Lurupa
Rigorous Error Bounds in Linear Programming

Christian Keil

Hamburg University of Technology, Institute for Reliable Computing
c.keil@tu-harburg.de

Abstract. Linear Programming has numerous applications. Recently it has been shown that many real world problems exhibit numerical difficulties due to ill-conditioning.
This paper describes Lurupa, a software package for computing rigorous bounds for the optimal value of a linear program. The package can handle point and interval problems. Numerical experience with the Netlib lp library is given.

Keywords. linear programming, rigorous error bounds, Netlib lp library, interval arithmetic

1 Introduction

It is well known that the errors introduced by floating point arithmetic affect the results of numeric computation. It is also known that the degree of influence depends on the condition number of the problem to be solved. What is less known is the fact, that for seemingly simple problems like linear programming the condition can be very poor even for non artificial, real world problems.

In a recent paper by Ordóñez and Freund [1] the authors show that 71% of the linear programs in the Netlib lp library [2] exhibit numerical difficulties due to ill-conditioning. This emphasizes the need for verification tools for these kinds of problems.

One approach to this is to use rational arithmetic to verify the optimality of the returned solution. This has been done for example by Gärtner [3]. He focuses on problems where either the number of constraints or variables is small. While this is common for problems from computational geometry, it is not common for linear programming in general. In fact only a handful of problems from the Netlib approximately satisfy this requirement. For the other problems his method, which utilizes an explicit inverse, is not applicable. Another variant of using rational arithmetic was investigated by Dhiflaoui et al [4]. They implemented methods that verify the primal or dual feasibility of a basis index set and an exact lp-solver that can start at a given basis or from scratch. The start basis can be taken from an approximate solver. This approach is applicable to general linear programming problems. A tool which only verifies the optimality of an approximate solution was described by Koch [5].

Dagstuhl Seminar Proceedings 05391
Algebraic and Numerical Algorithms and Computer-assisted Proofs
http://drops.dagstuhl.de/opus/volltexte/2006/445

The drawback of using rational arithmetic, however, is that it is only applicable to problems with a rational solution. While this is certainly the case for linear programming, for semidefinite programming for example it is not. Second no sensitivity analysis is performed. Computing the exact solution does not guarantee that it is meaningful for a physical problem.

All of the above problems of using rational arithmetic can be addressed with tools using interval arithmetic. Lurupa is such a tool designed to compute rigorous bounds for the optimal value of a linear program. In contrast to rational arithmetic it allows uncertainties in the input data. The computational complexity is an additional benefit of the algorithms implemented in Lurupa with respect to branch–and–bound frameworks for global optimization. The rigorous lower bound can in most cases be computed in $O(n^2)$ operations where n is the number of variables. This is the same order of complexity which is required to solve subproblems unverified using hot-start facilities. Hence a rigorous branch–and–bound algorithm should be slowed down at most by a constant factor. Notice that obtaining the lower bound by a verification of the Karush–Kuhn–Tucker conditions or the Fritz–John conditions (see Kearfott [6] and Hansen and Walster [7]) would require $O(n^3)$ operations and slow down the algorithm at least by a factor of n. A generalization of the ideas to the semidefinite case along with numerical experience can be found in [8].

For describing Lurupa we will start with a look at the theory behind the computations done in the package. Then we will investigate the software itself, the architecture and typical usage. Following is a survey of the numerical experience with the Netlib lp library. Finally we will take a look at some limitations and future work.

2 Theory

The algorithms to compute the rigorous bounds for the optimal value that are implemented in Lurupa are based on the ones developed by Jansson [9]. They are modified with respect to the set of variables that are solved for to satisfy the constraints. In Jansson's paper two theorems are presented, which are repeated here without proof. The idea is to derive bounds for the optimal value from boxes that are verified to contain feasible points. These boxes are obtained iteratively by the solution of slightly perturbed linear programs.

To investigate the theorems let us look at a linear program of the form

$$\begin{aligned} f^* := \quad & \min \quad c^T x \\ & s.t. \quad Ax \leq a \\ & \qquad Bx = b \\ & \qquad \underline{x} \leq x \leq \overline{x}. \end{aligned} \tag{1}$$

We can describe this linear program with the parameter tuple $P := (c, A, a, B, b)$ and the simple bounds $\underline{x}, \overline{x}$. Some or all simple bounds may be infinite; that is

$\underline{x}_i = -\infty$ and $\overline{x}_i = \infty$ is allowed. The linear program's dual is

$$\begin{array}{rll} f^* := & \max & a^T y + b^T z + \underline{x}^T u + \overline{x}^T v \\ & s.t. & A^T y + B^T z + u + v = c \\ & & y \leq 0, u \geq 0, v \leq 0. \end{array} \tag{2}$$

To deal with uncertainties in the input data, we can substitute the elements of P with interval parameters leading to interval problems $\mathbf{P} := (\mathbf{c}, \mathbf{A}, \mathbf{a}, \mathbf{B}, \mathbf{b})$. We do not consider uncertainties in the simple bounds as these are often exactly known such as the positiveness of variables.

Theorem 1 (Lower Bound). *Given an interval linear program* $\mathbf{P}$ *and simple bounds* $\underline{x} \leq \overline{x}$. *Suppose interval vectors* $\mathbf{y} \leq 0$, $\mathbf{z}$ *satisfy*

1. *for all free* x_j *(i.e.,* $\underline{x}_j = -\infty, \overline{x}_j = \infty$*) and all* $A \in \mathbf{A}$, $B \in \mathbf{B}$ *there exists* $y \in \mathbf{y}$, $z \in \mathbf{z}$ *such that*

$$c_j - (A_{:j})^T y - (B_{:j})^T z = 0$$

   holds, and
2. *for all variables* x_j *bounded on one side only the defects*

$$\mathbf{d}_j := c_j - (A_{:j})^T \mathbf{y} - (B_{:j})^T \mathbf{z}$$

   are nonnegative if the variable is bounded from below and nonpositive if it is bounded from above.

Then $\mathbf{y}, \mathbf{z}$ *contain a dual feasible solution* $y(P), z(P)$ *for each* $P \in \mathbf{P}$, *and a lower bound for the optimal value can be computed as*

$$\inf_{P \in \mathbf{P}} f^*(P) \quad \geq \quad \underline{f}^* := \inf\{\mathbf{a}^T \mathbf{y} + \mathbf{b}^T \mathbf{z} + \sum_{\underline{x}_j \neq -\infty} \underline{x}_j \mathbf{d}_j^+ + \sum_{\overline{x}_j \neq \infty} \overline{x}_j \mathbf{d}_j^-\}. \tag{3}$$

Theorem 2 (Upper Bound). *Given an interval linear program* $\mathbf{P}$ *and simple bounds* $\underline{x} \leq \overline{x}$. *Suppose interval vector* $\mathbf{x}$ *satisfies*

$$\mathbf{A}\mathbf{x} \leq \mathbf{a}, \quad \underline{x} \leq \mathbf{x} \leq \overline{x},$$

and for all $B \in \mathbf{B}$, $b \in \mathbf{b}$ *exists* $x \in \mathbf{x}$ *with*

$$Bx = b.$$

Then $\mathbf{x}$ *contains a primal feasible solution* $x(P)$ *for each* $P \in \mathbf{P}$, *and an upper bound for the optimal value can be computed as*

$$\sup_{P \in \mathbf{P}} f^*(P) \quad \leq \quad \overline{f}^* := \max\{\mathbf{c}^T \mathbf{x}\}. \tag{4}$$

Moreover, if the objective function is bounded from below for every linear program with input data $P \in \mathbf{P}$, *then each problem has an optimal solution.*

3 Software

Lurupa was designed with modularity and flexibility in mind. The aim is to provide a fast implementation of rigorous algorithms for linear programming problems. These shall be available as standalone versions and as a library to be integrated into larger frameworks. The implementation is in ANSI C++.

3.1 Architecture

The overall architecture is depicted in Figure 1. The main work is performed by a computational core, which uses the PROFIL/BIAS library [10] for the rigorous computations. This core is instructed either via the command line client or using the API, that is directly calling the methods exposed by the core. To do the approximative computations the core itself accesses arbitrary linear programming solvers via wrapper classes with a common interface. Beside these components are the classes for reporting and model storing.

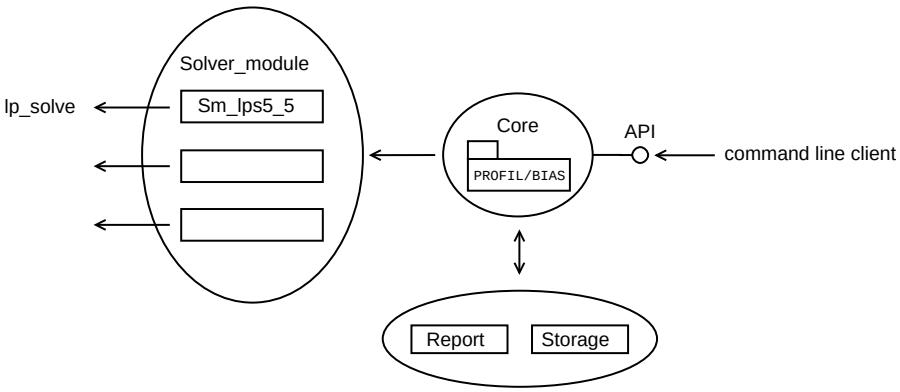

Fig. 1. Architecture

Taking a tour of the essential parts and starting with the computational core, we see in Figure 2 a UML Class diagram of the actual worker class `Lurupa`. The main routines to use the core are `set_solver_module`, `read_lp`, `solve_lp`, `lower_bound`, and `upper_bound`. The former two are responsible for setting up the environment. That is selecting a solver module and thus a linear programming solver and reading the linear program itself. To represent uncertainties in the model, the parameters can be inflated to intervals with a specified relative radius. With `solve_lp` the solver is instructed to compute an approximate solution to the problem. The subsequent verification is performed by the last two methods, which compute the rigorous lower and upper bound for the optimal value. To fine-tune the computations the remaining methods may be used to

change algorithm parameters. For details concerning the role of the parameters refer to Jansson [9]. The reports can be customized via the `Report` class. Calling `set_verbosity` adjusts the verbosity level of displayed messages. The two remaining parameters specify whether messages are printed with prepended time and whether intermediate vectors and matrices are stored to disk for later examination.

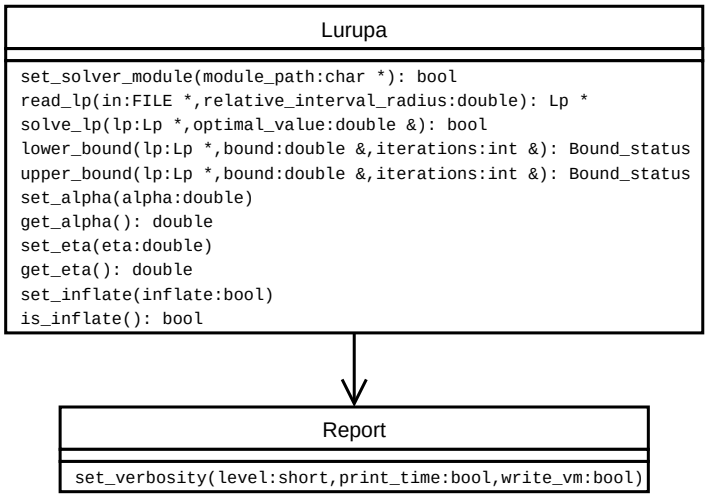

Fig. 2. Core

Looking closer at the solver modules in Figure 3, we find the common interface `Solver_module` with the general methods `read_lp`, `solve_original`, `solve_primal_perturbed`, `solve_dual_perturbed`, and `set_module_options`. Reading an lp from a file is the task of `read_lp`. An object of the storage class is initialized with the model from the specified file. The lp parameters can be inflated to intervals and the algorithm parameter `eta` is adjusted to the model. The methods to solve the original and primal and dual perturbed models have two parameters. All three need the model to be solved. Solving the original lp returns the optimal value in the parameter `optimal_value`. The perturbed methods require the perturbation to be applied. With `set_module_options` solver specific settings can be changed in a command line argument way.

These methods are inherited and implemented by the solver specific modules, depicted by the exemplary lp_solve [11] module `Sm_lps5_5`. The solver modules have to translate the above calls to corresponding calls to the solver. As each solver stores the model and associated data in a different format they also have to translate these structures to the representation of Lurupa and keep track of any additional solver specific information. This information can be attached to Lurupa's model representation.

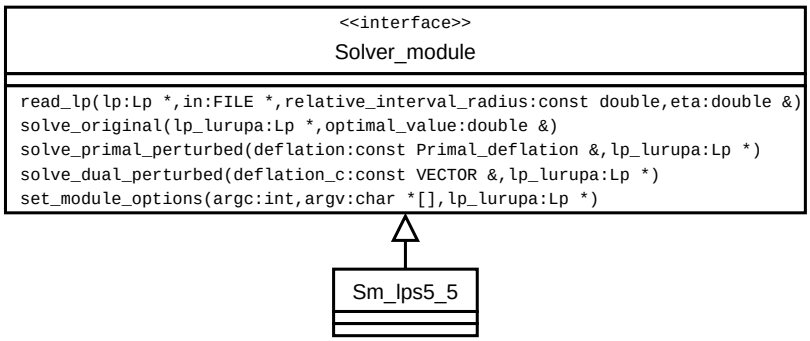

Fig. 3. Solver module

The final missing piece is the `Lp` class for storing the model as seen in Figure 4. It stores the tuple $\mathbf{P}$ and $\underline{x}, \overline{x}$, along with meta data like the name of the model, and the number and indices of the free variables. Further it stores the information about the approximate primal and dual solutions $\mathbf{x}, \mathbf{y}, \mathbf{z}$. The dual solution is split into a part corresponding to less equal– and equal–constraints. Storing solver specific information is shown in the case of lp_solve with the mapping of less equal– and equal–constraint indices to overall constraint indices, `mp_le_con` and `mp_eq_con`, respectively.

3.2 Usage

The usage of Lurupa depends on the actual environment and task. One way to use the software is via the command line client the other directly via the API.

Using the software in a stand-alone fashion with the command line is the easier part without the need for further programming. The command line client displays some meta data from the model like the name and direction of optimization, formats the results returned by the core, and adds time ratios and relative accuracies of the bounds. All the options that are available are selected through the use of command line parameters. These are divided into general and solver specific parameters.

The main general parameters are `-lp <path/to/lp>`, `-lb`, and `-ub`, which specify the lp to be processed and request the lower and upper bound to be computed, respectively. Summarizing the general parameters are displayed in Table 1.

To select a solver module the `-sm <path/to/solver module>` parameter is used. Further parameters depend on the selected module. They include for example algorithm settings for the solver and timeout settings. The parameters available with the lp_solve module are contained in Table 2.

A typical call with the command line client is

```
lurupa -sm Sm_lps5_5 -lp lp.mps -lb -ub -v3
```

-alpha *d*	Set algorithm parameter alpha to *d*.
-csv <*file*>	Append the results to the csv file <*file*>[*.csv*], with the extension being appended if not present.
-eta *d*	Set algorithm parameter eta to *d*.
-i *d*	Compute bounds for an interval problem derived from the one specified. Change all parameters to intervals with a relative radius of *d*.
-inflate	Try inflating the model if a perturbed one seems to be infeasible.
-latex <*file*>	Append the results to the latex table in the file <*file*>[*.tex*] with the extension being appended if not present.
-lb	Compute the lower bound.
-lp <*file*>	Read the linear program to be processed from <*file*>. Must be in a format that can be interpreted by the chosen solver module. If this switch is not present, the model is read from stdin.
-sm <*file*>	Use the solver module <*file*> to solve the linear programs.
-t	Prepend time information to messages.
-ub	Compute the upper bound.
-vn	Select verbosity level: **-v0** No messages **-v1** Errors **-v2** Warnings (default) **-v3** Brief **-v4** Normal **-v5** Verbose **-v6** Full
-write_vm	Write intermediate vectors and matrices to disk.

Table 1. General command line parameters

-sm,timeout,<*sec*>	Set solver timeout in seconds.
-sm,vn	Set solver verbosity: **v0**: NEUTRAL **v1**: CRITICAL **v2**: SEVERE **v3**: IMPORTANT (default) **v4**: NORMAL **v5**: DETAILED **v6**: FULL

Table 2. Lp_solve module command line options

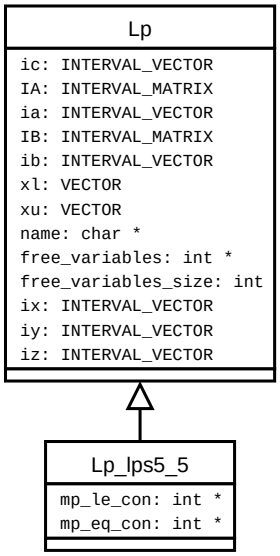

Fig. 4. Lp

This call uses solver module `Sm_lps5_5` to processes the model `lp.mps`. The lower and upper bound for the optimal value are computed. Verbosity is set to level 3, which is 'Brief', algorithm parameters are left at their default values.

The integration of Lurupa into larger frameworks is possible using the package as a library through the API. While the command line client adds some output there is no further difference in functionality or available features to the command line client.

Lurupa exposes its functionality through the core `Lurupa` class. Looking back at Figure 2, the example from above would look like Listing 1 when done via the API. After the calls to `lower_bound` and `upper_bound` the lower and upper bound are contained in `lbound` and `ubound`, respectively. The value of `literations` and `uiterations` indicates the number of necessary algorithm iterations.

4 Numerical Experience

The Netlib lp library of numerous problems from practical background is a well fitting collection of test problems. Here only an overview of our numerical experience is given. Detailed results including interval problems can be found in [12].

Ordòñez and Freund [1] defined a condition number for a linear program based on the distances to the nearest primal infeasible and dual infeasible problem, ρ_p and ρ_d, respectively. The condition number follows as the scale invariant

```
Lurupa l;
l.set_solver_module("Sm_lps5_5");
l.report.set_verbosity(3, false, false);

FILE *in = fopen("lp.mps", "r");
Lp lp = l.read_lp(in, 0);

double optimal, lbound, ubound;
int literations, uiterations;
l.solve_lp(lp, optimal);
l.lower_bound(lp, lbound, literations);
l.upper_bound(lp, ubound, uiterations);
```

Listing 1. API Usage

reciprocal of the minimal distance to infeasibility. The results show that the lower and upper bound is computed if the distance to dual and primal infeasibility, respectively, is greater than 0.

Table 3 contains an overview of the results obtained in [12]. For 76 out of 89 problems a finite lower bound could be computed. Only 3 of the remaining problems have a distance to dual infeasibility being greater than 0. The others are dual ill-posed. Examining the upper bound, 35 problems yield a finite one. From the remaining problems only 2 have a distance to primal infeasibility being greater than 0. It seems reasonable that bounds for the remaining problems with a distance to infeasibility greater than 0 can be computed by fine tuning the algorithms. In 32 cases both bounds were finite. For each of these groups the table contains the median values for the relative accuracy

$$\mu(a,b) := \frac{|a-b|}{\max\{1, \frac{|a+b|}{2}\}}$$

and the required time ratios. The time to solve the problem approximately is denoted by t_{f^*}, the times to compute the bounds by $t_{\underline{f}^*}$ and $t_{\overline{f}^*}$.

The median values of the relative accuracy show us approximately 8 correct digits for all three groups, which is close to optimal when taking into account the set stopping tolerance 10^{-9} of the used lp-solver. While the lower bound is cheaper than solving the problem itself, the upper bound is more expensive. This can be attributed to the equation systems that have to be solved when computing the upper bound.

5 Limitations and Future Work

At the moment the interval representation of the linear program is dense due to PROFIL/BIAS not supporting sparse matrix structures. I am working on an implementation of such structures to be available in a future version of PROFIL/BIAS.

76 finite lower bounds

$$\text{med}(\mu(\underline{f}^*, f^*)) = 2.183e-8 \qquad \text{med}(t_{\underline{f}^*}/t_{f^*}) = 0.500$$

35 finite upper bounds

$$\text{med}(\mu(\overline{f}^*, f^*)) = 8.034e-9 \qquad \text{med}(t_{\overline{f}^*}/t_{f^*}) = 5.250$$

32 finite pairs

$$\text{med}(\mu(\overline{f}^*, \underline{f}^*)) = 5.620e-8$$

Table 3. Overview of Netlib results

Of great interest is also the connection to the work of Ordóñez and Freund. They show the distances to infeasibility to be computable as the minimal objective value of a number of linear programs. This makes Lurupa applicable to compute verified distances to infeasibility and thus verified condition numbers for linear programs. Connected is the topic of certificates for infeasibility and unboundedness, which will be implemented in Lurupa.

Ordòñez and Freund also observed that preprocessing has a considerable impact on the condition number of the problem. Fourer and Gay [13] showed, however, that preprocessing can change the state of a linear program from feasible to infeasible and vice versa. This suggests investigation of verified preprocessing.

The ideas used in Lurupa for well-posed linear programs can be extended to ill-posed problems. Also a generalization to arbitrary convex optimization problems is possible (see Jansson [14], [15]).

References

1. Ordóñez, F., Freund, R.: Computational experience and the explanatory value of condition measures for linear optimization. SIAM J. Optimization **14** (2003) 307–333
2. Netlib: (Netlib linear programming library) http://www.netlib.org/lp.
3. Gärtner, B.: Exact arithmetic at low cost – a case study in linear programming. Computational Geometry **13** (1999) 121–139
4. Dhiflaoui, M., Funke, S., Kwappik, C., Mehlhorn, K., Seel, M., Schömer, E., Schulte, R., Weber, D.: Certifying and repairing solutions to large lps how good are lp-solvers? In: SODA. (2003) 255–256
5. Koch, T.: The final netlib-lp results. Technical Report 03-05, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraße 7, D-14195 Berlin-Dahlem, Germany (2003)
6. Kearfott, R.: Rigorous Global Search: Continuous Problems. Kluwer Academic Publisher, Dordrecht (1996)
7. Hansen, E., Walster, G.W.: Global Optimization Using Interval Analysis. Second edition edn. Pure and Applied Mathematics. Dekker (2003)

8. Jansson, C., Keil, C.: Rigorous Error Bounds for the Optimal Value in Semidefinite Programming. (2005) Submitted, and electronically published http://www.optimization-online.org/DB_HTML/2005/01/1047.html.
9. Jansson, C.: Rigorous Lower and Upper Bounds in Linear Programming. SIAM J. Optim. **14** (2004) 914–935
10. Knüppel, O.: PROFIL/BIAS and extensions, Version 2.0. Technical report, Inst. f. Informatik III, Technische Universität Hamburg-Harburg (1998)
11. Berkelaar, M., Notebaert, P., Eikland, K.: lp_solve. (World Wide Web) http://groups.yahoo.com/group/lp_solve.
12. Keil, C., Jansson, C.: Computational Experience with Rigorous Error Bounds for the Netlib Linear Programming Library. (2006) To appear in Reliable Computing, electronically published http://www.optimization-online.org/DB_HTML/2004/12/1018.html.
13. Fourer, R., Gay, D.M.: Experience with a primal presolve algorithm. In Hager, W.W., Hearn, D.W., Pardalos, P.M., eds.: Large Scale Optimization: State of the Art. Kluwer Academic Publishers Group, Norwell, MA, USA, and Dordrecht, The Netherlands (1994) 135–154
14. Jansson, C.: Termination and Verification for Ill-posed Semidefinite Programming Problems (2005) http://optimization-online.org/DB_HTML/2005/06/1150.html.
15. Jansson, C.: A rigorous lower bound for the optimal value of convex optimization problems. J. Global Optimization **28** (2004) 121–137