First steps to solve MINLP problems with Muriqui Optimizer

Wendel Melo
Computer Science College
Federal University of Uberlandia, Brazil
wendelmelo@ufu.br

Marcia Fampa
Institute of Mathematics and COPPE
Federal University of Rio de Janeiro fampa@cos.ufrj.br

Fernanda Raupp
National Laboratory for Scientific Computing (LNCC)
Ministry of Science, Technology and Innovation, Brazil
fernanda@lncc.br

ABSTRACT

Muriqui Optimizer is a software to solve Mixed Integer Nonlinear Programming (MINLP) problems. Muriqui includes an executable file to be used with the AMPL system and a powerful C++ library for integrating with various software. The main differential of Muriqui is that it brings the possibility of using the main MINLP algorithms of the literature, with a series of customizable parameters. This text provides general instructions for obtaining, installing and using Muriqui.

1 What is the Muriqui Optimizer?

Muriqui Optimizer is a solver of convex Mixed Integer Nonlinear Programming (MINLP) problems. Algebraically, Muriqui Optimizer solves problems in the format:

$$
\begin{array}{lllll}
(P) & z := \min\limits_{x} & f(x) + \frac{1}{2}x'Q_0x + c'x + d, & & \\
& \text{subject to:} \quad l_{c_i} \leq & g_i(x) + \frac{1}{2}x'Q_ix + a_ix & \leq u_{c_i}, & i = 1, \ldots, m, \\
& l_x \leq & x & \leq u_x, & \\
& & x_j \in \mathbb{Z}, \text{ for } j \in \mathcal{I}, & &
\end{array} \tag{1}
$$

Note that x denotes the vector with the decision variables. The vectors l_x and u_x bring lower and upper bounds for x, and $\mathcal{I}$ is the set of indexes of variables constrained to integer values. Concerning the objective function, $f(x)$ represents a nonlinear term, Q_0 is a symmetric matrix used to describe the quadratic term, c is a column vector used to describe the linear term and d is a constant term. For each constraint i, $i = 1, \ldots, m$, $g_i(x)$ represents a nonlinear term, Q_i is a symmetric matrix used to describe a quadratic term, a_i

is a row vector used to describe a linear term and l_{c_i} and u_{c_i} are lower and upper bounds, respectively.

Certainly, linear and quadratic terms can be represented in the functions $f(x)$ and $g_i(x), i = 1, \ldots, m$. Nevertheless, the decomposition in linear, quadratic and general nonlinear terms aims to make possible a better use of the characteristics of problem (P). In addition, some available nonlinear programming (NLP) solvers can take advantage of this decomposition to solve relaxations of problem (P) more quickly.

Also, we point out that some of the elements in (P) may be "missing". For example, some components of the vectors l_c or l_x can be defined as $-\infty$, while some of the components of the vectors u_c or u_x can be defined as $+\infty$, some of the functions $g_i(x)$ or $f(x)$ can be null, as well as the matrices Q_i and the vectors c and a_i.

Muriqui has been developed to solve convex problems. However, non-convex problems can also be addressed, without the guarantee of obtaining optimal solution. Ideally, the functions $f(x)$ and $g_i(x)$ must be doubly differentiable and evaluated at any point x that satisfies the box constraints (1), but it may be possible, in some cases, to circumvent this last requirement with approximations for the derivatives.

Muriqui brings the idea that a generic configuration may not be the best choice in a specific application. Thus, a remarkable feature of Muriqui is the wide range of customizable options available. Therefore, users have a wide range of choices of applications including the choice of the algorithm to be used, the parameters setting of these algorithms, and the solvers used to handle the generated subproblems. From this perspective, users are encouraged to pass default settings and test different configurations to find the best option for each specific situation.

2 Who are we?

We are a group of researchers interested in the MINLP optimization area. As part of our work, we developed our own solver called Muriqui, whose initial goal was to only support our research. Nevertheless, we would be glad if our work had applicability to other MINLP researchers or practitioners. We are looking forward to receiving feedback, even if they are bad reports in other to improve it.

3 License and terms of use

For the time being, Muriqui is a totally free and open MINLP solver, and can be used through the MIT license. We kindly ask the users to cite our paper describing the implemented algorithms on the documents where they report its use. We also ask the users to have some patience when using Muriqui, especially the first versions, that may contain some bugs. We will be very grateful if the users can report to us errors and bugs found.

It is important to note that, although Muriqui is a free software, it is not self-contained. This means that the installation of other software and libraries are required for its correct operation (see Section 4). Each of these software may have its own usage and license policies. For proper use of Muriqui, these software must be installed under the responsibility of the user.

4 Required software

If the user intends to install Muriqui using its source code, he/she will initially need a C++ 2011 compiler. On our tests, we have succeeded with distinct versions of compilers GCC

and ICPCin Linuxsystems. In order to take full advantage of the features offered, it is recommended to install: the AMPL Solver Library (ASL), at least one solver for Mixed Linear Programming (MILP) problems, and at least one solver for Nonlinear Programming (NLP) problems. In the following, we include some instructions for these installations.

4.1 AMPL Solver Library

The AMPL Solver Library (ASL) is necessary to integrate Muriqui with the AMPL system. Once it is installed, it is possible to generate an executable file of Muriqui capable of reading and processing models in the nl format (compiled models generated from AMPL). However, if your goal is to use Muriqui with your C++ API, you do not need to install this library. Nevertheless, we strongly recommend the use of the built-in ASL executable whenever possible, since the AMPL language greatly facilitates the writing and manipulation of optimization models, besides avoiding derivative coding. But if you still have to opt for using the API, for example, to integrate another software project, you should be aware that you will need to correctly code the first and second order derivatives of each nonlinear term $f(x)$ and $g_i(x)$.

4.1.1 Installing the ASL

The ASL can be downloaded at `http://www.netlib.org/ampl/solvers.tgz`. After unzipping the downloaded file, go into the folder generated and run the configuration script with the command:

```
> configure
```

Go into the directory generated and run the command "make". It is worth mentioning that it is always good to perform an optimized compilation for performance gain purposes. So if you can, add optimization flags for the compiler in the make file, such as "-O3" or "-march=native". If you are not familiar to these flags or cannot add them for some reason, please ignore this comment. Run:

```
> make
```

If the execution succeeded and you have not changed any default options, a file called "amplsolver.a" should be generated. To conform to the default library names, we recommend that you rename this file to "libamplsolver.a".

4.2 MILP Solvers

To run some MINLP algorithms in Muriqui, it is necessary to solve Mixed Integer Linear Programming (MILP) subproblems that can be generated. As solving MILP problems is outside the scope of Muriqui, the installation of at least one of the following supported MILP solvers is required:

- Cplex (most recommended);
- Gurobi (recommended);
- Xpress (midterm recommended);
- Mosek (little recommended);

- GLPK (less recommended).

In most of our tests, we have used Cplex to solve the MILP subproblems generated by Muriqui. For this reason, the chances of some interface bug with Cplex are lower. This is the main reason why this is the most recommended solver, besides the fact that a license for academic use is easy to obtain. However, if you believe that Gurobi or Xpress may work best for you, you are encouraged to try them out. If obtaining a commercial software license is out of your reach or desire, you still have, as a last alternative, the option of using GLPK, which is a free solver. However, GLPK is practically not recommended because it is not thread safe, which can hinder the execution in some contexts. If you are aware that GLPK has become thread safe, please let me know and we will upload its recommendation rating.

After choosing one of the supported MILP solvers, you should perform the installation according to the instructions given by its developers. After installation, you should be able to generate a C executable program using the API of the solver chosen to proceed with the requirements for the Muriqui installation (try compiling and running the codes in C/C++ of the examples distributed with the software chosen).

4.3 NLP Solvers

To run some MINLP algorithms in Muriqui, it is necessary to solve Nonlinear Programming (NLP) subproblems that can be possibly generated. As solving NLP problems is also outside the scope of Muriqui, the installation of at least one of the following supported NLP solvers is required:

- Mosek (if chosen, at least one of the following solvers should be chosen as well);
- Ipopt;
- Knitro;

We observe that we have a slight preference for Mosek, because it is able to handle more easily the separable terms of problem (P), and because it is easy to obtain an academic license for it. However, you are welcome to try out the other solvers, especially if you have a Knitro license. For those who do not want or cannot use commercial software, the option indicated is the free solver Ipopt, which may, in some cases, even work better than the others. It is important to note that if you choose Mosek, we still recommend installing Ipopt, as Mosek do not address non-convex problems (not even to search for local optimal). So if the user is interested in addressing non-convex MINLP problems, Ipopt or Knitro are the proper choices.

After choosing one or more of the supported solvers, the user should install the solvers as instructed by their developers. After the installation, the user should be able to generate an executable program in C using the API of the solver chosen to proceed with the requirements for the Muriqui installation (try compiling and running the codes in C/C++ of the examples distributed with the software chosen).

5 Installing Muriqui Optimizer

5.1 Downloading Muriqui Optimizer

The source code of Muriqui can be downloaded from `http://www.wendelmelo.net/softwares`. Before proceeding, it is necessary to install the software required for the correct use of Muriqui, as explained in Section 4.

5.2 Compiling Muriqui Optimizer

After downloading and unzipping Muriqui into a folder of your preference, referred to here as $MURIQUIDIR, the next step is to configure the compilation to use the options and libraries chosen. This configuration is done by editing the files WAXM_config.h and make.inc, present in $MURIQUIDIR, in an integrated and consistent way, as explained below:

5.2.1 Editing the WAXM_config.h file

The WAXM_config.h file contains definitions for the correct compilation of Muriqui. The flags indicate the presence of specific libraries and functions to be used. For example, to compile Muriqui integrated to the AMPL system through the ASL library, the user should edit the line:

```
#define WAXM_HAVE_ASL                    0
```

changing it to:

```
#define WAXM_HAVE_ASL                    1
```

The user should also edit the lines corresponding to the MILP and NLP solvers used. For example, if the user wants to compile Muriqui with Cplex, Gurobi, Mosek and Ipopt, he/she should edit the lines:

```
#define WAXM_HAVE_CPLEX                  0
#define WAXM_HAVE_GUROBI                 0
#define WAXM_HAVE_MOSEK                  0
#define WAXM_HAVE_IPOPT                  0
```

changing them to:

```
#define WAXM_HAVE_CPLEX                  1
#define WAXM_HAVE_GUROBI                 1
#define WAXM_HAVE_MOSEK                  1
#define WAXM_HAVE_IPOPT                  1
```

Note that it is possible to compile Muriqui with several MILP and NLP solvers simultaneously, allowing the user to choose among these software only when applying the algorithms implemented in Muriqui.

Other flags are: `WAXM_HAVE_CLOCK_GETTIME`, which indicates the presence of the C function clock_gettime (usually present in Unix systems); `WAXM_HAVE_POSIX`, which indicates the presence of libraries described in the Posix standard (also present in Unix systems).

5.2.2 Editing the make.inc file

The make.inc file contains settings for the compilation of Muriqui, such as the compiler to be used, compilation flags, and the location of required inclusion files and libraries. Please note that special attention must be paid when editing this file so that it conforms to WAXM_config.h. For example, if WAXM_config.h contains the definition `#define WAXM_HAVE_CPLEX 1`, which means that Muriqui is configured to be compiled with Cplex. Therefore, the make.inc file should contain the definition of the CPLEXINC variable with the inclusion definitions and CPLEXLIB with the library link definitions. By default, make.inc contains commented lines with the definition of these variables, so it is possible to uncomment the respective lines and make the necessary changes in order to enable the correct compilation. For example, assuming that Cplex is installed in the $CPLEXDIR folder, a possible setting for these variables would be:

```
CPLEXINC = -I${CPLEXDIR}/cplex/include/ilcplex/
CPLEXLIB = -L${CPLEXDIR}/cplex/lib/x86-64_linux/static_pic/ -lcplex -lpthread -pthread
```

Note that the above definitions may vary depending on the system settings. The user should make similar definitions for each software library that is set to 1 in WAXM_config.h, including the ASL. In our example in Section 5.2.1, we configure Muriqui to be compiled also with Gurobi, Mosek and Ipopt. Therefore, the user would also need to define the variables $GUROBIINC, $GUROBILIB, $MOSEKINC, $MOSEKLIB, $IPOPTINC, and $IPOPTLIB, in the make.inc file (the user can uncomment the respective lines and make the necessary edits). In most cases, to integrate ASL in the Muriqui compilation, it should be sufficient to follow the steps described in Section 4.1, change the definition of `WAXM_HAVE_ASL` to 1 (Section 5.2.1), and change the line in make.inc with the definition of the $ASLDIR variable, with the value of the folder where the libamplsolver.a file is located.

5.3 Running the Makefile

Once the user has made the correct configuration of the compilation as described in the previous subsections, he/she is ready to run the program make in $MURIQUIDIR by typing:

```
> make
```

If the configuration was correctly done, the executable file muriqui and the library libmuriqui are generated. {a, lib}. The executable muriqui is built to be used in an integrated way to the AMPL system or to read .nl files (compiled AMPL models). Therefore, it only makes sense to use it if the ASL library has been included in the compilation. The library libmuriqui contains definitions and routines available in the Muriqui API. To make sure that the compilation has succeeded, the user may run the examples distributed with the source code of Muriqui (Section 5.4).

If the compilation was not successful, the user may try editing the variable definitions in make.inc or WAXM_config.h. The user may feel free to also ask for help on the Muriqui mailing list. We will try to assist him/her in this task as much as possible.

5.4 Running examples

5.4.1 Via AMPL

The file t4gross.nl present in $MURIQUIDIR contains the compilation of an AMPL model. To run it, the user should call the executable muriqui compiled with ASL, passing the name of the file as a parameter:

```
> ./muriqui t4gross.nl

------------------------------------------------------------------------------
Muriqui MINLP solver, version 0.7.00 compiled at Dec  2 2017 20:40:39
by Wendel Melo, Computer Scientist, Federal University of Uberlandia, Brazil
collaborators: Marcia Fampa (UFRJ, Brazil), Fernanda Raupp (LNCC, Brazil)

if you use, please cite:
W Melo, M Fampa & F Raupp. Integrating nonlinear branch-and-bound and outer
approximation for convex Mixed Integer Nonlinear Programming. Journal of Global
Optimization, v. 60, p. 373-389, 2014.
------------------------------------------------------------------------------

muriqui: parameter 1: t4gross.nl
```

```
muriqui: input file: t4gross.nl
muriqui: Reading user model:
muriqui: Maximization problem addressed as minimization
muriqui: Done
muriqui: Reading user parameters:
muriqui: Trying reading algorithm choice file muriqui_algorithm.opt . Not found!
muriqui: Done
muriqui: Trying read input parameter file muriqui_params.opt
muriqui: Done
muriqui: preprocessing...

Starting Branch-and-Bound algorithm

muriqui: milp solver: cplex, nlp solver: mosek
iter lb  ub  gap nodes at open

1  -13.5901 -5.32374 8.26636 2 0
muriqui: Appliyng Outer Approximation on the root
cpu time: 1.03897
cpu time: 1.04079
------------------------------------------------------------------------------
Problem: t4gross.nl Algorithm: 1004 Return code: 0 obj function: -8.0641361104
time: 0.70 cpu time: 1.04 iters: 21
------------------------------------------------------------------------------
An optimal solution was found! Obj function: 8.064136. CPU Time: 1.040791
```

Note that the output of each execution may vary a little, depending on the system and on the configuration.

5.4.2 Via API

The folder $MURIQUIDIR/examples contains examples of use of the Muriqui API in C++. The user may navigate to one of them, for example, $MURIQUIDIR/examples/t1gross and run make from this point. If the execution succeeds, the executable t1gross should be generated, which solves an example using several MINLP algorithms implemented in Muriqui:

```
> make
...
> ./t1gross
```

6 Implemented algorithms

For the time being, the following MINLP algorithms are implemented in Muriqui Optimizer and available for use:

6.1 Exact

- LP/NLP based Branch-and-Bound (LP/NLP-BB) [9];
- Hybrid Outer Approximation Branch-and-Bound (HOABB) [8];
- Outer Approximation (OA) [5, 6];
- Extended Cutting Plane (ECP) [10];

- Extended Supporting Hyperplane (ESH) [7];
- Bonmin Hybrid Branch-and-Bound (BHBB) [2].

We note that the application of a pure Branch-And-Bound algorithm can also be performed by choosing the HOABB algorithm and disabling the subprocedures of outer approximation.

6.2 Heuristics

- Feasibility Pump (FP) [4];
- Diving heuristic [4];
- Outer Approximation based Feasibility Pump (OAFP) [3];
- Relaxation Enforced Neighborhood Search (RENS) [1];
- Integrality Gap Minimization Heuristic 1 (IGMA1), to be submitted for publication;
- Integrality Gap Minimization Heuristic 2 (IGMA2), to be submitted for publication.

7 Which algorithm should I use?

Since Muriqui provides several MINLP algorithms to be applied, a natural question is how to choose the algorithm to be applied. Although the user is encouraged to perform various tests in order to determine the best approach for his/her specific application, we know that this may be impractical in some situations. Choosing the right algorithm for each situation can take a number of factors into account. For example, if the user only wants to quickly obtain a feasible solution, he/she should prefer to apply a feasibility heuristic (e.g. IGMA2, if the problem is binary, OAFP if the problem is convex, or FP for a more general problem).

If a problem needs to be solved to optimality, a lot of information can be taken into account. We have observed, for example, that linear approximation algorithms usually do well on convex problems. So, if you know in advance that your problem is in this category, an algorithm in that class would be the most natural choice. Otherwise, the best option may be HOABB. We discuss below some characteristics of the algorithms and their implementations that can help the user in this decision.

7.1 Linear approximation algorithms

7.1.1 LP/NLP based Branch-and-Bound (LP/NLP-BB)

In our tests, this algorithm has shown the best practical performance on convex problems. However, the current implementation of Muriqui only allows its application on a single Thread, which means that it cannot take advantage of possible multiple processing units available on the computer used. If the user wants to solve a convex MINLP problem on a multiprocessor computer and want to make the best possible use of the hardware, another linear approximation algorithm such as OA or ECP may be more appropriate. In addition, for now, this algorithm can only be applied if the Cplex or Gurobi API is available in the execution environment.

7.1.2 Outer Approximation

This algorithm may be the most suitable in general situations where the user wishes to solve a convex problem on a computer with multi-processors. However, the OA algorithm solves, at each iteration, one or two NLP problems, which are subproblems constructed from (P). If for some reason the user suspects that solving continuous relaxation or any of the (P) subproblems requires too much effort, it may be a good idea to test algorithms such as ECP or ESH.

7.1.3 Extended Cutting Plane

The ECP algorithm is characterized by not solving any NLP subproblems during its execution. This algorithm also has the friendly feature of being a totally first order method, which means that it does not use any information from second derivatives or even from approximations of them. If the user suspects that his/her (convex) MINLP problem has many computationally expensive second-order derivatives to be calculated, or if NLP packages have difficulty in solving their continuous relaxation, this may be the most appropriate algorithm. If the user is using Muriqui through the API, where the derivatives need to be coded, the user does not need to build callbacks for the calculation of the second order derivatives, in case this algorithm is chosen. As OA, the ECP algorithm is also enabled to use all processors available in the hardware.

7.1.4 Extended Supporting Hyperplane

The ESH algorithm can be seen as a good alternative to the ECP algorithm. However, the current implementation in Muriqui needs to solve an NLP subproblem at the beginning of the algorithm, which may require the computation of second order derivatives or their approximations. From this point, the ESH algorithm proceeds as a first-order method as well. Preliminary tests showed some superiority of ESH when compared to ECP, on problems with low percentage of linear constraints. As OA and ECP, ESH is also enabled to use all processors available in the hardware.

7.1.5 Bonmin Hybrid Branch-and-Bound

This algorithm is the hybrid Branch-and-Bound implemented in the solver Bonmin [2]. It is based on the application of LP/NLP-BB with the use of OA and the resolution of continuous NLP relaxations of subproblems through the enumeration tree. In our tests, our implementation of this algorithm failed to outperform the pure LP/NLP-BB. However, if you know in advance that the Bonmin implementation performs well on your (convex) MINLP problem, this algorithm may be a good choice (in any way, we recommend the user to also test pure LP/NLP-BB in this case). Since this algorithm is based on the execution of the LP/NLP-BB iterations, its implementation in Muriqui has the same usage restrictions as the latter (see Subsection 7.1.1).

7.2 Hybrid Outer Approximation Branch-and-Bound

This option seems to be the most suitable for handling non-convex problems, since approximation algorithms are strongly based on convexity, and not rarely, declare infeasibility for non-convex feasible problems. It should be noted, however, that this algorithm also does not guarantee optimality when solving non-convex problems, and can only be used as an heuristic. This algorithm is based on the application of the non-linear Branch-And-Bound with the recurrent application of OA. By adjusting its parameters, this OA application

can be disabled, thus making the algorithm a pure non-linear Branch-And-Bound. The difficulty in solving NLP relaxations results in many of the cases observed, in linear approximation algorithms to perform better than HOABB. However, if a situation appears where linear approximation algorithms present poor performance, adopting HOABB may be a good choice.

Referências

[1] Timo Berthold. Rens. *Mathematical Programming Computation*, 6(1):33–54, Mar 2014.

[2] Pierre Bonami, Lorenz T. Biegler, Andrew R. Conn, Gérard Cornuéjols, Ignacio E. Grossmann, Carl D. Laird, Jon Lee, Andrea Lodi, François Margot, and Nicolas Sawaya. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, May 2008.

[3] Pierre Bonami, Gérard Cornuéjols, Andrea Lodi, and François Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119:331–352, 2009. 10.1007/s10107-008-0212-2.

[4] Pierre Bonami and João Gonçalves. Heuristics for convex mixed integer nonlinear programs. *Computational Optimization and Applications*, pages 1–19, 2008. 10.1007/s10589-010-9350-6.

[5] Marco Duran and Ignacio Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986. 10.1007/BF02592064.

[6] Roger Fletcher and Sven Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994. 10.1007/BF01581153.

[7] Jan Kronqvist, Andreas Lundell, and Tapio Westerlund. The extended supporting hyperplane algorithm for convex mixed-integer nonlinear programming. *Journal of Global Optimization*, 64(2):249–272, 2016.

[8] Wendel Melo, Marcia Fampa, and Fernanda Raupp. Integrating nonlinear branch-and-bound and outer approximation for convex mixed integer nonlinear programming. *Journal of Global Optimization*, 60(2):373–389, 2014.

[9] Ignacio Quesada and Ignacio E. Grossmann. An lp/nlp based branch and bound algorithm for convex minlp optimization problems. *Computers & Chemical Engineering*, 16(10-11):937 – 947, 1992. An International Journal of Computer Applications in Chemical Engineering.

[10] Tapio Westerlund and Frank Pettersson. An extended cutting plane method for solving convex minlp problems. *Computers & Chemical Engineering*, 19, Supplement 1(0):131 – 136, 1995. European Symposium on Computer Aided Process Engineering.