

MOS: A Mathematical Optimization Service

James Hubert Merrick*, Tomás Tinoco De Rubira†

October 11, 2022

Abstract

We introduce MOS, a software application designed to facilitate the deployment, integration, management, and analysis of mathematical optimization models. MOS approaches mathematical optimization at a higher level of abstraction than existing optimization modeling systems, enabling its use with all of them. The sole requirement to harness MOS is a simple annotation of the code specifying the formulation of an optimization model. With this, the model becomes accessible to humans through the automatic generation of a user interface, and to machines through an associated API and client libraries. All this is achieved while avoiding the ad hoc code typically required to obtain such features.

1 Introduction

Whilst known by different names in different settings, mathematical optimization influences billions of dollars in the modern economy, impacting every industry. It consists of the maximization or minimization of some objective function subject to constraints, and is a natural paradigm for solving problems in many fields. In planning applications in operations research and management science, optimization models act as a decision-support tool for a human decision-maker. In other, more operational, settings, they enable decisions to be automated, with example problems including pricing, scheduling, allocation of scarce resources, and routing. In the natural sciences, optimization models capture the physical laws governing the behavior of natural systems. In machine learning, they provide the tools for obtaining parameterized models that best fit a particular data set. [Boyd and Vandenberghe \(2004\)](#) and [Luenberger and Ye \(2021\)](#) provide comprehensive introductions to the theory and applications of mathematical optimization.

Key tools in mathematical optimization are algebraic modeling systems. Examples of these are CVXPY ([Diamond and Boyd, 2016](#)), JUMP ([Dunning et al., 2017](#)), PYOMO ([Hart et al., 2017](#)), OPTMOD ([Tinoco De Rubira, 2020](#)) and GAMS ([Bussieck and Meeraus, 2004](#)). These tools greatly facilitate the process of constructing and solving optimization models on computers. They allow users to construct optimization problems by writing intuitive mathematical expressions, and can utilize many numerical solvers without the need of custom code that expresses the problem in solver-specific data structures and formats.

*jmerrick@alumni.stanford.edu

†ttinoco@alumni.stanford.edu

From the authors’ experience in developing optimization models to support and automate decisions, once a model is formulated using one of the above modeling systems, there is often an additional non-trivial programming exercise required to facilitate a human or application to interact with the model. In the absence of this code, using the model requires familiarity with the model’s internal and low-level details, which is seldom documented and user friendly, creating barriers for human users and adding complexity to application pipelines. A custom solution, on the other hand, typically requires time and resources, including dedicated software engineers.

By approaching the modeling problem at a higher level of abstraction than existing tools, and by capturing and standardizing common model properties and structure, MOS provides essential deployment, integration, management and analysis features automatically, removing the need for custom solutions. The sole requirement is a simple annotation of the file containing the model code. This allows a focus, at the development stage, on the core modeling task itself, and at the production stage, on the model usage itself, reducing the barriers to obtaining value from a model.

Guericke and Cassioli (2019) propose a framework for deploying optimization models based on microservice architectures, and highlight a gap between solution methods in literature and solution methods in production environments. MOS also attempts to contribute to the closing of this gap through a proposed concrete and universal model representation, a modular and flexible architecture, and an open-source implementation.¹

2 Model representation

Boyd and Vandenberghe (2004) introduce an optimization problem as being represented by the following:

$$\begin{aligned} & \text{minimize} && f_0(x) \\ & \text{subject to} && f_i(x) \leq b_i, i = 1, \dots, m, \end{aligned} \tag{1}$$

where $x = (x_1, \dots, x_n)$ is the vector of variables to be optimized, $f_0 : \mathbf{R}^n \rightarrow \mathbf{R}$ is the objective function, and functions $f_i : \mathbf{R}^n \rightarrow \mathbf{R}$ together with constants b_i for $i = 1, \dots, m$ define the constraints. MOS considers optimization models as objects that not only consist of optimization problems having variables, functions, and constraints, as in (1), but also of inputs, outputs, and intermediate objects. The intermediate or “helper objects” correspond to objects that are created either in a pre-optimization phase, for facilitating the construction of problem variables, functions and constraints, or in a post-optimization phase, for facilitating the construction of outputs. This model representation is helpful for establishing a layer of abstraction that enables the definition and implementation of tools for interacting with, and analyzing models. Figure 1 illustrates the MOS optimization model representation.

3 Design

Figure 2 shows the architecture of MOS, which includes the following components:

- Backend: Manages model data and access, and provides a REST API for interacting with models.

¹MOS is available at <https://github.com/Fuinn>.

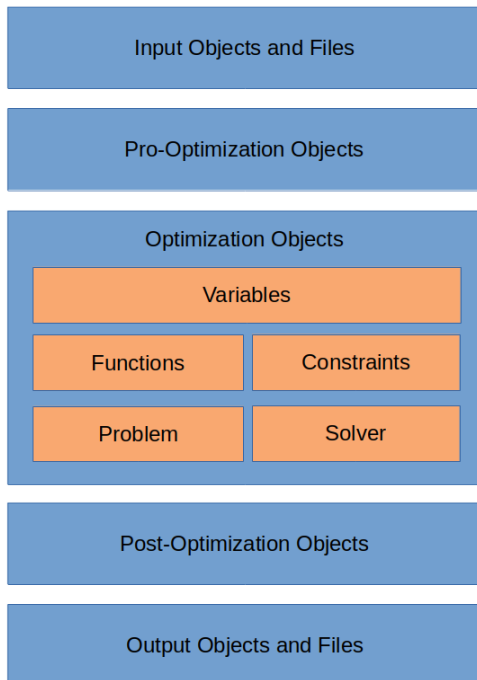


Figure 1: MOS model representation

- Frontend: Provides a graphical user interface for accessing, using, and analyzing models.
- Interface libraries: Allow users or other applications to interact with models using popular programming languages and integrate them with application pipelines.
- Compute workers: Run models locally or distributed over the network using modeling-system-specific computational kernels.

MOS accepts an optimization model file, in any of the supported programming languages and modeling systems, with annotations that identify the components of the model, described in the previous section. Appendix A shows an extract from an annotated model file, illustrating the nature of the annotation, requiring labeling of different sections of the code with certain structural keywords marking out common model features, preceded by a language-specific tag. The degree of structure required by MOS in annotation is variable and optional, with more structure enabling more features in the interface. Appendix B shows an example use of the Python interface library. Appendix C shows screenshots of the MOS user interface.

4 Benefits

MOS enables an appropriately annotated optimization model to have the following features:

- *Deployable*: Readily deployable in the cloud or on an organization’s own servers.

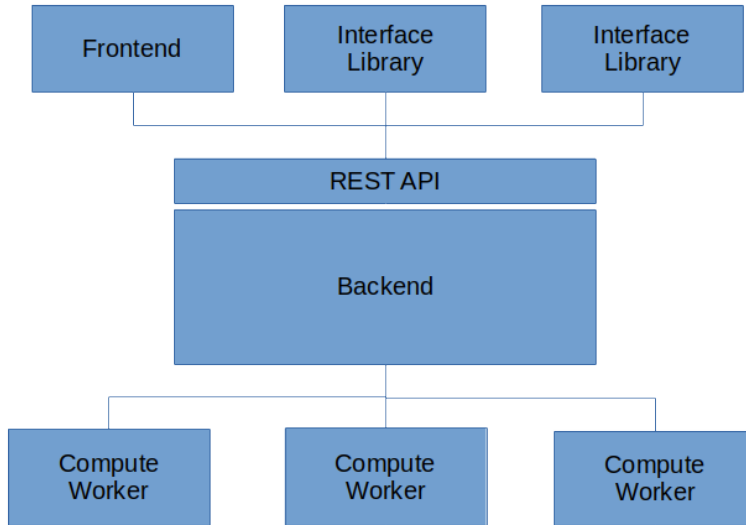


Figure 2: Architecture of MOS

- *API Access:* Through automatic generation of an API through which an optimization model may be called, MOS facilitates integration of mathematical optimization into an organization’s data flows and software applications. For example, MOS may be used as a microservice by enterprise applications to integrate optimization capabilities. This integrability also enables a modular type of development, suitable for scaling as an optimization model is updated, without requiring a re-write of supporting custom infrastructure code.
- *A browsable, graphical, intuitive, representation of a model, associated data, and results:* The model structure, documentation, input data, and model results, may be browsed through, increasing model transparency and understanding. Model data and assumptions may also be changed through the user interface, and the model solved by clicking a button instead of doing so from the command line.
- *Leverage of existing optimization technologies and investments:* MOS works with a range of domain-specific languages designed for optimization such as CVXPY, JUMP, and GAMS, not tying an organization to one specific approach as needs evolve, and additionally leveraging an organization’s previous investments in optimization model code development.

While a well designed custom interface may provide many of these features, and they are commonly deployed by organizations, MOS provides these features automatically, while potentially providing a platform for further benefits. MOS is readily extendible to incorporate logs of model history, and automated analysis of model solutions.

5 Conclusion

Mathematical optimization is used across all industries, and the number of applications are growing as more digital data is available and the cost of computing declines. MOS allows the avoidance of

expensive development costs associated with supporting custom infrastructure code around an optimization model. It does this by approaching the modelling problem at a higher level of abstraction than existing tools.

References

- Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, Cambridge.
- Bussieck, M. R. and Meeraus, A. (2004). General Algebraic Modeling System (GAMS). In *Modeling Languages in Mathematical Optimization*, pages 137–157. Springer.
- Diamond, S. and Boyd, S. (2016). CVXPY: A Python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 17(1):2909–2913.
- Dunning, I., Huchette, J., and Lubin, M. (2017). JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320.
- Guericke, S. and Cassioli, A. (2019). A Framework for Mathematical Optimization in Microservice Architectures. *Optimization Online*.
- Hart, W. E., Laird, C. D., Watson, J.-P., Woodruff, D. L., Hackebeil, G. A., Nicholson, B. L., Siirola, J. D., et al. (2017). *Pyomo - Optimization Modeling in Python*. Springer, 2nd edition.
- Luenberger, D. G. and Ye, Y. (2021). *Linear and Nonlinear Programming*. Springer, New York, 5th edition.
- Tinoco De Rubira, T. (2020). OPTMOD 0.0.1. <https://github.com/ttinoco/OPTMOD>.

A Extract of annotated model file

```
#@ Constraint: P_limits
#@ Description: Generator active power limits
P_limits = []
for gen in network.generators:
    P_limits.extend([P[gen.index] >= gen.P_min, P[gen.index] <= gen.P_max])

#@ Objective: gen_cost_obj
gen_cost_obj = optmod.minimize(gen_cost)

#@ Problem: problem
problem = optmod.Problem(gen_cost_obj,
                        constraints=power_balance+angle_ref+P_limits)

#@ Solver: solver
solver = optalg.opt_solver.OptSolverINLP()
solver.set_parameters('feastol': feastol, 'maxiter': 100)

#@ Execution: info
info = problem.solve(solver)

#@ Output Object: output_obj
output_obj = list(info.values())
```

B Example use of MOS Python Interface

```
from mos.interface import Interface

interface = Interface(url, token)

model = interface.get_model_with_name('DCOPF Model')

model.set_interface_object('feastol', 1e-3)
model.set_interface_file(case, 'ieee14.m')

model.show_recipe()
model.show_components()
model.run()

model.get_status()
model.get_execution_log()
```

C MOS user interface screenshots

Name	Id	System	Factored	Status	Description	
DCOPF Model	42	optmod	true	created	Sample DCOPF model in OPTMOD	Delete
Portfolio Model	43	cvxpy	true	created	Sample portfolio optimization model in CVXPY	Delete
Transportation Model	44	gams	true	created	Sample transportation model in GAMS	Delete
Knapsack Model	45	jump	true	created	Sample knapsack model in JuMP	Delete

Figure 3: MOS user interface: list of models available

Name	Id	Description	
P_limits	3	Generator active power limits	Delete
angle_ref	2	Voltage angle reference	Delete
power_balance	1	Active power balance constraints	Delete

Figure 4: MOS user interface: browsing through model constraints