Knuth-Morris-Pratt algorithm : an analysis

Mireille Regnier

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

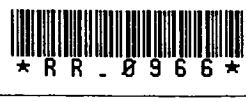
N° 966

Programme 1

KNUTH-MORRIS-PRATT ALGORITHM : AN ANALYSIS

Mireille RÉGNIER

Janvier 1989



KNUTH-MORRIS-PRATT ALGORITHM: AN ANALYSIS

ANALYSE EN MOYENNE DE L'ALGORITHME DE KNUTH-MORRIS-PRATT

Mireille REGNIER
INRIA-Rocquencourt
78 153 Le Chesnay-FRANCE

Abstract: *This paper deals with an analysis on the average of the Knuth-Morris-Pratt algorithm; the constant of linearity c is derived. In particular, when the cardinality q of the alphabet is large, it is proven that* $c \sim 1 + \frac{1}{q}$. *An algebraic scheme is used, based on combinatorics on words and generating functions.*

Résumé: *Nous réalisons une analyse en moyenne de l'algorithme de Knuth-Morris-Pratt; nous obtenons la constante de linéarité c. En particulier, nous prouvons que, pour de grandes cardinalités q de l'alphabet, on a:* $c \sim 1 + \frac{1}{q}$. *Nous utilisons une méthode algébrique, basée sur la combinatoire sur les mots et sur les fonctions génératrices.*

KNUTH-MORRIS-PRATT ALGORITHM: AN ANALYSIS

Mireille REGNIER
INRIA-Rocquencourt
78 153 Le Chesnay-FRANCE

Abstract

This paper deals with an analysis on the average of the Knuth-Morris-Pratt algorithm; the constant of linearity c is derived. In particular, when the cardinality q of the alphabet is large, it is proven that $c \sim 1 + \frac{1}{q}$. *An algebraic scheme is used, based on combinatorics on words and generating functions.*

I. INTRODUCTION:

We study here the Knuth-Morris-Pratt algorithm [KMP77] that searches a given pattern p in a text t. This algorithm avoids going backward in a text, which may be a big advantage for some types of memory. We are interested here in evaluating the number of comparisons performed to find all occurrences of a pattern of size $|p|$ in a text of size n. A preprocessing of the pattern p allows to build an automaton, that, after each mismatch, keeps the largest substring of the string made of the last characters read that still matches the pattern. Hence, a probabilistic analysis using Markov chains seems natural, and this is the method adopted in [BA85,BA88]. Unfortunately, as discussed below in Section 2., these methods of evaluation appear to be approximate and not powerful enough. We propose here an algebraic scheme via generating functions and combinatorics on words, that provides an exact expression of the number of comparisons, as a function of the cardinality q of the alphabet A. This allows a comparison with the naive algorithm, that is equivalent to Knuth-Morris-Pratt when q is big enough.

In Section II., we present a state of the art. We describe two variants of the algorithm and the probabilistic methods of evaluation previously used, and we discuss the results. In Section III., we recall some basic properties of generating functions as well as some classical methods to derive them for well-suited combinatorial structures. In Section IV., we derive the generating function of the number of comparisons performed, for one variant of the algorithm.

II. STATE OF THE ART:

In II.1. we present the original Knuth-Morris-Pratt algorithm as described in [KMP77] and the improvement suggested in [SE83]. In II.2. we discuss the analytical models (Markov chains) proposed by other authors [BA85,BA88].

II.1. The Knuth-Morris-Pratt Algorithm:

Let us first adopt some notations and definitions.

Definition 1: *The* **side** *of a string, or a word, w, on a q-ary alphabet is a non-empty subword which is a (strict) right factor and a (strict) left factor of w.*

Example (a): 01201201has two different sides: 01 and 01201.

In the following, p is a pattern that is searched in a text t. We note $p[i]$ (resp. $t[i]$) the i-th character of the pattern p (resp. the text t). The algorithm uses a text pointer TP and a pattern pointer PP that determine the next comparison to be performed: if $TP = i$ and $PP = j$, then the i-th character of the text is to be compared to the j-th character of the pattern. After a comparison, the pointers are updated, as indicated by the result. After a match, both pointers are moved one step forward. Note that if $PP = |p|$, the pattern has been found, and PP will point again to the first character of the pattern. If a mismatch occurs, two choices appear. If $PP = 1$, the text pointer is moved forward. In other cases, the text pointer is not updated. Never having to read backward is an advantage of the Knuth-Morris-Pratt algorithm over the naive algorithm. To update the pattern pointer PP, a function *next* is defined. For any given j, let the largest side of the substring $p[1]\dots p[j-1]$ already read be $p[1]\dots p[k-1]$. Then $next[j] = k$ and PP is moved to k. Remark that whenever the value of PP is 1, PP remains equal to 1 after a mismatch.

Example (b): Let $p = 012012123$ be the pattern and $t = 32012012012123321$ be the text. After the first comparison (a mismatch between $t[1] = 3$ and $p[1] = 0$), the text pointer is moved to 2, compared to 0 again. After the next move of TP to $t[3] = 0$, six matches occur. The string 012012 is recognized, PP points to $p[7] = 1$ and TP to $t[9] = 0$. After this mismatch, the longest side of $p[1]\dots p[6] = 012012$ is 012, then PP is moved to 4. Now, $t[9]$ matches to $p[4]$ and the whole pattern is found.

Remark: One can also use some additional information when a mismatch occurs, to move the text pointer even when PT is not $p[1]$. For example, assume:

$$p = 012012123$$
$$t = 320121012012123.$$

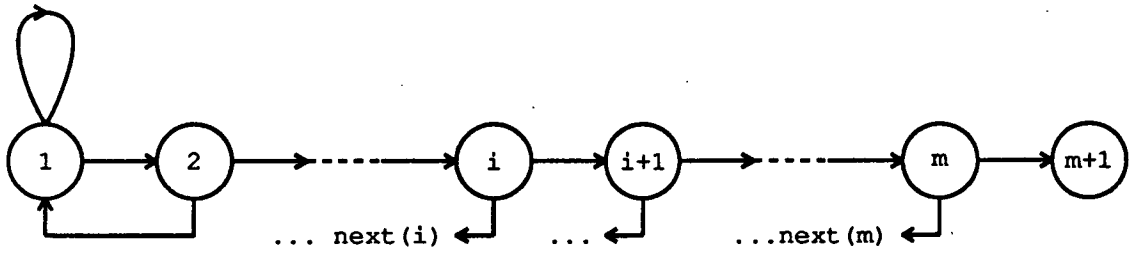
A mismatch occurs when $t[6]$ is compared to $p[4]$: obviously, $t[6]$ is also different of $p[1] = p[4]$. Hence, we can avoid this comparison: the text pointer TP is moved to $t[7]$ and compared to $p[1]$. This *variant* will be studied in Section V.

Using an automaton: One can also construct an automaton. Then each character in the text is compared only once to the pattern (our analysis does not apply then). But this automaton may have many states-between $q|p|$ and $q^{|p|}$- and be expensive to construct.

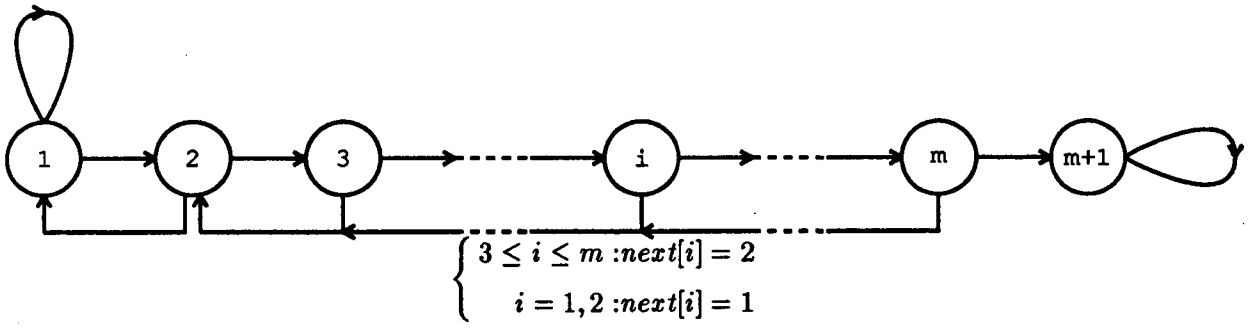
II.2. Probabilistic analysis:

A probabilistic approach -via Markov chains- to evaluate the number of text-pattern comparisons performed is proposed in[BA85] and improved in [BA88]. We describe both of them and show why this approach cannot provide exact evaluations.

To modellize the process by a Markov chain, Bath defines a set of states. State i, $0 \leq i \leq |p|+1$, occurs when $(i+1)$ matches have been found. Note that in state $|p|+1$, the entire pattern has been found: such a state is absorbant if we are only interested in finding the *first* occurrence of the pattern p. Now, from any state $i \leq |p|$, transitions are possible only to either one of the two states: $i+1$ or $next[i]$ depending whether the comparison ends in a match or in a mismatch. One gets the scheme:



It follows that different patterns (associated to different functions *next*) define different Markov chains. I.e. this Markov chain is ***not homogenous in space.*** For all possible Markov chains, transition probabilities should be defined. Considering that most of the transitions are to states 1 or 2, an approximate model is defined:



Then, the parameter of interest, the number of comparisons to be performed, is claimed to be equal to the **expectation of the absorbing state.** This is also, by definition, the number of steps the process makes from a start until absorption. Unfortunately, this claim is true for the naive algorithm, but is false for the Knuth-Morris-Pratt algorithm. As a matter of fact, in the naive algorithm, when a mismatch occurs on the character $p[j+1]$, the text pointer is always moved j steps backward. Thus, j lectures and comparisons are "forgotten", and will be repeated later. Hence, j is a correct estimation of the cost. As the corresponding transition of the Markov

chain is from j to 1, it is also the number of steps the process made since the last passage through state 1. But this claim turns out to be false for the Knuth-Morris-Pratt algorithm. The text pointer is never moved backward. When a mismatch occurs, the last character read, only, is "forgotten" and recompared to a character of the pattern as indicated by the transitions of the chain: the cost is *constant.* Hence, the final cost is now the number of times a backward transition is chosen. Such a parameter cannot be easily derived from the classical theory of Markov chains. In particular, the approximate model described above is of no help.

Baeza-Yates [BA88] improves this scheme. He considers the average number of comparisons to find all occurrences of a given pattern, that can be derived from the study of Markov chains. The basic idea is to use the **steady states** and find the fundamental matrix. The parameter to be evaluated is the **shift**: a shift is defined, for each mismatch, as the number of characters read since the last mismatch. Now, one out of these s characters is compared twice to the pattern. Hence, the cost of the algorithm is:

$$1 + E(\frac{1}{s}) .$$

From the steady state probability vector, one derives $E(s)$ and Kantorovitch inequality allows to get an approximation of $E(1/s)$ from $E(s)$.

The Markov chain that modellizes the algorithm is different from the one above. As remarked in [BA88], $i + 1$ states is not enough, because of the memory of the algorithm. (The transition probabilities out of a state may depend from the previous transition(s) to this state). Let us first consider an example, assuming the size of the pattern p is 2. We note $p = p[1]p[2]$. Two cases may occur: $p[1] = p[2]$ or $p[1] \neq p[2]$, which induce two different Markov chains. We are led to define 3 states.

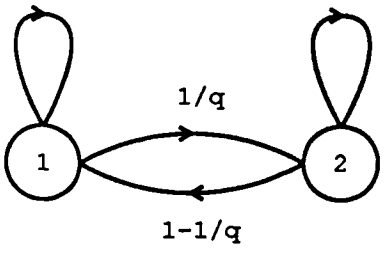
State 1: the current text character x is to be compared to $p[1]$, with no knowledge on x.

State 2: the current text character x is to be compared to $p[2]$.

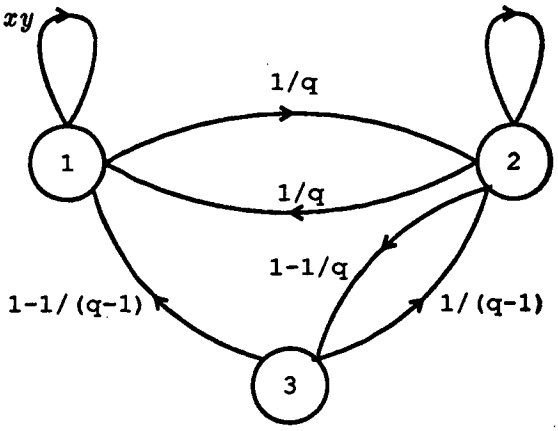
State 3: the current text character x is to be compared to $p[1]$, with the additional knowledge: $x \neq p[2]$.

At first, let us remark that State 3 is meaningful iff $p[1] \neq p[2]$. The assertion $x \neq p[2]$ implies $x \neq p[1]$ when $p[1] = p[2]$; hence there is no need for a comparison. We get to this case if we have a mismatch when comparing some text character x to $p[2]$. Such a character has to be recompared to $p[1]$. With the conditional knowledge $x \neq p[2]$, the probability for a match (i.e. to go to state 2) is $\frac{1}{q-1}$ and not $\frac{1}{q}$. The reader will notice that States 1 and 3 are almost equivalent: in both cases, a comparison to $p[1]$ is performed and the outgoing states are identical. But the transition probabilities are different as well as the ingoing states.

5g Finally, one gets two Markov chains:



(a) $p = xx$ (b) $p = xy$



Note that on a q-ary alphabet, q 2-patterns are associated to scheme (a) while $q(q-1)$ 2-patterns are associated to scheme (b).

The limits of the probabilistic methods appear now clearly. The Markov chains that modellize the algorithm are non homogenous in space. When the length k of the pattern p increases, the number of possible chains increases rapidly. Moreover, the knowledge of possible chains for $|p| = k$ does not help to get the possible chains when $|p| = k + 1$. Hence the model quickly becomes untractable. An approximation has to be done. In [BA88], the author realizes an embedding in a **homogenous** Markov chain. This approximation is illegal: one cannot ensure that the steady states for this approximate model is a correct approximation of the steady states of the non-homogenous Markov chain. In any case, the parameter of interest, namely $E(1/s)$ is not derived directly but deduced from $E(s)$ via the Kantorovitch inequality. This introduces an other, and important, approximation.

Our approach is algebraic. We derive the exact (algebraic) expression of the number of comparisons text-pattern performed by developing suitable generating functions. The basic idea to get these generating functions is the following. Any character in the text is read once or twice. This depends of the p *previous characters in the text*, at most. Hence, we will consider, classify and enumerate the possible occurrences of p-patterns, or substrings of p-patterns, in the text.

III. SOME PROPERTIES OF GENERATING FUNCTIONS:

Let E be a set such that a **size** can be defined on the elements of E. Here E will be a set of words and the size of a word will be its length. Let s be some parameter defined for all the elements (number of nodes in a graph, depth of a tree,...). The generating function associated to s is defined as:

$$S(z) = \sum_n s_n z^n$$

where s_n is the **average value** of the parameter s on the elements of size n. It may happen that all the elements of E can be built from simple elements of E via simple combinatorial constructions. If these constructions easily translate on relations on generating functions [FL88], the generating function approach may become a quite powerful tool. In the following, we give some translation rules. $A(z)$ and $B(z)$ are the generating functions associated to a same parameter computed on different subsets A and B.

(a) **additivity:** The generating function associated to the disjoint union of two sets: $A \cup B$, $A \cap B = \emptyset$, is the **sum** of the associated generating functions:

$$A(z) + B(z).$$

Proof: $\sum_{\omega \in A \cup B} z^{|\omega|} = \sum_{\omega \in A} z^{|\omega|} + \sum_{\omega \in B} z^{|\omega|}$.

(b) **Concatenation:** Let us consider the words c which are a concatenation of two words $a \in A$ and $b \in B$. The associated generating function is the **product:**

$$C(z) = A(z).B(z) .$$

Proof:

$$\sum_{c \in C} z^{|c|} = \sum_{a \in A, b \in B} z^{|a.b|} = \sum_{a \in A, b \in B} z^{|a|+|b|} = \sum_{a \in A} z^{|a|} . \sum_{b \in B} z^{|b|} = A(z).B(z).$$

(c) **Repetitions** of strings translate easily by an **exponentiation** of the variable to some power. For example, one associates

$$B(z^2) \text{ and } B(z^2).A(z)$$

to the concatenated words:

$$b.b \text{ and } b.a.b$$

Proof:

$$\sum_{a \in A, b \in B} z^{|bab|} = \sum_{a \in A, b \in B} z^{2|b|} . z^{|a|} = \sum_{b \in B} z^{2|b|} . \sum_{a \in A} z^{|a|} = B(z^2).A(z).$$

(d) $\sum_n \alpha^n [z^n] A(z) = A(\alpha)$. This relation is obvious from the definition. It will be useful in the paper.

Examples: We compute here the generating functions associated to some basic, and useful, sets of words.

(i) Let a be a given character. The set a^* (resp. the set a^+) is enumerated by:

$$a(z) = \frac{1}{1-z} \text{ (resp.} a^+(z) = \frac{z}{1-z}.$$

Proof: a^* contains exactly one word of size n: a repeated n times. Hence, applying (a), one gets:

$$a(z) = \sum_{n \geq 0} 1.z^n = \frac{1}{1-z},$$

while:

$$a^+(z) = \sum_{n \geq 1} 1.z^n = \frac{z}{1-z},$$

(ii) Let A be a q-ary alphabet. The generating function enumerating the number of words is:

$$A(z) = \frac{1}{1-qz}.$$

Proof: A^* is the disjoint union of the subset of words of size n. One has q^n words of length n. Hence:

$$A(z) = \sum_{n \geq 0} q^n z = \frac{1}{1-qz}.$$

Notation: Let $W_q(z) = \frac{1}{1-qz}$ be the generating function of the words on a q-ary alphabet.

(iii) The words beginning by a given character a are enumerated by the series S :

$$S(z) = z.W_q(z).$$

Finally, let us give some definitions and notations from combinatorics of words.

Definition 2: *A word $v \in A^*$ is said to be a* **factor** *of a word $x \in A^*$ if there exist words $u, w \in A^*$ such that:*

$$x = uvw.$$

A word $v \in A^*$ is said to be a **left factor** *or a* **prefix** *(resp.a* **right factor** *or a* **suffix** *) of a word $x \in A^*$ if there exists a word $w \in A^*$ such that:*

$$x = vw \ (\textbf{resp.} x = wv) .$$

We will note in the following:

$$v \preceq x (\textbf{resp.} v \subseteq x)$$

or, when the factors are strict:

$$v \prec x (\textbf{resp.} v \subset x).$$

IV. ENUMERATING THE COMPARISONS TEXT-PATTERN:

We first present our basic ideas. This leads us to define a notion of **quasi-mismatch.** The generating function of quasi-mismatches, studied in Section IV.2., gives an upper bound on the cost. Generalizing this notion of quasi-mismatch, we can deduce, in IV.3., the generating function of the total cost.

IV.1. The basic ideas:

We remark that the Knuth-Morris-Pratt algorithm is almost *memoryless.* Given a character a in a text, the number of comparisons performed on this character depends only of the word of size $|p|$ preceding it in the text, $|p|$ being the size of the searched pattern. Hence, we will enumerate possible preceding words inducing k comparisons. The average number of comparisons on a will follow. To do so, we introduce the fundamental notion of *quasi-mismatch*, which will provide an *upper bound*:

Definition 3: *Let a be some character in the alphabet A. A* **quasi-mismatch** *is the occurrence in the text of a pattern $p'a$ such that: $p' \preceq p$, $p'a \not\preceq p$*

Remark: A **quasi-mismatch** is the occurrence of a string p'a such that a comparison with **p** would imply a first mismatch on a **and** a second lecture of a.

We shall prove that a second lecture of a character a implies a quasi-mismatch. Hence, counting the expectation of quasi-mismatches in the text, we will get an upper bound on the number of extra-comparisons.

Proposition 1: *Let a be some character occurring in the text. A second lecture of this character implies the existence of a quasi-mismatch.*

Proof: Let a be a character occurring in a text. If the first comparison of a with the pattern ends in a match, or if this comparison is: $a?p[1]$, the text pointer is moved and a will not be compared again to the pattern. But, if the comparison ending in a mismatch was:

$$a \ ? \ p[j], \quad 1 < j \leq |p|,$$

then the largest side of $p[1]\dots p[j-1]$, say $p[1]\dots p[k-1]$, is searched and a is compared to $p[k]$. This is the only case of multiple comparison. It implies that the predecessors of a be $p[1]\dots p[j-1]$. This is a sequence $p' \preceq p$. Moreover, this ends in a mismatch iff $p[1]\dots p[j-1]a \not\preceq p$.

Remark:

(i) This necessary condition is not sufficient. Assume for example:

$$\begin{cases} p = 1021034 \\ t = \dots 1021037\dots \end{cases}$$

One has: $10 \preceq p$ and $103 \not\preceq p$ but 3 will be read only once, and match the pattern. It is worth noticing in this counter-example that 103 is a **factor** (but not a left factor) of the pattern.

(ii) The condition in Proposition 1 may be satisfied by several prefixes p'. For example, if:

$$\begin{cases} p = 01024 \\ t = \ldots 0103\ldots \end{cases}$$

one has $03 \not\preceq p$ and $0103 \not\preceq p$ while $0 \preceq p$ and $010 \preceq p$.

To evaluate the expectation of quasi-mismatches, we introduce the following notation:

Definition 4: *Let a be some character from an alphabet A. Note:*

$$F^{(a)}(z) = \sum_n f_n^{(a)} z^n$$

where $f_n^{(a)}$ is the number of prefixes of length n of all possible patterns that may induce a quasi-mismatch for a at position n. We note $F(z) = \sum_n f_n z^n$ the generating function counting quasi-mismatches at position n. One has:

$$\begin{cases} f_n = \sum_{a \in A} f_n^{(a)} = q.f_n^{(a)} \\ F(z) = \sum_{a \in A} F^{(a)}(z) = qF^{(a)}(z) = \sum_n f_n z^n \ . \end{cases}$$

In the next Section, we will derive this generating function, using the constructors recalled in Section III. As any additional comparison implies a quasi-mismatch, this will give an upper bound on the number of extra comparisons.

IV.2. An upper bound of the cost:

The aim of this Section is the derivation of Theorem 2, which establishes an upper bound to the number of extra comparisons performed when using the Knuth-Morris-Pratt algorithm.

Theorem 2: *For a given pattern p and a given text t of length n, let M_n and C_n be the random variables counting the number of quasi-mismatches and of extra comparisons . We note $\bar{M}_n$ and $\bar{C}_n$ the average values taken over all possible random patterns of size $|p|$ and texts of size n. One has:*

$$\begin{cases} C_n \leq M_n \\ \dfrac{\bar{C}_n}{n} \leq \dfrac{\bar{M}_n}{n} \end{cases}$$

and:

$$\frac{\bar{M}_n}{n} = F_p(\frac{1}{q^2}) = \frac{1}{q} - \frac{1}{q^{|p|}}$$

where:

$$\begin{cases} F(z) = qF_a(z) = qP_1(z).P_2(z) = \dfrac{q^2(q-1)z^2}{1-qz} \\ P_1(z) = q.\dfrac{z}{1-z}.\dfrac{(q-1)z}{1-(q-1)z} \\ P_2(z) = \dfrac{(1-z)(1-(q-1)z)}{1-qz} \end{cases}$$

and $F_p(z)$ is the truncation of F at order p.

Proof: Rule (b) in Section III will apply for suitable factorizations of the patterns. An example was treated informally in Section IV.1. above. More formally, we have:

Factorization Lemma 3: *Let a be a character. Then any non-empty string σ can be factorized, in a unique manner, as:*

$$\sigma = p_1 p_2 p_3$$

where:

$$p_1 = \begin{cases} a^* s_1, \\ ba^* s_1,\ b \in A - \{a\} \\ bs_1,\ b \in A - \{a\} \end{cases}$$

$$p_2 = \begin{cases} a^* s_2 \ldots a^* s_m \\ \epsilon \end{cases}$$

$$p_3 = \begin{cases} a^* \\ \epsilon \end{cases}$$

and s_i is a **non-empty** *word from $A^* - a^*$.*

Example (c): Let $\sigma = 011023105$. Depending on the character a, σ has different decompositions. We give three examples, corresponding to the three possible expressions for p_1.

***a=2:** (case (i))

$p_1 = 0110, b = 0, s_1 = 110$

$p_2 = 23105, s_2 = 3105.$

***a=1** (case (ii))

$p_1 = 011, s_1 = 11, b = 0$

$p_2 = 023105, s_2 = 23105$

***a=0** (case (iii))

$p_1 = 011023, b = 0, s_1 = 023$

$p_2 = 105, s_2 = 05.$

One can understand intuitively this factorization. a is chosen as a partitioning element as, clearly, a quasi-mismatch for a at position j implies $p[j] \neq a$. Only occurrences of prefixes σ of the searched pattern p, where $p_3 = \epsilon$, *i.e.* with a last character different from a, can induce quasi-mismatches, when this last character is replaced by a. p_1 represents the minimal prefixes that may induce quasi-mismatches, depending on the searched pattern p. When a comparison $a?p[1]$ fails, a is not read twice; hence $|p_1| > 1$. More precisely, no quasi-mismatch may occur before a position $j > 1$

such that: $p[j] \neq a$, and we get the three exclusive expressions. Finally, $f_n^{(a)}$ is the number of admissible prefixes p_1p_2, for all possible patterns p.

$$\begin{aligned} f_n^{(a)} &= \#\cup_{p_1p_2\in A^*}\{p_1p_2 \preceq p, |p_1p_2| = n\} \\ &= [z^n]\sum_{p_1p_2} z^{|p_1p_2|} = [z^n]\sum_{p_1p_2} z^{|p_1|}z^{|p_2|} \\ &= [z^n]\sum_{p_1} z^{|p_1|}\sum_{p_2} z^{|p_2|}\,. \end{aligned}$$

Note that the two basic generaring functions involved, $\sum_{p_1} z^{|p_1|}$ and $\sum_{p_2} z^{|p_2|}$ do not depend on the character a, for reasons of symmetry. We derive their expressions in the following two Lemmas.

Lemma 5: *Let* $P_2 = \cup_{m\geq 1}E^m \cup \{\epsilon\}$ *with:*

$$E = \{x.y/x \in a^* - \{\epsilon\}, y \in A^* - a^*\}.$$

The generating function $P_2(z)$ *counting the words in* P_2 *is:*

$$P_2(z) \;= = 1 + \sum_{j\geq 1}\left[\frac{z}{1-z}.\frac{(q-1)z}{1-(q-1)z}\right]^j.$$

Remark: The elements of P_2 are the strings p_2 of Lemma 3.

Proof of Lemma 5: The elements of E are non-empty words obtained by concatenation of a non-empty word in a^* and a non-empty word on a $(q-1)$-ary alphabet: $A - \{a\}$. Applying rule (b) yields the generating function:

$$\frac{z}{1-z}.[W_{q-1}(z) - 1]\,.$$

Lemma 6: *With the notations of Lemma 3, we note* P_1 *the set of the strings* p_1 *associated to a given character. The generating function counting the number of words in* P_1 *is:*

$$P_1(z) = q.\frac{z}{1-z}.\frac{(q-1)z}{1-(q-1)z}\,.$$

Proof of Lemma 6:

* Let E_1 be (case (i)) the subset of P_1: $E_1 = (a^* - \{\epsilon\}).(A^* - a^*)$. The generating function associated to the set $a^* - \{\epsilon\}$ and to the set $A^* - a^*$ are: $\frac{z}{1-z}$ (see Example (i)) and $\frac{1}{1-(q-1)z}$ (see Lemma 5). Hence:

$$E_1(z) \;=\; \frac{z}{1-z}.\frac{(q-1)z}{1-(q-1)z}\,.$$

* Let E_2 be (case (ii)) the subset of P_1:

$$E_2 = \cup_{b\in A-\{a\}}[b.(a^* - \{\epsilon\}).(A^* - a^*)].$$

As there are $(q-1)$ different choices for b, we get:

$$E_2(z) = (q-1)z.\frac{z}{1-z}.\frac{(q-1)z}{1-(q-1)z}.$$

* Let (case (iii)) $E_3 = \cup_{b\in A-\{a\}}[b.(A-\{a\})^*]$. As there are $(q-1)$ choices for b, we get the product:

$$E_3(z) = (q-1)z.\frac{(q-1)z}{1-(q-1)z}\ .$$

P_1 being the disjoint union $E_1 \cup E_2 \cup E_3$, Lemma 6 follows.

We can turn now to the proof of Theorem 2. From Lemmas F,G and H, we know that:

$$F^{(a)}(z) = P_1(z).P_2(z)\ .$$

An extra-comparison on a character x implies the existence of a quasi-mismatch (see Proposition 1). Hence: $C_n \leq M_n$ and $\frac{\bar{C}_n}{n} \leq \frac{\bar{M}_n}{n}$. As a quasi-mismatch always occurs at some position n such that $n \leq |p|$, the function F is truncated. Now, a given string $p'x$, $|p'x| = n$ occurs in a text with a probability $\frac{1}{q^n}$. And $p'x$ is a prefix of the searched pattern with a probability $\frac{1}{q^n}$. Hence, for a given character a:

$$E(quasi-mismatch) = \sum_{n=1}^{p} f_n^{(a)}.\frac{1}{q^{2n}} = F_p^{(a)}(\frac{1}{q^2})\ ,$$

and:

$$\frac{\bar{C}_n}{n} = F_p(\frac{1}{q^2})\ .$$

IV.3. The average cost of Knuth-Morris-Pratt algorithm:

This subsection is devoted to the demonstration of Theorem 7, which enunciates the average cost of Knuth-Morris-Pratt algorithm. In the previous subsection, we have defined the notion of quasi-mismatch which allows to derive an upper bound on the cost. The examples (i) and (ii) given above indicate the origin of this overestimation. The condition enunciated in Proposition 1 is not a sufficient condition; moreover, some mismatches may be counted several times. We evaluate here this overestimation.

Theorem 7: *Let $C_n(p)$ be the average number of extra-comparisons performed by a Knuth-Morris-Pratt algorithm on a text of size n for a given pattern p. Let $\bar{C}_n$ be the average value, taken over all patterns of size $|p|$. It satisfies, asymptotically:*

$$\frac{1}{q} - \frac{1}{q^{|p|}} - \frac{q-1}{q^2}S_{|p|-1}(\frac{1}{q^2}) - F_{|p|-1}(\frac{1}{q^2}) \leq \frac{\bar{C}_n}{n} \leq \frac{1}{q} - \frac{1}{q^{|p|}} - \frac{q-1}{q^2}S_{|p|-1}(\frac{1}{q^2}).$$

Here $M(z,x)$ is the function defined below in Proposition 10; $S_{|p|-1}$ and $F_{|p|-1}$ are the truncations at order $|p|-1$ of the entire functions S and F defined as:

$$\begin{cases} S(z) &= \sum_{m\geq 1} M(z^m, z) \\ F(z) &= \frac{1}{q}[\sum_{m\geq 1} M(z^{2m}, z^2) W_q(z) + sum_{m\geq 2} M(z^m, z)] + \sum_{p\geq 1}[(p+1)M(z^{2p+1}, z) + pM(z^{2p}, z)] \end{cases}.$$

Corollary 8: *$\bar{C}_n$ can be developed as a function of $\frac{1}{q}$:*

$$\frac{\bar{C}_n}{n} \sim \frac{1}{q} - \frac{2}{q^4} + \frac{1}{q^5} - \frac{2}{q^6} + O(\frac{1}{q^7}).$$

Remark: Developing F yields: $F_{|p|-1}(\frac{1}{q^2}) \sim \frac{1}{q^7}$. Hence, the result is known up to $O(\frac{1}{q^7})$.

The overestimation, *i.e.* the difference between the number of quasi-mismatches and the number of extra-comparisons is, on the average, very closed to $S_{|p|-1}(\frac{1}{q^2})$. In the following, we characterize quasi-mismatches that induce an overestimation. Let us first consider two examples.

Example (d): $p = 0102$ and $t = \ldots 0103 \ldots$

In this case, the occurrences of 03 and 0103 in the text will both be counted as quasi-mismatches, for the same mismatching character 3.

It may also happen that the condition (I) is not sufficient if a prefix is also a factor in the text:

Example (e): Let $p = 0121101345$. One has: $01 \preceq 012 \preceq p$ while 013 is a factor of p. According to the definition, any occurrence of 013 will be counted as a quasi-mismatch. Nevertheless, 3 is not a mismatching character, when 01211013 occurs in the text.

In both cases, some repetition appears in the pattern p. In order to characterize these repetitions, we state the following lemma.

Multiplicity Lemma 9: *Let p' and $p"$, $|p'| < |p"|$, be two sequences such that:*

$$\begin{cases} & p' \preceq p" \\ & p' \subseteq p" \\ p'a \not\preceq p,\ p"a \not\preceq p \end{cases}$$

Then:

$$\exists (u,v) \in A^* \times A^* \ s.t. \begin{cases} vu & \in P \\ p' & = u(vu)^* \\ p" & = p'.(vu)^* \\ a & \not\preceq vu. \end{cases}$$

Proof: A basic result of combinatorics on words [LOTH] yields that, whenever $p' \preceq p''$ and $p' \subseteq p''$:

$$\exists (u,v) \in A^* \times A^* \ s.t. \ \begin{cases} p' & = u(vu)^* \\ p'' & = p'.vu \ . \end{cases}$$

Now, if $vu \notin P$, then $vu = z^m, z \in P$. It follows that $u \subseteq z^m$; hence, there exists a factorization: $z = ae$ such that $u = e.z^q$. Finally:

$$p'' \ = \ e.(ae)^q.(ae)^{mp}.(ae)^m \ = \ e.(ae)^*, \ ae \in P.$$

As $p' \prec p''$ and $p'a \not\preceq p$, one has $p'a \not\preceq p''$; hence: $a \not\preceq vu$. ∎

We are led to enumerate such words.

Proposition 10: *Let* $M(z,x) = \sum_{\substack{vu \in P \\ u \in A^*}} x^{|u|}.z^{|vu|}$. *It satisfies:*

$$\sum_{m \geq 1} M(z^m, x) \ = \ (W_q(zx) - 1)W_q(z) - V(z,x)$$

where: $V(z,x) = \frac{1}{1-x}[x \sum_{m \geq 1} P(z^m x) - W_q(zx) + 1]$

and: $P(z) \ = \ \sum_d \mu(d)[W_q(z) - 1](z^d)$ *is the generating function of primitive words. Equivalently:*

$$M(z,x) = \sum_d \mu(d)[(W_q(zx) - 1)W_q(z) - V(z,x)](z^d)$$

where μ *is the Möbius function.*

Proof: We enumerate the words $u.vu$ where u is some word in $A^* - \{\epsilon\}$. The generating function associated to these words is: $(W_q(zx) - 1)W_q(z)$. Now, $vu = t^*, t \in P$, and either $u \subseteq t$ or $u = et^p$, with $e \subset t$ and $p \geq 1$. In the first case, one may rewrite $u.vu$ as $u.(v'u)^m, m \geq 1$, and these words are associated to:

$$\sum_{\substack{u \in A \\ v'u \in P}} x^{|u|}.z^{|v'u|m} \ = \ M(z^m, x).$$

In the second case, one has:

$$u.vu \ = \ et^p.t^m$$

with the condition: $p < m$, as $u \subseteq vu$. Then:

$$\begin{aligned} \sum_{\substack{t \in P \\ e \subset t, \\ m > p \geq 1}} x^{|et^p|}.z^{m|t|} &= \sum_{t \in P} \sum_{m \geq 2} z^{m|t|} \sum_{p=1}^{m-1} x^{p|t|}. \sum_{j=0}^{|t|-1} x^j \\ &= \frac{1}{1-x} \sum_{t \in P} \sum_{m \geq 2} z^{m|t|}(x^{|t|} - x^{m|t|}) \\ &= \frac{1}{1-x} \sum_{m \geq 2} [P(xz^m) - P((zx)^m)]. \end{aligned}$$

As: $\sum_{m\geq 1} P(z^m) = W_q(z) - 1$, one gets:

$$\frac{1}{1-x}.[\sum_{m\geq 1} P(xz^m) - W(zx) + 1] .$$

Note that the two conditions $u \subseteq t$ or $u = et^p$ are not exclusive. More precisely:

$$\left\{ \begin{array}{l} u = et^p, p \geq 1, e \subset t \\ u \subseteq t \end{array} \right\} \Longleftrightarrow \left\{ \begin{array}{l} u = t \\ v \in t^* \end{array} \right.$$

The associated generating function is:

$$\sum_{u\in P} x^{|u|} z^{m|u|} = \sum_{m\geq 1} P(xz^m) .$$

Finally:

$$(W_q(zx) - 1)W_q(z) = \sum_{m\geq 1} M(z^m, x) + V(z, x).$$

Applying Möbius inversion formulæ, we get the expression of $M(z, x)$. ∎

A refined upper bound: We can turn now to the proof of the refined upper bound in Theorem 7. The term $S_{|p|-1}(\frac{1}{q^2})$ is the average number of times we count quasi-mismatches for matching characters (see Example (e)). Such events occur when:

$$p = u(vu)^l p_3, \ a \preceq p_3, a \preceq vu$$

and when $p' = u(vu)^l a$ occurs in the text. In such cases, all the l sequences:

$$ua, u(vu)a, \ldots, u(vu)^{l-1}a$$

are counted as quasi-mismatches, although the sequence p' is a matching sequence. In Example (e), we had:

$$a = 3, u = 01, v = 211, p_3 = 345, l = 1 .$$

If $|u(vu)^l| = n$, the probability to search such a pattern is $P(u(vu)^l a \preceq p).P(a \preceq vu) = \frac{1}{q^{n+1}}.\frac{q-1}{q}$; the probability of occurrence of p' in the text is $\frac{1}{q^{n+1}}$. Hence, as we have q choices for a, we must deduce from $F_p(\frac{1}{q^2})$:

$$\frac{q-1}{q^2} \sum_{m\geq 1} M_{|p|-1}(z^m, z)(z = \frac{1}{q^2}) = S_{|p|-1}(\frac{1}{q^2}) \times \frac{q-1}{q^2} .$$

∎

A refined lower bound: We have substracted so far the quasi-mismatches associated to matching characters of the pattern. We have now to consider the case when several quasi-mismatches are associated to a single mismatching character (see Example (d)). This implies:

$$\left\{ \begin{array}{l} p' = u(vu)^l a \\ \quad u(vu)^l \preceq p \ , \ a \not\preceq vu \end{array} \right.$$

(Note that whenever $a \preceq vu$, $u(vu)^{l-1}a \preceq p$ and a single quasi-mismatch is counted). If all decompositions of the matching prefix are $u(vu)^l$, then the next steps of this naive algorithm will compare all the sequences of words $u(vu)^j a$ to p and all these comparisons will end in error. Hence, the $l+1$ quasi-mismatches are associated to $l+1$ effective comparisons of a. We have then to consider the case: $u(vu)^l = z(tz)^m$ It may happen that some comparisons $u(vu)^j a?p$ are skipped.

Example (f): $p = 101210101210134$ and the mismatching sequence 10121010121012 occurs in the text. The matching prefix is: $101.2101012101 = 1.(1012101)(1012101)$. The longest matching prefix preceding 2 is not 1.(012101) but 2101012101. The next comparison is 2?2 which ends in a match. Hence, the comparison $1012?p$ will never be done.

One proves easily that either l or m is 1, say m. Now, two cases may occur:

(i) $a \preceq vu, a \not\preceq tz$: two quasi-mismatches: $u(vu)^l a = z(tz)a$ and za have been counted. As $u(vu)^{l-1}a \preceq p$, there was only 1 extra-comparison on a. Hence, the overestimation is 1.

(ii) $a \preceq tz, a \not\preceq vu$: the comparisons $u(vu)^j a?p$ such that $|u(vu)^j a| < |z|$ are not performed. As $|z| < 1/2|u(vu)^l|$, the overestimation is at most $\frac{l+1}{2}$ (resp. $\frac{l}{2}$) when l is odd (resp. even).

To get a better approximation of our overestimation, we distinguish the cases: $j \geq 2$ and $j = 1$. In any case, we have: $|z| < |u(vu)^{l-1}|$.

$j \geq 2$: In case (i), the overestimation is upper bounded by: $\frac{1}{q}\sum_{m\geq 2} M(z^m, x)(z = x = \frac{1}{q^2})$ In case (ii), it is upper bounded by: $[\sum_{p\geq 1} pM(z^{2p}, x) + \sum_{p|geq1}(p = 1)M(z^{2p+1}, x)](z = x = \frac{1}{q^2})$.

$j = 1$: The only case of interest is (ii). Then: $z = x(yx)^*$ and $u = x(yx)^*$. Hence,applying rule (c), we get the upper bound:

$$1.\frac{1}{q}.\sum_{m\geq 1} M(z^{2m}, x^2), W_q(z)(z = x = \frac{1}{q^2}) .$$

Summing these results, we get an upper bound of our overestimation, hence a lower bound for our algorithm. ∎

V. CONCLUSION:

In this paper, we realized an analysis on the average of the Knuth-Morris-Pratt algorithm. This algorithm is linear on the average, and we derived the constant of linearity, as a function of the cardinality q of the alphabet. Notably, when q is large, this constant c is equivalent to $1 + \frac{1}{q}$. Our scheme of analysis is algebraic. We first reduce the problem of finding the number of comparisons to a problem of *word enumeration.* Applying some known results from combinatorics of words, we are able to compute the associated generating functions. This shows an example where constructions of words satisfying certain properties translate into generating functions.

We believe this scheme can be extended to other string matching algorithms. Notably, variants of Knuth-Morris-Pratt can be treated analogously. We are currently working on the analysis of

Boyer-Moore algorithm, using an algebraic scheme and generating functions.

References:

[BA85] G. BARTH "An Analytical Comparison of Two String Matching Algorithms" in *Information Processing Letters,* **30** (1985) 249–256.

[BA88] R. BAEZA-YATES "Analysis of String-Matching Algorithms " Waterloo Univ. Research Report

[CP88] M. CROCHEMORE AND D. PERRIN "Pattern Matching in Strings" in LITP (Paris) Research Report 88-5.

[FL84] PH. FLAJOLET *"Mathematical Methods in the Analysis of Algorithms and Data Structures"*, Lecture Notes for *A Graduate Course on Computation Theory*, Udine(Italy) 225–304(1984).

[FL85] PH. FLAJOLET "Elements of a Theory of Combinatorial Structures" in *Proc. FCT Conf.*, Lecture Notes in Computer Science, (1985) 112-127.

[GJ83] I. GOULDEN AND D. JACKSON *"Combinatorial Enumerations"*, Wiley, New York, (1983).

[GO81] L.J. GUIBAS AND A.M.ODLYZKO "String overlaps, pattern matching, and nontransitive games" in *J. Comb. Theory (A),* **30** (1981) 183–208.

[KMP77] D.E. KNUTH, J.H. MORRIS AND V.R. PRATT "Fast Pattern Matching in Strings" in *SIAM J. Comput.* **6**, *2* (1977) 323–350.

[LO82] LOTHAIRE *"Combinatorics on Words"*, Addison-Wesley, Reading,Mass., (1982).

[OD85] A.M. ODLYZKO *"Periodicities in Strings"* in Combinatorial Algorithms on Words, Springer NATO ASI SEr. F12, (1985) 241–254.

[RE88] M. RÉGNIER *"Enumeration of Sided Words"* in preparation

[SE83] R. SEDGEWICK *"Algorithms"* Addison-Wesley, Reading, Mass.,(1983).

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique