GRAVITY: A Mathematical Modeling Language for Optimization and Machine Learning

Hassan Hijazi
Los Alamos National Laboratory
Los Alamos, NM 87544
hlh@lanl.gov

Guanglei Wang
The Australian National University
Canberra, ACT, Australia

Carleton Coffrin
Los Alamos National Laboratory
Los Alamos, NM 87544

Abstract

GRAVITY is an open source, scalable, memory efficient modeling language for solving mathematical models in Optimization and Machine Learning. GRAVITY exploits structure to reduce function evaluation time including Jacobian and Hessian computation. GRAVITY is implemented in C++ with a flexible interface allowing the user to specify the numerical accuracy of variables and parameters. It is also designed to offer efficient iterative model solving, convexity detection, multithreading of subproblems, and lazy constraint generation. When compared to state-of-the-art modeling languages such as JUMP, GRAVITY is 5 times faster in terms of function evaluation and up to 60 times more memory efficient. GRAVITY enables researchers and practitioners to access state-of-the-art optimization solvers with a user-friendly interface for writing general mixed-integer nonlinear models.

1 Introduction

While modeling languages play a critical role across all scientific areas, they constitute a key stone in Computer Science and Applied Mathematics. Modeling tools are ubiquitous to fields like Constraint Programming (CP), Artificial Intelligence (AI) and Operations Research (OR). A non-exhaustive list of existing modeling languages include AMPL (1), JUMP (2), GAMS (3), AIMMS (4), CASADI (5), PYOMO (6; 7), and MINIZINC (8). With increasingly large data inputs and the desire to model complex nonlinear functions, the efficiency of modeling tools is becoming critical for the implementation of scalable solution algorithms. This is especially true for Machine Learning (ML) applications dealing with large data sets and using nonlinear learning functions. Note that tools like TENSORFLOW(9) and THEANO(10) are highly specialized modeling languages for deep learning. Modern mathematical modeling languages need to offer a new set of features. These include an efficient support for user-defined constraints, automatic convexity detection, hybrid numerical precision for variables and parameters, multi-threaded subproblem solving, lazy constraint generation, automatic reformulation and handling large data sets, just to name a few. GRAVITY is designed with these requirements in mind.

2 Design Choices

GRAVITY's efficiency in speed and memory can mainly be attributed to its structure exploiting design choices. Structure is what differentiate various classes of mathematical models, e.g., Linear Programs (LP), Quadratic Programs (QP) or general Nonlinear Programs (NLP). Problems solved by scientists and practitioners will inherently have structure, and exploiting it is key to performance boosting.

Below, we highlight some of the main design choices proper to GRAVITY:

Preprint. Work in progress.

Template Constraints. Structure can be exploited by considering the fact that most mathematical formulations have a small set of "template" constraints, i.e., an abstract/symbolic representation of the constraint where only variables and parameters' indices change.

Flexible Numerical Precision. Being able to select different numerical precision for different subset of variables and parameters is another critical design choice made in GRAVITY. An example where we mix double and short int parameters can be found here.

Graph Aware. Graphs and networks are also ubiquitous in computer science and especially in optimization and machine learning. GRAVITY has an underlying graph implementation and indexing can be done on nodes and edges of the graph. Some useful graph algorithms such as tree decomposition and cycle basis computation are also implemented in GRAVITY.

Plug And Play Depending on the nature of your mathematical formulation, GRAVITY will allow you to call the corresponding solver. GRAVITY currently links to Mixed-Integer Linear and Quadratic Optimization solvers CPLEX (11) and GUROBI (12), Nonlinear Optimization solver IPOPT(13), Convex Mixed-Integer Nonlinear solver BONMIN(14) and Semidefinite Programming solver MOSEK (15).

Symbolic Differentiation Automatic Differentiation (AD) has numerous benefits, including the ability to compute derivative for non-mathematical structures, e.g., computer code. There is an extensive literature on the subject, we only provide a few references herein (20; 21; 22; 23; 24). The common belief is that AD outperforms Symbolic Differentiation (SD) (see (23; 25)) . Let us emphasize that AD is currently the norm in all state-of-the-art mathematical modeling tools including ML frameworks such as THEANO (10) and TENSORFLOW (9). In this work, we show that a careful SD implementation can dominate AD by exploiting the structure of the underlying mathematical formulations. Avoiding redundant storage of symbolic nonlinear expressions helps our SD outperform state-of-the-art AD implementations (e.g., (2), which uses graph coloring methods for exploiting sparsity of the Hessian matrix).

Model Readability. Readability of the models was also a main concern during the design process of GRAVITY. This can be seen in the example presented in Figure 1.

Efficiency and Multithreading. GRAVITY is implemented in C++, allowing for a flexible memory management and various code optimization under the hood. The user can also simultaneously build multiple mathematical models and call GRAVITY's run_parallel() function.

$$\alpha(G) = \max_{x} \sum_{i=1}^{n} x_i$$
$$\text{s.t. } x_i + x_j \leq 1, \text{ if } \{i,j\} \in E,$$
$$x_i \in \{0,1\},\ i = 1,\ldots,n.$$

(a) Mathematical formulation

```
/**  IP model for the stable set problem. **/

/**  Variables **/
/* Declaring the n-dimensional Real space */
auto Rn = R(n);
var<bool> x("x");
model.add(x.in(Rn));

/**  Objective **/
model.max(sum(x));

/**  Constraints **/
Constraint c("Stable_Set");
c = x.from() + x.to();
model.add(c.in(graph.arcs) <= 1);

/**  Solver **/
SolverType stype = gurobi;
solver s(model,stype);
s.run();
```

(b) Implementation

Figure 1: Implementation of the Stable Set problem in GRAVITY

3 Numerical Experiments

Numerical experiments were conducted on HPE ProLiant XL170r servers featuring two Intel2.10 GHz 16 Core CPUs and 128 GB of memory. IPOPT v3.12 (13) compiled with HSL (26) was used for solving Nonlinear Programs. In the results tables, "mem." indicates that available memory was exceeded for the corresponding instance.

3.1 AC Optimal Power Flow

The Alternating Current Optimal Power Flow (ACOPF) Problem is a fundamental building block in Power Systems Optimization. The problem admits two nonconvex NLP formulations, one known

as the polar formulation (featuring trigonometric functions) and one known as the rectangular formulation (quadratically-constrained). A comprehensive description of the two formulations can be found in (27). Figure 2 is a performance profile illustrating percentage of instances solved as a function of time. The figure compares GRAVITY, JUMP and AMPL's NL interface (used by AMPL and PYOMO) on all standard instances found in the pglib benchmark library (28). The recorded time corresponds to the wall-clock time spent inside IPOPT. Figure 2 indicates that GRAVITY outperforms both JUMP and AMPL's NL interface on all instances, with speed improvements up to 300%. Note that the largest instance has $117,370$ variables, $187,371$ constraints, $666,023$ non zeros in the Jacobian, and $299,384$ non zeros in the Hessian matrix.

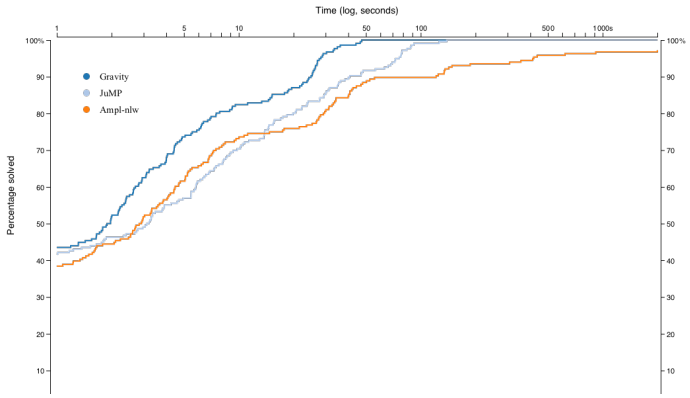

Figure 2: Performance profile on Polar and Rectangular ACOPF (216 instances).

3.2 Learning of Ising Models

Learning the structure and the parameters of an Ising model is a typical Machine Learning problem where data input quickly becomes problematic. Matrices with 92 million non-zero entries appear in the biggest instances. A comprehensive formulation of the problem can be found in (30) and (31). Figure 3 shows that both JUMP and AMPL crash after a certain threshold due to memory issues. GRAVITY is able to scale up on all tested instances. Since the problem has a natural parallelization procedure, a parallel implementation in GRAVITY was also tested on this problem. Figure 3 also compares GRAVITY running with 12 threads, showing gains up to one order of magnitude.

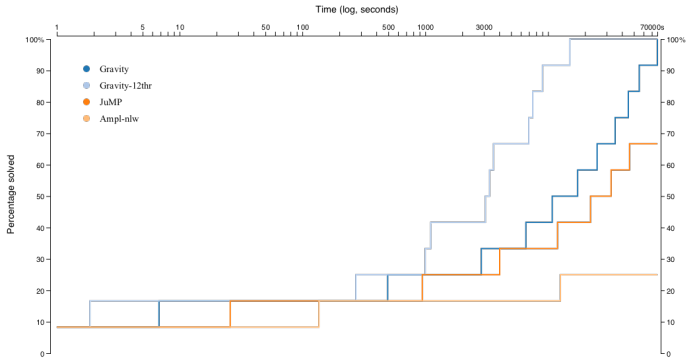

Figure 3: Performance profile on Inverse Ising problems (12 instances).

4 Links

GRAVITY has recently joined the COIN-OR https://www.coin-or.org initiative and can be downloaded here https://github.com/coin-or/Gravity.
Documentation and examples can be found at https://www.allinsights.io/gravity.

Acknowledgments
We acknowledge the support of Schloss Dagstuhl for hosting the seminar on “Designing and Implementing Algorithms for Mixed-Integer Nonlinear Optimization".

References

[1] Fourer, R., Gay, D.M., Kernighan, B.: Algorithms and model formulations in mathematical programming. Springer-Verlag New York, Inc., New York, NY, USA (1989) 150–151

[2] Dunning, I., Huchette, J., Lubin, M.: Jump: A modeling language for mathematical optimization. SIAM Review **59**(2) (2017) 295–320

[3] Bussieck, M.R., Meeraus, A.: General algebraic modeling system (gams). Applied Optimization **88** (2004) 137–158

[4] Bisschop, J.: AIMMS - Optimization Modeling. Lulu.com (2006)

[5] Andersson, J., Åkesson, J., Diehl, M.: CasADI: A symbolic package for automatic differentiation and optimal control. In Forth, S., Hovland, P., Phipps, E., Utke, J., Walther, A., eds.: Recent Advances in Algorithmic Differentiation. Volume 87 of Lecture Notes in Computational Science and Engineering. Springer, Berlin (2012) 297–307

[6] Hart, W.E., Watson, J.P., Woodruff, D.L.: Pyomo: modeling and solving mathematical programs in python. Mathematical Programming Computation **3**(3) (2011) 219–260

[7] Hart, W.E., Laird, C.D., Watson, J.P., Woodruff, D.L., Hackebeil, G.A., Nicholson, B.L., Siirola, J.D.: Pyomo–optimization modeling in python. Second edn. Volume 67. Springer Science & Business Media (2017)

[8] Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In: Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming. CP'07, Berlin, Heidelberg, Springer-Verlag (2007) 529–543

[9] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). (2016) 265–283

[10] Theano Development Team: Theano: A Python framework for fast computation of mathematical expressions. arXiv e-prints **abs/1605.02688** (May 2016)

[11] Ibm: IBM ILOG CPLEX Optimization Studio CPLEX User's Manual. (2017)

[12] Gurobi Optimization, I.: Gurobi optimizer reference manual (2017)

[13] Wächter, A., Biegler, L.T.: On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. Mathematical Programming **106**(1) (2006) 25–57

[14] Bonami, P., Biegler, L.T., Conn, A.R., Cornuejols, G., Grossmann, I.E., Laird, C.D., Lee, J., Lodi, A., Margot, F., Sawaya, N., Wächter, A.: An algorithmic framework for convex mixed integer nonlinear programs. Discrete Optimization **5**(2) (2008) 186 – 204

[15] Mosek ApS: The MOSEK optimization software

[16] Achterberg, T.: SCIP: solving constraint integer programs. Mathematical Programming Computation **1**(1) (2009) 1–41

[17] Dash Optimization: Xpress-MP (2001)

[18] Belotti, P.: Couenne: User manual. Published online at https://projects.coin-or.org/Couenne/ (2009) Accessed: 10/04/2015.

[19] Sorensson, N., Een, N.: Minisat v1. 13-a sat solver with conflict-clause minimization. SAT **2005**(53) (2005) 1–2

[20] Rall, L.B.: Automatic Differentiation: Techniques and Applications. Volume 120 of Lecture Notes in Computer Science. Springer, Berlin (1981)

[21] Griewank, A., Juedes, D., Utke, J.: Algorithm 755: Adol-c: A package for the automatic differentiation of algorithms written in c/c++. ACM Trans. Math. Softw. **22**(2) (June 1996) 131–167

[22] Rall, L.B., Corliss, G.F.: An introduction to automatic differentiation. Computational Differentiation: Techniques, Applications, and Tools **89** (1996)

[23] Fournier, D.A., Skaug, H.J., Ancheta, J., Ianelli, J., Magnusson, A., Maunder, M.N., Nielsen, A., Sibert, J.: Ad model builder: using automatic differentiation for statistical inference of highly parameterized complex nonlinear models. Optimization Methods and Software **27**(2) (2012) 233–249

[24] Bell, B.M.: CppAD: a package for C++ algorithmic differentiation. Computational Infrastructure for Operations Research (2012)

[25] Ahn, S., Kaplan, G., Moll, B., Winberry, T., Wolf, C.: When inequality matters for macro and macro matters for inequality. NBER Working Paper No. w23494 https://ssrn.com/abstract=2984671 (June 2017)

[26] U.K., R.C.: The hsl mathematical software library Published online at http://www.hsl.rl.ac.uk/.

[27] Hijazi, H., Coffrin, C., Van Hentenryck, P.: Convex quadratic relaxations for mixed-integer nonlinear programs in power systems. Mathematical Programming Computation **9**(3) (2017) 321–367

[28] The IEEE PES Task Force on Benchmarks for Validation of Emerging Power System Algorithms: PGLib Optimal Power Flow Benchmarks. Published online at https://github.com/power-grid-lib/pglib-opf Accessed: October 4, 2017.

[29] Jabr, R.: Radial distribution load flow using conic programming. Power Systems, IEEE Transactions on **21**(3) (Aug 2006) 1458–1459

[30] Vuffray, M., Misra, S., Lokhov, A., Chertkov, M.: Interaction screening: Efficient and sample-optimal learning of ising models. In: Advances in Neural Information Processing Systems. (2016) 2595–2603

[31] Lokhov, A.Y., Vuffray, M., Misra, S., Chertkov, M.: Optimal structure and parameter learning of ising models. arXiv preprint arXiv:1612.05024 (2016)