ALGORITHMS FOR FACTORIZING AND TESTING SUBSEMIGROUPS

Renato M. Capocelli
Dipartimento di Informatica ed Applicazioni
Universitá di Salerno
Salerno 84100, Italy

Christoph M. Hoffmann
Department of Computer Science
Purdue University
West Lafayette, Ind. 47907

ABSTRACT

Given a finite subset Γ of a fixed, finite alphabet Σ, we construct the basis B of the minimum subsemigroup of Σ^+ containing Γ, such that B has various properties. The properties we consider are that B be a uniquely decipherable, a finitely decipherable, a synchronizable, or a prefix code. The algorithm for constructing the uniquely decipherable and the finitely decipherable code B requires $O(n^2 L + L^2)$ steps, the algorithm for constructing the synchronizable code B requires $O(n L^2)$ steps, and the algorithm for constructing the prefix code B requires $O(L^2)$ steps. Here n is the cardinality of Γ and L is the sum of the lengths of the words in Γ. Finally, given a synchronizable or finitely decipherable code Γ, we also show how to determine its synchronizability or decipherability delay, in $O(n L)$ steps.

1. Introduction

Consider the problem of transmitting messages written in a source alphabet over a channel which has a different, smaller alphabet. In such a situation, the source message has to be encoded, and a particularly convenient method is to substitute, for each letter of the source alphabet, a nonempty word over the channel alphabet. Given a fixed channel alphabet Σ, a code is then a subset of Σ^+, the set of all nonempty words over Σ, and if we are presented with such a subset as proposed code, it is natural to investigate certain properties of this set.

The properties of unique decipherability, of finite decipherability, and of synchronizability are central to Coding Theory. They formalize whether a coded message can be unambiguously decoded, whether this decoding is possible without first receiving the entire message, and whether an infinite message can be decoded without knowing its beginning and its end.

Since every encoded message is a concatenation of words, these properties can also be understood as concepts from Semigroup Theory. Specifically, for finite sets $\Gamma \subset \Sigma^+$, the set Γ of code words generates the subsemigroup Γ^+ of Σ^+, where Σ^+ is the free semigroup with $|\Sigma|$ generators. Viewed in this way, Γ is uniquely decipherable precisely when Γ^+ is a free subsemigroup of Σ^+, Γ is finitely decipherable whenever Γ^+ is a weakly prefix subsemigroup [Cap 80], and Γ is synchronizable iff Γ^+ is a very pure subsemigroup [Res 74]. In each case, Γ must be the minimum generating set of Γ^+ as well.

In particular, Γ has a decipherability delay of zero iff no code word is prefix of another code word. Such a code is called a *prefix* code, and Γ^+ is a left unitary subsemigroup. Prefix codes play an important role in Coding Theory both for their intrinsic properties, as well as

NATO ASI Series, Vol. F12
Combinatorial Algorithms on Words
Edited by A. Apostolico and Z. Galil

because they are easily constructed.

Sardinas and Patterson [SP 53] and Levenshtein [Lev 62, Lev 64] provided criteria for testing whether a code is uniquely decipherable, finitely decipherable, or synchronizable. Variations of these criteria can be found in [Mar 62, Eve 63, Eve 64]. A unified treatment of the above properties can be found in [Ril 67, Spe 75, Cap 79].

In case a set Γ does not possess some or all of these properties, one might want to determine the "most convenient" codes Δ possessing them and such that Γ^+ is a subset of Δ^+. It turns out that such codes are the unique minimum generating sets for the minimum subsemigroup of Σ^+ containing Γ and possessing the desired properties. This question has been investigated in [Spe 75, Cap 77, Cap 80]. The purpose of this paper is to give efficient algorithms for finding such codes with the desired properties, given a set Γ not enjoying them. Such algorithms find application in, e.g., data compression [CR 78]. Past work on such algorithms has been done in [Spe 75, Cap 77, DR 79, Ber 79, Cap 80], but specific complexity bounds on their performance have not been established except in the most general sense. We rectify this situation by deriving efficient algorithms for the above mentioned problems and analyzing in detail their complexity. Moreover, we give algorithms for determining the decipherability delay and the synchronization delay for codes.

Most of our algorithms depend on a new test for unique decipherability whose performance puts it on equal footing with the most efficient unique decipherability tests previously reported [Rod 82, AG 84]. Unlike these other algorithms, the one given here has special properties which make it uniquely suitable to our purposes.

2. Terminology and Basic Algorithms

In the following, Σ is a fixed finite alphabet, Σ^+ the free semigroup generated by Σ, and $\Sigma^* = \Sigma^+ \cup \{\lambda\}$ is the free monoid over Σ. Let Γ be a finite subset of Σ^+. We call elements of Γ *code words*. Γ^+ is the set of all words in Σ^+ which are concatenations of code words. The elements of Γ^+ are usually called *messages*. When considering Σ^+ as the free semigroup generated by Σ, Γ^+ may be viewed as a subsemigroup of Σ^+.

Definition
A subsemigroup P of Σ^+ is *left unitary* if, for all $p \in P$ and all $m \in \Sigma^+$, $m \in P$ whenever $pm \in P$.

Definition
A set $\Gamma \subset \Sigma^+$ is a *prefix code* if no code word is prefix of another code word.

It is easily verified that a subsemigroup of a free semigroup is left unitary iff its unique minimum generating set, $P - P^2$, is a prefix code. Here P^2 is the set of all elements of P obtained as p_1p_2, where $p_1, p_2 \in P$. In the following, we refer to this unique minimum generating set as the *basis* of P.

Definition
A set $\Gamma \subset \Sigma^+$ is *uniquely decipherable* if, for all $c, c' \in \Gamma$ and $x, y \in \Gamma^*$ the equation $cx = c'y$ has no solution with $c \neq c'$.

If Γ is not uniquely decipherable then the solution to $cx = c'y$ yields the ambiguous message cx, which can be deciphered in at least two different ways. Clearly Γ is uniquely decipherable iff Γ^+ is a free subsemigroup of Σ^+ with basis Γ (see [Lal 79]).

Definition
A subsemigroup S of Σ^+ is *very pure* if, for all $x, y \in \Sigma^+$, $xy \in S$ and $yx \in S$ imply that $x, y \in S$.

Definition
A set $\Gamma \subset \Sigma^+$ is a *synchronizable code* if there exists a number $s > 0$ such that, for all $f \in \Gamma^s$, there exist $f_1, f_2 \in \Sigma^*$ such that $f = f_1 f_2$ and, for all $u, v \in \Sigma^*$, $uf v \in \Gamma^*$ implies that both $uf_1 \in \Gamma^*$ and $f_2 v \in \Gamma^*$.

The smallest integer s for which Γ satisfies this definition is the *synchronization delay* of Γ.

Intuitively, Γ is synchronizable if it is possible to decipher message fragments of sufficient length. Synchronizability delay s here means that whenever a fragment contains s consecutive code words, then a correct code word boundary can be determined. Note that our definition of synchronizability delay differs from the one made in terms of synchronizing pairs (e.g., [Lal 79]). However, the two definitions of *synchronizable code* are clearly equivalent.

Obviously, a very pure subsemigroup is a free subsemigroup. A finitely generated subsemigroup is very pure iff its basis is a synchronizable code [Res 74].

Definition
A subsemigroup D of Σ^+ is said to be *weakly prefix* if, for all $a \in D$ and $x, y \in \Sigma^+$, the relations $ax \in D$, $xy \in D$, and $yx \in D$, together imply $x, y \in D$.

Obviously a weakly prefix subsemigroup of Σ^+ is also a free subsemigroup. Moreover, a left unitary (or very pure) subsemigroup is also weakly prefix.

Definition
A subset $\Gamma \subset \Sigma^+$ is *finitely decipherable* if there is an integer $s \geq 0$ such that, for all $u \in \Sigma^*$, $x \in \Gamma^+$, and $y \in \Gamma^s$, $xyu \in \Gamma^*$ implies that $yu \in \Gamma^*$.

The smallest integer for which Γ satisfies the definition is called the *decipherability delay* of Γ. Intuitively, Γ has decipherability delay s if the next code word can be correctly deciphered only after s subsequent code words have been received (unless, of course, the message has ended). Conversely, a uniquely decipherable code with infinite decipherability delay does not permit a correct decoding until the entire message has been received. It can be shown that if D is a finitely generated, weakly prefix subsemigroup, then its basis is a code with finite decipherability delay [Cap 80].

We note that the notions of synchronizable codes and finitely decipherable codes are the natural extensions of comma-free codes [GGW 58] and prefix codes, respectively.

It is clear that the intersection of a family of left unitary (free, very pure, weakly prefix) subsemigroups of Σ^+ is again a left unitary (free, very pure, weakly prefix) subsemigroup of Σ^+. Therefore, given $\Gamma \subset \Sigma^+$, there always exists the *minimum* left unitary (free, very pure, weakly prefix) subsemigroup of Σ^+ containing Γ^+. The proofs can be found for free subsemigroups in

[Til 72], for left unitary subsemigroups in [Sch 73, Cap 77, Ber 79], for very pure subsemigroups in [DR 79, Cap 80], and for weakly prefix subsemigroups in [Cap 80].

The respective bases have been constructed for the minimum left unitary subsemigroup in [Cap 77, Ber 79], for the minimum very pure subsemigroup in [DR 79, Cap 80], for the minimum free subsemigroup in [Spe 75, Cap 77, Ber 79], and for the minimum weakly prefix subsemigroup in [Cap 80].

The Defect Theorem implies that if Γ is not uniquely decipherable or not finitely decipherable, then the cardinality of the basis of the minimum subsemigroup containing Γ must be less than the cardinality of Γ, [Ber 79, Cap 80b]. For the basis of the minimum very pure subsemigroup [DR 79] and the minimum left unitary subsemigroup [Ber 79] only a weaker statement holds, namely, that the cardinality of the basis cannot exceed the cardinality of Γ.

Let $\Gamma \subset \Sigma^+$, and assume that P, F, S, and D are, respectively, the minimum left unitary, free, very pure, and weakly prefix subsemigroup of Σ^+ containing Γ^+. Then the following inclusions hold:

$$\Sigma^+ \supset S \supseteq D \supseteq F \supseteq \Gamma^+$$

$$\Sigma^+ \supset P \supseteq D \supseteq F \supseteq \Gamma^+$$

The construction of bases for these minimum subsemigroups is considered next.

2.1 Algorithms for Minimum Subsemigroups

We now present the conceptual algorithms of Capocelli for obtaining the bases for minimum subsemigroups containing Γ^+ which are left unitary, free, very pure, or weakly prefix. The treatment given is, we believe, a good deal simpler than the alternatives put forth in the literature, as it is based explicitly on certain fundamental combinatorial properties of code words and their sequences.

Intuitively, the algorithms work by eliminating those code words which deprive Γ of the desired properties, substituting appropriate shorter words.

Definition
Let $\Gamma \subset \Sigma^+$. An *L-sequence* of prefix s_0 and suffix s_n is a finite sequence $\sigma = (s_0, c_1, ..., c_n, s_n)$ where the c_i are in Γ such that

$$\begin{aligned}
c_1 &= s_0 s_1, \\
c_2 &= s_1 s_2, \quad \text{or } s_1 = c_2 s_2, \\
c_3 &= s_2 s_3, \quad \text{or } s_2 = c_3 s_3, \\
&\cdots \\
c_n &= s_{n-1} s_n, \text{ or } s_{n-1} = c_n s_n,
\end{aligned}$$

and where the s_i, $0 \leq i \leq n$, are strings in Σ^+. The L-sequence is *minimal* if, for $0 \leq i, j < n$ and $i \neq j$, we have $s_i \neq s_j$

Definition
An L-sequence σ is a *dividing sequence* if s_0 and s_n are code words. If c_1 is the second code word in σ, then we also say that σ *divides* c_1.

Definition
An L-sequence σ is a *loop sequence* if $s_0 \in \Gamma$ and $s_n = s_i$ for some $i < n$.

Definition
An L-sequence σ is *cyclic* if $s_0 = s_n$

Intuitively, an L-sequence specifies a pair of messages x and y such that $x = s_0 u$ and $y = u s_n$, for some string u, or $x = s_0 y s_n$. The code words of the L-sequence specify how x and y may be deciphered, and the strings s_i show how prefixes of x and y which are in Γ^* are related.

Example
Consider the L-sequence $(s_0, c_1, ..., c_5, s_5)$, where $s_1 = c_2 s_2$, $s_2 = c_3 s_3$, $c_4 = s_3 s_4$, and $c_5 = s_4 s_5$. Then the message x is $c_1 c_5 = s_0 y s_5$, where $y = c_2 c_3 c_4$. ∎

Theorem (Capocelli)
The set $\Gamma \subset \Sigma^+$ is not uniquely decipherable (not finitely decipherable, not synchronizable) iff there exists a dividing sequence (loop sequence, cyclic sequence).

The algorithm for finding the basis of the minimum subsemigroup containing Γ which is free (weakly prefix, or synchronizable) proceeds by finding a dividing sequence (a dividing sequence or a loop sequence; a dividing sequence, a loop sequence, or a cyclic sequence) and replacing the leading code word $c = s_0 s_1$ of the sequence by the shorter words s_0 and s_1. Note that in a dividing sequence and in a loop sequence s_0 is in Γ.

Algorithm A

1. Initialize C to $\Gamma \subset \Sigma^+$.
2. Construct a dividing sequence (dividing sequence or loop sequence; dividing sequence, loop sequence, or cyclic sequence) σ. If none exists, then go to Step 6.
3. Set $U := C$.
4. Let $\sigma = (s_0, s_1, ...)$ be the sequence, $c = s_0 s_1 \in U$. Set $U := U - \{c\} \cup \{s_0, s_1\}$.
5. Set $C := U$ and return to Step 2.
6. Stop. C is now the basis for the minimum subsemigroup containing Γ which is free (weakly prefix, very pure).

The algorithm for finding the basis of the smallest left unitary subsemigroup containing Γ is the following

Algorithm B

1. Initialize C to $\Gamma \subset \Sigma^+$.
2. If no element of C is prefix of another element of C then go to Step 6.
3. Set $U := C$.
4. If an element p of U is prefix of another element q of U, say $q = pr$, then set $U := U - \{q\} \cup \{r\}$.
5. Set $C := U$ and return to Step 2.
6. Stop. C is now the basis for the minimum subsemigroup containing Γ which is left unitary.

Correctness of Algorithm A has been shown in [Cap 77, Cap 80], and correctness of Algorithm B in [Cap 77].

3. An Algorithm for Unique Decipherability

We wish to sketch an algorithm for testing unique decipherability on which we will base the implementation of Algorithm A. The unique decipherability algorithm is due to Hoffmann [Hof 84], and its asymptotic performance, $O(n\,L)$, is equal to that of the other two fast algorithms in the literature [Rod 82, AG 84].

There is a considerable literature dealing with unique decipherability testing, e.g., [SP 53, Mar 62, Lev 62, Eve 63, Blu 65, Spe 75, Cap 79, Rod 82, AG 84]. All proposed algorithms are similar in that they attempt to construct a counter example to unique decipherability. However, most algorithms do not construct an explicit representation for all deciphering ambiguities.

There are three efficient unique decipherability tests, namely [Rod 82, AG 84], and the algorithm to be given. Conceptually, our algorithm is related to Spehner's, [Spe 75], whereas the other two are implementations of the algorithm given in [SP 53]. Moreover, the others do not explicitly construct a representation for all deciphering ambiguities, and it is precisely this explicit representation which is needed later.

Spehner's algorithm could be modified to construct a graph whose vertices are the set of suffixes of the words in Γ. There is an edge (u, v) if $u = cv$ or $c = uv$ for some $c \in \Gamma$. Finding a dividing sequence now means finding a path from a certain set of initial suffixes to the empty suffix.

Instead of working with suffixes as vertex set, our algorithm takes the set of prefixes as vertex set of an equivalent graph. Each prefix simultaneously encodes all suffixes whose concatenation results in a code word. This graph is explored breadth-first to determine the existence of a path from a distinguished vertex set to a vertex representing a code word. Any such path is shown to yield a deciphering ambiguity, and such a solution can be constructed from it in linear time. We make use of the pattern matching technique of [AC 75].

3.1 The Graph $R(\Gamma)$

Let Γ be a finite set of nonempty words over a fixed alphabet Σ, and assume we wish to test whether Γ is uniquely decipherable. The heart of the algorithm to be presented is a graph $R(\Gamma)$, which makes explicit a number of properties of Γ. We define the set

$$\mathit{Prefix}(\Gamma) = \{u \in \Sigma^* \mid u < c,\ c \in \Gamma\},$$

the set of all prefixes of words in Γ. Of course $\Gamma \subset \mathit{Prefix}(\Gamma)$. We will consider a directed graph $\mathit{Tree}(\Gamma)$, defined by

(1) The vertex set of $\mathit{Tree}(\Gamma)$ is $\mathit{Prefix}(\Gamma)$.
(2) For $u, v \in \mathit{Prefix}(\Gamma)$, $a \in \Sigma$, if $ua = v$, then (u, v) is an edge of $\mathit{Tree}(\Gamma)$.
(3) Nothing else is an edge of $\mathit{Tree}(\Gamma)$.

It is clear that $\mathit{Tree}(\Gamma)$ is a tree whose edges are directed from the root λ towards the leaves. In the context of pattern matching [AC 75], $\mathit{Tree}(\Gamma)$ has been called the *goto function*.

We now define the graph $R(\Gamma)$. The vertex set of $R(\Gamma)$ is $\mathit{Prefix}(\Gamma)$. There are two kinds

of edges, a set

$$E_{\text{Reach}} = \{ (u,v) \mid uc = v,\ u \neq \lambda,\ c \in \Gamma \}$$

whose members we will call *reach edges*, and the set

$$E_{\text{Divisor}} = \{ (u,v) \mid c = uv,\ u \neq \lambda,\ v \neq \lambda,\ c \in \Gamma \}$$

whose members we will call *divisor edges*. We will show how to construct this graph using the Aho-Corasick pattern matching algorithm [AC 75].

In $R(\Gamma)$, we also distinguish the subset Γ of the vertices and the subset

$$S(\Gamma) = \{ c \in \Gamma \mid c < c',\ c' \in \Gamma \}$$

consisting of those words in Γ which are prefix of some other word(s) in Γ. In Section 3.2 below we will demonstrate that a set Γ is not uniquely decipherable if, and only if, in $R(\Gamma)$ there is a nontrivial path from a vertex in $S(\Gamma)$ to a vertex in Γ. Furthermore, we will show that properties such as finite decipherability and finite synchronizability are manifest as other properties of this graph.

3.2 Testing Unique Decipherability

In the following, $\Gamma = \{c_1, ..., c_n\}$, and $L = |c_1| + ... + |c_n|$. We assume that the reader is familiar with the pattern matching algorithm of [AC 75]. We give an algorithm for testing whether Γ is uniquely decipherable, but in this paper we are primarily interested in the first three steps which construct the graph $R(\Gamma)$.

Algorithm UD

1. Construct a pattern matching machine M with Γ the set of patterns.
2. Construct the edge set E_{Reach}.
3. Construct the edge set E_{Divisor}.
4. Search $R(\Gamma)$ breadth-first to locate a path from $S(\Gamma)$ to Γ. If such a path exists, then Γ is not uniquely decipherable; if no such path exists, then Γ is uniquely decipherable.

Step (1) is implemented using the algorithm of [AC 75]. A routine modification enables it to locate all matches in a subject, rather than the first match only. As shown in [AC 75], Step (1) can be implemented in $O(L)$ steps. Note that M contains $Tree(\Gamma)$ in its description.

Steps (2) and (3) will have to perform the following operations on $Tree(\Gamma)$:

(a) Given a vertex v of $Tree(\Gamma)$, find the length $|v|$ of v.
(b) Given a vertex v of $Tree(\Gamma)$, find the vertex u of a prefix u of v of prescribed length.

It is clear that both operations can be implemented in constant time assuming a preprocessing step creating index structures requiring $O(L)$ steps. Briefly, with every leaf v of $Tree(\Gamma)$ is associated a vector V such that $V[i]$ points to the prefix of length i of v. Each interior vertex u uses the vector associated with an arbitrarily selected leaf in the subtree rooted at u.

Consider the determination of E_{Reach} in Step (2). Here we run M on $Tree(\Gamma)$. Each time we have a match of the set $\{d_1, ..., d_k\} \subset \Gamma$ at vertex u, we add the arcs (u_i, u), $1 \leq i \leq k$, where u_i is the prefix of length $|u| - |d_i|$ of u.

Note that a traversal of $Tree(\Gamma)$ for matching purposes requires at most $O(L)$ steps, since the sum of the lengths of all root to leaf paths is bounded by L. Therefore, Step (2) is $O(L+q)$, where q is the number of edges in E_{Reach}. It is easy to see that $|E_{\text{Reach}}| \leq n\,L$, hence Step (2) requires $O(n\,L)$ steps.

Recall from [AC 75] the notation $f(u) = v$ which means that v is a maximal proper suffix of u which is in $Prefix(\Gamma)$, and recall that M contains the graph of f for all strings in $Prefix(\Gamma)$. Consider $c \in \Gamma$, and assume that $f(c) = y_1$, $f(y_1) = y_2$, ..., $f(y_k) = \lambda$, where $y_k \neq \lambda$. Note that $k < |c|$. We add the arcs (u_i, y_i) to E_{Divisor}, $1 \leq i \leq k$, where u_i is the prefix of c of length $|c| - |y_i|$. Tracing f from c, this can be implemented in $O(|c|)$ steps, hence Step (3) requires $O(L)$ steps in all.

Step (4) is a standard breadth-first search of $R(\Gamma)$. We have a "current" vertex set and all arcs originating in the set are explored. Newly reached vertices become the next current vertex set. Exploration ends when a vertex in Γ is reached or all edges have been examined. If a vertex in Γ is reached, then Γ is not a uniquely decipherable, otherwise it is. The initial current vertex set is $S(\Gamma)$ which is easily identified in Step (1). Clearly the time required for Step (4) is $O(|E_{\text{Reach}}| + |E_{\text{Divisor}}|)$ which is $O(n\,L)$. Correctness of the algorithm will be shown in Section 4. In summary, we have:

Theorem 3.1
Let $\Gamma = \{c_1, ..., c_n\}$ be a set of words in Σ^+, Σ a fixed alphabet, and let $L = |c_1| + ... + |c_n|$. Then in $O(n\,L)$ steps we can construct $R(\Gamma)$ and test whether Γ is uniquely decipherable.

4. Code Properties and $R(\Gamma)$

A number of properties of Γ correspond to structural properties of $R(\Gamma)$. We explain this correspondence now. In particular, we show how L-sequences correspond to paths in $R(\Gamma)$, and from this fact will follow the correctness of Algorithm UD of Section 3.

Proposition 4.1
There is a path in $R(\Gamma)$ from a vertex $s_0 \neq \lambda$ to a vertex q with $qs_n \in \Gamma$ iff $\sigma = \langle s_0, c_1, ..., c_n, s_n \rangle$ is an L-sequence for Γ.

Proof Recall the definition of L-sequence from Section 2. We will show how to find a path in $R(\Gamma)$ from the L-sequence σ. A complete proof of the Proposition will be apparent from this construction.

Since $c_1 = s_0 s_1$, s_0 is in $Prefix(\Gamma)$, and s_1 is a code word suffix. We let $q_0 = s_0$ and observe that $q_0 s_1 \in \Gamma$.

If $c_2 = s_1 s_2$, then s_1 is also in $Prefix(\Gamma)$, and there is an edge (s_0, s_1) in E_{Divisor}. In this case, let $q_1 = s_1$. If $s_1 = c_2 s_2$, then $s_0 c_2$ is in $Prefix(\Gamma)$, and there is an edge $(s_0, s_0 c_2)$ in E_{Reach}. In this case, let $q_1 = s_0 c_2$. In either case, observe that $q_1 s_2 \in \Gamma$. By induction, therefore, there is a path in $R(\Gamma)$ corresponding to σ, from $q_0 = s_0$ to a vertex q_{n-1} and $q_{n-1} s_n \in \Gamma$. ∎

Corollary 4.2
A set $\Gamma \subset \Sigma^+$ is not uniquely decipherable iff there is a nontrivial path in $R(\Gamma)$ from a vertex in $S(\Gamma)$ to a vertex in Γ.

Proof Clearly Γ is not uniquely decipherable iff there is a dividing sequence. Let $\sigma = (s_0, c_1, ..., c_n, s_n)$ be a dividing sequence. By Proposition 4.1, there is a path in $R(\Gamma)$ beginning at s_0 which is in $S(\Gamma)$, and ending at a vertex q_{n-1} where $q_{n-1}s_n \in \Gamma$. Since $s_n \in \Gamma$, there is a divisor edge from q_{n-1} to s_n in $R(\Gamma)$. Hence the path ends at a vertex in Γ and is nontrivial. ∎

The properties of finite decipherability delay as well as of synchronizability of Γ can be expressed in terms of L-sequences. Specifically, Γ is not finitely decipherable (synchronizable) iff there is a loop sequence (cyclic sequence) [Cap 79]. Because of the strong correspondence of L-sequences with paths in $R(\Gamma)$, we readily obtain the following

Corollary 4.3
A uniquely decipherable code $\Gamma \subset \Sigma^+$ is not finitely decipherable iff there is a cycle in $R(\Gamma)$ which is reachable from a vertex in $S(\Gamma)$. Moreover, Γ is synchronizable iff $R(\Gamma)$ is acyclic.

In Section 8 we explore in detail how to determine the decipherability and synchronization delays of a code given the graph $R(\Gamma)$.

5. The Minimum Free Subsemigroup

We give an $O(n^2L+L^2)$ algorithm which finds the basis of the minimum free subsemigroup of Σ^+ containing a finite set $\Gamma \subset \Sigma^+$. Here n is the cardinality of Γ and L is the sum of the lengths of the code words in Γ. Of course, if Γ is a uniquely decipherable code, then the basis of the minimum free subsemigroup is Γ. The algorithm is developed from a simpler, $O(n\,L^2)$ implementation of Algorithm A, in which Algorithm UD is used to find the required dividing sequences. The faster version is obtained from the simpler algorithm by grouping several iterations of Steps 2–5 in Algorithm A, such that the new set Γ', derived from Γ, has cardinality at most $|\Gamma| - 1$.

We wish to exploit the relationship between L-sequences and paths in $R(\Gamma)$ which we touched upon above in Section 4. Specifically, from the proof of Proposition 4.1 we readily obtain the following:

Proposition 5.1
Let $\sigma = (s_0, c_1, c_2, ...)$ be an L-sequence for Γ. If $c_2 = s_1 s_2$, then there is a divisor edge (s_0, s_1) in $R(\Gamma)$. If $s_1 = c_2 s_2$, then there is a reach edge $(s_0, s_0 c_2)$ in $R(\Gamma)$.

The crucial operation of Algorithm A is splitting the code word c_1 of a dividing sequence. We now state how this operation is done using the graph $R(\Gamma)$, referring to the version of Algorithm A which determines the basis of the minimum free subsemigroup as Algorithm A^{FS}. Note that this is the simple, $O(n\,L^2)$ implementation.

Corollary 5.2
Let Γ be a finite subset of Σ^+, and assume there is a path in $R(\Gamma)$ from $c \in S(\Gamma)$ to $c' \in \Gamma$ containing at least one divisor edge, corresponding to the dividing sequence σ. Let (u, v) be the first divisor edge of the path, and let w be defined by $cw = uv$ Then the set

$$\Gamma' := \Gamma - \{uv\} \cup \{w\}$$

is the set U of Algorithm A^{FS} after executing Step 4 for the first time with σ the chosen dividing

sequence.

Proof Let $\sigma = (c, c_1, ...)$ be a dividing sequence. Then there is a corresponding path from the vertex $c \in S(\Gamma)$ to a vertex $c' \in \Gamma$. Without loss of generality, this path terminates with a divisor edge. Let $c_1 = cs_1$. Then $c_1 = uv$ and $w = s_1$. ∎

Consider the following implementation of Algorithm A:

1. Construct $R(\Gamma)$ and locate a path from $S(\Gamma)$ to Γ. If none exists, stop: Γ is now the desired set of generators for the minimum free subsemigroup.
2. Locate the first divisor edge (u, v) on the path. ¿From it, determine uv and w. Remove uv from Γ and add w to Γ. Return to Step 1.

Clearly, this is a correct implementation of Algorithm A. As for the timing, note that the new set derived in Step 2 has cardinality not greater than the original set. Moreover, the longer word $uv = cw$ is replaced with the shorter word w, hence Step 2 cannot be executed more than L times, where L is the sum of the lengths of the words in the original set Γ. Since Step 1 requires $O(n\,L)$ steps, whereas Step 2 requires only $O(L)$ steps, it follows that the algorithm requires at most $O(n\,L^2)$ steps.

If $w \neq v$, then it is clear that w is split eventually into $w = c_2...c_k v$, where the c_i are in Γ. Hence one may, equivalently, replace uv with v in Step 2. The advantage here is that v and its prefixes already exist in the graph $R(\Gamma)$, hence no new vertices need to be added to the underlying $Tree(\Gamma)$. While this simplifies the algorithm somewhat, it does not alter its time complexity.

Another possible modification is to split *all* words c occurring in *some* dividing sequence of Γ. This involves locating all divisor edges (u, v) in $R(\Gamma)$ which are on some path from $S(\Gamma)$ to Γ. Then the words uv are removed from Γ, and the words v are added. By processing these edges by decreasing length uv one can assure that a word v which already is in Γ and is split through some other divisor edge (u', v'), $v = u'v'$, is removed from Γ right away. The resulting algorithm constructs the generating sets produced by the algorithm proposed by Berstel et al. in [Ber 79].

Curiously, we can only give an $O(L^3)$ time bound for this second version of the algorithm, since the intermediate sets Γ' produced may be of greater cardinality than the original set Γ, and also the sum of the word lengths may temporarily increase. Of course, an $O(L^3)$ bound is contrary to intuition. Yet this size problem, stemming from splitting the same code word in more than one way, would also impair a similar attempt at speeding up the $O(n^2L + L^2)$ algorithm which we describe next.

In the faster $O(n^2L + L^2)$ algorithm we exploit the following idea: After splitting the code word c_1 in the dividing sequence $\sigma = (c, c_1, c_2, ...)$, there is a new dividing sequence σ' for the set Γ' which is obtained by substituting $c_0 s_1$ for every occurrence of c_1 in σ, followed by dropping the leading occurrences of c_0. The order of words in the resulting sequence might have to be changed somewhat to account for the way in which the shorter words c_0 and s_1 overlap with each other and with the other words in the sequence. The corresponding path may be constructed without reconstructing the graph $R(\Gamma')$; rather, we modify $R(\Gamma)$ accordingly. After this has been repeated for these "dividing sequence tails", we eventually obtain a new set $\tilde{\Gamma}$ whose cardinality is smaller than that of Γ. We illustrate the idea by an example:

Example

Let $\sigma = (c_0, c_1, c_2, c_1, c_3, c_4)$ be a dividing sequence as shown in Figure 5.1. Here $c_1 = c_0 s_1$, $s_1 = c_2 s_2$, $c_1 = s_2 s_3$, and $s_3 = c_3 c_4$. We split $c_1 = c_0 s_1$ and replace c_1 with s_1 in Γ, thus obtaining Γ'.

Substituting $c_0 s_1$ for c_1 in the sequence and dropping the leading pair of c_0, we obtain the new dividing sequence $(c_2, s_1, c_0, c_3, s_1, c_4)$ shown in Figure 5.2. Note that the order of code words has been slightly altered. Here s_1 can be split $s_1 = c_2 s_2$, and so on.

After several of these split and modify steps, we eventually obtain the dividing sequence of Figure 5.3. Here c_4 is split, but since both u and s_2 are in the current set Γ', the next set $\bar{\Gamma}$ obtained from this split has cardinality at most $|\Gamma| - 1$. ∎

After each such group of split and modify steps, we construct the graph $R(\bar{\Gamma})$ and repeat this operation for a new dividing sequence until we obtain a uniquely decipherable code, i.e., until no more dividing sequences exist. Note that in each split and modify step a longer word is replaced with a shorter one.

We now explain how the split and modify step is expressed as operations on the graph $R(\Gamma)$. Assume that we have fixed a path from $S(\Gamma)$ to Γ such that the path does not have cycles and the last path edge is a divisor edge.

Step A (Splitting c_1)
Let (u, v) be the first divisor edge on the path, and assume that $c_1 = uv$. Define s_1 by $c_0 s_1 = c_1$. Delete c_1 from Γ and add s_1 to Γ, thereby obtaining Γ'. This may involve creating a new path from the root of $Tree(\Gamma)$ for s_1. Adjust the path in $R(\Gamma)$ as specified by Operation B.

Step B1 (Adjusting the first occurrence of c_1)
Delete the edge (u, v) from the path. If the edge was the first one, then $v = s_1$ and the path now begins at s_1. Otherwise, let $(p_1, p_2), (p_2, p_3), ..., (p_{k-1}, p_k)$ be the leading reach edges of the path, i.e., $p_1 = c_0$ and $p_k = u$. Define c_i by $p_i = p_{i-1} c_i$, $2 \leq i \leq k$. Note that $c_i \in \Gamma$ and that $c_2 c_3 ... c_k < s_1$. Remove the edges (p_{i-1}, p_i) from the path and add in their place the reach edges $(c_2, c_2 c_3), ..., (c_2 ... c_{k-1}, c_2 ... c_k)$. Add the divisor edge $(c_2 ... c_k, v)$. In this case, the new path begins at c_2. Note that $c_2 \in S(\Gamma')$. If any of the new edges terminate at a vertex in Γ, then delete all subsequent path edges.

Step B2 (Adjusting a subsequent reach edge belonging to c_1)
Let (p, q) be a reach edge on the path with $q = pc_1$. Replace the edge with the two reach edges (p, pc_0) and (pc_0, q). Note that $q = pc_0 s_1$. If $pc_0 \in \Gamma$, then remove the edge (pc_0, q) and all subsequent path edges.

Step B3 (Adjusting a subsequent divisor edge belonging to c_1)
Let (p, q) be a divisor edge on the path where $c_i = pq$. Since the path was without cycles, we have $p < c_0$ or $c_0 < p$.
If $p < c_0$, let r be the prefix of q of length $|q| - |s_1|$, i.e., $q = rs_1$. Replace the edge (p, q) with the divisor edge (p, r), belonging to c_0, and the reach edge (r, q). The change is shown in Figure 5.4.
If $c_0 < p$, there are two cases:

(1) The edge (p, q) is preceded by the reach edges $(p_k, p_{k-1}), ..., (p_2, p_1)$, where $p_1 = p$, $p_k < c_0$, and $c_0 < p_{k-1}$. See Figure 5.5.

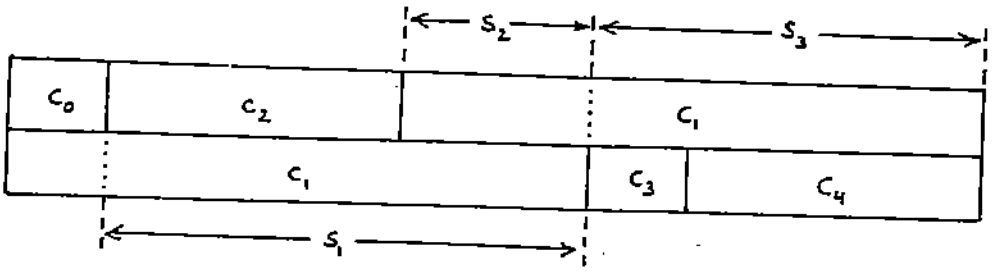

Figure 5.1

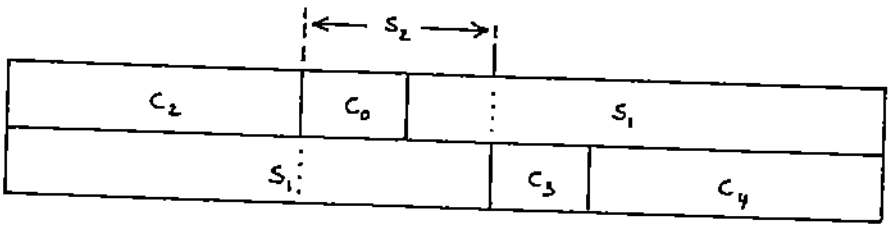

Figure 5.2

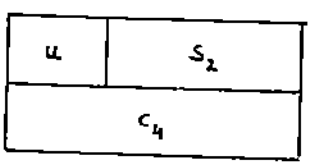

Figure 5.3

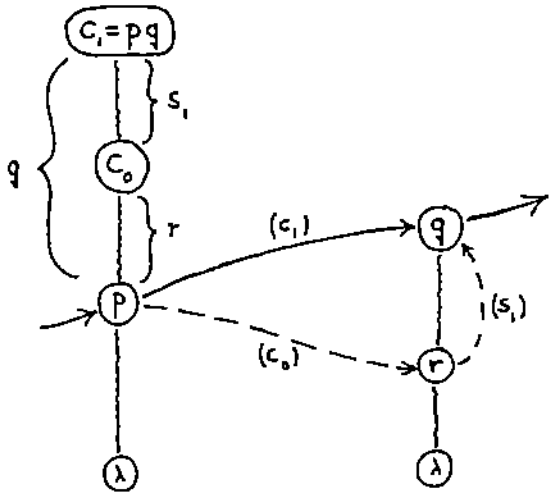

Figure 5.4
Pathmodification, Step B3, $p < c_0$

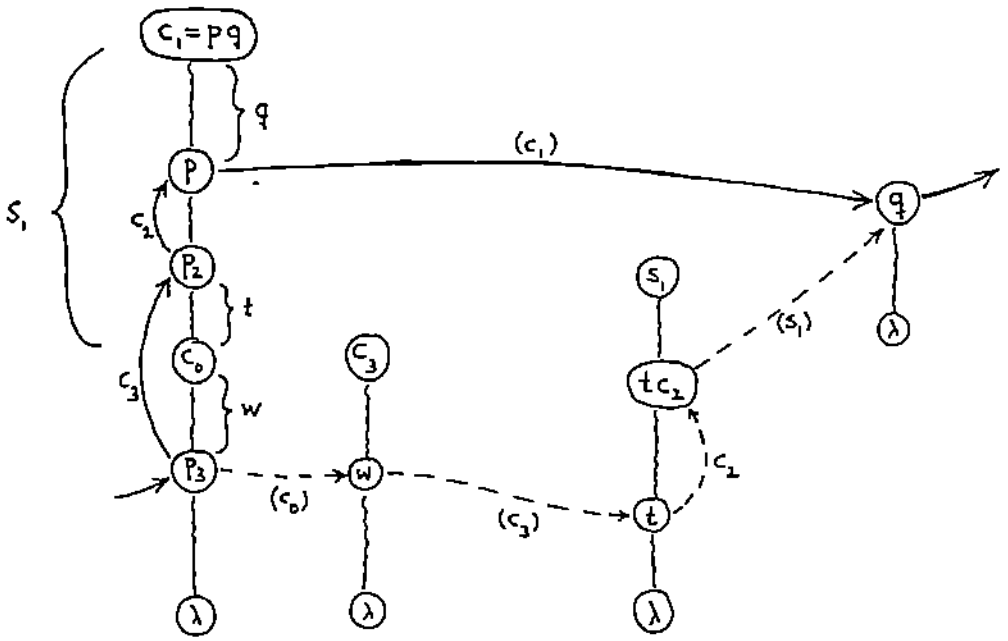

Figure 5.5
Pathmodification, Step B3, $c_0 < p$
Case (1), $k = 3$

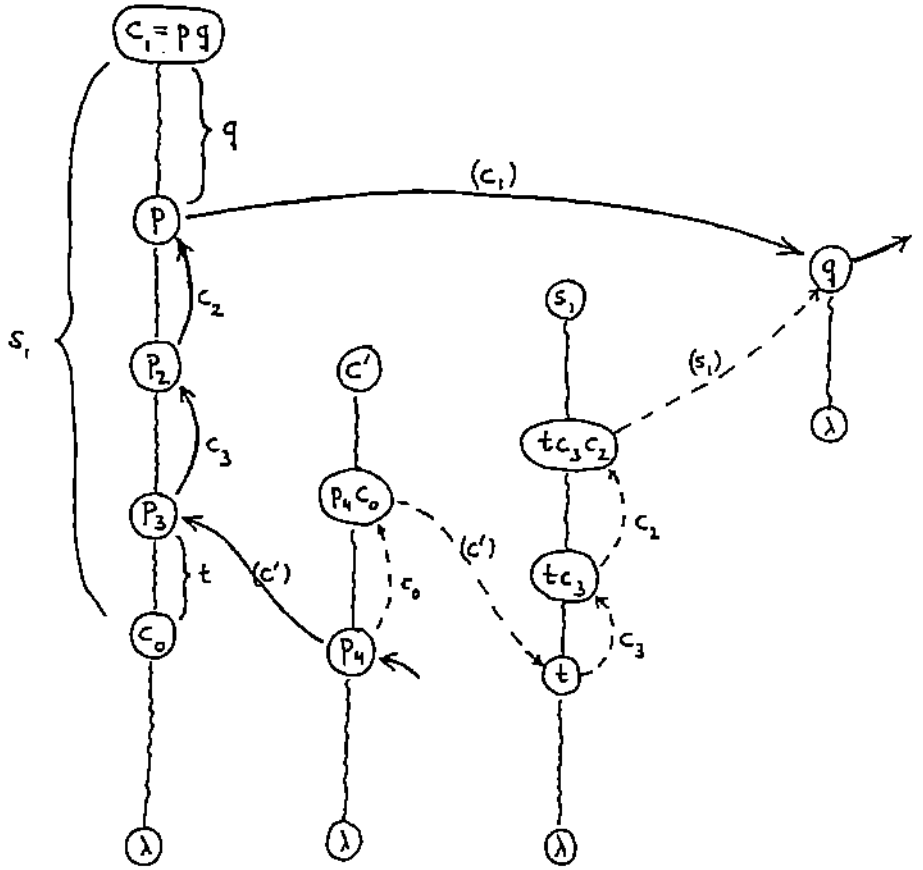

Figure 5.6
Pathmodification, Step B3, $c_o < p$
Case (2), $k = 4$

(2) The edge (p,q) is preceded by zero or more reach edges $(p_{k-1},p_{k-2}),...,(p_2,p_1)$, where $p_1 = p$, and by the divisor edge (p_k,p_{k-1}). Moreover, $c_0 < p_{k-1}$. See also Figure 5.6.

In Case (1), define c_i by $p_{i-1} = p_i c_i$, $2 \le i \le k$. Replace the reach edges (p_i,p_{i-1}) with the divisor edges (p_k,w) and (w,t), where $p_k w = c_0$ and $wt = c_k$, followed by the reach edges $(t,tc_{k-1}),(tc_{k-1},tc_{k-1}c_{k-2}),...,(tc_{k-1}...c_3,tc_{k-1}...c_3c_2)$. Also, replace the divisor edge (p,q) with the divisor edge $(tc_{k-1}...c_2,q)$. Note that $tc_{k-1}...c_2q = s_1$. The transformation is shown in Figure 5.5.
In Case (2), define t from $p_{k-1} = c_0 t$ and let $c' = p_k p_{k-1}$. If $k > 2$, define c_i by $p_{i-1} = p_i c_i$, $2 \le i < k$. Replace the divisor edge (p_k,p_{k-1}) with the reach edge $(p_k,p_k c_0)$ followed by the divisor edge $(p_k c_0,t)$. Replace the reach edges (p_i,p_{i-1}), $2 \le i < k$, with the reach edges $(t,tc_{k-1}),...,(tc_{k-1}...c_3,tc_{k-1}...c_2)$, and replace the divisor edge (p,q) with the divisor edge $(tc_{k-1}...c_2,q)$. Note that $tc_{k-1}...c_2q = s_1$. The transformation is shown in Figure 5.6.

In either case, if one of the new vertices on the path is in Γ, then delete all subsequent path edges.

Note that initially the path fixed in $R(\Gamma)$ does not contain any cycles. In the path transformation, however, cycles may be created. It is therefore necessary to make a pass over the transformed path deleting cycles, to insure that the maximum path length is bounded by L, the sum of the lengths of words in Γ. This results in the following

Algorithm FS

1. Initialize C to $\Gamma \subset \Sigma^+$.
2. Construct the graph $R(C)$ using Algorithm UD.
3. By breadth-first search, determine a path π in $R(C)$ from $S(C)$ to C, such that π is acyclic and ends with a divisor edge.
4. If no such path π exists, then stop; C is the basis of the minimum free subsemigroup of Σ^+ which contains Γ.
5. While π is not empty, do Steps 6–8. Thereafter, return to Step 2.
6. Perform Step A on the path π as described above.
7. Transform the path by performing the adjustments of Steps B1–B3 above.
8. Eliminate all loops from the transformed path.

Theorem 5.3
Algorithm FS correctly determines the basis of the minimum free subsemigroup containing Γ in $O(n^2L + L^2)$ steps.

Proof Correctness follows from Corollary 5.2 and the fact that the transformed path is a path in $R(C_1)$, where $C_1 = C - \{c_1\} \cup \{s_1\}$.

For the timing, observe first that Steps 2–4 are executed at most n times, since the loop of Steps 5–8 decreases the cardinality of C. Hence the total amount of work done by Steps 2–4 is at most $O(n^2L)$ steps.

Now consider Step 6. Here c_1 is split into c_0 and s_1, and the sum L of the lengths of words in C is decreased by $|c_0|$. It follows that Steps 6–8 can be executed at most L times ever. It remains to show that a single execution of Steps 6–8 requires no more than $O(L)$ steps.

In executing Steps 6–8, Step 7 dominates. Now it is easy to see that if a dividing sequence

contains m words, then the corresponding path has $m-2$ edges. Since path transformation corresponds to the substitution of two words for each occurrence of c_1, it follows that the transformed path contains at most twice as many edges as the original path. As loops are eliminated from the transformed path, each path considered in Step 7 has $O(L)$ edges maximum. Moreover, given the index structures of Algorithm UD, it is clear that the number of steps required to transform the path is proportional to the number of edges on the path. Consequently, a single execution of Steps 6–8 requires $O(L)$ steps, hence the total work done in Steps 5–8 is at most $O(L^2)$. ∎

Instead of eliminating all loops on the path, one may eliminate all edges following the first vertex u which repeats on the path. This is correct since u eventually becomes a word in C, when encountered for the first time by the splitting operation, hence it anticipates, in essence, the elimination of subsequent path edges due to an edge terminating at a vertex in C. Strictly speaking, one may lose the property that the path ends with a divisor edge, which we have assumed throughout in order to simplify the presentation. It is not difficult, however, to augment the transformations either eliminating this requirement, or replacing a terminal reach edge with the corresponding divisor edge.

6. The Minimum Weakly Prefix and Very Pure Subsemigroups

Algorithm A can be specialized to find the basis of the minimum subsemigroups containing Γ which are weakly prefix or very pure. The corresponding implementations differ from Algorithm FS in that different types of paths in $R(\Gamma)$ are considered, but are otherwise identical.

Recall Steps A and B used in Algorithm FS. Clearly these operations can be applied to paths corresponding to loop sequences and cyclic sequences. Here a path corresponding to a loop sequence is a path in $R(\Gamma)$ which begins at a vertex $c_0 \in S(\Gamma)$ and contains some vertex u both as an intermediate as well as the final path vertex. We will call such a path a *loop path*, and call u the *lead vertex*. Without loss of generality, we may assume that u is the only vertex occurring twice.

Consider the algorithm for finding the basis for the minimum weakly prefix subsemigroup containing Γ. Here we must consider all loop and dividing sequences. Since in either case the path begins at a vertex in $S(\Gamma)$, it follows that all intermediate sets Γ' have cardinality no greater than Γ. Moreover, when the path is completely split, the resulting set $\tilde{\Gamma}$ has cardinality at most $|\Gamma|-1$. We give the algorithm below as Algorithm WP. Note the similarity to Algorithm FS.

Algorithm WP

1. Initialize C to $\Gamma \subset \Sigma^+$.
2. Construct the graph $R(C)$ using Algorithm UD.
3. Determine a path π in $R(C)$ from $S(C)$ to C, or from $S(C)$ to a cycle of $R(C)$. The path should not contain unneccesary cycles.
4. If no such path π exists, then stop; C is the basis of the minimum weakly prefix subsemigroup of Σ^+ which contains Γ.
5. While π is not empty, do Steps 6–8. Thereafter, return to Step 2.
6. Perform Step A on the path π as described above.

7. Transform the path by performing the adjustments of Steps B1–B3 above.
8. Eliminate all edges following the first repeated vertex u from the transformed path.

Theorem 5.3
Algorithm WP correctly determines the basis of the minimum weakly prefix subsemigroup of Σ^+ containing Γ and requires at most $O(n^2L + L^2)$ steps.

The theorem is proved like Theorem 5.3.

Now consider the algorithm for determining the basis of the minimum very pure subsemigroup containing Γ. Here we must consider all loop sequences, cyclic sequences, and dividing sequences. The algorithm to be given requires $O(n\,L^2)$ steps. This inferior time bound is due to the fact that splitting a cyclic sequence need not result in a smaller set $\bar{\Gamma}$, since in the cyclic sequence $(s_0, c_1, ..., c_m, s_0)$ the word s_0 need not be in Γ. Hence splitting c_1 may result in a new set Γ' of cardinality $|\Gamma| + 1$. Here splitting the sequence tail completely eventually reduces the cardinality to $|\Gamma|$ or less. Note that the initial splitting step generates a set which is not a uniquely decipherable code.

Because of the initial increase in cardinality, it is crucial to split the tail sequence completely before considering other cyclic sequences. Otherwise we cannot guarantee that all intermediate sets have cardinality $O(|\Gamma|)$. However, as s_0 is added to Γ', we may equivalently split a suitable dividing sequence instead. The resulting algorithm is Algorithm VP:

Algorithm VP

1. Initialize C to $\Gamma \subset \Sigma^+$.
2. Construct the graph $R(C)$ using Algorithm UD.
3. If $R(C)$ contains no path from $S(C)$ to C and is acyclic, then stop: C is the basis of the minimum very pure subsemigroup of Σ^+ which contains Γ.
4. If there is a cycle π in $R(C)$, let (u, v) be a divisor edge on the cycle. Set $C' := C - \{uv\} \cup \{u, v\}$.
5. Using Algorithm WP, find the basis U of the minimum weakly prefix subsemigroup containing C'.
6. Set $C := U$ and goto Step 2.

Theorem 6.2
Algorithm VP is correct and requires at most $O(n\,L^2)$ steps.

Proof Correctness is evident. For the timing, recall that in Step 4 we have $|C'| \leq |C| + 1$ and $L' \leq L$, where L and L' are the sum of the lengths of words in C and C', respectively. Moreover, C' is not uniquely decipherable, hence $|U| \leq |C|$, and $L'' < L$, where L'' is the sum of the lengths of words in U. Hence Step 5 is executed at most L times. By the proof of Theorems 5.3 and 6.1, therefore, all invokations of Algorithm WP require a total of $O(n\,L^2)$ steps. The time bound now follows. ∎

7. The Minimum Left Unitary Subsemigroup

Algorithm B determines the minimum left unitary subsemigroup of Σ^+ which contains Γ.

It is easily implemented by repeatedly constructing *Tree*(C), followed by splitting each root to leaf path $p = u_1 u_2 \cdots u_k$ in accordance with which prefixes are in C. That is, p is split into the set u_1, u_2, ..., u_k, where u_1, $u_1 u_2$, ..., $u_1 u_2 \cdots u_k \in C$. The set is added to the next version of C. The algorithm terminates as soon as *Tree*(Γ) has no interior vertices in C. Evidently, the time bound for this implementation is $O(L^2)$, since at each stage the length of the words in C is decreased by at least one. Of course, if Γ is already a prefix code, then Γ is returned.

8. Determination of Synchronizability and Decipherability Delays

Corollary 4.3 implies an $O(n\,L)$ test whether a uniquely decipherable set Γ is synchronizable and whether it has finite decipherability delay. We now develop an algorithm for determining the decipherability delay of a finitely decipherable code, and the synchronization delay of a synchronizable code. It turns out, that both problems can be solved by essentially the same algorithm. So, we develop the algorithm for determining the synchronization delay first, followed by explaining how to modify it to determine the decipherability delay of a given code.

Recall that a synchronizable code Γ has the synchronization delay s if s contiguous code words in a message fragment suffice to detect a correct code word boundary in the fragment.

Example
Let $\Gamma = \{a, baa\}$. Clearly Γ is a synchronizable code. In the fragment aa of a possibly longer message we can correctly announce a word boundary following the second a, but in the fragment a we could not. Therefore, Γ has the synchronization delay 2.

We want to define a valuation of L-sequences which relates the synchronization delay of a code to path properties in the graph $R(\Gamma)$. First, we associate with each L-sequence a pair of strings in Σ^+, as already sketched in Section 2.1:

Definition
Let σ, be an L-sequence for Γ. The strings $x_\sigma, y_\sigma \in \Gamma^*$ *associated* with σ are defined by the following rules:

(1) If $\sigma = \langle s_0, c_1, s_1 \rangle$, then $x_\sigma = c_1$ and $y_\sigma = \lambda$.
(2) Let $\sigma = \langle s_0, c_1, ..., c_m, s_m \rangle$, for $m > 1$, where $\mu = \langle s_0, c_1, ..., c_{m-1}, s_{m-1} \rangle$. If $x_\mu s_{m-1} = s_0 y_\mu$, then $x_\sigma = x_\mu c_m$ and $y_\sigma = y_\mu$. Moreover, if $x_\mu = s_0 y_\mu s_{m-1}$, then $x_\sigma = x_\mu$ and $y_\sigma = y_\mu c_m$.

Lemma 8.1
Let x_σ, y_σ be the strings associated with the L-sequence σ. Then $x_\sigma s_m = s_0 y_\sigma$ or $x_\sigma = s_0 y_\sigma s_m$.

The lemma is easily proved by induction on the length of σ. Note that the code words $c_1, ..., c_m$ specify how to decipher the strings x_σ and y_σ.

Next, we associate with the L-sequence σ a pair of nonnegative integers $L(\sigma) = (i, j)$, where i and j are the number of code words in the strings x_σ and y_σ, respectively. The definition of $L(\sigma)$ closely parallels the definition of the strings x_σ and y_σ:

Definition
Let σ, be an L-sequence for Γ. The *word length pair* of σ, denoted by $L(\sigma)$, is defined as follows:

(1) If $\sigma = \langle s_0, c_1, s_1 \rangle$, then $L(\sigma) = (1,0)$.
(2) Let $\sigma = \langle s_0, c_1, ..., c_m, s_m \rangle$, for $m > 1$, where $\mu = \langle s_0, c_1, ..., c_{m-1}, s_{m-1} \rangle$ and $L(\mu) = (r,s)$. Then if $x_\mu s_{m-1} = s_0 y_\mu$, then $L(\sigma) = (r+1, s)$. Moreover, if $x_\mu = s_0 y_\mu s_{m-1}$, then $L(\sigma) = (r, s+1)$.

Finally, we define a valuation $\|\sigma\|$ of σ, which is directly related to the synchronization delay.

Definition
Let σ be an L-sequence with $L(\sigma) = (p,q)$. If $x_\sigma = s_0 y_\sigma s_m$, then $\|\sigma\| = \max(p-1, q+1)$. If $x_\sigma s_m = s_0 y_\sigma$, then $\|\sigma\| = \max(p,q)$.

Lemma 8.2
Let Γ be a synchronizable code with synchronization delay s. Then, for all L-sequences σ, $s \geq \|\sigma\|$.

Proof Let σ be an L-sequence with $L(\sigma) = (p,q)$. If $x_\sigma s_m = s_0 y_\sigma$, define u by $u s_m = y_\sigma$. Then u has a prefix of $q-1$ consecutive code words and a suffix of $p-1$ consecutive code words which are incompatible decipherings of u. Hence $s > \max(p-1, q-1)$.

If $x_\sigma = s_0 y_\sigma s_m$, then y_σ consists of q contiguous code words and contains a factor of $p-2$ contiguous code words leading to an incompatible deciphering of y_σ. Hence $s > \max(p-2, q)$. ∎

Theorem 8.3
Let $s = \max(\{ \|\sigma\| \mid \sigma \text{ an L-sequence for } \Gamma \})$, where Γ is a synchronizable code. Then Γ has the synchronization delay s. If there are no L-sequences, then Γ has delay one.

Proof By Lemma 8.2, the synchronization delay of Γ can be no smaller than s. Hence it suffices to show that in any message fragment containing some $f \in \Gamma^s$ we may correctly determine a code word boundary.

Assume that $f \in \Gamma^s$ cannot be correctly partitioned on a code word boundary. Then there must be an L-sequence σ such that either $L(\sigma) = (p,s)$ and $x_\sigma = s_0 y_\sigma s_m$, or $L(\sigma) = (s+1, q)$ and $x_\sigma s_m = s_0 y_\sigma$. All other cases are equivalent to one of these two. In either case we have $\|\sigma\| > s$, contradicting the definition of s. Hence all $f \in \Gamma^s$ can be correctly partitioned. ∎

We relate the valuation of σ to a metric defined on the path associated with σ in $R(\Gamma)$. The connection is established by determining $L(\sigma)$ from the edges in the path, but here we need to remember whether y_σ is a proper substring of x_σ:

Definition
Let p be a path in $R(\Gamma)$. The *modified length pair*, $L'(p)$ of p, is defined as follows:
(1) $L'(\langle p_0 \rangle) = (0,1;1)$.
(2) If $(p_{m-1}, p_m) \in E_{\text{Reach}}$, $m > 1$, and $L'(\langle p_0, ..., p_{m-1} \rangle) = (i,j;k)$, then $L'(p) = (i+1, j; k)$.
(3) If $(p_{m-1}, p_m) \in E_{\text{Divisor}}$, $m > 1$, and $L'(\langle p_0, ..., p_{m-1} \rangle) = (i,j;k)$, then $L'(p) = (j, i+1; \bar{k})$, where $\bar{1} = 0$ and $\bar{0} = 1$

Lemma 8.4
Let $\sigma = \langle s_0, ..., s_m \rangle$ be an L-sequence for Γ, p the path associated with σ as in Proposition 4.1. If $L'(p) = (i,j;k)$, then $\|\sigma\| = \max(i+k, j-k)$.

Proof By induction, it is easy to see that $L(\sigma) = (j,i)$ if $k = 1$, and $L(\sigma) = (i,j)$ if $k = 0$. Moreover, if $k = 0$, then $x_\sigma s_m = s_0 y_\sigma$, and if $k = 1$ then $x_\sigma = s_0 y_\sigma s_m$. The lemma now follows from the definition of $\|\sigma\|$. ∎

It is clear that for each path p in $R(\Gamma)$ we can compute $L'(p)$ in time proportional to the number of edges in p. By Theorem 8.3 and Lemma 8.4, we may determine the synchronization delay of Γ by evaluating $L'(p)$ for all paths in $R(\Gamma)$ and computing $\max(i+k, j-k)$, retaining the largest value obtained. This is of course impractical, as the number of paths in $R(\Gamma)$ may be exponential. We therefore seek to evaluate all paths simultaneously, retaining only those values $L'(p)$ which may contribute to the maximum value sought.

Specifically, consider a vertex u of $R(\Gamma)$ at which m paths p_t terminate, where $L'(p_t) = (i_t, j_t; k_t)$, $1 \le t \le m$. If $m \ge 2$, we select two values (not necessarily from distinct paths), $(i_1, j_1; k_1)$, and $(i_2, j_2; k_2)$, such that, for any value $(x, y; k)$ among the $L'(p_t)$, we have $x + k \le i_1 + k_1$ and $y - k \le j_2 - k_2$. These two values suffice to determine $\max(i+k, j-k)$, as is clear from the following, straightforward:

Lemma 8.5
Let p_1, p_2, p_3 be three paths in $R(\Gamma)$ ending at the vertex u, (u, v) an edge in $R(\Gamma)$, and let p_{t+3} be the path p_t extended by (u, v), for $t = 1, 2, 3$. For $1 \le t \le 6$, let $L'(p_t) = (i_t, j_t; k_t)$, and assume that $i_3 + k_3 \le i_1 + k_1$ and $j_3 - k_3 \le j_2 - k_2$. Then either $i_6 + k_6 \le i_4 + k_4$ and $j_6 - k_6 \le j_5 - k_5$, or $i_6 + k_6 \le i_5 + k_5$ and $j_6 - k_6 \le j_4 - k_4$.

The above derivation suggests the following approach to determining the synchronization delay of a code Γ:

Algorithm SD

1. Construct $R(\Gamma)$ and test whether Γ is a uniquely decipherable code, using Algorithm UD.
2. By topologically sorting $R(\Gamma)$ determine whether Γ is a synchronizable code. If Γ is not synchronizable, stop.
3. If $R(\Gamma)$ has no edges and $S(\Gamma)$ is empty, then output one and stop; otherwise, initialize $s := 1$.
4. Assign $(0, 1; 1)$ to each root of $R(\Gamma)$.
5. For each edge (u, v) in $R(\Gamma)$, in sorted order, perform Steps 6 and 7. Thereafter, output s and stop.
6. For each triple $(i, j; k)$ at u, if (u, v) is a reach edge, then place $(i+1, j; k)$ at vertex v, and if (u, v) is a divisor edge, then place $(j, i+1; k)$ at v.
7. If the number of triples at v is greater than 1, select up to two triples, $(i_1, j_1; k_1)$ and $(i_2, j_2; k_2)$, such that $i_1 + k_1$ and $j_2 - k_2$ are maximum, and discard the others. Update $s := \max(s, i_1 + k_1, j_2 - k_2)$.

Theorem 8.6
Algorithm SD is correct and requires at most $O(n\,L)$ steps, where n is the cardinality of Γ, and L is the sum of the lengths of the code words in Γ.

Proof Correctness is evident from Theorem 8.3 and Lemmas 8.4 and 8.5. For the timing, note that $R(\Gamma)$ has $O(L)$ vertices and $O(n\,L)$ edges. Steps 1–4, therefore, take $O(n\,L)$ steps. Since the edges are processed in sorted order, at most two triples $(i, j; k)$ are propagated along each edge, hence the work of Steps 5–7 is proportional to $O(n\,L)$. ∎

Now consider determining the decipherability delay of a finitely decipherable code Γ. Here we need only consider those L-sequences $\sigma = (s_0, \ldots, s_m)$ for which $s_0 \in \Gamma$. In analogy to Lemma 8.2, we have the obvious

Lemma 8.7
Let Γ be a code with finite decipherability delay d. If $\sigma = \langle s_0, ..., s_m \rangle$ is an L-sequence with $s_0 \in \Gamma$, then $d \geq \|\sigma\|$.

Theorem 8.8
Let $d = \max(\{ \|\sigma\| \mid \sigma = \langle s_0, ..., s_m \rangle$ an L-sequence for Γ with $s_0 \in \Gamma \})$, where Γ is a finitely decipherable code. Then Γ has decipherability delay d, unless the maximum is taken over the empty set, in which case the delay is zero.

Proof By Lemma 8.2, the decipherability delay of Γ can be no smaller than d. Hence it suffices to show that in any message prefix containing some $f \in \Gamma^{d+1}$ we may correctly determine the first code word.

Assume there exist $c \in \Gamma$, $u \in \Gamma^d$, and $w \in \Sigma^*$ such that $cuw \in \Gamma^*$ and $uw \notin \Gamma^*$. Then there is $c' \in \Gamma$ and $v \in \Gamma^*$ such that $c'v = cuw$ and $c \neq c'$. Since Γ is uniquely decipherable, we have $w \neq \lambda$.

If $c' < c$, then there is an L-sequence $\sigma = \langle c', c, ..., s_m \rangle$, where $x_\sigma s_m = c'y_\sigma$ and $x_\sigma = cu \in \Gamma^{d+1}$. If $c < c'$, then there is an L-sequence $\sigma = \langle c, c', ..., s_m \rangle$, where $x_\sigma = cy_\sigma s_m$ and $y_\sigma = u \in \Gamma^d$. In either case, $\|\sigma\| > d$, contradicting the definition of d. By Lemma 8.7, therefore, Γ has decipherability delay d. If there are no L-sequences for Γ, then Γ is a prefix code, hence has decipherability delay zero. ∎

In view of Theorem 8.8, a simple modification of Algorithm SD enables us to determine the decipherability delay of Γ. Instead of processing the entire graph $R(\Gamma)$, we process the subgraph of all edges reachable from a vertex in $S(\Gamma)$. Clearly paths in this subgraph correspond to L-sequences with $s_0 \in \Gamma$, and vice versa. Of course, if $S(\Gamma)$ is empty, then Γ is a prefix code and has decipherability delay zero. We thus obtain

Theorem 8.9
Let Γ be a finite subset of Σ^+. Then in $O(n\,L)$ steps we can decide whether Γ is finitely decipherable and determine its decipherability delay.

Note that according to our definitions the synchronization delay s of a code is no smaller than its decipherability delay d. This fact can be recovered from the algorithms by observing that for the determination of d only a subgraph of $R(\Gamma)$ is considered, whereas the entire graph is processed for the determination of s.

References

[AC 75] A. V. Aho and M. J. Corasick
"Efficient string matching: an aid to bibliographic search", *CACM 18:6* (1975) 333–343

[AG 84] A. Apostolico and R. Giancarlo
"Pattern matching machine implementation of a fast test for unique decipherability"
Inf. Proc. Letters, 18 (1984), 155–158

[Ber 79] J. Berstel, D. Perrin, J. F. Perrot, A. Restivo
"Sur le theoreme du defaut", *J. of Algebra 60* (1979) 169–180

[Blu 65] E. K. Blum
"Free subsemigroups of a free semigroup", *Mich. Math. J. 12* (1965) 179–182

[Cap 77] R. M. Capocelli
"On some free subsemigroups of a free semigroup", *Semigroup Forum 15* (1977) 125–135

[Cap 79] R. M. Capocelli
"A note on uniquely decipherable codes", *IEEE Trans. on Inf. Thy. IT-25* (1979) 90–94

[Cap 80] R. M. Capocelli
"On optimal factorization of free semigroups into free subsemigroups", *Semigroup Forum 19* (1980) 199–212

[Cap 80b] R. M. Capocelli
"On weakly prefix subsemigroups of a free semigroup", *Advances in Communications*, D. G. Lainiotis and N. S. Tzannes, eds., Reidel Publishing, Dordrecht 1980, 123–129

[CR 78] R. M. Capocelli and L. M. Ricciardi
"A heuristic approach to feature extraction for written languages", *Proc. 4th Intl. Conf. on Pattern Recognition*, Kyoto, Japan, 1978, 364–368

[DR 79] A. DeLuca and A. Restivo
"Synchronization and maximality for very pure subsemigroups of a free semigroup", *Springer Lect. Notes in Comp. Sci. 74* (1979) 363–371

[Eve 63] S. Even
"Tests for unique decipherability" *IEEE Trans. on Inf. Thy. IT-9* (1963) 109–112

[Eve 64] S. Even
"Tests for synchronizability of finite automata and variable length codes", *IEEE Trans. on Inf. Thy. IT-10* (1964) 185–189

[GGW 58] S. W. Golomb, B. Gordon, L. R. Welch
"Comma-free codes", *Can. J. of Math. 10* (1958) 202-209

[Hof 84] C. M. Hoffmann
"A note on unique decipherability", 11th *Intl. Symp. Math. Found. of Comp. Sci.*, (1984), Springer Lecture Notes in Comp. Sci.

[KMP 77] D. Knuth, J. Morris, V. Pratt
"Fast pattern matching in strings", *SIAM J. on Comp. 6:2* (1977) 323–350

[Lal 79] G. Lallement
"Semigroups and Combinatorial Applications", J. Wiley & Sons, New York (1979)

[Lev 62] V. I. Levenshtein
"Certain properties of code systems", *Sov. Phys. Doklady 6* (1962) 858–860

[Lev 64] V. I. Levenshtein

"Some properties of coding and self-adjusting automata for decoding messages", *Problemy Kybernetici 11* (1964) 63–121

[Mar 62] A. A. Markov
"Non-recurrent coding", *Problemy Kybernetici 8* (1962) 169–189

[McC 76] E. M. McCreight
"A space-economical suffix tree construction algorithm", *J. ACM 23:2* (1976) 262–272

[Res 74] A. Restivo
"On a question of McNaughton and Papert", *Inf. and Control 25* (1974) 93–101

[Ril 67] J. A. Riley
"The Sardinas-Patterson and Levenshtein theorems", *Inf. and Control 10* (1967) 120–136

[Rod 82] M. Rodeh
"A fast test for unique decipherability based on suffix trees", *IEEE Trans. on Inf. Thy. IT-28:4* (1982) 648–651;

[SP 53] A. A. Sardinas and G. W. Patterson
"A necessary and sufficient condition for the unique decomposition of coded messages", *IRE Intl. Conv. Rec. 8* (1953) 104–108

[Sch 73] B. M. Schein
"Semigroups whose every transitive representation by function is a representation by invertible functions", *Izv. Vyss. Ucebn. Zaved, Matem. 7* (1973) 112–121; in Russian.

[Spe 75] J. C. Spehner
"Quelques constructions et algorithmes relatifs aux sous-monoides d'un monoide libre", *Semigroup Forum 9* (1975) 334–353

[Til 72] B. Tilson
"The intersection of free submonoids of free monoids is free", *Semigroup Forum 4* (1972) 345–350