Computational Results For Various Error Localisation Algorithms

Discussion paper 03003

Ton de Waal

The views expressed in this paper are those of the author
and do not necessarily reflect the policies of Statistics Netherlands

Statistics Netherlands

Voorburg/Heerlen, July 2003

Explanation of symbols

.	= data not available
*	= provisional figure
x	= publication prohibited (confidential figure)
–	= nil or less than half of unit concerned
–	= (between two figures) inclusive
0 (0,0)	= less than half of unit concerned
blank	= not applicable
2002–2003	= 2002 to 2003 inclusive
2002/2003	= average of 2002 up to and including 2003
2002/’03	= crop year, financial year, school year etc. beginning in 2002 and ending in 2003

Due to rounding, some totals may not correspond with the sum of the separate figures.

Publisher
Statistics Netherlands
Prinses Beatrixlaan 428
2273 XZ Voorburg
The Netherlands

Printed by
Statistics Netherlands - Facility Services

Cover design
WAT ontwerpers, Utrecht

Information
E-mail: infoservice@cbs.nl

Where to order
E-mail: verkoop@cbs.nl

Internet
http://www.cbs.nl

ISSN: 1572-0314
Key figure: X-10
Production code: 6008303003

Statistics Netherlands

COMPUTATIONAL RESULTS FOR VARIOUS ERROR LOCALISATION ALGORITHMS

Summary: Over the last few years several algorithms for solving the so-called error localisation problem have been developed at Statistics Netherlands. For six data sets involving numerical data we present computational results for four of those algorithms in this paper. The algorithms are based on a standard mixed integer programming formulation, on the generation of the vertices of a certain polyhedron by means of an adapted version of Chernikova's algorithm, on a branch-and-bound algorithm using Fourier-Motzkin elimination, and on a cutting plane approach.

Keywords: branch-and-bound, cutting planes, error localisation, Fellegi-Holt paradigm, mixed integer programming, vertex generation

1. Introduction

In this paper we compare computational results for four different algorithms for automatic error localisation on six data sets involving numerical data. For the duration of this paper we define the error localisation problem as the problem of changing as few fields as possible so that all edit checks (or edits for short) become satisfied. That is, all algorithms we consider in this paper are based on the Fellegi-Holt paradigm of minimum change (see Fellegi and Holt, 1976). All four algorithms solve the error localisation problem to optimality.

The aim of our comparison study is not to perform a comprehensive evaluation study for all possible data sets, but rather to perform a succinct evaluation study that allows us to identify the most promising algorithm(s) for a number of realistic data sets. We restrict ourselves to data sets involving exclusively numerical data because automatic data editing of economic – and hence (mostly) numerical – data is a far more important subject for Statistics Netherlands than automatic data editing of social – and hence (mostly) categorical – data.

In literature some evaluation studies are already described, see Garfinkel et al. (1986 and 1988), Kovar and Winkler (1996), and Ragsdale and McKeown (1996). Garfinkel et al. (1986) concentrate on error localisation for purely categorical data. The other papers concentrate on error localisation for purely numerical data. It is difficult to compare our results to the described results for numerical data. First, because in most cases the actual computing speeds of the computer systems used in those studies are difficult to retrieve, and hence difficult to compare to the computing speed of present-day PC's. Second, because Garfinkel et al. (1988) and Ragsdale and McKeown (1996) use randomly generated data whereas we use realistic data. We feel that realistic data should be used for evaluation studies, because realistic data and randomly generated data have completely different properties. Kovar and Winkler (1996) use realistic data, but the data set used in their evaluation study is not generally available.

The algorithms we examine are:

- an algorithm based on a standard mixed integer programming (MIP) formulation that is solved by means of the commercial MIP-solver ILOG CPLEX;
- a vertex generation algorithm;
- a non-standard branch-and-bound algorithm;
- a cutting plane algorithm.

The remainder of this paper is organised as follows. In Section 2 we summarise the data sets that we have used for our evaluation study. In Section 3 we provide some information regarding the implementation of the above-mentioned algorithms. The computational results are summarised in Section 4. Section 5 concludes the paper with a brief discussion.

2. The data sets

In Table 1 below we give a summary of the characteristics of the six data sets. In this table the number of variables, the number of non-negativity constraints, the number of edits (excluding the non-negativity constraints), the total number of records, the number of inconsistent records (i.e. records failing edits or containing missing values), and the total number of missing values are listed. Besides, we present the number of records with more than six erroneous fields or missing values. For the purpose of our evaluation study we define these records to be ‘highly erroneous’ ones. In Section 4 we compare the computing time required for the records that are not highly erroneous to the computing time that is required for all records for two of the evaluated algorithms. Finally, we list the average number of errors per inconsistent record (excluding the missing values) and the average number of optimal solutions per inconsistent record.

Table 1. Characteristics of the data sets

	Data set A	Data set B	Data set C	Data set D	Data set E	Data set F
Number of variables	90	76	53	51	54	26
Number of non-negativity constraints	90	70	36	49	54	22
Number of edits [a]	8	20	36	15	21	18
Total number of records	4,347	274	1,480	4,217	1,039	1,425
Number of inconsistent records	4,347	157	1,404	2,152	378	1,141
Total number of missing values	259,838	0	0	0	2,230	195
Number of records with more than 6 errors or missing values	4,346	7	117	16	136	8
Average number of errors per inconsistent record [b]	0.2	2.5	2.6	1.6	5.8	3.0
Average number of optimal solutions per inconsistent record	6.1	12.0	6.9	23.3	1.2	11.6

[a] Excluding non-negativity constraints

[b] Excluding missing values

The numbers in the last two rows of Table 1 have been determined by carefully comparing the number of fields involved in the optimal solutions, respectively the number of optimal solutions, of the various algorithms to each other. The number of fields involved in the optimal solutions is assumed to be equal to the actual number of errors.

The number of variables, edits and records are in most of the six data sets quite realistic. Exceptions are data set A, where the number of edits other than non-negativity edits is very small, and data set B, where the number of records is very small. At Statistics Netherlands, a very large and complex data set to be edited automatically may involve slightly more than 100 variables, about 100 edits, and a few thousand records. These numbers are somewhat higher than for the data sets in Table 1, but for such large data sets the value of many variables equals zero. This simplifies the error localisation problem to some extent, for example, because this justifies replacing missing values by zeros in a pre-processing step.

The six data sets come from a wide range of business surveys, namely a survey on labour costs, a structural business survey on enterprises in the photographic sector, a structural business survey on enterprises in the building and construction industry, a structural business survey on the retail sector, and a survey on environmental expenditures. Besides these data sets we have also used the ABI data set; one of the evaluation data sets from the EUREDIT project. Due to confidentiality reasons the branch of industry to which the businesses in this data set belong has not been made public.

As far as we are able to tell, the six test data sets are not essentially different from other data sets arising in practice. In other words, to the best of our knowledge these data sets are representative for other data sets from business surveys. A good performance on the six data sets hence suggests that the performance on other data sets arising in practice will be acceptable. This is confirmed by practical experience at Statistics Netherlands, where nowadays almost all annual structural business surveys are treated by a combination of selective editing (for details on this implementation of selective editing see Hoogland, 2002) and automatic editing. For an overview of this approach for annual structural business surveys at Statistics Netherlands we refer to De Jong (2002). Automatic editing for annual structural business surveys is carried out by means of SLICE 1.0 (see, e.g., De Waal, 2001, for more information on SLICE), which is based on a vertex generation approach (see De Waal, 2003b). Because extensive use of time-consuming COM-components is made in the software architecture of SLICE, its computing times are of a higher order than those mentioned in Table 2 in Section 4 of this paper. Nevertheless, all involved structural business surveys can be treated by SLICE within a reasonable amount of time. Obviously, computing times vary over data sets of different surveys, but no data set with an exceedingly high computing time has been encountered so far. Our practical experience hence suggests that computational results for our test data sets can be carried over to other business data sets.

3. Implementation of the algorithms

The four algorithms we examine in this paper have been implemented in four prototype computer programs. We briefly discuss the implementation details of these programs in this section. The first algorithm, based on a standard MIP formulation (see e.g. Chapter 3 in De Waal, 2003a), we consider has been implemented by Van Riessen (Van Riessen, 2002), a student at the Hogeschool van Amsterdam (College of Amsterdam), while doing an internship at Statistics Netherlands. This algorithm has been implemented in Visual C++ 6.0, and calls routines of ILOG CPLEX (version 7.5), a well-known commercial MIP-solver, to actually solve the MIP problems involved (see *ILOG CPLEX 7.5 Reference Manual*, 2001). We refer to Van Riessen's program as ERR_CPLEX. It determines one optimal solution per erroneous record. ERR_CPLEX, or more precise the MIP-solver of ILOG CPLEX, suffers from some numerical problems. These problems arise because in (erroneous) records the largest values may be a factor 10^9 or more larger than the smallest values. Due to these numerical problems ERR_CPLEX occasionally generates suboptimal solutions containing too many variables. In some other cases it does not find a solution at all.

The second algorithm, based on vertex generation (see De Waal, 2003b, and Chapter 5 in De Waal, 2003a), has been implemented by the author. This program, CherryPi, has been developed in Delphi 3 (see De Waal, 1996). The implemented algorithm is an adapted version of Chernikova's algorithm. Improvements due to Rubin (1975 and 1977), Sande (1978), Schiopu-Kratina and Kovar (1989), and Fillion and Schiopu-Kratina (1993) on the original algorithm by Chernikova (1964 and 1965) have been implemented in CherryPi. The adapted version of Chernikova's algorithm uses a matrix to solve the error localisation problem. The number of rows of this matrix is implied by the number of edits and the number of variables. The number of columns is determined dynamically. Due to memory and speed restrictions a maximum for the allowed number of columns is set in CherryPi. If the actual number of columns exceeds the allowed maximum, certain columns are deleted. This influences the solutions that are found by CherryPi. Due to this pragmatic rule in some cases only non-optimal solutions may be found, and in some other cases no solutions at all may be found. Another effect of this pragmatic rule is that if columns have been deleted in order to arrive at solutions to an instance of the error localisation problem, the optimality of the found solutions is not guaranteed. The higher the allowed number of columns, the better the quality of the solutions found by CherryPi, but also the slower the speed of the program. Practical experience has taught us that in many instances setting the allowed number of columns to 4,000 gives an acceptable trade-off between the quality of the found solutions and the computing time of the program. In the version of CherryPi that was used for the comparison study the allowed number of columns was therefore set to 4,000. Besides the above-mentioned memory problems, CherryPi occasionally suffers from numerical problems, for the same reason as ERR_CPLEX. The program determines all optimal solutions for each erroneous record.

The third algorithm, based on a non-standard branch-and-bound approach (see, e.g., De Waal, 2000, and Chapter 8 of De Waal, 2003a), has been implemented in a prototype computer program called Leo. The program has been developed in Delphi 3. It requires that a maximum cardinality for the optimal solutions is specified beforehand. Leo determines all optimal solutions up to the specified maximum cardinality for each erroneous record. Records requiring more corrections are rejected for automatic editing by Leo. On two data sets, the data sets for which the computing times of Leo are comparatively bad, we have applied a special, alternative version of Leo in which equalities are treated more efficiently (see Chapter 8 in De Waal, 2003a). Leo sometimes suffers from memory problems, especially for records with many errors, because too many nodes with too many edits need to be stored. For records for which Leo suffers from memory problems, it cannot determine an optimal solution. Leo occasionally suffers from numerical problems, for the same reason as ERR_CPLEX and CherryPi.

The fourth algorithm is based on a cutting plane algorithm similar to the algorithms by Garfinkel et al. (1986, 1988) and Ragsdale and McKeown (1996) (see Chapter 10 in De Waal, 2003a). This algorithm has been implemented by Coutinho, while working temporarily at Statistics Netherlands. We refer to Coutinho's program as CUTTING. It has been developed in Delphi 3. The algorithm proposes potential solutions to the error localisation problem by solving a modified set-covering problem (see Ragsdale and McKeown (1996) and De Waal, 2003a). Subsequently, it checks the feasibility of each proposed solution, and generates additional constraints in case this proposed solution is infeasibility by eliminating the variables involved in the solution. A fundamental part of the program is a solver for modified set-covering problems. Using well-known ideas from literature, we have developed this solver, which is based on a recursive branch-and-bound algorithm, ourselves. We did not spend much time on optimising the performance of this solver. It may, therefore, be improved upon. CUTTING can determine all optimal solutions up to a user-specified maximum cardinality. Records requiring more changes than the user-specified maximum cardinality are rejected for automatic editing by CUTTING. The program can also work without such a maximum cardinality. Like Leo, CUTTING suffers from memory problems for some records containing many errors. For such records, it cannot determine an optimal solution. CUTTING occasionally suffers from numerical problems, for the same reason as the other three programs.

The computing times of ERR_CPLEX and CherryPi may possibly be improved upon if we include a restriction on the number of variables that may be changed, like we do for Leo and CUTTING. The stricter this restriction, the faster each program is likely to be. Including such a restriction in CherryPi will probably have less effect than for Leo and CUTTING, because the search process of CherryPi is based on manipulating edits rather than on treating variables directly. The effect of including a restriction on the number of variables that may be changed in ERR_CPLEX is not entirely clear. On the one hand, in order to include such a restriction an additional integer constraint would be required, which would slightly increase the computing time. On the other hand, the search process would be shortened, because certain possible solutions would not have to be examined. Considering the two opposite effects, we expect that including a restriction on the number of variables that may be changed in ERR_CPLEX leads to a reduced computing time, but this remains to be tested.

CherryPi, Leo and CUTTING determine all optimal solutions to each instance of the error localisation problem. This allows one to later use a more statistical criterion to select the "best" one. In contrast, ERR_CPLEX finds only one optimal solution to each instance of the error localisation problem. To find all optimal solutions we could – once an optimal solution to the current MIP problem has been determined – iteratively add an additional constraint, which basically states that the present optimal solution is excluded but other optimal solutions to the current MIP problem remain feasible, and solve the new MIP problem. This process of determining an optimal solution to the current MIP problem and adding an additional constraint to obtain a new MIP problem goes on until all optimal solutions to the error localisation problem have been found. We have not implemented this option, however. Resolving the problem from scratch for each optimal solution would be time-consuming. The alternative is to use a hot restart, where information generated to obtain an optimal solution to an MIP problem is utilised to obtain an optimal solution to a slightly modified MIP problem quickly. A problem with this possibility is that experiences at Statistics Netherlands with ILOG CPLEX so far, on linear programming (LP) problems arising in statistical disclosure control, show that ILOG CPLEX becomes numerical unstable if too many hot restarts in a row are applied.

The results of ERR_CPLEX are therefore only indicative. If the algorithms we have developed ourselves were clearly outperformed by ERR_CPLEX, this would suggest that standard MIP-solvers may be preferable to our algorithms. In that case, further studies with an extended version of ERR_CPLEX that aims to find all optimal solutions to the error localisation problem instead of only one would still be needed, however.

An important aspect in the evaluation of an algorithm is the time required to implement it in a computer program. The easiest algorithm/program to implement is ERR_CPLEX. The program only has to transform data and user-specified metadata, such as edits, into optimisation problems in a format that can be interpreted by ILOG CPLEX. To solve these optimisation problems routines from ILOG CPLEX are used. A bit more complicated is CUTTING. The two most important steps are the elimination of variables and solving modified set-covering problems. Both steps are actually quite simple to implement. Implementing CUTTING required about two months for a non-professional programmer. Slightly more complicated is Leo as this involves implementing a recursive algorithm, which is difficult to debug. The most complicated program to implement is CherryPi as several "tricks" (see De Waal, 2003b) need to be implemented in order to make this program sufficiently fast. To implement CherryPi about three to four months were required for a non-professional programmer.

4. Computational results

For Leo and CUTTING we have performed two kinds of experiments per data set. In the first kind of experiments we have set the maximum cardinality $N_{\max}$ to six. For many realistic data sets setting $N_{\max}$ to six is a good option as for records containing more than six errors it is unlikely that automatic error localisation will lead to data of sufficiently high statistical quality. Possible exceptions are data sets that contain many missing values, such as data set A. In the second kind of experiments for Leo we have set $N_{\max}$ as high as possible without encountering memory problems for many, i.e. 20 or more, records. In the second kind of experiments for CUTTING we have removed a maximum cardinality all together. For ERR_CPLEX and CherryPi we have only performed experiments without a specified maximum cardinality.

The experiments have been performed on a 1500 MHz PC with 256 MB of RAM. This PC is connected to a local area network. Computing times may therefore be influenced by the amount of data that was transmitted through the network at the time of the experiments. To reduce and to estimate this influence we have performed five experiments per data set at various moments during the day. In Table 2 we have mentioned the average computing times of these experiments for the entire data sets, and between brackets the standard deviation of these computing times over the corresponding five experiments. Note that some programs, such as Leo, have a random aspect that also influences the computing time. This random aspect is reflected in a relatively high standard deviation.

Table 2. Average computing times of the error localisation algorithms in seconds (between brackets the standard deviation of these computing times)

	Data set A	Data set B	Data set C	Data set D	Data set E	Data set F
ERR_CPLEX[a]	233 (1)	10 (0)	86 (1)	93 (9)	13 (0)	35 (0)
CherryPi	570 (38)	96 (1)	540 (7)	498 (30)	622 (3)	79 (0)
CUTTING	601 (17)	513 (12)	1913 (7)	1101 (20)	90 (1)	94 (2)
CUTTING ($N_{\max}$ = 6)	156 (10)	395 (23)	695 (31)	1036 (137)	50 (2)	92 (5)
Leo[b]	18 (0)	308 (10)	531 (4)	21 (1)	59 (34)	7 (0)
Leo ($N_{\max}$ = 6)	7 (0)	51 (1)	94 (2)	19 (0)	4 (1)	8 (1)

[a] These tests were performed on a special server. On this PC the only fully licensed version of ILOG CPLEX at Statistics Netherlands has been installed. For comparison reasons we have also used CherryPi for data set A on this machine. The average computing time for data set A on this machine is 528 seconds (compared to 570 seconds on the usual PC) with a standard deviation of 0 seconds. To compare the computing times of ERR_CPLEX to those of the other programs, we have therefore multiplied the original computing times on the special server by a factor of 570/528 = 1.08.

[b] To find the results for Leo for $N_{\max} > 6$, we have set $N_{\max}$ equal to 90 for data set A, to 8 for data sets B and C, and to 12 for data sets D, E and F.

Data set F contains only eight records for which six or more changes are required. The computing times of Leo and CUTTING are therefore almost equal to the computing times of Leo_6, respectively CUTTING_6 (i.e. Leo, respectively CUTTING with $N_{\max}$ = 6). In fact, due to the stochastic variability in the computing times Leo even outperformed Leo_6 in our experiments. Taking the standard deviation of the experiments into account, Leo and Leo_6 are about equally fast.

Due to numerical and memory problems, the programs could not always determine solutions. None of the programs can guarantee to find (all) optimal solutions for all records. For ERR_CPLEX, CherryPi, Leo, and CUTTING we have listed in Table 3 below for each data set the number of records for which these programs could not determine solutions to the error localisation problem. For all data sets, Leo_6 and CUTTING_6 found all optimal solutions for all records requiring six or less changes. Especially for data set A, this was very easy as there is only one record in data set A that has six or fewer errors or missing values (see Table 1)

Table 3. Number of records for which no solution could be found

	Data set A	Data set B	Data set C	Data set D	Data set E	Data set F
ERR_CPLEX	3	0	0	0	0	0
CherryPi	0	2	11	8	2	7
CUTTING	0	1	93	0	0	2
Leo[a]	0	1	58	0	53	0

[a] To find the results for Leo, we have set $N_{\max}$ equal to 90 for data set A, to 8 for data sets B and C, and to 12 for data sets D, E and F.

For 9 of the 58 records of data set C for which Leo_8 could not find a solution and for 13 of the 53 records of data set E for which Leo_12 could not find a solution, Leo suffered from memory problems. Those 9, respectively 13 records were excluded from the computational results for Leo in Table 2. As far as we have been able to determine, excluding these records from the computational results does not have a large effect and does not change the overall picture. Leo_8, respectively Leo_12, could not find solutions for the other records referred to in Table 3, because more than 8, respectively 12, changes were required.

Comparing the evaluation results of the various programs to each other is a complex task. If we rank the algorithms according to their computing times, and compare ERR_CPLEX, CherryPi, Leo (with $N_{\max} > 6$) and CUTTING with each other we see that ERR_CPLEX performs best for three out of the six data sets and second best for the other three data sets. Leo (with $N_{\max} > 6$) performs best for three out of six data sets and second best for two data sets. So, one might conclude that – purely looking at of the computing times – ERR_CPLEX is slightly better than Leo. Clearly worst is CUTTING.

Now, if we compare ERR_CPLEX, CherryPi, Leo_6, and CUTTING_6, and again rank the programs according to their computing times, we see that ERR_CPLEX performs best for two out of the six data sets and second best for three data sets. Leo_6 performs best for four out of six data sets and second best for the other two data sets. Here one might conclude that – purely looking at of the computing times – Leo_6 is better than ERR_CPLEX. The performances of CherryPi and CUTTING_6 are about equally good.

As we already mentioned in Section 3, for the two data sets for which ERR_CPLEX is faster than Leo_6, data sets B and C, we have applied a special version of Leo in which equalities are handled more efficiently. The results are given in Table 4. In this table we have mentioned the average computing times for the entire data sets, and between brackets the standard deviation of these computing times.

Table 4. Average computing times for Leo (in seconds) with more efficient handling of equalities (between brackets the standard deviation of these computing times)

	Data set B	Data set C
Leo[a]	308 (10)	531 (4)
Leo with efficient handling of equalities[a]	14 (2)	77 (1)
Leo ($N_{\max}$ = 6)	51 (1)	94 (2)
Leo with efficient handling of equalities ($N_{\max}$ = 6)	4 (1)	19 (1)

[a] To find the results for Leo for $N_{\max} > 6$, we have set $N_{\max}$ equal to 8 for both data sets.

For data set B the version of Leo_8 with efficient handling of equalities could not find a solution for one record. For data set C the version of Leo_8 with efficient handling of equalities could not find optimal solutions for 58 records, just like the standard version of Leo. The version of Leo with efficient handling of equalities did not suffer from memory problems, however.

Examining the results of Tables 2 and 4, we can conclude that as far as computing speed is concerned ERR_CPLEX and Leo (either with $N_{\max} = 6$ or with $N_{\max} > 6$) are the best programs. We note at the same time, however, that this conclusion is not completely justified as ERR_CPLEX determines only one optimal solution whereas the other programs (aim to) determine all optimal solutions.

Comparing the results of Tables 2 and 4 we see that the special version of Leo that handles equalities more efficiently indeed has a reduced computing time, at least for the two data sets examined. With this rule, Leo_6 is clearly faster than ERR_CPLEX for all data sets.

Besides computing speed other aspects are, of course, important too. We note that all programs, even the commercially available ILOG CPLEX, suffer from numerical problems. In addition, Leo sometimes suffers from memory problems. Due to its matrix with a fixed maximum number of columns, CherryPi does not always determine optimal solutions. Instead, it sometimes determines a less good, suboptimal solution. Summarising, it is hard to give a verdict on the quality of the solutions found by the programs as the programs suffer from a diversity of problems.

5. Discussion

McKeown (1981), in the context of Special Transportation Problems and Pure Fixed Charge Transportation Problems, remarks that ‘It is unclear in any of these contexts as to what makes a problem “easy” or “difficult” to solve’. This remark has again been confirmed in the context of the error localisation problem. From the characteristics of the data sets it is hard to establish beforehand whether the corresponding instances of the error localisation problem will be “easy” or “hard”. We can even extend the remark of McKeown to the following: it is unclear in our context as to what makes an algorithm a “good” or “bad” one. All algorithms we have examined have their good and bad aspects. In the end, the algorithm one favours is to some extent a subjective choice.

From our own developed algorithms, we consider the branch-and-bound algorithm (see Chapter 8 of De Waal, 2003a) the most promising one for solving the error localisation problem. The main reason for our choice is the excellent performance of Leo for records with up to six errors. For such records it determines all optimal solutions very fast. We admit that for records with more than six errors the results of Leo become less good, just like the other algorithms. The program begins to suffer from memory problems, and the computing time increases. To some extent these problems can be overcome by treating the equalities more efficiently as is done in the special version of Leo (see Table 4). Besides we feel that records with many errors should not be edited in an automatic manner, but in a manual manner. That is, we feel that records with more than, say, six errors should be rejected for automatic editing. Given this point of view, Leo seems to be an excellent choice.

In addition, it is not very complex to extend the branch-and-bound algorithm of Leo to a mix of categorical and continuous data. Statistics Netherlands has therefore decided to implement this algorithm in a module of version 1.5 (and future versions) of the SLICE system (see De Waal, 2001). This version reads an upper bound for the number of missing values per record as well as a separate upper bound for the number of errors (excluding missing values) per record. The former number is allowed to be quite high, say 50 or more, whereas the latter number is allowed to be moderate, say 10 or less. If the number of missing values or the number of errors (excluding missing values) in a record exceeds either of these upper bounds, this record is rejected for automatic editing. The new module is suitable for a mix of categorical and continuous data, and treats the equalities in an efficient manner. In addition, it contains a heuristic to handle integer data. The new module replaces the CherryPi-module, based on vertex generation, of SLICE 1.0.

One may argue that some users of SLICE will want to edit records with many erroneous fields, say 10 or more, automatically despite our arguments against editing such extremely contaminated records. Such users might then be disappointed, because the new module is not able to handle such records. To overcome this problem, we propose to opt for a simple heuristic treatment of these extremely erroneous records instead of applying the new module. We sketch three possible heuristics below.

The first heuristic is to split the set of edits into two subsets. First, we can apply the branch-and-bound algorithm on one of these subsets. One of the optimal solutions for this subset is chosen, and the corresponding fields are set to missing. Subsequently, we apply the branch-and-bound algorithm on the newly created record with possibly some additional missing values in comparison to the original record, using all edits. The solutions obtained in this way are, possibly suboptimal, solutions to the error localisation problem for the original record. This approach utilises the fact that the branch-and-bound algorithm works quite well for records with missing values.

A second approach, which utilises the same fact, to solve the error localisation problem for a record with many errors is to first determine a number of implausible values in a heuristic manner. These implausible values are set to missing. Subsequently, we apply the branch-and-bound algorithm on the newly created record with some additional missing values in comparison to the original record. The solutions obtained in this way are again, possibly suboptimal, solutions to the error localisation problem for the original record. Chung, a trainee at Statistics Netherlands, has studied this heuristic approach for solving extremely erroneous records (see Chung, 2003).

The final heuristic we mention consists of solving an LP approximation for the error localisation problem (see Section 13.2 in De Waal, 2003a).

All in all we are confident that records with many errors do not pose a threat for us if we apply the branch-and-bound algorithm in practice. We are willing to admit that our choice for the branch-and-bound algorithm is to some extent a subjective choice, but we feel that our choice is a justifiable one.

References

Chernikova, N.V. (1964), Algorithm for Finding a General Formula for the Non-Negative Solutions of a System of Linear Equations. *USSR Computational Mathematics and Mathematical Physics* 4, pp. 151-158.

Chernikova, N.V. (1965), Algorithm for Finding a General Formula for the Non-Negative Solutions of a System of Linear Inequalities. *USSR Computational Mathematics and Mathematical Physics* 5, pp. 228-233.

Chung, W.H. (2003), *Effectief Automatisch Opsporen van Foute*n (*Effective Automatic Error Localisation*), Internal report (BPA number: 1057-03-TMO), Statistics Netherlands, Voorburg.

De Jong, A. (2002), Uni-Edit: *Standardized Processing of Structural Business Statistics in the Netherlands*. UN/ECE Work Session on Statistical Data Editing, Helsinki.

De Waal, T. (1996), *CherryPi: A Computer Program for Automatic Edit and Imputation*. UN/ECE Work Session on Statistical Data Editing, Voorburg.

De Waal, T. (2000), *New Developments in Automatic Edit and Imputation at Statistics Netherlands*. UN/ECE Work Session on Statistical Data Editing, Cardiff.

De Waal, T. (2001), SLICE: Generalised Software for Statistical Data Editing. *Proceedings in Computational Statistics* (eds. J.G. Bethlehem and P.G.M. Van der Heijden), Physica-Verlag, New York, pp. 277-282.

De Waal, T. (2003a), *Processing Erroneous and Unsafe Data*. Ph. Thesis, Erasmus University Rotterdam.

De Waal, T. (2003b), Solving the Error Localization Problem by Means of Vertex Generation. To be published in *Survey Methodology*.

Fellegi, I.P. and D. Holt (1976), A Systematic Approach to Automatic Edit and Imputation. *Journal of the American Statistical Association* 71, pp. 17-35.

Fillion, J.M. and I. Schiopu-Kratina (1993), *On the Use of Chernikova's Algorithm for Error Localisation*. Report, Statistics Canada.

Garfinkel, R.S., A.S. Kunnathur and G.E. Liepins (1986), Optimal Imputation of Erroneous Data: Categorical Data, General Edits. *Operations Research* 34, pp. 744-751.

Garfinkel, R.S., A.S. Kunnathur and G.E. Liepins (1988), Error Localization for Erroneous Data: Continuous Data, Linear Constraints. *SIAM Journal on Scientific and Statistical Computing* 9, pp. 922-931.

Hoogland, J. (2002), *Selective Editing by Means of Plausibility Indicators*. UN/ECE Work Session on Statistical Data Editing, Helsinki.

ILOG CPLEX 7.5 Reference Manual (2001), ILOG, France.

Kovar, J. and W.E. Winkler (1996), *Editing Economic Data*. UN/ECE Work Session on Statistical Data Editing, Voorburg.

McKeown, P.G. (1981), A Branch-and-Bound Algorithm for Solving Fixed Charge Problems. *Naval Research Logistics Quarterly* 28, pp. 607-617.

Ragsdale, C.T. and P.G. McKeown (1996), On Solving the Continuous Data Editing Problem. *Computers & Operations Research* 23, pp. 263-273.

Rubin, D.S. (1975), Vertex Generation and Cardinality Constrained Linear Programs. *Operations Research* 23, pp. 555-565.

Rubin, D.S. (1977), Vertex Generation Methods for Problems with Logical Constraints. *Annals of Discrete Mathematics* 1, pp. 457-466.

Sande, G. (1978), *An Algorithm for the Fields to Impute Problems of Numerical and Coded Data*. Technical report, Statistics Canada.

Schiopu-Kratina, I. and J.G. Kovar (1989), *Use of Chernikova's Algorithm in the Generalized Edit and Imputation System*. Methodology Branch Working Paper BSMD 89-001E, Statistics Canada.

Van Riessen, P. (2002), *Automatisch Gaafmaken met Behulp van CPLEX* (*Automatic Editing by Means of CPLEX*). Internal report (BPA number: 975-02-TMO), Statistics Netherlands, Voorburg