Python-MIP – COIN-OR Cup 2019 Submission

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

September 20, 2019

1 Introduction

Python-MIP is a comprehensive collection of tools for modeling and solving Mixed-Integer Linear Programs (MIP) using Python. The package was designed with the following goals in mind:

(1) clarity - it should enable clear and easy high-level modeling;

(2) performance;

(3) extensibility and configurability.

Traditionally, some of these goals have been considered conflicting. On the one hand, 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. On the other hand, 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 improvements in the *model creation times* but also, and most importantly, the ones that can be obtained in *solution times* using features that enable a deeper integration with the solver engine (bi-directional communication) during the solution process. Cuts generation is an example of these features. Since state-of-the-art solvers such as CPLEX® are generally developed in the C language [2], the access to more advanced features was often only available to those using this language.

Recently, frameworks like JuMP [3] showed that goals (1)-(3) 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 to provide a tool that excels in goals (1)-(3) using the Python programming language.

Python is becoming the most popular programming language according to recent surveys [1]. 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. In this context, a Python package that enables the fast implementation of MIP models 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 discuss some characteristics of Python-MIP w.r.t. items (1)-(3).

1.1 Clarity

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. Let N be a set of cities (nodes) and A a dictionary that maps arcs (i, j), for all $i, j \in N$, to their respective distances. The Traveling Salesman Person (TSP) formulation [5] can be stated in only 10 lines of code using Python-MIP:

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

1.2 Performance

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 alternative options like Cython are: (*i*) C code can be included directly within Python modules, (*ii*) no recompilation is needed if a new version of a solver 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. While for Gurobi® this was not a problem, CBC is written in C++ and only a subset of functionalities was available in C in 2018. Therefore, 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 in which the the CBC C API is even more convenient to use than the C++ one. As for Gurobi®, using Python-MIP and PyPy to create MIP models can be up to *25 times faster* than using the official Gurobi® Python interface. Moreover, Python-MIP eases the implementation of alternative formulations for the same combinatorial optimization problem. This can result in dramatic performance gains, such as the one observed when solving the TSP with branch-&-cut with the sub-tour elimination constraints instead of using a compact formulation.

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

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 (or variables) and handling them requires bi-directional communication with the solver engine to include them on-demand. `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 initial commit in `Python-MIP` was less than a year ago. Nevertheless, we believe that an impressive number of well documented and tested features is already implemented. Our plan is to keep improving Python-MIP by adding new features and performance improvements. Furthermore, our objective is to keep and support all these features for multiple solvers. To make this objective feasible, we do not plan to support a large number of solvers. In fact, our priority is the open source solver `CBC` and one or two state-of-the-art commercial solvers.

For more information, the complete `Python-MIP` code and documentation can be accessed at the package's 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.