_u be a topological ordering of the subtrie $\mathcal{T}_u$ of $\mathcal{T}$ rooted in u. Also, let (p, q) be an edge of $\mathcal{T}_u$.

We say that (p, q) is a *long-bit edge (relative to the ordering π_u)* if the length of the π_u-interval of (p,q) is greater than 1, i.e., in symbols, $|Int_{\pi_u}(p,q)| > 1$.[a] Otherwise, i.e. if $|Int_{\pi_u}(p,q)| = 1$, we say that (p,q) is a *1-bit edge (relative to the ordering π_u)*. Additionally, if (p,q) is a long-bit edge of $\mathcal{T}_u$, we say that the label $lbl(p,q)$ of the edge (p,q) is an *engaged symbol* for the node u. It is convenient to define the following function and set

$$\begin{array}{rcl} \mathcal{L}_{\pi_u}(c) & =_{Def} & \{(p,q) \in \mathcal{T}_u \mid lbl(p,q) = c \text{ and } |Int_{\pi_u}(p,q)| > 1\} \\ \mathcal{A}_{\pi_u} & =_{Def} & \{c \in \Sigma \mid \mathcal{L}_{\pi_u}(c) \neq \emptyset\}\,, \end{array}$$

for c in the alphabet Σ, u in $\mathcal{T}$, and π_u a topological ordering of $\mathcal{T}_u$. In other words, $\mathcal{L}_{\pi_u}(c)$ is the collection of long-bit edges of $\mathcal{T}_u$ labeled by c, whereas $\mathcal{A}_{\pi_u}$ is the collection of all engaged symbols for u.

Finally, a topological ordering π of a trie $\mathcal{T}$ is said to be

- *safe*, if for each $c \in \Sigma$, the intervals in $\{Int_\pi(p,q) \mid (p,q) \in \mathcal{L}_\pi(c)\}$ are pairwise disjoint, i.e., if the π-intervals of any two distinct long-bit egdes labeled by a same character are disjoint;

- *weakly safe*, if for each $c \in \Sigma$, the intervals in $\{Int_\pi(p,q) \mid (p,q) \in \mathcal{L}_\pi(c)$ and $p \neq root(\mathcal{T})\}$ are pairwise disjoint, i.e., if the π-intervals of any two distinct long-bit egdes labeled by a same character and not originating from the root of $\mathcal{T}$ are disjoint.

Figures 3(B)-(C) show two different topological orderings of the trie in Figure 3(A). In particular, concerning the ordering π' relative to Figure 3(B), we have $\mathcal{L}_{\pi'}(a) = \{(3,6),(8,9)\}$ and $\mathcal{L}_{\pi'}(b) = \{(1,2)\}$; hence π' is a weakly safe topological ordering since $\pi'(9) = 6 < 10 = \pi'(3)$. On the other hand, the ordering π'' relative to Figure 3(C) is not weakly safe, since in this case we have $\mathcal{L}_{\pi''}(a) = \{(1,8),(3,6),(8,9)\}$, $\mathcal{L}_{\pi'}(b) = \emptyset$, and $\pi''(1) = 1 < \pi''(3) = 3 < \pi''(6) = 6 < \pi''(8) = 8$, i.e. $Int_{\pi''}(3,6) \subset Int_{\pi''}(1,8)$.

5.1. *The* Multiple-Trie-Shift-And *Algorithm*

Given a text T and a set $\mathcal{P} = \{P_1, P_2, \ldots, P_r\}$ of patterns, the Multiple-Trie-Shift-And algorithm which we present below searches for any pattern of $\mathcal{P}$ in the text T in $\mathcal{O}(n\lceil L/\omega\rceil)$-time, where $n = |T|$, $L = size(\mathcal{P})$, and ω is the size of a computer word. Besides the text T, it takes as input a pair $\mathcal{T}$ and π, where $\mathcal{T}$ is a trie for $\mathcal{P}$ and π is a weakly safe topological ordering of $\mathcal{T}$ (as will be shown in the next section, such $\mathcal{T}$ and π can be efficiently constructed starting from a minimal trie for $\mathcal{P}$). The Multiple-Trie-Shift-And algorithm simulates its input automaton $\mathcal{T}$ using bit-parallelism. Since $|Q_{\mathcal{T}}| \leq L + 1$, in general our algorithm deals with smaller automata than the algorithms reviewed in Section 4.2.

Let $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_h$ be the subtries of $\mathcal{T}$ rooted in the children of $root(\mathcal{T})$ and let $\{f_1, f_2, \ldots, f_k\}$ be the set of final states of $\mathcal{T}$. The algorithm initializes two bit-masks of length $L = |\mathcal{T}| - 1$, respectively the initial-state mask I and the final-state

[a] The notion of π_u-interval and the notation $|Int_{\pi_u}(p,q)|$ have been introduced just before Section 4.1.

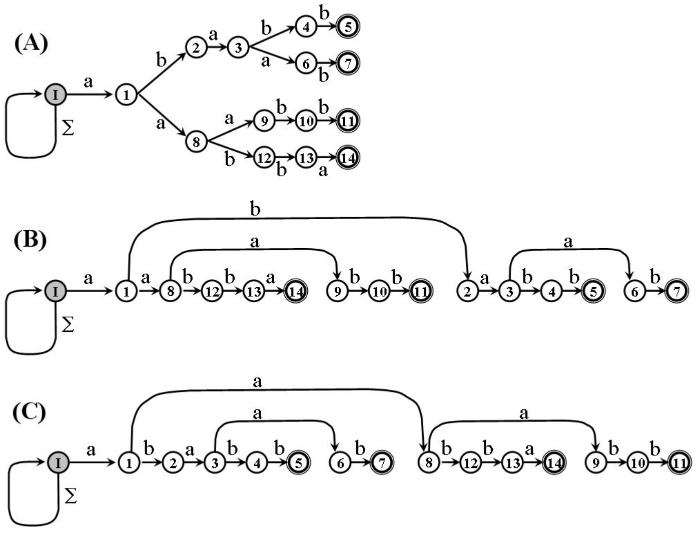

Figure 3: **(A)** The minimal trie of the set of patterns $\mathcal{P} = \{\mathtt{ababb}, \mathtt{abaab}, \mathtt{aaabb}, \mathtt{aabba}\}$. **(B)** A weakly safe topological ordering of the trie in (A). **(C)** A topological ordering of the trie in (A) which is not weakly safe.

mask M, as follows

$$\begin{aligned} I &= (\mathtt{0}^{|Q_{\mathcal{T}_h}|-1}\mathtt{1})\cdots(\mathtt{0}^{|Q_{\mathcal{T}_2}|-1}\mathtt{1})(\mathtt{0}^{|Q_{\mathcal{T}_1}|-1}\mathtt{1}) \\ M &= (\mathtt{1}\mathtt{0}^{\pi(f_k)-\pi(f_{k-1})-1})\cdots(\mathtt{1}\mathtt{0}^{\pi(f_2)-\pi(f_1)-1})(\mathtt{1}\mathtt{0}^{\pi(f_1)-1})\,. \end{aligned}$$

Subsequently, for each symbol $c \in \Sigma$, the Multiple-Trie-Shift-And algorithm initializes as shown below three more bit-masks of length L, namely $B[c], \mathit{IS}[c]$ and $\mathit{GS}[c]$, which allow to perform the automaton transitions.

For each state $q \in Q_{\mathcal{T}}$ such that $\mathit{lbl}(q)[\mathit{len}(q)-1] = c$, we set the $\pi(q)$-th bit of $B[c]$ to $\mathtt{1}$.

Let $\mathcal{L}_\pi(c) = \{(p_1, q_1), (p_2, q_2), \ldots, (p_t, q_t)\}$ be the set of long-bit edges in π labeled by the symbol c, arranged in such a way that $\pi(p_1) < \pi(q_1) \leq \pi(p_2) < \pi(q_2) \leq \cdots \leq \pi(p_t) < \pi(q_t)$. The mask $\mathit{IS}[c]$ is the *initial-shift* bit-mask of c. It marks all nodes in π from which a long-bit edge labeled with symbol c originates. In other words, for each edge $(p, q) \in \mathcal{L}_\pi(c)$, the $\pi(p)$-th bit of $\mathit{IS}[c]$ is set to $\mathtt{1}$. More formally,

$$\mathit{IS}[c] = (\mathtt{0}^{L-p_t}\mathtt{1})(\mathtt{0}^{p_t-p_{t-1}-1}\mathtt{1})\cdots(\mathtt{0}^{p_2-p_1-1}\mathtt{1})(\mathtt{0}^{p_1-1})\,.$$

Finally, the mask $\mathit{GS}[c]$ is the *gap-shift* bit-mask of c. For each long-bit edge $(p, q) \in \mathcal{L}_\pi(c)$, the bits of $\mathit{GS}[c]$ from position $(\pi(p)+1)$ up to position $(\pi(q)-1)$ are set to $\mathtt{1}$. More formally,

$$\mathit{GS}[c] = (\mathtt{0}^{L-q_t+1}\mathtt{1}^{q_t-p_t-1})(\mathtt{0}^{p_t-q_{t-1}+1}\mathtt{1}^{q_{t-1}-p_{t-1}-1})\cdots(\mathtt{0}^{p_2-q_1+1}\mathtt{1}^{q_1-p_1-1})(\mathtt{0}^{p_1})\,.$$

```
MULTIPLE-TRIE-SHIFT-AND ($T$, $\mathcal{T}$, $\pi$)

  /* INITIALIZATION */
 1.   $n = \text{length}(T)$
 2.   $\phi = \pi^{-1}$
 3.   $L = |Q_{\mathcal{T}}| - 1$
 4.   $I = M = 0^L$
 5.   for each $c \in \Sigma$ do $B[c] = IS[c] = GS[c] = 0^L$
 6.   $root = \phi(0)$
 7.   for each $q \in children_{\mathcal{T}}(root)$ do
 8.       $c = lbl(root, q)$
 9.       $B[c] = (B[c] \mid (0^{L-1}1 \ll (\pi(q) - 1)))$
10.   for $i = 1$ to $L$ do
11.       $p = \phi(i)$
12.       if IS_FINAL($p$) then $M = (M \mid (0^{L-1}1 \ll (i - 1)))$
13.       if $p \in children_{\mathcal{T}}(root)$ then $I = (I \mid (0^{L-1}1 \ll (i - 1)))$
14.       for each $q \in children_{\mathcal{T}}(p)$ do
15.           $c = lbl(p, q)$
16.           if $\pi(q) > i + 1$ then
17.               $IS[c] = (IS[c] \mid (0^{L-1}1 \ll (\pi(q) - 1)))$
18.               $GS[c] = (GS[c] \mid (0^{L-\pi(q)+\pi(p)+1}1^{\pi(q)-\pi(p)-1} \ll \pi(p)))$
19.           else $B[c] = (B[c] \mid (0^{L-1}1 \ll (\pi(q) - 1)))$

  /* SEARCHING PHASE */
20.   $D = 0^L$
21.   for $j = 0$ to $n - 1$ do
22.       if $D$ & $M \neq 0^L$ then print($j$)
23.       $D' = (D \ll 1)$ & $B[T[j]]$
24.       $D'' = ((((D$ & $IS[T[j]]) \ll 1) + GS[T[j]])$ & $\sim GS[T[j]])$
25.       $D = (D' \mid D'') \mid (I$ & $B[T[j]])$
```

Figure 4: The MULTIPLE-TRIE-SHIFT-AND algorithm for the multiple string matching problem.

During the searching phase (lines 20-25), a bit-mask D maintains the active state of the automaton. For each position j of the text T, the algorithm performs three main steps

1-bit transitions (line 22):

This is made in a simple way by shifting the mask D by one position to the left. Then all transitions labeled with symbols different from $T[j]$ are deleted by performing an AND operation with the bit-mask $B[T[j]]$. More formally, the operation that simulates 1-bit transitions is

$$(D \ll 1) \,\&\, B[T[j]]\,.$$

Long-bit transitions (line 23):

First, the operation $(D \,\&\, IS[T[j]])$ isolates all active states from which long-bit edges originate. Then the resulting mask is shifted by one position to the left and its value is added to the value of the bit-mask $GS[T[j]]$. This has the effect that, if $(p, q) \in \mathcal{L}_\pi(T[j])$ and p is an active state in D, then the $\pi(q)$-th bit of D is set to 1 and all bits from position $\pi(p)$ up to position $\pi(q)-1$ are set to 0. However, if $(p,q) \in \mathcal{L}_\pi(T[j])$ and p is not an active state in D, then all bits from position $\pi(p)+1$ up to position $\pi(q)-1$ maintain their value 1. These undesirable bits are deleted by performing an AND operation with the bit-mask $\sim GS[T[j]]$. More formally, long-bit transitions are simulated by the operation

$$((((D \ \&\ IS[T[j]]) \ \ll\ 1) \ +\ GS[T[j]]) \ \&\ \sim GS[T[j]]) \,.$$

Transitions from the initial state (line 24):

The transitions starting from the initial state are performed by computing an OR operation with the mask I. As in the 1-bit transition case, all transitions labeled with symbols different from $T[j]$ are deleted by performing an AND operation with the bit-mask $B[T[j]]$. Formally, transitions from the initial state are simulated by the following operation

$$(D \mid I) \ \&\ B[T[j]] \,.$$

The Multiple-Trie-Shift-And algorithm, shown in Figure 4, runs in $\mathcal{O}(n)$ time if $L \leq \omega$, where ω is the length of a computer word. However if $L > \omega$ the algorithm has a $\mathcal{O}(n\lceil L/\omega\rceil)$ worst-case time complexity.

In the following section we describe an algorithm that, given a minimal trie $\mathcal{T}$ for a set $\mathcal{P} = \{P_1, P_2, \ldots, P_r\}$ of patterns, it constructs another trie $\mathcal{T}'$, equivalent to $\mathcal{T}$, together with a weakly safe topological ordering π for $\mathcal{T}'$.

5.2. Constructing a Trie with a Weakly Safe Topological Ordering

Before entering into the details of the algorithm, we need to introduce some further useful concepts.

For each node $q \in Q_\mathcal{T}$ we define the set $B(q)$ of *binding symbols* of q as the collection of all characters which label some edge (p, p') originating from a predecessor p of q, but such that p' does not lie on the path from the $root(\mathcal{T})$ to q. In symbols

$$B(q) =_{Def} \{lbl(p,p') \ \mid p, p' \in \ Q_\mathcal{T} \,,\ lbl(p) \sqsubset lbl(q) \,,\ \text{and } lbl(p') \not\sqsubseteq lbl(q)\} \ .$$

In addition, for each node $q \in Q_\mathcal{T}$, we define the function $bind_q : \Sigma \rightarrow \{1, 2, \ldots, len(q)\}$ such that for each $c \in \Sigma$

$$bind_q(c) =_{Def} \begin{cases} 1 + \max \left\{ len(p) \ \middle|\ \begin{array}{l} p \in Q_\mathcal{T} \,,\ lbl(p) \sqsubset lbl(q) \,,\ \text{and} \\ c = lbl(p,p') \,,\ lbl(p') \not\sqsubseteq lbl(q) \\ \text{for some } p' \in Q_\mathcal{T} \end{array} \right\} & \text{if } c \in B(q) \\ 0 & \text{otherwise} \,. \end{cases}$$

Observe that, if $lbl(p) \sqsubset lbl(q)$, then $len(p) < len(q)$ and therefore $0 \leq bind_q(c) \leq len(q)$, for $c \in \Sigma$. For each $h \in \{1, \ldots, len(q)\}$ we define the set $B_h(q) \subseteq B(q)$ by putting

$$B_h(q) =_{Def} \{c \in B(q) \mid bind_q(c) = h\}.$$

Next, let again $q \in Q_{\mathcal{T}}$ and let $w = |children_{\mathcal{T}}(q)|$. Also, for each node $s \in children_{\mathcal{T}}(q)$, let π_s be a safe topological ordering for $\mathcal{T}_s$. We say that the set $children_{\mathcal{T}}(q)$ is *resolved w.r.t. the above orderings* π_s, if there exists an ordering $s_1, s_2, \ldots, s_w$ of the children of q in $\mathcal{T}$ such that the concatenation $\pi_{s_1}.\pi_{s_2}.\cdots.\pi_{s_w}$ yields a safe topological ordering π_q for $\mathcal{T}_q$. Observe that the edge (q, s_1) is a 1-bit edge for π_q, whereas the edges (q, s_i), for $i = 2, \ldots, w$, are long-bit edges for π_q.

Then, in order for π_q to be a safe topological ordering, we must have

$$lbl(q, s_i) \notin \bigcup_{j=1}^{i-1} \mathcal{A}_{\pi_q}(s_j)\,, \qquad \text{for each } i = 1, \ldots, w\,.$$

Additionally, observe that the set $B_{len(q)}(s) = \{ lbl(q, s') \mid s' \in children_{\mathcal{T}}(q) \setminus \{s\} \}$ defines the binding symbols on node s imposed by its predecessor q, for each $s \in children_{\mathcal{T}}(q)$. Thus, if $\mathcal{A}_{\pi_q}(s) \cap B_{len(q)}(s) \neq \emptyset$, for some $s \in children_{\mathcal{T}}(q)$, then the node s could violate some binding in $B_{len(q)}(s)$. To maintain such information during its execution, the algorithm in Figure 5 which we are about to describe performs a suitable coloring of the nodes. In particular, for each $q \in Q_{\mathcal{T}}$, we define the value $color(q)$ which can assume the following values:

WHITE: The color of a node q is WHITE provided that it has not been already visited by the algorithm. Thus, during the initialization phase, $color(q)$ is set to WHITE, for each $q \in Q_{\mathcal{T}}$.

GREEN/RED: Suppose that the visit of node q has been completed and that a safe topological ordering π_q of $\mathcal{T}_q$ has been constructed. Then $color(q)$ is set to GREEN, provided that π_q does not violate any binding imposed by its predecessor, i.e. provided that $\mathcal{A}_{\pi_q} \cap B_{len(q)-1}(q) = \emptyset$, otherwise is set to RED.

The algorithm which constructs a trie $\mathcal{T}'$ equivalent to a given input trie $\mathcal{T}$ and such that $\mathcal{T}'$ is endowed with a weakly safe topological ordering is shown in Figure 5. It performs a DFS visit of the trie $\mathcal{T}$, starting from $root(\mathcal{T})$. When the visit of a node $q \in Q_{\mathcal{T}} \setminus \{root(\mathcal{T})\}$ has been completed, a safe topological ordering π_q for the current subtrie rooted in q has been computed. The procedure for visiting a node $q \in Q_{\mathcal{T}}$ works in the following 6 main steps:

STEP 0 (Initialization)

During initialization, $\mathcal{A}(q)$ is set to $\emptyset$ and the ordering π_q is indirectly initialized by putting $\phi_q(0) = q$ (we recall that $\phi_q = \pi_q^{-1}$).

STEP 1. (Recursive calls)

After initialization, all $s \in children_{\mathcal{T}}(q)$ which have not been already visited are visited. Then, at the end of Step 1, it follows inductively that, for each $s \in children_{\mathcal{T}}(q)$, a safe topological ordering π_s has been defined and either $color(s) =$ GREEN or $color(s) =$ RED.

STEP 2. (Resolving nodes of set $Green(q)$)

Suppose $Green(q)$ and $Red(q)$ are the sets of, respectively, GREEN and RED nodes of $children_{\mathcal{T}}(q)$. By construction, no node in $Green(q)$ violates any

```
CONSTRUCT-SAFE-TOPOLOGICAL-ORDERING ($\mathcal{T}$)
    for each $q \in Q_{\mathcal{T}}$ do $color(q)$ =WHITE
    $\phi(0) = root(\mathcal{T})$
    $i = 1$
    for each $q \in children_{\mathcal{T}}(root(\mathcal{T})) \mid color(q) =$ WHITE do
        $\phi_q =$ VISIT$(q, \mathcal{T})$
        for $j = 0$ to $|Q_{\mathcal{T}_q}| - 1$ do $\phi(i + j) = \phi_q(j)$
        $i = i + |Q_{\mathcal{T}_q}|$
    return $(\phi, \mathcal{T})$

VISIT ($q$, $\mathcal{T}$)
  /* STEP 0 (Initialization) */
    $\phi_q(0) = q$, $i = 1$
    $\mathcal{A}(q) = \emptyset$
  /* STEP 1 (Recursive calls) */
    for each $s \in children_{\mathcal{T}}(q) \mid color(s) =$ WHITE do VISIT$(s, \mathcal{T})$
    $Green(q) = \{s \in children_{\mathcal{T}}(q) \mid color(s) =$ GREEN $\}$
    $Red(q) = \{s \in children_{\mathcal{T}}(q) \mid color(s) =$ RED $\}$
  /* STEP 2 (Resolving nodes of set Green(q)) */
    if $Green(q) \neq \emptyset$ then
        Let $s \in Green(q) \mid bind(lbl(q, s)) \geq bind(lbl(q, p)), \forall\ p \in Green(q)$
        for $j = 0$ to $|Q_{\mathcal{T}_s}| - 1$ do $\phi_q(i + j) = \phi_s(j)$
        $i = i + |Q_{\mathcal{T}_s}|$
        $\mathcal{A}(q) = \mathcal{A}(q) \cup \mathcal{A}(s)$
        $Green(q) = Green(q) - \{s\}$
    for each $s \in Green(q)$ do
        for $j = 0$ to $|Q_{\mathcal{T}_s}| - 1$ do $\phi_q(i + j) = \phi_s(j)$
        $i = i + |Q_{\mathcal{T}_s}|$
        $\mathcal{A}(q) = \mathcal{A}(q) \cup \mathcal{A}(s) \cup \{lbl(q, s)\}$
  /* STEP 3 (Resolving nodes of set Red(q)) */
    for each $s \in Red(q)$ do
        if $lbl(q, s) \notin \mathcal{A}(q)$ then
            $Red(q) = Red(q) - \{s\}$
            for $j = 0$ to $|Q_{\mathcal{T}_s}| - 1$ do $\phi_q(i + j) = \phi_s(j)$
            $i = i + |Q_{\mathcal{T}_s}|$
            if $\mathcal{A}(q) = \emptyset$ then $\mathcal{A}(q) = \mathcal{A}(q) \cup \mathcal{A}(s)$
            else $\mathcal{A}(q) = \mathcal{A}(q) \cup \mathcal{A}(s) \cup \{lbl(q, s)\}$
  /* STEP 4 (Pruning all remaining red nodes) */
    for each $s \in Red(q)$ do
        construct a new trie $\mathcal{T}'$ for $lbl(s)$
        for each $u \in Q_{\mathcal{T}'}$ do $color(u)$ =WHITE
        prune $\mathcal{T}_s$ from $\mathcal{T}$ and insert it at the end of $\mathcal{T}'$
        merge $root(\mathcal{T}')$ with $root(\mathcal{T})$
  /* STEP 5 (Setting color of node q) */
    if $\mathcal{A}(q) \cap B_{len(q)-1}(q) = \emptyset$ then $color(q) =$ GREEN
    else $color(q) =$ RED
    return $\phi_q$
```

Figure 5: The algorithm for computing a safe topological ordering of the trie $\mathcal{T}$.

binding imposed by q. Thus, it is more convenient to resolve first the nodes in $Green(q)$ and later the ones in $Red(q)$. If $Green(q) \neq \emptyset$, a node $s \in Green(q)$ such that $lbl(q, s)$ has the largest binding value $bind(lbl(q, s))$ is selected. In this way all engaged edges which could violate the binding closest to q are eliminated. Then the topological ordering π_s is concatenated to π_q, the edge (q, s) becomes a 1-bit edge in π_q, and $\mathcal{A}(q)$ is set to the value $\mathcal{A}(q) \cup \mathcal{A}(s)$.

For each remaining node $s \in \mathit{Green}(q)$, the ordering π_s is concatenated to π_q, so that all engaged nodes in π_s become engaged nodes in π_q. Observe that, after the first selection, the edge (q, s) is a *long-bit edge* of π_q, so that $\mathcal{A}(q)$ must be set to the value $\mathcal{A}(q) \cup \mathcal{A}(s) \cup \{\mathit{lbl}(q, s)\}$.

Step 3. (Resolving nodes of set $\mathit{Red}(q)$)

After that all green nodes have been resolved in Step 2, nodes in $\mathit{Red}(q)$ are also resolved. In particular, if $\mathit{Red}(q) \neq \emptyset$, then an attempt is made to select a node $s \in \mathit{Red}(q)$ such that the symbol $\mathit{lbl}(q, s)$ is not engaged in π_q, i.e. $\mathit{lbl}(q, s) \notin \mathcal{A}(q)$. If such a node s is found, the ordering π_s is concatenated to the ordering π_q and the set $\mathcal{A}(q)$ of engaged nodes in π_q is updated accordingly. Step 3 is reapeted until no further node $s \in \mathit{Red}(q)$ can be selected.

Observe that, if $\mathit{Green}(q) = \emptyset$ at the beginning of Step 2, then the first selected node in $\mathit{Red}(q)$ generates a 1-bit edge in π_q. This case is tested in lines 21-22.

Step 4. (Pruning all remaining red nodes)

If $\mathit{Red}(q) \neq \emptyset$ after Step 4, each subtree rooted at any node $s \in \mathit{Red}(q)$ is first detached from $\mathcal{T}$ and then re-attached to $\mathcal{T}$ through a freshly introduced linear path labeled by $\mathit{lbl}(s)$. Notice that Step 4 can cause the trie $\mathcal{T}$ to become nondeterminstic.

Step 5. (Setting color of node q)

Finally, if the engaged symbols of q violate some binding in $B(q)_{\mathit{len}(q)-1}$, i.e. $\mathcal{A}(q) \cap B(q)_{\mathit{len}(q)-1} \neq \emptyset$, $\mathit{color}(q)$ is set to red. Otherwise $\mathit{color}(q)$ is set to green.

At the end of the execution, the modified $\mathcal{T}$ and the function ϕ are returned. It turns out that ϕ^{-1} is a weakly safe topological ordering of $\mathcal{T}$.

Observe that there exist sets of patterns whose minimal tries admit no weakly safe topological ordering. The pruning of sub-tries in Step 4 is just intended to separate in $\mathcal{T}$ those patterns which cause troubles.

Let $\mathcal{P}$ be a set of patterns and let $\mathcal{T}$ be the minimal trie for $\mathcal{P}$. We evaluate the complexity of the algorithm in Figure 5 in terms of $L = \mathit{size}(\mathcal{P})$.

An efficient implementation of the algorithm Construct-Safe-Topological-Ordering maintains, for each node $q \in Q_{\mathcal{T}}$, the sets $B(q)_{\mathit{len}(q)-1}$ and $\mathcal{A}(q)$ in two bit-vectors. Thus, if we assume that $|\mathit{children}_{\mathcal{T}}(q)| \leq \omega$, for each $q \in Q_{\mathcal{T}}$, where ω is the length of a computer word, the operations of set union and set intersection can be performed in constant time and $\mathcal{O}(|Q_{\mathcal{T}}|)$ space. Such assumption is quite reasonable, since in practical cases the degree of a node is rarely greater than ω. This is especially true if the patterns belong to a natural language where consecutive symbols are not independent, rather they are strongly related in most cases. For instance the symbol "q" is almost always followed by the symbol "u", whereas in general the symbol "t" is followed only by the symbols "a,e,h,i,l,o,r,u,y".

Additionally, if we maintain the topological orderings π_q, for each node q, as linked-lists, the operations in lines 8, 13, and 19, which concatenate two different topological orderings, can be also performed in constant time.

The procedure Visit is called only once for each node $q \in Q_{\mathcal{T}}$. Since each node $s \in Q_{\mathcal{T}}$, with the exception of the root, will enter either set $Green(q)$ or set $Red(q)$, for only one node $q \in Q_{\mathcal{T}}$, we have that

$$\sum_{q \in Q_{\mathcal{T}}} (|Green(q)| + |Red(q)|) = |Q_{\mathcal{T}}| - 1\,.$$

Thus the overall complexity of Steps 2 and 3 is $\mathcal{O}(L)$, since $|Q_{\mathcal{T}}| = \mathcal{O}(L)$.

In Step 4, the pruning of a RED node s consists in following the path from the root of the trie to node s. Thus the overall work of Step 4 is bounded again by $\mathcal{O}(L)$.

Finally Step 0 and Step 5 are performed in constant time. Thus, it turns out that the algorithm Construct-Safe-Topological-Ordering has a $\mathcal{O}(L)$-time and -space complexity.

It must be remarked that in general the algorithm Construct-Safe-Topological-Ordering does not construct the *minimal* trie $\mathcal{T}'$, equivalent to a given trie $\mathcal{T}$, which is endowed with a weakly safe topological sorting. A natural variant which enforces minimality takes quadratic time.

On the other hand, some experimentations has shown that the heuristics embodied in Steps 2, 3, and 4 are quite effective in keeping the returned trie close to minimal.

6. Conclusion

In this paper we have presented a new algorithm for the multiple pattern matching problem, based on the bit-parallelism technique. In particular, our algorithm is based on the parallel simulation of a factor-based trie (not necessarily the optimal one) for the input set of patterns. In fact, our simulation requires that the factor-based trie admits a topological ordering which is weakly safe, in a sense amply explained before. The complexity of our algorithm is linear in the length of the text and in the size of the set of patterns.

We have also shown how to transform a given minimal trie into a trie which has a weakly safe topological ordering in linear time and space in the size of the set of patterns. The resulting trie is in general significantly smaller than the maximal tries used in the other multi-pattern matching algorithms based on bit-parallelism.

Further variations and improvements are still possible. For instance, we expect that our approach can be extended to obtain a space efficient version of the BNDM algorithm for the multiple pattern matching problem.

An interesting open problem is to find other suitable topological orderings on deterministic tries which guarantee that they can be easily simulated by bit-parallelism, without any need to modify their topology.

References

1. A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.
2. R. A. Baeza-Yates and G. H. Gonnet. A new approach to text searching. In N. J. Belkin and C. J. van Rijsbergen, editors, *Proceedings of the 12th International Conference on Research and Development in Information Retrieval*, pages 168–175, Cambridge, MA, 1989. ACM Press.
3. R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.
4. D. Cantone and S. Faro. A space efficient bit-parallel algorithm for the multiple string matching problem. In J. Holub and M. Šimánek, editors, *Proc. of the Prague Stringology Conference '05*, pages 109–124, Czech Technical University, Prague, Czech Republic, 2005.
5. B. Commentz-Walter. A string matching algorithm fast on the average. In H. A. Maurer, editor, *Proceedings of the 6th International Colloquium on Automata, Languages and Programming*, number 71 in Lecture Notes in Computer Science, pages 118–132, Graz, Austria, 1979. Springer-Verlag, Berlin.
6. M. Crochemore, A. Czumaj, L. Gąsieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter. Fast practical multi-pattern matching. Rapport 93-3, Institut Gaspard Monge, Université de Marne la Vallée, 1993.
7. M. Crochemore, A. Czumaj, L. Gąsieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter. Speeding up two string matching algorithms. *Algorithmica*, 12(4/5):247–267, 1994.
8. M. Crochemore and W. Rytter. *Text algorithms.* Oxford University Press, 1994.
9. R. N. Horspool. Practical fast searching in strings. *Softw. Pract. Exp.*, 10(6):501–506, 1980.
10. D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(1):323–350, 1977.
11. G. Navarro and M. Raffinot. A bit-parallel approach to suffix automata: Fast extended string matching. Technical Report TR/DC–98–1, Department of Computer Science, University of Chile, 1998.
12. G. Navarro and M. Raffinot. A bit-parallel approach to suffix automata: Fast extended string matching. In M. Farach-Colton, editor, *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching*, number 1448 in Lecture Notes in Computer Science, pages 14–33, Piscataway, NJ, 1998. Springer-Verlag, Berlin.
13. M. Raffinot. On the multi backward dawg matching algorithm (MultiBDM). In R. Baeza-Yates, editor, *Proceedings of the 4th South American Workshop on String Processing*, pages 149–165, Valparaiso, Chile, 1997. Carleton University Press.
14. S. Wu and U. Manber. Fast text searching with errors. Report TR-91-11, Department of Computer Science, University of Arizona, Tucson, AZ, 1991.