GBOML: Graph-Based Optimization Modeling Language

Bardhyl Miftari[*1], **Mathias Berger**[†1], **Hatim Djelassi**[1], **and Damien Ernst**[1, 2]

1 Department of Electrical Engineering and Computer Science, University of Liège **2** LTCI, Telecom Paris, Institut Polytechnique de Paris

DOI: 10.21105/joss.04158

Software

- Review
- Repository
- Archive

Editor: Frauke Wiese

Reviewers:

- @odow
- @leonardgoeke

Submitted: 28 January 2022
Published: 14 February 2022

Summary

The Graph-Based Optimization Modeling Language (GBOML) is a modeling language for mathematical programming enabling the easy implementation of a broad class of structured mixed-integer linear programs typically found in applications ranging from energy system planning to supply chain management. More precisely, the language is particularly well-suited for representing problems involving the optimization of discrete-time dynamical systems over a finite time horizon and possessing a block structure that can be encoded by a hierarchical hypergraph. The language combines elements of both algebraic and object-oriented modeling languages in order to facilitate problem encoding and model re-use, speed up model generation, expose problem structure to specialised solvers and simplify post-processing. The GBOML parser, which is implemented in Python, turns GBOML input files into hierarchical graph data structures representing optimization models. The associated tool provides both a command-line interface and a Python API to construct models, and directly interfaces with a variety of open source and commercial solvers, including structure-exploiting ones.

Statement of need

Many planning and control problems (e.g., in the field of energy systems) can be formulated as mathematical programs and the resulting models often possess special structure. Broadly speaking, two classes of tools can be used to implement such models, namely algebraic modeling languages (AMLs) and so-called application-specific modeling frameworks (ASMFs). On the one hand, AMLs usually make it possible to encode problems in a way that is close to mathematical notation. In addition, they are usually very expressive (e.g., any mixed-integer nonlinear program can be encoded), application-agnostic, and they also interface with a variety of solvers. AMLs can either be stand-alone, such as GAMS (Bussieck & Meeraus, 2004) or AMPL (Fourer et al., 1990), or they may be embedded in general-purpose programming languages, such as JuMP (Dunning et al., 2017) (in Julia) or Pyomo (Hart et al., 2011) and PuLP (Mitchell et al., 2011) (both in Python). Some of them are also open source (e.g., JuMP, Pyomo and PuLP). In short, AMLs enable users to compactly encode broad classes of optimization problems while decoupling models and solvers. On the other hand, ASMFs focus on a specific application (e.g., capacity expansion planning) and are best understood as modeling frameworks facilitating the construction of instances of a specific problem. As such, ASMFs often provide a set of pre-defined components that can be imported to build a model, along with advanced data pre- and post-processing features tailored to the application at hand. In summary, ASMFs enable the easy and modular construction of specific problem instances. In the field of energy systems, which is a particularly relevant application area for

[*] co-first author, corresponding author
[†] co-first author, corresponding author

GBOML, established ASMFs include PyPSA (power flow calculations and capacity expansion planning, among others) (Brown et al., 2018), PowerModels.jl (analysis and comparison of optimal power flow formulations) (Coffrin et al., 2018), Calliope (multi-energy carrier capacity expansion planning) (Pfenninger & Pickering, 2018), Dispa-SET (economic dispatch and unit commitment) (Kavvadias et al., 2018), Balmorel (long-term planning and short-term operational analyses) (Wiese et al., 2018) and OSeMOSYS (long-run energy planning) (Howells et al., 2011).

Unfortunately, both AMLs and ASMFs suffer from several drawbacks. More precisely, AMLs usually lack modularity. In particular, most AMLs fail to exploit the block structure that may exist in a model (e.g., to enable collaborative encoding or speed up model generation by parallelising it) or expose it for use by specialised solvers. Extensions to established AMLs were proposed to address the latter concerns, such as StructJuMP (Huchette & Developers, 2021), BlockDecomposition.jl (Marques & Developers, 2021) (both being extensions of JuMP), PySP (Watson et al., 2012) for Pyomo and SML (Colombo et al., 2009) for AMPL. However, these extensions were usually designed with the primary aim of exposing very specific problem structures (e.g., dual block angular structures found in stochastic linear programming models) or facilitating the use of specific structure-exploiting algorithms (e.g., branch-price-and-cut). On the other hand, ASMFs typically lack expressiveness and adding components is often cumbersome. Furthermore, they usually rely on established AMLs themselves, which implies that they automatically inherit any shortcomings of the specific AML on top of which they are built (e.g., some AMLs are notoriously slow or may not be open source). The tools that appear closest in spirit to our own are the recent SMS++ modeling framework (Frangioni et al., 2021) and Plasmo.jl (Jalving et al., 2020) (an extension of JuMP). Although the source code of SMS++ is publicly available, to the authors' best knowledge, documentation and examples are scarce at the time of writing. Plasmo seems to be under development and appears only partially documented.

In order to address some of these shortcomings, GBOML was designed with the following objectives in mind:

- allowing any mixed-integer linear program to be represented
- enabling any hierarchical block structure to be exposed and exploited
- facilitating the encoding and construction of time-indexed models
- allowing low-level model encoding to be close to mathematical notation
- making it easy to re-use and combine components and models
- interfacing with commercial and open source solvers, including structure-exploiting ones

The GBOML workflow is as follows. Models are first encoded by a user in GBOML input files and these files must be parsed by the GBOML parser, which is implemented in Python. A command-line interface as well as a Python API are available to work with models, which makes it possible to cater to a broad audience including both users with little programming experience and users who are proficient in Python. Model generation can also be parallelised based on the structure provided by the user. Models are then passed to open source or commercial solvers. Direct access to solver APIs is also provided, allowing users to tune algorithm parameters and retrieve complementary information (e.g., dual variables, slacks or basis ranges, when available). Finally, results are retrieved and can be either used directly in Python or printed to file. Two file formats are currently supported, namely CSV and JSON.

An early version of the tool was used in a research article studying the economics of carbon-neutral fuel production in remote areas where renewable resources are abundant (Berger et al., 2021). The tool is also used in the context of a research project focusing on the design of the future Belgian energy system.

Acknowledgements

The authors gratefully acknowledge the support of the Federal Government of Belgium through its Energy Transition Fund and the INTEGRATION project. The authors would also like to thank Adrien Bolland for his help with an early version of the documentation, Jocelyn Mbenoun for testing some early features of the tool, as well as Ghislain Detienne and Thierry Deschuyteneer for constructive discussions.

References

Berger, M., Radu, D., Detienne, G., Deschuyteneer, T., Richel, A., & Ernst, D. (2021). Remote Renewable Hubs for Carbon-Neutral Synthetic Fuel Production. *Frontiers in Energy Research*, *9*, 200. https://doi.org/10.3389/fenrg.2021.671279

Brown, T., Horsch, J., & Schlachtberger, D. (2018). PyPSA: Python for Power System Analysis. *Journal of Open Research Software*, *6*(3). https://doi.org/10.5334/jors.188

Bussieck, M. R., & Meeraus, A. (2004). General Algebraic Modeling System (GAMS). In J. Kallrath (Ed.), *Modeling Languages in Mathematical Optimization* (Vol. 88, pp. 137–157). Springer. https://doi.org/10.1007/978-1-4613-0215-5_8

Coffrin, C., Bent, R., Sundar, K., Ng, Y., & Lubin, M. (2018). PowerModels. JL: An open-source framework for exploring power flow formulations. *2018 Power Systems Computation Conference (PSCC)*, 1–8. https://doi.org/10.23919/PSCC.2018.8442948

Colombo, M., Grothey, A., Hogg, J., Woodsend, K., & Gondzio, J. (2009). A Structure-Conveying Modelling Language for Mathematical and Stochastic Programming. *Mathematical Programming Computation*, *1*(4), 223–247. https://doi.org/10.1007/s12532-009-0008-2

Dunning, I., Huchette, J., & Lubin, M. (2017). JuMP: A Modeling Language for Mathematical Optimization. *SIAM Review*, *59*(2), 295–320. https://doi.org/10.1137/15M1020575

Fourer, R., Gay, D. M., & Kernighan, B. W. (1990). A Modeling Language for Mathematical Programming. *Management Science*, *36*(5), 519–554. https://doi.org/10.1287/mnsc.36.5.519

Frangioni, A., Iardella, N., & Lobato, R. D. (2021). The SMS++ Project: A Structured Modelling System for Mathematical Models. In *Gitlab Repository*. Gitlab. https://smspp.gitlab.io/

Hart, W. E., Watson, J.-P., & Woodruff, D. L. (2011). Pyomo: Modeling and Solving Mathematical Programs in Python. *Mathematical Programming Computation*, *3*(3), 219–260. https://doi.org/10.1007/s12532-011-0026-8

Howells, M., Rogner, H., Strachan, N., Heaps, C., Huntington, H., Kypreos, S., Hughes, A., Silveira, S., DeCarolis, J., Bazillian, M., & Roehrl, A. (2011). OSeMOSYS: The Open Source Energy Modeling System: An Introduction to its Ethos, Structure and Development. *Energy Policy*, *39*(10), 5850–5870. https://doi.org/10.1016/j.enpol.2011.06.033

Huchette, J., & Developers, S. (2021). StructJuMP: A Block-Structured Optimization Framework for JuMP. In *GitHub Repository*. GitHub. https://github.com/StructJuMP/StructJuMP.jl

Jalving, J., Shin, S., & Zavala, V. M. (2020). *A Graph-Based Modeling Abstraction for Optimization: Concepts and Implementation in Plasmo.jl*. http://arxiv.org/abs/2006.05378

Kavvadias, K., Hidalgo Gonzalez, I., Zucker, A., & Quoilin, S. (2018). *Integrated Modelling of Future EU Power and Heat Systems: the Dispa-SET v2.2 Open-Source Model*. European Commission Joint Research Centre.

Marques, G., & Developers, BlockDecomposition. jl. (2021). BlockDecomposition.jl: Modelling Decomposable Programs in JuMP. In *GitHub Repository*. GitHub. https://github.com/StructJuMP/StructJuMP.jl

Mitchell, S., O'Sullivan, M., & Dunning, I. (2011). *PuLP: A Linear Programming Toolkit for Python*. http://www.optimization-online.org/DB_FILE/2011/09/3178.pdf

Pfenninger, S., & Pickering, B. (2018). Calliope: A Multi-Scale Energy Systems Modelling Framework. *Journal of Open Source Software*, *3*(29), 825. https://doi.org/10.21105/joss.00825

Watson, J.-P., Woodruff, D. L., & Hart, W. E. (2012). PySP: Modeling and Solving Stochastic Programs in Python. *Mathematical Programming Computation*, *4*(2), 109–149. https://doi.org/10.1007/s12532-012-0036-1

Wiese, F., Bramstoft, R., Koduvere, H., Pizarro Alonso, A., Balyk, O., Kirkerud, J. G., Tveten, Å. G., Bolkesjø, T. F., Münster, M., & Ravn, H. (2018). Balmorel Open Source Energy System Model. *Energy Strategy Reviews*, *20*, 26–34. https://doi.org/10.1016/j.esr.2018.01.003