Calculating minimum \( k \)-unsafe and maximum \( k \)-safe sets of variables for disclosure risk assessment of individual records in a microdata set

Akimichi Takemura
Faculty of Economics, University of Tokyo

January, 1999

Abstract

In the framework of disclosure control of a microdata set, an unique record is at risk of being identified. Even if a record is not unique in the microdata set, it may be considered risky if the frequency \( k \) of the cell, in which the record falls, is small. The notion of minimum unsafe combination introduced by Willenborg and de Waal (1996) is important in this respect. The purpose of this paper is to clearly define closely related notions and give an algorithm for obtaining relevant combinations of variables. We will define minimum \( k \)-unsafe and maximum \( k \)-safe sets of variables for each record and give an illustration to show the usefulness of the proposed technique.

Key words: Hasse diagram, hitting set problem, local suppression, NP-complete, sample unique.

1 Introduction

Consider a microdata set of \( n \) individuals and \( p \) variables. An unique individual of the microdata set (sample unique) is at risk of being identified if the individual happens to be a population unique as well. A sample unique individual is more likely to be a population unique, if he (or she) is already a sample unique with only a small number of variables out of \( p \) variables. Therefore it is important to find minimum set of variables, with which the individual becomes a sample unique.

It is also of interest to find maximum set of variables, with which an unique individual is no longer unique. This maximum set is obtained if we delete or locally suppress minimum number of variables to make the individual non-unique. Therefore this maximum set is important from the viewpoint of disclosure control based on local suppression.

In addition to uniqueness, we consider doubles, triples, etc. If an individual falls in a cell with frequency \( j \), we call him (or her) \( j \)-isolated. Suppose that we fix \( k \geq 1 \) and consider up to \( k \)-isolated individuals as being at identification risk. We call an individual \( k \)-unsafe (\( k \)-safe) if he is \( j \)-isolated with \( j \leq k \) (\( j > k \)). For a \( k \)-unsafe individual we will consider minimum \( k \)-unsafe set and maximum \( k \)-safe set of variables.
The notion of minimum unsafe combination was introduced and discussed in Section 5.4 of Willenborg and de Waal (1996) and further in Willenborg (1996). However in these papers the distinction between minimum unsafe set and maximum safe set was not necessarily clear. The first purpose of this paper is to define these notions more clearly and derive some relations between them. Another purpose of this paper is to give an algorithm and a working program to obtain minimum unsafe and maximum safe sets. We will show that these sets of variables can be computed in a reasonable amount of time even for a large data set.

In this paper we only discuss obtaining minimum unsafe and maximum safe sets for each unsafe record of a microdata set. Essentially we consider each unsafe record separately. This is another difference of our approach from the approach of Willenborg and de Waal (1996) and Willenborg (1996), where unsafeness of the whole data set is more emphasized. As these authors discuss, applying local suppression to unsafe records and evaluating the disclosure risk of resulting data set involves much more complicated optimization problem. Furthermore in this paper we are only concerned with rare records in the sample and do not discuss the notion of population uniques or rare individuals in the population. Some relevant results on estimation of the number of population uniques are given in Takemura (1997) and Hoshino and Takemura (1998).

The organization of this paper is as follows. In Section 2 we give a definition of minimum $k$-unsafe and maximum $k$-safe sets of variables and derive several relations between them. In Section 3 we present an algorithm for obtaining these sets. In Section 4 we apply our algorithm to a data set of considerable size and show that calculations can be done in reasonable amount of time. In Appendix B we present the source code of a working program, which was used for processing the data set of Section 4.

2 Minimum $k$-unsafe and maximum $k$-safe sets of variables

Suppose that a microdata set is given in the form of an $n \times p$ matrix $X$. Each row of $X$ corresponds to a record of $p$ observations of an individual. We refer to the columns of $X$ alternatively as “variables” or “fields”.

Let $\mathcal{J} = \{1, \ldots, p\}$ denote the set of variables. For a subset $J \subset \mathcal{J}$ denote the submatrix of $X$ consisting of the columns in $J$ by

$$X_J = (x_{ij}), \ j \in J.$$ 

Suppose that the $i$-th row of $X$ is an unique record of $X$. If it is already unique in $X_J$, we call the $i$-th record unique with respect to $J$ and call $J$ 1-unsafe for the $i$-th record. Generalizing this to $k > 1$, consider a $k$-unsafe record $i$. If it is already $k$-unsafe in $X_J$, we call the $i$-th record $k$-unsafe with respect to $J$ and call $J$ $k$-unsafe for the $i$-th record.

If $J$ is $k$-unsafe for $i$, then any superset $J'$ of $J$ ($J' \supset J$) is $k$-unsafe for $i$. On the other hand if $J$ is $k$-safe for $i$, then any subset $J'$ of $J$ ($J' \subset J$) is $k$-safe for $i$. Therefore we are naturally led to the following definition.
Definition 2.1. Fix a particular $k$-unsafe record $i$ and consider $k$-safeness of sets $J$ for $i$. A set $J \subset \mathcal{J}$ is minimal $k$-unsafe if $J$ is $k$-unsafe and any proper subset $J'$ of $J$ is $k$-safe. $J$ is minimum $k$-unsafe if it is minimal $k$-unsafe and its size $|J|$ is the smallest among minimal $k$-unsafe subsets.

A set $J \subset \mathcal{J}$ is maximal $k$-safe if $J$ is $k$-safe and any proper superset $J'$ of $J$ is $k$-unsafe. $J$ is maximum $k$-safe if it is maximal $k$-safe and its size $|J|$ is the largest among maximal $k$-safe subsets.

In Definition 2.1, all non-empty subsets of $\mathcal{J}$ may be $k$-unsafe for $i$. In this case minimal unsafe sets are the singletons $\{1\}, \ldots, \{p\}$, and maximal safe set is the empty set $\emptyset$.

We denote the size of minimum $k$-unsafe set for $i$ by $u_k(i)$ and the size of maximum $k$-safe set for $i$ by $s_k(i)$.

A convenient way of thinking of $k$-safe and $k$-unsafe sets is given in terms of an indicator function on a partially ordered set. Let $2^J$ denote the set of all subsets of $\mathcal{J}$:

$$2^J = \{\emptyset, \{1\}, \ldots, \{p\}, \{1, 2\}, \ldots, \mathcal{J}\}.$$ 

$2^J$ forms a partially ordered set with respect to the inclusion relation between subsets of $\mathcal{J}$. Let $\phi_i$ denote the indicator function of $k$-unsafeness of sets of variables for $i$:

$$\phi_i(J) = \begin{cases} 1, & \text{if } J \text{ is } k\text{-unsafe for } i, \\ 0, & \text{if } J \text{ is } k\text{-safe for } i. \end{cases}$$

Then $\phi_i$ is a non-decreasing function on $2^J$. We are considering minimal elements of $\phi_i^{-1}(1)$ and maximal elements of $\phi_i^{-1}(0)$.

In the following we will make extensive use of the Hamming distance. The Hamming distance between the $i$-th row and the $i'$-th row of $X$ is defined by

$$d(i, i') = \sum_{j=1}^{p} I(x_{ij} \neq x_{i'j}),$$

where $I(\cdot)$ denotes the indicator function. We also use the Hamming distance $d_J(i, i')$ between rows of submatrix $X_J$:

$$d_J(i, i') = \sum_{j \in J} I(x_{ij} \neq x_{i'j}).$$

For a particular individual $i$ we denote the Hamming distance to the nearest neighbor by

$$h_1(i) = \min_{i' \neq i} d(i, i').$$

Furthermore let $h_k(i)$ denote the Hamming distance to the $k$-th nearest neighbor:

$$h_k(i) = \text{the } k\text{-th smallest value in } \{d(i, i'), \ i' \neq i\}. \quad (2)$$

For $J \subset \mathcal{J}$ we define $h^J_1(i)$ and $h^J_k(i)$ based on $d_J(i, i')$ in an analogous manner.

Now we derive several relations between the above quantities.
Proposition 1. For k-unsafe \( i \) 

\[ u_k(i) \leq s_k(i) + 1, \]  

where the equality holds iff there exists \( m \) ( \( 0 < m \leq p \) ) such that \( J \) is k-unsafe iff \(|J| \geq m\).

**Proof.** By definition of \( s_k(i) \), any set of variables of size \( s_k(i) + 1 \) is k-unsafe for \( i \). Therefore we obviously have (3).

Now suppose that there exists \( m \) of the proposition. Then \( s_k(i) < m \) and \( u_k(i) \geq m \). Therefore \( s_k(i) < u_k(i) \) or \( s_k(i) + 1 \leq u_k(i) \). Together with (3) this implies \( s_k(i) + 1 = u_k(i) \). Conversely suppose that \( s_k(i) + 1 = u_k(i) = m \) holds. Then for any k-safe \( J \) and any k-unsafe \( J' \),

\[ |J| \leq s_k(i) < m = u_k(i) \leq |J'|. \]

To illustrate Proposition 1 we consider two simple examples.

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 a A</td>
</tr>
<tr>
<td>1 a B</td>
</tr>
<tr>
<td>1 a B</td>
</tr>
</tbody>
</table>

In Table 1 the first row \((i = 1)\) is unique \((k = 1)\). It is already unique with respect to the third column and \( u_1(1) = 1 \). On the other hand \( J = \{1, 2\} \) is the maximum 1-safe set and \( s_1(1) = 2 \). Therefore in Table 1 \( u_1(1) = 1 < s_1(1) + 1 = 3 \). It is instructive to draw the Hasse diagram of the partially ordered set \( 2^J \) (omitting the empty set) for illustrating 1-safeness of the fist row of Table 1. In Figure 1 we clearly see that \{3\} is minimum 1-unsafe set and \{1, 2\} is the maximum 1-safe set for the first row of Table 1.

Figure 1: Hasse diagram for Table 1

```
123
\!\!

12 OK
13 !
23 !

1 OK
2 OK
3 !
```

To illustrate Proposition 1 we consider two simple examples.

Next, consider Table 2. The first row \((i = 1)\) is unique \((k = 1)\). Table 2 consists of 0’s or 1’s only. Since we are concerned only whether the observations of the first row coincide
with those of other rows, without loss of generality we can let the first row \((0, 0, \ldots, 0)\) and the other rows consist of 0’s and 1’s. In Table 2 any 3-element set of variables is minimum 1-unsafe and \(u_1(1) = 3\). Any 2-element set is maximum 1-safe and \(s_1(1) = 2\) Therefore in Table 2 the equality in (3) holds with \(m = 3\) in Proposition 1. This situation is clearly understood by looking at the Hasse diagram of Figure 2.

<table>
<thead>
<tr>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 0 0</td>
</tr>
<tr>
<td>1 0 1 0</td>
</tr>
<tr>
<td>1 0 0 1</td>
</tr>
<tr>
<td>0 1 1 0</td>
</tr>
<tr>
<td>0 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
</tr>
</tbody>
</table>

Figure 2: Hasse diagram for Table 2

It is easy to give an example of general size along the lines of Table 2. Consider a table with 

\[ n = \binom{p}{l} + 1 \]

rows, whose first row is \((0, 0, \ldots, 0)\) and other rows have 1’s in fields corresponding to \(\binom{p}{l}\) ways of choosing \(l\) fields out of \(p\) fields. Considering the symmetry among the columns it is easily seen that \(u_1(1) = p - l + 1, s_1(1) = p - l\) and the equality in (3) holds with \(m = p - l + 1\). Furthermore if we repeat each row \(k\) times except for the first row and consider a table with \(n = k \times \binom{p}{l} + 1\) rows, we have an example where \(u_k(1) = p - l + 1, s_k(1) = p - l\).

Now for the case \(k = 1\), we derive a relation between maximum 1-safe set and nearest neighbor in the Hamming distance.

**Proposition 2**

\[ h_1(i) = p - s_1(i). \] (4)
Proof. Let \( c = s_1(i) \). Then there exists a set of \( c \) variables \( J \) such that the individual \( i \) is not unique with respect to \( J \). Hence there exists another individual \( i' \), whose observations coincide with those of the \( i \)-th individual for \( c \) variables in \( J \). Then \( d(i, i') \leq p - c \). This implies that \( h_1(i) \leq p - s_1(i) \).

Next let \( h_1(i) = d \). Then there exists another individual \( i' \) such that \( i \) and \( i' \) have the same observations on \( p - d \) variables. Hence \( i \) is not unique with respect to these variables and we have \( s_1(i) \geq p - d = p - h_1(i) \) or \( h_1(i) \geq p - s_1(i) \).

Combining above two inequalities we obtain \( h_1(i) = p - s_1(i) \).

Proposition 2 is easily checked in Table 1 and Table 2. Indeed \( h_1(1) = 1 = 3 - s_1(1) \) in Table 1 and \( h_1(1) = 2 = 4 - s_1(2) \) in Table 2.

The equality in Proposition 2 does not generalize to \( k > 1 \). We only have the following inequality.

**Proposition 3**

For \( k > 1 \)

\[
h_k(i) \leq p - s_k(i).
\]

**Proof.** Let \( c = s_k(i) \). Then there exists a subset \( J \) of size \( c = |J| \) and \( k \) rows \( i_1, \ldots, i_k \) (\( i_t \neq i \)) such that the rows \( i, i_1, \ldots, i_k \) share same observations for the variables of \( J \). Therefore \( d(i, i_t) \leq p - c, \ t = 1, \ldots, k \) and \( h_k(i) \leq p - c \).

It is easily seen that for \( k > 1 \) the equality in (5) does not usually hold.

### 3 Algorithm for computing minimum \( k \)-unsafe sets and maximum \( k \)-safe sets

Here we discuss how to calculate minimum \( k \)-unsafe and maximum \( k \)-safe sets of variables. Actually we shall discuss how to determine \( k \)-safeness or \( k \)-unsafeness of all subsets \( J \) of \( J \) and propose two algorithms for doing this. One reason for determining \( k \)-safeness of all subsets (rather than just minimum \( k \)-unsafe and maximum \( k \)-safe sets) is that except for maximum 1-safe set, which can be determined by obtaining the nearest neighbor in the Hamming distance (Proposition 2), it seems to be no more difficult to determine \( k \)-safeness of all subsets \( J \) than to obtain minimum \( k \)-unsafe and maximum \( k \)-safe sets only. This conjecture is based on the fact that obtaining minimum 1-unsafe set is an NP-complete problem as \( n \) and \( p \to \infty \). See Appendix A for detail.

**Step B:**

We first describe the obvious algorithm for extracting minimum \( k \)-unsafe, minimal \( k \)-unsafe, maximum \( k \)-safe and maximal \( k \)-safe sets from the complete list of \( k \)-safeness of all subsets of \( J \). We prepare a list of unchecked subsets, initially set to the list of all subsets of \( J \). Let \( m = |J| \) denote the size of subsets of \( J \). In order to list minimum \( k \)-unsafe set we start with \( m = 1 \) and increase \( m \) until we find a \( k \)-unsafe set with smallest \( m = m^* \). \( k \)-unsafe sets of size \( m^* \) are minimum \( k \)-unsafe sets. Delete all supersets of minimum \( k \)-unsafe sets from the list of unchecked subsets. Then increase \( m \) further until
we find another $k$-unsafe set in the list with smallest $m = m^**$. $k$-unsafe sets of size $m^**$ in
the list are (not minimum but) minimal $k$-unsafe sets. Delete all supersets of minimal $k$-
unsafe sets from the list of unchecked subsets. Repeating the above process by increasing
$m$ further, we can extract all minimal $k$-unsafe subsets. Extracting all maximum and
maximal $k$-safe sets is entirely analogous starting with $m = p$ downwards instead.

Now we present two algorithms for determining $k$-safeness of all subsets of $\mathcal{J}$. The
first algorithm is deterministic and the second one is probabilistic.

\textbf{Step A1:}

For the first algorithm we make use of the Hamming distance $d(i, i')$. Again we prepare
a one-dimensional list of unchecked subsets, initially set to the list of all subsets of $\mathcal{J}$.
For definiteness we order the subsets $J$ of $\mathcal{J}$ such that if $|J| > |J'|$ then $|J|$ precedes $|J'|$
in the list. We compute the Hamming distance $h_k(i)$ to the $k$-th nearest neighbor from
the row $i$. Proposition 3 implies that $J$ is $k$-unsafe if $|J| > p - h_k(i)$. Therefore we flag
all $J$ with $|J| > p - h_k(i)$ as $k$-unsafe and delete them from the list of unchecked subsets.

Now we check subsets of size $p - h_k(i)$. For $k = 1$ $J$ is 1-safe (actually maximum 1-safe
at this stage) iff there exists $i'$ with $d(i, i') = h_1(i)$ and

$$J = \{j \mid x_{ij} = x_{i'j}\}.$$ 

Therefore for $k = 1$ we can easily determine 1-safeness of $J$ by just checking all $i'$ with
d($i, i') = h_1(i)$. For $k > 1$ we need to check each $J$ of size $p - h_k(i)$ separately whether it
is $k$-unsafe or not by actually forming the subset $X_J$.

If $J$ of size $p - h_k(i)$ is $k$-safe, we flag $J$ and all its subsets as $k$-safe and delete them
from the list of unchecked subsets. Otherwise we form submatrix $X_J$ and compute $h_k^J(i)$. Then a subset $J' \subset J$ is $k$-unsafe if $|J'| > |J| - h_k^J(i)$ (including $J$ itself). We flag these
subset as $k$-unsafe and delete them from the list of unchecked subsets.

Now we can proceed as follows. We consider the first unchecked subset $J$ of the list of
unchecked subsets. We determine whether $J$ is $k$-safe or not. If $J$ is $k$-safe we flag $J$ and
all its subsets as $k$-safe and delete them from the list of unchecked sets. If $J$ is unsafe, we
form a submatrix $X_J$ and compute $h_k^J(i)$. We flag all $J' \subset J$ with $|J'| > |J| - h_k^J(i)$ as
$k$-unsafe and delete them from the list of unchecked subsets. Repeating this process, we
can determine $k$-safeness of all subsets of $\mathcal{J}$.

\textbf{Step A2:}

As a second algorithm we propose the following simple randomized algorithm. There are $2^p - 1$ nonempty subsets of $\mathcal{J}$. We randomly choose one of them and determine its
$k$-safeness. If it is $k$-safe, then all its subsets are $k$-safe. On the other hand if it is $k$-
unsafe, then all its supersets are $k$-unsafe. In either case we delete those sets from the list
of unchecked subsets. From the remaining unchecked subsets, we randomly choose one
and determine its $k$-safeness. Repeating this process, we can determine $k$-safeness of all
subsets of $\mathcal{J}$.
4 An illustration

Here we apply our algorithm to a fairly large data set and confirm that computation can be done in a reasonable amount of time. The data set is obtained from “The American Community Survey” page of U.S. Census Bureau home page. We downloaded PUMS (Public Use Microdata Samples) file of population records data of Ohio for 1997, which contained 17142 individuals. The data file contains a large number of variables, but we chose 7 variables which may be used as key variables for individual identification. These 7 variables are 1. RELT (Relationship), 2. SEX, 3. RACE, 4. AGE, 5. MARITAL (Marital status), 6. ROWNCHL (Own child), and 7. RAGECHL (Presence and age of own children). For detailed description of these variables, refer to the home page of The American Community Survey. In summary we worked with a data matrix of size 17142×7. The first 10 lines of this matrix are as follows.

| 00 1 01 31 5 0 0 |
| 00 2 02 23 5 0 1 |
| 02 2 02 02 5 1 0 |
| 00 1 01 29 1 0 0 |
| 01 2 01 26 1 0 4 |
| 00 1 01 25 5 0 0 |
| 10 2 01 26 5 0 4 |
| 00 2 02 22 5 0 1 |
| 02 1 02 02 5 1 0 |
| 03 1 02 17 5 0 0 |

Out of these 17142 individuals, 1721 (10.04%) were sample uniques and 896 (5.23%) were 2-isolated. The machine used to measure the processing time is equipped with Intel Pentium Pro processor and 64 MB of memory.

It took 770 CPU seconds to obtain minimal 1-unsafe and maximal 1-safe sets for 1721 sample uniques. The first few lines of the output are as follows.

```
row: n=17142, p=7, k=1, file=ohiotest.dat
MU: Minimal Unsafe sets MS: Maximal Safe sets
30: MU: 1000100 1011000 MS: 0111111 1101011 1110011
50: MU: 1011000 MS: 0111111 1101111 1110111 1111110
```

Note that sets of variables are represented by bit patterns in this output. For example 1000100 denotes the set \{1,5\}. Therefore, for example, the record No.30 is a sample unique and its minimal unsafe sets are \{1,5\} and \{1,3,4\}, with the former being the minimum unsafe set. The maximum safe set for the record No.30 is \{2,3,4,5,6,7\}.

Computation for 2-safeness for the same data set took 1246 CPU seconds for processing 2617 2-unsafe records. The first few lines of the output are now as follows.

```
row: n=17142, p=7, k=2, file=ohiotest.dat
MU: Minimal Unsafe sets MS: Maximal Safe sets
10: MU: 1001000 MS: 0111111 1110111
11: MU: 1001000 MS: 0111111 1110111
13: MU: 1011000 MS: 0111111 1101111 1110111
```

8
We see that although \( n \) in this case is large and the computation is fairly intensive, it can be done in a reasonable amount of time.

5 Some discussion

In Section 2 and 3 we discussed 1-safeness and more general \( k \)-safeness \((k > 1)\) in parallel. However the case \( k = 1 \) seems to be particularly simple. Here we discuss an alternative formulation of the problem, which reduces \( k \)-safeness problem to 1-safeness problem. As in Table 2 consider an \( n \times p \) data matrix \( X \) consisting 0’s or 1’s with the first row \((0,0,\ldots,0)\) and consider \( k \)-safeness of the first row. Construct an expanded data matrix \( \tilde{X} \) of size \((1 + \binom{n-1}{k}) \times p\) such that the first row of \( \tilde{X} \) is \((0,0,\ldots,0)\) and the other rows of \( \tilde{X} \) correspond to field-wise “or” of \( k \) rows out of \((x_{i1},\ldots,x_{ip}), \ i = 2,\ldots,n\). It is clear that \( k \)-safeness of the first row of \( X \) is equivalent to the 1-safeness of the first row of \( \tilde{X} \). Therefore by forming the expanded data matrix \( \tilde{X} \) the problem of \( k \)-safeness is reduced to the problem of 1-safeness. We did not take this approach because in our problem \( n \) is not small and forming \( \tilde{X} \) requires a large amount of storage.

A NP-completeness of calculating minimum 1-unsafe sets

The argument of this appendix was communicated to me by Daishin Nakamura. The following “hitting set problem” is known to be NP-complete.

Given a collection \( \mathcal{C} \) of subsets of a finite set \( S \) and a positive integer \( K \leq |S| \), determine whether there exists a subset \( S' \subseteq S \) with \( |S'| \leq K \) such that \( S' \) contains at least one element from each set of \( \mathcal{C} \).

See p.222 of Garey and Johnson (1979) and Karp (1972).

Given an instance of the hitting set problem

\[
S = \{1,\ldots,p\}, \quad \mathcal{C} = \{C_1,\ldots,C_{n-1}\}, \quad K,
\]

form an \( n \times p \) data matrix \( X = (x_{ij}) \) with the first row \((x_{11},\ldots,x_{1p}) = (0,\ldots,0)\) and the rest of the rows \((i \geq 2)\) defined by

\[
x_{ij} = \begin{cases} 
1, & \text{if } j \in C_{i-1}, \\
0, & \text{otherwise.}
\end{cases}
\]

Now obtain minimum 1-unsafe sets for the first row of \( X \) and let \( m \) be the size of the minimum 1-safe sets. Note that \( J \subseteq S \) is 1-unsafe iff \( J \) contains at least one element from \( C_i, \ i = 1,\ldots,n-1 \). Therefore the hitting set problem can be solved by comparing \( m \) and \( K \). We see that any algorithm of finding minimum 1-unsafe sets can be used to solve the hitting set problem. Since the hitting set problem is NP-complete, the problem of finding maximum 1-unsafe subsets is NP-complete as \( n \) and \( p \to \infty \).
B  Program for obtaining minimal $k$-unsafe and maximal $k$-safe sets

Here we present a C program for obtaining minimal unsafe and maximal safe sets. This program implements Step A1 and Step B of Section 3 and was used for computing the example in Section 4. Memory allocation of two-dimensional array is adapted from Press et al. (1988). Updated source of the following program is available via Internet from

http://www.e.u-tokyo.ac.jp/~takemura/minimalunsafe.html.

1 /*
2 Program for obtaining minimal unsafe sets and maximal safe sets
3 for each unsafe row of an nxp data file. We assume that the
4 values in the data file are integers separated by white spaces.
5 Akimichi Takemura (A.Takemura@e.u-tokyo.ac.jp)
6 Ver. 1. January 1999
7 */
8
9 #include <stdio.h>
10 #include <stdlib.h>
11
12 #define DEBUGPRINT0
13 #undef DEBUGPRINT1
14
15 #define LONGBITS 32 /* assuming that long is 32 bits. 
16 We are limited to 32 bits. 
17 Practically we are limited to 
18 about 15 bits (or variables) 
19 */
20 #define INFTY 0x7fffffffL
21 #define ALLONE 0xffffffff
22
23 #define SAFE 0
24 #define UNSAFE 1
25 #define UNCHECKED -1
26
27 /* is a subset of b? */
28 #define issubset(a,b) ( ((a)|(b)) == (b)? 1: 0)
29
30 long **data; /* data matrix available to subroutines */
31
32 /* powers of 2 */
33 long ipow2(short p)
34 {
35     long m = 1;
36     short j;
37     for (j = 1; j <= p; j++, m *= 2);
38     return m;
39 }
40
41 /* print least significant p bits of long variable */
42 void printbits(long longval, short p)
{  
  short j;
  for (j = p - 1; j >= 0; j--)
    printf("%d", (longval >> j) & 1 ? 1 : 0);
}

/* number of 1's in bit pattern of length p */
short bitweight(long longval, short p)
{
  short j, w = 0;
  for (j = p - 1; j >= 0; w += ((longval >> j) & 1), j--);
  return w;
}

/* allocate a long nxp matrix, routine adapted from the book
  "Numerical Recipes in C" */
long **
makematrix(long n, long p)
{
  long **m, i;
  /* allocate pointers to rows */
  m = (long **) malloc(n * sizeof(long));
  /* allocate rows and set pointers to them */
  m[0] = (long *) malloc(n * p * sizeof(long));
  if (m[0] == NULL) {
    printf("Memory allocation failure\n");
    exit(1);
  }
  for (i = 1; i <= n; i++)
    m[i] = m[i - 1] + p;
  /* return pointer to array of pointers to rows */
  return m;
}

/* binomial coefficient */
long binomial(short m, short j)
{
  long bin;
  short i;
  bin = 1;
  if ((j == 0) || (j == m))
    return bin;
  if (m - j < j)
    j = m - j;
  for (i = 0; i < j; i++)
    bin = bin * (m - i) / (i + 1); /* bin is always integer.
       However bin may overflow */
  return bin;
}

/* construct a list of integers from 0 .. 2^p-1

void makebitlist(long *bl, short p) {
    /* reserve an array of pointers. i-th pointer
     is an array of patterns of i+1 bits */
    long **bitlists;
    long m, j, bnsum;
    short i, k;

    bitlists = (long **) malloc(sizeof(long) * p);
    bitlists[0] = (long *) malloc(sizeof(long) * 2);
    m = 2;
    bitlists[0][0] = 0;
    bitlists[0][1] = 1;
    for (i = 1; i < p; i++) {
        m *= 2;
        bitlists[i] = (long *) malloc(sizeof(long) * m);
        bnsum = 0;
        for (k = 0; k <= i; k++) {
            for (j = 0; j < binomial(i, k); j++)
                bitlists[i][2 * bnsum + j] = bitlists[i - 1][bnsum + j];
            for (j = 0; j < binomial(i, k); j++)
                bitlists[i][2 * bnsum + binomial(i, k) + j] = (1 << i) | bitlists[i - 1][bnsum + j];
            bnsum += binomial(i, k);
        }
    }
    for (j = 0; j < m; j++)
        bl[j] = bitlists[p - 1][j];
    for (i = 1; i < p; i++)
        free(bitlists[i]);
    free(bitlists);
}

void makereversebitlist(long *bl, short p) {
    long i, m = ipow2(p);
    makebitlist(bl, p);
    for (i = 0; i < m; i++)
        bl[i] = bl[i] ^ ALLONE;
}

int hamming(long *a, long *b, short p, long bitpat) {
    int ham;
    short i;
    ham = 0;
    for (i = 0; i < p; i++)
        ham += (a[i] == b[i] ? 0 : 1) * ((bitpat >> (p - i - 1)) & 1 ? 1 : 0);
    return ham;
}
int minhamming_k(long i, long n, short p, long bitpat, short k) {
    long j, *minham_k;
    int h, tmpval;
    minham_k = (long *) malloc(k * sizeof(long));
    for (h = 0; h < k; h++)
        minham_k[h] = INFTY;
    /* store minimum k values in ascending order */
    for (j = 0; j < n; j++) {
        if (j == i)
            continue;
        if (hamming(data[i], data[j], p, bitpat) < minham_k[k - 1]) {
            minham_k[k - 1] = hamming(data[i], data[j], p, bitpat);
            h = k - 1;
            while (h > 0 && minham_k[h - 1] > minham_k[h]) {
                tmpval = minham_k[h];
                minham_k[h] = minham_k[h - 1];
                minham_k[h - 1] = tmpval;
                h--;
            }
            if (minham_k[k - 1] == 0)
                break;
        }
    }
    return minham_k[k - 1];
    free(minham_k);
}

void print_minimal_unsafe(signed char *safenesslist, short p, long *bitlist) {
    signed char *checklist;
    long i, j, m = ipow2(p);
    checklist = (signed char *) malloc(m * sizeof(signed char));
    for (i = 0; i < m - 1; i++)
        checklist[i] = UNCHECKED;
    /* going through safenesslist in reverse */
    for (i = m - 2; i >= 0; i--) {
        if (checklist[i] != UNCHECKED)
            continue;
        if (safenesslist[i] == UNSAFE) {
            printbits(bitlist[i], p);
            putchar(' ');
            for (j = i - 1; j >= 0; j--) {
                if (issubset(bitlist[i], bitlist[j]))
                    checklist[j] = UNSAFE;
            }
        }
    }
    free(checklist);
/* printout maximal safe sets */

void print_maximal_safe(signed char *safenesslist, short p, long *bitlist)
{
    signed char *checklist;
    long i, j, m = ipow2(p);
    checklist = (signed char *) malloc(m * sizeof(signed char));
    for (i = 0; i < m - 1; i++)
        checklist[i] = UNCHECKED;
    /* going through safenesslist */
    for (i = 0; i < m - 1; i++) {
        if (checklist[i] != UNCHECKED)
            continue;
        if (safenesslist[i] == SAFE) {
            printbits(bitlist[i], p);
            putchar(' ');
            for (j = i + 1; j < m - 1; j++) {
                if (issubset(bitlist[j], bitlist[i])
                    checklist[j] = SAFE;
                }
            }
    free(checklist);
}

/* construct the list of safeness for i-th row.
Assume that i-th row is k-unsafe */
void build_safeness_list(signed char *safenesslist, long i, long n, short p, long *bitlist, short k)
{
    long m = ipow2(p), j, h;
    int minham_k;
    for (j = 0; j < m - 1; j++)
        safenesslist[j] = UNCHECKED;
    for (j = 0; j < m - 1; j++) {
        if (safenesslist[j] != UNCHECKED)
            continue;
        minham_k = minhamming_k(i, n, p, bitlist[j], k);
        for (h = j; h < m - 1; h++) {
            if (!issubset(bitlist[h], bitlist[j]))
                continue;
            if (minham_k == 0)
                safenesslist[h] = SAFE;
            if (minham_k > 0 &&
                bitweight(bitlist[j], p) - bitweight(bitlist[h], p) < minham_k)
                safenesslist[h] = UNSAFE;
        }
    }

int main(int argc, char *argv[])
{
    short k, p;
259  long *bitlist, n, m, i, j;
260  signed char *safenesslist;
261  FILE *filep;
262
263  if (argc != 5) {
264      printf("Usage: %s n p filename\n", argv[0]);
265      printf("   where n is number of rows\n");
266      printf("   p is number of columns\n");
267      printf("   k is the threshold (in \"k-unsafe\")\n");
268      exit(1);
269  }
270  n = atoi(argv[1]);
271  p = atoi(argv[2]);
272  k = atoi(argv[3]);
273  if (NULL == (filep = fopen(argv[4], "r"))) {
274      printf("Can not open file: %s\n\n", argv[3]);
275      exit(1);
276  }
277  m = ipow2(p);
278  bitlist = (long *) malloc(sizeof(long) * m);
279  makereversebitlist(bitlist, p);
280  data = makematrix(n, p);
281
282  /* read in the data. values are assumed to be integers
283     separated by white spaces */
284  for (i = 0; i < n; i++) {
285      for (j = 0; j < p; j++)
286          fscanf(filep, "%ld", &data[i][j]);
287  }
288  safenesslist = (signed char *) malloc(m * sizeof(signed char));
289
290  printf("n=%ld, p=%d, k=%d, file=%s\n", n, p, k, argv[4]);
291  printf("row: MU:Minimal Unsafe sets MS:Maximal Safe sets\n");
292  for (i = 0; i < n; i++) {
293      if (minhamming_k(i, n, p, m - 1, k) == 0)
294          continue; /* k-safe */
295      else {
296          build_safeness_list(safenesslist, i, n, p, bitlist, k);
297          printf("%ld: ", i + 1);
298          print_minimal_unsafe(safenesslist, p, bitlist);
299          printf(" MU: ");
300          print_maximal_unsafe(safenesslist, p, bitlist);
301          printf(" MS: ");
302          putchar(\n");
303      }
304  }
305  free(bitlist);
306  free(safenesslist);
Acknowledgment

I am grateful to Daishin Nakamura for pointing out the NP-completeness of minimum 1-unsafe set problem and to Nobuaki Hoshino for some helpful comments.

References


