A COMPARISON OF SOFTWARE PACKAGES FOR VERIFIED LINEAR PROGRAMMING

CHRISTIAN KEIL *

Abstract. Linear programming is arguably one of the most basic forms of optimization. Its theory and algorithms can not only be applied to linear optimization problems but also to relaxations of nonlinear problems and branch-and-bound methods for mixed-integer and global optimization problems.

Recent research shows that against intuition bad condition numbers frequently occur in linear programming. To take this into account reliability is required. Here we investigate rigorous results obtained by verification methods. We will examine different current techniques and software tools for verified linear programming and compare numerical results for existing implementations.

Key words. linear programming, interval arithmetic, rigorous error bounds, comparison, software packages

AMS subject classifications. 90C05, 65G30, 65N15

1. Introduction.

Linear programming remains one of the most important forms of optimization in applications. As exemplified by Lovasz [28]: “If one would take statistics about which mathematical problem is using up most of the computer time in the world, then (not including database handling problems like sorting and searching) the answer would probably be linear programming”. That same year Eugene Lawler of Berkeley summarized: “It [linear programming] is used to allocate resources, plan production, schedule workers, plan investment portfolios and formulate marketing (and military) strategies. The versatility and economic impact of linear programming in today’s industrial world is truly awesome.” Looking at the documentation of collections of linear programming problems like the Netlib lp library [31] reveals such diverse applications as oil refinery problems, flap settings on aircraft, industrial production and allocation, and image restoration.

Rounding errors inevitably introduced by floating point arithmetic can well deteriorate the result of numerical computations. This is especially true for problems being ill-posed, which 71% of the linear programs in the Netlib collection are, as observed by Ordóñez and Freund [35].

The goal of this paper is to compare different optimization packages similar to the recent comparison of several state-of-the-art complete global optimization solvers by Neumaier, Shcherbina, Huyer, and Vinkó [34]. In our comparison, however, we are looking for rigorous results taking all rounding errors into account. We will examine different approaches and software packages and apply them to several random and real-world problems.

In the next section we will agree on the used notation. In Section 3 the techniques and software packages in question will be described. Section 4 discusses the justness of comparing such diverse methods. The ensuing Sections 5 and 6 detail the test environment and the numerical results, respectively. Finally some conclusions are given in Section 7.

2. Notation.

In the following we will treat the (*primal*) linear program (LP) of minimizing a linear objective function over a set of feasible points defined by linear

*Institute for Reliable Computing, Hamburg University of Technology, Schwarzenbergstraße 95, 21073 Hamburg, Germany (c.keil@tu-harburg.de).

constraints,

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

Relations between vectors are to be understood componentwise. We will denote the number of variables, equality, and inequality constraints by n, p, and m, respectively. Simple bounds, $\underline{x}_i$ and $\overline{x}_j$, may be infinite.

This LP's *dual* has the form

$$\begin{aligned} g^* := \max \quad & a^T y + b^T z + u^T \underline{x} + v^T \overline{x} \\ \text{s.t.} \quad & A^T y + B^T z + u + v = c \\ & y \leq 0, \ u \geq 0, \ v \leq 0. \end{aligned}$$

Using the convention $0 \cdot \pm\infty$ evaluates to 0, a finite dual objective value follows if and only if the components of u and v corresponding to infinite simple bounds are themselves 0. The dual has the important property that its objective value is always less than the primal one.

We will denote optimal primal and dual variables by x^* and y^*, z^*, u^*, and v^*. An approximation to the optimal value will be denoted by $\tilde{f}$, lower and upper bounds on the optimal value by $f^\triangle$ and $f^\triangledown$, respectively.

3. Available software. The available software capable of producing rigorous results for LPs can be categorized into four groups according to the algorithms they use. Algorithms for verified constraint programming, algorithms using a rational arithmetic, verified global optimization algorithms using interval arithmetics, and algorithms specifically designed to rigorously solve LPs in the presence of rounding errors and uncertainties in the input data.

3.1. Verified constraint programming. Algorithms for constraint programming search for points satisfying a given set of constraints. Rigorous versions return a list of interval vectors (boxes) that may contain feasible points and a list of boxes that are verified to contain satisfying points. These lists are exhaustive, if both are empty, the problem is claimed to have no feasible solution.

Constraint programming algorithms do not support the concept of an objective function. Nevertheless rigorous bounds for the optimal value of an LP can be derived in the following way. Assume we have an approximation $\tilde{f}$ and choose a small $\Delta > 0$. Now we apply a verified constraint programming algorithm to the original set of constraints, adding a constraint on the objective function of the form $c^T x \leq (1-\Delta)\cdot\tilde{f}$. If the algorithm recognizes this set of constraints to be infeasible, we know $(1-\Delta)\cdot\tilde{f}$ to be a rigorous lower bound on the true optimal value. A rigorous upper bound could be computed using a similar approach for the dual problem with the additional overhead of explicitly forming the dual in the first place. Obtaining an upper bound from a box verified to contain feasible points is considerably more involved from an algorithmic point of view. In the linear programming case the feasible points form a continuum, which is difficult to handle. Research into this area has only recently begun (see Neumaier [32], and Vu et al. [42, 43]).

An implementation of a verified constraint programming algorithm is RealPaver [11]. It uses a branch-and-prune algorithm, the pruning step merges constraint satisfaction techniques with the interval Newton method.

3.2. Algorithms using a rational arithmetic. Algorithms of this group exploit the fact, that the solution of an LP with floating point (rational) coefficients is itself rational. This has been done by Gaertner [9] for problems where either the number of variables or constraints does not go well beyond 30. Dhiflaoui et al. [6] combined the rational approach with an approximate standard LP solver. They build on the premise that the approximate optimal basis computed by a standard LP solver is close to the true optimal one, counting simplex steps. Starting from an approximate basis, they perform rational simplex steps until they arrive at the true optimum. Koch [25] provides an implementation just checking the optimality of a basis together with results on all Netlib problems. He remarks in cases where the basis proves to be suboptimal, increasing the precision of the approximate computations may help the LP solver to find the true optimal basis. This observation in turn has recently been used by Applegate et al. [2], who iteratively increase the precision of the approximate computations until the computed basis proves to be optimal.

A fully rational simplex method is exlp [22] by Kiyomi. The implementation by Koch, perPlex [24], checks the optimality of a given basis but does not offer any means to compute it in the first place or go on from there if it proves to be suboptimal. Applegate et al. published their ideas in the solver QSopt_ex [1].

3.3. Verified global optimization. Verified global optimization solvers handle problems with objective function and constraints defined by smooth algebraic expressions. Thus they can solve LPs rigorously. Their output consists of candidate and verified to be enclosures of feasible points and an enclosure of the optimal value or the verified claim that no feasible point exists.

Implementations are COSY [5], GlobSol [17], ICOS [27], and Numerica [40]. COSY uses a branch-and-bound scheme featuring a Taylor model arithmetic for the bounding step. Unfortunately on inquiry it was not possible to obtain a copy of COSY. The current policy of the project seems to deny researchers from the interval community access to the code. GlobSol combines an interval branch-and-bound procedure with additional techniques, such as automatic differentiation, constraint propagation, interval Newton methods, and additional, specialized techniques. Originally starting as a constraint programming algorithm, ICOS supports optimization since the current release (0.1 from May 2008). It is based on constraint programming, interval analysis, and global optimization techniques. In addition it uses safe linear relaxations together with the finite case of the rigorous bounds by Jansson [16] and Neumaier and Shcherbina [33] (see below). We will see later how this contributes to the results of ICOS.

3.4. Verified linear programming with uncertainties. Finally there are the algorithms specifically designed for verified linear programming in the presence of rounding errors and uncertainties in the input data. Considering rounding errors in linear programming goes back to Krawczyk [26]. His aim was to compute a rigorous enclosure of the optimal solution by verifying the optimality of a basic index set. From this, rigorous bounds on the optimal value can be derived easily. He also considered errors in the input data, allowing the algorithm to be applied to problems with interval data. His ideas were used and refined by Beeck [3] and Rump [39]. Jansson [15] introduced means to also handle degenerate solutions starting from a basic index set and applying a graph search method to a graph of the adjacent basic index sets. Requiring the solution of interval linear systems, all these methods share a computational work being cubic in $\min\{m+p, n\}$. Several years later, independently and at the same time Neumaier and Shcherbina [33] and Jansson [16] devised methods to rigorously bound the optimal value with a quadratic complexity. This is achieved by deriving the rigorous bounds from certain duality properties. The output here consists of rigorous enclosures of the optimal value and feasible but suboptimal points. Neumaier and Shcherbina did this for the case where finite simple bounds on all variables are known, Jansson also considered the case of unbounded and free variables.

In returning enclosures of feasible but suboptimal points, the character of the solutions returned by these algorithms differs considerably from the previous ones'. While not getting the optimal solution, the user obtains near optimal, feasible points in the well-posed case. In the ill-conditioned case wide or no bounds are returned. This indicates numerical instabilities and may point to inconsistencies in the model. Whether one kind of solution is superior to the other depends on the actual application.

Lurupa [20] belongs to this fourth category. Implementing the algorithms developed by Jansson [16], Lurupa tries to enclose feasible, near-optimal solutions for the primal and the dual problem. The iterative algorithm starts by solving the original problem with a standard LP solver. Based on the approximate solution, interval arithmetic is now applied to enclose a feasible point. If this succeeds, the rigorous bound on the optimal value follows with the range of the objective function over this enclosure. Otherwise the algorithm starts over, perturbing the LP to force the approximate solution further into the relative interior of the feasible region. In the presence of uncertainty in the input data, only approximate solutions to midpoint problems are necessary, and still a standard LP solver suffices. Lurupa implements verified certificates of infeasibility and unboundedness. Furthermore it is the only package in this comparison that computes verified condition numbers as defined by Renegar [36]. To do this, the algorithm is applied to Ordóñez and Freund's linear programming characterization of the distances to primal and dual infeasibility [35]. This is especially interesting when solving real-world applications as it allows to evaluate the underlying models.

Lurupa has proven capable of computing rigorous error bounds for the optimal value of LPs with several thousands of variables and constraints (see [21]). Roughly speaking, finite lower and upper bounds could be computed if and only if the respectively primal and dual problems were not ill-posed. For various problems of the Netlib LP library, lower and upper bounds were obtained, certifying the existence of optimal solutions.

Consisting of modules for the different functional parts, Lurupa is fully implemented in ISO C++ and easily extendible. The aim is to provide a fast implementation of the algorithms, available as a stand-alone and a library version to be integrated into larger frameworks. For the interval computations PROFIL/BIAS [23] is used. Solver modules are currently available for lp_solve [4] in different versions. Additional solver modules have to implement a common interface, their main function is transforming data between the representations used in Lurupa and in the solver.

It is interesting to note that few of these algorithms are specially devised for convex programming problems. This despite Nemirovski [30] and Vandenberghe and Boyd [41] observing that a large number of applications can be modelled as convex problems. Rockafellar [37] remarks: “In fact the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity”. Only Lurupa, VSDP [14]—a MATLAB package written by Jansson generalizing the approach to semidefinite programming—, and the exact rational algorithms fall into this category. The rational algorithms are of course only applicable to problems that can be solved in the rationals.

4. Apples and oranges?. Comparing these software packages is not an easy task. They are written for different purposes and deliver different kinds of output. There are certainly scenarios in which some of these packages fit and others do not. In the following comparison we want to apply all solvers to the same set of problems and see which problems the packages can solve. Therefore we have to define what solving in this comparison means.

We will say that a package solves a problem if it returns the aforementioned rigorous results. Enclosures have to be of reasonable width. We will not accept answers like $[-\infty, \infty]$ for all problems as solutions. The requirements on the width, however, depend on the application and the user. Even wide finite bounds verify at least feasibility, which approximate algorithms do not. Lurupa returns five infinite upper bounds for real-world problems in our test set. These are exactly the five problems of our test set for which Ordóñez and Freund [35] computed a distance to primal infeasibility of 0. This means an arbitrarily small perturbation of the problem data may render these problems infeasible. Thus the infinite upper bound exactly reflects the ill-posedness of the problem and is regarded as a solution.

While the results returned by the different solvers differ in character and demand, they offer a similar benefit to the user: the assurance of rigorosity. Whether this is achieved by providing the exact solution, an enclosure of it, or an enclosure of near optimal, feasible points is often secondary for real-world applications. And it's the rigorous bounds on the objective value that are mandatory for fully rigorous branch-and-bound algorithms.

The next question that arises is whether it is fair to compare general purpose algorithms with algorithms specifically targeted at linear programming. First all these packages are able to solve LPs so we will have a look at how they perform. Second we precisely want to see if it is necessary to make use of special structure in the problem, and we will use linear programming as a class of problems with special structure. And third solving LPs is rather easy compared with nonlinear problems. *If a general purpose algorithm cannot solve an LP of certain dimensions, it seems unlikely it will be able to solve nonlinear problems of that size.* Thus linear programming is a good benchmark also for nonlinear optimization packages.

5. Test environment. We will use three test sets to compare the software packages. The first one consists of random problems generated with Algorithm 1, which is similar to the one suggested by Rosen and Suzuki [38]. All random choices are uniformly distributed and integral.

Using this procedure the solution and optimal value are integral and known exactly, and the LP is non-degenerate. Dual degeneracy would lead to a continuum of primal optimal solutions, posing an additional challenge for the constraint satisfaction and the complete global optimization codes as already noted. *Degeneracy, however, is not a rare phenomenon, but rather common in linear programming and mathematical programming in general.* Often this is a result of inherent model properties as investigated by Greenberg [12]. Also the problems generated with this procedure are dense. Thus they do not indicate the size of sparse problems a package can solve. Especially the rational solvers have inherent problems with the growing number of nonzero coefficients in large dense problems. Nevertheless there are applications that naturally result in dense optimization problems. Several sparse and degenerate problems can be found in the other two test sets.

Algorithm 1 Random problems after Rosen and Suzuki

```
Given: $m, p, n$ with $p \le n \le m + p$
    1. Choose the components of the optimal solution
        • $x^*$ between $-9$ and $9$,
        • $z^*$, nonzero, between $-10$ and $10$, and
        • $n-p$ components of $y^*$, between $-10$ and $-1$, the remaining components
          of $y^*$ become 0.
    2. Set simple lower and upper bounds of $-10$ and $10$ on all variables. Choose
       constraint matrices $A$ and $B$ for the inequalities and equations, respectively,
       with coefficients between 0 and 10. Build the matrix of active constraints
       (i.e., $y_i^* \neq 0$)
           $$\begin{pmatrix} A_{active} \\ B \end{pmatrix},$$
       and add 1 to the diagonal elements, ensuring its regularity.
    3. Compute the right hand sides of the constraints
           $$a = Ax^* \qquad b = Bx^*,$$
       and increment $a_i$ belonging to inactive constraints (i.e., the corresponding
       value in $y^*$ is 0) by 1.
    4. Compute the coefficients of the objective function
           $$c = A^T y^* + B^T z^*$$
```

We generate problems with 5 to 1500 variables, with the same number of inequalities and the number of equations being half of that. For each triplet of number of inequalities, equations, and variables ($m = n$, $p = 0.5n$, n), three problems are generated to rule out the influence of problem properties that are present by coincidence. The problem instances used in the following can be found under `http://www.ti3.tu-harburg.de/~keil`.

The second test set contains the real-world problems with less than 1500 variables from the Netlib collection [31] and the *Misc* section of Meszaros collection [29] [1]. Since most of these problems have some infinite simple bounds, a large finite simple bound on the variables was set for ICOS and GlobSol, which do not handle infinite bounds. If possible, this bound was chosen to not change the feasible region. If not, it was chosen an order of magnitude larger than the approximate optimal solution's largest component.

Finally we will run the solvers on the infeasible problems in Meszaros *infeas* section with less than 1500 variables[2].

The following versions of the software packages will come to use:

[1] Netlib's *standgub* is removed from this set, as it uses a special feature of the MPS format that some solvers do not support. Further information can be found in the readme file contained in the Netlib lp collection. From Meszaros Misc collection, *model1* is removed as it does not contain an objective function, which circumvents the RealPaver approach to compute a rigorous bound. Also from Meszaros Misc, *zed* contains a second objective function, which is ignored in the following.

[2] *gas11* from this section is unbounded and thus removed, because only exlp and Lurupa have means to verify this.

- ICOS 0.1
- RealPaver 0.4
- exlp 0.6.0
- perPlex 1.01 in combination with lp_solve 5.5.0.6 to compute the approximate optimal basis
- QSopt_ex 2.5.0
- GlobSol released on November 22, 2003
- Numerica 1.0
- Lurupa 1.1 with lp_solve 5.5.0.6 solver module 1.1

All packages apart from ICOS and Numerica, which are not available in source, are compiled with a standard gcc 4.1.2 [8]. Where applicable a configuration with compiler optimization is chosen. The computations are performed on an Intel XEON with 2.2 GHz and 3 GByte of RAM available to the process under SUSE Linux 10.0. The timeout threshold differs from problem to problem, a solver run is aborted if it takes more than 100 times longer than the fastest solver on this problem. A run includes everything from reading the LP to reporting the solution. It does not include the time to compute the initial approximate $\tilde{f}$ necessary for RealPaver, but an approximate solver is in median orders faster than RealPaver and the time can therefore be neglected.

The numerical results of Numerica will not be displayed here because it was sold dongled to the hardware, and thus we cannot test it on the same platform. But taking the differences between the platforms into account and setting appropriate timeouts, Numerica is only able to solve problems with 5 variables and 5 inequalities. Adding variables or constraints causes Numerica to timeout. Of the real-world problems Numerica solves *kleemin3* to *kleemin6*, on all other ones it runs out of memory ($\approx$ 700 MByte in this environment).

The programs will be run with their default settings for strategy and stopping tolerances. As lp_solve is no integral part of perPlex, we may try some different settings like scaling options to not punish perPlex for suboptimal bases computed by lp_solve. For GlobSol, there are some additional limits that can be set for the computations [18]. The upper bounds on resource parameters, like `MAXITR` or `NROWMAX`, will be increased if GlobSol aborts due to these limits. Peeling will be enabled for all bounds. For the problems GlobSol cannot solve, the databox will be enlarged and peeling disabled. It might be possible to obtain better results by adjusting the stopping tolerances and strategy to the specific problem instances but this seems not suitable for a benchmarking situation. This also reflects the experience of a user without a deeper insight into the algorithm and its parameters.

To generate the input for RealPaver, we will use the known optimal value for the random problems and the rationally computed one for the real-world problems. We will put the perturbation Δ at 10^{-6}, which seems like a reasonable guarantee for practical problems. The bounds computed by GlobSol, ICOS, and Lurupa in median provide at least this guarantee. If the optimal value is 0, we will use an absolute perturbation instead of a relative one (the test set's nonzero optimal value with smallest absolute value is of order 10^{-1}).

Finally a comparison of rigorous optimization algorithms would not be complete without checking the validity of the claims made by the solvers. To check the feasibility of solutions, we will use a simple and therefore easy to check program that takes a solution, computes the exact values of the constraints, and checks if they are satisfied. All this is done using the GNU Multiple Precision Arithmetic Library (GMP) [10].

Solver	27 random	Solved out of 102 feasible real-world	27 infeasible real-world	Total	Errors
ICOS	9 (33%)	2 (2%)	17 (63%)	28 (18%)	18
RealPaver	0 (0%)	8 (8%)	14 (52%)	22 (14%)	0
exlp	9 (33%)	90 (88%)	26 (96%)	125 (80%)	15
perPlex	18 (67%)	94 (92%)	–	112 (87%)	0
QSopt_ex	18 (67%)	102 (100%)	27 (100%)	147 (94%)	0
GlobSol	0 (0%)	3 (3%)	2 (7%)	5 (3%)	7
Lurupa	27 (100%)	99 (97%)	23 (85%)	149 (96%)	0

Table 6.1
Number of problems solved per package

To identify suboptimal solutions and wrong claims of infeasibility, we compare the results from the different solvers and check for inconsistencies.

6. Numerical experience.

6.1. Summary. The main result of this comparison can be seen in the summary in Table 6.1. There we find solver by solver the count and percentage of problems solved from each test set, the total count and percentage of problems solved, and the total number of errors produced (i.e., how often a solver computes a wrong result). The '–' in the row of perPlex is explained by the fact perPlex was designed just to verify the optimality of an approximate optimal basis, it does not implement any logic to verify infeasibility. Hence the total percentage of perPlex is with respect to the feasible problems. We clearly see

- RealPaver is not suitable for the feasible problems. It is naturally better at identifying the infeasible ones.
- The rational algorithms, exlp, perPlex, and QSopt_ex, solve almost all of the feasible problems. They time out for the larger random problems. Only QSopt_ex verifies the infeasibility of all infeasible test problems.
- GlobSol solves only 5 problems in total. It is unsuitable for these problems. ICOS shows marginally better results for the feasible problems. Its origin in constraint programming clearly shows in the higher number of problems verified to be infeasible. Lurupa solves all but three feasible problems. It identifies the infeasibility of a high percentage of the infeasible problems.

As already mentioned, ICOS uses of safe linear relaxations together with rigorous bounds. Nevertheless the relaxations are only used to reduce the bounds on the individual variables of the problem. ICOS examines the whole search region and does not exploit the linear structure of the problem itself. This explains why the results are worse than that of Lurupa despite employing an akin algorithm.

6.2. Details. We start with a closer examination of the random problems. As already mentioned, the size of the problem is the number of variables and inequality constraints, the number of equality constraints is half of this (two equations for size 5). For each size there are three distinct problems. The running times of the solvers for the different problems are displayed in the double logarithmic plot in Figure 6.1. Each runtime is denoted by a corresponding dot, so there are three dots per solver and size of problem. As we can see, there is almost no variation in runtime for any solver and size, the corresponding dots almost lie on top of each other. The fastest solver, Lurupa, shows the smallest increase in runtime. Therefore it seems unlikely that a solver will meet the runtime limit for a problem of a certain size after failing

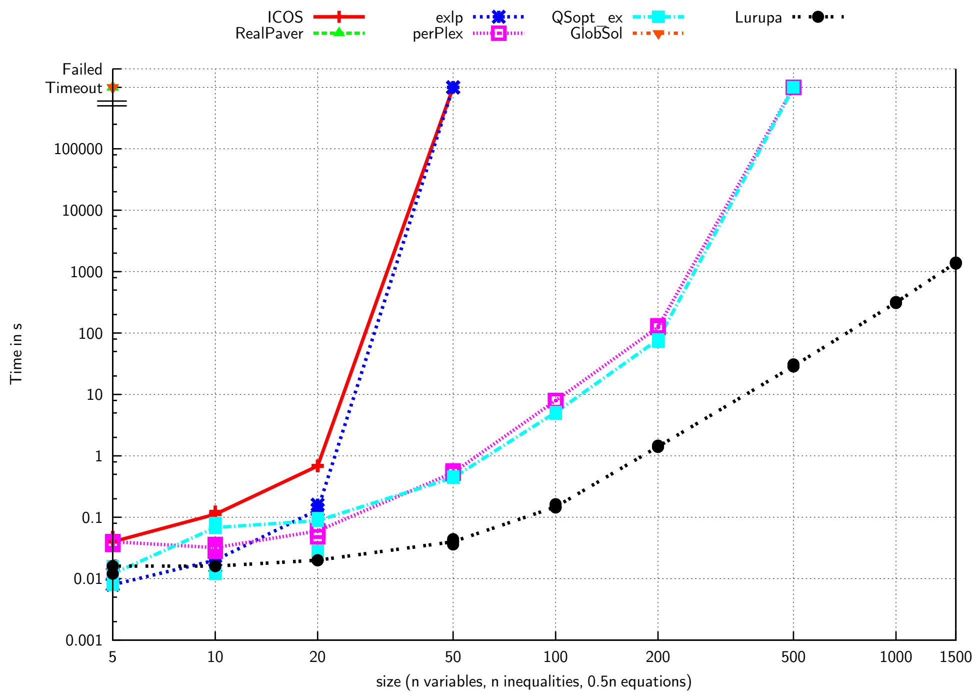

FIG. 6.1. *Running times for random problems*

to do so for all three problems of a smaller size. Hence we will not carry out the computations for these larger problems.

Let us now have a detailed look at the individual solvers. RealPaver and GlobSol already fail to solve the smallest problems within a time factor of 100 to the fastest solver. Kearfott has a development version of GlobSol implementing linear relaxations [19]. This presumably allows GlobSol to solve these problems but is not yet available.

The other solvers start with comparable runtimes. Showing the largest increase, exlp and ICOS solve problems with 5–20 variables. perPlex and QSopt_ex solve problems with up to 200 variables before failing to meet the timeout threshold.

Lurupa proves to be the fastest solver for this test set. For size 50 it is already an order of magnitude faster than all others, and this factor increases to almost two orders of magnitude for size 200. Lurupa is the only package solving problems of size greater than 500.

Figure 6.2 is a performance profile of the random problem runtimes as introduced by Dolan and Moré [7]. For this purpose each runtime $t_{s,p}$ for solver s on problem p is transformed into a runtime ratio to the fastest solver for this problem

$$r_{s,p} = \frac{t_{s,p}}{\min_s\{t_{s,p}\}}.$$

The performance profile is the cumulative distribution function of these ratios for each solver

$$\rho_s(\tau) = \frac{|\{p \mid r_{s,p} \le \tau\}|}{|\{p\}|}$$

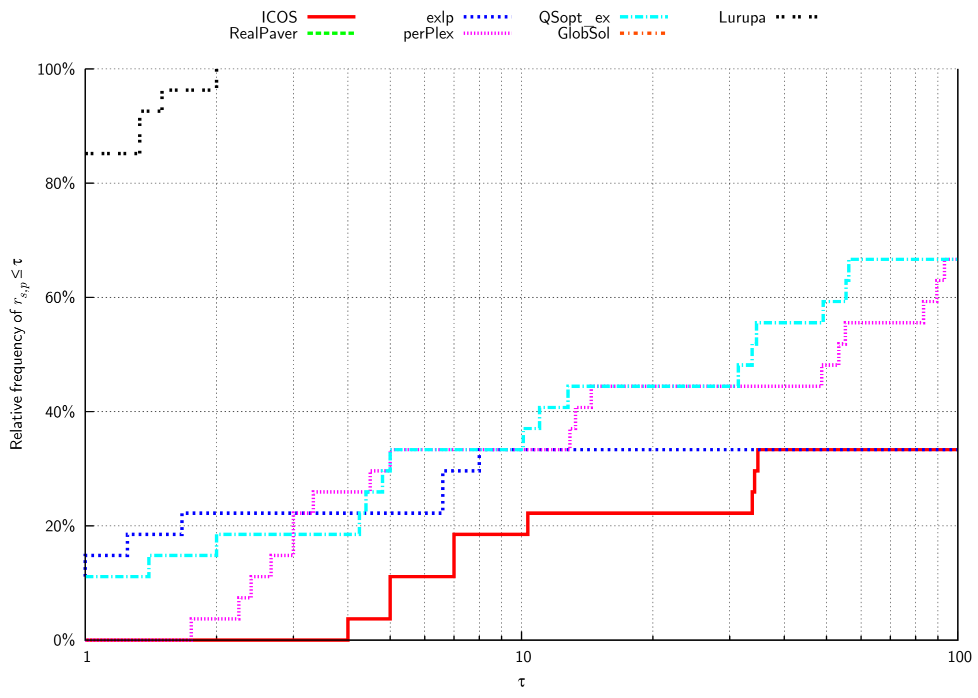

FIG. 6.2. *Performance profile on random problems*

that is the proportion of problems which can be solved with a runtime ratio $r_{s,p}$ less than τ. Two points of special interest in the profile are $\rho_s(1)$ and $\lim_{\tau\to\infty} \rho_s(\tau)$. These denote respectively the proportion of problems that solver s solves fastest and the proportion of problems solved at all. The more the profile moves to the upper left of the axes, the higher percentage of problems are solved with small ratios $r_{s,p}$.

The performance profile supports the previous observations. We can clearly make out three groups, RealPaver and GlobSol, then ICOS, exlp, perPlex, and QSopt_ex, and finally Lurupa. The plot emphasizes the power of Lurupa's algorithm for this test set. For 90% of the problems it is the fastest, the remaining 10% are solved within a factor of two to the fastest solver. Allowing different τ values between 1 and 10 each of exlp, perPlex, and QSopt_ex can have the second highest ρ_s value. For higher time ratios their ranking stays the same.

Now we take a look at the results for the feasible real-world problems. Figures 6.3 and 6.4 show their running times and variable counts. As the size of a problem is a weak indicator of its difficulty here, the problems are sorted by increasing runtime for Lurupa instead of by size as in Figure 6.1.

ICOS, RealPaver, and GlobSol solve only few of these problems. For most problems they time out or fail.

exlp, perPlex, QSopt_ex, and Lurupa show similar increases in runtime with growing differences with increasing problem size. perPlex and QSopt_ex tend to be the fastest solvers, Lurupa and exlp follow.

This test set reveals some errors and limitations in the solvers. We will examine these in the next section.

The performance profile on this set of problems in Figure 6.5 reveals the two

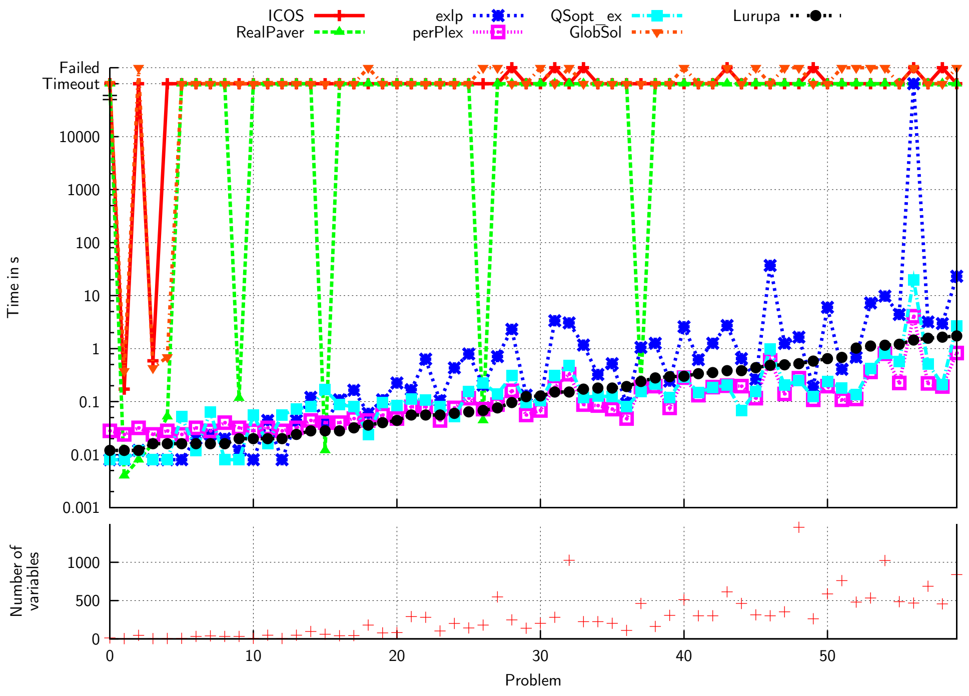

FIG. 6.3. *Running times for feasible real-world problems 0–59*

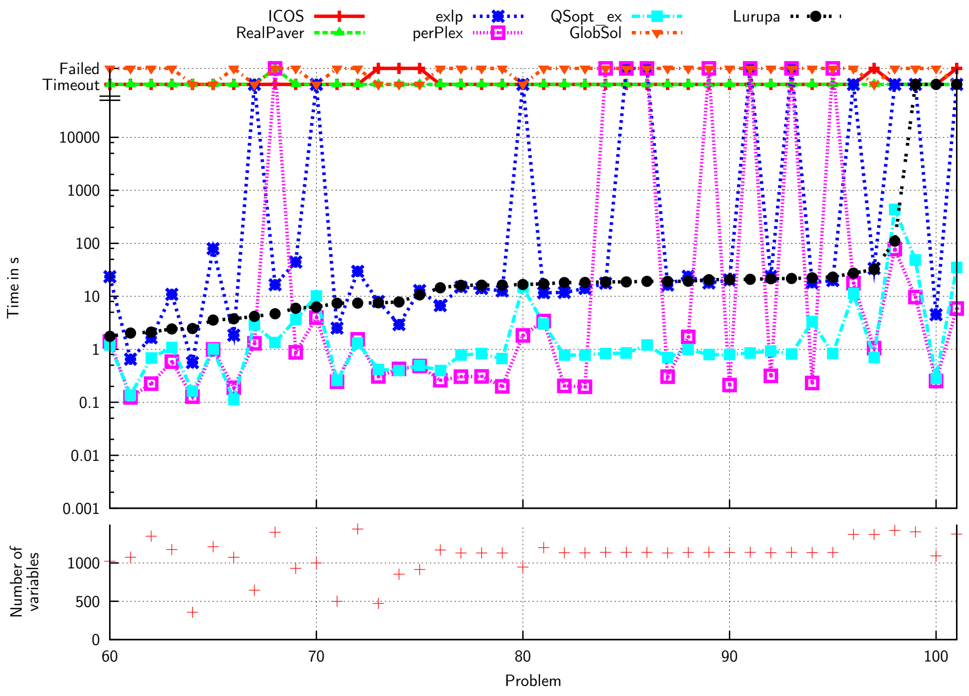

FIG. 6.4. *Running times for feasible real-world problems 60–101*

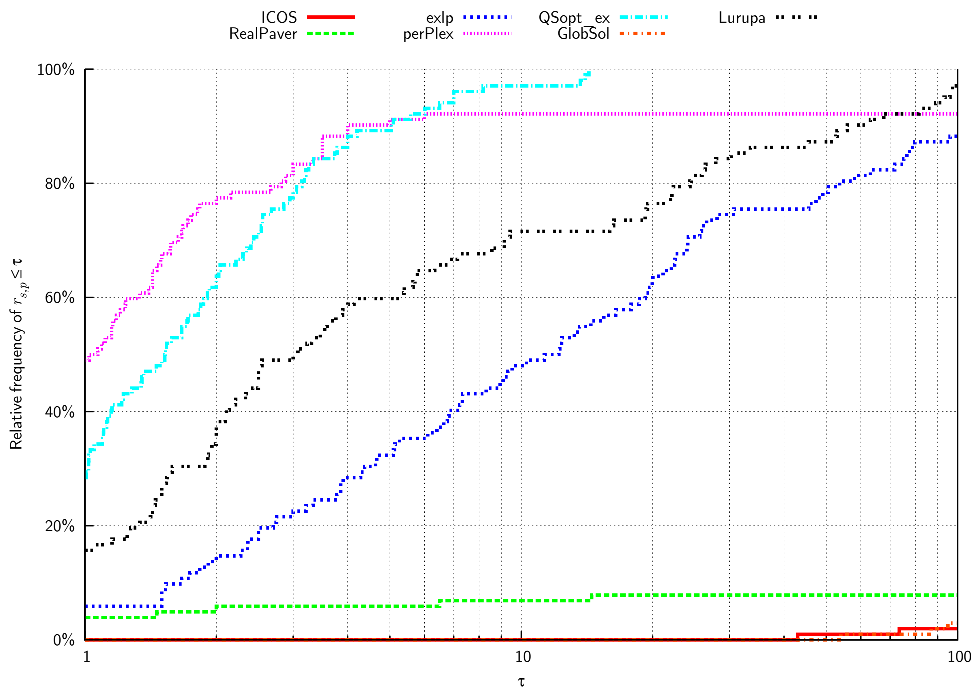

FIG. 6.5. *Performance profile on feasible real-world problems*

groups, on the one hand perPlex, QSopt_ex, exlp, and Lurupa and on the other hand RealPaver, ICOS, and GlobSol. With runtime ratios less than 6, perPlex solves the highest percentage of problems. Allowing larger ratios QSopt_ex takes the lead and solves all problems within a factor of 11. Approaching τ values of 100, Lurupa beats perPlex and solves more problems in total. RealPaver, ICOS, and GlobSol are unsuitable for these problems, solving less than 10% in total.

Finally we look at the infeasible problems. Their runtimes are found in Figure 6.6. Again sorted by Lurupa's runtime we see a somewhat different picture from the feasible problems. As already mentioned, perPlex cannot verify the infeasibility of a problem and is thus missing from these plots.

GlobSol solves only two of these problems and needs almost two orders of magnitude longer than the fastest solver. ICOS and RealPaver verify the infeasibility of several, for most problems the factor in runtime is less than one order of magnitude to the fastest solver.

exlp, QSopt_ex, and Lurupa show a similar behaviour to the feasible case. They have comparable runtime increases with growing variance. The ranking tends to stay QSopt_ex, Lurupa, exlp. Only QSopt_ex verifies the infeasibility of all test problems.

In Figure 6.7 we find the performance profile on this final set of problems. We clearly observe the better performance for ICOS and RealPaver. The ranking of QSopt_ex, Lurupa, exlp holds when allowing runtime ratios up to a factor of 10. Going to higher ratios, exlp ultimately verifies the infeasibility of more problems than Lurupa and their profiles cross.

6.3. Errors and limitations. The real-world test sets reveal some errors and limitations in the solvers. Of the feasible real-world problems, ICOS claims 12 to be

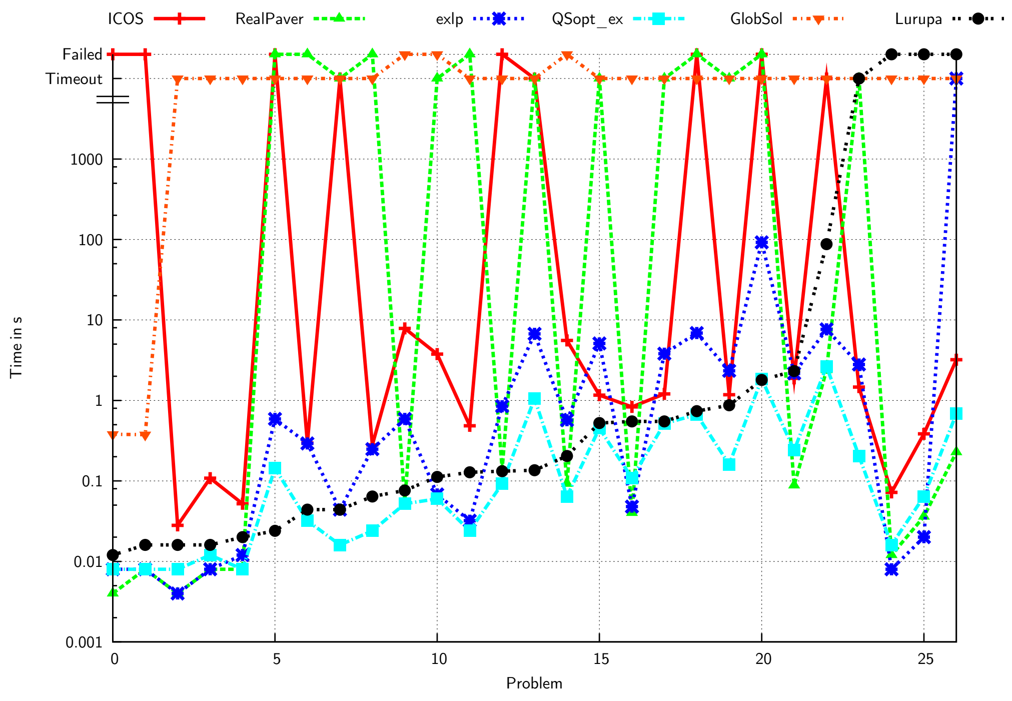

FIG. 6.6. *Running times for infeasible real-world problems*

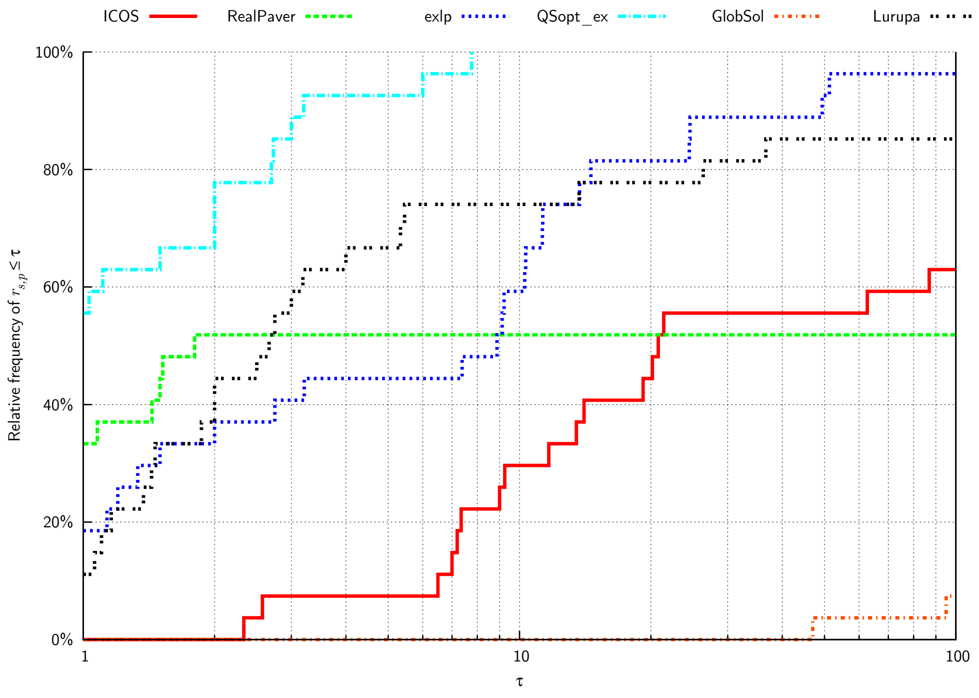

FIG. 6.7. *Performance profile on infeasible real-world problems*

infeasible. In addition six solver runs of ICOS on infeasible problems abort due to software errors.

Comparing the optimal values and checking the solutions for feasibility reveals that exlp returns feasible but suboptimal solutions on seven problems, which can be fixed by disabling preprocessing. On one problem the solution with preprocessing enabled is suboptimal and infeasible, the one without preprocessing is feasible and agrees with the other solver's optimum. On another six problems infeasible solutions are returned with and without preprocessing. A single problem results in exlp wrongly complaining about a malformed problem file.

Testing perPlex on this set of problems reveals a discrepancy in interpreting an approximate basis. lp_solve and other tested LP solvers (soplex 1.3.0 [44], QSopt_ex, and CPLEX 9.0.0 [13]) return bases with free variables being non-basic. Judging from their output, they treat these variables as being 0. perPlex, however, handles all non-basic variables as being at their lower bound and thus complains about such bases putting variables at minus infinity. This problem appears for 16 bases returned by lp_solve. For seven of these LPs, CPLEX returns a basis that perPlex accepts and verifies to be optimal, for one problem QSopt_ex does. The corresponding runtimes are included in the displayed results. For the remaining eight problems no tested LP solver returns a basis that perPlex accepts.

41 problems cannot be submitted to GlobSol due to limitations on the length of a line in the input format. Seven of these feasible problems GlobSol claims to not contain feasible points. The wrong claims were investigated and in part already solved by Kearfott.

7. Conclusions. Even for these rather small LPs, the solvers that do not exploit the structure, ICOS, RealPaver, and GlobSol, cannot keep up with the ones that do, namely Lurupa and the rational ones, exlp, perPlex, and QSopt_ex. Making use of the special structure of an LP is a necessity when aiming for fast and rigorous results.

Lurupa's way of combining approximate results with interval arithmetic produces an enclosure and not the exact solution. While clearly outperforming the rational algorithms for the dense random problems, it solves the more difficult, real-world problems within a factor of two orders of magnitude to the fastest rational implementation, perPlex or QSopt_ex. Lurupa's algorithm on the other hand can indicate model inconsistencies and numerical problems by delivering wide bounds and verified condition numbers. It allows to compute enclosures if the input data are not exactly known but subject to uncertainty. In contrast to the rational algorithms, Lurupa's algorithm can be generalized to convex optimization as has been done in VSDP, and it is not limited to problems that can be solved in the rationals.

REFERENCES

[1] D. L. Applegate, W. Cook, S. Dash, and D. G. Espinoza, *Qsopt_ex*. World Wide Web. `http://www.dii.uchile.cl/~daespino/QSoptExact_doc/main.html`.

[2] ———, *Exact solutions to linear programming problems*, submitted to Operations Research Letters, (2006).

[3] H. Beeck, *Linear Programming with Inexact Data*, Tech. Rep. 7830, Abteilung Mathematik, TU München, 1978.

[4] M. Berkelaar, P. Notebaert, and K. Eikland, *lp_solve*. World Wide Web. `http://groups.yahoo.com/group/lp_solve`.

[5] M. Berz et al., *COSY Infinity*. World Wide Web. `http://www.bt.pa.msu.edu/index_files/cosy.htm`.

[6] M. Dhiflaoui, S. Funke, C. Kwappik, K. Mehlhorn, M. Seel, E. Schömer,

R. Schulte, and D. Weber, *Certifying and repairing solutions to large LPs how good are LP-solvers?*, in SODA, 2003, pp. 255–256.

[7] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2001), pp. 201–213. DOI 10.1007/s101070100263.

[8] Free Software Foundation, *gcc*. `http://gcc.gnu.org`.

[9] B. Gärtner, *Exact arithmetic at low cost – a case study in linear programming*, Computational Geometry, 13 (1999), pp. 121–139.

[10] T. Granlund et al., *GNU multiple precision arithmetic library*. World Wide Web. `http://gmplib.org`.

[11] L. Granvilliers and F. Benhamou, *Algorithm 852: RealPaver: An Interval Solver using Constraint Satisfaction Techniques*, ACM Transactions on Mathematical Software, 32 (2006), pp. 138–156. `http://doi.acm.org/10.1145/1132973.1132980`.

[12] H. J. Greenberg, *An Analysis of Degeneracy*, Naval Research Logistics Quaterly, 33 (1986), pp. 635–655.

[13] ILOG, *Cplex*. `http://www.ilog.com/products/cplex`.

[14] C. Jansson, *VSDP: Verified SemiDefinite Programming, User's Guide*. Beta Version 0.1, to appear.

[15] ———, *A Self-Validating Method for Solving Linear Programming Problems with Interval Input Data*, Computing, Suppl. 6 (1988), pp. 33–45.

[16] ———, *Rigorous Lower and Upper Bounds in Linear Programming*, SIAM J. Optimization (SIOPT), 14 (2004), pp. 914–935.

[17] R. B. Kearfott, *GlobSol*. World Wide Web. `http://interval.louisiana.edu`.

[18] ———, *Globsol: History, composition, and advice on use*, in Global Optimization and Constraint Satisfaction, C. Bliek, C. Jermann, and A. Neumaier, eds., vol. 2861 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2003, pp. 17–31.

[19] ———, *Globsol – present state and future developments*, Feb. 27 2007. Talk given at the International Workshop on Numerical Verification and its Applications, Waseda University, Tokyo.

[20] C. Keil, *Lurupa – Rigorous Error Bounds in Linear Programming*, in Algebraic and Numerical Algorithms and Computer-assisted Proofs, B. Buchberger, S. Oishi, M. Plum, and S. Rump, eds., no. 05391 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. `http://drops.dagstuhl.de/opus/volltexte/2006/445`.

[21] C. Keil and C. Jansson, *Computational Experience with Rigorous Error Bounds for the Netlib Linear Programming Library*, Reliable Computing, 12 (2006), pp. 303–321.

[22] M. Kiyomi, *exlp*. World Wide Web. `http://members.jcom.home.ne.jp/masashi777/exlp.html`.

[23] O. Knüppel, *PROFIL/BIAS and extensions, Version 2.0*, tech. rep., Inst. f. Informatik III, Technische Universität Hamburg-Harburg, 1998.

[24] T. Koch, *perPlex*. World Wide Web. `http://www.zib.de/koch/perplex`.

[25] ———, *The final netlib-lp results*, Tech. Rep. 03-05, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraße 7, D-14195 Berlin-Dahlem, Germany, 2003.

[26] R. Krawczyk, *Fehlerabschätzung bei linearer Optimierung*, in Interval Mathematics, K. Nickel, ed., vol. 29 of Lecture Notes in Computer Science, Springer Verlag, Berlin, 1975, pp. 215–222.

[27] Y. Lebbah, *ICOS (Interval COnstraints Solver)*. World Wide Web. `http://www.essi.fr/~lebbah/icos/index.html`.

[28] L. Lovasz, *A New Linear Programming Algorithm – Better or Worse Than the Simplex Method?*, The Mathematical Intelligencer, 2 (1980), pp. 141–146.

[29] C. Mészáros, *Linear programming test problems*. `http://www.sztaki.hu/~meszaros/bpmpd`.

[30] A. Nemirovski, *Lectures on Modern Convex Optimization*, 2003.

[31] Netlib, *Netlib linear programming library*. `http://www.netlib.org/lp`.

[32] A. Neumaier, *Complete Search in Continuous Global Optimization and Constraint Satisfaction*, Acta Numerica, 13 (2004), pp. 271–369.

[33] A. Neumaier and O. Shcherbina, *Safe bounds in linear and mixed-integer programming*, Mathematical Programming, Ser. A, 99 (2004), pp. 283–296.

[34] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinkó, *A comparison of complete global optimization solvers*, Math. Program., 103 (2005), pp. 335–356.

[35] F. Ordóñez and R. M. Freund, *Computational experience and the explanatory value of condition measures for linear optimization*, SIAM J. Optimization, 14 (2003), pp. 307–333.

[36] J. Renegar, *Linear programming, complexity theory and elementary functional analysis*,

Math. Program., 70 (1995), pp. 279–351.

[37] R. T. Rockafellar, *Lagrange multipliers and optimality*, SIAM Review, 35 (1993), pp. 183–238.

[38] J. B. Rosen and S. Suzuki, *Construction of Nonlinear Programming Test Problems*, Communication of ACM, 8 (1965), p. 113.

[39] S. M. Rump, *Solving Algebraic Problems with High Accuracy. Habilitationsschrift*, in A New Approach to Scientific Computation, U. Kulisch and W. Miranker, eds., Academic Press, New York, 1983, pp. 51–120.

[40] P. Van Hentenryck, P. Michel, and Y. Deville, *Numerica: A Modelling Language for Global Optimization*, MIT Press Cambridge, 1997.

[41] L. Vandenberghe and S. Boyd, *Semidefinite programming*, SIAM Review, 38 (1996), pp. 49–95.

[42] X.-H. Vu, D. Sam-Haroud, and M.-C. Silaghi, *Approximation techniques for non-linear problems with continuum of solutions*, in Proceedings of The 5th International Symposium on Abstraction, Reformulation and Approximation (SARA'2002), R. C. H. Sven Koenig, ed., vol. LNAI 2371 of Lecture Notes in Computer Science - Lecture Notes in Artificial Intelligence, Canada, August 2002, Springer-Verlag, pp. 224–241. © Springer-Verlag.

[43] ———, *Numerical constraint satisfaction problems with non-isolated solutions*, in 1st International Workshop on Global Constrained Optimization and Constraint Satisfaction (COCOS'2002), France, October 2002, COCONUT Consortium.

[44] R. Wunderling, *Soplex*. World Wide Web. `http://soplex.zib.de`.