Performance variability in MILP modeling

Rémi Garcia

Nantes Université, LS2N, France

remi.garcia@univ-nantes.fr

Keywords : *optimization, milp modeling, partial order, solver, performance variability*

1 Introduction

Since a few decades, research is going through the so-called replication crisis. It is known that different software versions or CPU architectures will impact the performance and results of an algorithm. This awareness has a positive impact on the research as more and more published papers pay more attention at the reproducibility aspect by giving more details about the proposed algorithms and experiment parameters. Yet, implementation choices are often left out of the publishing process and using the exact same machine and parameters as the original authors is usually impossible. With this work, we give some information on how the modeling language JuMP passes the constraints to the solvers and we present a new julia tool called SortModel which allows for changing the order of the constraints in the JuMP model. Using SortModel, we show the performance variability [3] induced by constraint ordering and provide a simple way to experiment with.

2 JuMP ordering

The following applies to JuMP 1.4.0 and should remain true for most, if not all, the JuMP 1.x versions [1]. First, it should be noted that the order of the constraints seems to be out of scope of the JuMP project.

Currently, JuMP stores the linear constraints of the model in three different blocks: the constraints "LessThan" the right hand side (RHS); the ones "GreaterThan" the RHS; and the equality constraints. When a process such as writing an LP file or sending the constraints to an optimizer is called, the constraints are passed block by block making it difficult to mix blocks together.

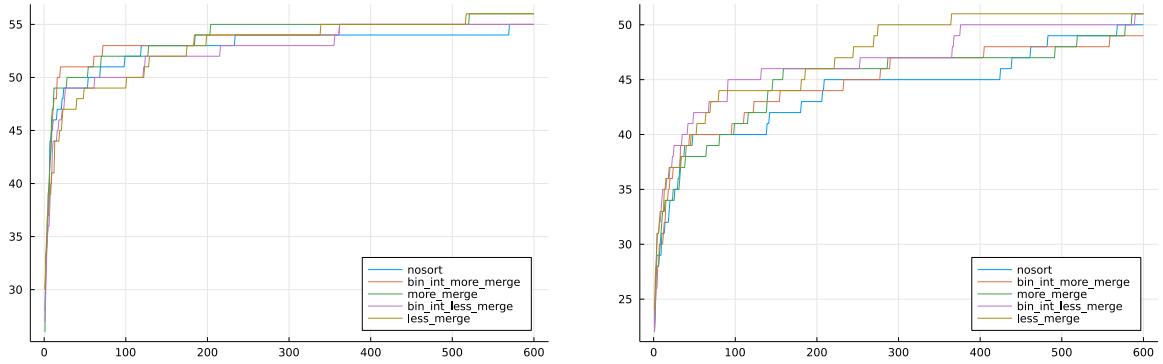
The order of the constraints within a block is determined by the declaration order, however, the order of the blocks is fixed by JuMP and depends on the function called, *e. g.*, when writing an `.lp` file, the constraints of the type $\leq$ are written first, followed by the $\geq$ ones and finishing with the equality constraints.

Following the order in which the constraints are given to the solver is not an easy task and could be of no interest if no impact on the solver could be observed.

3 SortModel to shuffle the deck

To explore the impact of different constraint ordering, we propose a julia package[1] that can reorder the constraints within the blocks. Our package exports the function `sort!` which can be applied to JuMP models. Without any keyword argument, the constraints are shuffled.

[1] `https://github.com/remi-garcia/SortModel.jl`



(a) Solver: Gurobi (b) Solver: CPLEX

FIG. 1: Number of instances solved at optimality after 1 to 600 seconds

Constraints can be sorted with respect to multiple criteria and not just shuffled. Currently, we propose the following criteria:

- ordering constraints by the number of involved variables;
- ordering constraints starting with the ones in which there is only binary variables;
- ordering constraints starting with the ones in which there is only discrete variables;
- ordering constraints starting with the ones in which there is only continuous variables.

Due to the JuMP way of storing constraints, blocks are still reordered separately, yet the we implemented an option that permits to merge the "LessThan" and "GreaterThan" blocks together. We will provide an option to decompose equality constraints into two "LessThan" constraints aiming at combining all the blocks into a single one.

4 Reordering effect

We experimented different sorting strategies and constraints shuffling on a model that solves the Multiple Constant Multiplication (MCM) problem [2]. We use 86 instances from the digital signal processing domain and solved our model with Gurobi and CPLEX. A time limit per instance of 10 minutes has been fixed.

In Figure 1, we show the number of instances solved at optimality for different sorting strategies: `nosort` means that we optimize without using SortModel; `more` (`less`) means that constraints with more (less) variables are passed first; `bin` (`int`) means that constraints with only binary (discrete) variables are passed first. We show that no strategy clearly outperform compared to the others. However, this clearly illustrates that different constraints orders will lead to different solving times. Particularly, on the instance `YLI01_30`, using Gurobi, while most sorting strategies timed out at the 600 seconds time limit, one of the shuffling proved optimality in less than a second.

We plan to do extensive testing of these strategies on different models from various domains. A specific order of the constraints might be preferred as it outperforms any other for a fixed model. In any case, when comparing models, we believe that multiple orders of the constraints should be tested to have a significant comparison.

References

[1] Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[2] Rémi Garcia, Anastasia Volkova, and Alexandre Goldsztejn. A New Model for the Multiple Constant Multiplication Problem. ROADEF2022, February 2022.

[3] Andrea Lodi and Andrea Tramontani. Performance Variability in Mixed-Integer Programming. In *Theory Driven by Influential Applications*, pages 1–12. INFORMS, September 2013.