tau-Argus online help for CTA package

1 Introduction

The RCTA package implements the restricted controlled tabular adjustment (RCTA) method for the protection of statistical tabular data. Details about RCTA can be found in [1, 2]. This package is used and it was motivated for the Protection of Structural Business Statistics by Eurostat [9]; it was later applied to the protection of Balance of Payment data again by Eurostat; it was finally extended for the protection of animal production statistics by Eurostat. It can be used in other applications developing ad-hoc main programs that interface with the RCTA callable library.

The current version of the RCTA package is linked with six state of the art solvers: CPLEX [12], XPRESS [6], GLPK [10], CBC [8], CLP [7] and SYMPHONY [13]. This has been done thanks to Osi [5], which provides an abstract interface to communicate with solvers.

CPLEX and XPRESS are commercial solvers and you need a license, but GLPK, CBC, CLP, and SYMPHOYNY are license free solvers. The package was tested with CPLEX release 12.5, XPRESS 2011a, GLPK 4.45, CBC 2.7.6, CLP 1.14.6 and SYMPHONY 5.4.4. Others versions may work but have to be tested, XPRESS version 2011a is the minimum version to be used.

The RCTA formulation solved in the package is as follows. Given (i) a set of cells $a_i, i = 1, \ldots, n$, that satisfy m linear relations $Aa = b$ (a being the vector of a_i's); (ii) a lower and upper bound for each cell $i = 1, \ldots, n$, respectively l_{a_i} and u_{a_i}, which are considered to be known by any attacker; (iii) a set $\mathcal{S} = \{i_1, i_2, \ldots, i_p\} \subseteq \{1, \ldots, n\}$ of indices of sensitive cells; (iv) and a lower and upper protection level for each sensitive cell $i \in \mathcal{S}$, respectively lpl_i and upl_i, such that the released values satisfy either $x_i \geq a_i + upl_i$ or $x_i \leq a_i - lpl_i$; the purpose of CTA is to find the closest safe values $x_i, i = 1, \ldots, n$, according to some distance L, that makes the released table safe. This involves the solution of the following optimization problem:

$$\begin{array}{ll} \min\limits_{x} & ||x - a||_L \\ \text{s. to} & Ax = b \\ & l_{a_i} \leq x_i \leq u_{a_i} \quad i = 1, \ldots, n \\ & x_i \leq a_i - lpl_i \text{ or } x_i \geq a_i + upl_i \quad i \in \mathcal{S}. \end{array} \tag{1}$$

If we allow $l_{a_i} = u_{a_i}$ for some subset of cells, the values of these cells are preserved. This stronger variant of CTA is named Restricted CTA (RCTA). Both names will be used indistinctly in this document. Problem (1) can also be formulated in terms of deviations from the current cell values. Defining $z_i = x_i - a_i, \quad i = 1, \ldots, n$ —and similarly $l_{z_i} = l_{x_i} - a_i$ and $u_{z_i} = u_{x_i} - a_i$—, (1) can be recast as:

$$\begin{array}{ll} \min\limits_{z} & ||z||_L \\ \text{s. to} & Az = 0 \\ & l_{z_i} \leq z_i \leq u_{z_i} \quad i = 1, \ldots, n \\ & z_i \leq -lpl_i \text{ or } z_i \geq upl_i \quad i \in \mathcal{S}, \end{array} \tag{2}$$

$z \in \mathbb{R}^n$ being the vector of deviations. The CTA package implements the L_1 distance. Using this distance, after some manipulation, (2) can be written as

$$\begin{array}{ll} \min\limits_{z^+,z^-,y} & \sum\limits_{i=1}^{n} w_i(z_i^+ + z_i^-) \\ \text{s. to} & A(z^+ - z^-) = 0 \\ & 0 \le z_i^+ \le \quad u_{z_i} \quad i = 1,\ldots,n \\ & 0 \le z_i^- \le -l_{z_i} \quad i = 1,\ldots,n \\ & upl_i\ y_i \le z_i^+ \le u_{z_i}\ y_i \quad i \in \mathcal{S} \\ & lpl_i(1-y_i) \le z_i^- \le -l_{z_i}(1-y_i) \quad i \in \mathcal{S}, \end{array} \tag{3}$$

$w \in \mathbb{R}^n$ being the vector of cell weights, $z^+ \in \mathbb{R}^n$ and $z^- \in \mathbb{R}^n$ the vector of positive and negative deviations in absolute value, and $y \in \mathbb{R}^p$ being the vector of binary variables associated to protections directions. When $y_i = 1$ the constraints mean $upl_i \le z_i^+ \le u_{z_i}$ and $z_i^- = 0$, thus the protection direction is "upper"; when $y_i = 0$ we get $z_i^+ = 0$ and $lpl_i \le z_i^- \le -l_{z_i}$, thus protection direction is "lower". Model (3) is a (difficult) mixed integer linear problem (MILP).

The package allows both the optimal and a heuristic solution of (3). The heuristic solution is computed by a Block Coordinate Descent (BCD) approach described in Section 3. The package also implements a continuous CTA model without binary variables for very fast—although of less quality—solutions. This approach formulates a multiobjective optimization approach solved by a "Lexmin" method. This variant is named Lexmin approach in this document, and it is described in Section 4. The full set of options of the CTA package is reported in Section 5. If you have a tough instance, see the guidelines for difficult instances of Section 6. Additional package features are explained in Section 7.

More details can be found in the User's Manual of the RCTA package [4].

2 Graphical user interface

A GUI is provided with the package, which works in both Linux and Windows systems. The GUI is based on the wxWidgets library [14]. The GUI is shown in the screenshot of Figure 1.

In Windows, the library wxWidgets is self-contained in the provided CTA package and there is no need to install wxWidgets to run the GUI. However, to compile the CTA package wxWidgets must be installed.

In Linux, wxWidgets must be installed because shared libraries are used. A "HOW_TO_INSTALL" file is provided to install wxWidgets in both Windows and Linux systems. The minimum wxWidgets version required is 2.9.4.

To execute the GUI, both GUI and main_CTA executables must be in the same directory.

With the GUI you have the same options as with the command line main_CTA code, but in a more friendly way.

In the left side are the options to solve the problem, these options dynamically change in function of the type of problem and the chosen model. In the right side is the output of CTA application, it is the same output that the command line application. When the mouse is over an option in the bottom appears a short description.

When the application request some data, more time for example, the input box must be used. When the execution finish appear three buttons to clear the output, show the log file and show the solution file.

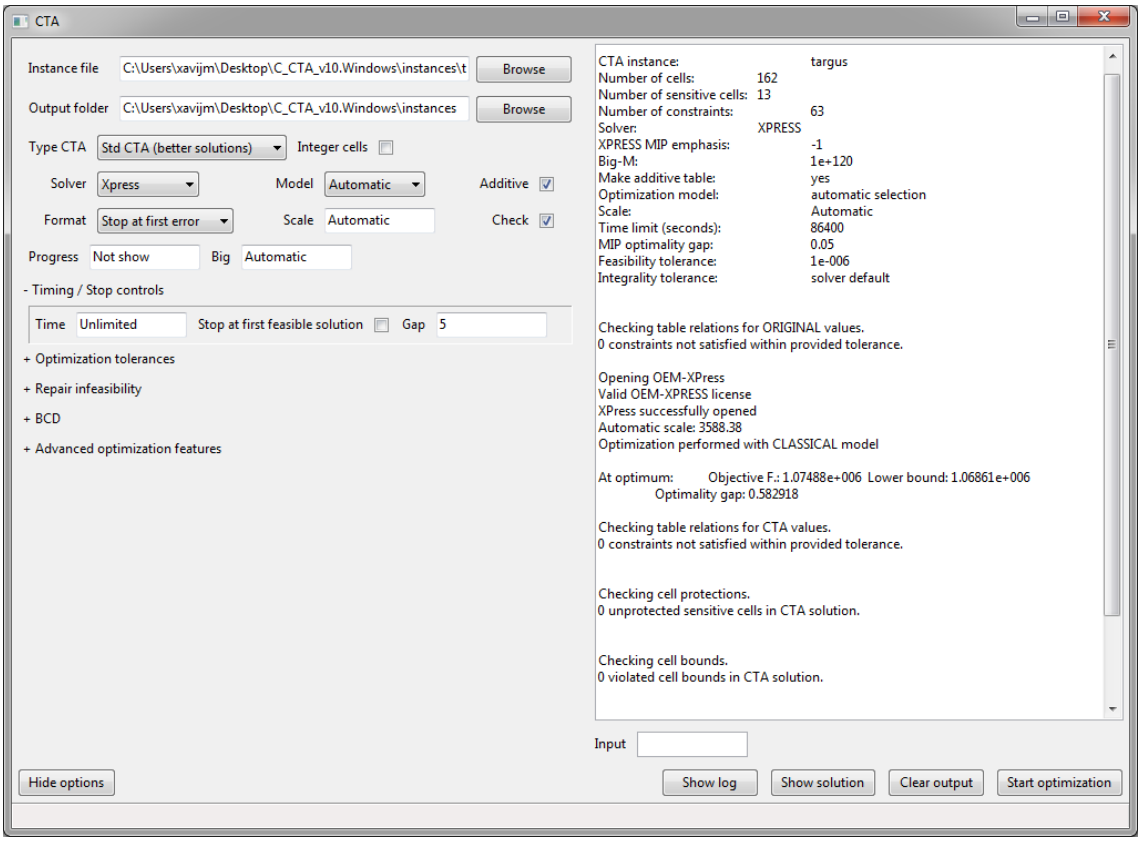

Figure 1: Screenshot with the GUI of the CTA package

3 Block Coordinate Descent (BCD) heuristic

The BCD approach applied to CTA was introduced in [11]. Briefly, it consists of a sequence of CTA subproblems, each of them optimizing the objective function over the cell deviations and a subset of the decision variables y, while the remaining variables are kept fixed to some direction. Provided that we start from a feasible y, the method can move from a solution to another, hopefully better. Although there could be uncountable strategies to determine the subset of variables to be optimized, typically the $\mathcal{S}$ set is partitioned into K clusters (or blocks) and the algorithm iterates through them. However, BCD could perform indefinitely, starting again with the same or with another partition. Stopping criteria normally employed are: only one cycle of K clusters; a time limit, or a specified number of subproblems without improvement in the objective function. Since the method does not account for dual information there are no means to compute a gap for the solution. Despite this, the experience suggests that BCD reaches sub-optimal but still good solutions in significantly lower time than plain branch-and-cut schemes.

3.1 Basic options of BCD

Experience with BCD has shown that, in general, the performance of the method improves as the number of blocks decreases, and two blocks seems to be the best choice. You can select the number of blocks in the standalone program through the `-bcd-clusters` option or, shorter, `-l` (beware: lowercase 'L' character). This option (as well as next ones) can also be provided through the GUI. For instance:

```
-l 2
```

Higher numbers lead to simpler problems, but it has been observed that, when K is relatively large (say, more than ten; which means that less than 10% of the sensitive cells are optimized), the sequence of subproblems does not decrease quickly because of the difficulty to improve the solution through few sensitive cells, poorly related between them. Therefore, a large number of clusters can help to find a fast, feasible protected table at the expense of a reduction of solution quality.

Besides of the general time limit, `-time` or `-t`, option, BCD includes another time limit, `-bcd-time` or `-T`, to indicate a limit for each specific subproblem. So, if you want to start a process which should not last more than ten minutes, and each subproblem should not last more than one minute, you may select the combination:

```
-l 2 -t 600 -T 60
```

Notice that, in order to save some space, the example has replaced the long path of the output directory by a specific case, '.' which means *the actual directory*. This is irrelevant to the purpose of the example, the use of the time limit options.

Normally, BCD creates a partition of the sensitive cells set, $\mathcal{S}$, according to the parameter K, so that there is the same number of cells in each cluster (if $|\mathcal{S}|$ is not divisible by K, the last cluster is a bit smaller). The allocation in the clusters is at random, since we have not yet found another procedure showing better performance. However, this matter is still under research. A cycle is composed by the sequence of the K subproblems. Many tests indicate that rebuilding the partition of blocks at the end of the cycle is clearly preferable to keep the original division. The user can control this behavior through the `-bcd-cycle` or `-I` option. The default choice is 'c', from "*change* the partition"; other choices are 'o', only "*one* cycle and stop", and 'r', "*repeat* the first partition" at every cycle. An example:

```
-l 3 -t 60 -I o
```

In this case, the instance will be solved using just one cycle of three subproblems and stopping, or sixty seconds, whichever comes first.

3.2 Less frequent options of BCD

Another two stopping rules the user can activate: it is not unusual to deal with an instance which cannot improve along several subproblems. The optimal solution of a subproblem is the same than the previous one, even if the set of sensitive cells to determine optimal directions has changed (some directions could have changed, but the objective function remains the same). The program includes a protective option to avoid iterating without improvement: `-bcd-nstop` or `-N`. If the number in the option is less than or equal than 1, the program will stop after $10K$ consecutive subproblems failing to improve the solution; otherwise, the number given indicates the user's preference.

The second rule is `-bcd-objf` or `-F`: the attached value is used as a threshold to be compared with the objective function value at each subproblem, and the execution is stopped when a subproblem reaches the threshold. Unless the user has a good knowledge about the instance to be solved, it is not advisable to use this option alone, without any other stopping rule, since the program could run for a long time (the default for `-time` is 24 hours). For instance, the options

```
-l 3 -t 60 -F 1e4 -N 8
```

would be for an execution programmed with three clusters, and to stop after sixty seconds, if it finds a solution with objective function less than 10,000, or if a sequence of 8 consecutive subproblems is not able to decrease the objective function, whichever comes first.

The user has an option to use BCD as a strategy to find a good feasible starting point for the plain branch-and-cut scheme. The option `-bcd-endbc` or `-B` together with a number B means that B seconds of time have to be reserved from the total execution time to solve the problem with only one cluster, that is, through the standard CTA formulation. For instance: you may want to give as much as 30 minutes to solve a problem, but the first 10 minutes are devoted to iterate with BCD to set an initial solution, and the remainder will be used to deal with the problem as a whole through B&C. Then, you may consider options:

```
-l 4 -t 1800 -B 1200
```

Finally, there is an option to choose the seed of the random number generator. It is actually fixed to a number, 21071969, so the program is able to reproduce exactly the same result if you repeat exactly the same input data and parameters (and, possibly, the same computer architecture). If, for whatever reason, you want to change the seed and use another number, or even you don't need to worry about this matter and prefer more real randomness in your tests, there is `-bcd-seed` or `-S`. This option with 0 or a negative number puts a random seed (based in the actual system-time); elsewhere uses the argument as the new seed. For instance:

```
-l 2 -S 783457
```

3.3 Dynamic BCD

The current behavior of BCD has been described above, and it is based on some passive partition of the $\mathcal{S}$ set, in the sense that the clusters composition is not affected by the results obtained in

the previous subproblems. The procedure implemented in the program, called "dynamic BCD", is an attempt to integrate some knowledge in the decision process. Basically, it is based on the premise that a large deviation in some sensitive cell may be corrected with a change of direction or, at least, including the cell as a candidate to be changed, which only can be made if it is in the cluster. So the method does not partition the cells into K different clusters, giving each cell one and only one chance to decide the protection direction at every cycle. Instead, the algorithm tries to choose the most promising cells that could decrease the objective function: so, there is no *partition* concept now, since some cells can be optimized in two or more different clusters per cycle.

Specifically, the dynamic procedure considers two parameters:

- Z_D, percentage of sensitive cells to be included in the cluster. It can be set with the option `-dyn-sprop` or `-Z`. By default, it takes the value 50%.
- Y_D, probability to discard a sensitive cell to be included in the cluster. It can be set with the option `-dyn-perturb` or `-y`. By default, it takes the value 50%.

Moreover, the preceding options are taken into account only if the user set the option `-dynamic` or `-L` with the choice 'y'. Other options incompatible with the dynamic method (as `-bcd-cycle`) would be ignored.

The method works as follows. The first time, the cells are chosen at random; for instance, if $Z_D = 50\%$, half of the sensitive cells will be taken without further consideration. Once the subproblem has found a solution, the array of terms related to sensitive cells appearing in the objective function in problem (3) is sorted in decreasing order:

$$\left\{ w_{(1)}(z^+_{(1)} + z^-_{(1)}), w_{(2)}(z^+_{(2)} + z^-_{(2)}), \ldots, w_{(|\mathcal{S}|)}(z^+_{(|\mathcal{S}|)} + z^-_{(|\mathcal{S}|)}) \right\}$$

That is: $w_{(1)}(z^+_{(1)} + z^-_{(1)})$ is the largest term in the objective function, related to the $i_{(1)}$-th sensitive cell, and so on. The procedure goes down the array, choosing a cell with probability Y_D, until the cluster is full (the size $Z_D|\mathcal{S}|$ is reached) or the array is finished. Next, the subproblem with this cluster is solved, and the process restarts again if the time has not been exhausted yet. For instance, with the options:

```
-t 300 -L y -Z 33 -y 25
```

the instance would be solved in at most five minutes with the dynamic method, where each cluster will take one third of the sensitive cells, and the cells are included in the cluster with probability 0.75: so not all the cells in the first 33% of the list will be considered, one out of four would drop from the cluster in benefit of other cells down in the array. It is advisable to introduce some uncertainty in the procedure, otherwise the program might fall into a loop whenever a subproblem cannot improve the previous solution, but the algorithm is forced to reuse again the same cluster of sensitive cells. For this reason, avoid probabilities Y_D close to 0.

3.4 Starting point

A disadvantage of BCD is the need to find a feasible assignment of directions for the sensitive cells in order to start the process. The SAT strategy [11] (`-startingpoint` or `-k` with the choice 's') is useful to find both a proposal of starting point, or a set of valid inequalities which can tight better

the feasible region. Alternatively, the following model has the same set of feasible solutions as the model (3) but halving the number of continuous variables z, which are not positive anymore:

$$\begin{array}{lll} \min\limits_{z,y} & 0 & \\ \text{s. to} & Az = 0 & \\ & l_{z_i} \leq z_i \leq u_{z_i} & i = 1, \ldots, n \\ & z_i \leq -lpl_i(1 - y_i) + u_{z_i} y_i & i \in \mathcal{S} \\ & z_i \geq upl_i y_i + l_{z_i}(1 - y_i) & i \in \mathcal{S} \\ & y_i \in \{0, 1\}, & i \in \mathcal{S} \end{array} \tag{4}$$

The model (4), or "Compact" model (`-startingpoint` or `-k` with the choice 'c'), has an irrelevant objective function equal to zero because it is not optimized at all: the model is used just to find a feasible solution, and usually it finds one very quickly, then stops and allows to set an initial starting point for BCD.

At last, it is always possible to read a list of values from a file for every sensitive cells, assigning a direction for each one (`-startingpointf` or `-K` followed by the name of the file).

4 Lexmin approach

The difficulty of the CTA problem (3) are the binary variables y associated to the direction of protection of sensitive cells. If we a priori fix the values of y, then we have an easy continuous linear optimization problem. This is controlled by the option `-fixdir` or `-X`: a value of 'n' means using the standard CTA model (3), while another value means using the continuous CTA variant. This variant is named "Fast CTA (better timing)" in the GUI of CTA. Full details can be found in [3].

4.1 How it works?

Pre-fixing the values of y the resulting LP may be infeasible. Feasibility can be recovered by three means:

- Allowing changes in the table relations; α^+, α^- are the vectors of positive and negative changes to these relations.
- Increasing the cell lower and upper bounds; β_l and β_u are the vectors of increments to lower and upper bounds.
- Reducing the lower and upper protections of sensitive cells; γ_l and γ_u are the vectors of reductions to lower and upper protection levels.

Our objective is now to minimize

$$\min(f_1(z), f_2(\alpha^+, \alpha^-), f_3(\beta_l, \beta_u), f_4(\gamma_l, \gamma_u))$$

where

- $f_1(z) = ||z||_1$ is the ℓ_1 norm of the vector of cell deviations of (3).
- $f_2(\alpha^+, \alpha^-) = ||(\alpha^+, \alpha^-)||_1$ is the ℓ_1 norm of the vector of changes in table relations.

- $f_3(\beta_l, \beta_u) = ||(\beta_l, \beta_u)||_1$ is the ℓ_1 norm of the vector of increments of cell bounds.
- $f_4(\gamma_l, \gamma_u) = ||(\gamma_l, \gamma_u)||_1$ is the ℓ_1 norm of the vector of decrements of cell protections.

This results in a multiobjective optimization problem, which is solved by a *lexmin* approach. The lexmin approach considers a preference order for the four different objectives: i_1, i_2, i_3, i_4. Then it first optimizes for the first objective in this order f_{i_1}, obtaining a solution of cost $f^*_{i_1}$. Next, it optimizes for the second objective f_{i_2} setting as additional constraint that $f_{i_1} = f^*_{i_i}$, that is, it looks for the best solution for f_{i_2} among those which are optimal for f_{i_1}. And so on, until the four objectives have been optimized. The preference order can be provided by the user with the option `--order` or `-O`; for instance we can set `--order 4-3-2-1`. The default order is 4-2-3-1.

4.2 Controlling the lexmin approach

The following options allow the user to control the lexmin approach:

- By default, to improve the quality of the solution, the maximum cell deviations allowed in the protected cell values is a 2% of the cell value. This value may be changed with the `-pct-bounds` or `-P` option. Setting a value of -1 means using the original table bounds.
- The extra constraints $f_{i_j} = f^*_{i_j}$ to be added at each subproblem may cause the problem to be reported as infeasible due to numerical errors. These constraints are thus replaced in practice by $f_{i_j} \leq \max(f^*_{i_j}(1 + \epsilon_f), \epsilon_{rhs})$. The tolerances ϵ_f and ϵ_{rhs} and can be set with the options `-epsf` and `-epsrhs`; their default values are, respectively, 10^{-4} and 0.
- Since the subproblems are linear optimization problems, we can choose an interior-point, a primal simplex, or a dual simplex algorithm. The algorithm can be selected with the option `-method`, providing the values 'b' (barrier), 'd' (dual simplex), 'p' (primal simplex), or 'a' (automatic). The default is 'a'. In general, for large tables the interior-point barrier algorithm is preferred.
- If the interior-point algorithm has been selected, then we can obtain a non vertex solution (which may contain a larger number of non-zero cell deviations). To move from that solution to a vertex solution (with a likely number of zero cell deviations), the `-crossover` or `Q` option has to be activated (by default is active).
- The objectives $f_2(\alpha^+, \alpha^-)$, $f_3(\beta_l, \beta_u)$ and $f_4(\gamma_l, \gamma_u)$ are the ℓ_1 norm of the corresponding vectors, weighting every component v_i of the vector v by $1/v_i^w$. The exponent w is by default 1/2. It can be changed with the option `-expobj` or `-J J`.

4.3 Starting point

A good assignment of the binary variables is instrumental both for the feasibility and the quality of the solution provided by the lexmin approach. This assignment is controlled by the option `-fixdir` or `-X`, which can take the following values:

- 'n': no assignment is performed, thus the lexmin approach is not used, instead the standard CTA problem is solved.
- 'r': the value of the binary variable y_i is set to 0 or 1 randomly. The seed of the random generator can be set by the option `-seed` or `-S`.

- 'f': the assignment is read from an external file. The file is provided by the option `-fixdirfn` or `-H`.
- 's': the SAT heuristic (which is also available for BCD, see Subsection 3.4) is applied (see [11] for a description).
- 't': a *network heuristic* based on a network representation of the interrelations between sensitive cells and constraints is applied (see [3] for a description).
- 'b': both SAT and the network heuristic are applied.

It has been observed that, in general, the best results are obtained with the SAT or network heuristics, or both together; the random assignment usually provides protected tables of minor quality.

5 Full set of options of the CTA package

The general options of the package, available for the different methods, are:

```
options
        --solver, -s {s}
                s = 'c' (Cplex) or 'x' (Xpress) or 'g' (Glpk) or 'b' (Cbc)
                or 's' (Symphony) or 'l' (Clp), default 'x'. Clp only can
                be used with fixdir option. Cbc and Symphony cannot be used
                with fixdir option
        --time, -t {t}
                t = initial limit time in seconds for optimization,
                default 86400
        --check, -c {c}
                check input table and solution c = 'n' (no) or 'y' (yes)
                or 'a' (all), default 'y'
        --additive, -a {a}
                make table additive if not originally a = 'n' (no) or
                'y' (yes), default 'y'
        --format, -z {z}
                check all the format errors or stop at first, z = 'a' (all)
                or 'f' (first), default 'f'
        --scale, -C {C}
                scale (scale factor) is optional, -1 automatic scale,
                default -1. If option intdev is used scale is always 1
```

The options related to the mixed integer problem (MIP) formulation of CTA (the one with binary variables) are:

```
        --first, -f {f}
                stop at first feasible solution, f = 'n' (no) or
                'y' (yes), default 'n'
        --mipgap, -g {g}
```

```
        g = optimality % gap required, g <= 1.0e-3%, default 5%;
        increase it if execution too slow
--integrality, -i {i}
        i = integrality tolerance, 1 >= i >= 0, default solver
        default; i >= e in XPRESS
--feasibility, -e {e}
        e = feasibility tolerance, e >= 1.0e-9, default 1.0e-6
--big, -b {b}
        b = big value to be used, at most, for bounds on
        deviations, default Infinity;
        b = -1 -> automatically set by the code; if problems, set
        a decent big value as 1.0e+8
--emphasis, -h {h}
        emphasis for XPRESS h = -1,0,1,2,3 -> quality 0 -- speed 3,
        default -1
--mipemphasis, -m {m}
        mipemphasis for CPLEX m = 0,1,2,3,4, default 0 -> balanced
--varsel, -v {v}
        variable selection criteria in CPLEX, v = -1,0,1,2,3,4,
        default 0
--model, -o {o}
        CTA optimization model to be used o = 'n' (new),
        'c' (classical), 'a' (automatic selection), default 'a'
--repair, -r {r}
        apply repair infeasibility procedure r = 'n' (no) or
        'y' (yes), default 'n'
--repairfn, -x {x}
        x = file name with information for repair infeasibility
        tool, if repair option used
--vineq, -j {j}
        Valid inequalities, j = 'y' (yes) or 'n' (no) or
        'c' (check), default 'n'
--startingpoint, -k {k}
        starting Point, k = 'c' (compact) or 's' (sat) or
        'n' (no), default 'n'
--startingpointf, -K {K}
        if startingpoint option is 'f', K must be the file with all
        the cell values. For each line the file must contains the
        cell index and the value
--progress, -d {d}
        show progress of optimization procedure on screen at most
        every d seconds, default 86400 (only for Xpress and Cplex)
--intdev, -D {D}
        force output cells to be integer, D = 'y' (yes) or 'n' (no),
        default 'n'
--intdevtol, -E {E}
        integrality tolerance to round the output cells in the
        solution file when intdev is used, default 0
```

The options related to the linear problem (LP) formulation of CTA (the one without binary

variables) are:

```
--lptol, -G {G}
        G = lp optimality tolerance required, if G = -1 uses
        solver default, default 1e-8; increase it if execution
        too slow
--feasibility, -e {e}
        e = feasibility tolerance, e >= 1.0e-9, default 1.0e-6
--intdev, -D {D}
        force output cells to be integer, D = 'y' (yes) or
        'n' (no), default 'n'
--intdevtol, -E {E}
        integrality tolerance to round the output cells in the
        solution file when intdev is used, default 0
--seed, -S {S}
        seed to use in random methods, S <= 0 (random seed) or
        S > 0 (seed to be used), default 21071969
--fixdir, -X {X}
        fix the direction of the cells perturbations. X = 'n' (no)
        or 'r' (random) or 'f' (file) or 's' (sat) or 't' (net) or
        'b' (both sat and net), default 'n'
--fixdirfn, -H {H}
        when fixdir option is 'f' (file), H = name of directions
        file, for each fixedcell a line with the cell index and
        the direction (0, lower or 1, upper)
--order, -O {O1-O2-O3-O4}
        when fixdir option is used, selects the order of the
        objectives functions for the lexmin approach. Oi = {1..4}.
        By default the order is 4-2-3-1. See the manual for more
        information about each objective function
--pct-bounds, -P {P}
        when fixdir option is used, the bounds of cells will be
        P% above and below the cells values, instead of the
        table bounds. If P = -1 uses the table bounds. Default P = 2
--crossover, -Q {Q}
        when fixdir option is used, crossover after barrier method,
        Q = 'y' (yes) or 'n' (no), default 'y'
--method, -M {M}
        when fixdir option is used, select the method to solve the
        problem, M = 'b' (barrier method) or 's' (simplex method),
        default 'b'
--expobj, -J {J}
        when fixdir option is used, J is the exponent for objective
        2, 3 and 4, J = 0 or 1 or 0.5 or -1 (logarithm is used),
        default 1
```

The options related to the block coordinate heuristic (BCD) for the MIP-CTA problem are:

```
--bcd-clusters, -l {l}
        number of clusters for Coordinated Descent, l = file name
```

```
        or number, '1' (no BCD), file name cannot start with a number
--bcd-time, -T {T}
        if BCD used, maximum time in seconds that can use each BCD
        cluster (default 86400)
--bcd-objf, -F {F}
        if BCD used, stop if objective function is lower than F
        (default -Infinity)
--bcd-endbc, -B {B}
        if BCD used, when BCD ends do a global B&C, B = maximum
        seconds of execution, (default 0, no B&C)
--bcd-cycle, -I {I}
        BCD cycles, I = 'o' (one cycle) or 'r' (repeat the same
        cycle indefinitely) or 'c' (repeat the cycle changing
        the clusters), default 'r'
--bcd-nstop, -N {N}
        number of consecutive subproblem executions before stopping
        if the objective value has not improved, N <= 1 (adaptive
        value, number of clusters * 10) or N >= 2 (value to be used),
        default adaptive
--dynamic, -L {L}
        dynamic bcd, L = 'y' (yes) or 'n' (no), default 'n'
--dyn-sprop, -Z {Z}
        if dynamic option used, % sens vars to consider in dynamic
        bcd, Z in {0..100}, default 50
--dyn-perturb, -y {y}
        if dynamic option used, % probability discard sens var to
        be included in dyn bcd, y in {0..100}, default 50
--seed, -S {S}
        seed to use in random methods, S <= 0 (random seed) or
        S > 0 (seed to be used), default 21071969
```

The adavanced options for the expert user are:

```
--prepsens, -p {p}
        preprocess sensitive cells, p = 'n' (no) or 'y' (yes),
        default 'n'
--write, -w {w}
        write problem in lp file, w = 'y' (yes) or 'n' (no),
        default 'n'
--epsf {e}
        when lexmin model is used, epsF is used to calculate
        the rhs of the new restrictions where the rhs is
        max(objF*(1+epsF), epsrhs), default 1.0e-4
--epsrhs {e}
        when lexmin model is used, epsrhs is used to calculate
        the rhs of the new restrictions where the rhs is
        max(objF*(1+epsF), epsrhs), default 0
```

A short explanation of the main different options/parameters follows:

solver: Solver to be used between CPLEX, XPRESS, GLPK, CBC, CLP and SYMPHONY. CLP can only be used with fixdir option. CBC and SYMPHONY cannot be used with fixdir option.

time: CPU time limit in seconds. The optimization will be stopped once this limit has been reached, and the package will ask for more CPU time (if 0 is entered, it will definitely stop). In commercials solvers this time is quite accurate but the behaviour in free license solvers is not always correct.

check: If this parameter is "y" some simple checks about the input table and the solution obtained is performed and reported on the screen. These checks include feasibility of linear table relations, protection of sensible cells, lower and upper bounds of adjusted table values, and quality of internal optimization model variables (i.e., that no both the positive and negative variables z_i^+ and z_i^- of cell i are positive in the solution of the mathematical programming model (3)).

additive: This parameter is described in Subsection 7.1.

format: When reading the input file, stop at first format error or check all file and list errors.

scale: Scale factor to divide the values of the input table. When automatic is selected, scale is calculated using $\exp\left(\sum_{b \in Bounds} \frac{ln(|b|)}{2n}\right)$. This parameter is described in Subsection 6.

first: If yes, the package will stop once the first feasible solution has been found, and it will ask for more CPU time (if 0 is entered, it will definitely stop).

mipgap: Optimality gap measures the quality of the solution as a relative distance from the current solution to a known lower bound of the optimal solution. Setting g=0% asks for the real optimal solution, but it may be very expensive. Increasing g from the default 5% (to, e.g., 50%) helps in producing a feasible sub-optimal solution quickly.

integrality: Integrality tolerance, i.e., the amount by which the binary variables in the RCTA model can be different from 0 or 1, and still be considered 0 or 1. The CPLEX default is 1.0e−5; the XPRESS default is 5.0e−6. In CPLEX it must be a value greater or equal than 0; in XPRESS it must be greater or equal than the feasibility tolerance. Due to this non-zero integrality tolerance and the bad scaling of RCTA (because of the presence of very large and small values in a table), the solution provided by the solver may violate the protection levels of some cells. In this case it may help to decrease this integrality tolerance (e.g., 1.0e−10). However, this may significantly increase the solution time. Moreover, in XPRESS you are forced to decrease the feasibility tolerance too, and then the solver may falsely conclude the problem is infeasible. Indeed the above feasibility and this integrality tolerances may need to be fine tuned for particular tables. No unique set of values were able to solve all the tables tested; the default values in main_CTA are just reasonable ones. See Subsection 6 for guidelines for solving difficult instances.

feasibility: Feasibility tolerance, i.e., the degree in constraints/bounds violations allowed by the optimization procedure. In CPLEX it must be greater or equal than 1.0e−9; in XPRESS it must be greater or equal than 0. If it is too tight (e.g., 1.0e−9) the solver may falsely conclude the problem is infeasible. By default 1.0e−6 is used. If the problem is reported as infeasible, then you may try to increase it a bit (e.g., 1.0e−5, or 5.0e−5). However, this may affect the quality of the solution: the solver may finish at a solution reported as optimal, that may lead to underprotection of some cells (see Subsection 6 for details).

big: Big value for bounds on allowed (either positive or negative) deviations from current original cell values. The default huge value of 1.0e+120 ($\approx \infty$) guarantees that the bounds given by the user in the input file will be used). This may cause problems with feasibility and integrality tolerances (see comments on these parameters). Tightening the bounds in the input file is a good practice to avoid numerical problems in the solver. Otherwise, a smaller "b" big value may be given (e.g., b=1.0e+5 would be fine). However, be aware that if "b" is set to a too small value, then the problem may become infeasible.

emphasis: XPRESS MIP heurdivespeedup parameter (similar to CPLEX mipemphasis). It controls the tradeoff between solution quality and diving speed in the MILP algorithm. The meaning is:

h=−1: Automatic selection.

m= 0,1,2,3: Emphasis bias from emphasis on quality (0) to speed (3).

The default value in main_CTA is h=−1. If the problem is wrongly reported as infeasible, h=0 may be tried. If the solution time is too large, h= 3 may be tried.

mipemphasis: CPLEX MIP emphasis parameter (similar to XPRESS heurdivespeedup). It controls the tradeoff between speed, feasibility, and optimality in the MILP algorithm. The meaning is :

m=0: Balance optimality and feasibility.

m=1: Emphasize feasibility over optimality.

m=2: Emphasize optimality over feasibility.

m=3: Emphasize moving best bound.

m=4: Emphasize finding hidden feasible solutions.

The default value in main_CTA is m=0. If the problem is wrongly reported as infeasible, m=1 may be tried. If the solution time is too large, m= 2 may be tried.

varsel: Variable selection parameter of CPLEX MIP optimization. See CPLEX user manual for details.

model: CTA optimization model to be used, described in Subsection 7.3.

repair: Apply repair infeasibility procedure, described in Subsection 7.4.

repairfn: File name with information for repair infeasibility tool.

vineq: Add valid inequalities to the model in order to reduce the solution space.

startingpoint: This parameter is described in Subsection 3.4.

process: Show progress of optimization procedure on screen at most every indicated seconds (only for XPRESS and CPLEX).

intdev: Force output cells to be integer.

intdevtol: Integrality tolerance to round the output cells in the solution file when intdev is used.

lptol: LP optimality tolerance required. If the execution is too slow increase it.

seed: Seed to use in random methods.

fixdir: Fix the direction of the cells perturbations. A LP model is used instead of a MIP model. This decrease the execution time significantly in big problems but the solution can be of less quality. There are several methods to fix the cells directions: random, using a satisfiability procedure to fix a subset of cells and the rest randomly, defined by the user in a file. This parameter is described in Section 4.

fixdirfn: Name of directions file when fixdir option has file mode selected. The file format has a line for each fixed cell that contains the direction (0, lower or 1, upper).

order: For the fixdir option a lexmin model is used to solve the multiobjective problem. Four problems are solved, each one with a different objective function. This parameter selects the order they will be solved. The format to provide the order is a list of "-"-separated numbers (e.g., 4-2-3-1).

pct-bounds: When fixdir option is used, the cells bounds may be a certain percentage above and below the cells values, instead of the table bounds. If the value of this option is positive, this percentage will be used; if the value is -1, the table bounds will be used.

method: When fixdir option is used without intdev option (that means that the problem is LP) two different methods can be used to solve the problem, simplex and barrier.

crossover: When barrier method is used a crossover can be done after solving the problem.

expobj: When fixdir option is used, this option selects the exponent for objectives 2, 3 and 4. This parameter is useful when the table is not well scaled and has very big and little values. By default is 0.5.

bcd-clusters: Number of clusters for BCD. This parameter is described in Section 3.

bcd-time: Maximum time in seconds that can be spent in each BCD cluster.

bcd-objf: For BCD, stop if objective function is lower than this value.

bcd-endbc: For BCD, when BCD ends performs a global B&C.

bcd-cycle: For BCD, you can select only to run a single cycle (one execution of each cluster), repeat the same cycle indefinitely or repeat the cycle changing the clusters.

bcd-nstop: Number of consecutive subproblem executions before stopping if the objective value has not improved.

dynamic: When BCD is used, dynamic parameter equal to 'y' activates dynamic allocation of sensitive cells in the cluster to be used at each algorithm iteration. Briefly, the algorithm chooses for each subproblem which are the sensitive cells to be considered, through binary variables, according to the results from the previous subproblem. Specifically, cells taken are the ones with the largest influence in the objective function, with regard to the value of parameters that follow.

dyn-sprop: When dynamic BCD is used, selects the percentage of sensitive cells to consider in dynamic BCD.

dyn-perturb: When dynamic BCD is used, selects the probability of inclusion for a sensitive cell in the active cluster (until this reaches the desired size).

prepsens: When this preprocessing is active, any sensible cell with a zero lower protection level and a positive upper protection level will be automatically considered as "cell to be protected upwards" (since, otherwise, the original value would be safe since the lower protection level is zero). Similarly, any sensible cell with a zero upper protection level and a positive lower protection level will be automatically considered as "cell to be protected downwards".

write: Write the generated problem in a lp file.

epsF: For fixdir option, epsF is used to calculate the rhs of the new restrictions where the rhs is $\max(objF * (1 + epsF), epsrhs)$

epsrhs: When fixdir option is used, epsrhs is used to calculate the rhs of the new restrictions where the rhs is $\max(objF * (1 + epsF), epsrhs)$

When solving an instance, `main_CTA` provides three types of output.

- Output on screen, with minimum information about the instance features, and checks performed (it this option was not deactivated by the user).
- A file named `instance_solver.log`, where `instance` is the instance file and `solver` is either `cplex`, `xpress`, `glpk`, `cbc` or `sym`, generated by the solver with a summary of the optimization procedure. In a long run, this file may be used to check the progress of the branch-and-cut algorithm. The output depends on the solver—and the version of the solver—used; but in general, the three main values to be checked are: the current best solution, the best lower bound, and the optimality gap (as a percentage). The optimality gap is defined as

$$gap = \frac{best - lb}{1 + |best|} \cdot 100\%,$$

  best being the best current solution, and *lb* the best current lower bound. Note: With the version of CBC and SYMPHONY used it is not possible to get neither the lower bound nor to compute the gap. Nor exist them when the problem is LP.
- A file named `instance_solver.sol`, where `instance` is the instance file, and `solver` is either `cplex`, `xpress`, `glpk`, `cbc`, `clp` or `sym`, with the CTA solution table (if the optimization procedure finished successfully). The format of this file is: one line for each cell, providing 4 values i, a_i, x_i and p_i; i is the cell number, a_i the original cell value, x_i the CTA cell value, and p_i is 1 is this cell is sensible, and 0 otherwise.

6 Guidelines for difficult CTA instances

Several package options allow the user to control the solution of the mathematical programming model of CTA of (3). These options were listed in Section 5. Unfortunately, CTA is a difficult problem and no set of default options is valid for any CTA instance. This applies to all used solvers. The main parameters to be adjusted, if difficulties appear in the solution of some instance, are the following:

- **Large and difficult unsolvable problems**. If you deal with a large and difficult problem, which requires a lot of CPU time for its solution, you have two options. The first is to consider the Block Coordinate Descent (BCD) heuristic [11], which is described in Section

3. This heuristic suboptimally solves the CTA problem. It decomposes the problem in a set of simpler subproblems, named clusters (details in Section 3), and it may report a decent solution within the time limit. The number of clusters or subproblems to be considered depends on the particular instance. The more clusters, the faster the solution time, but at the expense of reducing the solution quality. A number of 2, 3 or 4 clusters can be a good choice in general. One cluster means solving the standard CTA problem.

The second option is to consider the variant of CTA where binary decisions are pre-fixed. In this case CTA becomes a continuous linear programming problem. Since it may be infeasible, with this variant we allow for changes in the input data: bounds, protection levels and table additivity constraints. This results in three different objectives, which together with the original one (cell deviations) amounts to four opposite objectives. The resulting multiobjective optimization problem is solved by assigning priorities to the four objectives, and applying a lexmin optimization approach. This variant may solve very large problems in few seconds, though the quality of the resulting table may be lower. More details can be found in Section 4.

- **Feasibility tolerance.** This is the degree in constraints/bounds violations allowed by the optimization procedure. In CPLEX it must be greater or equal than 1.0e−9; in XPRESS it must be greater or equal than 0. If it is too tight (e.g., 1.0e−9) the solver may falsely conclude the problem is infeasible. By default 1.0e−6 is used. If the problem is reported as infeasible, and you believe it is feasible, then try to increase the feasibility tolerance a bit (e.g., to 1.0e−5, or 5.0e−5). However, this may affect the quality of the solution: the solver may finish at a solution reported as optimal, that may lead to underprotection of some cells. The explanation is the following: Model (3) includes constraints

$$0 \leq z_i^+ \leq u_{z_i}\ y_i \qquad 0 \leq z_i^- \leq -l_{z_i}(1-y_i),$$

where u_{z_i} and $-l_{z_i}$ are the maximum cell deviations upwards and downwards, respectively. If the cell bounds are large, u_{z_i} and $-l_{z_i}$ may be large as well. The above constraints force that when $y_i = 1$ (protection direction is "upper") the downwards deviation must satisfy $z_i^- \leq -l_{z_i}(1-y_i) = 0$, thus it is 0. However, in practice, because of the feasibility tolerance, we may have for instance $y_i = 1-\epsilon$, and thus if $-l_{z_i} = M$, and M is a big-value, the constraint imposes $z_i^- \leq -l_{z_i}(1-y_i) = M(1-(1-\epsilon)) = M\epsilon > 0$. Therefore we allow a downwards deviation is a cell that was "upper" protected, leading to an underprotection. A similar reasoning applies for "lower" protected cells (i.e., $y_i = \epsilon$ instead of $y_i = 0$).

Decreasing the feasibility tolerance, we reduce the above ϵ value, but we make the problem much harder, and the solver may report it is infeasible. A best option, if possible, would be to avoid big-values M for cell deviations, but this means the real cell bounds (lower and upper bounds) should be small. If they were about 1.0e+4 or 1.0e+5, the above underprotection issue would not appear. However, in practice, real tables contain very big cell values, and the above "small" bounds are not possible. The user may try to tight them, if she/he has information about the data. Unfortunately, if the imposed bounds are too tight, the problem may become a "real" infeasible problem. The package includes an option (option "b" of `main_CTA`) to play with, which automatically sets a maximum bound for all deviations.

- **Integrality tolerance**. This is the amount by which the binary variables in the RCTA model can be different from 0 or 1, and still be considered 0 or 1. The CPLEX default is 1.0e−5; the XPRESS default is 5.0e−6. In CPLEX it must be a value greater or equal than 0; in XPRESS it must be greater or equal than the feasibility tolerance. This parameter is related with the above feasibility tolerance. Indeed combining both of them we may try to obtain

feasible/optimal solutions with no underprotected cells. We discussed in previous item how to avoid underprotections by tuning the feasibility tolerance. The integrality tolerance provides a new possibility: if it is set to a very small value, e.g., 1.0e−10, we are asking for binary solutions that are far from 0 or 1 at most 1.0e−10. Therefore the problem with constraints $z_i^+ \leq u_{z_i}\ y_i$ and $z_i^- \leq -l_{z_i}(1-y_i)$, explained above, may be avoided. Unfortunately, there are two drawbacks of this approach. The first is that this may significantly increase the solution time of the branch-and-cut procedure (very significantly, indeed). The second is that (unlike CPLEX) in XPRESS, as said above, the integrality tolerance must be greater or equal than the feasibility tolerance. Then if we reduce the integrality tolerance, we must reduce the feasibility tolerance as well, and then the algorithm may falsely conclude the problem is infeasible.

- **Infeasible problems**. If a not-too-small (or the default) feasibility tolerance is being used yet, and the solver is still reporting the problem as infeasible, it may help to tune the MIP emphasis parameters. They change the behaviour of the solver in the branch-and-cut tree, and may lead to feasible solutions. This behaviour may be changed with parameters "m" and "h" of `main_CTA`. A more detailed description of how these parameters affect the branch-and-cut procedure must be found in the CPLEX and XPRESS user's manuals [6, 12].
- **Scale**. Scale factor to apply to the input table. When *automatic* is selected, the scale is obtained from $\exp\left(\sum_{b\in Bounds} \frac{ln(|b|)}{2n}\right)$, where *Bounds* is the set of lower and upper bounds, l_{a_i}, and u_{a_i}, for every cell $i = 1, \ldots, n$, provided that the bound is not zero. When a proper scale factor is applied, the resulting problem is numerically better conditioned, and it is likely to reach a solution faster and to avoid undesirable protections, that is, sensitive cells actually inside their protection levels. If the automatic scale seems not to help the solver, you may try with another values manually entered. The best alternative choices may be slightly lower values (e.g. divide the automatic scale by two, four, etc., but not much more). Anyway, don't discard other options you feel sensible.

7 Some other package features

The CTA package implements several advanced features, which are described in next subsections. These features have have been added to the package within successive funded research projects.

7.1 Non-additive tables

If the original cell values of the tables do not satisfy $Aa = b$ (i.e., the table is non-additive), the deviations may be asked to make the resulting table both protected and additive. By default, when the package detects a non-additive table, it will make the resulting table additive, unless stated by the user. This is controlled by option `-a a` of main_CTA and main_TCTA: by default a='y', i.e. the deviations will make the table additive; otherwise, the user may set a='n'. Internally, RCTA computes the possible infeasibilities of the table as $b - Aa$, and then the constraints of the CTA model are $Az = b - Aa$ (instead of $Az = 0$, as in (2)). Indeed, note that if the original table does not satisfy $Aa = b$ then such a z makes the resulting table feasible:

$$A(a + z) = Aa + Az = Az + (b - Az) = b.$$

If the original table is already additive, then $b - Az = 0$ and thus $Az = b - Az = 0$ is equivalent to the constraints of (2).

7.2 Integrality of output tables

The package allows the cells deviations of the table to be integer. The resulting optimization problem to be solved by the package has thus integer variables for the cells deviations; in general this problem may become computationally more expensive than the standard model. To apply this option the input table may be integer (otherwise a solution can not be guaranteed). This option is activated with the `-intdev` or `-D` parameter. It can be used both with the standard CTA model and with the lexmin approach of Section 4—which does not includes binary variables. The related option named `-intdevtol` or `-E` is an integrality tolerance to round the output cells in the solution file when intdev is used.

7.3 Negative protection levels

If the problem has negative protection levels (which can be useful for the sequential protection of correlated tables), the optimization model (3) is no longer valid (let us name it the "classical" model). When negative protection levels are detected by the RCTA package, it automatically switches to the alternative model

$$\begin{array}{ll} \min\limits_{z^+,z^-,y} & \sum\limits_{i=1}^{n} w_i(z_i^+ + z_i^-) \\ \text{subject to} & A(z^+ - z^-) = b - Aa \\ & l_z \leq z^+ - z^- \leq u_z \\ & z_i^+ - z_i^- \geq upl_i y_i + l_{z_i}(1 - y_i) \quad i \in \mathcal{S} \\ & z_i^+ - z_i^- \leq -lpl_i(1 - y_i) + u_{z_i} y_i \quad i \in \mathcal{S} \\ & (z^+, z^-) \geq 0 \\ & y_i \in \{0, 1\} \quad i \in \mathcal{S}. \end{array} \tag{5}$$

This model, that will be referred as the "new" model, is valid for any kind of instance, with either positive or negative protection levels. However, it is less efficient than the classical model, and then, for problems with only positive protection levels, the classical model should be the best option. If the classical model has some difficulty (e.g., it is not able to obtain an optimal, even a feasible solution, by, e.g., numerical tolerances or any other cause) then the new model could be tried.

The model to be used is controlled by option `-o o` of main_CTA: by default o='a', i.e. automatic selection of the model: if there is a negative protection level, the new model is used, otherwise the classical model will be applied. Setting o='n' (new model) or o='c' (classical model), the user may set a particular model. Note that for instances with negative protection levels, the option o='c' is forbidden. For problems with only positive protection levels, either model can be used. The output on screen clearly states which was the model used for the optimization stage. Recently, a new model, named the "hybrid" model, was obtained, mixing the new and classical models. This model can be used for both negative and positive protection levels, and it is as efficient as the classical model if there are few negative protection levels. This hybrid model will be included in new releases of the RCTA package.

7.4 Repair infeasibility tool

If the problem is found to be infeasible by the solver, that is: there is no solution that can satisfy both $Aa = b$ and the protection levels, the case should be stated in another way such that it is possible to protect the table. Finding out which parameters (usually, variable bounds or protection

levels) are too tight for the problem is not an easy task, since the infeasibility could come from the interaction among several parameters, which individually may seem not troubling but jointly lead to an insoluble situation.

The code of main_cta includes now a tool for pseudo-automatic analysis of infeasible instances. The options controlling the feature are:

- `-r r`: apply repair infeasibility procedure: `r`='n' (no, by default) or `r`='y' (yes);
- `-x x`: file name with information for repair infeasibility tool (if needed).

The procedure tries to find a quasi-solution by relaxing the variable and the constraint bounds. Though the protected table provided is infeasible, it is intended to include minimal violations at specific cells, which could be enlightening to find a suitable way to protect the table.

If the user wants to relax only either a set of cells or a subset of $Aa = b$, and not any variable or constraint, a file can be used to tell the tool which units may be violated, through the control `-x x`, where `x` is the name of the file. The structure of the file should be like:

```
nr, number of constraints allowed to be violated (may be 0)
constraint number 1
...
constraint number nr
nx, number of cells allowed to be violated (may be 0)
cell number 1
...
cell number nx
ns, number of sensitive cells allowed to violate their protection levels (may be 0)
cell number 1
...
cell number ns
```

It should be taken into account that the numbers for the constraints have to be in the range from 0 to $m-1$, both included, and the numbers for the cells have to be in the range from 0 to $n-1$, both included. Besides, the `ns` cell numbers have to correspond to sensitive cells. The procedure stops if any inconsistency is found.

When a cell (sensitive or not) is included in the second section, its upper bound is relaxed, but not its lower bound. In turn, when a sensitive cell is included in the third section, the constraints:

$$\begin{aligned} upl_i\ y_i \le z_i^+ \le u_{z_i}\ y_i \quad i \in \mathcal{S} \\ lpl_i(1-y_i) \le z_i^- \le -l_{z_i}(1-y_i) \quad i \in \mathcal{S}, \end{aligned}$$

or, if negative protection levels are present (see section 7.3), the constraints:

$$\begin{aligned} z_i^+ - z_i^- \ge upl_i y_i + l_{z_i}(1-y_i) \quad i \in \mathcal{S} \\ z_i^+ - z_i^- \le -lpl_i(1-y_i) + u_{z_i} y_i \quad i \in \mathcal{S} \end{aligned}$$

may be relaxed, which in practice means that both protection levels can be violated.

If there exists a solution for the relaxed problem, the program main_CTA writes in an output file information about the reached table, specifically the information related to infeasible constraints or cells. The name of the file is the name of the case, with the extension `'inf'`.

References

[1] J. Castro, Minimum-distance controlled perturbation methods for large-scale tabular data protection, *European Journal of Operational Research*, 171 (2006) 39–52.

[2] J. Castro and S. Giessing, Testing variants of minimum distance controlled tabular adjustment, in Monographs of Official Statistics. Work session on Statistical Data Confidentiality, Eurostat-Office for Official Publications of the European Communities, Luxembourg, 2006, 333-343. ISBN 92–79-01108-1.

[3] J. Castro and J.A. González, A multiobjective LP approach for statistical disclosure control of tabular data, working paper, to be submitted (2014).

[4] J. Castro, J.A. González, X. Jiménez and D. Baena, Use's and programmer's manual of the RCTA package (v.2), Dept. of Statistics and Operations Research, Universitat Politècnica de Catalunya, 2014.

[5] COIN-OR, *OSI* https://projects.coin-or.org/Osi.

[6] Dash Optimization, *XPRESS Optimizer Reference Manual*, DASH, (2011).

[7] J. Forrest, D. de la Nuez and R. Lougee-Heimer, *CLP User Guide*, IBM Research, (2004).

[8] J. Forrest and R. Lougee-Heimer, *CBC User Guide*, IBM Research, (2005).

[9] S. Giessing, A. Hundepool and J. Castro, Rounding methods for protecting EU-aggregates , Joint UNECE/Eurostat Work Session on Statistical Data Confidentiality, Manchester (United Kingdom), December 2007.

[10] GNU Linear Programming Kit, *Reference Manual for GLPK Version 4.45*, DRAFT, (2010).

[11] J.A. González and J. Castro, A heuristic block coordinate descent approach for controlled tabular adjustment, *Computers & Operations Research*, 38 (2011) 1826–1835.

[12] ILOG CPLEX, *ILOG CPLEX 11.0 Reference Manual*, ILOG, (2007).

[13] T.K. Ralphs, M. Güzelsoy and A. Mahajan, *SYMPHONY 5.4.0 User's Manual*, (2011).

[14] J. Smart, R. Roebling, V. Zeitlin, R. Dunn, S. Csomor, F. Montorsi, B. Petty, et al, wx*Widgets* http://docs.wxwidgets.org/trunk/, (2012).