Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 242 (2024) 460–465

Procedia Computer Science

www.elsevier.com/locate/procedia

11th International Conference on Information Technology and Quantitative Management (ITQM 2024)

Lua APIs for mathematical optimization

Milan Stanojević[a], Bogdana Stanojević[a,b,*]

[a] University of Belgrade, Faculty of Organizational Sciences, Jove Ilića 154, 11000 Belgrade, Serbia
[b] Mathematical Institute of the Serbian Academy of Sciences and Arts, Kneza Mihaila 36, 11000 Belgrade, Serbia

Abstract

In this paper we present our Lua programming language libraries that enable modeling and solving mixed integer linear and nonlinear mathematical optimization problems. On one side, these libraries provide a framework for developing algorithms written in Lua that use certain well known solvers within more complex procedures. On the other side, they facilitate the transformation of both input and output data of a mathematical programming problem, in order to compute the standardized coefficients that have to be transmitted to the solvers or prepare solution of solved problems for further data processing. Both of these use cases, together with Lua programming language simplicity, versatility and performance, make Lua a suitable programming language for use in scientific and engineering researches.

© 2024 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the 11th International Conference on Information Technology and Quantitative Management

Keywords: Dynamic callable library; AMPL/GMPL languages; CBC solver; ALGENCAN; mathematical optimization

1. Scientific programming

Scientific programming (SP) is the development of programs intended to be used in scientific and engineering research. A programming language that is suitable for this kind of programming is called a scientific programming language (SPL). In a wide sense an SPL is able to efficiently implement algorithms for computational science or computational mathematics, but in stronger sense is a language optimized to permit easy implementations of complex mathematical calculations.

Some of the main properties of SP and the demands of the SPLs that follow them are:

[*] Corresponding author. Tel.: +381-11-2630170 ; fax: +381-11-2186105.
E-mail address: bgdnpop@mi.sanu.ac.rs

1877-0509 © 2024 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (https://creativecommons.org/licenses/by-nc-nd/4.0)
Peer-review under responsibility of the scientific committee of the 11th International Conference on Information Technology and Quantitative Management
10.1016/j.procs.2024.08.160

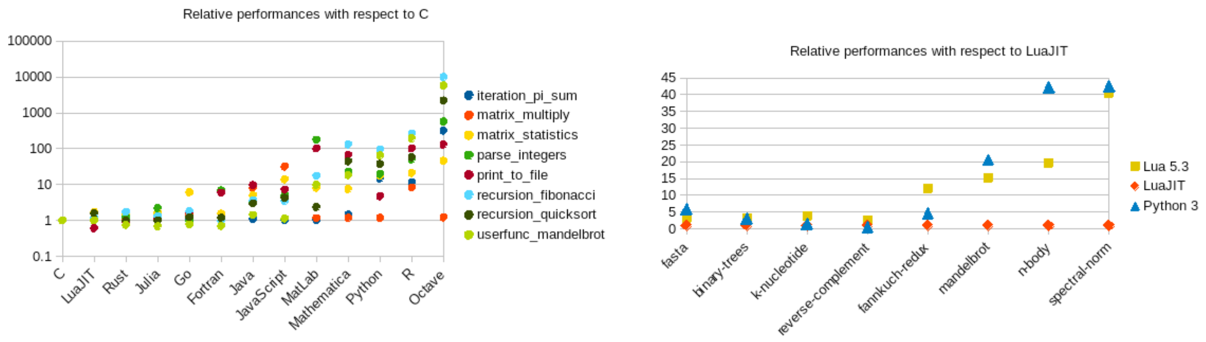

Fig. 1. Julia micro-benchmark results in a logarithmic scale to compare the performances of several programming languages.

- Those who develop the software are also its users, thus the programs do not have to have sophisticated or graphic interface. In the most cases they are just console applications that take input data from files or a database and store the results in a similar fashion.
- Those who develop the software are not necessary professional developers, thus the programming language has to have a simple and yet expressive syntax. For this demand the interpreted languages have a clear advantage over compiled and strong typed ones. On the other side, the drawback is that interpreted languages are several orders of magnitude slower.
- The need for high computation performances is inconsistent with the previous demand. A possible solution is to identify extremely complex parts of procedures, implement them in some high performance languages (typically C/C++, FORTRAN), and then merge them as dynamic libraries with the rest of the code. So, the possibility and easiness of linking libraries developed in other languages is a key demand for this compromise solution.
- Even with libraries for fast calculation, the rest of the code should be as fast as possible. In the interpreted languages world, a high running speed can be achieved by Just In Time (JIT) compilers.

Matlab together with its open source variants Octave and SciLab, Mathematica, R, Prolog, AMPL are some example of specialized languages for SP. On the other side, Julia, Python and Lua are general programming languages which are suitable for SP.

Figure 1 shows some comparative results about the performances of several programming languages. The implementations of the algorithms were downloaded from [7] and run on the same computer. On the left, the performances were reported to to the performance of the C language, while on the right, they are presented with respect to LuaJIT.

One may conclude that LuaJIT performed better than both Lua 5.3 and Python 3 for all tests with only one exception. Python 3 succeeded to outperform LuaJIT for reverse-complement benchmark due to the wise use of co-routines for reading in the 500 MB input file.

2. Lua programming language

Lua programming language [6] has all mentioned qualities needed for a SPL in a wide sense. Although Lua has a small footprint (i.e. the size of its interpreter is only 600 kB), it has many features that the modern SPLs have. Among these features it is worth mentioning the expressiveness, dynamic typing, simplicity, speed and extensibility. Most of all, Lua is freely available as an open source project. Lua interpreter itself is written in C programming language, so its support for integration with C programs is quite natural and intuitive. To combine Lua and C one can either embed Lua into C programs, thus extending it with advanced syntax capabilities; or expand Lua with libraries written in C, thus speeding up critical complex computation. Lua does not incorporate in the language all the desirable possibilities of SPLs like mathematical formulas with matrices manipulation. It rather uses external libraries that extend the language capabilities. And

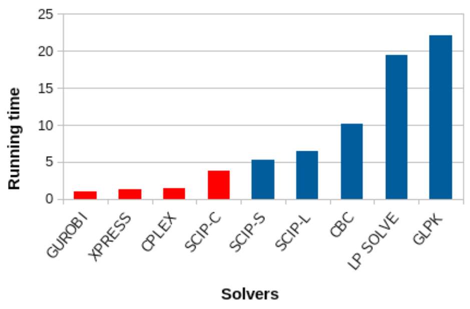

Fig. 2. Speed comparison of commercial and open source solvers

here the Lua's extensibility shines. There are several available libraries that provide some SPL capabilities to Lua. NumLua [10] and Torch [14] are two of them. NumLua is focused on complex numbers, matrices computation, statistics, random numbers generation; while Torch, in addition is directed to neural networks, image manipulation, graphical representation and some rudimentary optimization algorithms.

The usefulness of Lua in implementing algorithms within scientific computing can be concluded from the recent literature (see for instance Ono et al. [11] who describe a modular visualization framework for large-scale data sets that uses Lua as scripting language). Cacho et al. [4] developed a Lua-based aspect-oriented programming infrastructure by creating an aspect class used to define aspects that are dynamically weaved by a meta-object protocol.

The main reasons for using Lua were summarized by the team that designed, implemented and maintain it [6] as it follows: Lua is a proven, robust language; fast, portable, embeddable, powerful (but simple), small, and free. Moreover, Lua is multi paradigm programming language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.

3. Optimization solvers

Mixed integer programming (MIP) open source solvers, beside their flexibility of use and lack of cost, lag behind their commercial counterparts considering performance. Figure 2 shows the speed span of several commercial (red bars) and open source (blue bars) MIP solvers. The numerical values used in Figure 2 are recalled from [1]. Meindl and Templ [9] explained how the data was collected. Briefly, the maximal running time for each solver was limited to 1h; and the times were scaled such that the running time for GUROBI® - which was the fastest solver in the experiment - was set to 1. Smaller values are better. GUROBI®, XPRESS® and CPLEX® are well known high performance commercial MIP solvers that are constantly fine-tuned and improved by building in the state of art techniques and algorithms. On the other hand CBC, LP_SOLVE and GLPK are mature and reliable open source solvers, but the effort put in their improvement is a way smaller comparing to commercial ones.

Solver SCIP can solve constraint programming, MIP and mixed integer nonlinear programming (MINLP) problems using branch-cut-and-price method, but uses an external LP solver. In the benchmark represented in Figure 2, SCIP used CPLEX (-C), CLP (-L) and SoPlex (-S) LP solvers. SCIP-C can be considered as a commercial solver since it needs a valid license for CPLEX.

The comparison of nonlinear optimization solvers is much harder than for the linear ones.

4. Optimization in Lua programming language

We have implemented a set of libraries that enable the use of both open source and commercial solvers in Lua with a decent modeling and model manipulation capabilities. Among others, we developed APIs for AMPL and all the solvers that can be used from it, CBC solver, and ALGENCAN.

Table 1. Running times comparison on TSPLIB benchmarks

TSPLIB instance	n	Gurobi	CPLEX	CBC	GLPK
ftv3	34	0.51	0.65	1.30	0.57
ftv47	48	5.88	9.69	66.66	30.85
ftv55	56	8.05	4.23	37.43	108.45
	Total	14.44	14.57	105.39	139.87

4.1. AMPL

One of our libraries is a wrapper of the AMPL's C++ API version adjusted to the Lua style data manipulation. The dynamic library *ampllua* is written in C++ and can be loaded in Lua (5.1 and newer and LuaJIT) code. The main purpose of our API is to provide the procedural portion of modeling in optimization processes. The mathematical problem remains written in AMPL but data manipulation for both preparing and afterwards analytic is done in Lua. All entities from the model can be accessed from Lua directly, in Lua fashion. The input values can be read in many ways, combining AMPL's native .dat files and assignments in Lua code, preceded by database or any data files reading [12].

4.2. CBC solver

CBC is a project of COIN-OR Foundation [5] that supports open source software for the operations research community. It is written in C++ and issued under Eclipse Public License [2]. COIN-OR stands for Computational Infrastructure for Operations Research. COIN-OR successfully supports researchers to develop mathematical applications. The COIN-OR includes tools for linear programming, nonlinear programming, mixed integer programming, and algebraic modeling environments.

The CBC API wrapper was developed as a dynamic Lua library, called *cbclua*, which can be imported and used within any Lua program. We developed corresponding Lua functions for the majority of CBC API functionalities [13]. In the wrapper creation the attention was put on the simplicity of input data structure with accent on native Lua collection structures – tables.

The GLPK (GNU Linear Programming Kit) is a well-known open source package for solving large-scale linear programming (LP), MIP, and other related problems. It is a set of routines organized as a callable library written in ANSI C. For mathematical modeling it uses the GMPL (GNU MathProg modelling language) a derivative and a subset of the commercial AMPL language. Defining an optimization problem by forming a constraint matrix is convenient for dense matrices, but is not suitable for sparse ones. For such models, a mathematical modeling language is more appropriate to be used. In *cbclua* library that ability is introduced by incorporating GMPL. We opted for using GMPL due to its syntax which is very similar to AMPL, and its quality of being part of an open source package. Combining the CBC solver with GLPK modeling environment enables: (i) to define complex mathematical models and insert data in GMPL format; (ii) to read in both the model and the data into a CBC mode within Lua code; and finally (iii) to perform the optimization with CBC solver (that is faster then GLPK native MIP solver, see Figure 2) and extract the results.

Table 1 reports the running time (in seconds) needed by two commercial and two open-source solvers to solve three instances of Asymmetric TSP problem taken from the well known library TSPLIB [15]. CBC solver was run using the developed Lua interface, while the others were run from the available modeling environments.

The values reported on the bottom row of Table 1 properly illustrate the performances of the compared solvers: the commercial solvers are much better then the free ones; and CBC solver outperforms the GLPK solver. However, some anomalies can be noticed analyzing the results obtained for each instance separately: Gurobi needed almost double time than CPLEX to solve ftv55, and vice versa for ftv47 instance; GLPK behaved better than CBC on instance ftv47; and GLPK outperforms CPLEX when solving the instance ftv33. However, these anomalies well illustrate the capriciousness of combinatorial optimization problems as well as the dependence of solving algorithms on their tuning parameters (all used solvers were run with their default settings).

4.3. ALGENCAN solver

ALGENCAN [3] is a procedure based on traditional ideas and written in FORTRAN for general constrained optimization. It is able to solve extremely large optimization problems within reasonable time. At each iteration, the algorithm minimizes the objective function plus a quadratic penalty function, thus being an approximate minimizer. The optimization algorithm is based on the Augmented Lagrangian method for nonlinear programming problems. It involves many parameters that highly influence the algorithmic behavior. ALGENCAN has interfaces with AMPL, C/C++, CUTEr, Matlab, Python, Octave and R (statistical computing).

Unlike the previous two libraries that use the external modeling environments AMPL and MathProg, respectively, *luaalgencan* implements an original modeling environment for formulating mathematical models. For this purpose, it utilizes native Lua data structures, the tables – the associative arrays (in other languages known as: dictionary, hash array, etc.). Beside Lua's C API and Algencan, *luaalgencan* incorporates the mathematical expression parser and evaluator TinyExpr [16].

The basic modeling objects in *luaalgencan* environment are:

1. Parameters (denoted by `par`) are numerical values that can be changed during run-time,
2. Variables (denoted by `var`) are the decision variables of the mathematical model,
3. Functions (denoted by `fun`) are used to represent both the objective function and constraints.

Each of the objects must have a name and may have some attributes. Reserved name for the objective function is `min` if the function is minimized, or `max` if it is maximized. All other names are arbitrary (excluding some keywords).

The body of the function objects are formulated as strings with the help of iterators that provide more concise expressions. The iterators may be used to defining variables, too.

The example *luaalgencan* code for Model (1)

$$\min f(x) = -\left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^4(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|,$$

$$\text{subject to} \quad g_1(x) = 0.75 - \prod_{i=1}^{n} x_i \leq 0, \qquad g_2(x) = \sum_{i=1}^{n} x_i - 7.5n \leq 0, \tag{1}$$

where $n = 20$ and $0 \leq x_i \leq 10$, $i = 1, \ldots, n$, is given in Figure 3. Problem (1) was recalled from [8].

Note that two dots "`..`" is the operator for string concatenation; and "`ub`" and `"lb"` are attributes for upper and lower bounds for both variables and constraints. The argument `100` of `m:solve` command is number of optimization multi starts with randomized initial variables values. The initial values can be given as the unnamed argument of variable declaration (in line 2 at Figure 3, initial value for all variables is 0.1). The second optional argument of `m:solve` command can be a random seed.

```
require "luaalgencan"
m = newModel()
m:varSet{xi = {0.1, lb = 0, ub = 6.5}, {i = 1, 20}}
m:funSet{min = {"-abs((" .. sum({i=1,20}, "(cos xi)^4") .. " - 2 * " ..
                 prd({i=1,20}, "(cos xi)^2") .. ") /sqrt(" ..
                 sum({i=1,20}, "i * xi^2") .. "))"}}
m:funSet{g1 = {prd({i=1,20}, "xi"), lb = 0.75}}
m:funSet{g2 = {sum({i=1,20}, "xi"), ub = 150}}
m:solve(100)
m:showSolution(true)
```

Fig. 3. Lua code for solving mathematical problem 1

5. Conclusion

We developed three libraries: *ampllua*, *cbclua*, and *luaalgencan*. The libraries that enable the use of the CBC solver and solvers supported by AMPL within Lua code, thus facilitating the solving of MIP problems with unbounded number of variables using Lua, incorporate external modelling environments: MathProg language from GLPK, and AMPL. On the other side, the library that enables the use of the non-linear solver Algencan with Lua includes our new developed modelling environement that facilitates the mathematial formulations.

By developing these libraries we aimed to improve Lua's performances toward mathematical optimization.

In our future research we will focus on incorporating the access to an open-source non-linear solver through Lua libraries. The main difficulty that has to be overcome is related to reading in the mathematical model. An algebraic language is desirable to be developed in order to permit a very general input form for non-linear expressions that describe the objective functions and constraints.

References

[1] , . The computer language benchmarks game, lua versus python 3 fastest programs. URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html. last accessed 2 May 2024.

[2] , t..E., . URL: https://www.eclipse.org/legal/epl-2.0/.. last accessed 2 May 2024.

[3] ALGENCAN, . TANGO project. URL: https://www.ime.usp.br/~egbirgin/tango/codes.php. last accessed 2 May 2024.

[4] Cacho, N., Batista, T., Fernandes, F., 2005. A Lua-based AOP infrastructure. Journal of the Brazilian Computer Society , 7–20.

[5] COIN-OR, . Computational infrastructure for operations research - open-source software for the operations research community. URL: www.coin-or.org. last accessed 2 May 2024.

[6] Ierusalimschy, R., . The programming language lua. URL: https://www.lua.org. last accessed 2 May 2024.

[7] Julia, . Mikro-benchmarks. URL: https://julialang.org/benchmarks/. last accessed 2 May 2024.

[8] Liang, J., Runarsson, T., Mezura-Montes, E., Clerc, M., Suganthan, P., Coello, C., Deb, K., 2006. Problem definitions and evaluation criteria for the cec 2006 special session on constrained real-parameter optimization. Nangyang Technological University, Singapore, Tech. Rep 41.

[9] Meindl, B., Templ, M., 2012. Analysis of commercial and free and open source solvers for linear optimization problems. URL: http://hdl.handle.net/20.500.12708/37465.

[10] Numlua, . Numeric lua. URL: https://github.com/carvalho/numlua. last accessed 2 May 2024.

[11] Ono, K., Nonaka, J., Kawanabe, T., Fujita, M., Oku, K., Hatta, K., 2020. HIVE: A cross-platform, modular visualization framework for large-scale data sets. Future Generation Computer Systems , 875–883.

[12] Stanojević, M., Stanojević, B., 2020. Mathematical optimization in lua programming language environment, in: Proceedings of SYM-OP-IS 2020, Saobraćajni fakultet, Belgrade. pp. 473–478.

[13] Stanojević, M., Stanojević, B., 2023. Mathematical optimization using cbc solver in Lua programming language, in: Proceedings of SYM-OP-IS 2023, Media centar Odbrana, Belgrade. pp. 903–908.

[14] Torch, . A scientific computing framework for luajit. URL: http://torch.ch/. last accessed 2 May 2024.

[15] TSPLIB, . Instances for the TSP (and related problems). URL: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. last accessed 2 May 2024.

[16] Winkle, L.V., . Tinyexpr. URL: https://codeplea.com/tinyexpr. last accessed 17 June 2024.