Python-MIP COIN-OR Cup 2019 Submission

Santos, H.G. and Toffolo, T.A.M.

September 20, 2019

1 Introduction

The `Python-MIP` module is a comprehensive collection of tools for the modelling and solution of Mixed-Integer Linear Programs in Python. The following design goals were considered in the creation of this package: (1) clear, high-level modelling; (2) performance and (3) extensibility and configurability. Traditionally, some of these goals have been considered conflicting. High-level languages languages such as `AMPL`[4] were usually the best choice for the rapid development of models that didn't require advanced configuration and interaction with the solver engine and low level languages like `C` were the best option for those who wanted to achieve maximum performance. By performance gains we consider here both improvements in the model creation times but also, and most importantly, the ones that can be obtained in solution times with a deeper integration with the solver engine (bi-directional communication) during the solution process to guide it. Cut generation is one of example of these features. Since state-of-the-art solvers such as `CPLEX`® were usually developed using in `C`[2], the access to all those advanced features was often restricted to this language.

Recently, frameworks like `JuMP`[3] showed that these goals are not necessarily conflicting. The availability of highly expressive languages such as `Julia` and fast just-in-time compilers can be used to quickly develop efficient optimization codes in a convenient high-level language. Thus, `JuMP` scores quite well w.r.t. goals 1-3. The objective of the `Python-MIP` project is the development of a tool that excels in goals 1-3 using the Python programming language. Python is becoming the most popular[1] programming language according to recent surveys. Part of this success is due to the availability of a very large number of easily accessible third-party packages ($\approx 200,000$ currently) with various functionalities. Data science and machine learning are some of the prominent uses of Python currently. Thus, a Python package that enables the fast development of high performance solvers can benefit from this ecosystem rich in data processing and analysis solutions to speedup the development of effective decision making tools. In the next subsections we'll discuss some characteristics of `Python-MIP` w.r.t. items 1-3.

1.1 Clear, High Level Modelling

Python naturally provides convenient data structures such as sets and dictionaries. In Python-MIP the expression of linear constraints can be conveniently stated using operators over objects in these data structures. If N is a set of cities and A is a dictionary that maps arcs (i, j) with their respective distances, then the Traveling Salesman Person (TSP) formulation[5] can be stated in 10 lines of code:

```
m = Model()
n, n0 = len(N), min(N)
x = {a: model.add_var(var_type=BINARY) for a in A.keys()}
y = {i: model.add_var() for i in N}
m.objective = minimize(xsum(A[a]*x[a] for a in A.keys()))
for i in N:
    m += xsum(x[a] for a in A if a[0]==i) == 1
    m += xsum(x[a] for a in A if a[1]==i) == 1
for (i, j) in [a for a in A.keys() if n0 not in [a[0], a[1]]]:
    m += y[i] - (n+1)*x[(i, j)] >= y[j]-n
```

1.2 Performance

From the beginning, Python-MIP was written in modern, statically typed Python to be fully compatible with the high performance Just-In-Time compiler PyPy. To provide a deep integration with the supported solver engines, Python-MIP uses CFFI[1] to communicate directly with native dynamic loadable libraries. Some of the advantages of this choice over Cython are: (i) C code can be included directly in Python modules, (ii) no recompilation is needed if a new version of solver engine library is released and (iii) PyPy is specially optimized[2] to work with it. One shortcoming of this approach is that only C functions can be called. For Gurobi® this was not a problem, but for CBC it was: CBC is written in C++ and only a subset of functionalities was available in C in 2018. Thus, in the previous months, we expanded and improved the CBC C API adding several advanced features such as cut callbacks, lazy constraints and an incumbent solution callback where the solution in terms of the original variables (not the pre-processed ones) can be queried. Furthermore, to speedup the creation of models, we implemented a module to buffer successive calls for problem modifications. Some of these features are not available in the C++ API. With our additions, we believe that now that are many cases where it is even more convenient to use the CBC C API than the C++ one. As for Gurobi®, using Python-MIP to create MIP models for it can be up to *25 times faster* than using the official Gurobi® Python interface. As previously mentioned, the largest performance gains are usually gains in the solution times. Python-MIP eases the implementation of many alternative formulations for the same combinatorial optimization problem. Dramatic performance gains can be observed executing the TSP solver

[1] The C Foreign Function Interface for Python
[2] https://pypy.org/compat.html

example with the compact formulation and the one that uses a branch-&-cut with the sub-tour elimination constraints.

1.3 Extensibility and configurability

While the performance of standalone MIP solvers is continuously improving, it is often the case that the solution times obtained simply by feeding the MIP model to the MIP solver and querying the results is poor. The best MIP formulations often have an exponential number of constraints and handling them requires bi-directional communication with the solver engine to include them on-demand, during the tree search. `Python-MIP` has solver independent callbacks: cut generators and lazy constraints are available. The integration with problem dependent heuristics is also quite easy using MIPStarts and the incumbent callback can be used to improve the performance in the production of high quality, feasible solutions.

2 Final remarks

The first commit in `Python-MIP` was less than a year ago. We believe that an impressive number of well documented and tested features is already implemented. Our plan is keep improving Python-MIP by adding new features and performance improvements. Our objective is to include a large number of features and keep them supported on all solvers supported by Python-MIP. To make it feasible, we do not plan to support a large number of solvers, our priority is the open source solver `CBC` and one or two state-of-the-art commercial solvers. The complete `Python-MIP` documentation can be accessed at the readthedocs website.

References

[1] Python has brought computer programming to a vast new audience. *The Economist*, July 2018.

[2] Robert E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15, 2002.

[3] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A Modeling Language for Mathematical Optimization. *SIAM Review*, 59(2):295–320, 2017.

[4] Robert Fourer, David Gay, and Brian Kernighan. *AMPL: A mathematical programming language*. AT & T Bell Laboratories, 1987.

[5] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.