A Julia package for bilevel optimization problems

Mathieu Besançon[1, 2, 3]

1 École Polytechnique de Montréal, QC, Canada **2** GERAD, QC, Canada **3** INOCS, INRIA Lille Nord-Europe, France

DOI: 10.21105/joss.01278

Software

- Review
- Repository
- Archive

Submitted: 30 January 2019
Published: 01 July 2019

Summary

Mathematical optimization is the discipline dealing with the determination of the best (or almost best) decision with respect to a specific cost function and to a set of constraints on the decision. Bilevel optimization is a class of mathematical optimization problems with the optimality conditions of a lower-level problem embedded in the constraints. BilevelOptimization.jl is a toolbox built on top of the JuMP.jl ecosystem for mathematical optimization (Dunning, Huchette, & Lubin, 2017). Bilevel optimization is used to tackle various problems in areas such as power systems, security applications, network design or market equilibria. See Dempe (2018) for an overview of applications and recent formulations and theoretical progress.

The computation of an optimal solution to a bilevel problem is in general hard. Even with all the constraints and the objectives at the two levels being linear, the resulting problem is non-convex and NP-hard, with a possibly disjoint feasible set. Optimization practitioners often rely on problem-specific properties and modeling techniques or heuristics. The goal of this package is to offer a both flexible model of a general class of bilevel problems and a solver which is compatible with the JuMP workflow.

Bilevel optimization

A generic formulation for a bilevel problem is:

```
min_x F(x,y)
s.t.  G_k(x,y) <= 0 for k in {1..mu}
      y in arg min_y {f(x,y)
               s.t.
               g_i(x,y) <= 0 for i in {1..ml}
              }
```

If the lower-level problem is convex, i.e., if the functions $f(x, y)$ and $g_i(x, \cdot)$ are convex and if Slater's qualification constraints hold, the Karush-Kuhn-Tucker conditions can be used to characterize the optimality at the lower-level.

For most lower-level problems, there are several optimal solutions (different solutions yielding the same optimal value of the objective). Several methodologies have been developed for such cases, the two primary approaches being the optimistic and pessimistic bilevel formulations (Dempe, 2018), regularizing the set-valued problem by guaranteeing the uniqueness of the lower-level solution. The approach used in *BilevelOptimization.jl* is the optimistic one.

This package is initially designed for a problem of the form:

```
min_x cx^T x + cy^T y
s.t. G x + H y <= q
     x >= 0
     x[j] integer for j in Jx
     y in arg min_y {d^T y + x^T F y
           s.t. A x + B y <= b
                y >= 0
                }
```

The single-level reduction of the optimistic version of this problem is:

```
min_{x,y,lambda} cx^T x + cy^T y
s.t. G x + H y <= q
     A x + B y <= b
     x, y >= 0
     x[j] integer for j in Jx
     d + F^T x + B^T * lambda = 0
     lambda[i] * (b - A x - B y)[i] = 0 for i in {1..ml}
```

The last equation is a complementarity constraint, corresponding to the fact that at least one of (λ_i, s_i) has to be equal to zero, where s_i is the slack variable associated with constraint i. This non-convex, non-linear constraint cannot be tackled efficiently by common optimization solvers and needs to be re-formulated. The two common approaches are linearization using a binary variable and "big-M" primal and dual upper bounds and Special Ordered Sets of type 1 (SOS1) (Pineda & Morales, 2019). A special ordered set 1 is a group of two or more variables, of which at most one can be non-zero. Mixed-Integer Linear solvers use this information for branching directly on the two variables. In the case of the bilevel problem presented above, the sets contain the slack variable and dual variable associated with each lower-level constraint, forcing at least one of them to zero.

Types and methods for bilevel optimization

The data for a bilevel linear problems are stored in the `BilevelLP` structure, with the same notation as used above. A `JuMP.Model` can be built from scratch from these data or passed on to the `build_blp_model` function which adds the lower-level feasibility and optimality constraints. This function supports different signatures through multiple dispatch. To build the model from scratch using the data from `BilevelLP`, the function is called as follows:

```
build_blp_model(bp::BilevelLP, solver; [comp_method])
```

It returns a tuple `(m, x, y, lambda, s)` with:

- `m`: the JuMP model,
- `x`: the upper-level vector of variables,
- `y`: the lower-level vector of variables,
- `lambda`: the lower-level dual variables, and
- `s`: the lower-level slack variables.

Other signatures allow users to modify an already-existing JuMP model without storing every constraint or variable into a `BilevelLP` structure.

All signatures of the build_blp_model function include a keyword argument comp_method for the choice of method to tackle complementarity constraints. The two methods mentioned in the previous section are represented as types, SOS1Complementarity and BoundComplementarity. For the second option, primal and dual bounds can either be scalars or vectors to use bounds adapted to each constraint. Some problem-specific methods are often used in the literature to handle complementarity constraints in an efficient way. Users of the package can define a new type, optionally sub-typed from ComplementarityMethod, and define the following function:

```
add_complementarity_constraint(m, cm::CM, s, lambda, y, sigma)
```

where CM is the complementarity constraint type.

Application to toll-setting problems

The toll-setting problem is a class of bilevel optimization where the two levels of decision are taken on a graph (Brotcorne, Labbé, Marcotte, & Savard, 2001). It belongs to the more general framework of network pricing problems with applications in road management (Harks, Schröder, & Vermeulen, 2019) or telecommunication network reliability (Hayrapetyan, Tardos, & Wexler, 2007).

In this problem, the upper level decides on a toll to apply on some arcs of a directed graph. Each arc has an initial cost and a cost resulting from an upper-level decision. The lower-level problem then consists in finding the minimum-cost flow from a source to a sink with a minimum circulating flow. This problem can be entirely modeled using the framework presented above, using a composite datatype defined in the package for holding all required data, and allowing users to bypass the re-formulation of the model from its algebraic JuMP form to a standard form. Users can describe their problem using the BilevelFlowProblem struct containing:

- The initial matrix of arc costs init_cost,
- A boolean matrix indicating which edges are taxable taxable_edges,
- The capacity matrix capacities,
- The different levels of tax that can be applied to each arc tax_options, and
- The minimum amount of flow that the follower needs to pass from source to sink minflow

Building the JuMP.Model is done similarly to the generic bilevel problem, using the following signature:

```
build_blp_model(bfp::BilevelFlowProblem, solver; [comp_method])
```

More general problem formulations

Even though BilevelOptimization.jl is designed for linear-linear bilevel problems which can be described by the BilevelLP type, the API allows users to bypass the upper-level problem specification. They can provide a pre-built JuMP.Model with the upper-level objective and constraints already set, for instance for quadratic or conic upper level formulations. The only requirement is that the solver must support both the type of constraints specified in the model and in the comp_method. This flexibility allows users to leverage some recent advances on mixed-integer convex optimization and solvers tackling these problems (Lubin, Yamangil, Bent, & Vielma, 2016). As of the current state of BilevelOptimization.jl, the only restricted part of the model is the linear-quadratic lower-level, which is required to exploit Karush-Kuhn-Tucker conditions.

References

Brotcorne, L., Labbé, M., Marcotte, P., & Savard, G. (2001). A bilevel model for toll optimization on a multicommodity transportation network. *Transportation science*, *35*(4), 345–358. doi:10.1287/trsc.35.4.345.10433

Dempe, S. (2018). *Bilevel optimization: Theory, algorithms and applications*. TU Bergakademie Freiberg, Fakultät für Mathematik und Informatik.

Dunning, I., Huchette, J., & Lubin, M. (2017). JuMP: A modeling language for mathematical optimization. *SIAM Review*, *59*(2), 295–320. doi:10.1137/15m1020575

Harks, T., Schröder, M., & Vermeulen, D. (2019). Toll caps in privatized road networks. *European Journal of Operational Research*. doi:10.1016/j.ejor.2019.01.059

Hayrapetyan, A., Tardos, É., & Wexler, T. (2007). A network pricing game for selfish traffic. *Distributed Computing*, *19*(4), 255–266. doi:10.1007/s00446-006-0020-y

Lubin, M., Yamangil, E., Bent, R., & Vielma, J. P. (2016). Extended formulations in mixed-integer convex programming. In Q. Louveaux & M. Skutella (Eds.), *Integer Programming and Combinatorial Optimization: 18th International Conference, IPCO 2016, Liège, Belgium, June 1-3, 2016, Proceedings* (pp. 102–113). Cham: Springer International Publishing. doi:10.1007/978-3-319-33461-5_9

Pineda, S., & Morales, J. M. (2019). Solving linear bilevel problems using big-ms: Not all that glitters is gold. *IEEE Transactions on Power Systems*, 1–1. doi:10.1109/TPWRS.2019.2892607