SOLVE_LP

A CLI TO SOLVE LINEAR PROGRAMMING PROBLEMS

WRITTEN BY: EDUARDO DOS SANTOS TEIXEIRA
SOURCE CODE AVAILABLE AT: HTTPS://GITHUB.COM/EDUARDO281/SOLVE_LP
CRITICS AND SUGGESTIONS CAN BE SENT TO: EDUARDO281.DEV@GMAIL.COM

Table of Contents

SECTION 1

Introduction

1.1 CLI Commands

1.1.1 Introduction

Solve_LP is a Command Line Interface (CLI), and so all its utilization is done through commands in a terminal. By now we are assuming users know the basic usage of this kind of tool. In future versions of this manual we will include some more informations to make it more beginner-friendly. In this Section we present all the Solve_LP arguments.

1.1.2 Solve_LP Commands

`--help/-h`

Used to display the general Solve_LP usage instructions. The text is auto generated by the Python's `argparse` module using small help instructions provided by the developer, and can be used as a quick reminder of all the Solve_LP commands.

`--file INPUT_FILE_PATH`

Solve_LP is a tool to solve linear programming problems, and so every time it is called, the user needs to specify which is the data file to be input. The `file` command is used to do it, and it is the only required argument that must be passed every time Solve_LP is invoked.

Usage example:

```
solve_lp --file DATA_FILE.mat
```

`--time/-t TOTAL_SOLVING_TIME`

Used to specify the total solution time of the optimization process (in general, mixed-integer linear programming solvers use Branch-and-Cut methods) in seconds.
Usage example:

```
solve_lp --file DATA_FILE.mat --time 30
```

`--solver/-s SOLVER_NAME`

Used to specify which optimization solver will be called by Solve_LP to tackle the input problem. The available solvers depend upon the modeling language used, and for each language it can change for different reasons. We will present a complete list of possible combinations of solver and modeling language in another Section of this manual.
Usage example:

```
solve_lp --file DATA_FILE.mat --solver CBC
```

`--gap/-g REL_MIP_GAP_VALUE`

Integer programming problems may involve very deep Branch-and-Cut trees, which may cause the solution procedure to be very time consuming. When this is the case, the solver will try to reach sufficiently close to an optimal solution (in practice, almost any computational mathematics problem is solved approximately). The gap represents this tolerance level accepted by the solver to consider that a solution is optimal.
Usage example:

```
solve_lp --file DATA_FILE.lp --solver CBC --time 600 --gap 0.01
```

`--digits/-d DIGITS_ON_SOLUTION`

Since computational mathematics problems are solved using approximately values, it may cause the final solutions obtained to be shown as approximately numbers, and so the user may get $x_1 = 0.99999$ instead of $x_1 = 1$. The `digits` command is used to round the values by specifying the maximum number of digits to be considered.
Usage example:

```
solve_lp --file DATA_FILE.json --digits 5 --time 60
```

`--threads/-r NUM_THREADS`

Some optimization solvers can perform parallel solution procedures, using more than a thread to tackle a problem by applying Branch-and-Cut, heuristics and maybe other solution strategies. The `threads` command is used to specify the maximum number of threads the solver is allowed to use, assuming the specified solver can use more than only one.
Usage example:

```
solve_lp --file DATA_FILE.lp --time 120 --threads 4
```

--modeling-language

Used to specify which modeling language will be used by Solve_LP among the ones available, which we describe in a later Section. Each modeling language have its own pros and cons, and sometimes this will not have any impact on the overall solution process. Experienced users may find this useful to test/apply some particular aspect of a desirable modeling language.

Usage example:

```
solve_lp --file DATA.mat --modeling-language OR-TOOLS --solver CBC
```

--zeros/--no-zeros

The command `zeros` is used to display all the variables in the final result, even the ones that have zero value. In cases the expected result is mostly made of zero values, the user can apply `no-zeros` to avoid an unnecessarily large result print.

Usage example:

```
solve_lp --file DATA_FILE.mat --time 300 --zeros
solve_lp --file DATA_FILE.lp --solver SCIP --no-zeros
```

--log/--no-log

Optimization solvers commonly offer the option of printing a log of the solution procedure. The commands `log` and `no-log` are used to control if the user want to see this log or not.

Usage example:

```
solve_lp --file DATA_FILE.mat --time 900 --solver CBC --log
solve_lp --file DATA_FILE.lp --modeling-language GUROBIPY --no-log
```

--verbose/--no-verbose

The pair of commands `verbose` and `no-verbose` are used to show or not to show the Solve_LP solving process steps log. We emphasize this is not the solver log, but a small log of the steps described by Solve_LP as it read the input data file, process it and solve the problem.

Usage example:

```
solve_lp --file DATA_FILE.json --verbose --no-log
solve_lp --file DATA_FILE.mps --time 30 --no-verbose --no-zeros
```

--export-solution-json JSON_FILEPATH

Solve_LP offers the option of exporting the solving results in a .json file through the command `export-solution-json`.
Usage example:

```
solve_lp --file DATA.mat --export-solution-json SOLUTION.json
```

`--export-mps-model MPS_FILEPATH`

Used to export the input model in a .mps file.
Usage example:

```
solve_lp --file DATA.mat --export-mps-model MODELFILE.mps
```

`--export-lp-model LP_FILEPATH`

Used to export the input model in a .lp file.
Usage example:

```
solve_lp --file DATA.mat --export-lp-model MODELFILE.lp
```

`--export-logfile LOGFILE_PATH`

Used to export the solver log in a text file. There is not standard format to this file, but we suggest to use .log as file type.
Important Note: This functionality was implemented by re-directing the solver log output to the logfile "manually". By the time this manual was written we did not find any error in this process, but this is still an experimental option.
Usage example:

```
solve_lp --file DATA.lp --export-logfile LOGFILE.log
```

`--matrix-input`

Tells Solve_LP to assume the input data file is in .mat format.
Usage example:

```
solve_lp --file DATA_FILE.mat --matrix-input
```

`--json-input`

Tells Solve_LP to assume the input data file is in .json format.
Usage example:

```
solve_lp --file DATA_FILE.json --json-input
```

`--mps-input`

Tells Solve_LP to assume the input data file is in .mps format.
Usage example:

```
solve_lp --file DATA_FILE.mps --mps-input
```

`--lp-input`

Tells Solve_LP to assume the input data file is in .lp format.
Usage example:

```
solve_lp --file DATA_FILE.lp --lp-input
```

`--infer-input`

Solve_LP can analyze the input file and try to detect its format without the need of indicating it explicitly. This process can lead to a few slower data reading process, but it is a good option in general cases.
Important Note: Although Solve_LP can recognize .mps and .lp files, the optimization solver only solve these types of files if they have the right file extension. In other words, if the user input an mps file named "my_data" to Solve_LP, it will recognize it is a .mps file, but the optimization solver will not solve it because it was not able to identify it. The user should rename it to "my_data.mps".
Usage example:

```
solve_lp --file DATA_FILE.mat --infer-input
```

`--skip-solve`

Used to only read the input data and execute model export commands, but not to solve the problem. Useful to perform conversions between different model formats
Usage example:

```
solve_lp --file DATA.mat --skip-solve --export-mps-model MODELFILE.mps
```

`--memory-limit MEMORY_LIMIT_IN_GB`

Allows to impose a limit on the total amount of memory available to the solving process. Can only be used when the modeling language is set to GUROBIPY.
Usage example:

```
solve_lp --file DATA.mat --modeling-language GUROBIPY --memory-limit 6
```

`--nodefile-start-size NODEFILE_START_SIZE_IN_GB`

Used to store the branch-and-cut tree nodes compressed in the disk, as an way to reduce RAM memory usage. When the total of memory used by the solver reaches the imposed limit value, the solver starts to write the nodes to the disk. Can only be used when the modeling language is set to GUROBIPY
Usage example:

```
solve_lp --file DATA.mat --modeling-language GUROBIPY
--nodefile-start-size 0.5
```

SECTION 2

Using Solve_LP

2.1 Mathematical Notations

Solve_LP receives as input the matricial representation of a Linear Programming Problem, following the notations:

• **x** represents the decision variables, where:

$$\mathbf{x} = [x_1, \quad x_2, \quad x_3, \quad ..., \quad x_n]$$

• **c** represents the costs vector, where:

$$\mathbf{c} = [c_1, \quad c_2, \quad c_3, \quad ..., \quad c_n]$$

• **b** represents the resources vector (a.k.a. the RHS - Right Hand Side - vector):

$$\mathbf{b} = [b_1, \quad b_2, \quad b_3, \quad ..., \quad b_m]$$

• **A** represents the coefficients matrix, given by:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

In which follows, it's considered as a linear programming problem in *standard format* a minimization problem where all the constraints are equalities and all the variables are

greater or equal to zero and do not have any upper bound. Using the formal mathematical notation, it can be generically represented as in (1).

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geqslant 0 \end{aligned} \tag{1}$$

An example of problem in this format is given in (LP1):

$$\begin{array}{lrcrcrcrcrcr} \min & 7x_1 & + & 9x_2 & + & 8x_3 & + & 5x_4 & + & 4x_5 & & \\ \text{s.t.} & 9x_1 & + & 4x_2 & + & 6x_3 & + & 6x_4 & + & 5x_5 & = & 42 \\ & 4x_1 & + & 8x_2 & + & 3x_3 & + & 4x_4 & + & 1x_5 & = & 62 \\ & 8x_1 & + & 5x_2 & + & 5x_3 & + & 4x_4 & + & 7x_5 & = & 46 \\ & x_1, & & x_2, & & x_3, & & x_4, & & x_5 & \geqslant & 0 \end{array} \tag{LP1}$$

It is not hard to solve (LP1) to optimality and find that its optimal value is:

$$x_1 = 0, x_2 = \frac{20}{3}, x_3 = 0, x_4 = 2, x_5 = \frac{2}{3}.$$

Solve_LP allows users to input problems in more general formats, including inequality constraints and redefining the variables lower and upper bounds. Problems input defined only by the parameters **A**, **b** and **c** will be interpreted as problems in the standard format.

2.2 File Formats

There are different ways to input data to Solve_LP, some of them are classical linear optimization file formats, and others were developed to make Solve_LP friendly to solve small problems, and offer human-readable ways to work with input data.

2.2.1 Matrix Format

The simplest format users can adopt to input data, the matrix format was especially developed to offer Solve_LP an easy and user-friendly way to input models and data using a vetorial notation similar to other mathematical softwares like SciLab and Octave.

2.2.2 Json Format

The json file format is very important in many different contexts when working with technology, particularly in web development. Users can pass data to Solve_LP using .json files with notations similar to the ones used in the matrix format.

2.2.3 LP Format

An user-friendly file format commonly accepeted by commercial and open source Optimization solvers, based on record the model and its data like the original mathematical representation.

2.2.4 MPS Format

A classical Linear Programming file format used to store models and its data. Not an user-friendly format and not the most used format to work with Optimization nowadays. More information can be found at:
https://en.wikipedia.org/wiki/MPS_(format)

3

SECTION

Solvers and Modeling Languages

3.1 Modeling Languages

In its present version Solve_LP can process the input data using three different modeling languages:

- OR-Tools: Open source project developed by Google that allows to solve continuous linear programming problems, as well as integer/mixed integer linear programming problems, constraint programming problems and includes some specialized solvers for some classic problems. In the case of Solve_LP, we only use its linear programming capabilities.
- Gurobipy: Modeling language developed by the Gurobi Optimization to be used along with the Gurobi Optimizer. It allows to model continuous and mixed integer linear and quadratic programming problems. In this project we are only interested in the linear programming features.
- Python-MIP: Open source modeling language developed as an academic initiative on Brazil that became part of the COIN-OR Foundation in 2019. Compatible with PyPy and with support to two important linear programming solvers, it is one of the options available on Solve_LP.

3.1.1 Useful Links

The user interested in read more about the modeling languages may find the following links useful:

• https://developers.google.com/optimization
Google developers page for OR-Tools.
• https://github.com/google/or-tools
OR-Tools's GitHub page.
• https://en.wikipedia.org/wiki/OR-Tools
OR-Tools's Wikipedia page.
• https://www.gurobi.com/
Gurobi official website.
• https://github.com/Gurobi
Gurobi's GitHub profile.
• https://www.python-mip.com/
Python-MIP's official page.
• https://github.com/coin-or/python-mip
Python-MIP's GitHub page.

3.2 Available Solvers

Each modeling language offers support to different optimization solvers. In this Sub-Section we present a brief description about them, and describe which solver can be used in combination with each modeling language. Most part of the solvers have easy-to-find webpages, where the owners/developers may clarify the less trivial doubts of the user. Please remember that part of the solvers require licenses to work, and that commercial uses of the solvers may required permission, even in the case of free for academic purposes solvers.

3.2.1 OR-Tools

The OR-Tools modeling language offers support to a lot of different linear optimization solvers, which we describe below:

- CLP: The COIN-OR Linear Programming solver. An open source project developed to solve continuous linear programming problems.
- CBC: The COIN-OR Branch-and-Cut solver. An open source project developed to solve binary/integer/mixed-integer linear programming problems.
- GLOP: An open source linear programming solver developed by Google. Solves only continuous problems.
- BOP: A free binary optimization solver.
- SAT or CP_SAT: The Google's CP-SAT solver. A constraint programming solver that uses SAT (satisfiability) methods.
- SCIP: A free for academic, non-commercial purposes solver for linear programming problems, continuos, integer or mixed-integer.

- GUROBI: A solver developed by the Gurobi Optimization group. Can be used to solve continuous, integer or mixed-integer linear programming problems.
- CPLEX: A solver developed by IBM. Can be used to solve continuous, integer or mixed-integer linear programming problems.
- XPRESS: A solver developed by FICO. Can be used to solve continuous, integer or mixed-integer linear programming problems.
- GLPK: The GNU Linear Programming Kit, developed to solve both continuous and mixed-integer linear programming problems.
- PDLP: A solver developed by Google that can be used to solve linear programming problems by using a Primal-Dual Hybrid Gradient method.

3.2.2 Gurobipy

The Gurobipy modeling language was especially developed to be used together with the Gurobi solver. It has some particular interesting tools users can utilize, as described in the command line arguments Section above.

3.2.3 Python-MIP

At the moment this manual was written, the Python-MIP official site was showing a message saying that Python-MIP is currently integrated with the C libraries of COIN-OR CBC solver and the commercial solver Gurobi, and that other solvers will be supported soon. The user may refer to the site to see if the support to any different solver was already made available in the last days.