

# A new algorithm for finding Minimal Sample Uniques for use in Statistical Disclosure Assessment

A. M. Manning

Department of Computer Science  
University of Manchester, Oxford Rd.  
Manchester, M13 9PL, England  
anna@cs.man.ac.uk

D. J. Haglin

Dept. of Computer & Information Sciences  
Minnesota State University  
Mankato, MN 56001, USA  
David.Haglin@mnsu.edu

## Abstract

We present *SUDA2*, a recursive algorithm for finding Minimal Sample Uniques (MSUs). *SUDA2* uses a novel method for representing the search space for MSUs and new observations about the properties of MSUs to prune and traverse this space. Experimental comparisons with previous work demonstrate that *SUDA2* is not only several orders of magnitude faster but is also capable of identifying the boundaries of the search space, enabling datasets of larger numbers of columns than before to be addressed.

## 1. Introduction

Improvements and innovations in computer processing power, disk storage and networks have led to dramatic increases in the ability to accumulate and analyze personal data. However, if personal data is made available, even in an anonymized form, there is a risk of individuals being identified using statistical disclosure through the matching of known information with the anonymized data, resulting in material specific to those individuals being revealed. This work focuses on the identification of individual records with a high risk of disclosure, a process otherwise known as Statistical Disclosure Assessment. The records belonging to certain individuals have a significant chance of being identified as their contents, or attributes, are unique and therefore have the potential to be matched directly with details (including names and addresses) from another dataset. An illustration of a ‘risky’ record of this type is a sixteen-year-old widow in a population survey. A record can contain more than one such unique pattern and its classification often depends on the number and size of such attribute sets (referred to as *uniques*) that it contains [6].

The ability to comprehensively locate and grade such records leads to more efficient Statistical Disclosure Control (SDC) of released data. In order to carry out an ex-

haustive search of this nature all possible attribute sets must be checked (directly or indirectly) for uniqueness. While it may be possible to concentrate only on attributes most likely to be problematic, there would always be the possibility of missing some.

Existing techniques can find *outliers* (unusual records) in a dataset if a generic dissimilarity measure between records can be constructed (such as a distance metric in  $n$ -dimensional space where  $n$  is the number of attributes per record, or a measure of variation between DNA sequences in biological data) [2]. However, many datasets contain categorical variables (such as *Marital Status* in a population survey) for which generic dissimilarity measures are difficult to derive and for which distance metrics are not relevant. In general, a record classified as an outlier does not automatically contain unique patterns which can be directly matched with records in an independent dataset and the detection of outliers does not guarantee that all records containing unique attribute patterns are identified.

Previous work in SDC has led to the development of algorithms designed to protect the confidentiality of individual records under certain conditions, for example, techniques for modifying classifiers so as to protect record-level privacy [7] and techniques for assessing the maximum number of queries that can be made without compromising the confidentiality of a given dataset [3], but these have not addressed the uniques problem directly. Although some areas of SDC research have focused on the location of ‘risky’ records [5, 8, 12] this work, although theoretically interesting, does not overcome the combinatorially explosive properties of the search for uniques and cannot yet be used to provide a full analysis of the risk associated with any given dataset.

A sequential algorithm, *SUDA* (Special Unique Detection Algorithm), has been designed and implemented specifically for this problem [6] and is currently used by the Office for National Statistics (ONS) in London. The

ONS releases many data samples from very large confidential datasets, such as the Census of Great Britain. These samples are assessed for potential risk of statistical disclosure by applying SUDA. SUDA was inspired by techniques used in association rule discovery [1, 4].

SUDA employs a two-stage approach: firstly, all uniques (up to a user-specified size) are located at record level and, secondly, the size and distribution of uniques within each record is used to make statistical inferences about the records that are likely to be most ‘risky’ in the entire dataset [6]. Only unique attribute sets without any unique subsets — *Minimal Sample Uniques* (MSUs) — are considered in order to avoid the use of redundant information and to keep the classification process as focused as possible; the smaller the number of attributes contained in a unique pattern the more ‘risky’ it is considered to be [6]. It is therefore important to know if a unique is minimal.

The search for MSUs is extremely challenging as the problem of finding the smallest combination of attributes that have a unique pattern for a particular record is an  $\mathcal{NP}$ -complete problem [13]. The deleterious effect of the massive computational requirement is exacerbated by the fact that SDC is often an iterative process, requiring a complete risk assessment as each masking technique is tested for effectiveness.

Although SUDA has greatly increased the depth of risk assessment possible, the demanding levels of execution time required to find all MSUs in stage one mean that it is restricted to small datasets, particularly in terms of the number of columns that they possess. This problem formed the motivation for the development of a new algorithm, SUDA2.

## 2. Notation, terms and assumptions

The target datasets are assumed to have rows of equal length as this is usually the case for survey-type data. Consider a dataset as a *table* with *rows* and *columns*. For convenience, the columns are labeled  $A, B$ , etc. An *item* is a column label juxtaposed with a value that is valid for that column. Examples of items in Table 1 are:  $A_1, A_2$  and  $E_3$  (corresponding to  $A=1, A=2$  in column 1 and  $E=3$  in column 5). The number of times an item occurs in a dataset is referred to as its *repetition count*. The following discussion refers to sets of items as an *itemset*. An itemset  $\mathcal{M}$  is a Minimal Sample Unique if it satisfies two properties: (i) the pattern described by  $\mathcal{M}$  appears in exactly one row of the dataset (the *reference row*), and (ii) every proper subset of  $\mathcal{M}$  appears in multiple rows of the table (the minimality constraint). A  $k$ -MSU is a MSU of size  $k$ .

The *Minimal Uniques Problem* can be outlined as follows: given a table of discrete data — both numerical data (such as AGE) and numerically-coded categorical data

(such as Marital Status with 1=single, 2=married etc.) — containing  $C$  columns and  $R$  rows, the problem is to find all  $k$ -MSUs,  $1 \leq k \leq M$ , where  $M \leq C$  and is user-specified. It is assumed that the entire table (and all data structures used for the algorithm) fit into main memory completely. The terms “table” and “dataset” are used interchangeably in the following sections.

## 3. SUDA

The SUDA algorithm essentially generates all possible column subsets in a depth-first manner and scans the input dataset for unique patterns in those columns. Using the observation that once an MSU is discovered no superset of the MSU need be considered, it is possible to prune parts of the search space. By selecting in succession all column subsets with the same prefix any extensions of a unique prefix at a given row are ignored. For example, given four columns, labeled  $A, B, C$  and  $D$ , the column subsets with prefix  $B$  are:  $B, BC, BCD$  and  $BD$ . If column  $B$  was found to have a unique value for row  $S$  then column subsets  $BC, BCD$  and  $BD$  could be ignored for that row. This has the effect of reducing the number of rows that need to be considered for each column subset while at the same time minimizing memory usage [6].

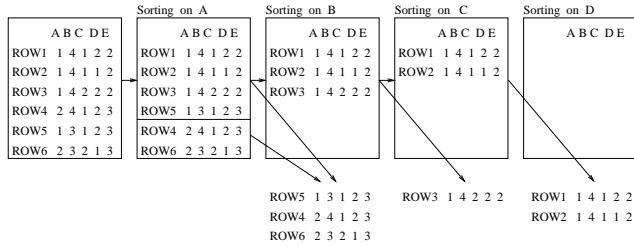
The process of identifying MSUs is conducted by partitioning the rows of the dataset according to the value of each of the columns in a given column subset. For example, if the column subset was (AGE, SEX) the dataset would be divided into groups of rows containing items such as (AGE=20, SEX=male), (AGE=40, SEX=female) etc. Any group containing only one row represents a unique itemset and a check for minimal uniqueness would then be made to determine MSU status. The minimal uniqueness of a unique itemset  $X$  of size  $n$  (where  $n \geq 2$ ) is determined by confirming that all subsets of  $X$  of size  $n - 1$  are non-unique. The partitioning method of SUDA has the effect of minimizing the amount of data storage that is necessary to identify minimal uniques by localizing the required information. The generation of column subsets according to their prefixes also allows this partitioning procedure to be undertaken efficiently and without redundant sorting [6]. This process continues until either all possible column subsets have been considered or all those up to the artificial upper limit on  $k, M$ , have been addressed.

Figure 1 shows a dataset with six rows (labeled ROW1 to ROW6) and five columns (labeled A to E). The diagram illustrates the process of finding MSUs with column subsets  $A, AB, ABC, ABCD$  and  $ABCDE$ . Sorting on column  $A$  divides the dataset into two partitions but does not yield any unique itemsets. When each of these partitions is further sorted on column  $B$  three unique itemsets are discovered ( $A_2B_4, A_1B_3$  and  $A_2B_3$ ), at rows 4, 5 and 6. These are

**Table 1. Example of a dataset**

	A	B	C	D	E
Row 1	1	4	1	2	2
Row 2	1	4	1	1	2
Row 3	1	4	2	2	2
Row 4	2	4	1	2	3
Row 5	1	3	1	2	3
Row 6	2	3	2	1	3

all MSUs. When the remaining rows (1, 2 and 3) are sorted on column C row 3 is unique. However, this does not contain an MSU as A=1, C=2 is unique. Sorting on D yields unique itemsets at both rows 1 and 2, but only that at row 1 is an MSU. There are no rows left to check for ABCDE. The search then proceeds to ABCE, ABD, ABDE etc.



**Figure 1. Sorting process for SUDA**

## 4. SUDA2

The maximum size of an MSU is often far smaller than the number of columns in the input dataset. However, SUDA makes little attempt to identify the boundaries of the search space. Although its method of pruning has the effect of reducing the number of rows in the sorting procedures associated with each column subset, it has less impact on the number of column subsets considered. If any two rows in the dataset are identical then all possible column subsets will be generated for checking. It is important to note, however, that SUDA is driven by the data itself and only considers itemsets that exist within a row of the dataset (i.e. itemsets that have the potential to be an MSU). If this were not the case redundant searching could ensue.

SUDA2 aims to use a more flexible representation of the search space and to identify its boundaries but, at the same time, maintain the data driven approach of SUDA.

### 4.1. Representing the search space for MSUs

Table 1 shows a table with six rows and five columns. The search space for MSUs in this table can be viewed as all possible, distinct subsets of each of these rows.

This search space can be altered as new information becomes available. For example, if  $A_2B_3$  is found to be an

**Table 2. The effect of removing one 2-MSU from the search space**

	Search Space	2-MSUs
Row 1	$A_1B_4C_1D_2E_2$	
Row 2	$A_1B_4C_1D_1E_2$	
Row 3	$A_1B_4C_2D_2E_2$	
Row 4	$A_2B_4C_1D_2E_3$	
Row 5	$A_1B_3C_1D_2E_3$	
Row 6	$A_2C_2D_1E_3, B_3C_2D_1E_3$	$A_2B_3$

**Table 3. The effect of removing one item,  $A_2$ , from the search space**

	Search Space	2-MSUs
Row 1	$A_1B_4C_1D_2E_2$	
Row 2	$A_1B_4C_1D_1E_2$	
Row 3	$A_1B_4C_2D_2E_2$	
Row 4	$B_4C_1D_2E_3$	$A_2B_4, A_2C_1, A_2D_2$
Row 5	$A_1B_3C_1D_2E_3$	
Row 6	$B_3C_2D_1E_3$	$A_2B_3, A_2C_2, A_2D_1$

MSU at row 6 the search space at row 6 can be reduced from all subsets of  $A_2B_3C_2D_1E_3$  to all subsets of  $A_2C_2D_1E_3$  and  $B_3C_2D_1E_3$  so that  $A_2B_3$  is not reconsidered, as shown in Table 2. Each time a new MSU  $\mathcal{M}$  is identified the search space associated with that row can be reduced so that  $\mathcal{M}$  is no longer a subset of any of its itemsets. Row-level search spaces can continue to shrink in this way as MSUs are discovered. The search will terminate when all subsets have been considered, either directly or indirectly.

However, keeping track of row-level search spaces is likely to be complex and potentially demanding on memory. A more methodical approach is therefore required.

It is possible to consider all MSUs containing a given item independently. For example, Table 3 shows all MSUs pertaining to  $A_2$  and the corresponding effect on the search space; the search space at rows 4 and 6 is adjusted by removing  $A_2$ , a step which is straightforward to implement. Table 4 shows the search space and the MSUs after  $A_2, B_3, C_2$  and  $D_1$  have been removed and Table 5 shows the final list of MSUs (i.e. when the search space is empty). In summary, as more MSUs are discovered the search space shrinks.

The above example illustrates a dynamic method of representing the search space which maintains the data driven aspect of SUDA. By considering items individually, rather than in blocks of entire columns of the table, it is possible to take advantage of properties that they possess, such as their differing levels of repetition counts, as discussed in the next section.

**Table 4. The effect of removing four items,  $A_2, B_3, C_2, D_1$ , from the search space**

	Search Space	2-MSUs
Row 1	$A_1 B_4 C_1 D_2 E_2$	
Row 2	$A_1 B_4 C_1 E_2$	$A_1 D_1, B_4 D_1, C_1 D_1, D_1 E_2$
Row 3	$A_1 B_4 D_2 E_2$	$A_1 C_2, B_4 C_2, C_2 D_2, C_2 E_2$
Row 4	$B_4 C_1 D_2 E_3$	$A_2 B_4, A_2 C_1, A_2 D_2, B_4 E_3$
Row 5	$A_1 C_1 D_2 E_3$	$A_1 B_3, A_1 E_3, B_3 C_1, B_3 D_2$
Row 6	$E_3$	$A_2 B_3, A_2 C_2, A_2 D_1, B_3 C_2, B_3 D_1, C_2 D_1, C_2 E_3, D_1 E_3$

**Table 5. The completed search for MSUs**

	2-MSUs	3-MSUs	4-MSUs
Row 1		$C_1 D_2 E_2$	$A_1 B_4 C_1 D_2$
Row 2	$A_1 D_1, B_4 D_1, C_1 D_1, D_1 E_2$		
Row 3	$A_1 C_2, B_4 C_2, C_2 D_2, C_2 E_2$		
Row 4	$A_2 B_4, A_2 C_1, A_2 D_2, B_4 E_3$		
Row 5	$A_1 B_3, A_1 E_3, B_3 C_1, B_3 D_2$		
Row 6	$A_2 B_3, A_2 C_2, A_2 D_1, B_3 C_2, B_3 D_1, C_2 D_1, C_2 E_3, D_1 E_3$		

## 4.2. Properties of MSUs

As the problem of finding MSUs is known to be  $\mathcal{NP}$ -complete [13], the best that can be achieved (unless  $\mathcal{P} = \mathcal{NP}$ ) is to find effective heuristics that prune large portions of the search space. The SUDA2 algorithm relies upon a number of properties of MSUs as follows:

- **Support Row Property** Given a  $k$ -MSU  $\mathcal{M}$  at row  $R$  there must be at least  $k$  rows other than  $R$  containing itemsets that differ from  $\mathcal{M}$  by exactly one item so that each possible  $(k - 1)$ -subset of  $\mathcal{M}$  is matched. These rows are called *support rows*. The presence of these support rows ensures the minimality constraint on the  $k$ -MSU; any one of the items removed from  $\mathcal{M}$  must result in a pattern found in at least one other (support) row. The Support Row Property provides an important upper bound for candidate MSUs. If the smallest repetition count of the items contained in a candidate MSU is  $k$ , then there are not enough support rows for any  $l$ -MSU with  $l > k$ .
- **Uniform Support Property** A unique  $k$ -itemset  $\mathcal{M}$  that contains the same item  $I_j$  in each of its support rows cannot be an MSU; it will be a superset of the  $(k - 1)$ -MSU  $\mathcal{L}$  formed by removing the item  $I_j$  from  $\mathcal{M}$ . If this were not the case then  $\mathcal{M}$  could not be unique in the dataset as  $\mathcal{L}$  would occur in at least two rows with  $I_j$ . This can be compared to Parent Equivalence Pruning in Maximal Frequent Itemset mining [4].
- **Recursive Property** Consider a  $k$ -MSU  $\mathcal{M}$  that appears in a row  $R$  in a table  $T$ . Let  $\mathcal{N}$  be the  $k - 1$  items

remaining after removing one item,  $I_j$ , from  $\mathcal{M}$ . Let  $T_{I_j}$  be the subset of  $T$  consisting of only those rows containing  $I_j$ . Then it can be seen that  $\mathcal{N}$  is a  $(k - 1)$ -MSU in  $T_{I_j}$ ; if  $\mathcal{N}$  were non-unique in  $T_{I_j}$  then  $\mathcal{M}$  would be non-unique in  $T$  and if  $\mathcal{N}$  were unique but not minimal in  $T_{I_j}$  then the same would be true for  $\mathcal{M}$  in  $T$ .

- **Perfect Correlation Property** If two items are perfectly correlated — that is, they appear in exactly the same set of rows — then they cannot coexist in an MSU as the Support Row Property does not hold. However, the knowledge that two items are perfectly correlated can be used to prune the search space. Given two perfectly correlated items  $A_j$  and  $B_k$ , for any MSU  $\mathcal{P}$  containing item  $A_j$  there is a corresponding MSU  $\mathcal{Q}$  with  $A_j$  replaced by  $B_k$ .  $\mathcal{P}$  has the same support rows as  $\mathcal{Q}$  since:
  1.  $A_j$  and  $B_k$  are perfectly correlated and
  2.  $\mathcal{P} - A_j = \mathcal{Q} - B_k$ .

## 4.3. Ordering the items

All non-unique items are arranged in ascending order of repetition count as this helps to improve the potential for pruning via the Uniform Support Property. Unique items, or 1-MSUs, are recorded immediately and then removed from the search as they cannot yield further MSUs. Any item that appears in all rows of the dataset cannot lead to an MSU by the Uniform Support Property and is removed from the search altogether. If two items have the same repetition count then:

- intra-column items are sorted in ascending order of their value — e.g.  $A_2$  comes before  $A_3$ ;
- inter-column items are sorted in ascending order of their column position — e.g. column B comes before column C.

The ordered item list is referred to below as ITEM-LIST. The relative position within ITEM-LIST is called the *rank* of an item. By giving the items a complete ordering, every itemset — in particular, every MSU — contains an item with lowest rank, called the *reference* item.

## 5. The SUDA2 algorithm

### 5.1. Algorithm details

The first stage of SUDA2 is to scan the input dataset in order to find repetition counts for all items that appear at least once. Unique items (1-MSUs) are recorded. The list of non-unique items is sorted in ascending order of repetition count to form ITEM-LIST as described in Section 4.3.

Items with the same repetition count are checked for perfect correlation. One of every pair of perfectly correlated items is removed from ITEM-LIST. For each item that remains in ITEM-LIST the set of items that are perfectly correlated with it is stored. This step is not used in the recursion process.

The main data structures are then built as a pre-processing step. Firstly, table T is created with each item replaced by its rank which means that ranks and relative ranks can be determined immediately. Secondly, each item is associated with a set of rows that contain it so that, for any given item, the rows in which it occurs can be found immediately. This, in turn, enables the coexisting items (i.e. items within the same row) to be determined from T.

- The items in ITEM-LIST are considered as independent subsearches in ascending order of rank. For each item of rank  $r$ , SUDA2 builds a subtable  $T_r$  of rows containing  $r$ . SUDA2 is applied recursively to  $T_r$  to find all  $(k - 1)$ -MSUs  $\mathcal{N}$  in  $T_r$ , where  $2 \leq k \leq C$ , to be used as candidates for finding  $k$ -MSUs in  $T$ .
- A candidate  $(k - 1)$ -MSU,  $\mathcal{N}$ , appearing in row  $i$  leads to a  $k$ -MSU for row  $i$  provided two properties hold.
  1. All of the  $k - 1$  items in  $\mathcal{N}$  have rank higher than  $r$  in table  $T$ .
  2. There is a row in  $T$  (and not in  $T_r$ ) holding all of the items in  $\mathcal{N}$  but not holding the reference item. This row is known as the *special row*. The other  $k - 1$  support rows are guaranteed to exist by the Recursive Property since  $\mathcal{N}$  is a  $(k - 1)$ -MSU in  $T_r$ . There is no need to check the special row for MSU candidates of size 2 as the second item in the candidate itemset will automatically have repetition count greater than or equal to the reference item (as it appears higher in ITEM-LIST) and will therefore appear in at least one other row.
- When an MSU is discovered the corresponding MSUs of perfectly correlated items are added.

## 5.2. Example

The following example applies SUDA2 to the data in Table 1 in order to find the MSUs shown in Table 5.

- The items are sorted in ascending order of repetition count to form the following ITEM-LIST:  $A_2, B_3, C_2, D_1, E_2, E_3, A_1, B_4, C_1, D_2$ . In this example items are not replaced by their ranks in the following tables as it is clearer to demonstrate the mechanisms of the algorithm when the items themselves are

**Table 6. Table  $T_{A_2}$**

	col A	col B	col C	col D	col E
Row 4		$B_4$	$C_1$	$D_2$	
Row 6		$B_3$	$C_2$	$D_1$	

**Table 7. Table  $T_{E_2}$**

	col A	col B	col C	col D	col E
Row 1			$C_1$	$D_2$	
Row 2			$C_1$		
Row 3				$D_2$	

presented. A subtable of T of rows containing  $A_1$  is denoted  $T_{A_1}$ , a subtable of  $T_{A_1}$  of rows containing  $B_2$  is denoted  $T_{A_1B_2}$  etc.

- There are no unique items, no items occurring in all rows and no perfectly correlated items in this dataset.
- Subtable  $T_{A_2}$  is created and is shown in Table 6. The column for  $E_3$  is removed from  $T_{A_2}$  as  $E_3$  occurs in both rows 4 and 6 and is therefore redundant by the Uniform Support Property. The MSUs are  $A_2B_4, A_2C_1$  and  $A_2D_2$  at row 4 and  $A_2B_3, A_2C_2$  and  $A_2D_1$  at row 6. No check for the *special row* was necessary as the candidate MSUs are of size 2.

The algorithm progresses in a similar manner for items  $B_3, C_2$  and  $D_1$ ; Table 4 shows the MSUs for all items with repetition counts of 2.

- Subtable  $T_{E_2}$  is shown in Table 7. There are no unique items in  $T_{E_2}$  so the rows containing the next item in the ITEM-LIST for  $E_2$ ,  $C_1$ , are extracted to get  $T_{C_1E_2}$ , which is shown in Table 8.  $D_2$  is unique in  $T_{C_1E_2}$  and  $C_1D_2$  is a candidate MSU for  $T_{E_2}$ . A check for the *special row* shows that  $D_2$  occurs independently of  $C_1$  in  $T_{E_2}$  (at row 3) and thus  $C_1D_2$  is an MSU in  $T_{E_2}$ . The itemset  $C_1D_2E_2$  is a candidate MSU for the whole dataset  $T$ . This is confirmed by establishing that  $C_1D_2$  occurs independently of  $E_2$  (at rows 4 and 5) in  $T$ .
- Subtable  $T_{E_3}$  is shown in Table 9. Two 2-MSUs,  $A_1E_3$  and  $B_4E_3$  are thrown up immediately. Subtable  $T_{C_1}$  is empty and the search proceeds to the next item.
- Subtable  $T_{A_1}$  is shown in Table 10. There are no unique items so rows containing  $B_4$  are extracted to get  $T_{A_1B_4}$  as shown in Table 11. As there are still no

**Table 8. Table  $T_{C_1E_2}$**

	col A	col B	col C	col D	col E
Row 1				$D_2$	
Row 2					

**Table 9. Table  $T_{E3}$**

	col A	col B	col C	col D	col E
Row 4		$B_4$	$C_1$	$D_2$	
Row 5	$A_1$		$C_1$	$D_2$	
Row 6					

**Table 10. Table  $T_{A1}$**

	col A	col B	col C	col D	col E
Row 1		$B_4$	$C_1$	$D_2$	
Row 2		$B_4$	$C_1$		
Row 3		$B_4$		$D_2$	
Row 5			$C_1$	$D_2$	

unique items subtable  $T_{A1B4C1}$  (Table 12) is generated where  $D_2$  is unique and  $C_1D_2$  is a candidate MSU for  $T_{A1B4}$ . A check for the *special row* shows that  $C_1$  occurs independently of  $D_2$  at row 2 and therefore  $C_1D_2$  is an MSU in  $T_{A1B4}$  and  $B_4C_1D_2$  is a candidate MSU for  $T_{A1}$ . Further checking shows that  $C_1D_2$  occurs independently of  $B_4$  at row 5 and therefore  $B_4C_1D_2$  is an MSU in  $T_{A1}$ . Now  $A_1B_4C_1D_2$  is a candidate MSU for  $T$ , which is confirmed by observing that  $B_4C_1D_2$  occurs independently of  $A_1$  at row 4.

- Items  $B_4$ ,  $C_1$  and  $D_2$  yield no MSUs.

## 6. Performance results

### 6.1. Comparison with SUDA2

Performance tests were applied to SUDA and SUDA2 on a Dell Workstation with a 3GHz Pentium 4 processor with 2Gbytes main memory. The operating system used was Red Hat Linux 3.3.2-5. SUDA was written in C and SUDA2 in C++ and both were compiled with gcc version 3.2.2 using -O3 optimization.

The effectiveness of SUDA and SUDA2 were tested on three datasets as follows:

- **1991 SARs** Both SUDA and SUDA2 were applied to one geographical area of the 2% 1991 Individual SAR — the 2% Sample of Anonymized Records released from the 1991 Census of Population of Great Britain [11] of approximately 9000 rows and 43 columns. For SUDA, MSUs of no bigger than size six were found over a matter of days. SUDA2 was able to find all

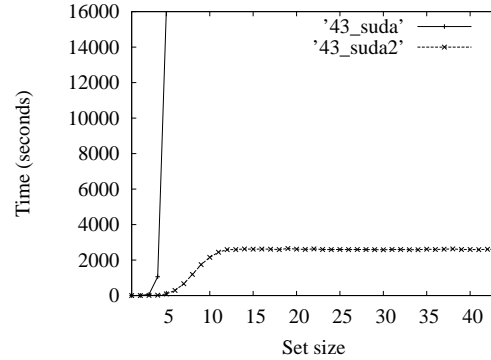
**Table 11. Table  $T_{A1B4}$**

	col A	col B	col C	col D	col E
Row 1			$C_1$	$D_2$	
Row 2			$C_1$		
Row 3				$D_2$	

**Table 12. Table  $T_{A1B4C1}$**

	col A	col B	col C	col D	col E
Row 1				$D_2$	
Row 2					

MSUs with no limit in under one hour. The largest MSU in this dataset had size 15. Figure 2 shows the execution time in seconds for SUDA and SUDA2. The SUDA2 performance curve flattens off as the marginal number of MSUs falls.



**Figure 2. Running time for the 1991 SARs data**

- **The British Labour Force Survey (LFS) SUDA and SUDA2** were applied to the LFS for the quarter December – February 2000 [10]. This dataset has 137823 rows; 84 columns were selected. The LFS is a quarterly sample survey of households living at private addresses in Great Britain. The survey contains information on respondents’ personal circumstances and their labour market status during a specific response period (normally of one week or four weeks, depending on the topic) immediately prior to the interview. The LFS presents a significant challenge to both SUDA and SUDA2 as the number of MSUs it contains is very large — a total of 1,570,173,746 for MSUs up to size 6. SUDA2 is able to make much better progress than SUDA as shown in Figure 3.
- **The Mushroom Dataset** The mushroom dataset (available from the UCI repository [9]) is an 8124-row, 23 column dataset which describes hypothetical samples of gilled mushrooms. Each mushroom is classified as either definitely edible or definitely poisonous (with unknown edibility or not recommended belonging to the latter category). Besides the classification property there are another 22 columns describing properties of the mushrooms such as their appearance and odour. Figure 4 shows the performance curves for

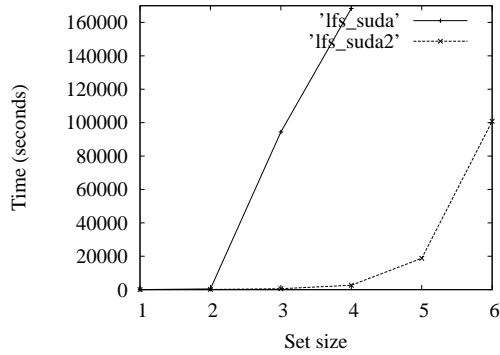


Figure 3. Running time for the LFS data

SUDA and SUDA2 using a log scale (which is why the y-axis starts at 1). SUDA struggled to make much progress with this dataset whereas SUDA2 was able to find all 11,507 MSUs in three seconds. The largest size of MSU was 10.

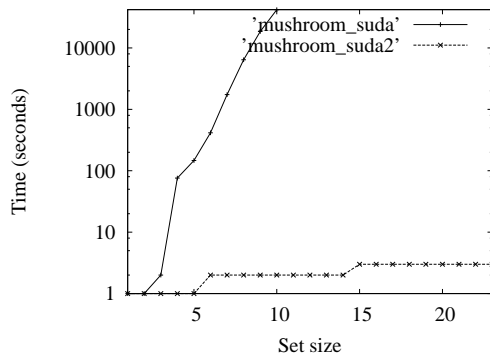


Figure 4. Running time for the Mushroom data

SUDA2 can be seen to outperform SUDA on all these datasets and its ability for its performance curve to flatten off after the largest MSU is clearly demonstrated in two of the examples.

## 6.2. Further performance tests for SUDA2

The execution times for SUDA2 were considered according to rising numbers of rows on the same machine and environment as in the previous section and the results are shown in Table 13 (“X”s refer to jobs that were too large to run on this resource). The data samples were selected at regular values of column AGE from the 1991 SARs [11] and the columns were chosen at random. Figures 5, 6, and 7 plot the log of the number of rows against the log of the execution time for SUDA2. An n-squared line — the hypothetical case where the execution time is  $O(n^2)$ , where  $n$  is

Table 13. Execution times for SUDA2 for rising numbers of rows with time in hours:minutes:seconds

Number of rows	8 columns	16 columns	32 columns
1000	00:00:00	00:00:00	00:00:34
2000	00:00:00	00:00:01	00:01:47
4000	00:00:00	00:00:01	00:05:47
8000	00:00:00	00:00:02	00:19:00
16000	00:00:01	00:00:07	01:06:09
32000	00:00:04	00:00:19	03:52:59
64000	00:00:10	00:00:50	14:05:52
128000	00:00:10	00:02:29	53:07:55
256000	00:00:24	00:07:28	X
512000	00:01:05	X	X
1024000	X	X	X

the number of rows — is also shown. It can be seen that the execution time for SUDA2 falls below this line.

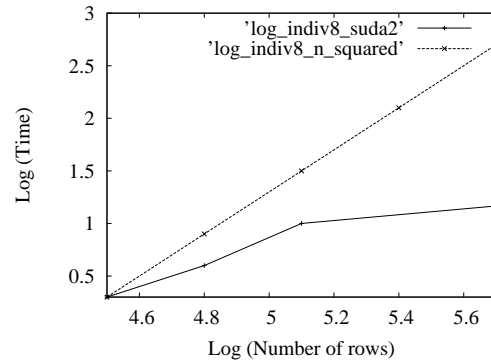
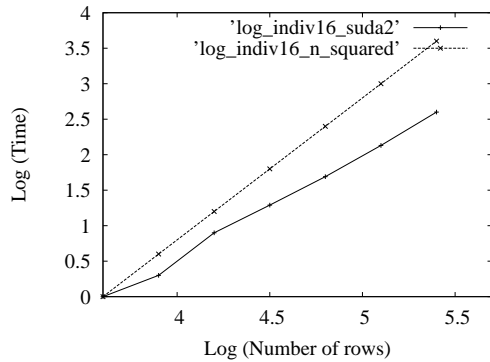


Figure 5. Log plot for the 8 column 1991 SARs data

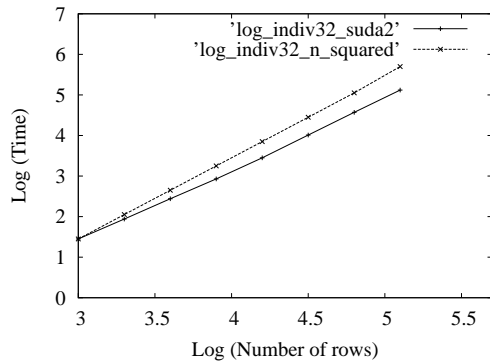
## 7. Conclusion and further work

SUDA2 has been shown to have the ability to identify the boundaries of the search space for MSUs with an execution time which is several orders of magnitude faster than that of SUDA. This has been brought about by a more sophisticated representation of the search space, the identification of a number of properties of MSUs, improved pruning techniques and a more efficient traversal of the search space. Not only will these developments provide statistical agencies with a much faster tool to work with, but the ability to assess data samples with more columns than before will now be possible.

The next step will be to put SUDA2 onto more powerful machines and architectures. SUDA2 is a good candidate for parallelism as the searches can be divided up according to



**Figure 6. Log plot for the 16 column 1991 SARs data**



**Figure 7. Log plot for the 32 column 1991 SARs data**

rank; the challenge will be to produce efficient load balancing.

## 8. Acknowledgments

The research presented here was funded by the Engineering and Physical Sciences Research Council (Grant Numbers GR/S27790/01 & GR/S22295/01). We would also like to thank members of the Centre for Novel Computing at Manchester University for providing invaluable feedback.

## References

- [1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 International Conference on Management of Data (SIGMOD 93)*, pages 207–216, May 1993.
- [2] A. Arning, R. Agrawal, and P. Raghavan. A linear method for deviation detection in large databases. In *Proceedings of*

- the Second International Conference on Knowledge Discovery and Data Mining, Portland, Or.*, pages 164–169. AAAI Press, 1996.
- [3] L. Brankovic, M. Miller, and J. Siran. Towards a practical auditing method for the prevention of statistical database compromise. In *Proceedings of the Australasian Database Conference*, pages 177–184, 1996.
- [4] D. Burdick, M. Calimlim, and J. Gehrke. Mafi a: A maximal frequent itemset algorithm for transactional databases. In *Proceedings ICDE 2001*, pages 443–452, 2001.
- [5] M. J. Elliot. A new approach to the measurement of statistical disclosure risk. *International Journal of Risk Management*, 2(4), 2000.
- [6] M. J. Elliot, A. M. Manning, and R. W. Ford. A computational algorithm for handling the special uniques problem. *International Journal of Uncertainty, Fuzziness and Knowledge Based Systems*, 10(5):493–509, 2002.
- [7] V. Estivill-Castro and L. Brankovic. Data swapping: Balancing privacy against precision in mining for logic rules. In *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery DaWak'99 (as a DEXA-Conference along with the 10th International Conference on Data and Expert Systems Applications) Florence, Italy*, pages 389–398, August 1999.
- [8] S. E. Fienberg and U. E. Makov. Confidentiality uniqueness and disclosure limitation for categorical data. *Journal of Official Statistics*, 4(4), 1998.
- [9] G. Merz and P. Murphy. Uci repository of machine learning databases. *Technical Report, University of California, Department of Information and Computer Science*: <http://www.ics.uci.edu/mllearn/MLRepository.html>, 1996.
- [10] L. F. S. Labour force survey, available at: <http://www.esds.ac.uk/government/lfs/>.
- [11] S. A. R. S. The samples of anonymised records, available at: <http://www.ccsr.ac.uk/sars/>.
- [12] C. J. Skinner and D. J. Holmes. Estimating the re-identification risk per record. *Journal of Official Statistics*, 14(4):361–372, 1998.
- [13] A. Takemura. Minimum unsafe and maximum safe sets of variables for disclosure risk assessment of individual records in a microdata set. *Journal of the Japan Statistical Society*, 32(1):107–117, 2002.