We consider the following problem:

**Instance:** a finite alphabet $A$, a biprefix code $X=(x,y)$ whose elements are primitive, $weA^*$.

**Question:** find every maximal factors of $w$ which are prefixes of a word of $X^*$. We present an algorithm which solves the problem in time linear of the length of $w$, after a preprocessing phase applied to the set $X$.

1. **INTRODUCTION**

In Computer Science, the concept of factorization plays a prominent part for the applications and problems that it introduces.

For instance, given a subset $X$ in the free monoid $A^*$, it is of interest to compute the "rank" of $X$, i.e. the minimal cardinality of a finite set $Y$ such that $X \leq Y^*$ (i.e. every word of $X$ is the concatenation of words belonging to $Y$). It is established ([N 88]) that this problem is NP-complete and if $X$ has cardinality $k$ then there exists an $O(n^k)$ algorithm, where $n$ stands for the sum of the lengths of the words in $X$. 
As another example the problem of the "shortest common superstring" consists, given a set of words X, to construct a word w of minimal length such that every word of X factorizes w. This is also a classical NP-complete problem ([GJ 78], [TU 88]).

From another point of view, several "pattern matching" problems are known to be decidable by linear time algorithms. Let us mention the classical Knuth Morris and Pratt's algorithm [KMP 77], generalized in [AC 75], and an algorithm computing the longest common factor of two words [C 87]). All these algorithms make use of a notion of "failure function" which is also a main feature of the algorithm presented in this paper. "Failure functions" have recently found an application to text formatting ([HL 87]).

Given a biprefix set X=(x,y)xA whose all the elements are primitive, and given a word w, the problem is to find the longest factors of w which are prefixes of a word of X*. We present an algorithm which solves the problem in time linear in the length of w after a preprocessing phase applied to the set X. A simple modification of our algorithm permits to compute the longest factors of w which belong to X* in time linear of the length of w. This solution is a first step towards a linear algorithm to compute the rank of a three element code.

Let us now examine how a sequential algorithm can solve our problem. For a given word w∈A* let ϕ(w) be the longest suffix of w which is prefix of a word of X*. Assume that we have computed ϕ(w), for a given prefix w of the input word. Let a∈A such that ϕ(wa)≠ϕ(w)a. The word ϕ(w) induct an "X-interpretation" (cf [S 79]) of ϕ(wa)a⁻¹ which consists in fact to consider the word ϕ(wa)a⁻¹ as a factor of a word of X*. According to a result of [LS 67], thoroughly examined in [BL 85], if ϕ(wa)a⁻¹ is long enough then it is possible to precisely describe it. This leads to introduce two sets of prefixes of words of X*, namely L and S. Indeed, for every word w∈A* the word ϕ(w) belongs to LxA*S and consequently w belongs to A*L∪A*X*S. In this way, in a preprocessing phase, we shall compute two automata whose behaviour are respectively A*L and A*X*S. The complexity of the construction of such automata can be expensive, on account of the cardinality of the alphabet A. We solve this problem in introducing two particular suffixes of w, namely f(w)∈L and g(w)∈S. In fact, if after the reading of two words w,w' we are in the same state of the automaton whose behaviour is A*L (resp. A*X*S), then f(w)=f(w') (resp. g(w)=g(w')). This allows to define the functions f and g on the states of the corresponding automata -thus f and g are "failure functions"- and to