ZERO: Playing Mathematical Programming Games

Gabriele Dragotto*, Sriram Sankaranarayanan†, Margarida Carvalho‡, Andrea Lodi§

Abstract

We present ZERO, a modular and extensible C++ library interfacing Mathematical Programming and Game Theory. ZERO provides a comprehensive toolkit of modeling interfaces and algorithms for Reciprocally Bilinear Games (RBGs), i.e., simultaneous non-cooperative games where each player solves a mathematical program with a linear objective in the player’s variable and bilinear in its opponents’ variables. This class of games generalizes the classical problems of Operations Research to a multi-agent setting. ZERO modular structure gives users all the elementary ingredients to design new game-theoretic models and algorithms for RBGs, and find their Nash equilibria. The library provides additional extended support for integer non-convexities, linear bilevel problems, and linear equilibrium problems with equilibrium constraints. We provide an overview of the software’s key components and showcase a Knapsack Game, i.e., a game where each player solves a binary knapsack problem. Aiming to boost practical methodological contributions at the interplay of Mathematical Programming and Game Theory, we release ZERO as open-source software. Source code, documentation and examples are available at www.getzero.one.

1 Why Games and Equilibria?

The pioneering book from Morgenstern and Von Neumann (1953) and the seminal papers from Nash (1950, 1951) transformed the scientific perspective on strategic behavior. The ubiquitous concepts of Nash equilibrium and rationality are now cornerstone concepts in Game Theory, with applications ranging from Economics to Social Sciences. The growing interest in game dynamics in the Operations Research community reflects a need to extend classical decision-making frameworks to multi-agent settings that can account for interactions
among multiple decision-makers. The community devoted particular interest – to name a few – to bilevel programming (e.g., Hu and Ralph (2007), DeNegre and Ralphs (2009), Labbé and Violin (2013), Caprara et al. (2016), Fischetti et al. (2017), Basu et al. (2019), Kleinert et al. (2021)) and its application in electricity markets and network pricing problems (Feijoo and Das 2015, Labbé et al. 1998, Brotcorne et al. 2001), equilibrium problems with equilibrium constraints (Luo et al. 1996, Carvalho et al. 2019), and more recently to integer programming games (Carvalho et al. 2018, 2020, 2021, Cronert and Minner 2021, Guo et al. 2021, Köppe et al. 2011, Dragotto and Scatamacchia 2021). On the one hand, such empowering modeling capabilities unquestionably offer a tempting opportunity for extending the domain of influence of Operations Research. Arguably, multi-agent optimization frameworks can help provide enhanced models by contemplating the interactions decision-makers often take by pondering the influence of other stakeholders (e.g., other players). Additionally, they can help embed socially-beneficial outcomes by enlightening the nature of interaction among selfish decision-makers. For instance, Carvalho et al. (2019) provide insights on the role of a carbon tax in competitive international energy markets, Carvalho et al. (2017) prove that the most rational outcome in their cross-border kidney exchange maximizes the social welfare (e.g., the sum of the objectives of all players). On the other hand, multi-agent models are as helpful as one can efficiently compute equilibria (or equivalent solution paradigms), thus highlighting the importance of theoretical and practical contributions for computing them. We believe that free and open-source software can foster experimentation in both practitioners’ and researchers’ communities, and hopefully lead to novel methodological advancements in the field.

1.1 Background.

In this context, we introduce ZERO, a modular C++ package to handle Reciprocally-Bilinear Games (RBGs), a special class of Mathematical Programming Games (MPGs). An MPG is a simultaneous game among \( n \) players, each of which solves a mathematical program whose objective function is parametrized in other players’ variables, and whose feasible region’s description does not include other players’ variables. Although MPGs are also Nash equilibrium problems (NEPs) (Facchinei and Pang 2003), the MPGs taxonomy we propose follows three assumptions: (i.) a set of constraints, for instance, a set of linear constraints and integer requirements, represent each player’s moves. This set may be unbounded, contain infinitely or finitely many elements, and generally does not have a special structure. We do not assume the players’ feasible sets to be continuous (i.e., in contrast to most of the NEPs literature), nor that computing equilibria necessarily requires the solution of a complementarity problem, (ii.) we aim to build a language intersecting both elements of Game Theory and Mathematical Programming, (iii.) we aim to preserve the structure that constraints give to each player’s problem. For instance, we may not drop any constraints to simplify the game without damaging its modeling capability. For the above three reasons, we introduce the class of MPGs to represent a wide variety of games among
optimization problems.

ZERO provides support for a fundamental class of MPGs, namely the class of RBGs. Let the operator \(( \cdot )^{-i}\) define \(( \cdot )\) except \(i\); e.g., if \(x = (x^1, \ldots, x^n)\), then \(x^{-2} = (x^1, x^3, \ldots, x^n)\).

**Definition 1 (Reciprocally-Bilinear Game (Carvalho et al. 2021))** A Reciprocally-Bilinear Game (RBG) is an MPG among \(n\) players, where each player \(i = 1, 2, \ldots, n\) solves the optimization problem

\[
\min_{x^i} f^i(x^i, x^{-i}) = (c^i)^	op x^i + (x^{-i})^	op C^i x^i
\]  

s.t. \(x^i \in X^i\) \hspace{1cm} (1a)

where \(X^i \subseteq \mathbb{R}^{m_i}\), and \(C\) and \(c\) are a matrix and a vector of appropriate dimensions, respectively. An RBG is polyhedrally-reprensentable if \(c_{\text{conv}}(X^i)\) is a polyhedron for each \(i\), and one can optimize a linear function over each \(X^i\).

In RBGs, the \(i\)-th player objective function \(f^i(x^i, x^{-i})\) – or payoff function for \(i\) – is linear in \(x^i\) and contains bilinear products with \(x^i\) and \(x^{-i}\). Further, since RBGs are MPGs, the description of each player’s feasible region \(X^i\) does not contain other players’ variables, and the \(i\)-th player optimization problem is parametrized in \(x^{-i}\), namely plugging \(x^{-i}\) as a parameter results in an optimization problem purely in the variables \(x^i\). When \(n = 1\), the RBG in Definition 1 is a single optimization problem in \(x^i\). Whenever \(n > 1\), RBGs become expressive models extending typical Operations Research tasks – such as resource allocation, scheduling, or routing – to a multi-agent setting. Consider, for instance, the emblematic 0/1 Knapsack Problem; given a set of items, a decision-maker selects some of them to maximize the sum of the profits associated with each item, subject to a capacity constraint. A multi-agent extension of this problem is the so-called Knapsack Game as in Example 1, where \(n\) players simultaneously solve a 0/1 Knapsack Problem.

**Example 1 (Knapsack Game)** A Knapsack Game is an RBG where each player solves the optimization problem

\[
\max_{x^i} \{ (c^i)^	op x^i + (x^{-i})^	op C^i x^i : (a^i)^	op x^i \leq b^i, x^i \in \{0, 1\}^{m_i} \} \hspace{1cm} (2)
\]

where \(m_i\) is the number of items for player \(i\), \(b^i \in \mathbb{Z}\), \(a^i \in \mathbb{Z}^{m_i}\), \(c^i \in \mathbb{Z}^{m_i}\), and \(C^i\) is an integer-valued matrix of appropriate size.

In this game, player \(i\) has not only to consider a feasible packing of items maximizing the profits associated with the vector \(c^i\), but has to look out for the positive or negative impact of the interaction of its packings with the ones of its opponents (the \(C^i\) products). Besides being an RBG, the Knapsack Game is also an Integer Programming Game (IPG), namely an MPG where each player solves a mixed-integer problem (Köppe et al. 2011). The sets \(c_{\text{conv}}(X^i)\) are the so-called integer hulls associated with each player’s 0/1 knapsack polytope, and
each point $x^i \in \mathcal{X}^i$ is a pure-strategy for $i$, namely a solution to the knapsack problem for $i$. In general, each $\sigma^i \in \text{cl conv}(\mathcal{X}^i)$ is a mixed-strategy, namely a point inside the 0/1 knapsack polytope. The central question is then to determine what is a solution to the above game. In an optimization problem, we usually search for an optimal solution that maximizes (minimizes) the objective function while fulfilling the constraints. However, in a game, a solution should be stable, meaning that it should be mutually optimal for all the players, and not only a subset of them. The most famous solution paradigm in Game Theory is the one of Nash Equilibrium, a solution where each player cannot unilaterally deviate from it while improving its payoff. We formally define the concept of Nash equilibrium for RBGs in Definition 2; we remark that in Definition 1 players are minimizing their objective functions, and improving a payoff means decreasing it.

**Definition 2 (Pure-Strategy Nash Equilibrium)** A strategy profile $\pi = (x^1, \ldots, x^n)$ is a Pure-Strategy Nash Equilibrium for an RBG as in Definition 1 if, for each player $i$ and strategy $\bar{x}^i \in \mathcal{X}^i$, then $f^i(x^i, x^{-i}) \leq f^i(\bar{x}^i, x^{-i})$.

In other words, at the equilibrium $\pi = (x^1, \ldots, x^n)$, no player $i$ can possibly pick a strategy $\bar{x}^i \neq x^i$ so that $f^i(x^i, x^{-i}) > f^i(\bar{x}^i, x^{-i})$. In this sense, the equilibrium strategy is resilient to the moves of each player’s opponents and provides a mutually-optimal solution. The Mixed-Strategy Nash equilibrium relaxes the definition of Pure-Strategy Nash equilibrium by allowing players to select not only pure-strategies, but in general mixed-strategies.

## 2 Our Contributions

ZERO provides advanced and modular C++ toolkits to formulate RBGs and compute their Nash equilibria, with high-level APIs for practitioners and low-level ones for researchers and experienced users. We summarize the most important contributions as follows.

(i.) ZERO is the first library to support non-cooperative simultaneous games where players solve mathematical programs. Other Game Theory solvers, such as Gambit (McKelvey et al. 2006) only support finite games in normal form (games with finitely many players, finitely many strategies and outcomes).

(ii.) The library has a modular structure designed for allowing extensibility. Each component – or module – independently performs a specific task and interacts with the others through well-defined interfaces. For instance, the natively embedded algorithms interface with the base modules allowing the development of sophisticated computational routines. Users can either use the included algorithms or implement custom ones depending on the desired level of control.
(iii.) The library is an abstract layer bridging typical Mathematical Programming and Game Theory and focuses on the interaction and orchestration among external libraries and native modules. We delegate most of the standard mathematical programming routines to specialized software, thus integrating popular and well-maintained tools available in the Operations Research community. For instance, we solve mathematical programs through Gurobi (Gurobi Optimization, LLC 2021) and PATH (Ferris and Munson 1999), we generate cutting planes with Coin-OR Cgl (Lounge-Heimer 2003), and we perform linear algebra operations through Armadillo (Sanderson and Curtin 2016).

(iv.) ZERO can work as an off-the-shelf solver for RBGs without the need for a deep technical understanding of the algorithmic details. We provide a series of high-level interfaces designed specifically for some classes of RBGs, along with standardized instance file schemes and plug-and-play shell executables. On the one side, ZERO provides high-level APIs for practitioners and industrial parties to experiment with our high-level APIs. On the other side, we target experienced users by offering advanced tools to build sophisticated models and algorithms.

3 Overview

We briefly give an overview of ZERO: the detailed documentation for the software is available online at www.getzero.one. Our library currently supports any polyhedrally-representable RBG, and further provides additional tools (i.e., high-level modeling APIs) for two specific types of games. First, IPGs, namely MPG where each player solves an integer program; in particular, ZERO supports IPGs that are also RBGs, and hence have a bilinear objective as in Definition 1. Second, Nash games Among Stackelberg Players (NASPs), a class of Equilibrium Problems with Equilibrium Constraints among the leaders of continuous bilevel games (Carvalho et al. 2019).

Modules and Namespaces. ZERO’s modules are classes defined inside a suitable namespace, namely a larger scope grouping modules with similar functions or goals. In the sequel, we provide an overview of the software architecture. The namespace MathOpt contains the necessary optimization tools for defining and solving mathematical programs – for instance, MathOpt::IP_Param for parametrized mixed-integer linear programs, and MathOpt::LCP for linear complementarity problems (LCPs) – as well as helper functions (e.g., MathOpt::convexHull for computing the convex hull of a union of polyhedra). This class provides a layer between ZERO and the external solvers such as Gurobi and PATH. Arguably, the most relevant namespace is the one of Games, which implements the abstraction of specific RBGs, such as Games::IPG for IPGs, and Games::EPEC for NASPs. The modules inside this namespace orchestrate a tight integration among all the other modules and provide several
low-level APIs to the user. The namespace **Algorithms** contains the algorithms to compute the Nash equilibria for **RBGs**. Such algorithms are inside the modules of this namespace and closely coordinate with the modules in **Games**; for instance, the class **Algorithms::IPG::CutAndPlay** associated with the Cut-And-Play algorithm for **IPGs** and **NASPs** (Carvalho et al. 2021) coordinates with both **Games::EPEC** and **Games::IPG**. Other than advanced users, **ZERO** aims to target practitioners that may only be interested in plug-and-play usage of the software. Thus, in the namespace **Models** we provide high-level APIs allowing users to quickly model and solve off-the-shelves instances of **IPGs** and **NASPs**. Furthermore, we propose a standardized format for instances encoded through the data-interchange format **JSON** (Pezoa et al. 2016), and integrate complementary helper functions to manage the input and output files. We also include two shell executables working with standardized instance formats allowing users to deploy the algorithms and solve instances on the run. Finally, the namespace **Utils** provides some simple helper functions for writing and reading files, as well as additional numerical and linear algebra utilities. **Figure 1** provides a schematic representation of the architecture.

**Figure 1:** A schematic view of **ZERO**’s modules, 10000 lines of code, 50 files, 40 classes, and 450 functions. The **namespaces** are in gray, and the relative content is grouped below. The primitive classes are in purple, and the associated inheritor classes are in blue. Nested namespaces are in green.

### 4 Modeling the Knapsack Game

We showcase how to model an instance of the Knapsack Game of **Example 1** with **ZERO**. Let **blue** be Player 1 and **red** be Player 2. Each player $i$ seeks to pack $m_i = 2$ items into its knapsack with capacity $b^i = 5$. The optimization
problems for blue and red are in (3) and (4), respectively.

\[
\begin{align*}
\text{(3a)} \quad & \max_{x^1} x^1_1 + 2x^1_2 - 2x^1_1 x^2_1 - 3x^1_1 x^2_2 \\
\text{s.t.} \quad & 3x^1_1 + 4x^1_2 \leq 5, x^1 \in \{0,1\}^2
\end{align*}
\]

\[
\begin{align*}
\text{(4a)} \quad & \max_{x^2} 3x^2_1 + 5x^2_2 - 5x^2_1 x^1_1 - 4x^2_1 x^2_2 \\
\text{s.t.} \quad & 2x^2_1 + 5x^2_2 \leq 5, x^2 \in \{0,1\}^2
\end{align*}
\]

This problem has 3 Nash equilibria: the Pure-Strategy Nash equilibria \((x^1_1, x^1_2, x^1_1^2, x^1_1^2) = (0, 1, 1, 0), (x^1_1, x^1_2, x^1_1^2, x^1_1^2) = (1, 0, 0, 1)\), and the Mixed-Strategy Nash equilibrium \((x^1_1, x^1_2, x^1_1^2, x^1_1^2) = (\frac{2}{5}, \frac{3}{5}, \frac{3}{5}, \frac{2}{5})\). We attempt to find one of them by using the \textit{Cut-And-Play} algorithm from (Carvalho et al. 2021). Intuitively, this algorithm iteratively refines each players’ feasible region starting from its linear relaxations (i.e., the polyhedron given by dropping the integrality constraint in either (3) or (4)). Specifically, the algorithm iteratively refines the linear relaxations adding cutting planes (some of which generated by Cgl from Coin OR (Lougee-Heimer 2003)) or by branching until it finds a Nash equilibrium.

**Modeling and solving with ZERO.** Figure 2 demonstrate the use of our high-level API for \textit{IPGs} by modeling the Knapsack Game in (3) and (4). We start by including the only header file \texttt{zero.h} in Line 1 – which contains the specifications for the entire library – and by creating a new Gurobi environment in Line 4. In Line 5 we create a new empty \textit{IPG} instance (\texttt{Models::IPG::IPGInstance}), which we will later populate with the programs in (3) and (4). From Line 7 to Line 12, we create the objects holding the data for the integer programs, for instance, the vector \texttt{a} for the knapsack constraint and the vector \texttt{IntegerIndexes} containing the indices of the integer-constrained variables. We fill in the data from (3) from Line 14 to Line 21, and create the (parametrized) integer program for player \textit{blue} in Line 24 with a constructor of \texttt{MathOpt::IP_Param}. The latter class infers the number of parameters – namely the number other players variables – by counting the number of rows of \(C^1\); in this case, the parameters are 2, and they are associated to the choices of Player 2. From Line 26 to Line 29, we iterate this data-filling process for \textit{red}, and eventually add the two parametrized integer programs to the \textit{IPG Instance} in Lines 32 and 33. In Line 34, we save the instance with the standardized data format for ZERO instances. The solution process starts from line Line 35, where we instantiate – in the object \texttt{KnapsackGame} – an \textit{IPG} model with the data contained in \texttt{IPG Instance}. We employ the constructor of \texttt{Models::IPG::IPG} by also specifying a pointer to the Gurobi environment. In Line 37, we instruct ZERO to use the \textit{Cut-And-Play} algorithm to solve \texttt{KnapsackGame}. In Lines 38 and 43, we set some extra options, and finally start computing the Nash equilibria in Line 45 by calling the method \texttt{Models::IPG::IPG::findNashEq()}. We print the Nash equilibrium found by the \textit{Cut-And-Play} in Lines 46 and 47.
5 Conclusions and Future Directions

We introduced ZERO, a multi-purpose C++ library offering the base ingredients to help users model and solve RBGs. On the one side, ZERO implements high-level and intuitive APIs to formulate RBGs and solve them. On the other side, its modular and extensive design enables advanced users and researchers to build customized algorithms. A current limitation of ZERO is the availability of only two mathematical programming solvers. We plan to extend further the support for other solvers, such as SCIP (Gamrath et al. 2020). Furthermore, we believe future methodological advancements will likely enable us to extend our support to other classes of MPG s and RBGs. Naturally, this is conditional to the development of the appropriate mathematical tools to do so. Indeed, we release ZERO with the ambition to foster methodological and applied research in this newly developing field at the intersection of Game Theory and Mathematical Programming.

Acknowledgements

This work has been supported by the Canada Excellence Research Chair in Real-time Decision-making (DS4DM).

References

Amitabh Basu, Christopher Thomas Ryan, and Sriram Sankaranarayanan. Mixed-integer bilevel representability. Mathematical Programming, August 2019. ISSN 1436-4646. doi: 10.1007/s10107-019-01424-w. (Cited on page 2.)

Luce Brotcorne, Martine Labbé, Patrice Marcotte, and Gilles Savard. A Bilevel Model for Toll Optimization on a Multicommodity Transportation Network. Transportation Science, 35(4):345–358, 2001. doi: 10.1287/trsc.35.4.345.10433. URL https://doi.org/10.1287/trsc.35.4.345.10433. (Cited on page 2.)

Alberto Caprara, Margarida Carvalho, Andrea Lodi, and Gerhard J. Woeginger. Bilevel Knapsack with interdiction constraints. INFORMS Journal on Computing, 28(2):319–333, 2016. ISSN 15265528. doi: 10.1287/ijoc.2015.0676. (Cited on page 2.)

Margarida Carvalho, Andrea Lodi, João Pedro Pedroso, and Ana Viana. Nash Equilibria in the two-player Kidney Exchange Game. Mathematical Programming, 161(1-2):389–417, January 2017. ISSN 0025-5610, 1436-4646. doi: 10.1007/s10107-016-1013-7. URL http://link.springer.com/10.1007/s10107-016-1013-7. (Cited on page 2.)

Margarida Carvalho, Andrea Lodi, and João Pedro Pedroso. Existence of Nash Equilibria on Integer Programming Games. In A. Ismael F. Vaz, João Paulo Almeida, José Fernando Oliveira, and Alberto Adrego Pinto, editors, Operational Research, volume 223, pages 11–23. Springer International Publishing, Cham, 2018. ISBN 978-3-319-71582-7 978-3-319-71583-4. doi: 10.1007/978-3-319-71583-4_2. URL http://link.springer.com/10.1007/978-3-319-71583-4_2. Series Title: Springer Proceedings in Mathematics & Statistics. (Cited on page 2.)
Margarida Carvalho, Gabriele Dragotto, Felipe Feijoo, Andrea Lodi, and Sriram Sankaranarayanan. When Nash Meets Stackelberg. arXiv, 2019. URL https://arxiv.org/abs/1910.06452. (Cited on pages 2 and 5.)

Margarida Carvalho, Andrea Lodi, and Joao Pedro Pedroso. Computing Nash Equilibria for Integer Programming Games. arXiv preprint arXiv:2012.07082, 2020. URL https://arxiv.org/abs/2012.07082. (Cited on page 2.)

Margarida Carvalho, Gabriele Dragotto, Andrea Lodi, and Sriram Sankaranarayanan. The Cut and Play Algorithm: Computing Nash Equilibria via Outer Approximations. arXiv, abs/2111.05726, 2021. URL https://arxiv.org/abs/2111.05726. (Cited on pages 2, 3, 6, and 7.)

Tobias Cronert and Stefan Minner. Equilibrium Identification and Selection in Integer Programming Games. SSRN Preprint 3762380, page 38, 2021. doi: http://dx.doi.org/10.2139/ssrn.3762380. (Cited on page 2.)

S. T. DeNegre and T. K. Ralphs. A Branch-and-cut Algorithm for Integer Bilevel Linear Programs. In John W. Chinneck, Bjarni Kristjansson, and Matthew J. Saltzman, editors, Operations Research and Cyber-Infrastructure, pages 65–78. Springer US, Boston, MA, 2009. ISBN 978-0-387-88842-2 978-0-387-88843-9. doi: 10.1007/978-0-387-88843-9_4. URL http://link.springer.com/10.1007/978-0-387-88843-9_4. (Cited on page 2.)

Gabriele Dragotto and Rosario Scatamacchia. ZERO Regrets Algorithm: Optimizing over Pure Nash Equilibria via Integer Programming. arXiv, abs/2111.06382, 2021. URL https://arxiv.org/abs/2111.06382. (Cited on page 2.)

Francisco Facchinei and Jong-Shi Pang. Finite-dimensional variational inequalities and complementarity problems. Springer series in operations research. Springer, New York, 2003. ISBN 978-0-387-95580-3 978-0-387-95581-0. (Cited on page 2.)

Felipe Feijoo and Tapas K. Das. Emissions control via carbon policies and microgrid generation: A bilevel model and Pareto analysis. Energy, 90:1545–1555, 2015. ISSN 0360-5442. doi: https://doi.org/10.1016/j.energy.2015.06.110. URL https://www.sciencedirect.com/science/article/pii/S036054421500866X. (Cited on page 2.)

Michael C. Ferris and Todd S. Munson. Interfaces to PATH 3.0: Design, Implementation and Usage. Computational Optimization and Applications, 12(1/3):207–227, 1999. ISSN 09266003. doi: 10.1023/A:1008636318275. URL http://link.springer.com/10.1023/A:1008636318275. (Cited on page 5.)

Matteo Fischetti, Ivana Ljubić, Michele Monaci, and Markus Sinnl. A New General-Purpose Algorithm for Mixed-Integer Bilevel Linear Programs. Operations Research, 65(6):1615–1637, December 2017. ISSN 0030-364X, 1526-5463. doi: 10.1287/opre.2017.1650. URL http://pubsonline.informs.org/doi/10.1287/opre.2017.1650. (Cited on page 2.)

Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. ZIB-Report 20-10, Zuse Institute Berlin, March 2020. (Cited on page 8.)
Conrad Sanderson and Ryan Curtin. Armadillo: a template-based C++ library for linear algebra. *Journal of Open Source Software*, 1(2):26, 2016. (Cited on page 5.)
```cpp
#include <zero.h>

int main(int argc, char **argv) {

    int numItems = 2, numPlayers = 2;

    arma::vec c(numItems);  // Profits c in the objective
    arma::sp_mat C((numPlayers-1) * numItems, numItems);  // C terms in the objective
    arma::vec b(1);  // RHS for constraints
    arma::vec IntegerIndexes(numItems);  // The index of the integer variables
    VariableBounds VarBounds = {{0, 1}, {0, 1}};  // Implicit bounds (LB,UB) on variables.

    b(0) = 5;  // Knapsack Capacity
    for(unsigned int i = 0; i < numItems; ++i)
        IntegerIndexes.at(i) = i;

    C(0, 0) = 2;  C(1, 1) = 3;  // C terms in the objective for player Blue
    a(0, 0) = 3; a(0, 1) = 4;  // Knapsack Constraints
    c(-1);  // The standard is minimization, hence minus

    // Create a parametrized Integer Program for player Blue
    MathOpt::IP_Param PlayerBlue(C, a, b, c, IntegerIndexes, VarBounds, &GurobiEnv);

    // Create a parametrized Integer Program for player Red
    C(0, 0) = 5; C(1, 1) = 4; a(0, 0) = 2; a(0, 1) = 5; c(-3); c(-5);

    MathOpt::IP_Param PlayerRed(C, a, b, c, IntegerIndexes, VarBounds, &GurobiEnv);

    // Add the players to the instance. We can also specify a file path to write the instance
    IPG_Instance.addIPParam(PlayerBlue, "PlayerBlue_KP");
    IPG_Instance.addIPParam(PlayerRed, "PlayerRed_KP");
    IPG_Instance.save("A_Knapsack_Game");  // Save the instance with the standardize format

    Models::IPG::IPGKnapsackGame(&GurobiEnv, IPG_Instance);  // Create a model from the instance

    // Select the equilibrium to compute a Nash Equilibrium
    KnapsackGame.setAlgorithm(Data::IPG::Algorithms::CutAndPlay);

    // A few optional settings
    KnapsackGame.setDeviationTolerance(3e-4);  // Numerical tolerance
    KnapsackGame.setNumThreads(4);  // How many threads, if supported by the solver?
    KnapsackGame.setLCPAlgorithm(Data::LCP::Algorithms::MIP);  // How do we solve the LCPs?
    KnapsackGame.setTimeLimit(5);  // Time limit in second
    KnapsackGame.finalize();  // Lock the model

    // Run and get the results
    KnapsackGame.findNashEq();

    KnapsackGame.getX().at(0).print("Player Blue:");  // Print the solution
    KnapsackGame.getX().at(1).print("Player Red: ");
}
```

Figure 2: An Example of a C++ instantiation of a 2-player Knapsack Game in ZERO.