Transactions on Data Privacy 6 (2013) 147–159

Analysis of Commercial and Free and Open Source Solvers for the Cell Suppression Problem

Bernhard Meindl[*,***], **Matthias Templ**[*,**,***]

[*]data-analysis OG, Bergheidengasse 8/1/12 Vienna, 1130, Austria.

[**]Vienna University of Technology, Wiedner Hauptstr. 7, Vienna, 1030, Austria.

[***]Statistics Austria, Guglgasse 13, Vienna, 1110, Austria.

E-mail: bernhard.meindl@gmail.com, matthias.templ@gmail.com

Abstract. In this contribution, software tools that can be used to solve (mixed integer) linear optimization problems are described and compared. These kind of problems occur for instance when solving the secondary cell suppression problem (CSP) for which we tested the tools.

Especially, for the CSP fast and efficient tools are needed. While experience gained in different projects on confidentiality regarding particular commercial or open-source tools, we aim to compare the relevant tools at once based on this problem.

An overview of existing comparisons of both open-source and commercial solvers is given. Moreover, the performance of different solvers is evaluated on the basis of solving multiple attacker problems - a linear problem - in a case study. This problem class is important when dealing with the CSP.

Keywords. statistical disclosure limitation, secondary cell suppression, linear programming, software

1 Introduction

Statistical agencies generally do not publish microdata to the public, but disseminate information in form of aggregated data. Aggregated data are usually represented as statistical tables with totals in the margins.

Anonymization of such tables provided to the public is a major task of statistical agencies.

Due to the linear relationships in tables it is not sufficient to protect tables by identifying (primary) sensitive cells (that may lead to identify a person or establishment) and suppressing its values. In order to avoid the recalculation or the possibility to gain good estimates for sensitive cells, additional cells have to be suppressed or perturbed. The problem of finding additional cells to be suppressed is called secondary cell suppression problem. Note that also other but similar methods can be used to provide confidentiality in tables, such as controlled tabular adjustment, but cell suppression is still the most popular choice for statistical agencies, which can be proofed simple by looking at the published tables from them.

The task to find and identify cells that are needed to be suppressed or changed in order to protect the primary sensitive cells is complex and computer-intensive since in statistical practice statistical tables are often hierarchical, multidimensional and/or linked.
The secondary cell suppression problem (CSP) [formally defined in, e.g., 6] consists of selecting a set of non primary sensitive cells so that it is not possible for an attacker to calculate values of any primary sensitive table cells within given bounds [see also 20]. Solving this problem is a complex task, since tables are usually multidimensional and, in addition, the variables used for tabulation can possess a hierarchical structure.

Due to the huge number of constraints and variables that are required when modeling the CSP directly as a mixed linear integer problem, this approach is only feasible for small problem instances. Therefore, proposed algorithms [20, 6] heavily depend on solving linear programs to derive a solution to the CSP. The quality of the solution and the required time needed to obtain solutions is of cruicial importance.

For this reason, it is clear that software highly depends on the choice of the underlying linear program (LP) solver. Fortunately, various solvers exist that are able to solve specific linear optimization problems.

In section 2 we give an overview of existing free and open source solvers as well as commercial software. We go on and review the literature where the performance of different LP-solvers is discussed in section 3. In section 4 we perform a small case study in which we compare the performance of selected solvers (both free and commercial) given a specific kind of problem class that is common when solving the secondary cell suppression problem. Finally, we review and discuss results in section 5.

2 LP-solver

Available LP-solvers differ in many ways. They come with different licenses and of course different features, for example in terms of how problems can be specified. Detailed information on most available solvers, its licenses as well as brief information is available at wikipedia [19, Linear Programming].

We now continue and list the most popular free and open source solvers below.

Popular and well-known free and open source solvers:

GLPK: the GLPK (GNU Linear Programming Kit) [12] is a free and open source software written in ANSI C that allows to solve (mixed integer) linear programming problems.

LP_SOLVE: lp_solve [2] is also an open source solver that can be used to solve linear (mixed integer) programs and is written in ANSI C.

CLP: is a solver created within the Coin-OR project [8, 4] written in C++ to tackle linear optimization problems. The Coin-OR project aims at creating open software for the operations research community. CLP is used in many other projects within Coin-OR such as SYMPHONY (library to solve mixed-integer linear programs), CBC (an LP-based branch-and-cut library) or BCP (framework for constructing parallel branch-cut-price algorithms for mixed-integer linear programs).

SCIP: [1] is a framework for solving integer and constraint programs which is available as a C callable library or a standalone solver. SCIP has also open LP solver support which means that it is possible to call a range of open source and commercial LP solvers. The codebase of SCIP is freely available and the framework can be used without restrictions for academic research but not for commercial use.

SoPlex: SoPlex [11] is a Linear Programming solver that is based on the revised simplex algorithm and has been implemented in C++. The source code is publically available and the solver can be used as a standalone solver or it can be embedded into other programs using a C++ class library. The source code is released and the solver can be used for free for academic research. For commercial use of the solver it is necessary to obtain a licence.

All these solvers (with small exceptions regarding SoPlex and SCIP) can be used without any restrictions in any software since the source code is released public domain and usually very well documented. Thus it is possible to compile the solvers on different platforms and architectures. It should also be noted that almost all open source programming kits also have some kind of structured API [18, API] that can be used to exchange data between any software product and a LP-solver. Additionally, for some LP-solvers the available API is already exploited in a way that it is possible to call the solvers from `R` [16].

Popular and well-known commercial Lp-Solvers:

Cplex: The IBM ILOG CPLEX Optimization Studio [10] which is often referred to simply as Cplex is an commercial solver designed to tackle (among others) large scale (mixed integer) linear problems. Cplex is now actively developed by IBM. The software also features several interfaces so that it is possible to connect the solver to different program languages and programs. However, also a stand-alone executable is provided. The optimizer is also accessible through modeling systems.

Xpress: The Xpress Optimization Suite [5] (also referred to simply as Xpress) is a commercial, proprietary software that is designed to solve (among others) (mixed integer) linear problems.
Xpress is available on most common computer platforms and also provides several interfaces including a callable library APIs for several programming languages as well as a standalone command-line interface.

Gurobi: The Gurobi Optimizer [7] is a modern solver for (mixed integer) linear as well as other related (non-linear, e.g.) mathematical optimization problems. The Gurobi Optimizer is written in C and it is available on all computing platforms and accessable from several programming languages. Standard independent modelling systems can be used to define and to model problems.

It is beyond the scope of this report to list all features of different optimization solvers in detail. This information is documented in great detail on the corresponding product pages. It is however safe to say that all solvers differ in terms of licenses, costs and also in the way in which they can be called as well as in the algorithms that are incorporated to solve (mixed integer) linear problems.

Since the differences regarding the solvers are large, it is of great interest to obtain information, for example, about problem sizes that can be solved in reasonable time even with non-commercial solvers. This would help e.g. to derive suggestions for problem sizes with respect to the the secondary cell suppression problem for which it might not be possible to obtain solutions using any open source solvers and for which commercial solvers need to be used.

More generally speaking, comparing the performance of different solvers on different platforms applied to (difficult) standard problems has already been conducted. Therefore, we now continue to summarize some results of such a study in the next section.

3 Review of performance comparisons

One of the most informative sources for performance comparison of different (mixed integer) linear solvers applied to different problem classes is provided by Hans Mittelmann [14]. On this webpage several instances of linear problems using serial and parallel solvers are benchmarked using real world testcases from the Netlib repository [15].

In this section we will focus to analyze results for mixed integer problems, i.e. some (or all) variables of a problem are of type integer (or binary). The reason for this approach is that solving mixed integer linear programs using standard algorithms such as branch and bound or branch and cut, various (relaxed) linear problems need to be solved too. Thus, comparing the performance of different solvers for mixed integer linear problems also gives hints about the performance of solvers when dealing with problems that only contain continuous variables.

On his webpage, Mittelmann provides benchmarks for mixed integer linear problems. The testcases are taken from the MIPLIB library [3]. The problem instances available in this library provide - among others - a set of real world mixed integer linear problems that differ in complexity and can be used to compare different solvers. It should be noted that not only the results but also log-files of the server run for each server and problem instance are made available so that results can be reproduced since the settings used for each solver that has been applied are known. Also, the system archictecture on which the testcases have been run is reported. Furthermore results and running times are listed.

The benchmark test data set which is the base of the results discussed below consists of a total of 87 problem instances that can be solved to optimality by at least one solver within two hours on a high-end personal computer. All these problems can be classified into one of the following categories:

- BP: all variables are binary
- IP: all variables are integer
- MBP: all variables are binary or continuous
- MIP: all variables are integer or continuous

We now want to try to summarize some of the results found without being too technical. The aim of this section will be to outline main findings and trends of the benchmark com-

parisons with respect to commercial vs. non-commercial or rather open source solvers.

The following data are based on data from the 28th of April 2013. The following solvers have been used in the test on PC with an Intel Xeon X5680 processor with 2*6 cores, 32 gigabytes of RAM on a linux 64bit operating system. A total of 9 different solvers were compared using a single thread[1]. The solvers are listed along with the corresponding version numbers below:

- CPLEX (12.4.0.0)
- GUROBI (4.6.1)
- SCIP-C (2.1.1 using CPLEX as LP-solver)
- SCIP-L (2.1.1 using CLP as LP-solver)
- SCIP-S (2.1.1 using SoPlex as LP-solver)
- CBC (2.7.4)
- XPRESS (7.2.1)
- GLPK (4.47)
- LP_SOLVE (5.5.2)

	running time	scaled running time	instances solved	solved (%)
CBC	1707.00	13.20	35	40.23
CPLEX	148.30	1.15	78	89.66
GLPK	3466.00	26.90	3	3.45
GUROBI	129.00	1.00	79	90.80
LP_SOLVE	3034.00	23.40	5	5.75
SCIP-C	589.70	4.75	62	71.26
SCIP-L	1001.00	7.76	52	59.77
SCIP-S	770.50	5.79	62	71.26
XPRESS	167.00	1.29	78	89.66

Table 1: Results for different solvers applied to benchmark problem instances

Table 1 shows the main results. In this table the first column shows the solver used. In the second column the running times (geometric mean for all corresponding instances, in seconds) are shown for each solver. The maximal running time was limited to 60 minutes. In cases in which a solver failed to find an optimal solution within the arbitarily chosen time limit of one hour, we used this time limit in further calculations such as the computation of the geometric mean of running times. Finally, the times were modified in a way that the fastest solver (GUROBI in this example) was scaled to 1. Figure 1 displayes the performance with respect to running times. In this case smaller values are of course better.

The third column of Table 1 displays the number of problem instances that have been successfully solved by each solver and in the last column the corresponding percentage is

[1] data based on 4 and 12 threads are available too for a restricted set of solvers [14]

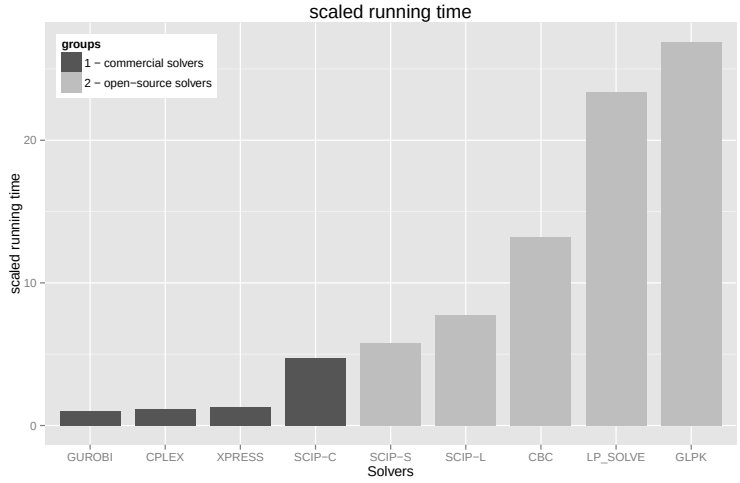

Figure 1: Running times of several solvers applied to benchmark test data

listed. This information is shown graphically in Figure 2 in which higher percentages equal better performance.

Figures 1 and 2 clearly show that commercial solvers (displayed in black) outperform open source solvers (displayed in grey) not only in terms of running time but also with respect to the percentage of solved problem instances when the results of all 87 problem instances are used. It should be noted however that in some cases open source solvers provide an optimal solution faster than commercial solvers. For example LP_SOLVE was able to obtain an optimal solution for problem instance 'eil33-2' (`http://miplib.zib.de/miplib2010.php`) in 65 seconds while it took GUROBI 124 seconds to derive an optimal solution to this problem. However, this instance can be considered as an easy to solve feasible problem with constraints of size 32 columns and 4516 rows.

A very interesting fact is that GLKP and LP_SOLVE only manage to calculate optimal solutions for 3 or rather 5 of the 87 problem instances while closed source solvers (GUROBI, XPRESS, CPLEX, SCIP-C) manage to obtain an optimal solution to at least 72% of the instances within an hour. But even when using commercial and expensive solvers there is no guarantee to obtain results for specific problems in adequate time. It is also worth to note that the three purely closed source and commercial solvers (GUROBI, XPRESS, CPLEX) are quite similar in their performance with respect to both running time and number of problems solved.

In Section 4 we will now test the performance of GUROBI, CLPEX, SCIP-S, CBC, GLPK and LP_SOLVE in a small case study with special interest to the secondary cell suppression problem. Because of licencing issues it was not possible to compare all 9 solvers but the chosen 5 solvers represents state-of-the-art commercial and free solvers.

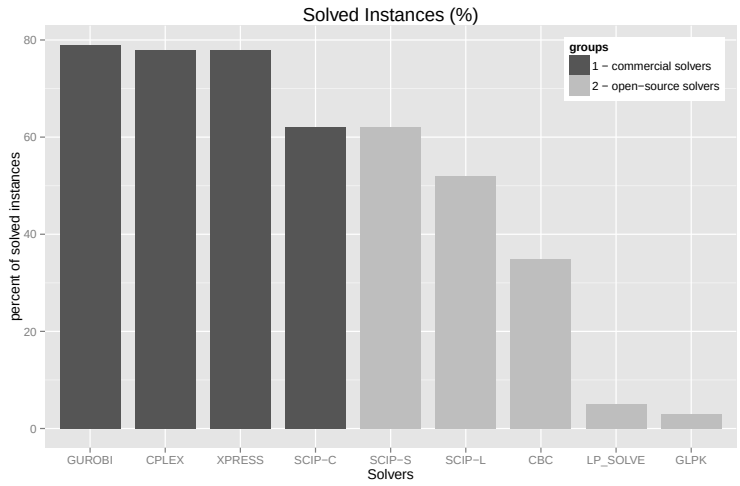

Figure 2: Percentage of solved instances for several solvers applied to benchmark test data

4 Case study

In this section we describe the set up of the case study we have performed using `R` and discuss the results we have derived. The general idea was to perform the case study in a similar way than it has been done by [14].

4.1 The CSP and the attacker's problem

The focus is given on linear problems only since solving the secondary cell suppression problem requires to solve the so called Attacker's problem [6] repeatedly. The Attacker's problem however does not require any variables in the problem to be integer or binary.

A multidimensional hierarchical table can be expressed by a data vector $\boldsymbol{a} = (a_1, \ldots, a_n)$, constraints $\boldsymbol{My} = \boldsymbol{b}$ and upper and lower bounds $lb_i < a_i < ub_i$, with $i = 1, \ldots, n$, the index expressing the cells of a table. Note that the matrix $\boldsymbol{M}$ can become considerable large for real-world problems. The aim is to find an optimal suppression pattern so that an attacker can only calculate an suffiencient large upper and lower bound for primary suppressed cells. For the mathematical notation of the object function and the notation of all constraints we refer to [6]. The attacker itself knows the anonymised table including the complete suppression pattern (SUP), i.e. the attacker knows $y_i = a_i \forall i \notin SUP$ and solves $\boldsymbol{My} = \boldsymbol{b}$ to obtain upper and lower bounds $\hat{lb}_i < y_i < \hat{ub}_i$ whereas $\hat{lb}_i \leq lb_i$ and $\hat{ub}_i \geq ub_i$, $\forall i$, otherwise the table is not save.

Typically, the secondary cell suppression problem is solved using a branch and bound or branch and cut algorithm. Thus, the performance of solvers for linear problems is crucial. To compare the performance of GLPK, LP_SOLVE, CLP, GUROBI and CPLEX we set up the case study as it is described now.

4.2 Set up

Using `sdcTable` [13] we have randomly generated a total of 100 2-dimensional and 100 3-dimensional problem instances. The problem instances differ with respect to the hierarchical structure of the variables defining the tables. The dimensions feature up to 4 levels and differ in the number of codes within each level. In section 4.4 we give a summary on key statistics of the problem instances that have been generated.

In order to be able to measure time we have opted to take the approach to generate static output-files in a standard format that can be used as an input for the linear program solvers under consideration. These text files are read into the solver and the output-files are saved. We then run code to solve all problems for each solver and measure the time it was needed to solve the problems. However, we also note how many of these Attacker's problem need to be solved for a given problem instance in one run. This number equals to the two times the number of primary sensitive table cells because the solution to the Attacker's problem gives the upper and lower bound that an attacker can derive for a specific primary sensitive table cell given the structure of the table and (sensitive) bounds on suppressed table cells.

Thus, the time that is required for a specific solver to solve the Attacker's problem for a given problem instance is calculated as the time it took to solve the 2 problems related to the lower and upper bound for a given cell times times the number of primary sensitive table cells for the specific problem instance. This is of course a simplification step that has been made.

In section 4.3 we give an overview of the system that has been used to perform this case study. We also give information on the solvers that have been used.

4.3 System information and solver information

This case study has been performed on a dual-core system featuring an intel Core2 Duo. Each of the two cores runs at a maximum speed of 2.40GHZ and has a 4096KB cache. The system has 20GB of RAM and runs a 64-bit linux operating system with kernel 2.6.34. The simulation script has been written in `R` and the computations have been performed using `R` in version 2.13.0.

The linear solvers GLPK, LP_SOLVE and CLP have been downloaded from the corresponding webpages in their last versions (4.47 for GLPK, 5.5.2 for LP_SOLVE and 1.14.6 for CLP) and have been compiled using the GNU Compiler Compilation gcc in version 4.5 using the default settings. CPLEX was downloaded in version 12.3.0.0 and an academic license has been used that does not restrict the use of the solver in any way. GUROBI was also used under an academic license in the latest stable version of the software, 4.6.1 in this case.

In principle it would have been possible to include solvers with different versions to benchmark the performance changes between versions, but this work was clearly out of scope for this contribution. We now go on to discuss some main features of the Attacker problems that have been generated and were used in this case study.

4.4 Lp-Problems

As it has already been indicated we considered a total of 200 problem instances for this case study. 100 of these problem instances are based on problems with two dimensions and 100 problem instances are three-dimensional.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
nrConstraints	9	93.2	205	317.7	354.0	3072
nrVariables	24	238.2	498.5	788.5	893.2	7520
nrPrimSupps	1	4.8	12	19.4	21.5	190

Table 2: Summary statistics for 2-dimensional problem instances

Table 2 shows that the number of constraints that form the basis of the Attacker's problem range from 9 to 3072 for the two-dimensional problem instances while the number of variables goes from 24 to 7520 with the mean being 788.5. The mean number of primary sensitive table cells in these problem instances is 19.4 with the minimum being 1 and the maximum being 190 primary suppressed table cells.

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
nrConstraints	144	1119	3068	4877	7272	19800
nrVariables	438	3186	8387	13240	19680	53930
nrPrimSupps	5	40	126	214.4	320	853

Table 3: Summary statistics for 3-dimensional problem instances

One can observe from Table 3 that the average number of constraints with the mean being 4877 is larger than for the two-dimensional problem instances. This is however an obvious fact since the number of constraints represents the complexity of the underlying table, also for the Attacker's problem.

One can also observe that the number of variables ranges from 438 to 53930 with the mean number of variables being 13240. The mean number of primary sensitive table cells given three-dimensional problem instances is 214.4 and therefore higher as for the two-dimensional test cases (19.4).

In Section 4.5 we will now discuss the results that were obtained when solving the simulated problems with 5 different linear program solvers.

4.5 Results

Tables 4, 5 and 6 compares the running times that it took each of the five linear program solvers to find solutions to the 200 two-dimensional, 200 three-dimensional and all 400 problem instances. The reason that a total of 400 problem instances had to be solved when only 200 problem instances have been generated is that in order to solve the Attacker's problem for a given table cell and problem instance it is required to both minimize and maximize any given problem.

Data on running times have been scaled in a similar way as it has been done by [14]. The running time of the fastest solver has been scaled to one and the running times of the other linear solvers were scaled to reflect this scaling.

It should be noted that every solver was able to provide a solution to each problem. Thus it was not necessary to introduce some kind of penalty time. All aggregated results that are listed in the following tables correspond to observed running times.

We can learn from Table 4 that CPLEX had the best performance which means the fasted running time. It took the other commercial solver (GUROBI) 4 times as long to compute solutions for the two-dimensional test-cases than CPLEX. The performances of GLPK and CLP ar comparable to each other, however the running times are considerably slower than the performance of CPLEX. The results also show that LP_SOLVE had by far the worst performance, even for this quite simple two-dimensional problem instances.

	glpk	lpsolve	clp	gurobi	cplex
total	180.1	2861.9	273.9	73.6	20.9
scaled	9	137	13	4	1

Table 4: Total (in seconds) and scaled running times for 2-dimensional problem instances

Table 5 lists results of scaled running times for the generated three-dimensional problem instances. One can see that also for this class of problems CPLEX performed best. What is interesting is that even though GUROBI performed better than the three remaining open source solvers, the performance of GLKP is quite comparable to GUROBI. Both CLP and especially LP_SOLVE performed worse than the two commercial solvers and also when compared to GLKP.

	glpk	lpsolve	clp	gurobi	cplex
total	48405	979876	666792	38634	236
scaled	205	4149	2823	164	1

Table 5: Total (in seconds) and scaled running times for 3-dimensional problem instances

Table 6 shows scaled aggregated performances for all two- and three-dimensional problem instances together. The interpretation of the results is of course similar to the discussion of the results above - done with respect to two- and three-dimensional test cases seperately.

The second best performance next to CPLEX was once again by GUROBI, the second commercial solver. LP_SOLVE once again performed worst, with CLP being only a bit better. GLPK was had the best performance of the open source solvers but still was clearly slower than CPLEX and also GUROBI.

The results we have now discussed clearly indicate that open source solvers perform worse than standard commercial solvers when applied to instances of the Attacker's problem. In Section 5 we will not continue to talk about possible implications of the small case study that has been conducted.

	glpk	lpsolve	clp	gurobi	cplex
total	48585	982737	667066	38708	257
scaled	189	3822	2594	151	1

Table 6: Total (in seconds) and scaled running times for all problems

5 Discussion

Comparing the free solvers, GLKP was significantly faster than LP_SOLVE and CLP in our case study but is slowest in [14] (see section 3). In addition, we obtained huge differences in the 3-dimensional case between commercial solvers, whereas CPLEX was considerable faster. This is also in contradiction to results from [14] that we presented in section 3. We conclude no general tests may reasonable to evaluate the cell suppression problem, i.e. the application shown in this paper let us give different conclusions.

The main conclusion obtained from this case study is that with real world scaled problem sizes, commercial solvers outperform open source solvers in solving the secondary cell suppression problem. This conclusion is not new since for example τ-Argus [9] already depends on XPRESS to calculate solutions for the CSP. However, small problems can be solved using free and open source solvers as well, since here the time needed to solve the problem is reasonable anyway, although the best free solver that has been compared (GLKP) is significantly slower than the commercial ones (which is in contradiction to a similar study by [17]). However, the reality especially in National Statistical Offices is that most problems experts in this field are dealing with are large.

Of course it would be benefical if open source solvers were available that have similar performance than commercial solvers since commercial products are expensive. However, this is not (yet) the case. So NSIs will have to either invest money and buy licenses for commercial solvers such as CPLEX, GUROBI or XPRESS or they will face problems in either obtaining solutions for given cell suppression problems or at least obtaining solutions in reasonable time.

The implications for further development of software are quite clear as well. New and improved software that tackles to find solutions to the secondary cell suppression problem should be written in a modular way. This would mean that any solvers can be plugged in. This would reduce the dependence on specific solvers since any commercial or open source solver could be used to solve the required (mixed integer) linear problems that occur when solving the CSP.

Ideally, such an approach would allow NSI's that already have licenses for commercial solvers to use those as well as NSI's that do not want to invest money on commercial software could still try to get solutions for given problems. If software is designed in a clear, abstracted way it will also be possible to expand the software tool easily in order to use additional solvers in case new solvers emerge.

It should however be made clear to any user of a software that using only free and open source solvers such as GLKP, CLP or LP_SOLVE will likely not lead to a fast solution of

their problem if any. An additional reason that has not been studied in the small case study for this contribution is that most of the available open source linear program solvers do not allow to use them in parallel way on multi cpu-systems. This is a further drawback since machines with 20+ cpu cores are getting cheaper and not using the available processing power is of course bad. Commercial solvers such as CPLEX and GUROBI do a much better job in this setting too, as it was for example shown by Mittelmann [14] for mixed linear integer test cases.

Interfaces to the discussed solvers are provided by the `R` package `sdcTable` [13] that is distributed for free at the comprehensive `R` archive network.

Acknowledgement: This work was funded by Eurostat and Statistics Netherlands within the project *ESSnet on common tools and harmonised methodology for SDC in the ESS*. Visit http://neon.vb.cbs.nl/casc/ESSNet2index.htm for more information on the project.
We would like to thank two anonymous reviewers for their constructive and helpful reviews.

References

[1] T. Achterberg. SCIP - a framework to integrate Constraint and Mixed Integer Programming. Technical Report 04-19, Zuse Institute Berlin, 2004.

[2] M. Berkelaar, K. Eikland, and P. Notebaert. lp_solve 5.5, open source (mixed-integer) linear programming system. Software, 2004. URL `http://lpsolve.sourceforge.net/5.5/`.

[3] T. Berthold, G. Gamrath, D. Steffy, and T. Koch. Mixed Integer Problem Library. Software, 2012. URL `http://http://miplib.zib.de/`.

[4] CoinOR. COmputational INfrastructure for Operations Research. Software, 2012. URL `http://www.coin-or.org`.

[5] Fico. Xpress Optimization Suite. Software, 2012. URL `http://www.fico.com/en/Products/DMTools/Pages/FICO-Xpress-Optimization-Suite.aspx`.

[6] M. Fischetti and J-J. Salazar. Solving the cell suppression problem on tabular data with linear constraints. *Management Science*, 47(7):1008–1027, 2001.

[7] Gurobi Optimization Inc. Gurobi Optimizer. Software, 2012. URL `http://www.gurobi.com/welcome.html`.

[8] L.R. Heimer. The Common Optimization Interface for Operations Research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47, 2003.

[9] A. Hundepool. *tau-ARGUS - a software designed to protect statistical tables*, 2012. URL `http://neon.vb.cbs.nl/casc/tau.htm`. software version 3.5.0.

[10] IBM. IBM ILOG CPLEX Optimization Studio. Software, 2012. URL `http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/`.

[11] Konrad-Zuse-Zentrum für Informationstechnik Berlin. The Sequential object-oriented simPlex. Software, 2002. URL `http://soplex.zib.de/`.

[12] A. Makhorin. *The GNU Linear Programming Kit (GLPK)*. GNU Software Foundation, 2000. URL http://www.gnu.org/software/glpk/glpk.html.

[13] B. Meindl. *sdcTable: Methods for SDC (statistical disclosure control) in tabular data*, 2012. URL http://cran.r-project.org/package=sdcTable. R package version 0.10.2.

[14] H. Mittelmann. Benchmarks for optimization software. http://plato.asu.edu/bench.html, 2011.

[15] Netlib. Testcases for real world LP instances. Software, 2012. URL ftp://ftp.netlib.org/lp/data/index.html.

[16] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL http://www.R-project.org/. ISBN 3-900051-07-0.

[17] J-J. Salazar. On open source software for statistical disclosure limitation. In *Joint UNECE/Eurostat work session on statistical data confidentiality, Manchester, UK*, 2009.

[18] Wikipedia. Application programming interface — Wikipedia, the free encyclopedia, 2012. URL http://en.wikipedia.org/wiki/API. [Online; accessed 11-February-2012].

[19] Wikipedia. Linear programming — Wikipedia, the free encyclopedia, 2012. URL http://en.wikipedia.org/wiki/Linear_programming. [Online; accessed 11-February-2012].

[20] L. Willenborg and T. de Waal. *Statistical Disclosure Control in Practice*. Springer, 1996.