MINING DETERMINING SETS FOR PARTIALLY DEFINED FUNCTIONS USING ENTROPY

BY

DAN A. SIMOVICI, DAN PLETEA and ROSANNE VETRO

Abstract. This paper describes an algorithm that determines the minimal sets of variables that determine the values of a discrete partial function. The algorithm is based on the notion of entropy of a partition and is able to achieve an optimal solution. A limiting factor is introduced to restrict the search, thereby providing the option to reduce running time. Experimental results are provided that demonstrate the efficiency of the algorithm for functions with up to 24 variables. The effect of the limiting factor on the optimality of the algorithm for different sizes of partial functions is also examined.

Key words: partially defined function, entropy of a partition, determining set.

2000 Mathematics Subject Classification: 03G10, 54C70, 05A18.

1. Introduction

Partially defined finite functions are studied by both mathematicians and engineers due to their many technical applications, particularly in designing switching circuitry. They model such diverse circuits as logical programmable arrays, or content addressable memory. The performance of such circuits (including wiring complexity, power dissipation, etc.) is heavily influenced by the number of arguments on which the function implemented by the circuit depends effectively.

The goal of this paper is to present an algorithm to generate various sets of input variables on which a partial function depends using a clustering technique for discrete functions starting from the conditional entropy that measures the discrepancy between the kernel partitions of these functions.

The proposed algorithm is based on the partial order that is naturally defined on the set of partitions of a set and on the fundamental observation that a superset of a determining set for a partially defined function $f$ is itself a
determining set for $f$. We use this dual heredity property of determining sets to formulate an algorithm that computes the determining sets by evaluating the conditional entropy of the output variable relative to subsets of the set of input variables of a partial function. Experimental results show the effectiveness of the algorithm.

This problem has been addressed in T. Sasao's seminal paper [1] using an algebraic minimization algorithm that is applied to functions that depend on small number of variables. Our approach is distinct and involves techniques inspired by data mining. Additionally, it has the advantage of not being linked to any value of the input or output radix of the partial function $f$.

In [2] we developed an Apriori-like algorithm [3], [4], [5] that computes the determining sets by traversing the entire lattice of subsets of the set of variables. The current approach applies techniques based on entropy of partitions and uses a limiting factor to narrow the search scope and considerably improve running time without affecting the quality of the solutions.

The rest of the paper is organized as follows. In Section 2, the notion of determining set for a partial function is introduced and a few properties of these sets that play a role in our algorithm are examined. Section 3 introduces the conditional entropy of partitions that is used to find determining sets. The algorithm is presented in Section 4. Section 5 discusses experimental work related to the algorithm. Finally, Section 6 presents our conclusions.

2. Determining Sets for Partially Defined Functions

We denote the finite set \{0,1,...,n-1\} by $n$. The partial functions that we study have as domain a subset of the finite set $r^n$ and as range a subset of the finite set $p$ for some positive natural numbers $r$ and $p$, referred to as the input radix and the output radix of the function, respectively. The set of all such partial functions is denoted by $PF(r^n, p)$. If $f \in PF(r^n, p)$ we denote by $\text{Dom}(f)$ the set of all $n$-tuples $(a_1,\ldots,a_n)$ in $r^n$ for which $f(a_1,\ldots,a_n)$ is defined.

A partial function $f \in PF(r^n, p)$ is specified as a table $T_f$ having columns labelled by the argument variables $x_1,\ldots,x_n$ and by the output variable $y$. The set of variables that occur in a table $T_f$ is denoted by $\text{Var}_f$.

If $f(a_1,\ldots,a_n) = b$ we have in the table $T_f$ the $(n + 1)$-tuple $t = (a_1,\ldots,a_n,b)$. For example, in Table 1 we show a partial function defined on all triplets in $3^3$ that contain at least two non-zero elements, and ranging in the set 4. The number of rows of the table that represents a partial function defined on $r^n$ can range between 0 and $r^n$. Usually, the number of rows of such a function is smaller than $r^n$ and, often this number is much smaller. Tuples $(a_1,\ldots,a_n)$ that do not belong to the definition domain of $f$ are considered as “don't care” tuples, that is, as input sequences that are unlikely to occur as inputs of the functions, or the output of the function for such inputs is indifferent to the designer.

For a tuple $t$ in $T_f$ and a set of variables $U \subseteq \text{Var}_f$ we denote by $t[U]$ the projection of $t$ on $U$, that is, the restriction of $t$ to the set $U$. If
\( U = \{ u_1, u_2, \ldots, u_m \} \) is a set of variables we will use the alternative notation \( U = u_1 u_2 \ldots u_m \).

### Table 1

<table>
<thead>
<tr>
<th>x_1</th>
<th>x_2</th>
<th>x_3</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Definition 2.1.** A set of variables \( V = \{ x_{i_1}, \ldots, x_{i_p} \} \subseteq \text{Var}_f \) is a determining set for the partial function \( f \) if for every two tuples \( t \) and \( s \) from \( T_f \), \( t[ V] = s[ V] \) implies \( t[ y] = s[ y] \).

In other words, \( V \) is a determining set for the partial function \( f \) if \( t = (a_1, \ldots, a_n, b) \) and \( s = (c_1, \ldots, c_n, d) \) in \( T_f \) such that \( a_{i_k} = c_{i_k} \) for \( 1 \leq k \leq p \) implies \( b = d \). The collection of determining sets for \( f \) is denoted by \( \text{DS}(f) \).

\( V \) is a minimal determining set for \( f \) if \( V \) is a determining set for \( f \) and there is no strict subset of \( V \) that is a determining set for \( f \). The set of minimal determining sets of \( f \) is denoted by \( \text{MDS}(f) \). Our main purpose is to present an algorithm that extracts the minimal determining sets for a partially specified function.

We introduce a partial order relation “\( \sqsubseteq \)” on the set \( \text{PF}(r^n, p) \) by defining \( f \sqsubseteq g \) if \( \text{Dom}(f) \subseteq \text{Dom}(g) \) and \( f(a_1, \ldots, a_n) = g(a_1, \ldots, a_n) \) for every \( (a_1, \ldots, a_n) \). In other words, we have \( f \sqsubseteq g \) if \( g \) is an extension of \( f \).

The following simple statement is crucial to the proposed algorithm.

**Theorem 2.2.** Let \( f \) and \( g \) be two partial functions in \( \text{PF}(r^n, p) \). If \( V \in \text{DS}(f) \) and \( V \subseteq W \), then \( W \subseteq \text{DS}(f) \). Furthermore, if \( f \sqsubseteq g \), then \( \text{DS}(g) \subseteq \text{DS}(f) \).

**Proof.** If \( V \) and \( W \) are two sets of variables such that \( V \subseteq W \) and \( t, s \) are two tuples in \( T_f \), then \( t[ W] = s[ W] \) implies \( t[ V] = s[ V] \). Therefore, if \( V \) is a determining set for \( f \) and \( t[ W] = s[ W] \), it follows that \( t[ V] = s[ V] \), which implies \( t[y] = s[y] \). Thus, \( W \) is a determining set for \( f \).

For the second part of the theorem, observe that if \( f \sqsubseteq g \) and \( V \in \text{DS}(g) \),
then \( t[V] = s[V] \) implies \( t[y] = s[y] \), for every \( t, s \in \text{Dom}(g) \). Since \( \text{Dom}(f) \subseteq \text{Dom}(g) \), the same implication holds for any two tuples in \( \text{Dom}(f) \), so \( V \in \text{DS}(f) \).

Note that if \( f \preceq g \) and \( V \in \text{MDS}(g) \), then there exists \( Z \in \text{MDS}(f) \) such that \( Z \subseteq V \).

### 3. Entropies Associated with Partial Functions

Entropy is a probabilistic concept that lies at the foundation of information theory. The entropy of a partition takes advantage of the partial order that is naturally defined on the set of partitions of a set. In [6] a generalized notion of entropy for partitions, with Shannon entropy as a special case, is introduced.

A partition on a set \( S \) is a collection \( \pi \) of nonempty, pairwise disjoint sets

\[
\pi = \{B_1, \ldots, B_m\},
\]

such that \( \bigcup_{i=1}^m B_i = S \). The sets \( B_i \) are referred to as the blocks of \( \pi \). The set of partitions of \( S \) is denoted by \( \text{PART}(S) \). If \( \pi, \rho \in \text{PART}(S) \) we say that \( \pi \preceq \rho \) if every block of \( \pi \) is included in a block of \( \rho \); equivalently, \( \pi \preceq \rho \) if every block of \( \rho \) is a union of blocks of \( \pi \). For \( \pi, \rho \in \text{PART}(S) \) we say that \( \rho \) covers \( \pi \) if \( \pi \preceq \rho \) and there is no \( \sigma \in \text{PART}(S) \) distinct from \( \pi \) and \( \rho \) such that \( \pi \preceq \sigma \preceq \rho \). This is denoted by \( \pi \prec \rho \). It can be shown that \( \pi \prec \rho \) if the blocks of \( \rho \) coincide with the blocks of \( \pi \), with the exception of one block of \( \rho \) which is the union of two blocks of \( \pi \).

The partially ordered set \( (\text{PART}(S), \preceq) \) is actually a lattice. If \( \pi, \rho \in \text{PART}(S) \), \( \pi = B_1, \ldots, B_m \) and \( \rho = (C_1, \ldots, C_n) \), the greatest lower bound of \( \pi \) and \( \rho \) is the partition \( \pi \wedge \rho \) given by

\[
\pi \wedge \rho = \{B_i \cap C_j \mid B_i \cap C_j = \emptyset\}.
\]

If \( C \) is a subset of \( S \) and \( \pi = \{B_1, \ldots, B_m\} \in \text{PART}(S) \), the trace of \( \pi \) on \( C \) is the partition \( \{C \cap B_1, \ldots, C \cap B_m\} \in \text{PART}(S) \). Unless stated otherwise, all logarithms are in base 2.

**Definition 3.1.** Let \( S \) be a finite set and let \( \pi = B_1, \ldots, B_n \) be a partition of \( S \). The Shannon entropy of \( \pi \) is the number:

\[
\mathcal{H}(\pi) = \sum_{i=1}^n \frac{|B_i|}{|S|} \log \frac{|B_i|}{|S|}.
\]
The Shannon entropy can be used to evaluate the uniformity of the distribution of elements of \( S \) in the blocks \( \pi \) since the entropy value increases with the uniformity of the distribution of the elements of \( S \).

In [7] we have shown that the entropy of a partition is a dually monotonic function. In other words, if \( \pi \leq \rho \), we have \( \mathcal{H}(\rho) \geq \mathcal{H}(\pi) \).

If \( \pi, \rho \in \text{PART}(S) \), \( \pi = B_1, \ldots, B_n \) and \( \rho = (C_1, \ldots, C_n) \), then the conditional entropy of \( \pi \) on \( \rho \) is the number

\[
\mathcal{H}(\pi|\rho) = \sum_{j=1}^{n} \frac{|C_j|}{|S|} \mathcal{H}(\pi_{C_j}),
\]

that is, the weighted average of the entropies of the traces of \( \pi \) on the blocks of \( \rho \). An equivalent expression of the conditional entropy can be obtained as follows:

\[
\mathcal{H}(\pi|\rho) = \sum_{j=1}^{n} \frac{|C_j|}{|S|} \mathcal{H}(\pi_{C_j})
= \sum_{j=1}^{n} \frac{|C_j|}{|S|} \sum_{i=1}^{m} \frac{|B_i \cap C_j|}{|C_j|} \log \frac{|B_i \cap C_j|}{|C_j|}
= \sum_{j=1}^{n} \sum_{i=1}^{m} \frac{|B_i \cap C_j|}{|S|} \log \frac{|B_i \cap C_j|}{|C_j|}
= \sum_{j=1}^{n} \sum_{i=1}^{m} \frac{|B_i \cap C_j|}{|S|} \log \frac{|B_i \cap C_j|}{|S|} \frac{|S|}{|C_j|}
= \sum_{j=1}^{n} \sum_{i=1}^{m} \frac{|B_i \cap C_j|}{|S|} \log \frac{|B_i \cap C_j|}{|S|} - \sum_{j=1}^{n} \sum_{i=1}^{m} \frac{|B_i \cap C_j|}{|S|} \log \frac{|C_j|}{|S|}
= \mathcal{H}(\pi \wedge \rho) - \mathcal{H}(\rho)
\]

Our algorithm uses the conditional entropy of partitions defined as follows. If \( U, V \) are two sets of attributes, the entropy of \( U \) conditioned upon \( V \) is the difference

\[
\mathcal{H}(U|V) = \mathcal{H}(UV) - \mathcal{H}(V)
\]
The monotonicity of \( \mathcal{H} \) implies that the function \( \mathcal{H}(\cdot): \mathcal{P}(\text{Var}_f) \to \mathbb{R} \) is monotonic in its first argument \( U \).

**Lemma 3.2.** Let \( S \) be a set and let \( C, D \) be two disjoint, non-empty subsets of \( S \). If \( \pi \in \text{PART}(S) \), then

\[
\mathcal{H}(\pi_{C \cup D}) \geq \frac{|C|}{|C \cup D|} \mathcal{H}(\pi_C) + \frac{|D|}{|C \cup D|} \mathcal{H}(\pi_D).
\]

**Proof.** Let \( \pi = \{B_1, \ldots, B_m\} \in \text{PART}(S) \). Since the function \( f : (0,1] \to \mathbb{R} \) given by \( f(x) = -x \log x \) for \( x \in (0,1] \) is concave, we have

\[
-(ap \log p - bq \log q) \leq -(ap + bq) \log (ap + bq),
\]

for every non-negative numbers \( a, b \) such that \( a + b = 1 \) and every \( p, q \in (0,1] \).

If we choose \( a = \frac{|C|}{|C \cup D|}, b = \frac{|D|}{|C \cup D|}, p_i = \frac{|B_i \cap C|}{|C|}, \) and \( q_i = \frac{|B_i \cap D|}{|D|} \)

note that \( ap_i + bq_i = \frac{|B_i \cap (C \cup D)|}{|C \cup D|} \).

Thus, we obtain the inequality:

\[
- \frac{|C|}{|C \cup D|} \frac{|B_i \cap C|}{|C|} \log \frac{|B_i \cap C|}{|C|} - \frac{|D|}{|C \cup D|} \frac{|B_i \cap D|}{|D|} \log \frac{|B_i \cap D|}{|D|}
\]

\[
\leq - \frac{1}{|C \cup D|} \frac{|B_i \cap (C \cup D)|}{|C \cup D|} \log \frac{|B_i \cap (C \cup D)|}{|C \cup D|}.
\]

If we sum up these inequalities for \( 1 \leq i \leq n \), the inequality of the lemma follows immediately.

**Theorem 3.3.** Let \( S \) be a set and let \( \pi, \sigma \in \text{PART}(S) \). The conditional entropy \( \mathcal{H} = (\pi|\sigma) \) is monotonic relative to \( \sigma \).

**Proof.** It suffices to show that if \( \sigma \prec \sigma' \), then \( \mathcal{H}(\pi|\sigma) \leq \mathcal{H}(\pi|\sigma') \).

Without loss of generality we can assume that \( \sigma = \{C_1, \ldots, C_{n-2}, C_{n-1}, C_n\} \) and
\sigma' = \{C_1, ..., C_{n-2}, C_{n-1} \cup C_n\}.

In view of the definition of conditional entropy we need to prove that

\[ \frac{|C_{n-1}|}{|S|} \mathcal{H}(\pi_{C_{n-1}}) + \frac{|C_n|}{|S|} \mathcal{H}(\pi_{C_n}) \leq \frac{|C_{n-1} \cup C_n|}{|S|} \mathcal{H}(\pi_{C_{n-1} \cup C_n}), \]

which follows immediately from Lemma 3.2.

**Definition 3.4.** Let \( f \) be a partial function, \( f \in PF(r^n, p) \) and let \( V \) be a set of variables of \( f, V \subseteq \{x_1, ..., x_n, y\} \). Define the partition \( \pi^V \) of \( \text{Dom}(f) \) by its corresponding equivalence \( \sim_v \), where \( u \sim_v w \) if \( u[V] = w[V] \).

The entropy of \( V \mathcal{H}(V) \) is the entropy \( \mathcal{H}(\pi^V) \) of the partition \( \pi^V \).

Note that if \( V \) and \( V' \) are two sets of variables such that \( V \subseteq V' \), then \( \pi^V \leq \pi^{V'} \).

Thus, \( V \subseteq V' \) implies \( \mathcal{H}(V) \geq \mathcal{H}(V') \), so the entropy is monotonic with respect to inclusion of attribute sets.

The role of the conditional entropy in detecting determining sets is highlighted by the next statement.

**Theorem 3.5.** Let \( f \) be a partial function, \( f \in PF(r^n, p) \) and let \( U \) be a set of variables of \( f \). Then, \( X \) is a determining set of \( f \) if and only if \( \mathcal{H}(y \mid X) = 0 \).

**Proof.** Suppose that \( X \) is a set of variables such that \( \mathcal{H}(y \mid X) = 0 \) and that \( \pi^X = \{C_1, ..., C_n\} \). Then, by the definition of conditional entropy we have

\[ \sum_{j=1}^{n} \left| \frac{|C_j|}{|S|} \mathcal{H}(\pi^X_{C_j}) \right| = 0, \]

which implies \( \mathcal{H}(\pi^X_{C_j}) = 0 \) for \( 1 \leq j \leq n \). Thus, each block \( C_j \) of \( \pi^X \) is included in a block of \( \pi^V \), so the values of the variables in \( X \) determine the value of the output variable \( y \). Therefore, \( X \) is a determining set.

Conversely, if \( X \) is a determining set, \( \pi^X \leq \pi^V \), so

\[ \mathcal{H}(y \mid X) = \mathcal{H}(\pi^V \cap \pi^X) - \mathcal{H}(\pi^X) = 0, \]

because \( \pi^V \cap \pi^X = \pi^X \).
Example 3.6. Let \( f: 3^2 \rightarrow 3 \) be the partial function specified by Table 2. The values of \( \mathcal{H}(y \mid U) \) for the subsets \( U_1 = x_1 \) and \( U_2 = x_2 \) indicates that \( U_1 \) is a determining set for \( f \), while \( U_2 \) is not.

\[
\begin{align*}
\mathcal{H}(y_{x_1}) &= -3 \cdot \frac{2}{6} \log \frac{2}{6} = -3 \cdot \log \frac{1}{3} = 1.585 \\
\mathcal{H}(x_1) &= -3 \cdot \frac{2}{6} \log \frac{2}{6} = -3 \cdot \log \frac{1}{3} = 1.585 \\
\mathcal{H}(y_{x_1}) &= \mathcal{H}(y, x_1) - \mathcal{H}(x_1) = 0 \\
\mathcal{H}(y_{x_2}) &= -6 \cdot \log \frac{1}{6} = -6 \cdot \log \frac{1}{6} = 2.585 \\
\mathcal{H}(x_2) &= -\left( \frac{3}{6} \log \frac{3}{6} + \frac{2}{6} \log \frac{2}{6} + \frac{1}{6} \log \frac{1}{6} \right) = 1.46 \\
\mathcal{H}(y_{x_2}) &= \mathcal{H}(y_{x_2}) - \mathcal{H}(x_2) = 2.585 - 1.46 = 1.125.
\end{align*}
\]

4. An Algorithm for Mining MDSs Using Entropy

The algorithm uses the conditional entropy \( \mathcal{H}(y \mid X) \) described in the previous section to find determining sets. It starts evaluating \( \mathcal{H}(y \mid X) \) for single element subsets \( X = x \) and increases the size of the subsets with each successive iteration. In this way, all possible subsets of variables with size \( \alpha \) are evaluated before any subset of \( S \) with size \( \beta > \alpha \). The algorithm is not redundant because it does not evaluate \( \mathcal{H}(y \mid X) \) for a subset of variables \( X \) more than once. For instance, if both subsets \( X_1 = \{1\} \) and \( X_2 = \{2\} \) are expanded in a subsequent iteration, then the subset \( x_3 = \{1,2\} \) common to both expansions will be evaluated once.

The proposed algorithm takes as input a partially defined function \( f \) and a limiting factor \( \ell \) in the range (0,1] used to reduce the search space. The output is a collection of determining sets for \( f \). The algorithm performs a search on the
power-set of the set of variables \( V = x_1, x_2, \ldots, x_n \) of \( f \). The limiting factor \( \ell \) defines the subsets of variables that are expanded among all subsets within a given size. Namely, \( \ell \) corresponds to a fraction of subsets \( X \) with equal size and lowest \( \mathcal{H}(y \mid X) \) value. When the limiting factor equals 1, all the possible subsets are evaluated until one or more solution sets with equal size are found. The search stops when \( \mathcal{H}(y \mid X) = 0 \) for all the possible subsets \( X \) with a given size; these sets are referred to as determining sets for \( f \). The minimum number found corresponds to the size of the first solution set since the search proceeds with increasing order of the subsets size and all the remaining subsets that have not been checked have a greater size.

### Table 3

**Partially Defined Function Example**

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example 4.1.** Table 3 shows another example of a partially defined function. The values of conditional entropies involved are given by:

\[
\begin{align*}
\mathcal{H}(y \mid x_1) &= \mathcal{H}(y, x_1) - \mathcal{H}(x_1) = 7.229, \\
\mathcal{H}(y \mid x_2) &= \mathcal{H}(y, x_2) - \mathcal{H}(x_2) = 12.381, \\
\mathcal{H}(y \mid x_3) &= \mathcal{H}(y, x_3) - \mathcal{H}(x_3) = 6.703, \\
\mathcal{H}(y \mid x_4) &= \mathcal{H}(y, x_4) - \mathcal{H}(x_4) = 7.229, \\
\mathcal{H}(y \mid x_3, x_4) &= \mathcal{H}(y, x_3, x_4) - \mathcal{H}(x_3, x_4) = 0, \\
\mathcal{H}(y \mid x_4, x_1) &= \mathcal{H}(y, x_4, x_1) - \mathcal{H}(x_4, x_1) = 0. 
\end{align*}
\]

When the limiting factor takes a value equal to 0.25, the algorithm only indicates \( x_1, x_3 \) as a determining set for the partially defined function presented in Table 3. Note that \( x_1, x_4 \) is also a determining set for the given function and is part of the solution when the limiting factor takes a value that is at least 0.5.

Table 4 and Table 5 present the methods and variables used in the algorithm.
### Table 4

**Methods for Computing** $\text{MDS}(f, \ell)$

<table>
<thead>
<tr>
<th>Methods</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ADD}(L, X)$</td>
<td>Inserts set $X$ in the list of sets $L$</td>
</tr>
<tr>
<td>$\text{REMOVE}(L, idx)$</td>
<td>Removes an element(set) at index $idx$ from the list of sets $L$</td>
</tr>
<tr>
<td>$\text{CLEAR}(L)$</td>
<td>Removes all elements(sets) from the list of sets $L$</td>
</tr>
<tr>
<td>$\text{NEXT}(L)$</td>
<td>Gets the next element(set) of the list of sets $L$</td>
</tr>
<tr>
<td>$\text{GET}(L, idx)$</td>
<td>Gets the element(set) at index $idx$ on the list of sets $L$</td>
</tr>
<tr>
<td>INDEX_MIN($E$)</td>
<td>Gets the index of the first element on the list $E$ with lowest conditional entropy value</td>
</tr>
<tr>
<td>SIZE($X$)</td>
<td>Gets the size of the set $X$</td>
</tr>
<tr>
<td>$\text{COLLECTIONADD}(\mathcal{D}, X)$</td>
<td>Inserts set $X$ in the collection of determining sets $\mathcal{D}$</td>
</tr>
<tr>
<td>$\text{BINOMIAL}(l, p)$</td>
<td>Evaluates the binomial coefficient $l$ chooses $p$, which will be multiplied by the limiting factor $\ell$ in order to find the fraction of sets of variables with lowest entropy with a certain size $p$; it takes into consideration the size $l$ of the complete set of variables of $f$</td>
</tr>
<tr>
<td>$\text{COMPUTE_ENTROPY}(X)$</td>
<td>Evaluates the entropy value of a set of variables $X$</td>
</tr>
</tbody>
</table>

### Table 5

**Variables for Computing** $\text{MDS}(f, \ell)$

<table>
<thead>
<tr>
<th>Variables</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>The set of variables of a partially defined function $f$</td>
</tr>
<tr>
<td>SETLIST</td>
<td>A list of sets of variables</td>
</tr>
<tr>
<td>LOW_ENTROPY_SET_LIST</td>
<td>A list of sets of variables with the lowest entropy values among the ones in SETLIST</td>
</tr>
<tr>
<td>ENTROPY_LIST</td>
<td>A list of entropy values</td>
</tr>
<tr>
<td>$\ell$</td>
<td>A number in $\mathcal{0,1}$ corresponding to the limiting factor $\ell$</td>
</tr>
<tr>
<td>$\mathcal{D}$</td>
<td>A collection of determining sets</td>
</tr>
<tr>
<td>set_size</td>
<td>The size of a subset</td>
</tr>
<tr>
<td>dset_found</td>
<td>Indicates whether at least one determining set of variables was found</td>
</tr>
<tr>
<td>var</td>
<td>A variable of $f$</td>
</tr>
<tr>
<td>$X$</td>
<td>A subset of variables of $S$</td>
</tr>
<tr>
<td>entropy</td>
<td>An entropy value</td>
</tr>
<tr>
<td>$lf$</td>
<td>A fraction of the total number of sets of variables with a certain size that have lowest entropy values</td>
</tr>
<tr>
<td>index</td>
<td>An index of a list</td>
</tr>
</tbody>
</table>
**Input:** A partially defined function and a limiting factor

**Result:** A collection \( D \) of determining variables sets

1 begin
2 \( D \leftarrow \emptyset \)
3 \( \text{set}_\text{size} \leftarrow 1 \)
4 \( \text{dset}_\text{found} \leftarrow \text{false} \)
5 foreach \( \text{var} \in S \) do
6 \( X \leftarrow \text{var} \)
7 \( \text{ADD}(	ext{SETLIST}, X) \)
8 \( \text{entropy} \leftarrow \text{COMPUTE_ENTROPY}(X) \)
9 \( \text{ADD}(	ext{ENTROPY}_\text{LIST}, \text{entropy}) \)
10 if \( \text{entropy} = 0 \) then
11 \( \text{COLLECTIONADD}(D, X) \)
12 \( \text{dset}_\text{found} \leftarrow \text{true} \)
13 while \( \text{set}_\text{size} \leq \text{SIZE}(S) \) and \( \text{dset}_\text{found} = \text{false} \) do
14 \( l f \leftarrow l \times \text{BINOMIAL}(	ext{SIZE}(S), \text{set}_\text{size}) \)
15 repeat
16 \( \text{index} \leftarrow \text{INDEX_MIN}(	ext{ENTROPY}_\text{LIST}) \)
17 \( X \leftarrow \text{GET}(	ext{SETLIST}, \text{index}) \)
18 \( \text{ADD}(	ext{LOW_ENTROPY_SETLIST}, X) \)
19 \( \text{REMOVE}(	ext{SETLIST}, \text{index}) \)
20 \( \text{REMOVE}(	ext{ENTROPY}_\text{LIST}, \text{index}) \)
21 \( l f \leftarrow l f - 1 \)
22 until \( l f = 0 \)
23 \( \text{CLEAR}(	ext{SETLIST}) \)
24 \( \text{CLEAR}(	ext{ENTROPY}_\text{LIST}) \)
25 \( \text{set}_\text{size} \leftarrow \text{set}_\text{size} + 1 \)
26 while \( \text{LOW_ENTROPY_SETLIST} \neq \emptyset \) do
27 \( X \leftarrow \text{NEXT}(	ext{LOW_ENTROPY_SETLIST}) \)
28 foreach \( \text{var} \in S \) do
29 if \( X \cup \text{var} \ni \text{SETLIST} \) and \( \text{var} \ni X \) then
30 \( \text{ADD}(	ext{SETLIST}, X \cup \text{var}) \)
31 \( \text{entropy} \leftarrow \text{COMPUTE_ENTROPY}(X \cup \text{var}) \)
32 \( \text{ADD}(	ext{ENTROPY}_\text{LIST}, \text{entropy}) \)
33 if \( \text{entropy} = 0 \) then
34 \( \text{COLLECTIONADD}(D, X \cup \text{var}) \)
35 \( \text{dset}_\text{found} \leftarrow \text{true} \)
36 end
37 end

Fig. 1 - Computing \( \text{MDS}(f, l) \).

The core of the algorithm is the function \( \text{COMPUTE_ENTROPY}(f, X) \) that has as input arguments a partial function \( f \) and a set of variables \( X \). This function returns the Shannon Entropy corresponding to \( X \). The function is
presented in Fig. 2 and Tables 6 and 7 define the methods and variables used in function COMPUTE_ENTROPY($f$, $X$).

Table 6

<table>
<thead>
<tr>
<th>Methods</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_XKEYS($v$, $X$)</td>
<td>Gets the values assigned in a registered vector $v$ of a partial function $f$ to the set of variables $X$</td>
</tr>
<tr>
<td>GET_YXKEYS($v$, $X$)</td>
<td>Gets the values assigned in a registered vector $v$ of a partial function $f$ to the set of variables $X$ and the function value $y$</td>
</tr>
<tr>
<td>GET($M$, $k$)</td>
<td>Gets the number of occurrences of the key $k$ in the map $M$</td>
</tr>
<tr>
<td>ASSIGN($M$, $k$, $o$)</td>
<td>Maps the key $k$ to the number of occurrences $o$ in the map $M$</td>
</tr>
<tr>
<td>ADD($M$, $k$, $o$)</td>
<td>Inserts the mapping with key $k$ and number of occurrences $o$ to the hash structure $M$</td>
</tr>
<tr>
<td>ENTROPY($M$)</td>
<td>Evaluates the entropy of the sets of variables in map $M$</td>
</tr>
</tbody>
</table>

5. Experimental Results

We carried out experiments on a Windows Vista 64-bit machine with 8Gb RAM and 2 × Quad Core Xeon Proc E5420, running at 2.50 GHz with a 2 × 6Mb L2 cache. The algorithm was written in Java 6.

We analyze the results in terms of running time and minimum number of variables of a determining set found as a function of the number of tuples in $T_f$ and the limiting factor $l$.

A program that randomly generates comma separated text files representing partially defined functions with 8, 16 or 24 variables was developed. These values were chosen based on the experiments made in the related work of T. Sasao [1] and in [2].

One hundred files were randomly generated for each type of partially defined function (with 8, 16, and 24 variables) using an input radix $r = 3$ and an output radix $p = 5$.

Note that a totally defined function with 8 variables and $r = 3$ has $3^8 = 6561$ tuples. In our experiments, we randomly generated 1000 tuples for partially defined functions with 8 variables. For functions that depend on 16 and 24 arguments we generated 5000 tuples because the number of tuples for completely defined functions with 16 or 24 variables is much higher.
Input: A partially defined function, a subset $X$ of the complete set of variables of the partially defined function given

Output: An entropy value

begin
  foreach $v \in F$ do
    $keyX \leftarrow$ GET_XKEYS($v$, $X$)
    if $keyX \in MAP_X$ then
      $val \leftarrow$ GET(MAP_X, $keyX$)
      ASSIGN(MAP_X, $keyX$, $val + 1$)
    else
      ADD(MAP_X, $keyX$, 1)
      $keyYX \leftarrow$ GET_XYKEYS($v$, $X$)
      if $keyYX \in MAP_YX$ then
        $val \leftarrow$ GET(MAP_YX, $keyYX$)
        ASSIGN(MAP_YX, $keyYX$, $val + 1$)
      else
        ADD(MAP_YX, $keyYX$, 1)
        $entropyX \leftarrow$ ENTROPY(MAP_X)
        $entropyYX \leftarrow$ ENTROPY(MAP_YX)
        $entropy \leftarrow$ $entropyYX - entropyX$
    end
  end
  return $entropy$

Fig. 2 – COMPUTE_ENTROPY($f$, $X$).

<table>
<thead>
<tr>
<th>Variables</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MAP_X$</td>
<td>A hash structure that stores objects containing a key corresponding to the values assigned to the set of variables $X$ and the number of occurrences of those keys in the map</td>
</tr>
<tr>
<td>$MAP_YX$</td>
<td>A hash structure that stores objects containing a key corresponding to the function values - $y$ - as well as the corresponding values assigned to the set of variables $X$, and the number of occurrences of those keys in the map</td>
</tr>
<tr>
<td>$X$</td>
<td>A subset of the complete set of variables of a partially defined function</td>
</tr>
<tr>
<td>$v$</td>
<td>A registered vector from a partially defined function $f$</td>
</tr>
<tr>
<td>$F$</td>
<td>A partially defined function</td>
</tr>
<tr>
<td>$keyX$</td>
<td>The values assigned in $v$ to the variables in $X$</td>
</tr>
<tr>
<td>$keyYX$</td>
<td>The values assigned in $v$ to $y$ - the function value – and the variables in $X$</td>
</tr>
<tr>
<td>$val$</td>
<td>The number of occurrences of a key in a map structure</td>
</tr>
<tr>
<td>$entropyX$</td>
<td>Entropy of a set of variables</td>
</tr>
<tr>
<td>$entropyYX$</td>
<td>Entropy of function values and the corresponding set of variables</td>
</tr>
<tr>
<td>$entropy$</td>
<td>Entropy corresponding to the set of variables $s$</td>
</tr>
</tbody>
</table>
In the experiments, we evaluate the performance of the algorithm with a varying number of tuples and limiting factor. By Theorem 2.2, if \((f_1,f_2,\ldots,f_k)\) is a sequence of functions such that

\[ f_1 \subseteq f_2 \subseteq \ldots \subseteq f_k, \]

we have

\[ DS(f_k) \subseteq \ldots \subseteq DS(f_2) \subseteq DS(f_1). \]

In other words, when we start with a partial function \(f_1\) with a small specification table \(T_{f_1}\) and we expend sequentially the specification of the functions, the number of determining sets will decrease. The experiments compare the results for files with 8, 16 and 24 variables and they contain averages of the values corresponding to time and number of variables the function depends on as a function of the number of tuples and limiting factor. In our case, \(k \in \{10, 20, 30, 50, 90, 100, 200\}\). The averages are evaluated over 100 functions within each group of generated functions (8, 16 and 24 variables).

As shown in Figs. 3, 4 and 5, the running time increases with the number of tuples because in most cases, the larger the subset of variables \(X\), the greater is the conditional entropy \(H(y|X)\). Likewise, the running time increases with the limiting factor \(\ell\) since the search space increases as \(\ell\) increases. Also, the time increases exponentially with the number of variables. It is clear that the number of subsets evaluated during the search depends on the original number of variables of a partial function \(f_1\).

![Time / Limiting Factor - 8 Variables](image)

Fig. 3 – Dependency of average time on number of tuples and limiting factor for 8 variables.
Finally, Figs. 6, 7 and 8 show that the number of variables the function depends on is related to the number of tuples $k$. As $k$ increases, the constraints imposed on the problem become more extensive, and the number of variables that determines the value of the function increases. Notice however that the limiting factor $l$ has a remarkable insignificant contribution to the number of variables that a partial function depends on provided by the algorithm. Therefore, the algorithm provides solutions that are optimal or very close to
optimal, even when $\ell$ takes an extremely low value. This is an important result since as previously mentioned, the running time for the algorithm to find a solution is highly affected by the limiting factor chosen.

Fig. 6 – Average size of minimal determining set for 8 variables, as a function of the number of tuples and limiting factor.

Fig. 7 – Average size of minimal determining set for 16 variables, as a function of the number of tuples and limiting factor.
This paper introduces an algorithm that seeks to identify the sets of variables that determine a partially defined function. The algorithm uses a clustering technique for discrete functions starting from the conditional entropy that measures the discrepancy between the kernel partitions of these functions. A limiting factor is used to reduce the search space. The search and evaluation proceed in increasing order of subsets size. When the limiting factor value equals 1 all subsets of the power set of the sets of variables of the partially defined function are evaluated, until the determining sets with smallest size are found. In this case the algorithm determines the minimum number of variables on which a partially defined function depends on, as well as all sets of variables with minimum number of elements that define the function. A limiting factor that takes values between 0 and 1 generates a search that is not complete but significantly faster depending on the factor chosen. Nevertheless, the effect caused by the limiting factor on the optimality of the algorithm is remarkably small. The use of a small limiting factor improves the running time to find a solution while keeping the solution optimal or close to an optimal solution. We believe that the algorithm will be helpful for digital circuit design since it allows determining the possible sets of variables on which a partial function depends starting from a tabular specification of the function.

Received: June 5, 2009

University of Massachusetts, MA, Boston, USA,
Department of Computer Science
e-mails: dsim@cs.umb.edu
dpletea@cs.umb.edu

Fig. 8 – Average size of minimal determining set for 24 variables, as a function of the number of tuples and limiting factor.
REFERENCES


IDENTIFICAREA MULŢIMILOR DETERMINANTE PENTRU FUNŢII PARŢIAL DEFINITE FOLOSIND ENTROPIA

(Rezumat)

În această lucrare se prezintă un algoritm pentru identificarea mulţimilor minimale de variabile care determină valorile unei funcţii discrete, parţial definite. Algoritmul este bazat pe notiunea de entropie a unei particii. Se introduce notiunea de factor de limitare, pentru a restricţiona spaţiul soluţiilor şi a controla timpul de execuţie a algoritmului. Rezultatele experimentale demonstrează eficienţa algoritmului pentru funcţii care depind de până la 24 de variabile. Totodată, este studiat şi efectul factorului de limitare asupra optimalităţii algoritmului pentru mărimi diferite ale specificărilor funcţiilor parţial definite.