Assessing the Performance of the Open-Source Linear Programming Solver in Cell Suppression Problems

Haoluan Chen, Steven Thomas
Statistics Canada, 150 Tunney's Pasture Driveway, Ottawa, ON, K1A 0T6

Abstract
The current implementation of complementary cell suppression methodology at Statistics Canada relies on a linear programming (LP) solver finding the feasible solution in SAS. As an alternative, open-source LP solvers are being investigated. Among these solvers, it is not clear which one would perform better for the suppression problem until we actually use them and assess performance. Therefore, a Python version of suppression was implemented using open-source linear programming packages. There are several challenges in comparing the performance of solvers. For example, it is difficult to assess the solution of the linear programming problem since the heuristic method requires solving the LP problem sequentially. This presentation discusses the performance of alternative solvers in relation to typical suppression problems.

Key Words: Confidentiality, Cell Suppression, Linear Programming

1. Introduction

G-Confid[1] is a tabular confidentiality system developed at Statistics Canada. It can be used to identify sensitive cells, create and validate a complementary cell suppression (CCS) pattern for economic data at various aggregate levels. The current implementation involves solving a series of linear programming (LP) problems with the SAS/OR® LP solver to find a feasible solution. There are many alternative open-source LP solvers that may provide compatible performance. Statistics Canada has developed a Python prototype that integrates open-source LP packages as a means to evaluate the capabilities of these solvers.

The paper covers the linear programming formulation and implementation details for the CCS strategy as it is currently performed. A comparison between G-Confid and the Python prototype implementation in terms of speed and suppression quality will be detailed. Lastly, we will discuss the findings and challenges in developing and comparing the alternative solution.

2. Methodology of SUPPRESS

The main objective of the G-Confid macro SUPPRESS is to find complementary cells that provide the desired protection level to the sensitive cells (those presenting a disclosure risk) while minimizing the loss of information. The sensitive cells would have been identified using a linear sensitivity measure as discussed by Cox (1981). The loss of information could be represented by the number of cells being suppressed or the total value of the suppressed cells. To determine the CCS pattern, the macro is solving the following linear programming problem for each of the primary sensitive cell X_{sen}(Frolova 2013).

Matrix formulation of LP Problem (to find X and Y that minimizes the objective function):

$$\text{Minimize } W'X + W'Y$$
$$\text{Subject to } CX - CY = 0 \qquad \text{(Constraint 1)}$$
$$0 \le X, Y \le T \qquad \text{(Constraint 2)}$$

[1] Available for download at https://www150.statcan.gc.ca/n1/en/catalogue/10H0109.

$$x_{sen} \geq S_{sen} \quad \text{(Constraint 3)}$$
$$y_{sen} = 0 \quad \text{(Constraint 4)}$$

Where
W: n-dimensional vector of weight/cost, where n is the total number of cells in a table
X: n-dimensional vector of positive shifts to cell totals
Y: n-dimensional vector of negative shifts to cell totals
C: m × n matrix of coefficients, where m = number of linear equations generated by the table.
T: n-dimensional vector of cell totals
x_{sen}: the component of the vector X corresponding to the current sensitive cell
y_{sen}: the component of the vector Y corresponding to the current sensitive cell
S_{sen}: sensitivity of X_{sen}

For each sensitive cell, we are solving an LP problem. Given the sensitivity of a cell, which indicates the protection required, the LP problem is trying to find the cells that can provide the required amount of protection at the lowest cost. X and Y (considered 'shifts' to the table) are the variables in our objective function to which we want to assign values such that they minimize the objective function. There are also constraints to our problem. Constraint 1 forces the additivity of our table. Constraint 2 states that a cell can only provide protection up to its total and the shifts cannot be negative. Constraint 3 forces the positive shift of the current sensitive cell to be the sensitive value indicating the protection needed, then we find the corresponding shifts to maintain the additivity of the table. The last constraint forces Y for the current sensitive cell to be 0. In the end, all cells that are shifted are suppressed to protect the sensitive cell. The process is repeated sequentially for all the sensitive cells in the table.

Among all suppression patterns, we would like to select the one that minimizes the cost of suppression in the table. That is to minimize *W'X+W'Y*. Our objective function is to describe the cost of suppression. In G-Confid, we can choose from four values for the weight / cost vector corresponding to different costs of suppression of a cell. The use of different cost vectors will result in different suppression patterns. The "Constant" form of the objective function assigns an equal cost of suppression to each non-sensitive cell (w_i = 1), which may lead to suppressing fewer cells with a larger total. The "Size" objective function assigns a cell's total value as the cost ($w_i = t_i$), which leads to suppression of the smaller cells. The "Information" objective function assigns the cost to be $w_i = \frac{\log(1+t_i)}{1+t_i}$. This objective function leads to suppression of a small number of large cells, as it is a decreasing function of t_i. Lastly, the "Digit" objective function assigns the cost to be $w_i = \log(1 + t_i)$, which balances between the Size and Information objective function. The default objective function in G-Confid is "Digit". For any objective function, the cost of suppressing primary sensitive cells is equal to zero, because it must be suppressed to protect confidential data.

3. Performance Comparison

We reprogrammed G-Confid in Python based on the LP formulation described in section 2. We ran four test cases on an AMD EPYC 7763 64-Core Processor with 8 GB of RAM to assess the time took to solve the whole CCS problem. To compare the performance, we put timers throughout the code to capture the data processing time between solving LP problems as well as the time for solving the LP problem. We found that the data manipulation part between iterations was negligible. Also, note that the four test cases cover a variety of complexity in terms of size and number of primary sensitive cells. The optimality is measured using the number of cells suppressed and the total values suppressed.

3.1 Modeling Packages and Solvers

In order to solve the objective function described above a linear programing optimization solution is required. In G-Confid, we have relied on the SAS solution offered with PROC OPTMODEL[2]. This solution offered a reasonable CCS pattern in terms of optimality and speed. PuLP is an open-source LP modeling Python package that allow user to connect to various solvers, including both open-source solvers and commercial solvers, such as CLP, SCIP, GLPK, CPLEX, GUROBI and many more. Although PuLP provides high flexibility in the solvers that it can use, it comes with several limitations, which will be covered in Section 4. Based on our experience with the creation of a Python based additive rounding program, which solves a Mixed Integer Linear Programming (MILP) problem, the SCIP solver demonstrated promising results in term of speed and solution quality. Thus, the PySCIPOpt Python package that is built as an interface for the SCIP optimization suite was also included in the comparison. In the next section the performance of PuLP-CLP and PySCIPOpt-SCIP and SAS-OPTMODEL are evaluated and compared.

3.2 Performance

Example 1: 4272 cells, 176 constraints, 329 primary sensitive cells			
Solver	Time	Number of Suppressed Cells	Value Suppressed
SAS	9s	402	6082811
PySCIPOpt-SCIP	40s	402	6082811
PuLP-CLP	48s	402	6082811

Example 2: 9955 cells, 7309 constraints, 1036 primary sensitive cells			
Solver	Time	Number of Suppressed Cells	Value Suppressed
SAS	90s	4393	38259242
PySCIPOpt-SCIP	400s	5154	39650109
PuLP-CLP	518s	5161	39664502

Example 3: 29791 cells, 43245 constraints, 903 primary sensitive cells			
Solver	Time	Number of Suppressed Cells	Value Suppressed
SAS	180s	11370	72024492
PySCIPOpt-SCIP	1647s	12525	79788286
PuLP-CLP	1446s	12540	79811939

Example 4: 50625 cells, 94500 constraints, 1321 primary sensitive cells			
Solver	Time	Number of Suppressed Cells	Value Suppressed
SAS	820s	24105	1.7161E8
PySCIPOpt-SCIP	18216s	27079	1.9906E8
PuLP-CLP	3452s	27091	1.9933E8

Table 1: Runtime and Optimality comparison between four solvers in four testing scenarios

In example 1, we see that all of the solvers can solve the problem in a very short period of time and they found the same suppression pattern. As the table complexity (size and number of constraints) increases and as the number of primary sensitive cells increases, PySCIPOpt-SCIP and PuLP-CLP started to over-suppress slightly as shown in the example 2 table. As we move on to tables with thirty thousand cells, we also see that PySCIPOpt-SCIP and PuLP-CLP are over suppressing and are slower compared to SAS. For the largest table in the experiment, PySCIPOpt-SCIP took about five hours to complete, where Pulp-CLP took about one hour, and SAS took only thirteen minutes.

3.3 Findings

Although not described here, we also found that there are cases that were solved by SAS but were not solved by the open-source solvers and vice versa. However, in terms of time, SAS is relatively faster among the three and produce much better suppression patterns.

The analysis was limited to the scenarios above to ensure that we have a feasible solution for each solver. It seems clear that problems in terms of speed and optimization are observed in these

[2] https://support.sas.com/documentation/onlinedoc/or/132/optmodel.pdf

'simple' examples. For context, at Statistics Canada, it is reasonable to see tables of size 50k cells and 25k primary sensitive cells, where SAS typically takes around two hours to complete. These 'real world' examples were tested on the Python solvers, but the Python versions failed to solve the problem.

Through these experiments, it was discovered that PuLP has insufficient precision. For example, if we had a sensitivity value of one million and one, then in our constraint, it would require the positive shift for this sensitive cell to be larger or equal to one million and one. However, with PuLP, the positive shift would be just one million.

4. Challenges

There were several challenges experienced throughout the project. From the current G-Confid implementation, there were programming steps that were efficient in SAS but did not offer the same efficiency when the ideas were reimplemented in Python.

Additionally, it was difficult to spot bugs in the programs because of the heuristic algorithm. It is not clear why specific open-source solvers were over-suppressing for larger tables when they were able to find the exact same result as SAS in the simpler tables.

It was also hard to predict the outcome with different solver and solver options in terms of success, quality and time. For PySCIPOpt, it has numerous solver options that you can tune and adjust for your specific problems, which PuLP does not have. We experimented with various LP algorithms and various scaling in PySCIPOpt, but it did not have a significant effect on the run time. In general, the effect of different solver and solver options was only observed with specific experiments. As we see from the examples, solvers may produce different patterns and there is no guarantee if it is the best pattern, and it is difficult to understand the reasons why we see the difference.

Lastly, moving to Python does not seem to help to solve some of our specific problem cases. G-Confid was having numerical instability with some extreme input data. Although not shown in this paper, the python version also has trouble solving it.

5. Conclusion and Future Work

In this paper, we reimplemented the complementary cell suppression algorithm that is currently available in the SAS-based solution G-Confid. The Python programming language was studied with two open-source linear programming solvers PuLP-CLP and PySCIPOpt-SCIP. The experiments suggested that the open-source solvers were capable of solving tables with moderate complexity within reasonable amount of time. However, as the complexity increased, the open-source solvers offered less optimal solutions in longer times when compared to SAS.

In the future, we hope to experiment with commercial linear programming solvers including CPLEX, GUROBI and SCIPY to study if they offer performance similar to what was shown by Meindl (2013) and compare those solutions to our SAS solution. If necessary, we can move to different mathematical formulation of the complementary cell suppression problem. We are also considering different architecture including cloud infrastructure and hope that will allow the Python solution to achieve similar performance compared to SAS.

References

Bestuzheva, K., Besançon, M., Chen, W.-K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., … Witzig, J. (2021). *The SCIP Optimization Suite 8.0.* Retrieved from Optimization Online website: http://www.optimization-online.org/DB_HTML/2021/12/8728.html

C. M. Sullivan, *An overview of disclosure principles*, U.S. Bureau of the Census Research Report Series (1992), RR–92/09

Cox, L. H. (1981). Linear sensitivity measures in statistical disclosure control. Journal of Statistical Planning and Inference, 5(2), 153–164.

Frolova, O., Fillion, J.-M., and Tambay, J.-L., *Confid2: Statistics canada's new tabular data confidentiality software*, Proceedings of the Survey Methods Section, SSC (2009)

Meindl, Bernhard & Templ, Matthias. (2013). Analysis of Commercial and Free and Open Source Solvers for the Cell Suppression Problem. Tansaction on Data Privacy. Volume 6. 147-159.

Mitchell, S., Kean, A., Mason, A., O'Sullivan, M., Phillips, A., Peschiera, F., *Optimization with PuLP- PuLP 2.7.0 documentation* Optimization with PuLP — PuLP 2.7.0 documentation (coin-or.github.io)