Deciding Multiset Decipherability

Tom Head† and Andreas Weber†


Abstract — An $O(n^2 L)$ time and $O((n + k)L)$ space algorithm is provided for deciding whether or not a finite set $C$ consisting of $n$ words having total length $L$, where all words are taken over a $k$-element alphabet, is a multiset decipherable code. The algorithm is based on a technique related to dominoes. At an early stage it decides in $O(nL)$ time and $O((n + k)L)$ space whether or not the set $C$ is uniquely decipherable.

Index Terms — Code, domino, multiset decipherability, unique decipherability.

1 Introduction

Let $A$ be a nonempty finite set which will be used as a code alphabet. Let $A^*$ be the set of all words of finite length over $A$, including the null word $\epsilon$. Each nonempty finite set $C$ of nonnull words in $A^*$ will be called a code over $A$. The words in $C$ will be called codewords. A message over $C$ is a word in $A^*$ that is a concatenation of codewords. The code $C$ is called uniquely decipherable, abbreviated UD, if each message can be factored in only one way into codewords. A. Lempel initiated the investigation of multiset decipherable codes in [10] where he observed that, for several communication purposes, a condition weaker than unique decipherability is adequate. He defined a code to be multiset decipherable, abbreviated MSD, if any two factorizations of the same message into codewords yield the same multiset of codewords.

By definition, every UD code is MSD. An MSD code is called proper if it is not UD. The code $C = \{0, 0111110, 10101, 1111\}$ over the binary alphabet $A = \{0, 1\}$, which was communicated to us by F. Guzman [8], is an example of a proper MSD code. In fact, because of the equation

$$0111110 \cdot 10101 \cdot 1111 \cdot 0 = 0 \cdot 1111 \cdot 10101 \cdot 0111110$$

*The results of this research were presented under the title “The finest homophonic partition and related code concepts” at the 19th Symposium on Mathematical Foundations of Computer Science, Košice, Slovakia, 1994. The work of T. Head was partially supported by NSF under Grant CCR-9201345.

†T. Head is with the Department of Mathematical Sciences, State University of New York at Binghamton, Binghamton, NY 13902-6000, U.S.A., E-mail: tjhead@bingsums.cc.binghamton.edu.

‡A. Weber is with Fachbereich Informatik, Johann Wolfgang Goethe-Universität, D–60054 Frankfurt am Main, Germany, E-mail: weber@informatik.uni-frankfurt.de. In winter 1994/95 he was with Institut für Informatik, Martin–Luther–Universität, Halle an der Saale, Germany.
there is a message over $C$ which can be factored in two ways into codewords. Therefore, $C$

is not UD. Note that each of the four codewords occurs exactly once in each of the given

factorizations. Thus both factorizations yield the same multiset of codewords. This means

that the above equation does not demonstrate that $C$ is not MSD. In fact this set is an MSD

code (see Sections 3 and 5). The first example of a proper MSD code, consisting of the four

words 01110101, 101, 110, and 11011, was given by A. Lempel in [10]. A list of 172 proper

MSD codes of “small” size was produced automatically by F. Guzman [8].

As examples of communications for which MSD codes suffice, A. Lempel gave in [10]
on-line compilation of inventories, construction of histograms, and updating of relative fre-

quencies. He discussed Kraft’s inequality which holds for UD codes and thereby limits the

number of short words that can occur in such sets. He raised the question of whether this

inequality must also hold for MSD codes. A. Restivo has since shown that Kraft’s inequality

need not hold for MSD codes [11] which leaves open the possibility that there may exist

situations in which MSD codes can provide greater efficiency than UD codes. For further

background on codes and coding theory the reader may wish to consult textbooks such as [1]

and [4].

The aim of this communication is to present a new algorithm, called the MSD algorithm.
The algorithm takes as input a code, say, $C$ over a $k$-element code alphabet consisting of $n$

words of total length $L$ and decides in $O(n^2L)$ time and $O((n+k)L)$ space whether or not $C$
is multiset decipherable. At an early stage this algorithm also detects the UD property when

it is present. It decides in $O(nL)$ time and $O((n+k)L)$ space whether or not $C$ is uniquely
decipherable. In case of a fixed code alphabet these bounds match asymptotically the best

known algorithms for this purpose ([3], [9], [12]).

The design of the MSD algorithm is based on a “domino approach.” First, we associate

with a code $C$ a “domino graph” $G$ and a “domino function” $d$ on its edges (see Section 2).

Our “dominoes” are derived from those in [6] and [7]. Next, we characterize the UD and MSD

properties of $C$ in terms of $G$ and $d$ (see Section 3). In order to compute the domino graph

and function we make use of concepts and methods from pattern matching in words due to

A. Aho and M. Corasick ([2], see Section 4). Finally, we decide the UD and MSD properties

of $C$ by means of the above characterizations (see Section 5). Most of our procedures are

similar to elementary graph algorithms. For background on such algorithms the reader may

wish to consult textbooks such as [5, Sec. 23]. Historical remarks about the domino approach

can be found in a forthcoming full version of [14].

Recently the authors have developed further algorithms based on the domino approach for
deciding whether or not a code is “numerically decipherable” [14], for computing the “finest

homophonic partition” of a code [14], and for computing its deciphering delay [13].
2 Defining the domino graph and function

Let $A$ be a code alphabet. Let $C$ be a code over $A$ consisting of $n$ words of total length $L$. The set of all prefixes of words in $C$ is denoted by $\text{Prefix}(C)$. We associate with $C$ a directed graph $G$ called the domino graph. The graph $G = (V, E)$ is determined by the vertex set

$$
V = \{\text{open}, \text{close}\} \cup \{(u, \varepsilon) : u \in \text{Prefix}(C) \setminus \{\varepsilon\}\} \cup \{(\varepsilon, u) : u \in \text{Prefix}(C) \setminus \{\varepsilon\}\}
$$

and by the edge set $E = E_1 \cup E_2 \cup E_3 \cup E_4$ where

$$
E_1 = \{(\text{open}, (u, \varepsilon)) : u \in C\} \cup \{(\text{open}, (\varepsilon, u)) : u \in C\},
$$

$$
E_2 = \{((u, \varepsilon), \text{close}) : u \in C\} \cup \{((\varepsilon, u), \text{close}) : u \in C\},
$$

$$
E_3 = \{((u, \varepsilon), (uv, \varepsilon)) : v \in C\} \cup \{((\varepsilon, u), (\varepsilon, uv)) : v \in C\},
$$

and

$$
E_4 = \{((u, \varepsilon), (\varepsilon, v)) : uv \in C\} \cup \{((\varepsilon, u), (\varepsilon, v)) : uv \in C\}.
$$

The domino function associated with $C$ is the mapping $d : E \rightarrow C \times \{\varepsilon\} \cup \{\varepsilon\} \times C$ which is defined

- on $E_1$ by $d(\text{open}, (u, \varepsilon)) = (\varepsilon, u)$ and $d(\text{open}, (\varepsilon, u)) = (u, \varepsilon),$

- on $E_2$ by $d((u, \varepsilon), \text{close}) = (u, \varepsilon)$ and $d((\varepsilon, u), \text{close}) = (\varepsilon, u),$

- on $E_3$ by $d((u, \varepsilon), (uv, \varepsilon)) = (\varepsilon, v)$ and $d((\varepsilon, u), (\varepsilon, uv)) = (v, \varepsilon),$ and

- on $E_4$ by $d((u, \varepsilon), (\varepsilon, v)) = (uv, \varepsilon)$ and $d((\varepsilon, u), (\varepsilon, v)) = (\varepsilon, uv).$

The pair $d(e)$ denotes the domino associated with the edge $e$ of $G$. Its first (second) component is denoted by $d_1(e)$ (by $d_2(e)$, respectively). Note that if $d_1(e)$ ($d_2(e)$) is nonnull then it is regarded to be an element of $C$, not a word in $A^*$. The so-defined mappings $d_1 : E \rightarrow C \cup \{\varepsilon\}$ and $d_2 : E \rightarrow C \cup \{\varepsilon\}$ are called domino functions, as well. In order to stress the analogy to real dominoes we often write $[^e\varepsilon]$ rather than $(u, \varepsilon)$ and $[^e\varepsilon]$ rather than $(\varepsilon, u)$ in order to denote a vertex of $G$ different from open or close and $[^e\varepsilon]$ rather than $(d_1(e), d_2(e))$ in order to denote the domino associated with an edge $e$ of $G$.

For every path $p$ in $G$ consisting of the edges $e_1, e_2, \ldots, e_m$ we define the word

$$
d(p) = d(e_1) \cdot d(e_2) \cdot \ldots \cdot d(e_m) \in (C \times \{\varepsilon\} \cup \{\varepsilon\} \times C)^*\n$$

and the words

$$
d_1(p) = d_1(e_1) \cdot d_1(e_2) \cdot \ldots \cdot d_1(e_m) \in C^*\n$$

3
\[ d_2(p) = d_2(v_1) \cdot d_2(v_2) \cdot \ldots \cdot d_2(v_n) \in C^*. \]

A path \( p \) in \( G \) initiating at open and terminating at some vertex \( [v_1] \) \( ([v_n]) \) is trying to find two factorizations of the same message over \( C \) into codewords beginning with distinct codewords. The factorizations obtained so far are \( d_1(p) \) and \( d_2(p) \). The word \( u \) in \( A^* \) denotes the "backlog" of the first (second) factorization as against the second (first) one.

We observe that \( G \) has at most \( 2L + 2nL \) vertices and at most \( 4n + 2(L-n) - n + 2(L-n) \) \( \leq 4nL \) edges. It is shown in a forthcoming full version of [14] that the latter bound is optimal up to a constant factor.

As an example let us again consider the code \( C = \{0, 0111110, 10101, 1111\} \) over the binary alphabet \( A = \{0, 1\} \). The domino graph and function associated with this code is displayed in Figure 1. The domino associated with an edge of the domino graph is represented as the label of this edge. The reader is invited to take any path in the domino graph initiating at open and terminating at close and to see how the dominoes associated with its edges match with each other.

3 Lemmas

Let the code alphabet \( A \), the code \( C \), the domino graph \( G \), and the domino functions \( d_1 \), \( d_2 \), and \( d_2 \) be as in Section 2. The two following lemmas characterize the UD and MSD properties of \( C \) in terms of the graph \( G \) and the functions \( d_1 \) and \( d_2 \).

Lemma 3.1 (See [9, Sec. 2]) \textit{The code} \( C \) \textit{is not UD if and only if} \( G \) \textit{has the following property. (P1):} \( G \) \textit{contains a path of length at least 3 that initiates at open and terminates at close.}

Lemma 3.2 \textit{The code} \( C \) \textit{is not MSD if and only if} \( G \) \textit{has the following property. (P2):} \textit{For some} \( w \) \textit{in} \( C \) \textit{there is a path} \( p \) \textit{in} \( G \), \textit{initiating at open and terminating at close, such that the symbol} \( w \) \textit{occurs a distinct number of times in the words} \( d_1(p) \) \textit{and} \( d_2(p) \).

Lemmas 3.1 and 3.2 can be easily derived from the definition of \( G \) and of \( d_1 \) and \( d_2 \). Note that (P2) implies (P1).

Let us illustrate Lemmas 3.1 and 3.2 for the code \( C = \{0, 0111110, 10101, 1111\} \) over the binary alphabet \( A = \{0, 1\} \). The domino graph and function associated with this code is displayed in Figure 1. Let us consider the path \( p_0 \) in the domino graph consisting of the vertices \( \{0\} \), \( \{1\} \), \( \{00\} \), \( \{01\} \), \( \{10\} \), \( \{11\} \), \( \{001\} \), \( \{110\} \), and \( \{111\} \), and \( \{1111\} \), and \( \{111111\} \). Note that \( d(p_0) = \)}
Figure 1: Domino graph and function (example).

The domino graph has property (P1) since $p_0$ initiates at open, terminates at close, and has length 8. It does not have property (P2) since for every path $p$ initiating at open and terminating at close the words $d_1(p)$ and $d_2(p)$ in $C^*$ contain every codeword the same number of times (see Section 5). For example, $d_1(p_0)$ and $d_2(p_0)$ contain each of the four codewords exactly once.

4 Computing the domino graph and function

Let $C$ be a code over a $k$-element code alphabet consisting of $n$ words having total length $L$. The purpose of this section is to compute efficiently the domino graph and function associated with $C$. The outcome is stated in the following theorem.
Theorem 4.1 There is an O(nL) time and O((n + k)L) space algorithm for computing the domino graph and function associated with C.

In order to prove Theorem 4.1 we make use of concepts and methods from pattern matching in words due to A. Aho and M. Corasick [2]. For the reader's convenience our presentation is independent from [2]. Our basic tools are two functions associated with C, called the prefix and suffix functions. At first, we define these functions and provide some properties of them (Lemmas 4.2–4.4). Then we present a procedure for computing the prefix and suffix function (Algorithm 4.5). An amortized analysis yields that this procedure requires O(L) time and O(kL) space. Let G be the domino graph associated with C. The crucial edges of G can be characterized in terms of the prefix and suffix function (Lemma 4.6). Using this characterization we are able to determine the domino graph and function associated with C in O(nL) time and space (Algorithm 4.7).

Let A be a code alphabet of cardinality k. Let C be a code over A consisting of n words of total length L. When a word u has a word v as a suffix, we write u ⊆ v. The unique longest member of a set S of words, if it exists, is denoted by max S. The prefix function \( \pi : \text{Prefix}(C) \rightarrow \text{Prefix}(C) \) associated with C is defined by setting

\[
\pi(u) = \begin{cases} 
\epsilon, & \text{if } u = \epsilon \\
\max\{v \in \text{Prefix}(C) : u \sqsupset v & u \neq v\}, & \text{otherwise}
\end{cases}
\]

for every u in Prefix(C). Note that iterating \( \pi \) on u yields the suffix chain \( u \sqsupset \pi(u) \sqsupset \pi^2(u) \sqsupset \ldots \sqsupset \epsilon \). The suffix function \( \sigma : \text{Prefix}(C) \rightarrow 2^C \) associated with C is specified by setting \( \sigma(u) = \{v \in C : u \sqsupset v\} \) for every u in Prefix(C). For technical reasons we further associate with C an append function which is a partial function \( \alpha : \text{Prefix}(C) \times A \rightarrow \text{Prefix}(C) \). The append function \( \alpha \) is defined by simply setting \( \alpha(u, a) = ua \) if \( ua \) is in \( \text{Prefix}(C) \) and \( \alpha(u, a) = \emptyset \) otherwise, for every \( (u, a) \) in \( \text{Prefix}(C) \times A \). The vertex set \( \text{Prefix}(C) \) together with the set of all edges of the form \( (u, \alpha(u, a)) \) with label a form the prefix tree for C rooted at \( \epsilon \).

The following lemma, which is implicitly contained in the proof of [2, Lem. 1], shows that the iterations of the prefix function \( \pi \) on a word u in \( \text{Prefix}(C) \) describe all suffixes of u which belong to \( \text{Prefix}(C) \).

Lemma 4.2 For every u in \( \text{Prefix}(C) \) and every v in \( A^* \), v is a suffix of u which belongs to \( \text{Prefix}(C) \) if and only if \( v = \pi^s(u) \) for some nonnegative integer s.

Proof: The "if" is clear from the definition of \( \pi \). In order to prove the "only if" let us consider a suffix \( v \) of u distinct from u which belongs to \( \text{Prefix}(C) \). Let \( s \geq 1 \) such that \( |\pi^{s-1}(u)| > |v| \geq |\pi^s(u)| \). Since \( \pi^{s-1}(u), v, \) and \( \pi^s(u) \) are all suffixes of u, we have that
u ⊢ π^{s-1}(u) ⊢ v ⊢ π^s(u). As v is distinct from π^{s-1}(u), the equality π^s(u) = π(π^{s-1}(u)) and the definition of π yield that v = π^s(u). □

Next we define for each (u, a) in the domain of α with nonnull u the set

\[ E(ua) = \{ π^s(u) : s ≥ 1 \text{ and } π^s(u) \cdot a ∈ \text{Prefix}(C) \}. \]

Our next lemma will be employed by Algorithm 4.5 in order to compute the prefix function π. It expresses the value of π on a word of the form ua as above in terms of the set \( E(ua) \).

**Lemma 4.3** (See [2, Lem. 1]) For every (u, a) in the domain of α,

\[ π(ua) = \begin{cases} 
ε, & \text{if } u = ε \text{ or if } u \neq ε \text{ and } E(ua) = \emptyset \\
(\max E(ua)) \cdot a, & \text{if } u \neq ε \text{ and } E(ua) \neq \emptyset.
\end{cases} \]

**Proof:** By Lemma 4.2, \( E(ua) \) is the set of all words \( v \) in \( A^* \) such that \( va \) is in \( \text{Prefix}(C) \) and \( v \) is a suffix of \( u \) distinct from \( u \). By definition of \( π \), \( π(ua) = \max \{ v ∈ \text{Prefix}(C) : ua ⊢ v \& va \neq v \} \). Let us first assume that \( π(ua) \) is nonnull. Then, \( u \) is nonnull as well and

\[ π(ua) = \max \{ va ∈ \text{Prefix}(C) : ua ⊢ va \& va \neq va \} = (\max E(ua)) \cdot a, \]

as desired. If \( π(ua) \) is the null word \( ε \) and \( u \) is nonnull, then there is no word \( v \) in \( A^* \) such that \( va \) is in \( \text{Prefix}(C) \) and \( va \) is a suffix of \( ua \) distinct from \( ua \). This implies that \( E(ua) \) is the empty set. □

The following lemma will be used by Algorithm 4.5 in order to compute the suffix function σ. It is a direct consequence of Lemma 4.2.

**Lemma 4.4** (See [2, Lem. 2]) For every \( u \) in \( \text{Prefix}(C) \), \( σ(ua) = C \cap \{ π^s(u) : s ≥ 0 \} \). In consequence, for every \( u \) in \( \text{Prefix}(C) \setminus \{ ε \} \), \( σ(u) = \{ u : u ∈ C \} \cup σ(π(u)) \).

We now present a procedure for computing the prefix function π and the suffix function σ associated with C. Let us first discuss some data structures. The elements of \( \text{Prefix}(C) \) are represented by the integers \( 0, 1, 2, \ldots, \#\text{Prefix}(C) - 1 \) in any reasonable way, where the null word \( ε \) is always represented by \( 0 \). Note that \( \#\text{Prefix}(C) ≤ L + 1 \). For every \( u \) in \( \text{Prefix}(C) \), the set \( σ(u) \) is represented by a list of its members ordered by decreasing length. According to Lemma 4.4, all these lists require only \( O(L) \) space. Note that the append function \( α \) associated with \( C \) is defined on only \( \#\text{Prefix}(C) - 1 \) elements of \( \text{Prefix}(C) \times A \). We use a sparse data structure for this function which requires \( O(kL) \) space, gives access to any particular value \( α(u, a) \) in constant time, and can be initialized in \( O(L) \) time. We define the functions \( β : \text{Prefix}(C) → 2^A \) and \( γ : \text{Prefix}(C) → 2^C \) by letting \( β(u) = \{ a ∈ A : α(u, a) ≠ ∅ \} \) and \( γ(u) = \{ u : u ∈ C \} \) for every \( u \) in \( \text{Prefix}(C) \). We proceed by the following algorithm.

**Algorithm 4.5** (See [2, Algs. 2 and 3]) Directly compute the set \( \text{Prefix}(C) \), the partial function \( α \), and the functions \( β \) and \( γ \). Compute the prefix function \( π \) and the suffix function \( σ \) associated with \( C \) as shown in Figure 2.
Figure 2: Computation of the functions $\pi$ and $\sigma$.

1. $\pi(\varepsilon) \leftarrow \varepsilon$; $\sigma(\varepsilon) \leftarrow \emptyset$;
2. initialize a queue $Q$ of elements in $\text{Prefix}(C)$ to be empty;
3. for all $a$ in $\beta(\varepsilon)$ do
4.     $w \leftarrow \alpha(\varepsilon, a)$;
5.     append $w$ to $Q$;
6.     $\pi(w) \leftarrow \varepsilon$; $\sigma(w) \leftarrow \gamma(w)$;
7. while $Q$ not empty do
8.     remove the first element, $u$, from $Q$;
9.     for all $a$ in $\beta(u)$ do
10.    $w \leftarrow \alpha(u, a)$;
11.    append $w$ to $Q$;
12.    $v \leftarrow \pi(w)$;
13.    while $v \neq \varepsilon$ and $\alpha(v, a) = \emptyset$ do
14.        $v \leftarrow \pi(v)$;
15.    if $\alpha(v, a) \neq \emptyset$ then
16.        $v \leftarrow \alpha(v, a)$; $\pi(w) \leftarrow v$; $\sigma(w) \leftarrow \gamma(w) \cup \sigma(v)$;
17.    else
18.        $\pi(w) \leftarrow \varepsilon$; $\sigma(w) \leftarrow \gamma(w)$;

It should be clear from the above discussion that the set $\text{Prefix}(C)$ and all values of $\alpha$, $\beta$, and $\gamma$ can be computed in $O(L)$ time and $O(kL)$ space. The procedure shown in Figure 2 basically performs a breadth-first search of the prefix tree for $C$ starting from the null word $\varepsilon$. It computes the values of $\pi$ and $\sigma$ (in Lines 1, 6, 16, and 18) according to Lemmas 4.3 and 4.4, respectively. The crucial point in the analysis of the procedure shown in Figure 2 is to observe that the number of runs of Line 14 is of order $O(L)$. Provided that this is true, it is easy to verify that Algorithm 4.5 requires $O(L)$ time and $O(kL)$ space.

In order to estimate the number of runs of Line 14 in Figure 2, let us take a word in $C$ consisting of the letters $a_1, \ldots, a_m$. Consider the "continuous" sequence of the words $v$ computed in the runs of the for loop directed by Line 9 applied for

$$(u, a) = (a_1, a_2), (a_1a_2, a_3), \ldots, (a_1 \ldots a_{m-1}, a_m).$$

That sequence begins with the null word $\varepsilon$. The length of $v$ is decreased in Line 14 and increased by 1 in Line 16. Therefore, in these runs of the for loop directed by Line 9, Line 14 is performed at most as often as Line 16, i.e., at most $m - 1$ times. In consequence, Line 14 is altogether performed at most $L - n$ times, as desired.

We now turn to the computation of the domino graph $G = (V, E)$ and the domino function $d : E \to C \times \{\varepsilon\} \cup \{\varepsilon\} \times C$ associated with $C$. Recall that $E$ was defined as the union of
Figure 3: Computation of $E_3$ and of $d$ on $E_3$.

1. $E_3 \leftarrow \emptyset$
2. for all $w$ in $\text{Prefix}(C) \setminus \{\varepsilon\}$ do
   3. color($w$) $\leftarrow$ white;
4. for all $z$ in $C$ do
   5. for $j \leftarrow 1$ to $|z|$ do
      6. $w \leftarrow \pi_n(j)$;
      7. if color($w$) = white then
         8. color($w$) $\leftarrow$ black;
      9. for all $v$ in $\sigma(w) \setminus \{w\}$ do
         10. $u \leftarrow \pi_n(j - |w|)$;
         11. $E_3 \leftarrow E_3 \cup \{(u, \varepsilon), (w, v), ((\varepsilon, u), (\varepsilon, w))\}$;
         12. $d((u, \varepsilon), (w, v)) \leftarrow (\varepsilon, v)$;
         13. $d((\varepsilon, u), (\varepsilon, w)) \leftarrow (v, \varepsilon)$;

four pairwise disjoint sets $E_1$, $E_2$, $E_3$, and $E_4$. The following lemma characterizes the edges in $E_3$ and $E_4$ in terms of the prefix function $\pi$ and the suffix function $\sigma$ associated with $C$.

**Lemma 4.6** The edge set $E_3$ consists of all pairs of vertices of $G$ having either the form $e_1 = ((u, \varepsilon), (w, \varepsilon))$ or $e_2 = ((\varepsilon, u), (\varepsilon, w))$ with $uw = w \neq \varepsilon$ and $v$ in $\sigma(w) \setminus \{w\}$. Furthermore, $d(e_1) = (\varepsilon, v)$ and $d(e_2) = (v, \varepsilon)$. The edge set $E_4$ consists of all pairs of vertices of $G$ having either the form $e_3 = ((u, \varepsilon), (\varepsilon, v))$ or $e_4 = ((\varepsilon, u), (\varepsilon, w))$ with $w = uv$ in $C$ and $v$ in $\{\pi^t(w) : s \geq 1\} \setminus \{\varepsilon\}$. Furthermore, $d(e_3) = (w, \varepsilon)$ and $d(e_4) = (\varepsilon, w)$.

**Proof:** The first characterization directly follows from the definition of $\sigma$. The second one is clear by Lemma 4.2. $\square$

In order to compute $G$ and $d$ we further need for each codeword $w$ in $C$ the function $\pi_w : \{0, \ldots, |w|\} \to \text{Prefix}(C)$ which is defined by letting $\pi_w(j)$ be the prefix of $w$ of length $j$. Note that $\pi_w(0) = \varepsilon$ and $\pi_w(j) = \alpha(\pi_w(j - 1), a_j)$ for every $j \geq 1$ where $a_j$ is the $j$th letter of $w$. We proceed by the following algorithm.

**Algorithm 4.7** Directly compute the vertex set $V$ and the edge sets $E_1$ and $E_2$ of the domino graph $G$ and the values of the domino function $d$ on $E_1 \cup E_2$. For each codeword $w$ in $C$ compute the function $\pi_w$ using the above equations. Compute the edge sets $E_3$ and $E_4$ of $G$ and the values of $d$ on $E_3 \cup E_4$ as shown in Figures 3 and 4.

It should be clear that the sets $V$, $E_1$, and $E_2$, the values of $d$ on $E_1 \cup E_2$, and the functions $\pi_w$ for all $w$ in $C$ can be computed in $O(L)$ time and $O(L)$ extra space. The procedures shown
Figure 4: Computation of $E_4$ and of $d$ on $E_4$.

1. $E_4 \leftarrow \emptyset$;
2. for all $w$ in $C$ do
3.   $v \leftarrow \pi(w)$;
4.   while $v \neq \varepsilon$ do
5.     $u \leftarrow \pi_w(|w| - |v|)$;
6.     $E_4 \leftarrow E_4 \cup \{((u, \varepsilon), (\varepsilon, v)), ((\varepsilon, u), (v, \varepsilon))\}$;
7.     $d((u, \varepsilon), (\varepsilon, v)) \leftarrow (w, \varepsilon)$;
8.     $d((\varepsilon, u), (v, \varepsilon)) \leftarrow (\varepsilon, w)$;
9.     $v \leftarrow \pi(v)$;

Figure 5: Prefix tree for $C$ (example).

in Figures 3 and 4 compute $E_3$ and $E_4$ and the values of $d$ on $E_3 \cup E_4$ according to Lemma 4.6. The first procedure requires $O(nL)$ time and space, the second one requires $O(L)$ time and space. Therefore, Algorithm 4.7 has altogether time and space complexity of order $O(nL)$.

Combining Algorithms 4.5 and 4.7 we conclude that the domino graph and function associated with $C$ can be computed in $O(nL)$ time and $O((n+k)L)$ space. By this, Theorem 4.1 has been established.

Finally, let us illustrate Algorithms 4.5 and 4.7 for the code $C = \{0, 0111110, 10101, 1111\}$ over the binary alphabet $A = \{0, 1\}$. The set $\text{Prefix}(C)$, the append function $\alpha$ associated with $C$, and the functions $\beta$ and $\gamma$ are given in Table 1. The prefix tree for $C$ can be seen in Figure 5. All vertices of this tree representing a codeword are marked by an asterisk. The computation of the prefix function $\pi$ and the suffix function $\sigma$ associated with $C$ according to the procedure of Figure 2 is demonstrated in Table 2. The procedure of Figure 3 determines the edges $(((1,0),(5,0)), ((2,0),(6,0)), ((6,0),(7,0)), ((8,0),(9,0))$, and $(((10,0),(11,0)))$ as members of the edge set $E_3$. The procedure of Figure 4 finds the edges $((5,0),(0,9))$, $((6,0),(0,1))$, $((9,0),(0,10))$, $((10,0),(0,2))$, $((11,0),(0,8))$, $((8,0),(0,14))$, $((10,0),(0,2))$, $((11,0),(0,8))$, $((8,0),(0,14))$,
Table 1: Prefix\((C), \alpha, \beta, \text{ and } \gamma \text{ (example).}\\

<table>
<thead>
<tr>
<th>u</th>
<th>\alpha(u,0)</th>
<th>\alpha(u,1)</th>
<th>\beta(u)</th>
<th>\gamma(u)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\varepsilon</td>
<td>0</td>
<td>8</td>
<td>{0,1}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>{1}</td>
<td>{1}</td>
</tr>
<tr>
<td>01</td>
<td>2</td>
<td>3</td>
<td>{1}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>4</td>
<td>{1}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>0111</td>
<td>4</td>
<td>5</td>
<td>{1}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>01111</td>
<td>5</td>
<td>6</td>
<td>{1}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>011111</td>
<td>6</td>
<td>7</td>
<td>{0}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>0111110</td>
<td>7</td>
<td>0</td>
<td>\emptyset</td>
<td>\emptyset</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>9</td>
<td>13</td>
<td>{0,1}</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>10</td>
<td>{1}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>101</td>
<td>10</td>
<td>11</td>
<td>{0}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>1010</td>
<td>11</td>
<td>12</td>
<td>{1}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>10101</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>{12}</td>
</tr>
<tr>
<td>11</td>
<td>13</td>
<td>0</td>
<td>14</td>
<td>{1}</td>
</tr>
<tr>
<td>111</td>
<td>14</td>
<td>0</td>
<td>15</td>
<td>{1}</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>{15}</td>
</tr>
</tbody>
</table>

\(((13, 0), (0, 13)), \text{ and } ((14, 0), (0, 8))\) as members of the edge set \(E_4\).

5 MSD algorithm

Let \(C\) be a code over a \(k\)-element code alphabet consisting of \(n\) words having total length \(L\). The purpose of this section is to decide efficiently whether or not \(C\) is a UD code or an MSD code. The outcome is stated in the following theorem.

**Theorem 5.1** There is an \(O(n^2 L)\) time and \(O((n+ k)L)\) space algorithm for deciding whether or not \(C\) is multiset decipherable. The same algorithm determines at an early stage in \(O(nL)\) time and \(O((n + k)L)\) space whether or not \(C\) is uniquely decipherable.

In order to prove Theorem 5.1 we employ the methods and results developed in Sections 2–4. First of all, the domino graph and function associated with \(C\), which was defined in Section 2, is computed as shown in Section 4. The UD property of \(C\) is decided by means of the characterization given in Lemma 3.1 (Algorithm 5.2). Then we compute for every codeword a new mapping on the vertices of the domino graph (Algorithm 5.3). These mappings are used in order to transform the characterization of the MSD property of \(C\) given in Lemma 3.2 into a decidable criterion (Lemma 5.4). Finally, the new criterion is decided (Algorithm 5.5).
Table 2: Computation of the functions $\pi$ and $\sigma$ (example).

<table>
<thead>
<tr>
<th>$u$</th>
<th>$v$</th>
<th>$\pi(w)$</th>
<th>$\sigma(w)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>${1}$</td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>0</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>1</td>
<td>$\sigma(1)$</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>3</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>14</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>11</td>
<td>$\sigma(1)$</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>3</td>
<td>$\sigma(15)$</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5</td>
<td>$\sigma(15)$</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>${12}$</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>6</td>
<td>$\sigma(15)$</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>7</td>
<td>${7} \cup \sigma(1)$</td>
</tr>
</tbody>
</table>

Let $A$ be a code alphabet of cardinality $k$. Let $C$ be a code over $A$ consisting of $n$ words of total length $L$. Let the domino graph $G$ and the domino functions $d$, $d_1$, and $d_2$ associated with $C$ be defined as in Section 2. Recall that $G$ has $O(L)$ vertices and $O(nL)$ edges. According to Theorem 4.1, the graph $G$ and the function $d$ can be computed in $O(nL)$ time and $O((n+k)L)$ space. We may assume that the edges of the graph $G$ are represented by adjacency-lists. Using breadth-first search of $G$ starting from open and breadth-first search of the transpose of $G$ starting from close, we "trim" the graph $G$ in $O(nL)$ time and space by removing all vertices and edges which do not appear on any path initiating at open and terminating at close. For convenience we will from now on use $G = (V, E)$ to denote this trimmed version of the domino graph associated with $C$.

We now turn to the proof of Theorem 5.1. According to Lemmas 3.1 and 3.2 it is sufficient to decide properties (P1) and (P2) of $G$. These procedures are carried out by Algorithms 5.2, 5.3, and 5.5.

**Algorithm 5.2** Decide whether or not the domino graph $G$ satisfies (P1), i.e., decide whether or not there is an edge of the form $(\text{open}, y)$ in $G$ such that either the first or the second vertex in the adjacency-list of $y$ in $G$ is distinct from close. According to Lemma 3.1, the code $C$ is UD if and only if such an edge does not exist.

Since there are $2n$ edges in $G$ initiating at open, Algorithm 5.2 requires $O(n)$ time and constant extra space.
Figure 6: Computation of $g_w$.

1. for all $y$ in $V$ do
2.     color($y$) ← white;
3. initialize a queue $Q$ of vertices of $G$ to be empty;
4. append open to $Q$;
5. color(open) ← black;
6. $g_w$(open) ← 0;
7. while $Q$ not empty do
8.     remove the first element, $y$, from $Q$;
9.     for all $e = (y, z)$ in $E$ do
10.        if color($z$) = white then
11.            append $z$ to $Q$;
12.            color($z$) ← black;
13.            $g_w$(z) ← $g_w$(y) + $f_w$(e);

For each codeword $w$ in $C$ and for each edge $e$ in $E$ we define

$$f_w(e) = \begin{cases} 
1, & \text{if } d(e) = (w, e) \\
-1, & \text{if } d(e) = (e, w) \\
0, & \text{otherwise.}
\end{cases}$$

For each path $p$ in $G$ consisting of the edges $e_1, e_2, \ldots, e_m$ we further define $f_w(p)$ to be $f_w(e_1) + f_w(e_2) + \ldots + f_w(e_m)$. Note that $f_w(p)$ denotes the difference of the number of occurrences of the symbol $w$ in the words $d_1(p)$ and $d_2(p)$ in $C^*$. In the next two algorithms the value of $f_w$ on an edge of $G$ is directly computed in constant time from the domino associated with this edge whenever required.

**Algorithm 5.3** Let $w$ be any codeword in $C$. Compute the mapping $g_w : V \to Z$ as shown in Figure 6.

For every fixed codeword $w$, the procedure of Figure 6 describes a breadth-first search of the domino graph $G$ starting from open which requires $O(nL)$ time and $O(L)$ extra space. Consider the breadth-first tree generated by this procedure. For every vertex $y$ of $G$ let $p$ be the uniquely determined path in the breadth-first tree initiating at open and terminating at $y$. Then, $g_w(y) = f_w(p)$. The next lemma characterizes property (P2) of $G$ in terms of the mappings $f_w$ and $g_w$.

**Lemma 5.4** The domino graph $G$ satisfies (P2) if and only if, for some codeword $w$ in $C$, $g_w$(close) ≠ 0 or there is an edge $e = (y, z)$ in $E$ for which $g_w(z) ≠ g_w(y) + f_w(e)$. 

13
Proof: "If." Suppose that there is a codeword \( w \) such that \( g_w(\text{close}) \neq 0 \). According to the above discussion there is a path \( p \) in \( G \) initiating at open and terminating at close such that \( g_w(\text{close}) = f_w(p) \). Then, \( f_w(p) \neq 0 \), i.e., \( G \) satisfies (P2). Suppose next that there is a codeword \( w \) and an edge \( e = (y, z) \) for which \( g_w(x) \neq g_w(y) + f_w(e) \). According to the above discussion there are paths \( p_1 \) and \( p_2 \) in \( G \) both initiating at open and terminating at \( y \) and \( z \), respectively, such that \( g_w(y) = f_w(p_1) \) and \( g_w(z) = f_w(p_2) \). Take any path \( q \) in \( G \) initiating at \( z \) and terminating at close. Since by assumption

\[
   f_w(p_1 \circ e) = g_w(y) + f_w(e) = g_w(z) = f_w(p_2),
\]
either \( f_w(p_1 \circ e \circ q) \neq 0 \) or \( f_w(p_2 \circ q) \neq 0 \). Thus, \( G \) has property (P2).

"Only if:" Suppose that \( G \) satisfies (P2). Then, there is a codeword \( w \) in \( C \) and a path \( p \) in \( G \), initiating at open and terminating at close, such that the symbol \( w \) occurs a distinct number of times in the words \( d_1(p) \) and \( d_2(p) \). If for each edge \( e = (y, z) \) of the path \( p \) the equality \( g_w(x) = g_w(y) + f_w(e) \) held, then it can be shown that \( g_w(\text{close}) = g_w(\text{open}) + f_w(p) = f_w(p) \). By assumption, \( f_w(p) \neq 0 \). Thus, either there is an edge \( e = (y, z) \) in \( E \) for which \( g_w(z) \neq g_w(y) + f_w(e) \) or \( g_w(\text{close}) \neq 0 \). \( \square \)

Algorithm 5.5 For each codeword \( w \) in \( C \) compute the mapping \( g_w \) according to Algorithm 5.3 and test the equalities \( g_w(\text{close}) = 0 \) and \( g_w(z) = g_w(y) + f_w(e) \) for every edge \( e = (y, z) \) in \( E \). According to Lemmas 3.2 and 5.4, the code \( C \) is MSD if and only if all these equalities hold.

Since \( C \) consists of \( n \) words and \( G \) has \( O(nL) \) edges and since Algorithm 5.3 requires \( O(nL) \) time and \( O(L) \) extra space, Algorithm 5.5 has time and space complexity \( O(n^2L) \) and \( O(L) \), respectively.

In conclusion, Theorem 5.1 has been established. The procedure for deciding whether or not \( C \) is an MSD code consists of the computation of the trimmed domino graph \( G \) and the domino function \( d \) according to Theorem 4.1 and of Algorithm 5.5. In order to decide the UD property of \( C \) we only compute \( G \) as above and then run Algorithm 5.2.

Finally, let us illustrate the crucial Algorithm 5.3 presented in this section for the code \( C = \{0,0111110,10101,1111\} \) over the binary alphabet \( A = \{0,1\} \). The domino graph \( G = (V, E) \) and function \( d \) associated with \( C \) is shown in Figure 1. Note that the graph \( G \) is trimmed by removing the vertices \([011]_e, [011]_l, [11]_e, [011]_l, [11]_l, [01111]_e, [01111]_l, \) and \([1]_l\) from it. Let us consider the first codeword \( w_1 = 0 \). The mapping \( g_{w_1} : V \to 2 \) computed by Algorithm 5.3 and the way it is computed is represented in Figure 7. The value \( g_{w_1}(y) \) is shown as the label of the vertex \( y \). The value \( f_{w_1}(e) \) can be easily derived from the label \( d(e) \) of the edge \( e \). An edge \( e = (y, z) \) is thick if in the procedure of Figure 6 the vertex \( z \) is "explored" from the vertex \( y \) by walking on \( e \), i.e., in Line 13 of Figure 6 \( g_{w_1}(z) \) is computed as \( g_{w_1}(y) + f_{w_1}(e) \).
Figure 7: Computation of $g_w$ (example).

The other edges are thin. The thick edges form the breadth-first tree of the trimmed domino graph $G$ being generated by Algorithm 5.3. Note that $g_{w_1}(\text{close}) = 0$ and, for each edge $e = (y, z) \in E$, $g_{w_1}(z) = g_{w_1}(y) + f_{w_1}(e)$. The same holds for the three other codewords. Therefore, $C$ is an MSD code.

References


