`BilevelJuMP.jl`: Modeling and Solving Bilevel Optimization in Julia

Joaquim Dias Garcia

LAMPS at PUC-Rio & PSR, joaquim@psr-inc.com

Guilherme Bodin

LAMPS at PUC-Rio & PSR, guilhermebodin@psr-inc.com

Alexandre Street

LAMPS at PUC-Rio, street@ele.puc-rio.br

In this paper we present BilevelJuMP, a new Julia package to support bilevel optimization within the JuMP framework. The package is a Julia library that enables the user to describe both upper and lower-level optimization problems using the JuMP algebraic syntax. Due to the generality and flexibility our library inherits from JuMP's syntax, our package allows users to model bilevel optimization problems with conic constraints in the lower level and all JuMP supported constraints in the upper level (Conic, Quadratic, Non-Linear, Integer, etc.). Moreover, the user-defined problem can be subsequently solved by various techniques relying on mathematical program with equilibrium constraints (MPEC) reformulations. Manipulations on the original problem data are possible due to MathOptInterface.jl's structures and Dualization.jl features. Hence, the proposed package allows quickly model, deploy, and thereby experiment bilevel models based on off-the-shelf mixed integer linear programming and nonlinear solvers.

Key words: Bilevel Optimization, Julia, JuMP, Algebraic Modeling Language, Automatic Reformulation

1. Introduction

Bilevel optimization has been a widely used modeling tool in mathematical programming, operations research and economics since its first introduction by Von Stackelberg (1952) in game theory. The broad range of applications include hyperparameter optimization in machine learning (Franceschi et al. 2018), toll setting in transportation networks (Labbé et al. 1998), multiple problems in energy and power systems (Pozo et al. 2017), defense applications (Bracken and McGill 1974), facility location (Küçükaydin et al. 2011) only to list a few areas.

Complete introductions to bilevel optimization can be found on books covering theoretical background and analysis, taxonomy, solutions algorithms for special classes, and selected applications (Dempe 2002, Bard 2013, Dempe et al. 2015). In addition to books, many

reviews on the subject have been published in the last decades (Vicente and Calamai 1994, Colson et al. 2007, Kalashnikov et al. 2015, Beck and Schmidt 2021). A very long list of publications related to bilevel optimization can be found in Dempe (2018).

It is well known that general bilevel optimization problems fall in the NP-hard class (Jeroslow 1985). Hence, there is no hope in finding efficient algorithms for generic problems. On the other hand, modeling bilevel problems is a theoretically more straightforward task, albeit, in practice, the modeling step can be the difference between finding a tractable model for which there is a reasonable solution approach in realistic cases or not. Because bilevel models are very complex and constitute a broad class of mathematical programming problems, many modeling languages lack the proper functionality to handle these problems.

In the following subsections, we present the main literature regarding available techniques and software to place and justify the contribution of our work properly. Notwithstanding, it is out of the scope of this section to provide a comprehensive review on the subject, for which we refer to previously reported books and reviews.

1.1. Solving bilevel optimization

Many strategies have been proposed to solve Bilevel Optimization problems. Some of the most widely known techniques are based on classical algorithms such as the simplex method, the branch and bound method, and the interior-point methods. Due to the inherent combinatorial nature of many Bilevel optimization problems, some of the early developed techniques are intrinsically combinatorial. Among a large set of enumeration algorithms, which can be seen as modifications of the simplex method for linear programming, we highlight the Kth-best algorithm (Bialas and Karwan 1982).

The fundamental technique that will be explored in this work is converting the bilevel problem into a single-level problem by adding the lower-level KKT conditions to the upper-level problem. The resulting optimization problem is known as Mathematical Programming with Equilibrium Constraints, MPEC (Kim et al. 2020). This group of techniques has also been labeled enumeration-based because the main difficulty is handling the complementarity constraints. Thus, a classic solution is a specially tailored Branch and Bound (Bard 1988, Hansen et al. 1992). Instead of writing a branch and bound method from scratch, one could reformulate the single-level problem into an amenable form for an off-the-shelf algorithm like mixed integer programming (MIP) solvers. This method was first presented relying upon

big-M formulations in Fortuny-Amat and McCarl (1981). More recently, a special ordered set of type 1, SOS1, formulations were developed in Siddiqui and Gabriel (2013).

Interestingly, non-linear programming problem (NLP), with additional regularization terms, can also be used to solve MPECs (Fletcher and Leyffer 2004, Scholtes 2001, Ralph and Wright 2004). In this case, it would not make as much sense to call the MPEC reformulations an enumeration-based method. On the other hand, the latter might be called local search methods as this strategy leads to local solutions, in contrast with the global ones provided by MIP-based methods. Some solvers were specially tailored to tackle MPEC's: KNITRO (Nocedal 2006), filterMPEC (Fletcher and Leyffer 2021, 2004), NLPEC (Ferris and GAMS 2021). A combination of NLP and MIP based methods was proposed by (Pineda et al. 2018).

Other strategies to solve bilevel problems include: bundle type algorithms (Falk and Liu 1995), semi-definite relaxations (Fampa et al. 2013), penalty function based methods (Aiyoshi and Shimizu 1984, White and Anandalingam 1993, Kleinert and Schmidt 2020a), Benders decomposition (Byeon and Van Hentenryck 2019). Cutting-planes approaches (Wu et al. 1998, Fischetti et al. 2017b, Tahernejad et al. 2020) have received attention recently because they can handle lower-level problems with integer variables — solvers were developed by Ralphs et al. (2019), Fischetti et al. (2017a). Descent methods were proposed by Kolstad and Lasdon (1990) and Vicente et al. (1994) to obtain quick local solutions. Heuristic methods were also developed to obtain practical solutions, for instance, the bi-objective-based method of Bard (1983). Finally, we refer to Sinha et al. (2017) for evolutionary approaches.

1.2. Modeling bilevel optimization

Algebraic modeling languages (AML) play a central role in unlocking the huge potential of optimization models through a mathematical friendly environment that integrates solvers and models in a very practical manner where problems of all disciplines can be efficiently modeled and solved. Bilevel Optimization interfaces, which mostly automate reformulations and pass to specific solver types, have also been proposed.

GAMS has an interface described in its Extended Mathematical Programming, GAMS EMP (2021). Variables and constraints are created as usual, and then they are annotated to specify the level they belong to in an external file. The annotated problem is reformulated by GAMS EMP routines using the KKT reformulations (Ferris et al. 2009).

Finally, the problem is optimized by the available MPEC solvers, namely, the above cited KNITRO and NLPEC. The follower subproblems can be linearly constrained with quadratic objectives (QP) or Variational Inequalities (VI). The upper-level problem is constrained by the selected solver capability.

The YALMIP MATLAB package for optimization modeling (Lofberg 2004) also provides a Bilevel Optimization interface YALMIP (2021). The variables and constraints are defined by the standard methods, and then they are passed as lists to a "solvebilevel" function. The lower level problem can be a QP, and the upper level can be anything supported by the YALMIP interface. The available solution methods are based on the MPEC reformulation where the handling of complementarity constraints is forwarded to external MIP, NLP or MPEC solvers or to an internal branch and bound that allows different solvers for upper and lower-level problems.

It is also possible to model Bilevel optimization in the Pyomo Adversarial Optimization, PAO (Hart and Castillo 2019), a Python package extending Pyomo (Hart et al. 2017). A (lower) submodel is created as an object attached to the main (upper) model, then variables and constraints are created directly in their owner models, the former can be shared in objectives and constraints. As in the previous two AMLs, the problem is automatically reformulated, and in Pyomo's case can be passed either to an NLP solver (Ralph and Wright 2004) or a MIP solver (Fortuny-Amat and McCarl 1981). Additionally, PAO has an interface to the MibS solver (Ralphs et al. 2019) and implements a Column Constraint Generation, CCG (Yue et al. 2019).

The Julia (Bezanson et al. 2017) package BilevelOptimization.jl (Besançon 2019) provides a very simple interface for Bilevel modeling in JuMP (Lubin and Dunning 2015, Dunning et al. 2017). Still, the interface remains very basic. For instance, although the upper level can represent arbitrary JuMP problems (as long as the selected solver supports them), the lower level is constrained to QP. Te package supports two MIP-based methods: Fortuny-Amat and McCarl (1981) and the SOS1 based reformulation. The critical issue is that the lower level is not represented by a JuMP-based syntax, preventing Julia users to fully enjoy the modeling power provided by JuMP. Instead, lower-level problems must be described by matrices, which can be easily manipulated to write KKT reformulations. This is one of the salient features and the primary goal of our propose package, BilevelJuMP,

namely, to allow representing the lower level problem within the JuMP syntax in a single and integrated new bilevel JuMP model.

It is important to highlight that, just like AIMMS (2021) and AMPL (2021), JuMP also has native support to complementarity constraints, thereby being capable of handling MPEC models. However, none of them can model Bilevel Optimization models directly.

1.3. Objective and Contributions

The main objective of this work is to provide a complete open-source interface for Bilevel Optimization fully integrated in the JuMP modeling language named BilevelJuMP.jl and available online at:

https://github.com/joaquimg/BilevelJuMP.jl

It is also readily available at the Julia Package manager, at the time of this paper, in version 0.5.1. Julia users can run `add BilevelJuMP` and have full access to all features of the library.

Regarding similar works, Besançon (2019) provided a great motivation and a nice first step to tackle bilevel models in Julia. However, it is incomplete as a modeling framework due to the strong limitations associated with the modeling of second-level problems as explained before. The consideration of a generic JuMP-based second-level model within a JuMP integrated interface significantly increases functionality parity with other AML's. Notwithstanding, and more importantly, it paves the way for new developments and computational applications based on bilevel optimization. Because the proposed package provides a simple an integrated interface, fully embedded into the JuMP language, new and expert users can easily prototype and test bilevel models enjoying all open-source and commercial linear, nonlinear, and MILP solvers integrated in JuMP, depending on the model characteristics. Hence, the newly proposed BilevelJuMP.jl can be used in teaching environments to introduce practical aspects of bilevel modeling as well as in practical applications inheriting many of the functionalities and advances directed to JuMP, one of the main packages of Julia.

Similar to other interfaces, BilevelJuMP.jl is also capable of reformulating bilevel problems and export the model to existing external solvers. In particular, many reformulations were implemented to allow practitioners to test each method in their particular problems. Some experiments are presented in this work to provide a first glance on the differences between existing methods.

Researchers of the Bilevel Optimization community are similarly benefited. The functionality exposed by BilevelJuMP.jl can be used as a benchmark in terms of performance and solution quality for new algorithms. These new algorithms might even be implemented in Julia with the data structures already defined in the package. Therefore, we also provide a software contribution with pieces of code that can be directly used in future implementations.

BilevelJuMP.jl was designed to be extensible, and the various implemented methods vouch for it. As we shall discuss, the key ingredient to developing all the solution methods is to rely on MathOptInterface.jl's API.

Advanced functionality is also part of the contributions. BilevelJuMP.jl can represent a Conic Program, CP, in the lower-level problem and can deal with upper-level constraints, including dual variables of lower-level problems (see Appendix A). Finally, the composability inherent in many Julia packages allows performing reformulations that enable the user to solve even more complex problem classes.

2. Conic Bilevel standard form

We will only consider optimistic bilevel problems (Dempe 2002), in short, the solution of the lower level will be the one that optimizes the upper level in case of degeneracy. We start by describing the main notation that will be used in the remainder of the work. z and x are vectors of decision variables, respectively, from the upper and lower-level problems. While x is n^U dimensional, z has n^L entries. $[x, z]$ is a $(n^L + n^U)$–vector with the elements of x and z stacked. Q^j, a_i^j, d_i^j, b_i^j, A_i^j, D_i^j, for $j \in \{U, L\}$ are matrices (upper case) and vectors (lower case) of constants. $\mathcal{C}_i^j$, for $j \in \{U, L\}$, are sets, of arbitrary finite dimension, which most commonly will be convex cones. m^U and m^L are the number of vector constraints in the upper and lower problems. As in traditional bilevel programming, z is decided in the upper level and passed to the lower level as a parameter and x might be seen as an upper-level variable constrained to be an optimal solution of the lower level. Hence, the optimistic bilevel problem follows:

$$
\begin{aligned}
\min_{x \in \mathbb{R}^{n^L}, z \in \mathbb{R}^{n^U}} \quad & \frac{1}{2}[x,z]^\top Q^U [x,z] + a_0^{U^\top} x + d_0^{U^\top} z + b_0^U \\
\text{s.t.} \quad & A_i^U x + D_i^U z + b_i^U \in \mathcal{C}_i^U, i = 1 \ldots m^U \\
& x(z) \in \operatorname*{arg\,min}_{x \in \mathbb{R}^{n^L}} \quad \frac{1}{2}[x,z]^\top Q^L [x,z] + a_0^{L^\top} x + d_0^{L^\top} z + b_0^L \\
& \qquad\quad \text{s.t.} \quad A_i^L x + D_i^L z + b_i^L \in \mathcal{C}_i^L, i = 1 \ldots m^L
\end{aligned}
$$

As detailed by Legat et al. (2020), describing constraints as function-set pairs is very flexible. For simplicity, we limited ourselves to affine functions contained in sets in the constraints of the above model. If all sets are all convex cones, we have a standard conic form for bilevel programs.

Keeping the lower level problem as a convex conic program is especially useful for writing KKT conditions when converting the problem into MPEC form. Although lower-level integer variables could be tackled by specialized solvers (Tahernejad et al. 2020), the same goes for non-linear constraints like the ones in (Sinha et al. 2017). The upper-level problem can be more complex, including non-linear constraints and integer variables, because they are not affected in MPEC reformulations.

2.1. KKT Reformulation of bilevel programs

Given a conic bilevel program in the standard form that we described in the previous section, we can formulate an equivalent MPEC applying the KKT reformulation to convert the lower level optimization problem into a set of linear and non-linear equations.

Let us focus on the following parametric convex quadratic conic problem that is equivalent to the lower level problem:

$$
\begin{aligned}
\min_{x \in \mathbb{R}^n} \quad & +\frac{1}{2}x^\top Q_1 x + x^\top Q_2 z + \frac{1}{2}z^\top Q_3 z + a_0{}^\top x + b_0 + d_0^\top z \\
\text{s.t.} \quad & A_i x + b_i + D_i z \in \mathcal{C}_i \qquad i = 1 \dots m \quad (1)
\end{aligned}
$$

Note that the L superscripts were dropped for simplicity, and we split the Q matrix in Q_1, Q_2, and Q_3. Because z are parameters, only Q_1 is required to be a positive semi-definite matrix. In the following, we will denote the dual cone of $\mathcal{C}_i$ as $\mathcal{C}_i^*$.

Following Boyd et al. (2004), we can write the KKT conditions as:

- Primal Feasibility:

$$
A_i x + b_i + D_i z \in \mathcal{C}_i, \quad i = 1 \dots m \tag{2}
$$

- Dual Feasibility:

$$
y_i \in \mathcal{C}_i^*, \quad i = 1 \dots m \tag{3}
$$

- Stationarity:

$$
Q_1 x + Q_2 z + a_0 - \sum_{i=1}^m A_i^\top y_i = 0 \tag{4}
$$

- Complementary slackness:

$$y_i^\top (A_i x + b_i + D_i z) = 0, \quad i = 1 \ldots m \tag{5}$$

Putting all together we arrive at the MPEC form of the bilevel conic program:

$$
\begin{aligned}
\min_{x \in \mathbb{R}^{n^L}, z \in \mathbb{R}^{n^U}} \quad & \frac{1}{2}[x,z]^\top Q^U [x,z] + a_0^{U\top} x + d_0^{U\top} z + b_0^U \\
\text{s.\,t.} \quad & A_i^U x + D_i^U z + b_i^U \in \mathcal{C}_i^U, \quad i = 1 \ldots m^U \\
& A_i^L x + b_i^L + D_i^L z \in \mathcal{C}_i^L, \quad i = 1 \ldots m^L \\
& y_i \in \mathcal{C}_i^{L*}, \quad i = 1 \ldots m^L \\
& Q_1^L x + Q_2^L z + a_0^L - \sum_{i=1}^m A_i^{L\top} y_i = 0 \\
& y_i^\top (A_i^L x + b_i^L + D_i^L z) = 0, \quad i = 1 \ldots m^L
\end{aligned}
$$

Such form is particularly useful to solve bilevel optimization problems as described in the previous sections. The main challenge being (5) constraints which are highly non-linear and non-convex. Dealing with the latter requires special solvers, tailor-made algorithms, or extra reformulation steps to reach the standard form of some NLP or MIP solver.

3. BilevelJuMP.jl

BilevelJuMP.jl is an extension of the JuMP modeling language (Dunning et al. 2017, Lubin and Dunning 2015) for optimization problems in the Julia language (Bezanson et al. 2017). Other packages successfully extending JuMP and MOI include SDDP.jl (Dowson and Kapelevich 2021), SumOfSquares.jl (Legat et al. 2021), InfiniteOpt.jl (Pulsipher et al. 2021). BilevelJuMP.jl has two main functionalities: modeling and solving bilevel optimization problems.

This open-source package heavily relies on MathOptInterface.jl, also referred to as MOI, (Legat et al. 2020), another Julia package that was written to be the new backend of JuMP which led to a complete rewrite of the latter. MathOptInterface.jl is an intermediary layer between JuMP's user-friendly AML interface and the diverse and typically matrix-oriented format of solvers. In BilevelJuMP.jl, MOI is used to store problem data from an extended JuMP interface and reformulate bilevel optimization problems into the MPEC form and then into a solver-compatible formulation of MPEC.

3.1. A Modeling Interface for Bilevel Optimization

The basic modeling interface of BilvelJuMP.jl relies on JuMP's extensible methods and macros to write and combine two optimization problems. Not surprisingly, other methods had to be created to accommodate the needs of bilevel optimization interfaces.

The main data structure in this software is the `BilevelModel`, which is a subtype of JuMP's `AbstractModel`. `BilevelModel` holds two other JuMP Models to represent the upper and lower optimization problems. Also, additional information is held to link the two problems and store additional JuMP data and attributes used in reformulations.

Just like a regular JuMP `Model`, the `BilevelModel` will need a solver constructor to solve an optimization problem. On the other hand, the BilevelModel will require a solution `mode` which will select the technique used in the solution process. The final pieces of the basic interface are the `Lower` and `Upper` methods that direct JuMP macros to the proper bilevel optimization levels.

We will exemplify the basic interface by modeling the following simple bilevel optimization problem from Dempe (2002), Chapter 3.2, Page 25:

$$
\begin{aligned}
\min_{y \in \mathbb{R}} \quad & 3x + y \\
\text{s.t.} \quad & x \leq 5 \\
& y \leq 8 \\
& y \geq 0 \\
& x(y) \in \arg\min_{x \in \mathbb{R}} \quad -x \\
& \qquad\qquad \text{s.t.} \qquad x + y \leq 8 \\
& \qquad\qquad\qquad\qquad 4x + y \geq 8 \\
& \qquad\qquad\qquad\qquad 2x + y \leq 13 \\
& \qquad\qquad\qquad\qquad 2x - 7y \leq 0
\end{aligned}
$$

The code to model, solve and query solutions is presented in Figure 1.

We can follow the general workflow: include packages; initialize the model jointly with a solver, `SCIP` (Bestuzheva et al. 2021) in this case, and the solution mode `SOS1Mode` (modes will be discussed in the following sections); add variables to the proper levels so that they can be used by all constraints and objectives; add constraints and objectives to the proper levels (which can be done in any order); optimize the model; and query solutions.

```
using JuMP, BilevelJuMP, SCIP
model = BilevelModel(SCIP.Optimizer, mode = BilevelJuMP.SOS1Mode())
@variable(Upper(model), y)
@variable(Lower(model), x)
@objective(Upper(model), Min, 3x + y)
@constraints(Upper(model), begin
    x <= 5
    y <= 8
    y >= 0
end)
@objective(Lower(model), Min, -x)
@constraints(Lower(model), begin
     x +  y <= 8
    4x +  y >= 8
    2x +  y <= 13
    2x - 7y <= 0
end)
optimize!(model)
objective_value(model) # = 3 * (3.5 * 8/15) + 8/15
value(x) # = 3.5 * 8/15
value(y) # = 8/15
```

Figure 1   **Code to solve the example of Dempe (2002), Chapter 3.2, Page 25**

3.2.   Solving Bilevel Optimization with Reformulations

MathOptInterface defines a unique and well-posed interface that makes it possible to perform reformulations in problem instances to convert from one format into others. We start from an arbitrary user formulation in JuMP stored as an MOI model, then we can rewrite this model in an alternate form, which will lead to an MOI model, and consequently, we can pass the model to a solver wrapper that implements the MOI API. The solver optimizes the model and returns the solutions, which flows back to JuMP by applying the necessary mappings and transformations. The simplest and most used of these transformations are bridges (Legat et al. 2020), which are applied in individual variables, constraints, and objectives. The bridge system automatically converts a problem in the specific form expected by the solver.

However, some transformations require looking at the model as a whole and not only at its pieces (variables, objectives, and constraints). The first implementation of a whole model transformation was Dualization.jl (Bodin et al. 2021). Dualization.jl's main function receives an MOI model and writes its dual in a second MOI model. Clearly, to perform such modification, the complete model must be known in advance. This feature is especially useful because some solvers only accept specific forms of mathematical programs. Hence, we can convert between primal and dual forms and solve the converted form without relying on the bridge system that might increase the problem size to reach the solver required form.

Dualization.jl also plays a key role in BilevelJuMP.jl's reformulations. Given a primal model like (1), where x are variables and z are parameters, we can obtain the dual form following Vandenberghe (2010):

$$\max_{y_1,\ldots,y_m,w} \quad -\frac{1}{2}w^\top Q_1 w + \frac{1}{2}z^\top Q_3 z - \sum_{i=1}^{m}(b_i + D_i z)^\top y_i + d_0^\top z + b_0$$

$$\text{s.t.} \qquad\qquad a_0 + Q_2 z - \sum_{i=1}^{m} A_i^\top y_i + Q_1 w = 0 \qquad (6)$$

$$y_i \in \mathcal{C}_i^* \quad i = 1\ldots m \quad (7)$$

We observe that the Dual feasibility constraints (6)–(7) are structurally very similar to the Dual Feasibility (3) and Stationarity (4) constraint sets from the KKT conditions. The only difference is that, in (6), Q_1 multiplies an additional variable w, and in (4) Q_1 multiplies a parameter z.

BilevelJuMP.jl performs a more complicated model transformation. The two JuMP models that described each level of the bilevel program must be combined in a particular way to create the corresponding MPEC. Even though each variable belongs to one level, they are created in both but tagged with additional data to mark their level and their corresponding variable in the other model, constraints, and objectives. However, they only exist in the level they were created.

The first part of the transformation is to copy the upper level into a new model to append the other pieces of the MPEC. The second step is to add the KKT conditions of the second level. The Primal Feasibility constraints of the lower level are added as new constraints to the model (using the variable map between the two original JuMP models). Then the lower level model is dualized, considering upper-level variables as constants, and its constraints are passed to the new model to represent Stationarity and Dual Feasibility constraints. The additional variables, w, created in the dual problem are mapped into the upper variables x. At this point, we only need to add complementarity constraints.

4. KKT Formulations

In this section, we describe the reformulation of the conic MPEC to obtain mathematical programs that can be passed to existing solvers.

4.1. Complementary slackness reformulations

Starting from a convex problem, all the KKT conditions lead to convex constraints except the complementary slackness constraints. The main challenge in KKT reformulations is dealing with such non-linearity. Now we present some possible formulations which were already implemented and tested in BilevelJuMP.

We will assume that y_i and $A_i x + b_i + D_i z$ are scalars, since almost all formulations rely on this assumption. The following formulations are restricted to: $\mathcal{C}_i \in \{\mathbb{R}_+, \mathbb{R}_-, \{0\}\}$. Without loss of generality we will assume $\mathcal{C}_i = \mathbb{R}_+$.

4.1.1. Special Ordered Sets of type 1

One SOS1 reformulation was presented in Kleinert and Schmidt (2020b) and in Siddiqui and Gabriel (2013). In BilevelJuMP.jl this formulation consists in replacing the complementarity constraints by the following:

$$f_i = A_i x + b_i + D_i z \tag{8}$$

$$[y_i; f_i] \in SOS1 \tag{9}$$

Considering this is the classic SOS1 set from Beale and Tomlin (1970), the SOS1 constraint implies that a solution is feasible only if at most one of the variables in the SOS1 set is different from zero. It is equivalent to the original formula because one of the two scalars will have to be zero to have the product equal to zero.

Many solvers can handle this kind of constraint, e.g., Cbc, CPLEX, Gurobi, SCIP, Xpress, which makes this formulation particularly useful for practitioners.

4.1.2. Indicator constraints

Indicator constraints (Belotti et al. 2016) and SOS1 are deeply related. A typical indicator constraint is defined by:

$$x = 0 \implies Ay \leq b \tag{10}$$

This means that the constraint $Ay \leq b$ is only considered if $x = 0$, where x is a binary variable. Analogously, another indicator constraint could depend on $x = 1$. Hence, one possible formulation for the complementarity slackness with indicator constraints is:

$$f = 0 \implies A_i x + b_i + D_i z = 0 \tag{11}$$

$$f = 1 \implies y_i = 0 \tag{12}$$

$$f \in \{0, 1\} \tag{13}$$

Many solvers are also capable of handling this kind of constraint which also makes this formulation very useful. As a final note on this formulation we note that a solver might not support Indicator Constraints for both $f = 0$ and $f = 1$, in this case we simply need one additional variable g and the constraint: $f + g = 1$.

4.1.3. Fortuny-Amat and McCarl

This formulation is commonly known by the name of the authors of Fortuny-Amat and McCarl (1981) and is extremely used in practice. In very few words, it is an application of the big-M method:

$$A_i x + b_i + D_i z \leq M_p f \tag{14}$$

$$y_i \leq M_d(1 - f) \tag{15}$$

$$f \in \{0, 1\} \tag{16}$$

In such formulation, M_p and M_d are large numbers. We have assumed that both $A_i x + b_i + D_i z$ and y_i are positive, thus, for each value of f one of the elements in the complementarity pair is forced to zero.

The main drawback of this method is that the values of M_p and M_d must be large enough so that the optimal solution of the problem is not excluded. One can usually develop bounds on primal variables because the variable might be bounded due to the problem physics. However, finding reasonable bounds for dual variables might be much harder on specific applications. The work by Pineda and Morales (2019) shows that commonly used heuristics to select the big-Ms can fail. Kleinert et al. (2020) go further and demonstrate that verifying big-Ms is at least as hard as solving the Bilevel Problem itself.

When good bounds are available, the Fourtuny-Amat and McCarl formulation is very efficient in practice (Kleinert and Schmidt 2020b). Moreover, no extra constraints are required from solvers. Therefore, less complete MIP solvers like GLPK can be used to solve this kind of reformulation. On the other hand, the difficulty of computing bounds makes the SOS1 and Indicator formulations very useful for experimentation.

4.1.4. Products

This is not a reformulation, because, in this case, the actual complementarity constraint in its product form is added to the optimization problem:

$$y_i^\top (A_i x + b_i + D_i z) = 0 \tag{17}$$

NLP solvers frequently use this form to seek local optimal solutions. Although no guarantees of global optimality are provided when using this method, it is useful the get initial solutions to be used as bounds or even for cases where MIP solving is not practical. An additional weakness of this method is that (17) does not satisfy constraint qualification (Scholtes 2001, Ralph and Wright 2004, Fletcher and Leyffer 2004) and is regularised as:

$$y_i^\top (A_i x + b_i + D_i z) \leq t \tag{18}$$

where t is a small number.

In theory, one could reformulate all the products with binary expansion techniques such as the one in Andrade et al. (2019) and use MIP solvers jointly with NLP solvers to reach solutions close to global optimal solutions. In practice, binary expansions also require bounds on the variables that are multiplied. This adds complications to the solution method because these cannot be added as regular constraints in the lower level; otherwise they would be dualized leading to more unbounded variables in both sides.

The binary expansion technique was implemented in QuadraticToBinary.jl (Dias Garcia 2021), which can be used as an intermediary layer between BilevelJuMP.jl (or JuMP) and the selected solver—allowing any MIP solver with an MOI interface to solve approximations of quadratically constrained problems.

This formulation easily extends to vector sets. Hence, conic bilevel problems will require this formulation in BilevelJuMP.

4.1.5.   Complements

Some solvers are able to handle explicit complement constraints like Knitro (Nocedal 2006), filterMPEC (Fletcher and Leyffer 2021, 2004), NLPEC (Ferris and GAMS 2021). These solvers receive the constraints as special structures: pairs of variables or variable-expression pairs. Internally, the solver will employ their own reformulations.

$$y_i \perp A_i x + b_i + D_i z \tag{19}$$

4.1.6.   Mixed mode

Usually, practitioners select one single formulation and apply them to all complementarity constraints in the problem, but this is not a technical requirement. Consequently, one could combine formulations and select which formulation will be used for each pair. For instance,

if one has good bounds for a specific pair, then just use Fortuny-Amat and McCarl for that constraint, while the other constraints would be reformulated with SOS1, for instance. Alternatively, even a conic bilevel with multiple linear constraints could be reformulated with SOS1 for all linear constraints and product mode (and binary expansions) for the conic constraints. We present an application of this method in Appendix B.

4.2. Primal Dual Equality reformulation

This formulation takes advantage of the fact that, under strong duality, the complementarity constraints are equivalent to enforcing that the primal and dual objective values are the same for a solution that is both primal and dual feasible. Therefore, the complementarity constraints are replaced by:

$$\frac{1}{2}x^\top P_1 x + x^\top P_2 z + a_0^\top x = -\frac{1}{2}w^\top P_1 w - \sum_{i=1}^{m}(b_i + D_i z)^\top y_i \tag{20}$$

Where the identical terms were already eliminated. This is also a non-convex quadratic constraint, even if the problem is linear due to z and y products.

One exciting feature of this formulation is that all the quadratic terms are concentrated in a single constraint, and the number of variable products might be much smaller than the number of complementarity constraints. Consequently, binary expansions were shown to be helpful to replace the quadratic terms and achieve approximate global optimal solutions in Pereira et al. (2005), Zare et al. (2019).

4.3. Comparison of methods

We present a brief comparison between the solution methods. Table 1 presents the method name in BilevelJuMP, section in which it is described, solver requirements and additional comments.

Method Name	Sec.	Solver requirement	Comments
SOS1Mode	4.1.1	MIP solver with SOS of type 1	No additional information. Only linear constraints.
IndicatorMode	4.1.2	MIP solver with Indicator Constraints	No additional information. Only linear constraints.
FortunyAmatMcCarlMode	4.1.3	MIP solver	Require non-trivial big-M. Only linear constraints.
ProductMode	4.1.4	Non-convex quadratic constraints	Works with conic constraints. Require regularization.
ComplementMode	4.1.5	Complementarity constraint	Few solvers supporting such constraints.
MixedMode	4.1.6	Requirements of selected methods	Pros and cons from selected methods.
StrongDualityMode	4.2	Non-convex quadratic constraints	Works with conic constraints.

Table 1   Reformulation methods

5.   Example

In this section, we describe a slightly more interesting example of a bilevel program. The main goal is to start from a non-trivial problem, model it in BilevelJuMP, and solve it with multiple methods to have a glimpse of the multitude of applications of the package.

As previously mentioned, hyperparameter tuning with bilevel optimization is a recent trend in the intersection of the Machine Learning and Optimization communities (Franceschi et al. 2018, Kunisch and Pock 2013, MacKay et al. 2019). Although most hyperparameter tuning methods based on bilevel optimization are usually heuristic with special considerations to the problem in question, this is a good case to describe the functionality of the package due to the simplicity of the model and because small enough instances can be solved by standard methods implemented in this package.

We have selected hyperparameter tuning in support vector regressions (SVR). The example will follow the one from Bennett et al. (2006), though with some simplifications. Given two data sets O and I with out-of-sample and in-sample data represented by the points labelled by i: $(y_i, \{x_{ij}\}_{j \in J})$, J is the feature set.

$$
\begin{aligned}
\min_{C \geq 0, \varepsilon \geq 0, \xi^U \geq 0} \quad & \sum_{i \in O} \xi_i^U \\
\text{s.t.} \quad & \xi_i^U \geq +y_i - \sum_j w_j x_{ij}, \quad i \in O \\
& \xi_i^U \geq -y_i + \sum_j w_j x_{ij}, \quad i \in O \\
& w(C, \varepsilon) \in \operatorname*{arg\,min}_{\xi^L \geq 0, w} \quad ||w||_2^2 + C \sum_{i \in I} \xi_i^L \\
& \qquad\qquad \text{s.t.} \quad \xi_i^L + \varepsilon \geq +y_i - \sum_{j \in J} w_j x_{ij}, \quad i \in I \\
& \qquad\qquad\qquad\quad \xi_i^L + \varepsilon \geq -y_i + \sum_{j \in J} w_j x_{ij}, \quad i \in I
\end{aligned}
$$

The lower model is responsible for obtaining the best possible support vectors w given the problem data and the hyperparameters C and ε, which are variables selected by the upper level so that the w optimized by the lower level has minimal out-of-sample error. The variables ξ^U and ξ^L denote the absolute value loss in the upper and lower models, respectively. The upper level is a linear program, while the lower level is quadratic. In Figure 2 we present BilevelJuMP.jl code to model the hyperparameter tuning of SVR described above. Thanks

to the JuMP syntax, the code greatly resembles the abstract model, simplifying the writing and documenting of the code.

```julia
using JuMP, BilevelJuMP
# sample data
Features = 2
Samples = 10
J = 1:Features
I = 1:div(Samples, 2)
O = (div(Samples, 2)+1):Samples
x = 2 * (rand(Samples, Features) .- 0.5)
w_real = ones(Features)
y = x * w_real .+ 0.1 * 2 * (rand(Samples) .- 0.5)
# model building
model = BilevelModel()
@variable(Upper(model), C >= 0)
@variable(Upper(model), eps >= 0)
@variable(Upper(model), xi_U[i=O] >= 0)
@variable(Lower(model), w[j=J])
@variable(Lower(model), xi_L[i=I] >= 0)
@objective(Upper(model),
    Min, sum(xi_U[i] for i in O))
@constraints(Upper(model), begin
    [i in O], xi_U[i] >= + y[i] - sum(w[j]*x[i,j] for j in J)
    [i in O], xi_U[i] >= - y[i] + sum(w[j]*x[i,j] for j in J)
end)
@objective(Lower(model),
    Min, sum(w[j]^2 for j in J) + C * sum(xi_L[i] for i in I))
@constraints(Lower(model), begin
    [i in I], xi_L[i] + eps >= + y[i] - sum(w[j]*x[i,j] for j in J)
    [i in I], xi_L[i] + eps >= - y[i] + sum(w[j]*x[i,j] for j in J)
end)
```

Figure 2   Code to model SVR hyper-parameter tuning

That same code was used to perform a series of comparisons between solvers. We started by creating instances with a different number of features and observations (dataset size). We randomly created the matrix x with a uniform distribution in $[-1, +1]$, then we created the *real* w as a vector of ones with appropriate dimension. Next, we defined $y = xw + \epsilon$, where ϵ follows a uniform distribution in $[-0.1, +0.1]$. Half of the dataset was considered in-sample data, while the other half was considered out-of-sample data. It is not our intention to be fully realistic here, our goal is to provide a didatic example.

We created instances with 10, 100 and 1000 samples. For all these sample sizes we created samples with 1, 2 and 5 features. For the datasets with 100 samples we also created datasets with 10, 20 and 50 features.

Finally, we optimized the bilevel problem for each data set with multiple reformulations and with multiple solvers. The only solver attribute we set was a time limit of 600 seconds

(10 minutes) and left all other attributes as default, which might differ considerably from one solver to the other. Again, our primary goal is not a detailed and rigorous comparison of solvers but to show the software's functionality in a usage example as practitioners and researchers might want to solve the same problem with multiple methods and select the one that best fits their needs.

We present results in the following tables. We used Julia 1.6.2, CPLEX 22.1 (IBM 2021), Gurobi 9.5 (Gurobi Optimization, LLC 2021), HiGHS 1.2 (Huangfu and Hall 2018), Ipopt 3.14 (Wächter and Biegler 2006), Knitro 13.0 (Nocedal 2006), SCIP 8.0 (Bestuzheva et al. 2021), Xpress 8.13 (FICO 2021). All the required code is in the benchmarks folder of the git repository, including exact package versions (see the *manifest.toml* file).

All tables have a similar format. The first column describes the instance, the first number being the sample size and the second the number of features. Then we have three columns for each solver, the first, *Obj*, presents the upper-level objective value returned by the solver (typically the best incumbent solution), the second contains the *Gap* in percent (%), Ipopt and KNITRO will not have gaps as they are NLP solvers, if no gap was reported the entry will be blank (with a "−"), the third is *Time* in seconds, if the time reaches 600 the entry will be blank (with a "−").

Table 2 presents results for SOS1Mode and IndicatorMode. Table 3 presents results for FortunyAmatMcCarlMode, with big-Ms set to 100, StrongDualityMode and ProductMode, the latter two with binary expansions, so the resulting problem is a MIP, where the variable bounds were set to $+/-100$. Finally, Table 4 presents the solutions of both ProductMode and StrongDualityMode for Non-Linear Programming solvers and Gurobi with its *NonConvex* mode activated.

We can draw some conclusions from the tables. We note that SOS1Mode and IndicatorMode perform well in smaller instances, with a slight advantage for SOS1Mode. Interestingly, CPLEX's solution for 1000/01 with IndicatorMode slightly disagrees with all solvers solution with the SOS1Mode. FortunyAmatMcCarlMode and StrongDualityMode seem very amenable to MIP solvers with Gurobi closing the gap within the given 10 minutes for all but one instance in the latter mode. However, we must be careful since we selected arbitrary bounds for those methods and StrongDualityMode also relies on binary expansion approximations, which led solvers to a solution that disagrees with the other methods on the 10/05 instance. On the other hand, ProductMode is the

worst strategy for MIP solvers in these instances. For NLP solvers both `ProductMode` and `StrongDualityMode` return objective values that are close to the ones found by MIP solvers, but in this case there is a slight advantage for `ProductMode`. Finally, Gurobi Non-Convex seems to work much better with `StrongDualityMode` claiming very good results in the instances with 1000 samples that agree with some of the other presented objective values.

The results are particular to a toy problem. However, the tables demonstrate that the software can interface with multiple solvers and consider multiple methods. Moreover, there is value in experimenting with multiple solvers and methods implemented in BilevelJuMP.

		CPLEX			Gurobi			SCIP			Xpress		
	Inst	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
SOS1Mode	10/01	0.30	0	0	0.30	0	0	0.30	0	0	0.30	0	0
	10/02	0.22	0	0	0.22	0	0	0.22	0	0	0.22	0	0
	10/05	0.09	0	0	0.09	0	0	0.09	0	0	0.09	0	0
	100/01	2.42	0	0	2.42	0	0	2.42	0	0	2.42	0	0
	100/02	2.40	4	-	2.40	4	-	2.40	4	-	2.40	4	-
	100/05	2.30	6	-	2.30	6	-	2.31	6	-	54.87	-	-
	100/10	8.54	392	-	79.59	-	-	79.59	-	-	79.59	-	-
	100/20	102.79	-	-	8.21	457	-	102.79	-	-	96.89	-	-
	100/50	23.35	307	-	23.35	299	-	23.35	-	-	23.35	350	-
	1000/01	25.02	0	-	28.63	14	-	25.02	0	-	25.02	0	-
	1000/02	323.30	-	-	323.30	-	-	323.30	-	-	323.30	-	-
	1000/05	533.37	-	-	533.37	-	-	533.37	-	-	533.37	-	-
IndicatorMode	10/01	0.30	0	0	0.30	0	0	0.30	0	0	0.30	0	0
	10/02	0.22	0	0	0.22	0	0	0.22	0	0	0.22	0	0
	10/05	0.09	0	0	0.09	0	0	0.09	0	0	0.09	0	0
	100/01	2.42	0	0	2.42	0	0	2.42	0	2	2.42	0	2
	100/02	2.40	4	-	2.40	4	-	9.08	294	-	2.42	5	-
	100/05	2.30	6	-	2.30	6	-	39.08	-	-	2.31	6	-
	100/10	79.59	-	-	79.59	-	-	79.59	-	-	79.59	-	-
	100/20	102.79	-	-	-	-	-	102.79	-	-	102.79	-	-
	100/50	23.35	-	-	23.35	-	-	23.35	-	-	23.35	576	-
	1000/01	25.06	0	-	-	-	-	77.45	209	-	195.11	680	-
	1000/02	323.30	-	-	-	-	-	323.30	-	-	-	-	-
	1000/05	533.37	-	-	533.37	-	-	533.37	-	-	-	-	-

Table 2   MIP solvers with `SOS1Mode` and `IndicatorMode`, Time in seconds (s), Gap in percent (%).

	Inst	CPLEX Obj	Gap	Time	Gurobi Obj	Gap	Time	HiGHS Obj	Gap	Time	SCIP Obj	Gap	Time	Xpress Obj	Gap	Time
FortunyAmatMcCarlMode	10/01	0.30	0	0	0.30	0	0	0.30	0	0	0.30	0	0	0.30	0	0
	10/02	0.22	0	0	0.22	0	0	0.22	0	0	0.22	0	0	0.22	0	0
	10/05	0.09	0	0	0.09	0	0	0.09	0	0	0.09	0	0	0.09	0	0
	100/01	2.42	0	0	2.42	0	0	2.42	0	0	2.42	0	1	2.42	0	0
	100/02	2.40	4	-	2.40	4	-	2.40	4	-	2.43	5	-	2.43	5	-
	100/05	2.30	6	-	2.29	5	-	54.87	-	-	39.08	-	-	2.31	6	-
	100/10	79.59	-	-	2.33	34	-	79.59	-	-	79.59	-	-	79.59	-	-
	100/20	22.11	-	-	22.17	-	-	102.79	-	-	102.79	-	-	-	-	-
	100/50	23.35	355	-	23.35	330	-	23.35	-	-	23.35	952	-	23.35	736	-
	1000/01	25.02	0	-	25.02	0	-	25.02	0	-	70.13	180	-	25.02	0	-
	1000/02	24.46	3	-	23.74	0	12	323.30	-	-	323.30	-	-	23.75	0	-
	1000/05	533.37	-	-	533.37	-	-	533.37	-	-	533.37	-	-	533.37	-	-
ProductMode	10/01	0.30	0	344	0.30	0	17	0.30	0	269	0.30	0	83	0.30	0	447
	10/02	1.82	739	-	0.22	0	-	3.12	-	-	0.33	0	495	0.30	38	-
	10/05	0.53	-	-	7.22	-	-	8.72	-	-	0.67	-	-	0.60	-	-
	100/01	18.08	647	-	14.37	494	-	23.28	0	52	21.87	803	-	20.74	0	55
	100/02	27.06	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	100/05	-	-	-	48.56	-	-	-	-	-	-	-	-	-	-	-
	100/10	-	-	-	75.06	-	-	-	-	-	-	-	-	78.62	-	-
	100/20	-	-	-	99.82	-	-	-	-	-	-	-	-	101.51	-	-
	100/50	247784.27	-	-	183.62	-	-	-	-	-	-	-	-	-	-	-
	1000/01	45.15	80	-	-	-	-	-	-	-	58.77	135	-	147.68	490	-
	1000/02	-	-	-	-	-	-	-	-	-	-	-	-	165.04	595	-
	1000/05	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
StrongDualityMode	10/01	0.30	0	512	0.30	0	-	0.30	0	-	0.30	0	-	0.30	0	-
	10/02	0.22	0	0	0.22	0	0	0.22	0	7	0.22	0	36	0.22	0	52
	10/05	0.00	0	2	0.00	0	3	0.00	0	165	0.00	0	37	0.00	-	4
	100/01	2.42	0	2	2.42	0	1	2.42	0	2	-	-	-	2.42	0	0
	100/02	2.30	0	7	2.30	0	10	2.30	0	63	2.36	2	-	-	-	-
	100/05	2.16	0	139	2.16	0	66	41.77	-	-	-	-	-	2.18	0	-
	100/10	1.73	0	269	1.73	0	34	75.84	-	-	78.61	-	-	-	-	-
	100/20	90.35	-	-	1.45	0	269	-	-	-	-	-	-	1.57	8	-
	100/50	176.21	-	-	160.70	-	-	-	-	-	-	-	-	-	-	-
	1000/01	25.01	0	153	25.01	0	22	25.01	0	32	25.01	0	20	25.01	0	10
	1000/02	23.74	0	39	23.74	0	23	23.75	0	-	-	-	-	23.74	0	20
	1000/05	529.71	-	-	24.43	0	543	408.46	-	-	-	-	-	24.43	0	482

Table 3   MIP solvers with **FortunyAmatMcCarlMode**, **ProductMode** and **StrongDualityMode**, Time in seconds (s), Gap in percent (%).

	Inst	Gurobi NonConvex			Ipopt			Knitro		
		Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
ProductMode	10/01	0.31	2	-	0.32	-	1	0.32	-	0
	10/02	0.22	3	-	0.22	-	1	0.22	-	0
	10/05	0.67	-	-	0.09	-	0	0.09	-	0
	100/01	2.42	0	6	2.43	-	18	2.46	-	0
	100/02	2.71	17	-	2.41	-	12	2.43	-	0
	100/05	54.87	-	-	2.47	-	24	2.54	-	0
	100/10	79.59	-	-	2.64	-	12	2.35	-	0
	100/20	102.79	-	-	3.43	-	11	3.51	-	0
	100/50	185.45	-	-	19.46	-	22	19.46	-	0
	1000/01	25.21	0	-	25.13	-	272	25.10	-	1
	1000/02	323.30	-	-	24.08	-	148	23.84	-	2
	1000/05	533.37	-	-	25.70	-	121	24.89	-	4
StrongDualityMode	10/01	0.30	1	-	0.32	-	0	0.33	-	0
	10/02	0.25	17	-	0.22	-	0	0.22	-	0
	10/05	0.13	-	-	0.09	-	0	0.09	-	0
	100/01	2.42	0	0	2.44	-	2	2.43	-	0
	100/02	2.41	4	-	2.44	-	2	2.53	-	0
	100/05	2.33	7	-	2.50	-	0	2.32	-	0
	100/10	2.41	39	-	2.21	-	1	2.08	-	0
	100/20	3.55	145	-	3.43	-	8	2.90	-	0
	100/50	64.20	-	-	23.34	-	1	185.45	-	0
	1000/01	25.03	0	-	54.89	-	3	25.08	-	3
	1000/02	23.80	0	-	71.32	-	9	23.78	-	6
	1000/05	24.86	1	-	29.52	-	175	24.75	-	4

Table 4   Gurobi NonConvex and NLP solvers with `ProductMode` and `StrongDualityMode`, Time in seconds (s),

Gap in percent (%).

6.   Package Comparison

A comparison between BilevelJuMP.jl and four other bilevel optimizations modeling interfaces that include solution methods is presented in Table 5. We include BilevelOptimization.jl as it was the key motivation for BilevelJuMP.jl; PAO, as the new bilevel interface of pyomo; GAMS that relies on EMP; and YALMIP that motivated the development of Dualization.jl.

In each table line, we briefly depict the answer to each of the following questions:

1. Which programming language does a user have to code the models?

2. What is the licensing scheme? (MIT is the most permissive among the ones shown).

3. Does the modeling interface support MIP solver-based methods, like SOS1 and big-M?

4. Does the modeling interface support NLP solver-based methods, like products?

5. Does the modeling interface support MPEC solvers that accept explicit complementarity constraints?

6. Can the user access dual variables of the lower-level problem and use them explicitly while modeling the upper-level problem?

7. Which problem classes are accepted in the lower-level problem?

8. Which problem classes are accepted in the upper-level problem?

The two latter questions used the following code: CP is Conic Programming, QP is linear programming with optional quadratic objective, NLP stands for Non-Linear Programming, VI represents Variational Inequalities, and Int is Integer Programming. Although one can model problems of given classes, specific solvers will be required to handle the resulting formulations. Finally, we note that all classes might not be supported simultaneously by all the interfaces, BilevelJuMP.jl supports all the described classes in the same model. Finally, it is worth mentioning that the possibility to consider bilevel models in which lower-level primal and dual variables are present in the first-level problem significantly enlarges the spectrum of practical applications that can be covered with the package. We marked YALMIP as ready to handle lower level duals because this can be achieved by explicitly calling their "kkt" function on the lower level data and appending to the primal problem. For instance, strategic bidding as well as market-power assessments in electricity markets highly depend on bilevel models with such dependencies Fanzeres et al. (2019).

7.   Conclusion

We presented BilevelJuMP, an open-source Julia library for Bilevel Optimization that allows the user to model a wide range of bilevel optimization problems very easily. Moreover, the

Name	BilevelJuMP.jl	BilevelOptimization.jl	PAO/Pyomo	GAMS	YALMIP
Language	Julia	Julia	Python	GAMS	MATLAB
License	MIT	MIT	BSD	Commercial	YALMIP
MIP solvers	Yes	Yes	Yes	No	Yes
NLP solvers	Yes	No	Yes	No	Yes
MPEC solvers	Yes	No	No	Yes	Yes
DualVar	Yes	No	No	Yes	Yes
Lower Level	CP/QP	QP (matrix form)	QP/Int	QP/NLP/VI	QP
Upper Level	CP/QP/NLP/Int	CP/QP/NLP/Int	QP/Int	QP/NLP/Int	CP/QP/NLP/Int

Table 5   Modeling interfaces for bilevel optimization

user has access to multiple reformulation techniques that can be considered to handle different problems better. More specifically, BilevelJuMP.jl allows modeling very general problems in the upper level (all JuMP supported formulations, such as non-linear, conic, mixed integer constraints) and conic problems in the lower level. Additionally, it implements multiple MPEC based reformulation techniques and MIP or NLP key as solution algorithms. This broad and flexible infrastructure of models and methods all built in a single open- source package for JuMP allows practitioners to use BilevelJuMP.jl for quick deploy and run experiments using state-of-the-art solvers and methods. It can be used by students introduced to the realm of bilevel optimization due to its easy-to-use and flexible structures, researchers and developers that can quickly test (or benchmark) new methods and models, and also develop new applications and packages, as well as by industry practitioners, who may not be familiar with bilevel solution strategies, but can rely on the package to address specific bilevel problems composing parts of real-world applications.

BilevelJuMP.jl joins a group of JuMP and MOI extensions that were made possible thanks to the good design of the latter two.

Just like JuMP and MOI, BilevelJuMP.jl is under active development, and more features are planned to be included. The library has gotten great interest from other contributors that are currently working towards new features, including support for integer variables in the lower-level with the solver MibS (Tahernejad et al. 2020). Further developments include: implementing other techniques such as valid inequalities (Kleinert et al. 2021), column-constraint generation based techniques (Yue et al. 2019); developing a file format for bilevel optimization based on MathOptFormat; integrating other solvers such as the one from Fischetti et al. (2017a).

Acknowledgments

Appendix A: Lower-level duals

This modeling feature enables the implementation of workflows where one (or more) of the upper-level variables is the dual of a lower-level constraint. In particular, in the energy sector, it is common to model the energy prices as the dual variable associated with the energy demand equilibrium constraint. One example of an application that uses this feature is Fanzeres et al. (2019), which focuses on strategic bidding in auction-based energy markets. A small and simplified example of the modeled problem would be the model:

$$\max_{\lambda, q_S} \quad \lambda \cdot g_S \tag{21}$$

$$\text{s.t.} \quad 0 \le q_S \le 100 \tag{22}$$

$$(g_S, \lambda) \in \underset{g_S, g_1, g_2, g_D}{\arg\min} \ 50g_{R1} + 100g_{R2} + 1000g_D \tag{23}$$

$$\text{s.t.} \quad g_S \le q_S \tag{24}$$

$$0 \le g_S \le 100 \tag{25}$$

$$0 \le g_1 \le 40 \tag{26}$$

$$0 \le g_2 \le 40 \tag{27}$$

$$0 \le g_D \le 100 \tag{28}$$

$$g_S + g_1 + g_2 + g_D = 100 \quad : \quad \lambda \tag{29}$$

Where λ is the dual of the load balance constraint (29), g_S, g_1, g_2 represent the generation of the strategic bidder and from two other (non-strategic) plants. g_D represents the deficit in generation. Finally, q_S is the quantity bid optimized by the strategic generator.

BilevelJuMP.jl allows users to implement similar models using the function `DualOf` that binds a new variable in the upper level to an existing constraint in the lower level. The model can be written as:

```
model = BilevelModel()
@variable(Upper(model), 0 <= qS <= 100)
@variable(Lower(model), 0 <= gS <= 100)
@variable(Lower(model), 0 <= gR1 <= 40)
@variable(Lower(model), 0 <= gR2 <= 40)
@variable(Lower(model), 0 <= gD <= 100)
@objective(Lower(model), Min, 50gR1 + 100gR2 + 1000gD)
@constraint(Lower(model), gS <= qS)
@constraint(Lower(model), demand_equilibrium, gS + gR1 + gR2 + gD == 100)
@variable(Upper(model), lambda, DualOf(demand_equilibrium))
@objective(Upper(model), Max, lambda*gS)
```

A.1. NLP solution

This model, can the be solved by selecting a reformulation and a solver. Here we select Strong-Duality reformulation, the Ipopt solver and call optimizes to perform the reformulation and solve it.

```
BilevelJuMP.set_mode(model, BilevelJuMP.StrongDualityMode())
set_optimizer(model, Ipopt.Optimizer)
optimize!(model)
```

A.2.   MIP solution

It is also possible to solve such problem by using a MIP formulation. The main issue is the product of variable in the upper level objective. However, this can be easily handled by using the aforementioned `QuadraticToBinary` package for automatic binary expansions. Because binary expansions require bounds on variables, we change the following line:

```
@variable(Upper(model), 0 <= lambda <= 1000, DualOf(demand_equilibrium))
```

Then, as before, we set a solver (now SCIP with the `QuadraticToBinary` wrapper) and a solution method (now Fortuny-Amat and McCarl):

```
set_optimizer(model,
    ()->QuadraticToBinary.Optimizer{Float64}(SCIP.Optimizer()))
BilevelJuMP.set_mode(model,
    BilevelJuMP.FortunyAmatMcCarlMode(dual_big_M = 100))
optimize!(model)
```

Appendix B: Conic Bilevel and Mixed Mode

Here we present a simple bilevel program with a conic lower level model described in example 3.3 from Chi et al. (2014).

$$\max_{x \in \mathbb{R}} \quad x + 3y_1 \tag{30}$$

$$\text{s.t.} \quad 2 \le x \le 6 \tag{31}$$

$$y(x) \in \arg\min_{y \in \mathbb{R}^3} -y_1 \tag{32}$$

$$\text{s.t.} \quad x + y_1 \le 8 \tag{33}$$

$$x + 4y_1 \ge 8 \tag{34}$$

$$x + 2y_1 \le 12 \tag{35}$$

$$y \in SOC_3 \tag{36}$$

In this problem most of the constraints are regular linear constraints while the last one, (36), is a second order cone constraint. Such constraint ensures that the vector y belongs to a second order cone of dimension 3, that is: $y_1 \ge \sqrt{y_2^2 + y_3^2}$. This problem can be encoded using regular JuMP syntax for conic programs:

```
model = BilevelModel()
@variable(Upper(model), x)
@variable(Lower(model), y[i=1:3])
@objective(Upper(model), Min, x + 3y[1])
@constraint(Upper(model), x >= 2)
@constraint(Upper(model), x <= 6)
@objective(Lower(model), Min, - y[1])
@constraint(Lower(model), con1, x +  y[1] <=  8)
```

```
@constraint(Lower(model), con2, x + 4y[1] >=  8)
@constraint(Lower(model), con3, x + 2y[1] <= 12)
@constraint(Lower(model), con4, y in SecondOrderCone())
```

B.1. NLP solution and start values

We can set, for instance, the product reformulation and selected Ipopt as a solver. As Ipopt does not have native support for second order cones, we use the non-default MOI bridge `SOCtoNonConvexQuad` to convert second order cones into quadratic constraints.

```
BilevelJuMP.set_mode(model,BilevelJuMP.ProductMode(1e-5))
set_optimizer(model,
     ()->MOI.Bridges.Constraint.SOCtoNonConvexQuad{Float64}(Ipopt.Optimizer()))
optimize!(model)
```

This problem is very simple, but more complex models might require more information such as starting points that can be passed on variable creation with standard JuMP syntax, for instance:

```
@variable(Upper(model), x, start = 6)
```

The user could also use the alternative JuMP syntax:

```
set_start_value(x, 6)
set_dual_start_value(con2, 0)
```

B.2. MIP solution and mixed mode

Alternatively, we could have used a Mixed Integer Second Order Cone Program (MISOCP) solver together with binary expansions. Complementarity of conic constraints is more difficult to handle because they require a sum of products that cannot be reformulated with other methods. Therefore, we rely on product reformulation for conic constraints. However, we can use other reformulations like indicator constraints for the non-conic constraints. Mixing the two of them can be done with Mixed Mode from Section 4.1.6.

The following code describes how to solve the problem with a MISOCP based solver.

```
set_optimizer(model,
     ()->QuadraticToBinary.Optimizer{Float64}(Xpress.Optimizer(),lb=-10,ub=10))
BilevelJuMP.set_mode(model,
     BilevelJuMP.MixedMode(default = BilevelJuMP.IndicatorMode()))
BilevelJuMP.set_mode(con4, BilevelJuMP.ProductMode(1e-5))
optimize!(model)
```

We set the reformulation method as Mixed Mode and selected Indicator constraints to be the default for the case in which we do not explicitly specify the reformulation. Then we set product mode for the second order cone reformulation.

As described in Appendix A, binary expansions require bounded variables, hence the `QuadraticToBinary` meta-solver accepts fallback to upper and lower bounds (`ub` and `lb`), used for variables with no explicit bounds.

References

AIMMS (2021) AIMMS complementarity problems. `https://documentation.aimms.com/language-reference/optimization-modeling-components/mixed-complementarity-problems/complementarity-problems.html`, accessed: 2021-04-17.

Aiyoshi E, Shimizu K (1984) A solution method for the static constrained stackelberg problem via penalty method. *IEEE Transactions on Automatic Control* 29(12):1111–1114.

AMPL (2021) AMPL complementarity problems. `https://ampl.com/products/solvers/all-solvers-for-ampl/`, accessed: 2021-04-17.

Andrade T, Oliveira F, Hamacher S, Eberhard A (2019) Enhancing the normalized multiparametric disaggregation technique for mixed-integer quadratic programming. *Journal of Global Optimization* 73(4):701–722.

Bard JF (1983) An efficient point algorithm for a linear two-stage optimization problem. *Operations Research* 31(4):670–684.

Bard JF (1988) Convex two-level optimization. *Mathematical programming* 40(1-3):15–27.

Bard JF (2013) *Practical bilevel optimization: algorithms and applications*, volume 30 (Springer Science & Business Media).

Beale EML, Tomlin JA (1970) Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. *OR* 69(447-454):99.

Beck Y, Schmidt M (2021) A gentle and incomplete introduction to bilevel optimization. Technical report, Lecture Notes, url: http://www. optimization-online. org/DB_HTML/2021/06/8450. html.

Belotti P, Bonami P, Fischetti M, Lodi A, Monaci M, Nogales-Gómez A, Salvagnin D (2016) On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications* 65(3):545–566.

Bennett KP, Hu J, Ji X, Kunapuli G, Pang JS (2006) Model selection via bilevel optimization. *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, 1922–1929 (IEEE).

Besançon M (2019) A julia package for bilevel optimization problems. *Journal of Open Source Software* 4(39):1278, ISSN 2475-9066, URL `http://dx.doi.org/10.21105/joss.01278`.

Bestuzheva K, Besançon M, Chen WK, Chmiela A, Donkiewicz T, van Doornmalen J, Eifler L, Gaul O, Gamrath G, Gleixner A, Gottwald L, Graczyk C, Halbig K, Hoen A, Hojny C, van der Hulst R, Koch T, Lübbecke M, Maher SJ, Matter F, Mühmer E, Müller B, Pfetsch ME, Rehfeldt D, Schlein S, Schlösser F, Serrano F, Shinano Y, Sofranac B, Turner M, Vigerske S, Wegscheider F, Wellner P, Weninger D,

Witzig J (2021) The SCIP Optimization Suite 8.0. Technical report, Optimization Online, URL `http://www.optimization-online.org/DB_HTML/2021/12/8728.html`.

Bezanson J, Edelman A, Karpinski S, Shah VB (2017) Julia: A fresh approach to numerical computing. *SIAM review* 59(1):65–98.

Bialas W, Karwan M (1982) On two-level optimization. *IEEE transactions on automatic control* 27(1):211–214.

Bodin G, Garcia JD, Legat B, Besançon M, Lubin M, Dowson O, Contributors (2021) jump-dev/Dualization.jl: v0.3.4. URL `http://dx.doi.org/10.5281/zenodo.4718987`.

Boyd S, Boyd SP, Vandenberghe L (2004) *Convex optimization* (Cambridge university press).

Bracken J, McGill JT (1974) Defense applications of mathematical programs with optimization problems in the constraints. *Operations Research* 22(5):1086–1096.

Byeon G, Van Hentenryck P (2019) Benders subproblem decomposition for discrete-continuous bilevel problems. *arXiv preprint arXiv:1902.04375* .

Chi X, Wan Z, Hao Z (2014) The models of bilevel programming with lower level second-order cone programs. *Journal of Inequalities and Applications* 2014(1):1–23.

Colson B, Marcotte P, Savard G (2007) An overview of bilevel optimization. *Annals of operations research* 153(1):235–256.

Dempe S (2002) *Foundations of bilevel programming* (Springer Science & Business Media).

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

Dempe S, Kalashnikov V, Pérez-Valdés GA, Kalashnykova N (2015) Bilevel programming problems. *Energy Systems. Springer, Berlin* .

Dias Garcia J (2021) joaquimg/QuadraticToBinary.jl: v0.2.4. URL `http://dx.doi.org/10.5281/zenodo.4718981`.

Dowson O, Kapelevich L (2021) SDDP.jl: a julia package for stochastic dual dynamic programming. *INFORMS Journal on Computing* 33(1):27–33.

Dunning I, Huchette J, Lubin M (2017) JuMP: A modeling language for mathematical optimization. *SIAM Review* 59(2):295–320, URL `http://dx.doi.org/10.1137/15M1020575`.

Falk JE, Liu J (1995) On bilevel programming, part i: general nonlinear cases. *Mathematical Programming* 70(1-3):47–72.

Fampa MH, Melo WA, Maculan N (2013) Semidefinite relaxation for linear programs with equilibrium constraints. *International Transactions in Operational Research* 20(2):201–212.

Fanzeres B, Ahmed S, Street A (2019) Robust strategic bidding in auction-based markets. *European Journal of Operational Research* 272(3):1158–1172, ISSN 0377-2217, URL `http://dx.doi.org/https://doi.org/10.1016/j.ejor.2018.07.027`.

Ferris M, GAMS (2021) NLPEC - GAMS solvers. `https://www.gams.com/33/docs/S_NLPEC.html`, accessed: 2021-04-17.

Ferris MC, Dirkse SP, Jagla JH, Meeraus A (2009) An extended mathematical programming framework. *Computers & Chemical Engineering* 33(12):1973–1982.

FICO (2021) FICO Xpress Optimizer Reference Manual. URL `https://www.fico.com/en/products/fico-xpress-solver`.

Fischetti M, Ljubić I, Monaci M, Sinnl M (2017a) Bilevel integer programming and interdiction problems - solver for mixed-integer bilevel linear problems. `https://msinnl.github.io/pages/bilevel.html`, accessed: 2021-04-17.

Fischetti M, Ljubić I, Monaci M, Sinnl M (2017b) A new general-purpose algorithm for mixed-integer bilevel linear programs. *Operations Research* 65(6):1615–1637.

Fletcher R, Leyffer S (2004) Solving mathematical programs with complementarity constraints as nonlinear programs. *Optimization Methods and Software* 19(1):15–40.

Fletcher R, Leyffer S (2021) filterMPEC neos solvers. `https://neos-server.org/neos/solvers/mpec:filterMPEC/AMPL.html`, accessed: 2021-04-17.

Fortuny-Amat J, McCarl B (1981) A representation and economic interpretation of a two-level programming problem. *Journal of the operational Research Society* 32(9):783–792.

Franceschi L, Frasconi P, Salzo S, Grazzi R, Pontil M (2018) Bilevel programming for hyperparameter optimization and meta-learning. *International Conference on Machine Learning*, 1568–1577.

GAMS EMP (2021) Bilevel programs. `https://www.gams.com/latest/docs/UG_EMP_Bilevel.html`, accessed: 2021-04-17.

Gurobi Optimization, LLC (2021) Gurobi Optimizer Reference Manual. URL `https://www.gurobi.com`.

Hansen P, Jaumard B, Savard G (1992) New branch-and-bound rules for linear bilevel programming. *SIAM Journal on scientific and Statistical Computing* 13(5):1194–1217.

Hart W, Castillo A (2019) PAO. [Computer Software] `https://doi.org/10.11578/dc.20201105.1`, URL `http://dx.doi.org/10.11578/dc.20201105.1`.

Hart WE, Laird CD, Watson JP, Woodruff DL, Hackebeil GA, Nicholson BL, Siirola JD (2017) *Pyomo-optimization modeling in python*, volume 67 (Springer).

Huangfu Q, Hall JAJ (2018) Parallelizing the dual revised simplex method. *Mathematical Programming Computation* 10(1):119–142, URL `http://dx.doi.org/10.1007/s12532-017-0130-5`.

IBM (2021) CPLEX Optimizer User's Manual. URL `https://www.ibm.com/analytics/cplex-optimizer`.

Jeroslow RG (1985) The polynomial hierarchy and a simple model for competitive analysis. *Mathematical programming* 32(2):146–164.

Kalashnikov VV, Dempe S, Pérez-Valdés GA, Kalashnykova NI, Camacho-Vallejo JF (2015) Bilevel programming and applications. *Mathematical Problems in Engineering* 2015.

Kim Y, Leyffer S, Munson T (2020) Mpec methods for bilevel optimization problems. *Bilevel Optimization*, 335–360 (Springer).

Kleinert T, Labbé M, Plein F, Schmidt M (2021) Closing the gap in linear bilevel optimization: a new valid primal-dual inequality. *Optimization Letters* 15(4):1027–1040.

Kleinert T, Labbé M, Plein Fa, Schmidt M (2020) There's no free lunch: on the hardness of choosing a correct big-m in bilevel optimization. *Operations research* 68(6):1716–1721.

Kleinert T, Schmidt M (2020a) Computing feasible points of bilevel problems with a penalty alternating direction method. *INFORMS Journal on Computing* .

Kleinert T, Schmidt M (2020b) Why there is no need to use a big-m in linear bilevel optimization: A computational study of two ready-to-use approaches. Technical report, Tech. rep. url: http://www. optimization-online. org/DB_HTML/2020/10/8065. html.

Kolstad CD, Lasdon LS (1990) Derivative evaluation and computational experience with large bilevel mathematical programs. *Journal of optimization theory and applications* 65(3):485–499.

Küçükaydin H, Aras N, Altınel IK (2011) Competitive facility location problem with attractiveness adjustment of the follower: A bilevel programming model and its solution. *European Journal of Operational Research* 208(3):206–220.

Kunisch K, Pock T (2013) A bilevel optimization approach for parameter learning in variational models. *SIAM Journal on Imaging Sciences* 6(2):938–983.

Labbé M, Marcotte P, Savard G (1998) A bilevel model of taxation and its application to optimal highway pricing. *Management science* 44(12-part-1):1608–1622.

Legat B, Dowson O, Garcia JD, Lubin M (2020) Mathoptinterface: a data structure for mathematical optimization problems. *arXiv preprint arXiv:2002.03447* .

Legat B, Weisser T, Kapelevich L, Huchette J, contributors (2021) jump-dev/sumofsquares.jl: v0.4.6. URL `http://dx.doi.org/10.5281/zenodo.4708982`.

Lofberg J (2004) Yalmip: A toolbox for modeling and optimization in matlab. *2004 IEEE international conference on robotics and automation (IEEE Cat. No. 04CH37508)*, 284–289 (IEEE).

Lubin M, Dunning I (2015) Computing in operations research using julia. *INFORMS Journal on Computing* 27(2):238–248, URL `http://dx.doi.org/10.1287/ijoc.2014.0623`.

MacKay M, Vicol P, Lorraine J, Duvenaud D, Grosse R (2019) Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *arXiv preprint arXiv:1903.03088* .

Nocedal J (2006) Knitro: An integrated package for nonlinear optimization. *Large-Scale Nonlinear Optimization*, 35–60 (Springer).

Pereira MV, Granville S, Fampa MH, Dix R, Barroso LA (2005) Strategic bidding under uncertainty: a binary expansion approach. *IEEE Transactions on Power Systems* 20(1):180–188.

Pineda S, Bylling H, Morales J (2018) Efficiently solving linear bilevel programming problems using off-the-shelf optimization software. *Optimization and Engineering* 19(1):187–211.

Pineda S, Morales JM (2019) Solving linear bilevel problems using big-ms: Not all that glitters is gold. *IEEE Transactions on Power Systems* 34(3):2469–2471.

Pozo D, Sauma E, Contreras J (2017) Basic theoretical foundations and insights on bilevel models and their applications to power systems. *Annals of Operations Research* 254(1-2):303–334.

Pulsipher JL, Zhang W, Hongisto TJ, Zavala VM (2021) A unifying modeling abstraction for infinite-dimensional optimization. *arXiv preprint arXiv:2106.12689* .

Ralph D, Wright SJ (2004) Some properties of regularization and penalization schemes for mpecs. *Optimization Methods and Software* 19(5):527–556.

Ralphs T, Tahernejad S, Vigerske S, Besançon M (2019) coin-or/mibs: Version 1.1.3. URL `http://dx.doi.org/10.5281/zenodo.3560359`.

Scholtes S (2001) Convergence properties of a regularization scheme for mathematical programs with complementarity constraints. *SIAM Journal on Optimization* 11(4):918–936.

Siddiqui S, Gabriel SA (2013) An sos1-based approach for solving mpecs with a natural gas market application. *Networks and Spatial Economics* 13(2):205–227.

Sinha A, Malo P, Deb K (2017) A review on bilevel optimization: from classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation* 22(2):276–295.

Tahernejad S, Ralphs TK, DeNegre ST (2020) A branch-and-cut algorithm for mixed integer bilevel linear optimization problems and its implementation. *Mathematical Programming Computation* 12(4):529–568.

Vandenberghe L (2010) The cvxopt linear and quadratic cone program solvers. *Online: http://cvxopt. org/-documentation/coneprog. pdf* .

Vicente L, Savard G, Júdice J (1994) Descent approaches for quadratic bilevel programming. *Journal of Optimization Theory and Applications* 81(2):379–399.

Vicente LN, Calamai PH (1994) Bilevel and multilevel programming: A bibliography review. *Journal of Global optimization* 5(3):291–306.

Von Stackelberg H (1952) *The theory of the market economy* (Oxford University Press).

Wächter A, Biegler LT (2006) On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming* 106(1):25–57.

White DJ, Anandalingam G (1993) A penalty function approach for solving bi-level linear programs. *Journal of Global Optimization* 3(4):397–419.

Wu S, Marcotte P, Chen Y (1998) A cutting plane method for linear bilevel programs. *Systems Science and Mathematical Sciences* 11:125–133.

YALMIP (2021) YALMIP bilevel programming. `https://yalmip.github.io/tutorial/bilevelprogramming/`, accessed: 2021-04-17.

Yue D, Gao J, Zeng B, You F (2019) A projection-based reformulation and decomposition algorithm for global optimization of a class of mixed integer bilevel linear programs. *Journal of Global Optimization* 73(1):27–57.

Zare MH, Borrero JS, Zeng B, Prokopyev OA (2019) A note on linearized reformulations for a class of bilevel linear integer problems. *Annals of Operations Research* 272(1-2):99–117.