Rigorous Error Bounds for
Finite Dimensional Linear Programming Problems

Vom Promotionsausschuss der
Technischen Universität Hamburg–Harburg
zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing)
genehmigte Dissertation

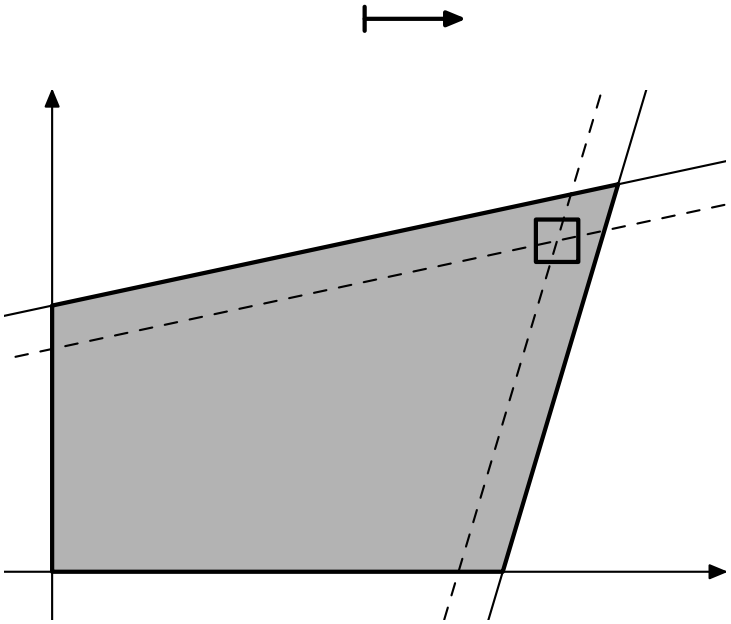
von
Christian Keil

aus
Hamburg

2009

1. Gutachter: Prof. Dr. Siegfried M. Rump
2. Gutachter: Prof. Dr. Dr. h.c. Frerich J. Keil
Tag der mündlichen Prüfung: 17. Dezember 2008

Rigorous Error Bounds for Finite Dimensional Linear Programming Problems



CHRISTIAN KEIL

erschienen bei: Books on Demand GmbH, Norderstedt
ISBN 9783837093353

Contents

Acknowledgements

I want to thank several people that made this work possible and that made the past four years seem to go by like a breeze.

First of all these are Professor S. M. Rump and P.D. C. Jansson, my scientific guidance, who started my interest in verified computations in the first place. Without their advice and hours of discussions the present form of this work would not have been possible.

Then I would like to thank the whole Institute for Reliable Computing of the Hamburg University of Technology for a wonderful time. Dirk, Malte, Ole, Robert, and Viktor for necessary distractions during breaks and many a discussion. Our technical staff S. Kubon for keeping our systems running and solving all technical problems as well as life-saving backups of data believed to be lost, H. Meyer for welcome and unwelcome comments and several talks to keep perspective. Finally our secretary U. Schneider for an always open office and uplifting tea breaks.

Last but not least I am very thankful to my parents and Bob for hours of proofreading incomprehensible mathematical texts and helping to remove all the awkward formulations that I put in in the first place.

Introduction

Linear Programming

Previous works were written by Fourier [30], de la Vallée Poussin [19], and Kantorovich [56], the success story of linear programming, however, really started in 1947 with a seminal discovery by Dantzig [15]. At this time he was Mathematical Adviser to the US Air Force Comptroller in the Pentagon.

In 1947 all planning of resources (it was not called optimization) was done by hand; there was no mathematical concept to guide the process. To get an impression of this, we will look at a simple example by Dantzig [17] himself:

Suppose we have to assign 70 men to 70 jobs. Each single assignment of man *i* to job *j* has a certain value or benefit attributed to it. The restrictions are: (i) each man has to be assigned a job, and (ii) each job has to be filled. This leaves us with 70! possible assignments, each having a total benefit given by the sum of benefits over all single assignments. Finding the best assignment, the one with the largest sum of benefits, requires to generate and compare all possible assignments with one another. Solving this task by brute force would require "a solar system full of nano-second electronic computers running from the time of the big bang until the time the universe grows cold to scan all the permutations in order to select the one which is best". And by today's standards this is a really small problem. To achieve results in a reasonable time, the planning was guided by so called *ad hoc ground rules* to reduce the computational work. These suggested strategies and patterns to look for when solving a problem. They were devised by the experts in their fields and reflected their years of empirical experience.

After being challenged by two colleagues to mechanize the process of computing deployment, training, and logistical supply plans for combat units, Dantzig formulated a mathematical model capturing the nature of the problem. He describes his most important contributions as:

1. *Recognizing (as a result of my wartime years as a practical program planner) that most practical planning relations could be reformulated as a system of linear inequalities.*

2. *Replacing the set of ground rules for selecting good plans by an objective function. (Ground rules at best are only a means for carrying out the objective, not the objective itself.)*

3. *Inventing the simplex method which transformed the rather unsophisticated linear-programming model of the economy into a basic tool for practical planning of large complex systems.*

Introducing an objective function, Dantzig can rightfully be seen as the father of linear and mathematical programming in general.

In the meantime several other algorithms were developed to solve linear programming problems. Among them the ellipsoid method and interior-point methods. Still the simplex method and its variants offer a good choice if a linear program has to be solved.

The importance of linear programming can be estimated when looking at quotes from Lovász in 1980 for example [77]: "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". The same year Eugene Lawler of Berkeley summarized it the following: "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." While the applications start to move to more complex forms of optimization like conic programming, linear programming retains an important role in such diverse areas as oil refinery problems, flap settings on aircraft, industrial production and allocation, and image restoration (see documentation in [87]).

Searching the Web of Science [113]—a database indexing thousands of scholarly journals—for "linear programming" excluding "integer linear programming", reveals over 500 articles just for the year 2007 from applied mathematics, traffic planning, control theory, software engineering, lithography, economics, biology, climate research, and several others.

In addition linear programming plays an important role in nonlinear and discrete optimization, where it is used to solve relaxations. One of the most successful global optimization solvers, BARON [111], uses linear relaxations exclusively in favor of nonlinear convex relaxations. The authors, Tawarmalani and Sahinidis, remark: "Linear programming technology has reached a level of maturity that provides robust and reliable software for solving linear relaxations of MILP problems. Nonlinear programming solvers, however, often fail even in solving convex problems. At the theoretical level, duality theory provides necessary and sufficient conditions for optimality in linear programming, whereas the KKT optimality conditions, that are typically exploited by nonlinear programming algorithms, are not even necessary unless certain constraint qualifications hold."

Verification

> *"I expect that most of the numerical computer programs of 2050 will be 99% intelligent 'wrapper' and just 1% actual 'algorithm,' if such a distinction makes sense. Hardly anyone will know how they work, but they will be extraordinarily powerful and reliable, and will often deliver results of guaranteed accuracy."*
>
> —Lloyd N. Trefethen, 2006

Subjecting results of numerical computations to scrutiny goes back to van Neumann and Goldstine [94], who analyzed the rounding errors introduced by the basic operations in fixed-point arithmetic. For the basic operations in floating point arithmetic this has been done by Wilkinson [122].

Using their results, one can consider two ways or kind of two directions to take rounding errors during the execution of an algorithm into account. The *forward error* of an operation is the error in the result introduced by it: applied to the execution of an algorithm, it is the distance between the mathematically correct result and the computed one.

The other direction, the *backward error*, describes the error that has to be made in the problem parameters for the computed result to be mathematically correct. This idea was to some extent implicit in the work of Turing [115] and van Neumann and Goldstine, but was described explicitly by Givens [37].

The link between these two types of error estimation is the *condition number* of a problem. This term seems to have been used first by Turing [115], although the term "ill-conditioned" was in use before. The condition number captures the sensitivity of a problem by quantifying the amplification of errors in the parameters of a problem into the error of the results. In general we can say that when solving a problem with a computer, we make small errors that cause a different problem to be solved. The backward error (the distance between the original and the solved problem) multiplied by the condition number is an upper bound on the forward error (the distance between the solution of the original problem and the computed result). A more thorough treatment can be found in Higham [44].

Although very successful over the years, the approximation property and further intricacies of floating point computations can easily lead to severe errors in the results [13, 109, 110]. Unfortunately in many cases these errors go unnoticed. Floating point software packages do not flag possible errors with warnings, or warnings are given for perfectly reasonable results as observed by Jansson, Chaykin, and Keil [54].

These problems can be dealt with by using the mathematical theory underlying the problem—in the context of this work duality theory—together with a kind of automatic forward error analysis. Using bounds on the forward error, one can represent the result of an operation by a pair of machine representable numbers, an *interval*, that is guaranteed to include the true result. The development of this idea and its generalizations to the field of *Interval Analysis*, which are generally acknowledged to have started with Moore's book [82],

are described by Moore himself in [84].

Together with the widespread standards for floating point arithmetic IEEE-754 [47] and IEEE-854 [48], fully rigorous software packages are possible that take all rounding errors into account. This is achieved by appropriately switching rounding-modes and computing with the endpoints of the intervals.

In this work we want to explore verification methods for linear programming problems and some generalizations. The intuition suggests that linear programming problems, like linear systems of equations, should not be susceptible to numerical problems. An investigation of the netlib linear programming test problems [36] by Ordóñez and Freund [97], however, shows that 71% of these problems are ill-posed. Ill-posed in this setting means that an arbitrarily small change in the problem parameters may render the problem infeasible or unbounded. The investigation by Ordóñez and Freund also shows that 19% of the netlib problems remain ill-posed if preprocessing heuristics are applied, but as Fourer and Gay observed [29], these heuristics may change the state of a linear program from infeasible to unbounded and vice versa.

The first chapter describes the theory of the rigorous methods. It begins with a description of the nature of rounding errors and the basics of interval arithmetic. Condition numbers and distances to infeasibility after Renegar and Ordóñez and Freund are introduced. After that the theory of linear programming is described. Theory and algorithms of the rigorous error bounds are given, accompanied by a convergence analysis and a description of certificates of infeasibility. The chapter closes with a description of generalizations; an application of the rigorous error bounds in mixed-integer linear programming and rigorous error bounds for second-order cone programming and semidefinite programming.

In the second chapter Lurupa, the software implementation of the rigorous methods, is depicted. It describes the architecture and details of the implementation. The different ways of using the software standalone, as a library, or from MATLAB are described and examples for each way are given. The chapter closes with a description of how Lurupa can be extended to support new linear programming solver.

The third chapter contains extensive numerical results obtained with Lurupa. It presents results on the netlib collection of linear programming problems. The results show that for most of these problems, rigorous lower bounds and several rigorous upper bounds can be computed at reasonable costs. In most cases, the bounds are computed for the problem instances with a nonzero distance to infeasibility. Following these results is a comparison with seven other software packages capable of producing rigorous results for linear programming problems. According to the techniques the solvers employ, they can be categorized into four groups: (i) verified constraint programming, (ii) algorithms using a rational arithmetic, (iii) verified global optimization, and (iv) verified linear programming with uncertainties, the latter having Lurupa as the only instance. Computational results of all solvers on a test set of more than 150 problems are presented. The different techniques used by the solvers are clearly reflected in their results. Making use of the special struc-

ture present in the problem is a necessity when aiming for fast and reliable results. The rigorous methods examined in this thesis and the methods using a rational arithmetic are faster by orders than the constraint programming and global optimization methods. While the former methods produce rigorous results for problems with several thousand variables and constraints, the latter methods are only applicable with problems with up to a few hundred variables. Whether Lurupa or the rational solvers are faster depends on the problem characteristic. The approach used by Lurupa, however, can be applied to problems only known with uncertainty, and it can be generalized to problems that cannot be solved in rational numbers.

Chapter 4 contains conclusions, and chapter 5 lists promising areas for future research.

CHAPTER 1

Theory

In this chapter we will first set some notation. We will recapitulate the necessary theoretical basis for the verified computations. The rest of the chapter deals with the rigorous bound computing algorithms.

1.1 Notation

"We will occasionally use this arrow notation unless there is danger of no confusion."

—R. Graham, "Rudiments of Ramsey Theory"

We use $\mathbb{R}$, $\mathbb{R}^n$, $\mathbb{R}^n_-$, and $\mathbb{R}^{m\times n}$ to denote the sets of real numbers, real vectors, real nonpositive vectors, and real $m \times n$ matrices, respectively. We use small Greek letters for numbers, small Roman letters for vectors, and capital letters for matrices. The dimensions are given explicitly if they are not obvious from the context. Comparisons $\leq$, absolute value $|\cdot|$, min, max, inf, and sup are used entrywise for vectors and matrices. In this context, 1 and 0 can be used as vectors of appropriate dimensions, $0 \leq x \leq 1$ means that every component of x lies between 0 and 1. If we need a vector of 1s in other places, we use e. The ith unit vector, being the vector of 0s with a 1 in the ith component, is denoted e_i; the identity matrix is denoted I.

The ith row and jth column of a matrix A are denoted $A_{i:}$ and $A_{:j}$, respectively. If $\mathcal{J}$ is a set of indices, $x_{\mathcal{J}}$ is the induced subvector of x, consisting of the components x_j with index in $\mathcal{J}$. Similarly $A_{:\mathcal{J}}$ is the submatrix formed by the columns with index in $\mathcal{J}$.

1.2 Rounding errors

To understand the nature and implications of rounding errors we make a short excursion into how floating point numbers are stored in a computer. Today most architectures adhere more or less to the IEEE standards for floating point

arithmetic 754 and 854 [47, 48]. These define floating point numbers to be of the form

$$(-1)^s d_0.d_1 d_2 \dots d_{p-1} \cdot b^E.$$

The sign is determined by s, the significand is formed by the base-b digits d_i. This is multiplied by b to the exponent E. Different precisions are determined by the base b, the length of the significand p and the exponent limits $E_{min} \leq E \leq E_{max}$. Most numerical software operates with binary floating-point numbers. The usual parameters for *single* and *double* precision are $p = 24$, $E_{min} = -126$, and $E_{max} = 127$ for single and $p = 53$, $E_{min} = -1022$, and $E_{max} = 1023$ for double precision.

Since the number of significand digits is finite, floating-point numbers can only approximate real numbers. If a number is not contained in the set of floating-point numbers, it is substituted with a floating point number by a process called *rounding*. Except for binary to decimal conversion and back, this process has to be carried out as if the result is computed to infinite precision and then rounded according to the set rounding mode. The standards require the default rounding mode "round to nearest" and "round toward $+\infty$", "round toward $-\infty$", and "round toward 0".

The set of floating-point numbers is not closed under the basic operations $+, -, \cdot, /$. If conforming to the standard and rounding to nearest, they instead produce best possible results of the type

$$\begin{aligned} x \boxplus y &= (x + y)(1 + \varepsilon) \\ x \boxminus y &= (x - y)(1 + \varepsilon) \\ x \boxdot y &= x \cdot y(1 + \varepsilon) \\ x \boxslash y &= x/y(1 + \varepsilon). \end{aligned}$$

The relative error ε is bounded by the *machine epsilon*, $|\varepsilon| \leq \varepsilon_A$. The machine epsilon is given by $0.5 \cdot b^{-(p-1)}$; in the case of binary single or double precision this is 2^{-24} or 2^{-53}, respectively.

Even basic laws of mathematics are not valid for floating point numbers. As basic operations return rounded results, associativity, for example, does not hold. With double precision and round to nearest, the following expressions are different

$$1 = (1 \boxplus 2^{-53}) \boxplus 2^{-53} \quad \neq \quad 1 \boxplus (2^{-53} \boxplus 2^{-53}) > 1.$$

The left hand side $1 \boxplus 2^{-53}$ being rounded to 1 instead of the next larger floating point number stems from the fact that ties are rounded to a number with an even last digit. A reasoning for this and further properties can be found in the excellent exposition of floating point arithmetic by Goldberg [38].

1.3 Interval arithmetic

We only need some basic properties of interval arithmetic which are presented here. Extensive treatments of interval arithmetic and self-validating methods

can be found in a number of highly recommendable textbooks. These include Alefeld and Herzberger [1], Moore [83], and Neumaier [88, 90].

If $\mathbb{V}$ is one of the spaces $\mathbb{R}$, $\mathbb{R}^n$, $\mathbb{R}^{m\times n}$, and $\underline{v}, \overline{v} \in \mathbb{V}$, then the box

$$\mathbf{v} := [\underline{v}, \overline{v}] := \{v \in \mathbb{V} \mid \underline{v} \leq v \leq \overline{v}\}$$

is called an *interval quantity* in $\mathbb{IV}$ with *lower bound* $\underline{v}$ and *upper bound* $\overline{v}$. All interval quantities are written in boldface. In particular, $\mathbb{IR}$, $\mathbb{IR}^n$, and $\mathbb{IR}^{m\times n}$ denote the set of real intervals $\mathbf{a} = [\underline{a}, \overline{a}]$, the set of real interval vectors $\mathbf{x} = [\underline{x}, \overline{x}]$, and the set of real interval matrices $\mathbf{A} = [\underline{A}, \overline{A}]$, respectively.

The real operations $A \circ B$ with $\circ \in \{+, -, \cdot, /\}$ between real numbers, real vectors, and real matrices can be generalized to *interval operations*. The result $\mathbf{A} \circ \mathbf{B}$ of an interval operation is defined as the interval hull of all possible real results

$$\mathbf{A} \circ \mathbf{B} := \cap\{\mathbf{C} \in \mathbb{IV} \mid A \circ B \in \mathbf{C} \text{ for all } A \in \mathbf{A}, B \in \mathbf{B}\}.$$

All interval operations can be easily computed from the lower and upper bounds of the interval quantities. For example in the simple case of addition, we obtain

$$\mathbf{A} + \mathbf{B} = [\underline{A} + \underline{B}, \overline{A} + \overline{B}].$$

Interval multiplications and divisions require a distinction of cases.

In a similar way operations between interval vectors and interval matrices can be executed. For example the i, j component of the product of two interval matrices $\mathbf{C}, \mathbf{X} \in \mathbb{IR}^{n\times n}$ is

$$(\mathbf{CX})_{ij} = \sum_{k=1}^{n} \mathbf{C}_{ik}\mathbf{X}_{kj}. \tag{1.1}$$

and the inner product

$$\text{trace}(\mathbf{C}^T\mathbf{X}) = \sum_{i,j=1}^{n} \mathbf{C}_{ij}\mathbf{X}_{ij}. \tag{1.2}$$

For interval quantities $\mathbf{A}, \mathbf{B} \in \mathbb{IV}$ we define

$$\check{\mathbf{A}} := (\underline{A} + \overline{A})/2 \qquad \text{the } \textit{midpoint}, \tag{1.3}$$

$$\mathring{\mathbf{A}} := (\overline{A} - \underline{A})/2 \qquad \text{the } \textit{radius}, \tag{1.4}$$

$$|\mathbf{A}| := \max\{|A| \mid A \in \mathbf{A}\} \qquad \text{the } \textit{absolute value}, \tag{1.5}$$

$$\mathbf{A}^+ := \overline{A}^+ := \max\{0, \overline{A}\}, \tag{1.6}$$

$$\mathbf{A}^- := \underline{A}^- := \min\{0, \underline{A}\}. \tag{1.7}$$

With midpoint and radius, we have an alternative way of denoting an interval: $[\underline{v}, \overline{v}] = \langle \check{\mathbf{v}}, \mathring{\mathbf{v}} \rangle$. The comparison in $\mathbb{IV}$ is defined by

$$\mathbf{A} \leq \mathbf{B} \quad \text{if and only if} \quad \overline{A} \leq \underline{B}.$$

Other relations are defined analogously. Real quantities v are embedded in the interval quantities by identifying v with $\mathbf{v} = [v, v]$.

Finally, if we have a linear system of equations with inexact input data, many applications require to compute an interval vector $\mathbf{x} \in \mathbb{IR}^n$ containing the *solution set*

$$\Sigma(\mathbf{A}, \mathbf{b}) := \{x \in \mathbb{R}^n \mid Ax = b \text{ for some } (A, b) \in (\mathbf{A}, \mathbf{b})\}, \tag{1.8}$$

where $\mathbf{A} \in \mathbb{IR}^{n \times n}$, and $\mathbf{b} \in \mathbb{IR}^n$. This is an NP-hard problem, but there are several methods that compute such enclosures $\mathbf{x}$. A precise description of appropriate methods, required assumptions, and approximation properties can be found for example in Neumaier [88]. Generally speaking it turns out that for interval matrices with $\|I - R\mathbf{A}\| < 1$ (R is an approximate inverse of the midpoint $\check{\mathbf{A}}$) there are several methods which compute an enclosure $\mathbf{x}$ with $\mathcal{O}(n^3)$ operations. The radius $\mathring{\mathbf{x}}$ decreases linearly with decreasing radii $\mathring{\mathbf{A}}$ and $\mathring{\mathbf{b}}$. For the computation of enclosures in the case of large-scale linear systems, see Rump [105].

In linear programming we usually have rectangular systems of equations with fewer equations than variables. We can use the aforementioned methods to compute an enclosure containing solutions for all systems in the interval if we have an approximate point solution $\tilde{x}$. For a $p \times n$ system of equations, we partition the set of variable indices into a set $\mathcal{F}$ of $n - p$ variables to be fixed and a set $\mathcal{V}$ of variables that will be turned into intervals. Moving the terms of the fixed variables to the right hand side, we make the system quadratic

$$\mathbf{A}x = \mathbf{b} \quad \rightarrow \quad \mathbf{A}_{:\mathcal{V}} x_{\mathcal{V}} = \mathbf{b} - \mathbf{A}_{:\mathcal{F}} \tilde{x}_{\mathcal{F}}.$$

Applying above methods to the new system yields the enclosure $(\tilde{x}_{\mathcal{F}}, \mathbf{x}_{\mathcal{V}})$ of the original solution set with a proper reordering of the components.

1.4 Linear programming

Linear programming is a part of mathematical programming, which deals with the optimization of a real *objective function* of real or integer variables subject to *constraints* on the variables. In linear programming the objective function and constraints are affine, the variables are continuous with lower and upper *simple bounds*. There are several universal formulations for a linear program (LP); in the following we will use

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Ax \le a \\ & Bx = b \\ & \underline{x} \le x \le \overline{x}. \end{aligned} \tag{1.9}$$

The vectors c, x, $\underline{x}$, and $\overline{x} \in \mathbb{R}^n$, we have m inequalities defined by $A \in \mathbb{R}^{m \times n}$ and $a \in \mathbb{R}^m$ and p equations defined by $B \in \mathbb{R}^{p \times n}$ and $b \in \mathbb{R}^p$. Simple bounds may be infinite; $\underline{x}_i = -\infty$, $\overline{x}_j = \infty$ is allowed.

An LP (1.9) can be identified with the parameter tuple $P := (A, B, a, b, c)$ and the simple bounds $\underline{x}, \overline{x}$. Used together, they determine the *feasible region*

$\mathcal{X}(P)$, which is the set of all x satisfying the constraints and simple bounds, and the *optimal value* $f^*(P)$. Depending on their simple bounds, the indices of the variables are collected in different sets $\mathcal{B}$. A bar above the set denotes a finite bound whereas a double dot an infinite one (reminding of an infinity symbol). A missing accent does not specify the bound. The lower bound is denoted in a similar way, and a double dot denotes a lower bound of $-\infty$. The set $\mathcal{B}$ thus contains all variable indices as no bound is characterized. The set $\underline{\bar{\mathcal{B}}}$ contains the indices of all variables with a finite lower bound and an infinite upper one. The set $\underline{\ddot{\mathcal{B}}}$ contains the indices of all variables with an infinite lower bound (i.e., $-\infty$) irrespective of the upper one.

If no point satisfies all constraints, the problem is called *infeasible* and the optimal value is ∞. If the feasible region is not bounded in the direction $-c$, the LP is called *unbounded* and the optimal value is $-\infty$.

Instead of simpler formulations like the *standard form* LP

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{1.10}$$

or the *inequality form* LP

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Ax \leq b \end{aligned} \tag{1.11}$$

Formulation (1.9) allows us to model equations and free variables directly. While the formulations are equivalent from a theoretical point of view, they behave differently when treated numerically. For example: transforming an equation into two inequalities or a free variable into the difference of two non-negative ones immediately makes the problem *ill-posed*. An arbitrarily small change of the problem parameters can make these problems infeasible. Writing an equation with two inequalities

$$B_{i:}x = b_i \qquad \longrightarrow \qquad \begin{aligned} B_{i:}x &\leq b_i \\ B_{i:}x &\geq b_i, \end{aligned} \tag{1.12}$$

the problem can be made infeasible with a slight change of the right hand side of one of the inequalities

$$\begin{aligned} B_{i:}x &\leq b_i - \varepsilon \qquad \varepsilon > 0 \\ B_{i:}x &\geq b_i. \end{aligned}$$

To substitute two positive variables for a free one, we have to add copies with opposite sign of the columns in the constraint matrices A and B and the corresponding entry in the objective vector c

$$\begin{aligned} & c_i x_i \\ & A_{:i} x_i \\ & B_{:i} x_i \\ & -\infty \leq x_i \leq \infty \end{aligned} \qquad \xrightarrow{x = x^+ - x^-} \qquad \begin{aligned} & c_i x_i^+ - c_i x_i^- \\ & A_{:i} x_i^+ - A_{:i} x_i^- \\ & B_{:i} x_i^+ - B_{:i} x_i^- \\ & x_i^+ \geq 0,\ x_i^- \geq 0. \end{aligned} \tag{1.13}$$

We will see shortly why this makes the LP ill-posed.

An important task in optimization is the computation of bounds on the optimal value, which gives rise to the topic of *duality*. A *dual* is another LP—in general another mathematical program—whose objective value is always a bound on the *primal* one. There are several different dual problems. Using the convention that $0 \cdot \infty$ evaluates to 0, the *Lagrangian dual* of (1.9) is

$$\begin{aligned} \max \quad & a^T y + b^T z + \underline{x}^T u + \overline{x}^T v \\ \text{subject to} \quad & A^T y + B^T z + u + v = c \\ & y \leq 0,\ u \geq 0,\ v \leq 0. \end{aligned} \tag{1.14}$$

Each of the dual variables (y, z, u, v), also called *Lagrange parameters*, can be mapped one-to-one to a primal constraint. The components of y map to the inequality constraints, z maps to the equations, u and v to the simple bounds. We obtain a finite objective value if the Lagrange parameters u_i, v_i that correspond to infinite simple bounds are fixed to 0. The dual feasible region is denoted $\mathcal{Y}(P)$.

Returning to the substitution of a free variable by two nonnegative ones (1.13), we see the ill-posedness of the LP by (1.14). The substitution introduced a new column with opposite sign in the original LP. The corresponding rows in the dual constraints contain two new positive Lagrange parameters for the lower bounds of the nonnegative variables. Consequently we again replaced one equality by two inequalities, this time making the dual ill-posed.

Each feasible point of problem (1.14) provides a lower bound for the objective value of each feasible point of the primal and vice-versa:

$$a^T y + b^T z + \underline{x}^T u + \overline{x}^T v \overset{(1.9)}{\leq} x^T (A^T y + B^T z + u + v) \overset{(1.14)}{=} x^T c. \tag{1.15}$$

An immediate consequence of (1.15) is that a primal unbounded LP has an infeasible dual and vice-versa. But how sharp is this bound otherwise? In linear programming equality holds in (1.15) with the only exception of a primal and dual infeasible LP. Generally one distinguishes between *weak duality* and *strong duality* depending on the features of the problem to solve. If strong duality holds, equality is assumed in (1.15). There are primal and dual feasible points with the same objective value, which is also sufficient for optimality. Strong duality holds when certain so-called *constraint qualifications* hold. One such constraint qualification for linear programming is the existence of feasible points. If either the primal or the dual has a nonempty feasible region, the *duality gap*—the difference between the primal and the dual optimal value—is 0.

An important property of the optimal solutions to the primal and the dual LP is the so-called *complementary slackness*. Existence of optimal solutions implies strong duality, so (1.15) holds with equality. The difference between the

middle and the left hand side has to be zero

$$\begin{pmatrix} a - Ax \\ Bx - b \\ x - \underline{x} \\ \overline{x} - x \end{pmatrix}^T \begin{pmatrix} -y \\ z \\ u \\ -v \end{pmatrix} = 0.$$

The left vector is positive with (at least) p zeros for every primal feasible x ($Bx = b$). The right vector is positive except for z, which gets multiplied by 0. Consequently complementary slackness holds if and only if all nonzero dual variables y, u, v correspond to active constraints (i.e., constraints holding with equality).

If exactly one component of each complementary pair is 0, this is called *strict* complementarity. If there are pairs where both components are zero, the solution is called *degenerate*. One differentiates between *primal* and *dual degeneracy* depending on whether the primal or the dual component is 0 while it does not have to be. The optimal primal solution is determined by n constraints. If there are more than n active constraints at optimality, the solution is primal degenerate. If one of the determining primal constraints has a corresponding 0 dual, the solution is dual degenerate. While degeneracy might seem a corner case, it frequently occurs in practical problems (see Gal [34]), often due to inherent model properties as observed by Greenberg [41].

Complementary slackness gives rise to a nice mechanical interpretation of the dual solution. Assume that we have a mass of $\|c\|$ inside the (nonempty) feasible region and subject it to gravity in direction of $-c$. The primal constraints are interpreted as walls, equality constraints can be seen as two adjacent walls with the mass being in between. The primal optimal solution is the position where the mass comes to rest. The dual optimal solution can be interpreted as the absolute values and directions of the forces the walls exert on the mass. The sign conditions on the dual variables illustrate that walls can only exert force in one direction. They cannot pull the mass away from the interior of the feasible region.

More information on the rich theory of linear programming can be found in Dantzig [16], Vanderbei [118], the Mathematical Programming Glossary [45], and Boyd and Vandenberghe [10]. The latter contains additional references.

To model uncertainties in the parameters, we may replace each of them with an interval. This leads to the family of linear programs $\mathbf{P} := (\mathbf{A}, \mathbf{B}, \mathbf{a}, \mathbf{b}, \mathbf{c})$ with simple bounds $\underline{x}$, $\overline{x}$, where each parameter is allowed to vary within the corresponding interval. We do not consider uncertainties in the simple bounds as these are usually known exactly[1] (e.g., nonnegativity conditions, bounds on available resources, bounds on available storage space).

[1] If they are not, we may use the most constraining values from the interval simple bounds. The verified algorithms work by computing a box that contains solutions for all LPs with parameters in $\mathbf{P}$. A solution for the most constraining simple bounds surely satisfies all others. If there is no feasible solution for the most constraining simple bounds, we cannot find such a box—the algorithms fail.

Algorithms

With the theory of linear programming established, we will now have a brief look at the most important algorithms for solving an LP. A more thorough overview of these methods as well as other algorithms can be found in the surveys of Todd [114] and Megiddo [80].

Simplex algorithm

The development of efficient algorithms for linear programming started with the mentioned invention of the simplex algorithm by Dantzig in 1947. Klee and Minty [69] gave the first example of a class of LPs revealing the exponential behavior of Dantzig's classic method. Since then several variants of the simplex algorithm have been shown to be exponential, while variants with subexponential but superpolynomial behaviour have also been proposed. Typically, however, the simplex algorithm requires at most $2m$ to $3m$ steps to attain optimality. This was already observed by Dantzig [16] and confirmed several times by numerical experiments, which probably delayed the development of further competitive algorithms.

The simplex algorithm builds on the fact that an affine function over a set defined by affine equations and inequalities assumes its extremal values in one or more vertices. The intuitive concept of a vertex generalizes to the n-dimensional space as the intersection of n hyperplanes. Hence a vertex corresponds one-to-one to a *basic index set* of variables also called a *basis* that determines which constraints are active at the vertex (i.e., which hyperplanes define the vertex). A basis is called feasible if the vertex determined by its active constraints is a feasible point of the LP.

The simplex algorithm now moves along the edges of the feasible region from vertex to vertex with nonincreasing objective function value. Because of the affine objective function, there has to be a proper path leading from any starting vertex to the optimal one. Conversely if all neighboring vertices have a larger objective value than the current, optimality is reached.

To start the simplex algorithm, a feasible initial vertex has to be known. This can be computed via a *phase 1* method. The LP (1.9) is changed into a trivially feasible one:

$$\begin{aligned} \min \quad & 1^T s_a + 1^T s_b^+ + 1^T s_b^- + 1^T s_x^+ + 1^T s_x^- \\ \text{subject to} \quad & Ax - s_a \leq a \\ & Bx + s_b^+ - s_b^- = b \\ & x + s_x^+ \geq \underline{x} \\ & x - s_x^- \leq \overline{x} \\ & s_a, s_b^+, s_b^-, s_x^+, s_x^- \geq 0 \end{aligned} \tag{1.16}$$

Starting from the edge

$$\begin{pmatrix} x \\ s_a \\ s_b^+ \\ s_b^- \\ s_x^+ \\ s_x^- \end{pmatrix} = \begin{pmatrix} 0 \\ \max\{0, a\} \\ \max\{b, 0\} \\ \min\{b, 0\} \\ \max\{\underline{x}, 0\} \\ \min\{\overline{x}, 0\} \end{pmatrix},$$

this LP is solved to optimality with the simplex algorithm. If the optimal value is 0, the optimal solution $(x, 0, 0, 0, 0, 0)$ gives a feasible vertex x of the original LP. If the optimal value is greater 0, the original LP is infeasible.

Ellipsoid method

Originally developed by Yudin and Nemirovski [125] and Shor [107] for convex nonlinear programming, the ellipsoid method was used by Khachiyan [66] in 1979 to derive a polynomial-time algorithm for linear programming. It was the first polynomial algorithm for linear programming. But since the average performance is, in contrast to the simplex method, close to its worst-case bound, it is mostly of theoretical interest and not competitive for solving LPs.

The ellipsoid method is a kind of bisection in higher dimensions. It starts with an ellipsoid containing all optimal solutions (e.g., based on the simple bounds). Considering feasibility and optimality of the center of the ellipsoid, a hyperplane determining a half-space of respectively infeasible or suboptimal points is constructed. The ellipsoid with minimal volume is found that contains all points that lie in the old ellipsoid and on the proper side of the hyperplane, and the method iterates. It can be shown that the volumes of the ellipsoids shrink with at least a factor $\exp(-\frac{1}{2n+2}) < 1$.

Interior-point method

Interior-point methods try to find the optimal solution by moving through the interior of the feasible region. Their roots can be traced back to barrier and penalty methods as described in the classic text of Fiacco and McCormick [26]. Interest in interior-point methods was, however, boosted by Karmarkar's [57] discovery of a polynomial-time version in 1984. Having only a slightly better complexity than the ellipsoid method, the interior-point method was of great practical importance because the average performance was considerably better than its worst-case bound.

Interior-point methods generate a sequence of points converging to the optimal solution. In the original version by Karmarkar, this sequence is computed through an alternation of projection and optimization steps. First a projective transformation moves the current iterate into the center of the feasible region. Then a nonlinear potential based on the objective function is optimized over a ball inscribed in the feasible region. While current implementations seldom use projective transformations, the idea of moving the current

iterate in some sense away from the boundary of the feasible region is still important.

Applications

Having its origins in military resource planning, the theory of linear programming found an enormous number of applications. In addition to the areas named in the introduction, the netlib collection of linear programs [87] contains such diverse problems as stochastic forestry problems, oil refinery problems, flap settings of aircraft, pilot models, audit staff scheduling, truss structure problems, airline schedule planning, industrial production and allocation models, image restoration problems, and multisector economic planning problems. Due to the intention of collecting interesting problems, these are mainly LPs that caused difficulties to the solvers.

An abundance of applications can be found in the area of operations research. Several references can be found in the "Interfaces" journal of INFORMS. Some of these are:

Bollapragada et al. [9] installed a system at NBC (National Broadcasting Company) for scheduling commercials with great success. The system uses linear programming to generate sales plans, determining which commercials are aired at what time and how much the advertising company has to pay for these. In four years the system increased revenues by $200 million and reduced manual sales-plan rework by over 80 percent.

LeBlanc et al. [75] developed a set of LPs to assist Nu-kote International in "planning shipments of finished goods between vendors, manufacturing plants, warehouses, and customers." The solution of these LPs suggested ways to improve shipments capable of reducing annual costs by $1 million and customer transit time by two days.

GE (General Electric) Asset Management Incorporated employs an algorithm based on sequential linear programming to optimize more than 30 portfolios valued at over $30 billion. The approach developed by Chalermkraivuth et al. [12] uses a sequence of LPs to model the nonlinearity in the task.

Caixeta-Filho et al. [11] assisted Jan de Wit Company in implementing a decision-support system to manage its lily flower business. "Between 1999 and 2000, company revenue grew 26 percent, sales increased 14.8 percent for pots of lilies and 29.3 percent for bunches of lilies, costs fell from 87.9 to 84.7 percent of sales, income from operations increased 60 percent, return on owner's equity went from 15.1 to 22.5 percent, and best quality cut lilies jumped from 11 to 61 percent of the quantities sold."

Applications from engineering include Huang and Fang [46]. They use an LP to model a soft quality-of-service (soft QoS) problem in wireless sensor networks. Soft QoS, as opposed to end-to-end QoS, uses only local information because path information is not available in wireless networks.

Maki et al. [78] develop a highly sensitive NMR spectrometer with a superconducting split magnet. They use linear programming to design solenoid coils as axial shim coils. These are necessary to accommodate for inhomoge-

neous magnetic fields, which are caused by structural asymmetry and limited production accuracy of the magnet.

Omer et al. [95] use an LP to optimize the design of an Energy Tower. An Energy Tower is a power plant that produces electricity by cooling air at the top of the tower with seawater and harnessing the cold air dropping to the floor of the tower.

Protein interaction is believed to be realized through interaction of certain regions of the proteins called domains. Identifying cooperation of multiple domains can provide deep insights into the mechanics of protein-protein interaction and improve accuracy of protein interaction prediction. Wang et al. [121] employ linear programming as a building block in an algorithm to find cooperative domains and predict protein interaction. Experimental results show this algorithm to offer a higher accuracy in predicting than previous methods.

Fung and Stoeckel [33] try to classify brain perfusion images to diagnose Alzheimer's disease. Their linear-programming-based approach exhibits better performance than current computer techniques or human experts on data gathered from four European institutions.

Mathematical applications include Xia et al. [124]. They want to compute the minimum average Hamming distance of a binary constant weight code with given length, size, and weight. Using methods from coding theory and linear programming, they derive lower bounds that allow to determine the minimum average Hamming distance exactly in several cases.

Applications from chemistry are, for example, Pomerantsev and Rodionova [98] who employ linear programming to solve the problem of multivariate calibration. This is among the oldest but still urgent problems exploited extensively in analytical chemistry. The task is essentially to match a multivariate model to experimental results.

"Stoichiometrically, exact candidate pathways or mechanisms for deriving the rate law [an equation that links the reaction rate with concentrations or pressures of reactants and constant parameters] of a catalytic or complex reaction can be determined through the synthesis of networks of plausible elementary reactions constituting such pathways." Such a method follows the framework of a combinatorial method established by Fan et al. [24] for process-network synthesis. The inclusion or exclusion of a step of each elementary reaction is facilitated by solving linear programming problems comprising a set of mass-balance constraints.

Condition numbers

The condition number of a problem, as already mentioned, describes the sensitivity of the solution to changes in the input data. Renegar [99, 100] introduced the appropriate measures for conic formulations of linear programming problems. He defines the distance to primal and dual infeasibility as the norm of the smallest perturbation of the LP parameters that renders the problem respectively primal and dual infeasible. Dividing the norm of the problem

instance by the minimal distance to infeasibility he arrives at the condition number for an LP.

Ordóñez and Freund [97] derived a linear programming characterization of the condition number of an LP based on the work of Renegar. They observed that the conic formulation used by Renegar results in different condition numbers depending on the transformation used to put an LP into conic format. To circumvent this problem and make LPs as they are submitted to Linear Programming solvers amenable to computational analysis, they extended the theory to the *ground-set* formulation

$$\begin{aligned} f^*(d) &:= \min_{x \in \mathcal{X}(d)} c^T x \\ \mathcal{X}(d) &:= \{x \in \mathbb{R}^n \mid Ax - b \in C_Y,\ x \in S\}. \end{aligned} \tag{1.17}$$

A specific problem in ground-set formulation is defined by its input data $d = (A, b, c)$, consisting of the real $m \times n$ matrix A and the real vectors $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. The set $C_Y \subseteq \mathbb{R}^m$ is a closed convex cone; the set $S \subseteq \mathbb{R}^n$ is closed and convex.

The corresponding dual problem is

$$\begin{aligned} f^*(d) &:= \max_{(y,v) \in \mathcal{Y}(d)} b^T y - v \\ \mathcal{Y}(d) &:= \{(y, v) \in \mathbb{R}^{m \times n} \mid (c - A^T y, v) \in C_S^*,\ y \in C_Y^*\}. \end{aligned} \tag{1.18}$$

Here C_Y^* denotes the dual cone of C_Y

$$C_Y^* := \{y \in \mathbb{R}^m \mid z^T y \geq 0 \ \text{ for all } \ z \in C_Y\},$$

and C_S^* denotes the dual cone of

$$C_S := \{(x, \theta) \mid x \in \theta S \ \text{ and } \ \theta > 0\}.$$

The distances to primal and dual infeasibility are defined as

$$\rho_P(d) := \inf\{\|\Delta d\| \mid \mathcal{X}(d + \Delta d) = \emptyset\} \tag{1.19}$$

$$\rho_D(d) := \inf\{\|\Delta d\| \mid \mathcal{Y}(d + \Delta d) = \emptyset\}, \tag{1.20}$$

where $\Delta d := (\Delta A, \Delta b, \Delta c)$ and

$$\|\Delta d\| := \max\{\|\Delta A\|, \|\Delta b\|_1, \|\Delta c\|_\infty\}.$$

The norm $\|\Delta A\|$ is the corresponding operator norm.

The condition number of a linear program is defined as the quotient of the norm of the input data and the smallest distance to infeasibility,

$$C(d) := \frac{\|d\|}{\min\{\rho_P(d), \rho_D(d)\}}. \tag{1.21}$$

A problem is ill-posed if the minimal distance to primal or dual infeasibility is 0 or equivalently its condition number $C(d)$ is ∞.

Using the displayed norms, Ordóñez and Freund show that the distances to infeasibility can be computed by solving $2n + 2m$ linear programming problems of size roughly that of the original problem. *This immediately makes any verification method for linear programming problems a method to compute rigorous distances to infeasibility. If additionally the norm of the problem is verified, we obtain rigorous condition numbers for LPs.*

Ordóñez and Freund approximately compute and list the condition numbers for the problems in the netlib LP library.

Our linear programming format (1.9) can be described in the ground-set format by aggregating the equality and inequality constraints to

$$\begin{pmatrix} A \\ B \end{pmatrix} x - \begin{pmatrix} a \\ b \end{pmatrix} \in C_Y := \begin{pmatrix} \mathbb{R}^m_- \\ 0 \end{pmatrix}$$

and using $S := \{x \in \mathbb{R}^n \mid \underline{x} \leq x \leq \overline{x}\}$. This transformation yields the condition number for our format. To denote this transformation and the calculation of distances to infeasibility and condition number, we will use $\rho_P(P), \rho_D(P)$, and $C(P)$ from now on. If the parameter set P is clear from the context, we drop it and just write ρ_P, ρ_D, and C.

1.5 Rigorous bounds

> *"In real-world applications of Linear Programming one cannot ignore the possibility that a small uncertainty in the data (intrinsic for most real-world LP programs) can make the usual optimal solution of the problem completely meaningless from a practical viewpoint."*
>
> —A. Ben-Tal and A. S. Nemirovski, 2000

Considering rounding errors in linear programming began with Krawczyk [73]. His aim was to compute a rigorous enclosure of the optimal solution by verifying the optimality of a basic index set. From this enclosure rigorous bounds on the optimal value can be derived easily. Krawczyk 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 [4] and Rump [104]. Jansson [51] 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 (i.e., basic index sets differing only in one member). 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, Jansson [52] and Neumaier and Shcherbina [92] devised methods to rigorously bound the optimal value with a quadratic complexity. This is achieved by deriving the rigorous bounds from duality properties. 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.

The theorems by Jansson are repeated here for reference adapted to our notation. The details can be found in [52]. The basic idea of the rigorous bounds is to determine interval vectors that contain a feasible solution for every $P \in \mathbf{P}$ being in the relative interior of the feasible region. This solution should be close to an optimal solution but sufficiently far away from degeneracy and infeasibility. Favorable characteristics of the primal enclosure $\mathbf{x}$ are given by next theorem.

Theorem 1 (Upper bound) *Let $\mathbf{P} = (\mathbf{A}, \mathbf{B}, \mathbf{a}, \mathbf{b}, \mathbf{c})$ be a family of linear programming problems with input data $P \in \mathbf{P}$ and simple bounds $\underline{x} \leq \overline{x}$. Suppose that there exists an interval vector $\mathbf{x} \in \mathbb{IR}^n$ such that*

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

and

$$\forall\, B \in \mathbf{B},\ b \in \mathbf{b}\ \exists\, x \in \mathbf{x} :\ Bx = b.$$

Then for every $P \in \mathbf{P}$ there exists a primal feasible solution $x(P) \in \mathbf{x}$, and the inequality

$$\sup_{P \in \mathbf{P}} f^*(P) \quad \leq \quad f^{\triangle} := \max\{\mathbf{c}^T\mathbf{x}\} \tag{1.22}$$

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

The importance of the last sentence is not to be underestimated. If a rigorous upper and lower bound can be computed, they provide a certificate of the existence of optimal solutions.

The dual enclosure $(\mathbf{y}, \mathbf{z}, \mathbf{u}, \mathbf{v})$ is characterized in the following way.

Theorem 2 (Lower bound) *Let $\mathbf{P} = (\mathbf{A}, \mathbf{B}, \mathbf{a}, \mathbf{b}, \mathbf{c})$ be a family of linear programming problems with input data $P \in \mathbf{P}$ and simple bounds $\underline{x} \leq \overline{x}$. Suppose that there exist interval vectors $\mathbf{y} \in \mathbb{IR}^m$ and $\mathbf{z} \in \mathbb{IR}^p$ such that*

(i) the sign condition

$$\mathbf{y} \leq 0$$

holds true,

(ii) for $i \in \ddot{\mathcal{B}}$ the equations

$$\forall\, A \in \mathbf{A}, B \in \mathbf{B}, c \in \mathbf{c}\ \exists\, y \in \mathbf{y}, z \in \mathbf{z} :$$
$$(A_{:i})^T y + (B_{:i})^T z = c_i$$

are fulfilled,

(iii) and for the remaining i the intervals

$$\mathbf{d}_i := \mathbf{c}_i - (\mathbf{A}_{:i})^T\mathbf{y} - (\mathbf{B}_{:i})^T\mathbf{z} \tag{1.23}$$

satisfy the inequalities

$$\mathbf{d}_i \leq 0 \quad \text{if } \underline{x}_i = -\infty$$
$$\mathbf{d}_i \geq 0 \quad \text{if } \overline{x}_i = +\infty.$$

Then the inequality

$$\inf_{P\in\mathbf{P}} f^*(P) \quad \geq \quad f^{\triangledown} := \min\{\mathbf{a}^T\mathbf{y} + \mathbf{b}^T\mathbf{z} + \sum_{i\in\underline{\mathcal{B}}} \underline{x}_i \mathbf{d}_i^+ + \sum_{i\in\overline{\mathcal{B}}} \overline{x}_i \mathbf{d}_i^-\} \qquad (1.24)$$

is fulfilled, and $f^{\triangledown}$ is a finite lower bound of the global minimum value. Moreover, if

(a) all input data are point data (i.e. $P = \mathbf{P}$),

(b) P has an optimal solution (y^*, z^*, u^*, v^*),

(c) $\mathbf{y} := y^*$, $\mathbf{z} := z^*$,

(d) the quantities in (1.23) and (1.24) are calculated exactly,

then the conditions (i),(ii), and (iii) are satisfied, and the optimal value $f^*(P) = f^{\triangledown}$; that is, this lower error bound is sharp for point input data and exact computations.

In the special case where all simple bounds are finite, the conditions (ii) and (iii) of Theorem 2 are trivially satisfied. Therefore each nonnegative interval vector $\mathbf{y}$ delivers a rigorous lower bound (1.24) with $\mathcal{O}(n^2)$ operations. Jansson suggests that if some finite simple bounds, however, are very large, tighter bounds can be obtained at additional costs by setting these simple bounds to $\pm\infty$ and computing a rigorous lower bound for the resulting LP.

Note that the previous analysis gives a rigorous certificate for the existence of optimal solutions if both bounds $f^{\triangledown}$ and $f^{\triangle}$ are finite.

The necessary interval vectors $\mathbf{x}, \mathbf{y}$, and $\mathbf{z}$ can be computed with Algorithms 1 and 2. In contrast to [52] we enforce the original simple bounds and not the perturbed ones in step 3 of Algorithm 1. This may result in a sharper bound at the cost of more iterations. But differences should only occur in corner cases, as the simple bounds are usually approximately satisfied.

Convergence

To analyze the convergence of the algorithms, we need to make some assumptions about the computations performed therein. In the following we assume the arithmetic conforms to IEEE 754 with rounding mode set to nearest; all basic operations are carried out with the machine epsilon ε_A. We assume that the LP solver returns approximate solutions that satisfy the constraints up to a relative error of ε_{lp} if the problem is feasible. Finally the interval solver used to solve $\mathbf{A}x = \mathbf{b}$ shall return enclosures with a radius of order $\mathcal{O}(\varepsilon_{Int})$. As shown by Neumaier [89] this can be guaranteed if $\mathring{\mathbf{A}}$, $\mathring{\mathbf{b}}$, $R\check{\mathbf{A}} - I$, $\check{\mathbf{b}} - \check{\mathbf{A}}\tilde{x}$ are all of order $\mathcal{O}(\sqrt{\varepsilon_{Int}})$ with small $\sqrt{\varepsilon_{Int}}$. The matrix R is an approximate inverse of $\check{\mathbf{A}}$, and $\tilde{x}$ is an approximate solution to $\check{\mathbf{A}}x = \check{\mathbf{b}}$; both are input to the interval solver.

Before we look at the convergence of the bound computing algorithms, we establish a connection between the existence of strictly primal feasible points and the distance to infeasibility.

Algorithm 1 Upper bound

1. Take arbitrary positive $t^a \in \mathbb{R}^m$, $t^{\underline{x}}$, $t^{\overline{x}} \in \mathbb{R}^n$.

2. Solve $P(t) := (\check{\mathbf{A}}, \check{\mathbf{B}}, \check{\mathbf{a}} - t^a, \check{\mathbf{b}}, \check{\mathbf{c}})$ with simple bounds

$$\underline{x}_i(t) := \begin{cases} \underline{x}_i & \text{if } \underline{x}_i = -\infty \\ \underline{x}_i + t^{\underline{x}}_i & \text{otherwise,} \end{cases}$$

   and

$$\overline{x}_i(t) := \begin{cases} \overline{x}_i & \text{if } \overline{x}_i = +\infty \\ \overline{x}_i - t^{\overline{x}}_i & \text{otherwise.} \end{cases}$$

   If the approximate solver does not compute a solution $\tilde{x}$, repeat step 2 with smaller t^a, $t^{\underline{x}}$, $t^{\overline{x}}$.

3. Enforce $\underline{x} \leq \tilde{x} \leq \overline{x}$ by

$$\tilde{x}_i = \begin{cases} \underline{x}_i & \text{if } \tilde{x}_i < \underline{x}_i \\ \overline{x}_i & \text{if } \tilde{x}_i > \overline{x}_i \\ \tilde{x}_i & \text{otherwise.} \end{cases}$$

4. If the problem does not contain equality constraints, check if $\mathbf{A}\tilde{x} \leq \mathbf{a}$, $\underline{x} \leq \tilde{x} \leq \overline{x}$ holds. If yes, return

$$f^{\triangle} := \sup\{\mathbf{c}^T \tilde{x}\}$$

   as the upper bound. If not, increase t^a, $t^{\underline{x}}$, $t^{\overline{x}}$ and go to step 2.

5. Compute an enclosure $\mathbf{x}$ of the solution of $\mathbf{B}\tilde{x} = \mathbf{b}$. Check if $\mathbf{x}$ satisfies $\sup\{\mathbf{A}\mathbf{x}\} \leq \mathbf{a}$, $\underline{x} \leq \mathbf{x} \leq \overline{x}$. If it does, return

$$f^{\triangle} := \sup\{\mathbf{c}^T \mathbf{x}\}$$

   as the upper bound. If not, increase t^a, $t^{\underline{x}}$, $t^{\overline{x}}$ and go to step 2.

Algorithm 2 Lower bound

1. Take arbitrary $t^c > 0$.

2. Solve $P(t) := (\check{\mathbf{A}}, \check{\mathbf{B}}, \check{\mathbf{a}}, \check{\mathbf{b}}, c(t))$ with simple bounds $\underline{x}, \overline{x}$. Set

$$c_i(t) := \begin{cases} \check{\mathbf{c}}_i + t_i^c & \text{if } i \in \overline{\ddot{\mathcal{B}}} \\ \check{\mathbf{c}}_i - t_i^c & \text{if } i \in \underline{\ddot{\mathcal{B}}} \\ \check{\mathbf{c}}_i & \text{otherwise.} \end{cases}$$

   If the approximate solver does not compute a solution $(\tilde{y}, \tilde{z}, \tilde{u}, \tilde{v})$, repeat step 2 with smaller t^c.

3. Enforce $\tilde{y} \leq 0$ by

$$\tilde{y}_i = \begin{cases} 0 & \text{if } \tilde{y}_i > 0 \\ \tilde{y}_i & \text{otherwise.} \end{cases}$$

4. If the problem does not contain free variables, check if

$$\begin{aligned} \mathbf{d}_i &:= c_i - (\mathbf{A}_{:i})^T \tilde{y} - (\mathbf{B}_{:i})^T \tilde{z} \leq 0 && \text{for } \underline{x}_i = -\infty \\ \mathbf{d}_i &:= c_i - (\mathbf{A}_{:i})^T \tilde{y} - (\mathbf{B}_{:i})^T \tilde{z} \geq 0 && \text{for } \overline{x}_i = +\infty \end{aligned}$$

   holds. If it does, return

$$f^{\triangledown} := \inf\{\mathbf{a}^T \tilde{y} + \mathbf{b}^T \tilde{z} + \sum_{i \in \underline{\mathcal{B}}} \underline{x}_i \mathbf{d}_i^+ + \sum_{i \in \overline{\mathcal{B}}} \overline{x}_i \mathbf{d}_i^-\}$$

   as the lower bound. Otherwise increase t^c and go to step 2.

5. Compute enclosures $\mathbf{y}, \mathbf{z}$ of the solution of $(\mathbf{A}_{:i})^T \tilde{y} + (\mathbf{B}_{:i})^T \tilde{z} = \mathbf{c}_i$ for $i \in \ddot{\mathcal{B}}$. Check if these satisfy

$$\begin{aligned} \mathbf{y} &\leq 0 \\ \mathbf{d}_i &:= \mathbf{c}_i - (\mathbf{A}_{:i})^T \mathbf{y} - (\mathbf{B}_{:i})^T \mathbf{z} \leq 0 && \text{for } i \in \overline{\ddot{\mathcal{B}}} \\ \mathbf{d}_i &:= \mathbf{c}_i - (\mathbf{A}_{:i})^T \mathbf{y} - (\mathbf{B}_{:i})^T \mathbf{z} \geq 0 && \text{for } i \in \underline{\ddot{\mathcal{B}}}. \end{aligned}$$

   If yes, return

$$f^{\triangledown} := \inf\{\mathbf{a}^T \mathbf{y} + \mathbf{b}^T \mathbf{z} + \sum_{i \in \underline{\mathcal{B}}} \underline{x}_i \mathbf{d}_i^+ + \sum_{i \in \overline{\mathcal{B}}} \overline{x}_i \mathbf{d}_i^-\}$$

   as the lower bound. Otherwise increase t^c and go to step 2.

Lemma 1 *Let the set $\mathcal{V}$ contain the indices of variables with different simple bounds (i.e., $\underline{x}_i \neq \overline{x}_i$). For an LP the following are equivalent*

$$\rho_P > 0 \tag{1.25}$$

$$\exists x \in \mathcal{X}(P) : Ax < a,\ \underline{x}_\mathcal{V} < x_\mathcal{V} < \overline{x}_\mathcal{V} \ \textit{and}\ \ \operatorname{rank} B_{:\mathcal{V}} = p \tag{1.26}$$

$$\operatorname{relint} \mathcal{X}(P) \neq \emptyset \ \textit{and}\ \ \operatorname{rank} B_{:\mathcal{V}} = p. \tag{1.27}$$

Proof. First we prove the equivalence of (1.25) and (1.26). Condition (1.25) implies (1.26) by the contraposition. If (1.26) does not hold, either each feasible point $x \in \mathcal{X}(P)$ violates (at least) one of

1. $Ax < a$
2. $\underline{x}_\mathcal{V} < x_\mathcal{V} < \overline{x}_\mathcal{V}$

or

3. the matrix $B_{:\mathcal{V}}$ is rank deficient.

We will look at these in turn and construct an arbitrarily small perturbation that makes the LP infeasible; the LP is ill-posed and $\rho_P = 0$, (1.25) is false.

1. If each feasible point violates $Ax < a$, one inequality holds with equality for all feasible point $A_{i:}x = a_i$. Subtracting a small positive amount from a_i makes all feasible points violate this constraint. The feasible region becomes empty.

2. An LP with a variable x_v, $v \in \mathcal{V}$, that is without loss of generality forced to its lower bound $\underline{x}_v$ is trivially infeasible for a small perturbation $\underline{x}_v(\tau) = \underline{x}_v + \tau$, $\overline{x}_v(\tau) = \overline{x}_v + \tau$. As ρ_P does not consider perturbations of the simple bounds, we transform the perturbed LP into an equivalent one with a perturbation of the right hand sides only. With e_i being the ith unit vector, the constraints of the perturbed LP can be written

$$\begin{aligned} A(x - \tau e_v) + \tau A_{:v} &\leq a \\ B(x - \tau e_v) + \tau B_{:v} &= b \\ \underline{x} \leq x - \tau e_v &\leq \overline{x}. \end{aligned}$$

A variable substitution $x_v - \tau \to x_v$ yields the perturbation

$$\begin{aligned} Ax &\leq a(\tau) \qquad & \Delta a &:= -\tau A_{:v} \\ Bx &= b(\tau) \qquad & \Delta b &:= -\tau B_{:v} \\ \underline{x} \leq x &\leq \overline{x}. \end{aligned}$$

3. Finally if $B_{:\mathcal{V}}$ is rank deficient, its columns do not form a basis of $\mathbb{R}^p$. There are vectors in $\mathbb{R}^p$ that are orthogonal to the columns of $B_{:\mathcal{V}}$. Perturbing b in one of these directions there is either no solution of $Bx = b$ or it requires a change of the fixed variables. The feasible region becomes empty.

Taken together we know that (1.25) implies (1.26). Next we prove that the converse is also valid. The feasible point satisfying (1.26) stays feasible under the perturbations considered by ρ_P.

Perturbations of c do not alter the feasible region.

No perturbation of a changes $Ax < a$ into $Ax > a$; neither do perturbations of A because their impact on the value of the constraints is limited

$$\|(A + \Delta A)x\| \leq \|Ax\| + \|\Delta Ax\| \leq \|Ax\| + \|\Delta A\|\|x\|.$$

With full column rank of $B_{:\mathcal{V}}$ small changes in B and b result in bounded changes of $x_{\mathcal{V}}$. Since all inequalities are strictly valid, they also hold for these small changes in $x_{\mathcal{V}}$. This establishes equivalence of (1.25) and (1.26).

The equivalence of (1.26) and (1.27) is a result of the definition of the relative interior. This is the set of points having a neighborhood that lies in $\mathcal{X}(P)$ when intersected with the affine hull of $\mathcal{X}(P)$. The affine hull of $\mathcal{X}(P)$ is the set of all points satisfying $Bx = b$. So (1.27) is another way of writing (1.26). ■

With Lemma 1, we can now show that Algorithm 1 succeeds in one iteration for point problems with $\rho_P > 0$.

Theorem 3 *If an LP has a positive distance to primal infeasibility*

$$\rho_P > 0,$$

Algorithm 1 delivers an upper bound in one iteration, provided the accuracy of the computations is high enough.

Proof. The idea of the proof is to bound the errors introduced in each step of Algorithm 1, and to derive limits for the deflation parameters that allow the algorithm to terminate successfully. These limits can be met for sufficiently high accuracies due to the distance to infeasibility being greater zero.

The condition $\rho_P > 0$ is by Lemma 1 equivalent to

$$\exists x \in \mathcal{X}(P) : Ax < a,\ \underline{x}_{\mathcal{V}} < x_{\mathcal{V}} < \overline{x}_{\mathcal{V}} \text{ and } \mathtt{rank}\, B_{:\mathcal{V}} = p. \tag{1.28}$$

Therefore the perturbed problem $P(t)$ solved in step 2 of Algorithm 1 has a feasible point provided that t^a, $t^{\underline{x}}$, and $t^{\overline{x}}$ are less than a positive $\bar{t}$ determined by $\rho_P(P(t))$ and the machine epsilon ε_A. The feasible set $\mathcal{X}(P(t))$ is nonempty, and the LP solver computes an approximation $\tilde{x}$ with

$$\begin{aligned}
A\tilde{x} &\leq a - t^a + |a - t^a|(\varepsilon_{lp} + \varepsilon_A) \\
B\tilde{x} &\in b \pm \varepsilon_{lp}|b| \\
\tilde{x}_i &\geq \underline{x}_i + t_i^{\underline{x}} - (\varepsilon_{lp} + \varepsilon_A)|\underline{x}_i + t_i^{\underline{x}}| \quad \text{for } \underline{x}_i \neq -\infty \\
\tilde{x}_i &\leq \overline{x}_i - t_i^{\overline{x}} + (\varepsilon_{lp} + \varepsilon_A)|\overline{x}_i - t_i^{\overline{x}}| \quad \text{for } \overline{x}_i \neq +\infty.
\end{aligned}$$

Enforcing the simple bounds in step 3 of Algorithm 1 changes $\tilde{x}$ into $\tilde{x} + \Delta\underline{x}$ with

$$\Delta\underline{x}_i := \begin{cases} -t_i^{\underline{x}} + (\varepsilon_{lp} + \varepsilon_A)|\underline{x}_i + t_i^{\underline{x}}| & \text{if } \tilde{x} < \underline{x}_i \\ -t_i^{\overline{x}} + (\varepsilon_{lp} + \varepsilon_A)|\overline{x}_i - t_i^{\overline{x}}| & \text{if } \tilde{x} > \overline{x}_i \\ 0 & \text{otherwise.} \end{cases}$$

The new $\tilde{x}$ satisfies

$$A\tilde{x} \leq a - t^a + (\varepsilon_{lp} + \varepsilon_A)|a - t^a| + |A|\Delta\underline{x}$$
$$B\tilde{x} \in b \pm \left(\varepsilon_{lp}|b| + |B|\Delta\underline{x}\right).$$

For a linear program without equality constraints, we now check if $\mathbf{A}\tilde{x} \leq \mathbf{a}$, $\underline{x} \leq \tilde{x} \leq \overline{x}$ holds. This is the case when the deflation parameters satisfy

$$t^a \geq \frac{(\varepsilon_{lp} + \varepsilon_A)|a| + |A|\Delta\underline{x}}{1 - \varepsilon_{lp} - \varepsilon_A} \tag{1.29}$$
$$t^{\underline{x}} \geq 0 \tag{1.30}$$
$$t^{\overline{x}} \geq 0. \tag{1.31}$$

If the accuracies of the used components are high enough, the fraction in (1.29) goes to 0. The upper bound $\bar{t}$ on t^a, $t^{\underline{x}}$, and $t^{\overline{x}}$ can be met. Algorithm 1 verifies feasibility of $\tilde{x}$, and returns the upper bound $c^T\tilde{x}$.

For an LP with equality constraints, (1.28) implies the existence of a regular $p \times p$ submatrix $B_{:\mathcal{R}}$ of B. There exists an $\hat{x}$ with $B\hat{x} = b$ that differs from $\tilde{x}$ only in the p components specified by $\mathcal{R}$. The distance between $\tilde{x}$ and $\hat{x}$ is bounded by

$$\|\hat{x} - \tilde{x}\|_\infty \leq \Delta\tilde{x} := \kappa(B_{:\mathcal{R}}) \cdot \frac{\|\Delta b\|_\infty}{\|b\|_\infty} \cdot \|\hat{x}\|_\infty$$
$$\text{with} \quad \Delta b := \varepsilon_{lp}|b| + |B|\Delta\underline{x}$$

The interval solver computes an enclosure $\mathbf{x}$ with radius $\mathring{\mathbf{x}}$ of order $\mathcal{O}(\varepsilon_{Int})$ that contains $\hat{x}$. Hence the maximal distance between points in $\mathbf{x}$ and $\tilde{x}$ is bounded by

$$\max\{\|\sup\{\mathbf{x}\} - \tilde{x}\|_\infty, \|\inf\{\mathbf{x}\} - \tilde{x}\|_\infty\} \leq \Delta\tilde{x} + \mathcal{O}(\varepsilon_{Int})$$

In turn $\mathbf{x}$ satisfies

$$\underline{x} + \Delta\underline{x} - (\Delta\tilde{x} + \mathcal{O}(\varepsilon_{Int}))e \leq \mathbf{x} \leq \overline{x} - \Delta\underline{x} + (\Delta\tilde{x} + \mathcal{O}(\varepsilon_{Int}))e$$

and

$$\sup\{A\mathbf{x}\} \leq a - t^a + (\varepsilon_{lp} + \varepsilon_A)|a - t^a| + |A|(\Delta\underline{x} + \Delta\tilde{x}e + \mathcal{O}(\varepsilon_{Int})e).$$

The ensuing check for feasibility, $\sup\{A\mathbf{x}\} \leq a$, $\underline{x} \leq \mathbf{x} \leq \overline{x}$, is therefore successful if the deflation parameters satisfy

$$t^a \geq \frac{(\varepsilon_{lp} + \varepsilon_A)|a| + |A|(\Delta\underline{x} + \Delta\tilde{x}e + \mathcal{O}(\varepsilon_{Int})e)}{1 - \varepsilon_{lp} - \varepsilon_A} \tag{1.32}$$
$$t_i^{\underline{x}} \geq \frac{(\varepsilon_{lp} + \varepsilon_A)|\underline{x}_i| + \Delta\tilde{x} + \mathcal{O}(\varepsilon_{Int})}{1 - \varepsilon_{lp} - \varepsilon_A} \qquad \text{for } \underline{x}_i \neq -\infty \tag{1.33}$$
$$t_i^{\overline{x}} \geq \frac{(\varepsilon_{lp} + \varepsilon_A)|\overline{x}_i| + \Delta\tilde{x} + \mathcal{O}(\varepsilon_{Int})}{1 - \varepsilon_{lp} - \varepsilon_A} \qquad \text{for } \overline{x}_i \neq +\infty. \tag{1.34}$$

As in the equation free case, the terms on the right hand side go to 0 if the accuracies of the computations are high enough. The upper bound $\bar{t}$ on t^a, $t^{\underline{x}}$, $t^{\overline{x}}$ can again be met. Algorithm 1 finds the box $\mathbf{x}$ containing the feasible point $\hat{x}$. The upper bound $f^{\triangle}$ is $\sup\{c^T\mathbf{x}\}$. ∎

It is important to note that the converse of Theorem 3 does not hold true. Even for problems with a zero distance to primal infeasibility, Algorithm 1 can compute an upper bound. Consider for example

$$\begin{aligned} \min \quad & x \\ \text{subject to} \quad & x = 0 \\ & 0 \le x \le 1. \end{aligned}$$

This problem becomes infeasible for an arbitrarily small increase of the lower bound 0 of x, hence $\rho_P = 0$. If the interval solver returns the exact 0 for $\mathbf{x}$, the algorithm directly delivers the upper bound 0. While this is a very simple example, the scenario occurs in real-world problems, as in *adlittle* from the netlib collection of linear programs [87]. The equality constraints force the 96th variable "`...195`" to its simple lower bound 0. The relative interior becomes empty. Nevertheless Lurupa successfully delivers an upper bound because the interval solver computes the enclosure $\mathbf{x}$ without overestimation in this component.

Similar relations hold for the dual problem and the lower bound. The connection between the distance to dual infeasibility and the existence of a strictly dual feasible point is given by the following lemma.

Lemma 2 *For an LP the following are equivalent*

$$\rho_D > 0 \tag{1.35}$$

$$\exists (y,z,u,v) \in \mathcal{Y}(P) : y < 0,\ u_{\underline{\mathcal{B}}} > 0,\ v_{\overline{\mathcal{B}}} < 0 \quad \textit{and} \quad \operatorname{rank}\left(A^T B^T\right)_{\ddot{\mathcal{B}}:} = |\ddot{\mathcal{B}}| \tag{1.36}$$

$$\operatorname{relint} \mathcal{Y}(P) \neq \emptyset \quad \textit{and} \quad \operatorname{rank}\left(A^T B^T\right)_{\ddot{\mathcal{B}}:} = |\ddot{\mathcal{B}}|. \tag{1.37}$$

Proof. The proof is analog to the primal case. The equivalence of (1.36) and (1.37) follows again from the definition of the relative interior, keeping in mind that components of u and v which belong to infinite simple bounds are fixed to 0.

It can be seen by the contraposition that (1.35) implies (1.36). If (1.36) does not hold, each dual feasible (y,z,u,v) violates (at least) one of $y < 0$, $u_{\underline{\mathcal{B}}} > 0$, $v_{\overline{\mathcal{B}}} < 0$, or the matrix $\left(A^T B^T\right)_{\ddot{\mathcal{B}}:}$ is rank deficient.

If one of the strict sign conditions is violated, the perturbation follows along the lines of the proof to Lemma 1. Is y_i forced to 0 for example, we perturb its lower bound to $\underline{y}_i = \tau > 0$. A variable transformation yields the infeasible set of constraints

$$A^T y + B^T z + u + v = c(\tau) \qquad \Delta c := -\tau \left(A^T\right)_{:i}$$
$$y \le 0,\ u \ge 0,\ v \le 0$$

with an arbitrarily small perturbation Δc; hence $\rho_D = 0$ and (1.35) is false.

For a rank deficient matrix $\left(A^T B^T\right)_{\ddot{\mathcal{B}}:}$, there are vectors in $\mathbb{R}^{|\ddot{\mathcal{B}}|}$ orthogonal to all matrix columns. Making a small perturbation Δc in one of these directions makes the corresponding part of the dual constraints

$$\left(A^T\right)_{\ddot{\mathcal{B}}:} y + \left(B^T\right)_{\ddot{\mathcal{B}}:} z = c_{\ddot{\mathcal{B}}} + \Delta c_{\ddot{\mathcal{B}}}$$

infeasible. The distance to dual infeasibility is 0. This establishes (1.35) implying (1.36).

Conversely, if there is a feasible point satisfying (1.36) and $\left(A^T B^T\right)_{\ddot{\mathcal{B}}:}$ has full rank, (1.35) follows. There cannot be an arbitrarily small perturbation rendering the LP dual infeasible.

Perturbations of a and b do not alter dual feasibility.

Considering perturbations of A, B, and c, we look at the individual dual constraints. Each of them corresponds to a primal variable. If variable x_i has finite bounds, u_i and v_i are allowed to be different from zero. Every perturbation of A, B, and c can be accounted for by setting u_i and v_i to appropriate values (these are the positive and negative parts of the intervals $\mathbf{d}_i$ in Theorem 2). If x_i has a finite lower bound, u_i is greater zero by (1.36). Small changes in A and B result in bounded changes of the left hand side value of

$$\left(A^T\right)_{i:} y + \left(B^T\right)_{i:} z + u_i + v_i = c_i.$$

These and small changes of c_i can be made up for by small changes in u_i, whether v_i is allowed to be different from 0 or not (i.e., irrespective of $\overline{x}_i$). A similar argument holds for a finite upper bound of x_i. For free variables $x_{\ddot{\mathcal{B}}}$ we have to satisfy

$$\left(A^T\right)_{\ddot{\mathcal{B}}:} y + \left(B^T\right)_{\ddot{\mathcal{B}}:} z = c_{\ddot{\mathcal{B}}}.$$

According to (1.36), however, the left hand side has full rank. Small perturbations of A, B, and c result in bounded changes of y and z that cannot violate dual feasibility. ∎

As in the primal case, we can now prove that a positive distance to dual infeasibility enables Algorithm 2 to find a lower bound for a point problem in one iteration.

Theorem 4 *If an LP has a positive distance to dual infeasibility*

$$\rho_D > 0,$$

Algorithm 2 delivers a lower bound in one iteration, provided the accuracy of the computations is high enough.

Proof. The proof follows the same lines as the proof of Theorem 3. A nonzero distance to dual infeasibility $\rho_D > 0$ is by Lemma 2 equivalent to

$$\begin{aligned} &\exists (y, z, u, v) \in \mathcal{Y}(P) : y < 0,\ u_{\underline{\mathcal{B}}} > 0,\ v_{\overline{\mathcal{B}}} < 0 \\ &\qquad \text{and} \quad \operatorname{rank}\left(A^T B^T\right)_{\ddot{\mathcal{B}}:} = |\ddot{\mathcal{B}}|. \end{aligned} \tag{1.38}$$

Therefore the perturbed problem $P(t)$ in step 2 of Algorithm 2 is feasible for t^c less than an upper bound $\overline{t}$. Let $e(t)$ be the n-dimensional indicator vector of the dual deflation

$$e_i(t) := \begin{cases} 1 & \text{if } c_i(t) > c_i, \\ -1 & \text{if } c_i(t) < c_i, \\ 0 & \text{otherwise.} \end{cases}$$

The approximate solver computes a solution satisfying

$$\begin{aligned} A^T\tilde{y} + B^T\tilde{z} + \tilde{u} + \tilde{v} &\in c(t) \pm \big((\varepsilon_{lp} + \varepsilon_A)|c(t)|\big) \\ \tilde{y} \le \varepsilon_{lp},\ \tilde{u} &\ge -\varepsilon_{lp},\ \tilde{v} \le \varepsilon_{lp} \\ \tilde{u}_i = 0 \quad &\text{for } \underline{x}_i = -\infty \\ \tilde{v}_i = 0 \quad &\text{for } \overline{x}_i = +\infty. \end{aligned}$$

Since the simple bounds for the dual variables are 0, we assume these to be satisfied up to an absolute ε_{lp}.

Enforcing $\tilde{y} \le 0$ in step 3 of the algorithm results in

$$A^T\tilde{y} + B^T\tilde{z} + \tilde{u} + \tilde{v} \in c(t) \pm \big((\varepsilon_{lp} + \varepsilon_A)|c(t)| + \varepsilon_{lp}|A^T|e\big).$$

In the absence of free variables we compute the defects $\mathbf{d}_i$ for all i with $\underline{x}_i = -\infty$ or $\overline{x}_i = \infty$

$$\begin{aligned} \mathbf{d}_i &= c_i - (A^T)_{i:}\tilde{y} - (B^T)_{i:}\tilde{z} \\ &= \tilde{u}_i + \tilde{v}_i - e_i(t)t_i^c \\ &\quad \pm \big((\varepsilon_{lp} + \varepsilon_A)|c(t)_i| + \varepsilon_{lp}|(A^T)_{i:}|e + \varepsilon_A|c_i - (A^T)_{i:}\tilde{y} - (B^T)_{i:}\tilde{z}|\big). \end{aligned}$$

For an infinite simple bound $\underline{x}_i$ or $\overline{x}_i$, the sum $\tilde{u}_i + \tilde{v}_i$ is less equal ε_{lp} or greater equal $-\varepsilon_{lp}$, respectively. The sign condition of step 4 is satisfied if the deflation parameter satisfies

$$\begin{aligned} t_i^c \ge \frac{\varepsilon_{lp}(1 + |c(t)_i| + |(A^T)_{i:}|e)}{1 - \varepsilon_{lp} - \varepsilon_A} \\ + \frac{\varepsilon_A(|c(t)_i| + |c_i - (A^T)_{i:}\tilde{y} - (B^T)_{i:}\tilde{z}|)}{1 - \varepsilon_{lp} - \varepsilon_A}. \end{aligned} \tag{1.39}$$

The numerator on the right hand side approaches 0 while the denominator approaches 1 if the computational accuracies of the used components are high enough. The upper bound $\overline{t}$ on the deflation parameter t^c can be satisfied. Algorithm 2 computes a rigorous enclosure $(\tilde{y}, \tilde{z}, \mathbf{d}^+, \mathbf{d}^-)$ of a feasible point. The lower bound f^∇ is

$$\min\{a^T\tilde{y} + b^T\tilde{z} + \sum_{i \in \underline{\mathcal{B}}} \underline{x}_i\mathbf{d}_i^+ + \sum_{i \in \overline{\mathcal{B}}} \overline{x}_i\mathbf{d}_i^-\}.$$

If the LP contains free variables, we have to compute enclosures $\mathbf{y}$ and $\mathbf{z}$ of the solution set of

$$(A_{:i})^T y + (B_{:i})^T z = c_i \quad \text{for } i \in \ddot{\mathcal{B}}.$$

But by (1.38) this system has full rank. We can compute enclosures $\mathbf{y}$ and $\mathbf{z}$ with the interval solver that contain exact solutions $\hat{y}$ and $\hat{z}$ with

$$\left\| \begin{pmatrix} \hat{y} - \tilde{y} \\ \hat{z} - \tilde{z} \end{pmatrix} \right\|_\infty \leq \delta := \kappa((A^T B^T)_{\ddot{\mathcal{B}}:}) \cdot \frac{\|\Delta c\|_\infty}{\|c\|_\infty} \cdot \left\| \begin{pmatrix} \hat{y} \\ \hat{z} \end{pmatrix} \right\|_\infty$$
$$\text{with } \Delta c := |(\varepsilon_{lp} + \varepsilon_A)c(t)| + \varepsilon_{lp}|A^T|e.$$

The radius of the computed enclosure $\begin{pmatrix} \mathring{\mathbf{y}} \\ \mathbf{z} \end{pmatrix}$ being of the order $\mathcal{O}(\varepsilon_{Int})$, the maximal distance between points in $\begin{pmatrix} \mathbf{y} \\ \mathbf{z} \end{pmatrix}$ and the approximate solution is bounded by

$$\sup \left\{ \left\| \begin{pmatrix} \mathbf{y} \\ \mathbf{z} \end{pmatrix} - \begin{pmatrix} \tilde{y} \\ \tilde{z} \end{pmatrix} \right\|_\infty \right\} \leq \delta + \mathcal{O}(\varepsilon_{Int}).$$

Consequently $\mathbf{y}$ satisfies

$$\mathbf{y} \leq \varepsilon_{lp} + \delta + \mathcal{O}(\varepsilon_{Int})$$

and

$$\begin{aligned} \mathbf{d}_i &= c_i - (A^T)_{i:}\mathbf{y} - (B^T)_{i:}\mathbf{z} \\ &= \tilde{u}_i + \tilde{v}_i - e_i(t)t_i^c \\ &\quad \pm \big((\varepsilon_{lp} + \varepsilon_A)|c(t)_i| + \varepsilon_{lp}|(A^T)_{i:}|e + \varepsilon_A|c_i - (A^T)_{i:}\tilde{y} - (B^T)_{i:}\tilde{z}| \\ &\qquad + (\delta + \mathcal{O}(\varepsilon_{Int}))(|(A^T)_{i:}|e + |(B^T)_{i:}|e) \big). \end{aligned}$$

To fulfill the sign conditions, the deflation parameter now has to satisfy

$$\begin{aligned} t_i^c \geq \frac{\varepsilon_{lp}(1 + |c(t)_i| + |(A^T)_{i:}|e) + \varepsilon_A(|c(t)_i| + |c_i - (A^T)_{i:}\tilde{y} - (B^T)_{i:}\tilde{z}|)}{1 - \varepsilon_{lp} - \varepsilon_A} \\ + \frac{(\delta + \mathcal{O}(\varepsilon_{Int}))(|(A^T)_{i:}|e + |(B^T)_{i:}|e)}{1 - \varepsilon_{lp} - \varepsilon_A}. \end{aligned} \tag{1.40}$$

As the additional parts on the right hand side, too, vanish for sufficiently large accuracies, the bound $\overline{t}$ on the deflation parameter t^c can be met. The box $(\mathbf{y}, \mathbf{z}, \mathbf{d}^+, \mathbf{d}^-)$ is a rigorous enclosure of a feasible point. Algorithm 2 returns the lower bound

$$f^\triangledown := \min\{a^T\mathbf{y} + b^T\mathbf{z} + \sum_{i \in \underline{\mathcal{B}}} \underline{x}_i \mathbf{d}_i^+ + \sum_{i \in \overline{\mathcal{B}}} \overline{x}_i \mathbf{d}_i^-\}. \qquad \blacksquare$$

Certificates of infeasibility

Certificates are sets of data that prove properties of a problem and are generally considerably easier to check than to compute. A certificate of optimality, for example, is a primal-dual pair of solutions with the same objective value. Because of duality neither the primal nor the dual can have better solutions: both are solved to optimality.

In the case of primal or dual infeasibility of an LP, certificates come with theorems of alternatives. These are pairs of systems of equations and inequalities of which exactly one has a feasible solution. Providing a solution to the alternative system of the primal or dual constraints proves respectively primal or dual infeasibility. For the standard and inequality form LP, theorems of alternatives were first described by Farkas [25]. It can be shown (see Vandenberghe and Boyd [10, Chapter 5.8]) that certificates for infeasibility and unboundedness are given by *improving rays*. An improving ray is a feasible solution of an LP with homogeneous constraints that has a negative objective value if the problem is minimized. The name comes from the ray exposing an unbounded direction in the feasible region—going in this direction cannot violate constraints—that improves the objective value.

The homogeneous dual for the LP (1.9) is

$$\begin{aligned} \max \quad & a^T y + b^T z + \underline{x}^T u + \overline{x}^T v \\ \text{subject to} \quad & A^T y + B^T z + u + v = 0 \\ & y \leq 0,\ u \geq 0,\ v \leq 0. \end{aligned}$$

Verifying a feasible point with positive objective value for this LP with Algorithm 2, duality tells us that the primal has an optimal value strictly greater 0. But since c is set to 0, the objective value of each feasible point *is equal* 0. Consequently the primal cannot have feasible points: it is infeasible. Computing a certificate for a family of LP problems **P** in this way proves the infeasibility of all problems with parameters in **P**.

It seems that another certificate would come with the Two-Phase simplex algorithm. If the phase 1 LP (1.16) has a positive optimal value, the original LP is infeasible. A dual feasible point for the phase 1 LP with an objective value strictly greater 0 thus certifies primal infeasibility, which could be verified with the rigorous lower bound. Explicitly notating the dual LP, however, shows that parts of the dual solution form a dual improving ray of the original LP. Similarly a primal improving ray can be determined with a dual phase 1 problem.

1.6 Generalizations

As already mentioned when discussing applications, linear programming is often used as a building block in other algorithms. In addition the basic ideas of the rigorous bounds can be generalized to other forms of optimization. In this section we want to examine what the previous topics look like in a more general context.

Mixed-integer linear programming

To the problem definition in linear programming, mixed-integer linear programming adds integer restrictions on some variables

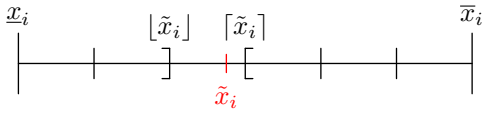
$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Ax \leq a \\ & Bx = b \\ & \underline{x} \leq x \leq \overline{x} \\ & x_{\mathcal{Z}} \in \mathbb{Z}. \end{aligned} \tag{1.41}$$

This makes the mixed-integer linear program (MILP) in general an NP-hard problem.

MILPs can be solved for example with *branch-and-bound* methods where a series of LP *relaxations* are solved to determine the optimal solution of the MILP. A relaxation in optimization is in general another optimization problem with special properties:

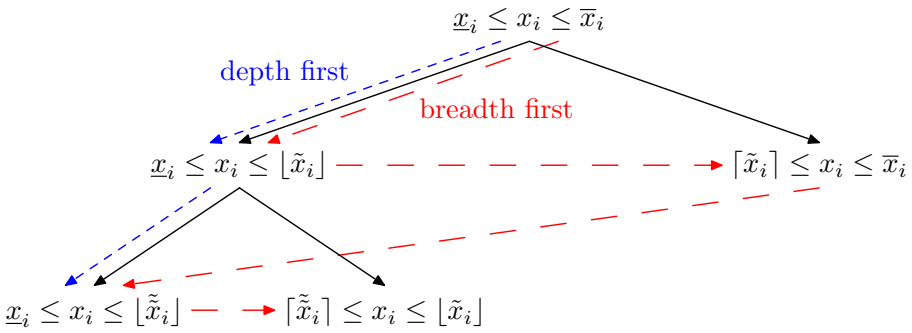
- It is easier to solve than the original problem,
- the feasible region of the relaxation contains the feasible region of the original problem,
- and the objective of the relaxation has to be less equal the original one for all feasible points of the original problem.

One relaxation of a MILP is obtained by dropping the integer constraints, resulting in an LP. With this relaxation, we can solve a MILP in the following way. All problems to be processed are stored in a queue, which at first contains only the original problem. We take the original MILP from the queue and solve the relaxation. Should the solution already satisfy the integrality conditions, we have completed our task. Otherwise an $\tilde{x}_i$, $i \in \mathcal{Z}$ is non-integral, and we add to the queue two MILPS that differ from the original one only in the simple bounds of $\tilde{x}_i$. The lower-branch MILP changes the upper bound of $\tilde{x}_i$ to $\lfloor \tilde{x}_i \rfloor$, which is the largest integer less than $\tilde{x}_i$. The upper-branch MILP sets the lower bound of $\tilde{x}_i$ to $\lceil \tilde{x}_i \rceil = \lfloor \tilde{x}_i \rfloor + 1$, the smallest integer larger than $\tilde{x}_i$. The important property here is that the union of the feasible regions excludes $\tilde{x}$ but includes all feasible points of the original problem.



Where the new MILPs are added in the queue depends on the branching strategy, which has a major influence on the performance of the algorithm. Appending to the queue results in a *breadth first* search, where all problems with the same "distance" to the original MILP are processed in a row. Adding to the head of the queue—changing the queue into a FIFO (First In First Out)

stack—, we obtain a *depth first* search. Here we follow one branch and immediately process the problems originating from the current one.



There are further parameters influencing the course of the algorithm, such as more sophisticated strategies in selecting the next subproblem and different rules in selecting the next variable to branch on. These, however, will not be discussed in this work.

Repeating this scheme until the problem queue is empty, we ultimately search the whole feasible region for integer solutions. The splitting into subproblems stops at the latest if lower and upper bound on the variables $x_{\mathcal{Z}}$ are equal. The best integer solution over all subproblems is the optimal solution of the MILP. Observing that the subproblems cannot have a better objective value than the problems they originate from, we can discard a problem if the optimal value of its relaxation is worse than the best known integer solution so far.

Of course mixed-integer linear programming suffers from rounding errors as well, and solving a MILP through a series of LPs makes it amenable to the presented rigorous bounds. This way a branch-and-bound method can be made completely rigorous as described by Neumaier and Shcherbina [92]. They present the following innocuous looking example of 20 variables

$$
\begin{aligned}
\min \quad & -x_{20} \\
\text{subject to} \quad & (s+1)x_1 - x_2 \ge s-1 \\
& -sx_{i-1} + (s+1)x_i - x_{i+1} \ge (-1)^i(s+1) \quad \text{for } i = 2,\dots,19 \\
& -sx_{18} - (3s-1)x_{19} + 3x_{20} \ge -(5s-7) \\
& 0 \le x_i \le 10 \quad \text{for } i = 1,\dots,13 \\
& 0 \le x_i \le 10^6 \quad \text{for } i = 14,\dots,20 \\
& \text{all } x \in \mathbb{Z}.
\end{aligned}
$$

With only integer variables and coefficients, the user might well expect an exact solution.

Setting $s = 6$ several state-of-the-art solver fail to find the optimal solution. Of CPLEX 8.000 [49] and the MILP solvers available in June 2002 through NEOS [14, 21, 42], namely bonsaiG [43], FortMP [96], GLPK [32], Xpress, Xpress-MP/Integer [18], and MINLP [27], only FortMP solves the problem. bonsaiG and Xpress do not find a solution. The remaining four

solvers, CPLEX, GLPK, Xpress-MP/Integer, and MINLP, even claim that no integer feasible solution exists. As we can easily check,

$$x = (1, 2, 1, 2, \ldots, 1, 2)^T$$

is a feasible solution. The solvers obviously discard this part of the feasible region due to rounding errors.

In the research paper by Doubli [23], a simple rigorous MILP solver is built using Lurupa—the software part of this thesis described in the next chapter—and the MATLAB [112] MILP solver MIQP [5]. The new solver no longer bases its decision on discarding or splitting a subproblem on approximate solutions but on the rigorous bounds computed by Lurupa. It solves the above example to optimality in less than 2 seconds on a Pentium 4 with 2.8 GHz. The optimal and only integer feasible solution is indeed $(1, 2, \ldots, 1, 2)^T$.

Looking at the intermediate results during the computation reveals the following behaviour of the approximate solver. The branch-and-bound algorithm starts with an approximate solution of the LP relaxation having $\tilde{x}_1 \gtrapprox 1$. Going depth first and always branching on the first integer variable, the first split is on x_1 with $\lceil \tilde{x}_1 \rceil = 1$, $\lfloor \tilde{x}_1 \rfloor = 2$. The original problem is transformed into the problem of finding the optimal solution with $0 \leq x_1 \leq 1$ and $2 \leq x_1 \leq 10$; the better one is the solution originally sought. The solver continues with the lower branch $0 \leq x_1 \leq 1$, and computes an approximate, again non-integer solution. Splitting yields the two problems with x_1 fixed to 0 and 1. The problem queue at this point contains the three problems

$$[(x_1 = 0),\ (x_1 = 1),\ (2 \leq x_1 \leq 10)].$$

In the parentheses, the differences to the original problem are displayed. The first problem fixes x_1 to 0, the second fixes x_1 to 1, and the third allows x_1 to vary between 2 and 10. All other constraints are unchanged. Correctly identified by the approximate solver, the first problem is infeasible and therefore discarded. The second problem is now split along the second variable since x_1 is fixed. Based on the approximate solution with $\tilde{x}_2 \gtrapprox 2$, the first split is $\lceil \tilde{x}_2 \rceil = 2$, $\lfloor \tilde{x}_2 \rfloor = 3$; the second split results in the problems with $0 \leq x_2 \leq 1$ and $x_2 = 2$, the first of which is again correctly discarded. This course repeats for the third and fourth variable, arriving at a current problem $(x_1 = 1, x_2 = 2, x_3 = 1, x_4 = 2)$ and a queue containing the problems

$$\left[\begin{pmatrix} x_1 = 1 \\ x_2 = 2 \\ x_3 = 1 \\ 3 \leq x_4 \leq 10 \end{pmatrix}, \begin{pmatrix} x_1 = 1 \\ x_2 = 2 \\ 2 \leq x_3 \leq 10 \end{pmatrix}, \begin{pmatrix} x_1 = 1 \\ 3 \leq x_2 \leq 10 \end{pmatrix}, (2 \leq x_1 \leq 10) \right].$$

At this point the rounding errors accumulate so much that the approximate solver makes wrong claims. It judges the current problem to be infeasible.

But this claim is not supported by a rigorous certificate of infeasibility, and the splitting continues; the subproblem that fixes $x_5 = 1$ is kept. Continuing until the first eleven variables are fixed to $(1, 2, 1, \ldots, 1)^T$, the approximate

solver does not find any feasible solution. Only with the fix of x_{11}, the approximate solver again diagnoses feasibility and returns an approximate solution close to the optimal one. At this point the upper bound becomes finite; feasibility is verified.

Conic programming

"In fact the great watershed in optimization isn't between linearity and nonlinearity, but convexity and nonconvexity."

—R. T. Rockafellar, 1993

As observed by Rockafellar [102] and further analyzed by Nemirovski and Yudin [85], convexity of a problem plays the crucial role in optimization.

Convex optimization problems can be specified in a number of ways. A universal formulation is Conic Programming (see Nesterov and Nemirovski [86]) in which a linear function is to be minimized subject to linear constraints on the variables x coming from a convex cone $\mathcal{C}$

$$\begin{aligned} \min \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x \in \mathcal{C}. \end{aligned}$$

In its most general appearance, x comes from a real normed vector space $\mathbb{X}$, c comes from the dual space $\mathbb{X}^*$ of linear functionals. The vector b comes from a real normed vector space $\mathbb{Y}$, and A is a continuous linear operator from $\mathbb{X}$ to $\mathbb{Y}$.

The dual conic optimization problem is

$$\begin{aligned} \max \quad & b^T y \\ \text{subject to} \quad & -A^* y + c \in \mathcal{C}^* \\ & y \in \mathbb{Y}^*. \end{aligned}$$

Here A^* denotes the *adjoint operator* of A, the cone $\mathcal{C}^*$ is the dual cone of $\mathcal{C}$ (i.e., the set of positive linear functionals on $\mathcal{C}$, which can be shown to be a convex cone itself).

Primal and dual conic program satisfy weak duality. In contrast to linear programming, feasibility, however, is not sufficient for strong duality. For the duality gap between primal and dual optimal value f_p^* and f_d^* to be zero, additional conditions called *constraint qualifications* have to be satisfied. Slater's constraint qualifications, for example, certify strong duality if strictly feasible solutions exist. This means that inequalities have to be satisfied strictly; in the case of the primal conic program, the vector x has to come from the interior of the cone $\mathcal{C}$.

This setting, however, is not sufficient to be efficiently solvable—at least not with current algorithms. For efficient algorithms to be available, additional mild computability and regularity assumptions have to be satisfied as described by Ben-Tal and Nemirovski [6]. These assumptions are

Polynomial computability The objective value and a subgradient as well as a measure of infeasibility and, if appropriate, a hyperplane separating an approximate solution from the feasible region can be computed within a polynomial number of operations on the Real Arithmetic computer. This is an idealized version of the usual computer that is capable to store countably many reals and performs the standard exact real arithmetic operations with them—the four basic arithmetic operations, evaluating elementary functions, like cos and exp, and making comparisons.

Polynomial growth The objective and the infeasibility measure grow polynomially with growing sum-norm of the approximate solution.

Polynomial boundedness of feasible sets The feasible region is contained in a Euclidean ball, centered at the origin, with a polynomially bounded radius.

Three very important classes of conic optimization problems satisfying these assumptions are linear programming, second-order cone programming, and semidefinite programming. The cones used in these problems are the positive orthant, the *second-order* cone, and the cone of symmetric, positive semidefinite matrices, respectively.

Some special applications for second-order cone programs and semidefinite programs will be mentioned later. In addition, conic-programming relaxations offer an advantage over linear-programming relaxations because they better capture nonlinearities in the model and therefore provide for tighter relaxations. In practice, however, one has to take the quality of the approximate solutions into account. As already noted, numerical experience by Tawarmalani and Sahinidis [111] suggests that linear programming is more mature in this regard.

The rigorous bounds described earlier can be extended to Conic Programming in infinite-dimensional spaces, as was done by Jansson [53]. He also gives theorems and explicit formulae for the rigorous bounds in the case of second-order cone programming and semidefinite programming, which we will now briefly consider.

The theorems require certain *boundedness qualifications* to hold, which, similar to constraint qualifications, are requirements that have to be satisfied by the solutions of the problems. The *primal boundedness qualification* (PBQ) is:

- Either the problem is primal infeasible, or
- the primal optimal value f_p is finite, and there is a simple bound $\overline{x} \in \mathcal{C}$ such that for every $\varepsilon > 0$ there exists a primal feasible solution $x(\varepsilon)$ satisfying $x(\varepsilon) \leq \overline{x}$ and $c^T x(\varepsilon) - f_p \leq \varepsilon$.

Similar the *dual boundedness qualification* (DBQ) is:

- Either the dual problem is infeasible, or
- the dual optimal value f_d is finite, and there is a simple bound $\overline{y}$ such that for every $\varepsilon > 0$ there exists a dual feasible solution $y(\varepsilon)$ satisfying $|y(\varepsilon)| \leq \overline{y}$ and $f_d - b^T y(\varepsilon) \leq \varepsilon$.

Note that PBQ and DBQ do not assume that optimal solutions exist. This is important in the infinite-dimensional case since existence of optimal solutions need not be proved.

Here the theorems are presented without boundedness qualifications similar to the linear-programming theorems described earlier. The difference between the versions with and without boundedness qualifications is that the former ones do not require feasible solutions to derive the rigorous bounds. Therefore, as Jansson remarks, the rigorous bounds are usually cheaper; rigorous bounds for ill-posed problems often can only be obtained this way. The linear-programming theorems can also be formulated with boundedness qualifications offering the same benefits for LPs. If the simple bounds in the boundedness qualifications, however, are large, it can be beneficial to sacrifice runtime and compute feasible solutions to obtain tighter rigorous bounds.

In conic programming we need a generalization of comparisons. Each convex cone $\mathcal{C}$ induces a partial ordering of elements

$$x \leq_{\mathcal{C}} y \quad \Leftrightarrow \quad y - x \in \mathcal{C}.$$

Claiming an upper bound $x \leq \overline{x}$ for a variable coming from a cone $\mathcal{C}$ has to be understood in the way $x \leq_{\mathcal{C}} \overline{x} \;\Leftrightarrow\; \overline{x} - x \in \mathcal{C}$. With this we can define lower and upper bounds of sets of vectors. A lower bound $\underline{x}$ of a set $\mathcal{X} \in \mathbb{X}$ satisfies $\underline{x} \leq_{\mathcal{C}} x$ for all $x \in \mathcal{X}$, an upper bound is defined accordingly. This also changes the meaning of $x^+ = \sup\{0, x\}$ and $x^- = \inf\{0, x\}$. These are now the respectively smallest upper and largest lower bound of $\{0, x\}$. Smallest and largest is again defined by the ordering cone.

Second-order cone programming

A Second-order Cone Program (SOCP)

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j^T x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} A_j x_j = b \\ & x_j =: (u_j, \theta_j) \in \mathcal{L}_j \qquad \text{for } j = 1, \ldots, n \end{aligned} \tag{1.42}$$

with $A_j \in \mathbb{R}^{m \times n_j}$, both c_j, and $x_j \in \mathbb{R}^{n_j}$, and $b \in \mathbb{R}^m$ derives its name from the x_j coming from second-order cones also called *ice-cream* cones or *Lorentz* cones

$$\mathcal{L} := \{(u, \theta) \mid \|u\|_2 \leq \theta\}.$$

Its dual has the following form

$$\begin{aligned} \max \quad & b^T y \\ \text{subject to} \quad & -A_j^T y + c \in \mathcal{L}_j \qquad \text{for } j = 1, \ldots, n. \end{aligned} \tag{1.43}$$

Applications of second-order cone programming as described by Lobo et al. [76] include robust linear programming, robust least-squares problems, problems involving sums or maxima of norms, and problems with hyperbolic constraints. Engineering applications are filter design, antenna array weight design, truss design, and grasping force optimization in robotics.

The rigorous lower and upper bounds for an SOCP are given by the following two theorems corresponding to Corollaries 5.1 and 5.2 from [53].

Theorem 5 (Lower SOCP bound) *Let* $\mathbf{P} = (\mathbf{A}_j, \mathbf{b}, \mathbf{c}_j)$ *define a family of second-order cone programs* (1.42) *with input data* $P \in \mathbf{P}$, *let* $\tilde{y} \in \mathbb{R}^m$, *set*

$$(\mathbf{d}_j, \delta_j) := -\mathbf{A}_j^T \tilde{y} + \mathbf{c}_j \quad \textit{for} \quad j = 1, \ldots, n,$$

and suppose that

$$(\underline{d}_j, \underline{\delta}_j) \leq (\mathbf{d}_j, \delta_j)^-.$$

Assume further that upper bounds for the primal feasible solutions of (1.42)

$$\|u_j\|_2 + \theta_j \leq \overline{\theta}_j \quad \textit{for} \quad j = 1, \ldots, n$$

are known, where $\overline{\theta}_j$ *may be infinite. If*

$$\underline{\delta}_j \geq 0 \quad \textit{for} \quad \overline{\theta}_j = +\infty,$$

then for every $P \in \mathbf{P}$ *the inequality*

$$\inf_{P \in \mathbf{P}} f_p^*(P) \quad \geq \quad f_p^\nabla := \inf\{\mathbf{b}^T \tilde{y} + \sum_{j=1}^{n} \underline{\delta}_j \overline{\theta}_j\} \tag{1.44}$$

is satisfied, and the right hand side of (1.44) *is finite.*

Theorem 6 (Upper SOCP bound) *Let* $\mathbf{P}$ *define a family of second-order cone programs* (1.42), *and suppose there exist interval vectors* $\mathbf{x}_j = (\mathbf{u}_j, \boldsymbol{\theta}_j)$ *for* $j = 1, \ldots, n$ *such that*

$$\mathbf{x}_j \subset \mathcal{L}_j$$

and

$$\forall\, b \in \mathbf{b},\ \forall\, A_j \in \mathbf{A}_j,\ j = 1, \ldots, n$$

$$\exists\, x_j \in \mathbf{x}_j : \ \sum_{j=1}^{n} A_j x_j = b.$$

Then the optimal value is bounded from above by

$$\sup_{P\in\mathbf{P}} f_p^*(P) \quad \leq \quad f_p^\triangle := \sup\{\sum_{j=1}^{n} \mathbf{c}_j^T \mathbf{x}_j\} \tag{1.45}$$

Moreover, if $\boldsymbol{\theta}_j > \mathbf{u}_j$ *and* $f_p^*(P)$ *is bounded from below,* $f_p^*(P) = f_d^*(P)$ *for every* $P \in \mathbf{P}$ *(no duality gap), and the dual supremum is attained.*

Semidefinite programming

In Semidefinite Programming one aims to solve an optimization problem that looks quite similar to an LP if written in the so-called block-diagonal form

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} \operatorname{trace}(C_j^T X_j) \\ \text{subject to} \quad & \sum_{j=1}^{n} \operatorname{trace}(A_{ij}^T X_j) = b_i \quad \text{for } i = 1,\dots,m \\ & X_j \succeq 0 \qquad\qquad\qquad\quad \text{for } j = 1,\dots,n. \end{aligned} \tag{1.46}$$

The difference to linear programming is that in an SDP, the variables X_j and parameters C_j and A_{ij} are symmetric $s_j \times s_j$ matrices; $b \in \mathbb{R}^m$. The constraints $X_j \succeq 0$ are on the eigenvalues of X, the matrices have to be positive semidefinite. If $s_j = 1$ for $j = 1,\dots,n$, all X_j, C_j, and A_{ij} are real numbers and (1.46) is a linear program in standard form.

The dual semidefinite program of (1.46) is

$$\begin{aligned} \max \quad & b^T y \\ \text{subject to} \quad & \sum_{i=1}^{m} y_i A_{ij} \preceq C_j \quad \text{for } j = 1,\dots,n. \end{aligned} \tag{1.47}$$

The constraints $\sum_{i=1}^m y_i A_{ij} \preceq C_j$ are called *linear matrix inequalities* and are satisfied if the matrix $C_j - \sum_{i=1}^m y_i A_{ij}$ is positive semidefinite.

Semidefinite programs appear in an even larger area than SOCPs since SOCPs and LPs are special cases of semidefinite programming. This includes optimal state space realizations, robust controller design, as well as eigenvalue problems in the form of minimizing the largest or the sum of the first few largest eigenvalues of a symmetric matrix X subject to linear constraints on X.

We will now look at the theorems that characterize the rigorous bounds for semidefinite programming. Details as well as proofs and numerical results can be found in Jansson [53] and Jansson, Chaykin, and Keil [54]. The numerical results were obtained with VSDP [50], a MATLAB package written by Jansson that implements the rigorous bounds for semidefinite programming. The theorems appear as given in [54], [53] contains slightly different versions using boundedness qualifications.

In addition to the tools used in the linear case, we need means to bound eigenvalues of an interval matrix. In interval arithmetic, several methods for

computing such rigorous bounds for all or some eigenvalues of interval matrices were developed. Among others these are Floudas [28], Mayer [79], Neumaier [91], and Rump [105, 106].

Theorem 7 (Lower SDP bound) *Let* $\mathbf{P} = (\mathbf{A}_{ij}, \mathbf{b}, \mathbf{C}_j)$ *define a family of semidefinite programs* (1.46) *with input data* $P \in \mathbf{P}$*, let* $\tilde{y} \in \mathbb{R}^m$*, set*

$$\mathbf{D}_j := \mathbf{C}_j - \sum_{i=1}^{m} \tilde{y}_i \mathbf{A}_{ij} \quad \textit{for} \;\; j = 1, \ldots, n,$$

and suppose that

$$\underline{d}_j \leq \lambda_{\min}(\mathbf{D}_j) \quad \textit{for} \;\; j = 1, \ldots, n.$$

Assume further that upper bounds for the maximal eigenvalues of the primal feasible solutions of (1.46)

$$\lambda_{\max}(X_j) \leq \overline{x}_j \quad \textit{for} \;\; j = 1, \ldots, n$$

are known, where $\overline{x}_j$ *may be infinite. If*

$$\underline{d}_j \geq 0 \quad \textit{for} \;\; \overline{x}_j = +\infty,$$

then for every $P \in \mathbf{P}$ *the inequality*

$$\inf_{P \in \mathbf{P}} f_p^*(P) \quad \geq \quad f_p^{\triangledown} := \inf\{\mathbf{b}^T \tilde{y} + \sum_{j=1}^{n} s_j \cdot \underline{d}_j^- \cdot \overline{x}_j\} \tag{1.48}$$

is satisfied, and the right hand side of (1.48) *is finite. Moreover, for every* $P \in \mathbf{P}$ *and every* j *with* $\underline{d}_j \geq 0$*, the linear matrix inequality*

$$\sum_{i=1}^{m} y_i A_{ij} - C_j \preceq 0$$

is feasible with $y := \tilde{y}$*.*

Theorem 8 (Upper SDP bound) *Let* $\mathbf{P}$ *define a family of semidefinite programs* (1.46)*, and suppose there exist interval matrices* $\mathbf{X}_j$ *for* $j = 1, \ldots, n$ *such that*

$$\forall\, b \in \mathbf{b},\ \forall\, A_{ij} \in \mathbf{A}_{ij},\ i = 1, \ldots, m,\ j = 1, \ldots, n$$

$$\exists\ \textit{symmetric}\ X_j \in \mathbf{X}_j : \ \sum_{j=1}^{n} \operatorname{trace}(A_{ij}^T X_j) = b_i,$$

and for $j = 1, \ldots, n$

$$X_j \succeq 0 \quad \textit{for all symmetric} \;\; X_j \in \mathbf{X}_j.$$

Then the optimal value is bounded from above by

$$\sup_{P \in \mathbf{P}} f_p^*(P) \quad \leq \quad f_p^{\triangle} := \sup\{\sum_{j=1}^{n} \operatorname{trace}(\mathbf{C}_j^T \mathbf{X}_j)\} \tag{1.49}$$

Moreover, if all symmetric $X_j \in \mathbf{X}_j$ *are positive definite and* $f_p^*(P)$ *is bounded from below,* $f_p^*(P) = f_d^*(P)$ *for every* $P \in \mathbf{P}$ *(no duality gap), and the dual supremum is attained.*

CHAPTER 2

Lurupa

Lurupa is the implementation of the algorithms for linear programming problems described in the previous chapter. It is 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 ISO C++. An earlier version of the software was already described in [62, 63].

2.1 Architecture

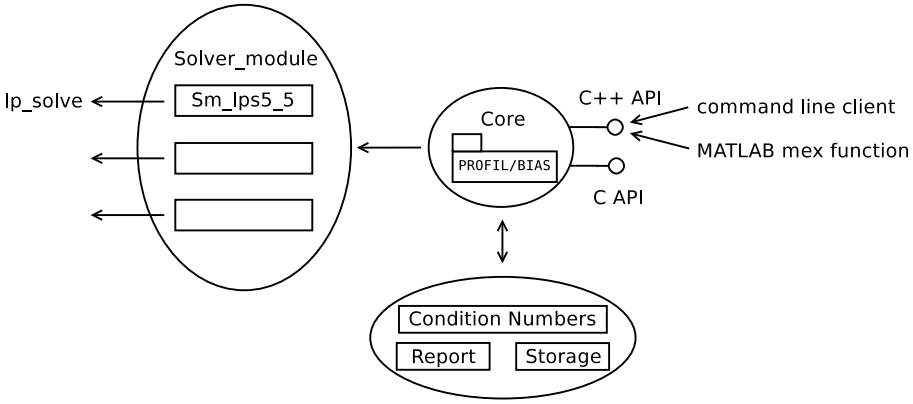
The overall architecture is depicted in Figure 2.1. The main work is performed by a computational core, which uses the PROFIL/BIAS library [70] for the rigorous computations. This core is instructed either via the command line client or using the C or C++ API (Application Programming Interface), that 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 setting up the condition number computations, reporting, and model storing. An extensive documentation of all classes and routines can be found at [61]. Here we are only interested in the main computational functions.

Computational core

Taking a tour of the essential parts and starting with the computational core, we see in Figure 2.2 a UML Class diagram of the actual worker class `Lurupa`. The main routines to use the core can be grouped according to their task.

The first two functions, `set_module` and `read_lp`, are responsible for setting up the environment. They are used to select a solver module and thus

Figure 2.1 Architecture



a linear programming solver and to read the linear program itself. To represent uncertainties in the LP, the model data can be inflated to intervals with a specified relative radius.

With `solve_lp` the solver is instructed to compute an initial approximate solution to the LP. The bound computation is performed by `lower_bound` and `upper_bound`. Certificates for primal and dual infeasibility are computed by `dual_certificate` and `primal_certificate`, respectively. Note the naming, primal infeasibility is certified by a tuple of dual variables and vice versa.

As mentioned in section 1.5, certificates are given by improving rays. Some solvers expose approximate improving rays in the case of primal or dual infeasibility. If the used solver does, Lurupa tries to verify the, possibly more than one, candidate improving rays. If the solver does not provide approximate improving rays, the phase 1 approach is tried. The phase 1 approach is also tried if the verification of approximate improving rays fails as it might still yield a verified certificate in this case.

The next three methods, `rho_p`, `rho_d`, and `cond`, can be called to obtain verified condition measures. The former two compute enclosures of the distances to primal and dual infeasibility. The final method, `cond`, itself uses `rho_p` and `rho_d` to compute a verified condition number.

To fine-tune the computations, the remaining methods may be used to change algorithm parameters `alpha`, `eta` and `inflate`. In the current implementation the deflation parameters t^a, $t^{\underline{x}}$, $t^{\overline{x}}$, and t^c of the bound computing Algorithms 2 and 1 are determined according to Jansson [52]

$$
\begin{aligned}
t^a &:= \texttt{alpha}\big(\mathring{\mathbf{a}} + (\mathring{\mathbf{A}}|\tilde{x}| + \varepsilon_{lp}(|\check{\mathbf{a}}| + |\check{\mathbf{A}}||\tilde{x}|)) + \texttt{eta}\, e \\
t_i^{\underline{x}} &:= \varepsilon_{lp}|\underline{x}_i| + \texttt{eta} \quad \text{for } \underline{x}_i \neq -\infty \\
t_i^{\overline{x}} &:= \varepsilon_{lp}|\overline{x}_i| + \texttt{eta} \quad \text{for } \overline{x}_i \neq \infty \\
t_i^c &:= \texttt{alpha}\big(\mathring{\mathbf{c}}_i + (\mathring{\mathbf{A}}^T)_{i:}|\tilde{y}| + (\mathring{\mathbf{B}}^T)_{i:}|\tilde{z}| \\
&\qquad + \varepsilon_{lp}(|\check{\mathbf{c}}_i| + |(\check{\mathbf{A}}^T)_{i:}||\tilde{y}| + |(\check{\mathbf{B}}^T)_{i:}||\tilde{z}|)) + \texttt{eta}.
\end{aligned}
$$

In contrast to the values suggested by Jansson, Lurupa sets `eta` to the maximum of 10^{-30} and $10^{-20}\|a\|_\infty$. The deflation parameters are increased when iterating in the following way

$$\begin{aligned}
&\texttt{eta} \to 10^2 \cdot \texttt{eta} \\
&t^a \to \texttt{alpha}(t^a + \texttt{eta}) \\
&t_i^{\underline{x}} \to \texttt{alpha}(t_i^{\underline{x}} + \mathring{\mathbf{x}} + \texttt{eta}) \\
&t_i^{\overline{x}} \to \texttt{alpha}(t_i^{\overline{x}} + \mathring{\mathbf{x}} + \texttt{eta}) \\
&t_i^c \to \texttt{alpha}(t_i^c + \texttt{eta}).
\end{aligned}$$

To obtain sharper bounds, this is only done for the perturbation parameters of violated constraints. If a constraint is satisfied, its deflation parameter is left at its current value. Whether a decrease of the deflation parameters is tried or not is set by `set_inflate`. If so the decrease is performed like

$$\begin{aligned}
&\texttt{eta} \to 10^{-4} \cdot \texttt{eta} \\
&t^a \to \frac{\texttt{alpha} + 1}{2 \cdot \texttt{alpha}}\, t^a \\
&t_i^{\underline{x}} \to \frac{\texttt{alpha} + 1}{2 \cdot \texttt{alpha}}\, t_i^{\underline{x}} \\
&t_i^{\overline{x}} \to \frac{\texttt{alpha} + 1}{2 \cdot \texttt{alpha}}\, t_i^{\overline{x}} \\
&t_i^c \to \frac{\texttt{alpha} + 1}{2 \cdot \texttt{alpha}}\, t_i^c.
\end{aligned}$$

This is done for all deflation parameters.

The reports can be customized via a call of `set_verbosity`, which is delegated to the `Report` class. The function 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.

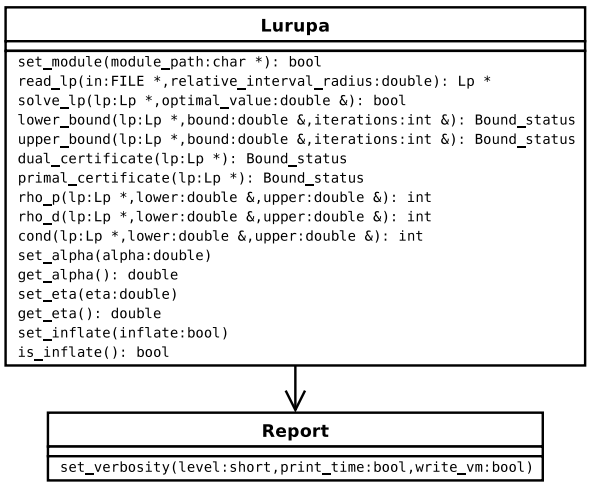
Solver modules

Looking closer at the solver modules in Figure 2.3, we find the common interface `Solver_module` with its most important methods to setup the module and an LP, solve LPs, and extract information from the solver.

A call to `get_accuracy` returns the accuracy of the approximate solver. With `set_module_options` solver-specific options can be provided in a way similar to command line arguments.

Linear programs are generated with the following four methods. 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. During the computation of primal and dual certificates, `set_primal_phase1` and `set_dual_phase1` change the LP into a phase 1 problem. The intermediate LPs needed to compute the distances to infeasibility are transformed from Lurupa's internal representation to the solver-specific one with `lp2solver`.

Figure 2.2 Computational core



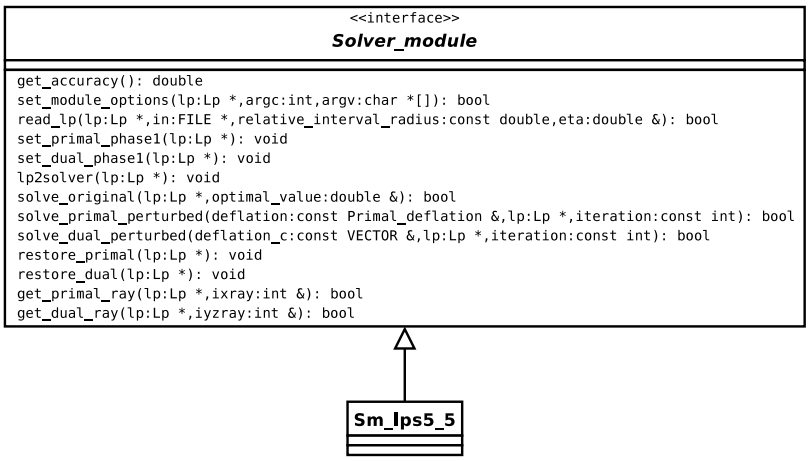
The next three methods are used to compute approximate solutions. A call to `solve_original` approximately solves the LP specified, sets the solution in the LP storage structure, and returns the optimal value in the second parameter. The methods that solve the perturbed problems, `solve_primal_perturbed` and `solve_dual_perturbed`, do not return the optimal value of the perturbed problem. Instead they take the deflation parameters to be applied as the first parameter. The third parameter `iteration` is used to generate the file name of intermediate LPs stored to disk for later examination. After computing the rigorous bounds, `restore_primal` and `restore_dual` return the LP in the approximate solver to its original, unperturbed state.

Finally `get_primal_ray` and `get_dual_ray` are used to extract candidates for approximate primal and dual improving rays from the solver. If more than one candidate is available these are distinguished with the second parameter. How this is done depends on the actual solver specific implementation.

All these methods are inherited and implemented by the solver-specific modules, depicted by the exemplary lp_solve [7] module `Sm_lps5_5`. The solver modules have to translate the above calls to corresponding calls to the solver API. 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.

One specialty about `Sm_lps5_5` is that lp_solve does not provide approximate improving rays if the LP is judged primal or dual infeasible. The approximate improving rays are extracted directly from lp_solve's internal structures. The second parameter of `get_primal_ray` and `get_dual_ray` is used as an index into this structure. The first call to these methods is done with an index value of 0. It is changed by these methods to the index of the current approximate improving ray. Supplying the new value with the next call

Figure 2.3 Solver module



guarantees that no ray is tried twice.

LP storage

The final missing piece is the `Lp` class for storing the model as seen in Figure 2.4. It stores

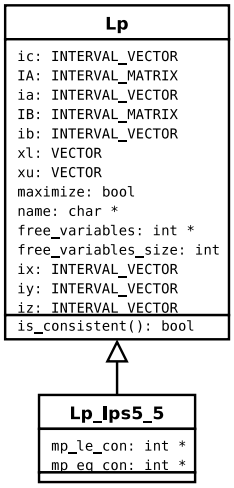
- the problem parameters $\mathbf{P} = (\mathtt{IA}, \mathtt{IB}, \mathtt{ia}, \mathtt{ib}, \mathtt{ic})$ with the simple bounds `xl`, xu and the direction of optimization,
- meta data like the name of the model and the number and indices of free variables,
- and approximate solutions and enclosures `ix`, `iy`, `iz`.

A call to `is_consistent` performs basic consistency checks such as agreement of dimensions and count and position of free variables. 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.

2.2 Usage

The usage of Lurupa depends on the actual environment and task. Its functionality can be accessed with the command line client, from MATLAB [112], or directly via the C or C++ API. The command line client and programming against the API provides the same functionality, the MATLAB interface currently offers only basic functionality.

Figure 2.4 Lp storage



Command line client

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 only mandatory general parameter is `-sm <solver_module>`, which selects the solver module to use. With `-lp <path/to/lp>`, the LP to be processed is specified. If this parameter is missing, the LP is read from the programs standard input (stdin). The important general parameters are display in Table 2.1. If neither a bound nor a certificate is selected (none of -lb, -ub, -pc, -dc), the command line client computes both bounds or the appropriate certificate if the LP is judged infeasible or unbounded by the approximate solver.

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 5.5 module are contained in Table 2.2.

A typical call with the command line client is

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

This call uses the 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', and scaling is activated.

API

The integration of Lurupa into larger frameworks is possible using the package as a library through the C or C++ interface.

Table 2.1 General command line parameters

Mandatory	
`-sm <solver_module>`	Use the solver module `solver_module` to solve the linear programs.
Optional	
`-alpha` *d*	Set algorithm parameter `alpha` to *d*.
`-cond`	Compute the condition number.
`-dc`	Compute dual certificate for primal infeasibility.
`-eta` *d*	Set algorithm parameter `eta` to *d*.
`-i` *d*	Compute bounds for an interval problem derived from the one specified. Replace all parameters by intervals with a relative radius of *d*.
`-inflate`	Try inflating the model if a perturbed one seems to be infeasible.
`-lb`	Compute the lower bound.
`-lp <path/to/lp>`	Read the LP to be processed from `<path/to/lp>`. 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.
`-pc`	Compute primal certificate for dual infeasibility.
`-rho_d`	Compute the distance to dual infeasibility.
`-rho_p`	Compute the distance to primal infeasibility.
`-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 <style>`	Write intermediate vectors and matrices to disk. If `<style>` is `octave`, the vectors and matrices are stored in one file named after the model to be solved. If `<style>` is `matlab` the vectors and matrices are stored in several files in a subdirectory named after the model to be solved.

Table 2.2 Lp_solve module command line options

`-sm,resetbas`	Try to re-solve LPs in the case of numerical failure with a basis reset to all slacks.
`-sm,s`	Use lp_solve's default scaling (Numerical range-based scaling).
`-sm,t`	Trace pivot selection
`-sm,timeout,i`	Set solver timeout to `i` seconds.
`-sm,vn`	Set solver verbosity: `v0`: NEUTRAL `v1`: CRITICAL `v2`: SEVERE `v3`: IMPORTANT (default) `v4`: NORMAL `v5`: DETAILED `v6`: FULL
`-sm,wmps,s`	Save intermediate problems in MPS format. Possible values of `s` are: `all`: Save all problems `ask`: Ask before saving `none`: Save no problems

Listing 1 API usage

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

FILE *in = fopen("lp.mps", "r");
Lp *lp = l.read_lp(in, 0);
char *options[] = {"-sm,s"};
l.set_module_options(lp, 1, options);

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

Lurupa exposes its functionality through the computational core `Lurupa`. Looking back at Figure 2.2, the previous command line example 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 values of `literations` and `uiterations` equal the number of necessary algorithm iterations.

Listing 2 MATLAB usage

```
lp.c = [-1; -1];
lp.A = [2 -1; -3 -3]; lp.a = [1; -1];
lp.B = [1 -2]; lp.b = 0;
lp.xl = [0; 0]; lp.xu = [1; 1];

lbound = lurupa('lower_bound', lp);
[ubound, flag, xInf, xSup, f, x] = ...
lurupa('upper_bound', lp);

pinf = lurupa('dual_certificate', lp);
[dinf, xcInf, xcSup] = ...
lurupa('primal_certificate', lp);
```

MATLAB interface

The MATLAB interface was implemented to support the research paper by Doubli (see page 28 in subsection Mixed-integer linear programming). Therefore it currently supports the computation of lower and upper bound and certificates only. It accepts linear programs defined in a MATLAB structure containing the LP parameters in the order c, A, a, B, b, $\underline{x}$, $\overline{x}$. All vectors have to be column vectors.

Accessing the functionality is done via the interface function `lurupa` whose first parameter is either `'lower_bound'` or `'upper_bound'` selecting the computation to be performed. The LP is specified as the second parameter. Depending on the number of output arguments, `lurupa` just returns the rigorous bound, an additional status flag and the primal or dual enclosure, or also the approximate optimal value and solution. A simple example is shown in Listing 2.

At the end `lbound` contains the rigorous lower bound. The upper bound is contained in `ubound`, `flag` is the status of the upper bound computation (i.e., success or why the bound computation failed). The primal enclosure $[$`xInf`, `xSup`$]$ is guaranteed to contain a primal feasible point. Approximate optimal value and solution are returned in `f` and `x`. Whether the LP is primal or dual infeasible is stored in `pinf` or `dinf`. If the problem is dual infeasible, a primal improving ray is contained in the box $[$`xcInf`, `xcSup`$]$.

2.3 Implementing new solver modules

To use Lurupa with a new approximate linear programming solver, an appropriate solver module has to be implemented. The main functionality of the solver module is to translate between the LP representation of Lurupa and the approximate solver, extract information about the model from the solver, and perturb the LP in the solver. Figure 2.3 shows the maximal interface that has to be supported by the solver module.

Including all supported solvers in the executable or library version of Lurupa would increase the size considerably. Also name conflicts would be very likely between the variables and functions in different solver modules included in this way. As the purpose of the approximate solvers is the same, the probability of two developers choosing the same name in two different solvers is high. This naturally occurs for different versions of the same approximate solver. Therefore it was decided to include the individual solver module dynamically at runtime via the C dlopen API. This functional layer, however, does not support the object-oriented features of C++. This is worked around by a C function present in every solver module that is accessible via the dlopen API. This function, called `get_func_pointers`, is called when loading a solver module, and it populates a structure of function pointers to the appropriate functions of the solver module. An additional benefit of this approach is that the solver modules can support a subset of the functionality by setting some of these function pointers to C null pointers. The computational core can inspect these pointers and enable or disable the appropriate functionality at runtime. In the current version the computation of rigorous certificates and distances to infeasibility and condition numbers are optional.

A minimal solver module has to support the following main functions

- `get_accuracy`,
- `set_module_options`,
- `read_lp`,
- `solve_original`,
- `solve_primal_perturbed`,
- `solve_dual_perturbed`,
- `restore_primal`,
- `restore_dual`.

The complete set of functions together with descriptions can be found in the online documentation [61].

CHAPTER 3

Numerical Experience

In this chapter we want to examine the numerical properties of Lurupa. In the first part we will investigate the performance of Lurupa on the netlib collection of linear programming problems. We will observe the connection between rigorous bounds and the distances to infeasibility. To judge the quality of the bounds for interval data, the algorithms will be applied to the netlib problems with added uncertainty.

In the second part of this chapter we want to look at how Lurupa compares to other software packages capable of computing rigorous results for an LP. To this end, eight packages using different techniques will be applied to a test set of over 150 problems.

The results of these experiments were published in [64, 65].

3.1 Tests with the netlib test set

The netlib collection of linear programming test problems [87], as already noted, includes many real world applications that appeared interesting. This has to be born in mind when considering the results of Ordóñez and Freund [97]. Applying their translation of Renegar's condition number to this test set, they revealed that 71% of these LPs are ill-posed. Using preprocessing techniques, 19% of the problems retain this property. Fourer and Gay [29], however, observed that rounding errors during preprocessing may turn a feasible LP infeasible and vice versa.

In the following we have a look at rigorous results for the netlib suite of linear programming problems. The linear programming solver used to compute approximate solutions is lp_solve 5.5 [7]. Compiler suite is the gcc version 3.3.1 [31]. All computations are performed on a PC with 2.8 GHz.

In order to compare the results, we choose exactly the set of problems that Ordóñez and Freund have computed condition numbers for. For the problems *degen3* and *pilot*, lp_solve has to be aborted because the original problem is not solved after 24 hours, leaving 87 problems in the test set.

The result are displayed in Tables A.1 to A.5 in the appendix. Summarizing, it can be seen that in almost all cases rigorous upper bounds $f^{\triangle}$ and rigorous lower bounds $f^{\triangledown}$ are computed if the distance to respectively primal infeasibility ρ_p and dual infeasibility ρ_d is greater than 0. Rigorous bounds and a certificate of the existence of optimal solutions are obtained for well-posed problems. The computational costs increase with decreasing distance to primal or dual infeasibility, that is with decreasing distance to ill-posedness.

Table A.1 shows the accuracy of the rigorous bounds. The first column contains the name of the problem. Columns two and three contain the distances to infeasibility ρ_d and ρ_p as computed by Ordóñez and Freund. Then the lower and upper bound rounded to 5 decimal digits of precision are displayed. The last column contains the relative error

$$\mu := \frac{|f^{\triangle} - f^{\triangledown}|}{\max\{1,\, 0.5(|f^{\triangle}| + |f^{\triangledown}|)\}}.$$

If one of the bounds in the quotient μ can not be computed, it is substituted by the approximate optimal value delivered by lp_solve. In the case of both bounds being infinite, μ is set to NaN (i.e., Not a Number).

Throughout our experiments we use lp_solve 5.5 with the default optimization parameters. From Table A.1, it can be seen that for almost all problems the relative error μ varies between $1 \cdot 10^{-8}$ and $1 \cdot 10^{-16}$. With lp_solve's default stopping tolerance of $1 \cdot 10^{-9}$, this is about the best one could expect.

We see that in almost all cases the rigorous lower and upper bound is finite if the distance to dual and primal infeasibility is greater than 0, respectively. Only the problems *scsd8* and *sctap1* deliver no upper bound despite a primal distance to infeasibility greater than 0. On the other hand, the problems *25fv47, 80bau3b, beaconfd, bnl2, cycle, d2q06c, e226, recipe, scrs8, standgub* deliver a lower bound and *80bau3b, adlittle, e226, finnis, gfrd-pnc, sc105, sc205, sc50a, sc50b* an upper one although the corresponding distance to infeasibility is equal to 0. Some of these problems might in fact have a distance to infeasibility that is greater than 0 as Ordóñez and Freund computed the distances numerically without verification. On the other hand, as the closer investigation of *adlittle* showed, finite rigorous bounds can be computed for ill-posed problems if the verification process does not introduce overestimation in critical components (see the remark after Theorem 3).

The large relative errors μ for the problems *scsd6, sctap2, sctap3* are due to the bad upper bounds. Whether the bad upper bounds are caused by problem properties, by approximate computations, or overestimation introduced in the verification process has to be answered by future research.

Table A.2 shows that although almost all problems in this collection have unbounded variables, in many cases the rigorous lower bound can be computed within a fraction of the computational work that is required for solving the problem approximately. There $i_{f^{\triangledown}}$, $i_{f^{\triangle}}$ denote the number of iterations for computing the lower and upper bound respectively and $t_{f^{\triangledown}}/t_{f^*}$, $t_{f^{\triangle}}/t_{f^*}$ denote the corresponding time ratios. If this ratio can not be computed due to problems being solved faster than the timer resolution of 0.01 sec, this column

is left empty. Only the problems *fit1d, fit2d, sierra* have finite simple bounds yielding an infinite distance to dual infeasibility. This results in a lower bound without the need of iterating. The problem *recipe* also delivers a lower bound without any iterations, and 65 problems deliver a lower bound in only 1 iteration. The huge time ratios for problems *agg*, *beaconfd*, and *scrs8* stem from lp_solve being aborted after 24 hours of trying to solve a perturbed problem.

Infinite error bounds for the optimal value are a result of ill-posed problems expressed by exceeding iteration counts, rank deficient constraint matrices, or in five cases, by numerical problems during the solution of perturbed linear programs. Table A.2 shows that the determination of an infinite bound is very time consuming if the iteration count exceeds the set limit of 31.

Since PROFIL/BIAS does not support sparse structures, the memory usage increases dramatically during the necessary transformation of the constraint matrices from lp_solve's sparse representation to PROFIL/BIAS's non-sparse one. The usage of non-sparse interval linear solvers adds to this effect. This is the reason why in some cases even few iterations result in large time ratios.

The bounds depend drastically on the quality of the approximate solutions. Even if we use the same solver but a previous version (lp_solve 3.2) the results get worse. Some rigorous lower bounds computed with lp_solve 3.2 are listed in Table A.3. In all cases the approximations computed by lp_solve are within the rigorous bounds.

To test the quality of the algorithms when using interval data, we multiply the problem parameters of the Netlib LPs by the interval $[1 - 10^{-6}, 1 + 10^{-6}]$ yielding a relative uncertainty. Table A.4 contains the error bounds for these problems with interval parameters, and Table A.5 shows the performance. Compared with the radius $r = 1 \cdot 10^{-6}$ of the interval input data, the algorithms give in most cases very accurate worst case bounds. The huge time ratios for *agg*, *beconfd*, *fffff800* and *pilot.we* again originate from lp_solve failing the 24 hour threshold while trying to solve perturbed problems.

3.2 Comparison with other software packages

Similar to the recent comparison of several state-of-the-art complete global optimization solvers by Neumaier et al. [93], we now want to compare Lurupa with different optimization packages. In this comparison, however, we only look for rigorous results taking all rounding errors into account.

Other software packages for verified computations in linear programming

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 algo-

rithms specifically designed to rigorously solve LPs in the presence of rounding errors and uncertainties in the input data.

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 [91], and Vu et al. [119, 120]).

An implementation of a verified constraint programming algorithm is RealPaver [40]. It uses a branch-and-prune algorithm, the pruning step merges constraint satisfaction techniques with the interval Newton method.

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 [35] for problems where either the number of variables or constraints does not go well beyond 30. Dhiflaoui et al. [20] 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 [72] 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. [3], who iteratively increase the precision of the approximate computations until the computed basis proves to be optimal.

A fully rational simplex method is exlp [68] by Kiyomi. The implementation by Koch, perPlex [71], 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 [2].

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 [8], GlobSol [58], ICOS [74], and Numerica [116]. 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 [52] and Neumaier and Shcherbina [92] (see below). We will see later how this contributes to the results of ICOS.

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. This is the category Lurupa belongs to.

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.

Besides computing rigorous bounds for the optimal value, enclosures of near-optimal solutions for the primal and the dual problem, and verified certificates of infeasibility and unboundedness, Lurupa is the only package in this comparison that computes verified condition numbers as defined by Renegar [101]. This is especially interesting when solving real-world applications as it allows to evaluate the underlying models.

It is interesting to note that few of these algorithms are specially devised for convex programming problems. This despite Ben-Tal and Nemirovski [6] and Vandenberghe and Boyd [117] observing that a large number of applica-

tions can be modelled as convex problems, and Rockafellar [102] emphasizing the importance of convexity, as already mentioned. Only Lurupa, VSDP [50]—the MATLAB toolbox 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.

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, however, 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 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 to consider is the fairness in comparing general purpose algorithms with algorithms specifically targeted at linear programming. In the first place, all these packages are able to solve LPs so we will have a look at how they perform. Secondly 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 finally, 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*. Therefore linear programming is also a good benchmark for nonlinear optimization packages.

Test environment

We will use three test sets to compare the software packages. The first one consists of random problems generated with Algorithm 3, which is similar to the

Algorithm 3 Random problems after Rosen and Suzuki

Given: m, p, n with $p \leq n \leq m + p$

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
   $$c = A^T y^* + B^T z^*$$

one suggested by Rosen and Suzuki [103]. All random choices are uniformly distributed and integral.

Using this procedure the solution and the 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.* This is often a result of inherent model properties as investigated by Greenberg [41]. The problems generated with this procedure are dense; 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.

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 both, the netlib collection and the *Misc* section of Meszaros collection [81] [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 not to 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 contained in Meszaros *infeas* section with less than 1500 variables[2].

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

- 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 [31]. 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.

[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.

The numerical results of Numerica will not be displayed here because it was distributed in a way that it only runs on the computer it was purchased for, 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 fail the timeout threshold. 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 [59]. 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}).

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

Numerical experience

Summary

The main result of this comparison can be seen in the summary in Table 3.1. 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 that 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

Table 3.1 Number of problems solved per package

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

perPlex is with respect to the feasible problems. We clearly see

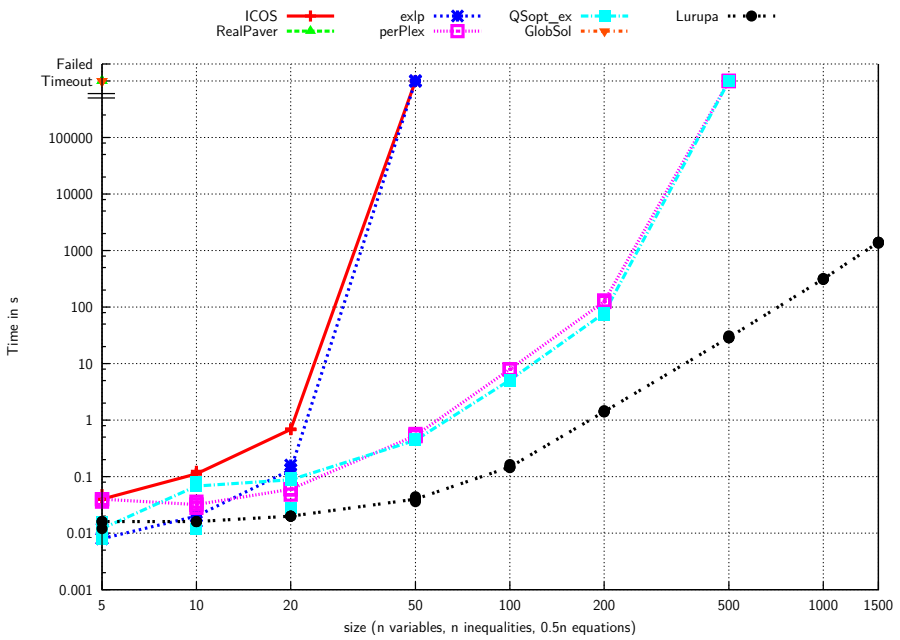
- RealPaver is not suitable for the feasible problems. It is naturally better at identifying the infeasible ones.
- The rational algorithms, exlp, perPlex, and QSopt_ex, solve almost all of the feasible problems. They run out of time for the larger random problems. Only QSopt_ex verifies the infeasibility 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 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 still 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.

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 3.1. 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 to do so for all three problems of

Figure 3.1 Running times for random problems



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 [60]. 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.

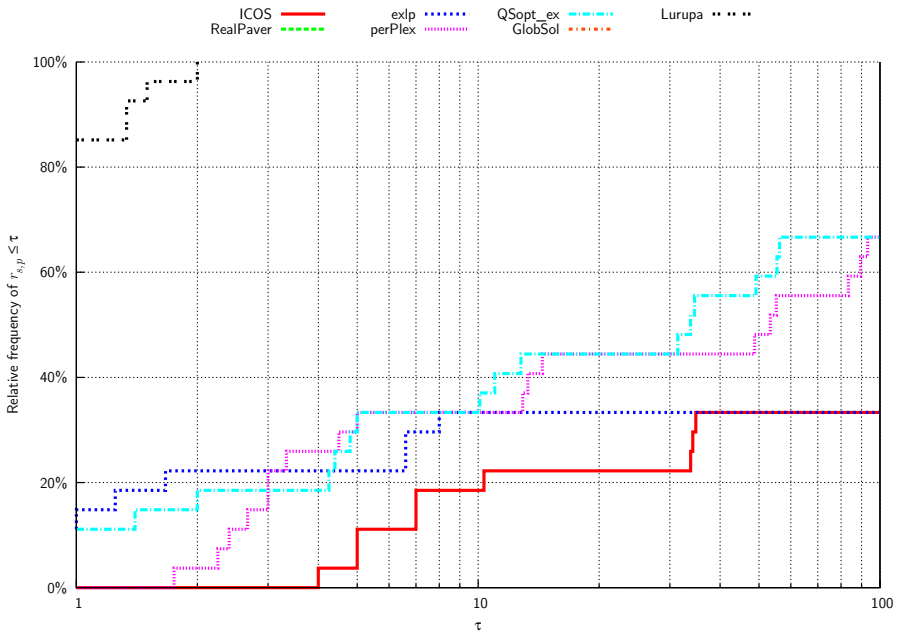
Lurupa proves to be the fastest solver for this test set. For size 50 it is already an order of magnitude faster than all others, and this factor increases to almost two orders of magnitude for size 200. Lurupa is the only package solving problems of size greater than 500.

Figure 3.2 is a performance profile of the random problem runtimes as introduced by Dolan and Moré [22]. 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

Figure 3.2 Performance profile on random problems



for each solver

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

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 coordinate system, the higher percentage of problems are solved with small ratios $r_{s,p}$.

The performance profile supports the previous observations. We can clearly recognize three distinct 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 3.3 and 3.4 display their running times and variable counts. As the size of these problems is a weak indicator of their difficulty, the problems are sorted by increasing runtime for Lurupa instead of by size as in Figure 3.1.

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

Figure 3.3 Running times for feasible real-world problems 0–59

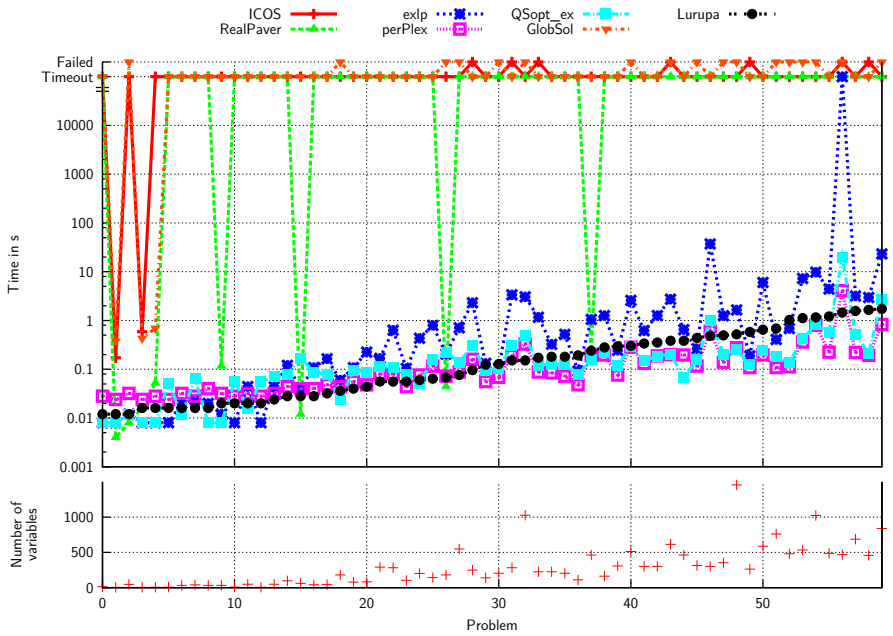


Figure 3.4 Running times for feasible real-world problems 60–101

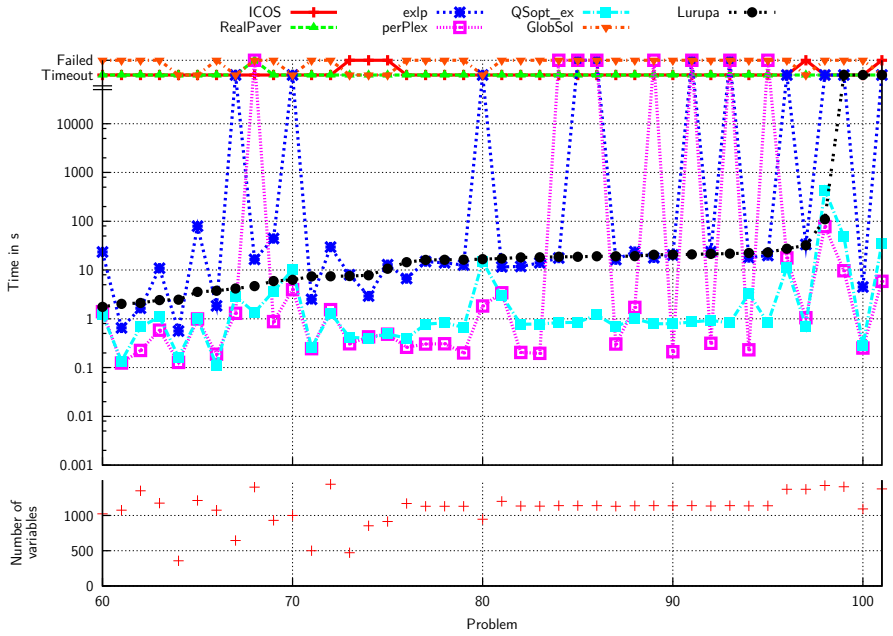
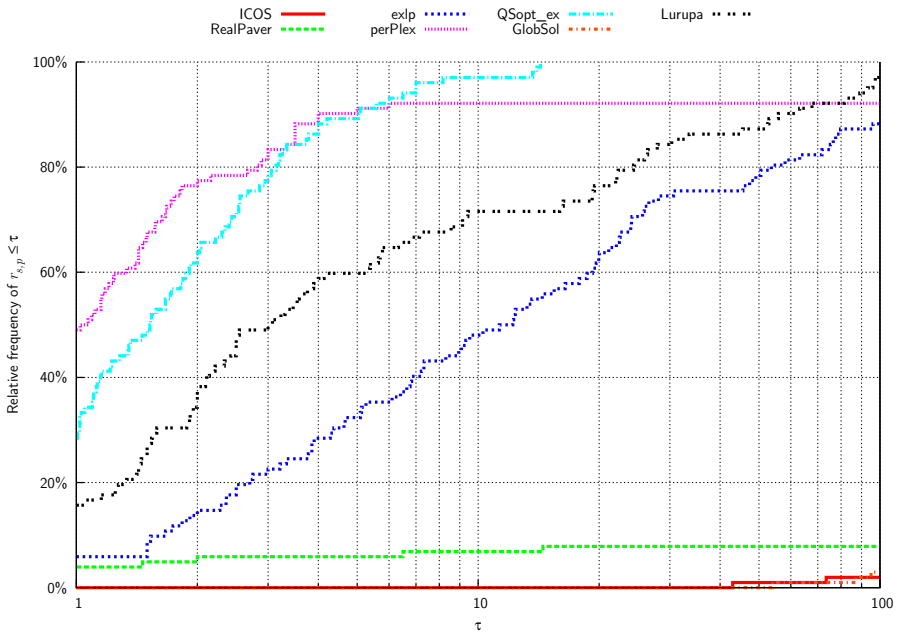


Figure 3.5 Performance profile on feasible real-world problems



exlp, perPlex, QSopt_ex, and Lurupa exhibit similar increases in runtime while the differences between them grow 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.

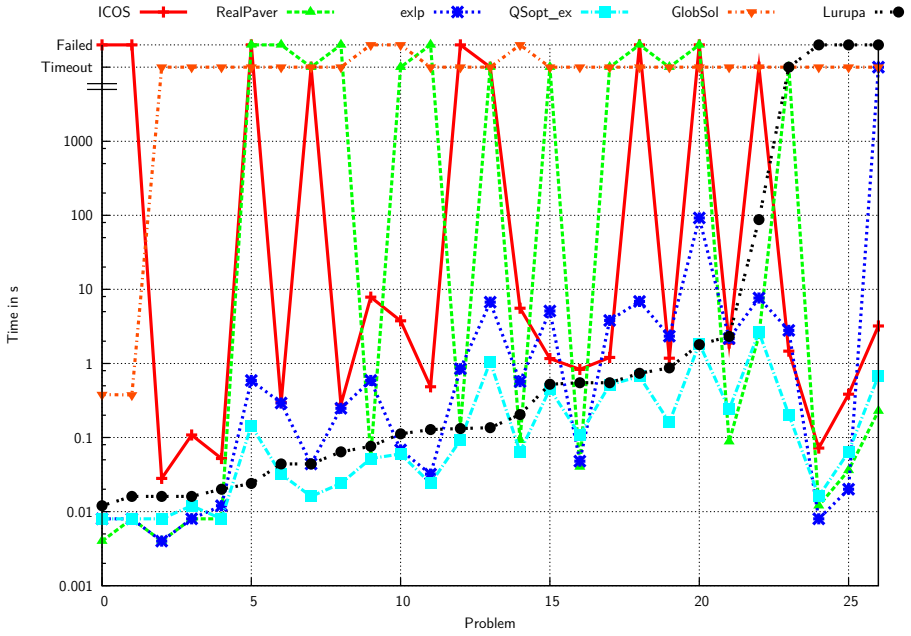
The performance profile on this set of problems in Figure 3.5 reveals the two groups, on the one hand perPlex, QSopt_ex, exlp, and Lurupa and on the other hand RealPaver, ICOS, and GlobSol. With runtime ratios less than 6, perPlex solves the highest percentage of problems. Allowing larger ratios QSopt_ex takes the lead and solves all problems within a factor of 11. Approaching τ values of 100, Lurupa surpasses 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 3.6. 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

Figure 3.6 Running times for infeasible real-world problems



tends to stay QSopt_ex, Lurupa, exlp. Only QSopt_ex verifies the infeasibility of all test problems.

In Figure 3.7 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.

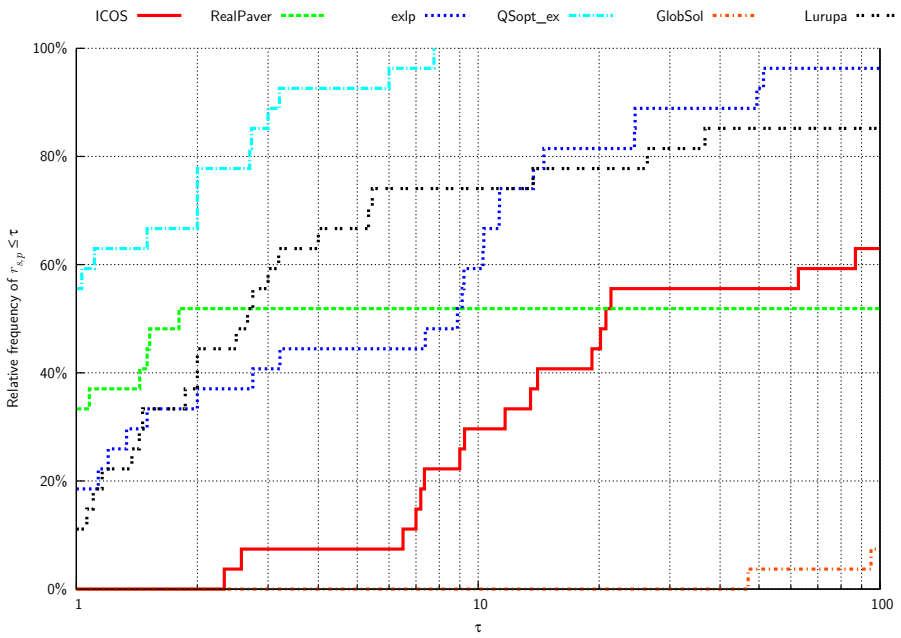
Errors and limitations

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

Figure 3.7 Performance profile on infeasible real-world problems



[123], QSopt_ex, and CPLEX 9.0.0 [49]) 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.

CHAPTER 4

Conclusion

Linear programming was, is, and will be an important tool in solving problems not only in operations research, engineering, and science. Like all computations performed in limited precision, linear programming suffers from rounding errors. Against intuition, frequently LPs are quite sensitive to these errors, which go unnoticed with traditional floating-point algorithms.

Verification methods, for example using interval arithmetic, provide rigorosity in the computed results, and they facilitate judging the problem formulation and sensitivity. The algorithms analyzed and implemented in this work offer all this at a reasonable cost. Computing rigorous error bounds for the optimal value of linear programming problems together with certificates of the existence of optimal solutions, infeasibility, or unboundedness has been shown to be possible in various areas of applications.

These bounds can also be used in global optimization and mixed-integer nonlinear programming whenever linear relaxations have to be solved in a branch-and-bound algorithm.

Comparing the implementation, Lurupa, with other verification packages for linear programming revealed that even for the rather small LPs of the test set, 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 at fast and rigorous results.

Lurupa's combination of 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 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 the user 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.

CHAPTER 5

Outlook

Several areas seem promising for further research and improvement of the software.

The most important improvement seems to be to extend PROFIL/BIAS with sparse structures. This is currently the limiting factor when going to higher dimensions. Storing interval representations of the whole constraint matrices of *stocfor3*, one of the largest netlib problems with 16675 constraints and 15695 variables, requires more than 4 GByte of memory. With less than 0.25 permille nonzero entries, this storage reduces to 1 MByte if stored in a sparse way. Adding to this, sparse structures not only save storage but also computations if the problems have sparse constraints. This is the type of problems where Lurupa is slower than the sparse rational solvers. Using sparse storage for the constraint matrices also requires changes in the interval solver as this currently uses approximate inverses, which are generally full. Verified sparse solvers are readily available for symmetric positive definite matrices, for more general matrices, algorithms are more involved [105]. The requirement of being symmetric positive definite can be met by using normal equations. Based on an approximate solution, we can enclose the normwise nearest true solution in this way because the normal equations yield the normwise minimal solution. The implementation of sparse structures is work in progress.

As already revealed in the numerical results by switching from one version of lp_solve to another, the approximate solver has a large influence on the quality of the enclosures. Implementing additional solver modules and comparing the obtained results would make it possible to quantify this influence and broaden the applicability of Lurupa. A cooperation with the developers of lp_solve has been initiated to incorporate Lurupa into a future release of lp_solve to obtain a powerful MILP solver with optional verification.

The results of Ordóñez and Freund [97] show that preprocessing considerably improves the condition number of the netlib LPs. Nevertheless Fourer and Gay [29] observed that preprocessing can change the state of an LP from

feasible to infeasible and vice versa. Therefore a verified preprocessing as described by Fourer and Gay would be valuable in computing bounds for ill-conditioned problems.

Improvements of the implementation could be possible in the following areas.

Several strategies were explored to select the variables to be fixed when computing an enclosure of a solution of an interval linear system of equations. The current implementation tries to heuristically keep the condition number of the quadratic part low by selecting the variables corresponding to the pivot elements of an LU decomposition with column pivoting of the linear system (see Jansson [52]). As already mentioned, switching to sparse structures and using normal equations would automatically try to enclose the nearest true solution to the approximate one. Perhaps further strategies for selecting the variables or computing an enclosure can be found that take into account the structure of the problem and the distance of the approximate solution to the inequalities and simple bounds.

In the current implementation, the computation of the deflation parameters and their change during the course of the algorithm as well as the perturbation scheme are empirical choices. A systematic investigation of these could reveal better choices resulting in sharper bounds with less computations.

As exposed by the numerical results for the netlib LPs *scsd6*, *sctap2*, and *sctap3*, there are still questions to the behavior of the bound computing algorithms for some problems. Investigations could result in better bounds for these specific problems as well as for a whole class of problems. They might also reveal useful theoretical insight to further improve the algorithms.

Specifying problems with uncertainty is currently only possible via the API by explicitly giving the uncertainty of each parameter in the program; no input format exists for these kind of problems. A specification of an input format supporting uncertainty would ease the specification and the computation of rigorous bounds for these kind of problems.

The MATLAB interface currently supports only the subset of Lurupa's functionality that was needed to build a rigorous MILP solver. A future version of the interface should offer the whole functionality to widen the applicabilty also from MATLAB.

APPENDIX A

Tables

Table A.1: Rigorous bounds for the Netlib problems
ρ_d – distance to dual infeasibility, ρ_p – distance to primal infeasibility,
$f^{\triangledown}$ – lower bound, $f^{\triangle}$ – upper bound, μ – relative accuracy

Name	ρ_d	ρ_p	$f^{\triangledown}$	$f^{\triangle}$	μ
25FV47	0	0	$5.5018e+03$	∞	$8.5111e-08$
80BAU3B	0	0	$9.8722e+05$	$9.8722e+05$	$5.6653e-08$
ADLITTLE	0.051651	0	$2.2549e+05$	$2.2549e+05$	$3.6470e-08$
AFIRO	1.000000	0.397390	$-4.6475e+02$	$-4.6475e+02$	$2.0481e-08$
AGG2	0.771400	0	$-2.0239e+07$	∞	$2.0868e-08$
AGG3	0.771400	0	$1.0312e+07$	∞	$7.3998e-08$
AGG	0.771400	0	$-3.5992e+07$	∞	$2.7323e-08$
BANDM	0.000418	0	$-1.5863e+02$	∞	$7.0742e-08$
BEACONFD	0	0	$3.3592e+04$	∞	$9.9997e-09$
BLEND	0.040726	0.003541	$-3.0812e+01$	$-3.0812e+01$	$1.3560e-07$
BNL1	0.106400	0	$1.9776e+03$	∞	$7.2244e-08$
BNL2	0	0	$1.8112e+03$	∞	$2.0899e-08$
BORE3D	0.003539	0	$1.3731e+03$	∞	$1.3362e-08$
BRANDY	0	0	$-\infty$	∞	NaN
CAPRI	0.095510	0.000252	$2.6900e+03$	$2.6900e+03$	$1.6905e-07$
CYCLE	0	0	$-5.2264e+00$	∞	$1.4574e-08$
CZPROB	0.008807	0	$2.1852e+06$	∞	$1.0915e-08$
D2Q06C	0	0	$1.2278e+05$	∞	$4.5242e-08$
D6CUBE	2.000000	0	$3.1549e+02$	∞	$1.6796e-08$
DEGEN2	1.000000	0	$-1.4352e+03$	∞	$9.3150e-09$
E226	0	0	$-2.5865e+01$	$-2.5865e+01$	$9.1411e-08$
ETAMACRO	0.200000	0	$-7.5572e+02$	∞	$4.4004e-09$
FFFFF800	0.033046	0	$5.5568e+05$	∞	$4.1052e-08$
FINNIS	0	0	$-\infty$	$1.7279e+05$	$4.8378e-08$
FIT1D	∞	3.500000	$-9.1464e+03$	$-9.1464e+03$	$6.1900e-09$
FIT1P	0.437500	1.271887	$9.1464e+03$	$9.1464e+03$	$1.1418e-08$
FIT2D	∞	317.000000	$-6.8464e+04$	$-6.8464e+04$	$4.8472e-09$
FIT2P	1.000000	1.057333	$6.8464e+04$	$6.8464e+04$	$6.9205e-09$
GANGES	1.000000	0	$-1.0959e+05$	∞	$3.5123e-09$
GFRD-PNC	0.347032	0	$6.9022e+06$	$6.9022e+06$	$5.5746e-08$
GREENBEA	0	0	$-\infty$	∞	NaN

continued...

Name	ρ_d	ρ_p	$f^{\triangledown}$	$f^{\triangle}$	μ
GREENBEB	0	0	$-\infty$	∞	NaN
GROW15	0.968073	0.572842	$-1.0687e+08$	$-1.0687e+08$	$3.5135e-09$
GROW22	0.968073	0.572842	$-1.6083e+08$	$-1.6083e+08$	$3.7475e-09$
GROW7	0.968073	0.572842	$-4.7788e+07$	$-4.7788e+07$	$3.6032e-09$
ISRAEL	0.166850	0.027248	$-8.9664e+05$	$-8.9664e+05$	$1.5935e-08$
KB2	0.018802	0.000201	$-1.7499e+03$	$-1.7499e+03$	$2.1792e-08$
LOTFI	0	0.000306	$-\infty$	$-2.5265e+01$	$4.5049e-09$
MAROS	0	0	$-\infty$	∞	NaN
MAROS-R7	0.628096	1.000000	$1.4972e+06$	$1.4972e+06$	$8.5236e-09$
MODSZK1	0.108469	0	$3.2057e+02$	∞	$1.5512e-04$
PEROLD	0.000943	0	$-9.3808e+03$	∞	$2.2012e-08$
PILOT4	0.000075	0	$-2.5811e+03$	∞	$2.8098e-08$
PILOT87	0	0	$-\infty$	∞	NaN
PILOT.JA	0.000750	0	$-6.1131e+03$	∞	$1.5904e-08$
PILOTNOV	0.000750	0	$-4.4973e+03$	∞	$3.2619e-08$
PILOT.WE	0.044874	0	$-2.7201e+06$	∞	$5.2748e-08$
QAP8	4.000000	0	$2.0350e+02$	∞	$5.2913e-08$
RECIPE	0	0	$-2.6662e+02$	∞	$4.2641e-16$
SC105	0.133484	0	$-5.2202e+01$	$-5.2202e+01$	$7.7626e-08$
SC205	0.010023	0	$-5.2202e+01$	$-5.2202e+01$	$9.0740e-08$
SC50A	0.562500	0	$-6.4575e+01$	$-6.4575e+01$	$5.6764e-08$
SC50B	0.421875	0	$-7.0000e+01$	$-7.0000e+01$	$5.7599e-08$
SCAGR25	0.034646	0.021077	$-1.4753e+07$	$-1.4753e+07$	$3.7821e-08$
SCAGR7	0.034646	0.022644	$-2.3314e+06$	$-2.3314e+06$	$3.9152e-08$
SCFXM1	0	0	$-\infty$	∞	NaN
SCFXM2	0	0	$-\infty$	∞	NaN
SCFXM3	0	0	$-\infty$	∞	NaN
SCORPION	0.949393	0	$1.8781e+03$	∞	$2.7948e-08$
SCRS8	0	0	$9.0430e+02$	∞	$3.4248e-08$
SCSD1	1.000000	5.037757	$8.6667e+00$	$8.6668e+00$	$1.0579e-05$
SCSD6	1.000000	1.603351	$5.0500e+01$	$5.0707e+01$	$4.0917e-03$
SCSD8	1.000000	0.268363	$9.0500e+02$	∞	$6.3831e-08$
SCTAP1	1.000000	0.032258	$1.4122e+03$	∞	$2.1640e-08$
SCTAP2	1.000000	0.586563	$1.7248e+03$	$1.9777e+03$	$1.3662e-01$
SCTAP3	1.000000	0.381250	$1.4240e+03$	$2.0866e+03$	$3.7748e-01$
SHARE1B	0.000751	0.000015	$-7.6589e+04$	$-7.6589e+04$	$1.7119e-07$
SHARE2B	0.287893	0.001747	$-4.1573e+02$	$-4.1573e+02$	$4.0674e-07$
SHELL	1.777778	0	$1.2088e+09$	∞	$4.6203e-09$
SHIP04L	13.146000	0	$1.7933e+06$	∞	$9.7665e-09$
SHIP04S	13.146000	0	$1.7987e+06$	∞	$1.0115e-08$
SHIP08L	21.210000	0	$1.9091e+06$	∞	$1.0593e-08$
SHIP08S	21.210000	0	$1.9201e+06$	∞	$1.1197e-08$
SHIP12L	7.434000	0	$1.4702e+06$	∞	$1.1950e-08$
SHIP12S	7.434000	0	$1.4892e+06$	∞	$1.3700e-08$
SIERRA	∞	0	$1.5394e+07$	∞	$5.3601e-14$
STAIR	0	0.000580	$-\infty$	$-2.5127e+02$	$5.4796e-09$
STANDATA	1.000000	0	$1.2577e+03$	∞	$1.2619e-08$
STANDGUB	0	0	$1.2577e+03$	∞	$1.2619e-08$
STANDMPS	1.000000	0	$1.4060e+03$	∞	$1.3776e-08$
STOCFOR1	0.011936	0.001203	$-4.1132e+04$	$-4.1132e+04$	$4.2148e-08$
STOCFOR2	0.000064	0.000437	$-3.9024e+04$	$-3.9024e+04$	$5.6996e-08$
TRUSS	10.000000	0.518928	$4.5882e+05$	$4.5882e+05$	$2.3769e-06$
TUFF	0.017485	0	$2.8677e-01$	∞	$5.3744e-03$
VTP.BASE	0.500000	0	$1.2983e+05$	∞	$3.4508e-08$
WOOD1P	1.000000	0	$1.4429e+00$	∞	$4.3361e-08$
WOODW	1.000000	0	$1.3045e+00$	∞	$2.4401e-08$

Table A.2: Performance of the Netlib bounds
i_{f^∇} – iterations to compute lower bound, t_{f^∇} – time to compute lower bound,
i_{f^Δ} – iterations to compute upper bound, t_{f^Δ} – time to compute upper bound,
t_{f^*} – time to compute approximate solution

Name	i_{f^∇}	t_{f^∇}/t_{f^*}	i_{f^Δ}	t_{f^Δ}/t_{f^*}
25FV47	1	0.102	0	0.030
80BAU3B	1	1.943	3	0.808
ADLITTLE	1	0.000	4	0.000
AFIRO	1		5	
AGG2	1	1.000	31	7.667
AGG3	1	0.750	31	5.750
AGG	1	0.500	2	4320000.000
BANDM	1	0.300	31	58.000
BEACONFD	1	1.000	12	8640100.000
BLEND	1	0.000	5	1.000
BNL1	1	0.519	0	3.444
BNL2	3	2.648	31	197.643
BORE3D	1	1.000	0	11.000
BRANDY	31	2.000	0	0.000
CAPRI	1	0.333	4	9.333
CYCLE	10	2.574	0	1.626
CZPROB	1	0.469	31	68.531
D2Q06C	1	0.277	31	26.499
D6CUBE	1	0.345	0	7.480
DEGEN2	1	0.152	0	1.273
E226	1	0.167	4	0.500
ETAMACRO	3	0.636	0	6.182
FFFFF800	1	0.421	31	35.000
FINNIS	31	3.500	2	2.250
FIT1D	0	0.000	2	0.125
FIT1P	1	0.321	10	68.887
FIT2D	0	0.003	1	0.009
FIT2P	1	0.722	12	243.487
GANGES	1	0.841	31	620.455
GFRD-PNC	1	0.833	7	173.417
GREENBEA	31	0.773	0	9.911
GREENBEB	31	0.813	0	10.568
GROW15	1	0.051	6	4.966
GROW22	1	0.060	9	13.667
GROW7	1	0.000	10	4.667
ISRAEL	1	0.000	1	0.333
KB2	1		4	
LOTFI	31	2.000	5	5.500
MAROS	31	0.919	0	2.364
MAROS-R7	1	0.381	10	105.517
MODSZK1	1	1.235	0	18.118
PEROLD	4	0.573	31	16.183
PILOT4	3	0.680	31	22.080
PILOT87	9	674.546	2	979.242
PILOT.JA	3	0.330	0	0.378
PILOTNOV	1	0.436	0	1.154
PILOT.WE	1	0.141	6	9.559
QAP8	10	21.895	0	0.091
RECIPE	0	0.000	0	0.000
SC105	1	0.000	1	0.000
SC205	1	0.500	1	2.000

continued...

Name	i_{f^∇}	t_{f^∇}/t_{f^*}	$i_{f^\triangle}$	$t_{f^\triangle}/t_{f^*}$
SC50A	1		1	
SC50B	1	0.000	1	0.000
SCAGR25	1	0.308	4	27.769
SCAGR7	1	0.000	4	4.000
SCFXM1	31	1.667	31	16.500
SCFXM2	31	1.000	31	25.103
SCFXM3	31	0.909	31	32.212
SCORPION	1	0.500	0	38.250
SCRS8	1	1.000	7	785536.364
SCSD1	1	0.500	13	12.500
SCSD6	1	0.571	15	16.429
SCSD8	1	0.654	20	42.904
SCTAP1	1	0.500	31	18.750
SCTAP2	1	1.310	28	124.517
SCTAP3	1	2.500	30	174.761
SHARE1B	1	0.500	8	5.000
SHARE2B	1	0.000	5	1.000
SHELL	1	0.941	0	40.706
SHIP04L	1	0.938	0	0.438
SHIP04S	1	1.000	0	0.333
SHIP08L	1	1.089	0	0.500
SHIP08S	1	1.259	0	0.556
SHIP12L	1	1.203	0	0.576
SHIP12S	1	1.102	0	0.475
SIERRA	0	0.480	0	14.960
STAIR	31	3.900	1	5.900
STANDATA	1	2.000	31	43.000
STANDGUB	1	8.000	0	2.000
STANDMPS	1	1.000	31	39.778
STOCFOR1	1	0.000	10	8.000
STOCFOR2	1	1.000	13	189.336
TRUSS	1	0.236	15	35.163
TUFF	14	4.286	0	0.286
VTP.BASE	1	1.000	31	17.000
WOOD1P	1	0.302	31	22.698
WOODW	1	0.602	7	61.329

Table A.3: Rigorous bounds for the Netlib problems using lp_solve 3.2
ρ_d – distance to dual infeasibility, ρ_p – distance to primal infeasibility,
f^∇ – lower bound, $f^\triangle$ – upper bound, μ – relative accuracy

Name	ρ_d	ρ_p	f^∇	$f^\triangle$	μ
SC105	0.133484	0	$-2.1696e+13$	$-5.2201e+01$	$1.0000e-00$
SC205	0.010023	0	$-\infty$	$-5.2201e+01$	$1.8125e-05$
SC50A	0.562500	0	$-5.8365e+04$	$-6.4574e+01$	$9.9889e-01$
SC50B	0.421875	0	$-7.3733e+02$	$-6.9999e+01$	$9.0506e-01$

Table A.4: Bounds for interval problems
ρ_d – distance to dual infeasibility, ρ_p – distance to primal infeasibility,
f^∇ – lower bound, $f^\triangle$ – upper bound, μ – relative accuracy

Name	ρ_d	ρ_p	f^∇	$f^\triangle$	μ
25FV47	0	0	$5.5013e+03$	∞	$9.2351e-05$
80BAU3B	0	0	$9.8720e+05$	$9.8726e+05$	$6.1059e-05$

continued...

Name	ρ_d	ρ_p	$f^{\triangledown}$	$f^{\triangle}$	μ
ADLITTLE	0.051651	0	$2.2549e+05$	∞	$3.0616e-05$
AFIRO	1.000000	0.397390	$-4.6476e+02$	$-4.6460e+02$	$3.4213e-04$
AGG2	0.771400	0	$-2.0240e+07$	∞	$2.7134e-05$
AGG3	0.771400	0	$1.0311e+07$	∞	$9.2632e-05$
AGG	0.771400	0	$-3.5993e+07$	∞	$3.3187e-05$
BANDM	0.000418	0	$-1.5864e+02$	∞	$7.2054e-05$
BEACONFD	0	0	$3.3592e+04$	∞	$1.1010e-05$
BLEND	0.040726	0.003541	$-3.0816e+01$	$-3.0803e+01$	$4.2114e-04$
BNL1	0.106400	0	$1.9775e+03$	∞	$8.6913e-05$
BNL2	0	0	$1.8112e+03$	∞	$2.2986e-05$
BORE3D	0.003539	0	$1.3731e+03$	∞	$2.1177e-05$
BRANDY	0	0	$-\infty$	∞	NaN
CAPRI	0.095510	0.000252	$2.6895e+03$	$2.6935e+03$	$1.4821e-03$
CYCLE	0	0	$-\infty$	∞	NaN
CZPROB	0.008807	0	$2.1852e+06$	∞	$1.3316e-05$
D2Q06C	0	0	$1.2278e+05$	∞	$4.8916e-05$
D6CUBE	2.000000	0	$3.1549e+02$	∞	$2.0404e-05$
DEGEN2	1.000000	0	$-1.4352e+03$	∞	$1.0747e-05$
E226	0	0	$-1.8753e+01$	∞	$3.1879e-01$
ETAMACRO	0.200000	0	$-7.5573e+02$	∞	$2.0841e-05$
FFFFF800	0.033046	0	$5.5565e+05$	∞	$4.7258e-05$
FINNIS	0	0	$-\infty$	∞	NaN
FIT1D	∞	3.500000	$-9.1464e+03$	$-9.1462e+03$	$1.8396e-05$
FIT1P	0.437500	1.271887	$9.1463e+03$	$9.1476e+03$	$1.4385e-04$
FIT2D	∞	317.000000	$-6.8465e+04$	$-6.8463e+04$	$1.5199e-05$
FIT2P	1.000000	1.057333	$6.8464e+04$	$6.8469e+04$	$8.6123e-05$
GANGES	1.000000	0	$-1.0960e+05$	∞	$1.6019e-04$
GFRD-PNC	0.347032	0	$6.9018e+06$	∞	$5.7493e-05$
GREENBEA	0	0	$-\infty$	∞	NaN
GREENBEB	0	0	$-\infty$	∞	NaN
GROW15	0.968073	0.572842	$-1.0687e+08$	$-1.0687e+08$	$1.6424e-05$
GROW22	0.968073	0.572842	$-1.6084e+08$	$-1.6083e+08$	$1.7269e-05$
GROW7	0.968073	0.572842	$-4.7788e+07$	$-4.7787e+07$	$1.6718e-05$
ISRAEL	0.166850	0.027248	$-8.9665e+05$	$-8.9664e+05$	$1.8453e-05$
KB2	0.018802	0.000201	$-1.7499e+03$	$-1.7498e+03$	$1.0083e-04$
LOTFI	0	0.000306	$-\infty$	$-2.5254e+01$	$4.1965e-04$
MAROS	0	0	$-\infty$	∞	NaN
MAROS-R7	0.628096	1.000000	$1.4972e+06$	$1.4973e+06$	$6.3798e-05$
MODSZK1	0.108469	0	$2.7040e+02$	∞	$1.6995e-01$
PEROLD	0.000943	0	$-\infty$	∞	NaN
PILOT4	0.000075	0	$-2.5813e+03$	∞	$7.4535e-05$
PILOT87	0	0	$3.0170e+02$	∞	$1.8799e-05$
PILOT.JA	0.000750	0	$-6.1134e+03$	∞	$5.1024e-05$
PILOTNOV	0.000750	0	$-4.4974e+03$	∞	$3.3681e-05$
PILOT.WE	0.044874	0	$-2.7204e+06$	∞	$1.2197e-04$
QAP8	4.000000	0	$-\infty$	∞	NaN
RECIPE	0	0	$-2.6662e+02$	∞	$2.3744e-05$
SC105	0.133484	0	$-5.2205e+01$	$-5.2199e+01$	$1.2454e-04$
SC205	0.010023	0	$-5.2206e+01$	∞	$7.3685e-05$
SC50A	0.562500	0	$-6.4578e+01$	$-6.4573e+01$	$7.5860e-05$
SC50B	0.421875	0	$-7.0003e+01$	$-6.9998e+01$	$7.7140e-05$
SCAGR25	0.034646	0.021077	$-1.4754e+07$	$-1.4752e+07$	$1.1701e-04$
SCAGR7	0.034646	0.022644	$-2.3315e+06$	$-2.3312e+06$	$1.2282e-04$
SCFXM1	0	0	$-\infty$	∞	NaN
SCFXM2	0	0	$-\infty$	∞	NaN
SCFXM3	0	0	$-\infty$	∞	NaN

continued...

Name	ρ_d	ρ_p	$f^{\triangledown}$	$f^{\triangle}$	μ
SCORPION	0.949393	0	$1.8781e+03$	∞	$2.9037e-05$
SCRS8	0	0	$9.0426e+02$	∞	$3.7130e-05$
SCSD1	1.000000	5.037757	$8.6665e+00$	$8.6677e+00$	$1.4261e-04$
SCSD6	1.000000	1.603351	$5.0499e+01$	$5.0506e+01$	$1.4791e-04$
SCSD8	1.000000	0.268363	$9.0494e+02$	$9.0550e+02$	$6.2245e-04$
SCTAP1	1.000000	0.032258	$1.4122e+03$	$1.4125e+03$	$2.2764e-04$
SCTAP2	1.000000	0.586563	$1.7248e+03$	$1.7260e+03$	$6.8090e-04$
SCTAP3	1.000000	0.381250	$1.4240e+03$	∞	$1.1257e-05$
SHARE1B	0.000751	0.000015	$-7.6602e+04$	$-7.6428e+04$	$2.2704e-03$
SHARE2B	0.287893	0.001747	$-4.1584e+02$	$-4.1559e+02$	$6.1583e-04$
SHELL	1.777778	0	$1.2088e+09$	∞	$1.4568e-05$
SHIP04L	13.146000	0	$1.7933e+06$	∞	$1.0788e-05$
SHIP04S	13.146000	0	$1.7987e+06$	∞	$1.1168e-05$
SHIP08L	21.210000	0	$1.9090e+06$	∞	$1.1798e-05$
SHIP08S	21.210000	0	$1.9201e+06$	∞	$1.2431e-05$
SHIP12L	7.434000	0	$1.4702e+06$	∞	$1.3344e-05$
SHIP12S	7.434000	0	$1.4892e+06$	∞	$1.5090e-05$
SIERRA	∞	0	$1.5376e+07$	∞	$1.1848e-03$
STAIR	0	0.000580	$-\infty$	$-2.5126e+02$	$3.7820e-05$
STANDATA	1.000000	0	$1.2577e+03$	∞	$1.3912e-05$
STANDGUB	0	0	$1.2577e+03$	∞	$1.3912e-05$
STANDMPS	1.000000	0	$1.4060e+03$	∞	$1.4991e-05$
STOCFOR1	0.011936	0.001203	$-4.1134e+04$	$-4.1126e+04$	$1.8826e-04$
STOCFOR2	0.000064	0.000437	$-3.9026e+04$	∞	$4.4179e-05$
TRUSS	10.000000	0.518928	$4.5877e+05$	$4.5910e+05$	$7.1544e-04$
TUFF	0.017485	0	$2.9213e-01$	∞	$1.9790e-05$
VTP.BASE	0.500000	0	$1.2982e+05$	∞	$5.9034e-05$
WOOD1P	1.000000	0	$1.4428e+00$	∞	$4.7868e-05$
WOODW	1.000000	0	$1.3044e+00$	∞	$2.6084e-05$

Table A.5: Performance of the interval bounds
$i_{f^{\triangledown}}$ – iterations to compute lower bound, $t_{f^{\triangledown}}$ – time to compute lower bound,
$i_{f^{\triangle}}$ – iterations to compute upper bound, $t_{f^{\triangle}}$ – time to compute upper bound,
t_{f^*} – time to compute approximate solution

Name	$i_{f^{\triangledown}}$	$t_{f^{\triangledown}}/t_{f^*}$	$i_{f^{\triangle}}$	$t_{f^{\triangle}}/t_{f^*}$
25FV47	1	0.107	0	0.030
80BAU3B	1	1.921	1	0.393
ADLITTLE	1		31	
AFIRO	1		9	
AGG2	1	0.750	31	3.250
AGG3	1	0.750	31	3.250
AGG	1	0.667	2	2880000.000
BANDM	1	0.273	31	14.455
BEACONFD	1	0.500	9	8640050.000
BLEND	1	0.000	9	1.000
BNL1	1	0.517	0	3.172
BNL2	1	1.809	31	87.787
BORE3D	1	0.333	0	6.667
BRANDY	31	1.333	0	0.333
CAPRI	3	1.000	8	7.250
CYCLE	19	11.365	0	1.670
CZPROB	1	0.420	31	40.407
D2Q06C	1	0.283	31	15.738
D6CUBE	1	0.360	0	7.500

continued...

Name	$i_{f^\triangledown}$	$t_{f^\triangledown}/t_{f^*}$	$i_{f^\triangle}$	$t_{f^\triangle}/t_{f^*}$
DEGEN2	1	0.152	0	1.818
E226	1	0.200	31	4.000
ETAMACRO	1	0.417	0	5.750
FFFFF800	1	0.421	3	454752.632
FINNIS	31	2.625	2	1.250
FIT1D	0	0.000	5	0.250
FIT1P	1	0.340	2	35.887
FIT2D	0	0.005	4	0.028
FIT2P	1	0.728	2	158.333
GANGES	1	0.889	31	588.844
GFRD-PNC	1	0.917	9	153.417
GREENBEA	31	0.391	0	9.666
GREENBEB	31	0.434	0	10.398
GROW15	1	0.033	2	3.083
GROW22	1	0.059	2	6.812
GROW7	1	0.200	2	2.800
ISRAEL	1	0.500	1	0.000
KB2	1		4	
LOTFI	31	1.500	2	3.500
MAROS	31	0.505	0	2.218
MAROS-R7	1	0.404	2	64.692
MODSZK1	11	5.688	0	19.500
PEROLD	13	4.159	31	14.732
PILOT4	7	1.250	31	11.917
PILOT87	1	0.073	31	0.711
PILOT.JA	7	0.582	0	0.365
PILOTNOV	1	0.436	0	1.103
PILOT.WE	6	0.337	18	29697.938
QAP8	18	753.096	0	0.088
RECIPE	0		0	
SC105	1		6	
SC205	1	0.500	31	8.500
SC50A	1		6	
SC50B	1		2	
SCAGR25	1	0.308	5	18.615
SCAGR7	1	0.000	3	3.000
SCFXM1	31	1.167	31	8.333
SCFXM2	31	0.621	31	12.655
SCFXM3	31	0.478	31	17.567
SCORPION	1	0.500	0	26.750
SCRS8	1	1.000	31	104.000
SCSD1	1	0.500	12	11.500
SCSD6	1	0.714	12	14.857
SCSD8	1	4.635	14	33.923
SCTAP1	1	0.750	9	7.000
SCTAP2	1	1.407	9	65.370
SCTAP3	1	2.667	31	104.622
SHARE1B	1	0.500	8	4.000
SHARE2B	1	0.000	6	1.000
SHELL	1	1.125	0	62.500
SHIP04L	1	0.941	0	0.353
SHIP04S	1	0.917	0	0.333
SHIP08L	1	1.125	0	0.482
SHIP08S	1	1.214	0	0.536
SHIP12L	1	1.239	0	0.769
SHIP12S	1	1.049	0	0.508

continued...

Name	$i_{f^\triangledown}$	$t_{f^\triangledown}/t_{f^*}$	$i_{f^\triangle}$	$t_{f^\triangle}/t_{f^*}$
SIERRA	0	0.480	0	14.320
STAIR	31	3.300	6	9.400
STANDATA	1	7.000	31	80.000
STANDGUB	1	4.000	0	1.000
STANDMPS	1	1.250	31	23.625
STOCFOR1	1	0.000	15	4.000
STOCFOR2	1	1.076	31	211.750
TRUSS	1	0.200	11	29.779
TUFF	12	4.286	0	0.429
VTP.BASE	1	0.500	1	0.500
WOOD1P	1	0.333	0	8.405
WOODW	1	0.531	3	46.261

Bibliography

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press, New York, 1983.

[2] D. L. Applegate, W. Cook, et al. QSopt_ex. World Wide Web. URL http://www.dii.uchile.cl/~daespino/QSoptExact_doc/main.html.

[3] D. L. Applegate, W. Cook, et al. Exact solutions to linear programming problems. *submitted to Operations Research Letters*, 2006.

[4] H. Beeck. Linear Programming with Inexact Data. Technical Report 7830, Abteilung Mathematik, TU München, 1978.

[5] A. Bemporad and D. Mignone. *miqp.m: A Matlab function for solving Mixed Integer Quadratic Programs, Version 1.06, User Guide*. Institut für Automatik, ETH – Swiss Federal Institute of Technology, ETHZ – ETL, CH 8092 Zürich, Switzerland.

[6] A. Ben-Tal and A. S. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*, volume 2 of *MPS/SIAM Series on Optimization*. SIAM, 3600 University City Science Center, Philadelphia, PA 19104-268, 2001.

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

[8] M. Berz et al. COSY Infinity. World Wide Web. URL http://www.bt.pa.msu.edu/index_files/cosy.htm.

[9] S. Bollapragada, H. Cheng, et al. NBC's Optimization Systems Increase Revenues and Productivity. *Interfaces*, 32(1):47–60, January–February 2002.

[10] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, The Edinburgh Building, Cambridge, CB2 2RU, UK, 2004.

[11] J. V. Caixeta-Filho, J. M. van Swaay-Neto, and A. d. P. Wagemaker. Optimization of the Production Planning and Trade of Lily Flowers at Jan de Wit Company. *INTERFACES*, 32(1):35–46, 2002. doi:10.1287/inte.32.1.35.13. URL http://interfaces.journal.informs.org/cgi/content/abstract/32/1/35.

[12] K. C. Chalermkraivuth, S. Bollapragada, et al. GE Asset Management, Genworth Financial, and GE Insurance Use a Sequential-Linear-Programming Algorithm to Optimize Portfolios. *INTERFACES*, 35(5):370–380, 2005. doi:10.1287/inte.1050.0164. URL http://interfaces.journal.informs.org/cgi/content/abstract/35/5/370.

[13] D. Clarke. The trouble with rounding floating point numbers. World Wide Web. URL http://www.theregister.co.uk/2006/08/12/floating_point_approximation.

[14] J. Czyzyk, M. P. Mesnier, and J. J. More. The NEOS Server. *IEEE Computational Science and Engineering*, 05(3):68–75, 1998. ISSN 1070-9924. doi:10.1109/99.714603.

[15] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T. Koopmans, editor, *Activity Analysis of Production and Allocation*, number 13 in Cowles Commission Monographs, pages 339–347. John Wiley & Sons, Inc, 440 Fourth Ave., New York City, 1951.

[16] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 41 William Street, Princeton, New Jersey 08540, first edition, 1963.

[17] G. B. Dantzig. Linear Programming. In J. K. Lenstra, A. H. G. R. Kan, and A. Schrijver, editors, *History of Mathematical Programming: A Collection of Personal Reminiscences*, pages 19–31. Elsevier Science Publishers, 1991.

[18] Dash Optimization. Xpress. URL http://www.dashoptimization.com.

[19] C. de la Vallée Poussin. Sur la méthode de l'approximation minimum. *Annales de la Société Scientifique*, 35:1–16, 1911.

[20] M. Dhiflaoui, S. Funke, et al. Certifying and Repairing Solutions to large LPs – How Good are LP-solvers? In *SODA*, pages 255–256. 2003.

[21] E. D. Dolan. NEOS Server 4.0 Administrative Guide. Technical Memorandum 250, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, May 2001.

[22] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91(2):201–213, January 2001. doi:10.1007/s101070100263.

[23] L. Doubli. *Numerische Experimente für gemischt-ganzzahlige Optimierungsprobleme*. Studienarbeit (research paper), Hamburg University of Technology, 2008.

[24] L. Fan, B. Bertok, and F. Friedler. A graph-theoretic method to identify candidate mechanisms for deriving the rate law of a catalytic reaction. *COMPUTERS & CHEMISTRY*, 26(3):265–292, February 2002. ISSN 0097-8485.

[25] J. Farkas. Theorie der einfachen Ungleichungen. *Journal für die reine und angewandte Mathematik*, 124:1–27, 1902.

[26] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, volume 4 of *SIAM Classics in Applied Mathematics*. Republished by SIAM, 3600 University City Science Center, Philadelphia, PA 19104-2688, 1990.

[27] R. Fletcher and S. Leyffer. MINLP. World Wide Web. URL http://neos.mcs.anl.gov/neos/solvers/minco:MINLP/ampl.html.

[28] C. A. Floudas. *Deterministic Global Optimization - Theory, Methods and Applications*, volume 37 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, Dordrecht, Boston, London, 2000.

[29] R. Fourer and D. M. Gay. Experience with a Primal Presolve Algorithm. In W. W. Hager, D. W. Hearn, and P. M. Pardalos, editors, *Large Scale Optimization: State of the Art*, pages 135–154. Kluwer Academic Publishers Group, Norwell, MA, USA, and Dordrecht, The Netherlands, 1994. URL citeseer.ist.psu.edu/fourer94experience.html.

[30] J. B. J. Fourier. Solution d'une question particulière du calcul des inégalités. *Nouveau Bulletin des sciences par la Société philomathique de Paris*, pages 99–100, 1826.

[31] Free Software Foundation. gcc. World Wide Web. URL http://gcc.gnu.org.

[32] Free Software Foundation. GLPK. World Wide Web. URL http://www.gnu.org/software/glpk/.

[33] G. Fung and J. Stoeckel. SVM feature selection for classification of SPECT images of Alzheimer's disease using spatial information. *KNOWLEDGE AND INFORMATION SYSTEMS*, 11(2):243–258, February 2007. ISSN 0219-1377. doi:10.1007/s10115-006-0043-5.

[34] T. Gal. Selected bibliography on degeneracy. *Annals of Operations Research*, 46(1):1–7, March 1993.

[35] B. Gärtner. Exact Arithmetic at Low Cost – a Case Study in Linear Programming. *Computational Geometry*, 13(2):121–139, June 1999.

[36] D. Gay. Electronic Mail Distribution of Linear Programming Test Problems. *COAL Newsletter*, 13:10–12, December 1985.

[37] W. Givens. Numerical computation of the characteristic values of a real symmetric matrix. Technical Report ORNL-1574, Oak Ridge National Laboratory, Oak Ridge, TN, 1954.

[38] D. Goldberg. What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Computaing Surveys*, 23(1):5–47, 1991.

[39] T. Granlund et al. GNU Multiple Precision Arithmetic library. World Wide Web. URL http://gmplib.org.

[40] L. Granvilliers and F. Benhamou. Algorithm 852: RealPaver: An Interval Solver using Constraint Satisfaction Techniques. *ACM Transactions on Mathematical Software*, 32(1):138–156, 2006. doi:10.1145/1132973.1132980.

[41] H. J. Greenberg. An Analysis of Degeneracy. *Naval Research Logistics Quaterly*, 33:635–655, 1986.

[42] W. Gropp and J. J. Moré. Optimization Environments and the NEOS Server. In M. D. Buhmann and A. Iserles, editors, *Approximation Theory and Optimization*, pages 167–182. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, United Kingdom, 1997.

[43] L. Hafer. bonsaiG: Algorithms and Design. Technical Report 1999-06, School of Computing Science, Simon Fraser University, Burnaby, BC, Canada, 1999.

[44] N. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM Publications, Philadelphia, 1996.

[45] A. Holder, editor. *Mathematical Programming Glossary*. INFORMS Computing Society, 2006–07. URL http://glossary.computing.society.informs.org. Originally authored by Harvey J. Greenberg, 1999-2006.

[46] X. Huang and Y. Fang. Multiconstrained QoS multipath routing in wireless sensor networks. *WIRELESS NETWORKS*, 14(4):465–478, August 2008. ISSN 1022-0038. doi:10.1007/s11276-006-0731-9.

[47] *IEEE 754 Standard for Floating-Point Arithmetic*, 1986.

[48] *ANSI/IEEE 854-1987, Standard for Radix-Independent Floating-Point Arithmetic*, 1987.

[49] ILOG. CPLEX. URL http://www.ilog.com/products/cplex.

[50] C. Jansson. VSDP: Verified SemiDefinite Programming. World Wide Web. URL http://www.ti3.tu-harburg.de/jansson/vsdp.

[51] C. Jansson. A Self-Validating Method for Solving Linear Programming Problems with Interval Input Data. *Computing*, Suppl. 6:33–45, 1988.

[52] C. Jansson. Rigorous Lower and Upper Bounds in Linear Programming. *SIAM J. Optimization (SIOPT)*, 14(3):914–935, 2004.

[53] C. Jansson. Guaranteed Accuracy for Conic Programming Problems in Vector Lattices, 2007. URL http://www.citebase.org/abstract?id=oai:arXiv.org:0707.4366.

[54] C. Jansson, D. Chaykin, and C. Keil. Rigorous Error Bounds for the Optimal Value in Semidefinite Programming. *SIAM Journal on Numerical Analysis*, 46(1):180–200, 2007. doi:10.1137/050622870. URL http://link.aip.org/link/?SNA/46/180/1.

[55] L. V. Kantorovich. *Mathematical methods in the organization and planning of production (in Russian)*. Publication House of the Leningrad State University, 1939.

[56] L. V. Kantorovich. Mathematical methods in the organization and planning of production. *Management Science*, 6:363–422, 1960. English translation of [55].

[57] N. K. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, December 1984.

[58] R. B. Kearfott. GlobSol. World Wide Web. URL http://interval.louisiana.edu.

[59] R. B. Kearfott. GlobSol: History, Composition, and Advice on Use. In C. Bliek, C. Jermann, and A. Neumaier, editors, *Global Optimization and Constraint Satisfaction*, volume 2861 of *Lecture Notes in Computer Science*, pages 17–31. Springer Berlin / Heidelberg, 2003.

[60] R. B. Kearfott. GlobSol – Present State and Future Developments, February 27 2007. Talk given at the International Workshop on Numerical Verification and its Applications, Waseda University, Tokyo.

[61] C. Keil. *Lurupa Documentation*. URL http://www.ti3.tu-harburg.de/~keil/lurupa.

[62] C. Keil. *LURUPA – Rigorose Fehlerschranken für Lineare Programme*. Diplomarbeit, Technische Universität Hamburg–Harburg, 2004. URL http://www.ti3.tu-harburg.de/~keil/pub/LRFfLP.ps.gz.

[63] C. Keil. Lurupa – Rigorous Error Bounds in Linear Programming. In B. Buchberger, S. Oishi, et al., editors, *Algebraic and Numerical Algorithms and Computer-assisted Proofs*, number 05391 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. URL http://drops.dagstuhl.de/opus/volltexte/2006/445.

[64] C. Keil. A Comparison of Software Packages for Verified Linear Programming, 2008. URL http://www.optimization-online.org/DB_HTML/2008/06/2007.html. Submitted to SIAM Journal on Optimization, preprint.

[65] C. Keil and C. Jansson. Computational Experience with Rigorous Error Bounds for the Netlib Linear Programming Library. *Reliable Computing*, 12(4):303–321, 2006. URL http://www.optimization-online.org/DB_HTML/2004/12/1018.html.

[66] L. G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979. English translation of [67].

[67] L. G. Khachiyan. A polynomial algorithm in linear programming (in Russian). *Doklady Akademiia Nauk SSSR*, 224:1093–1096, 1979.

[68] M. Kiyomi. exlp. World Wide Web. URL http://members.jcom.home.ne.jp/masashi777/exlp.html.

[69] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities, III*, pages 159–175. Academic Press, New York, NY, 1972.

[70] O. Knüppel. PROFIL/BIAS and extensions, Version 2.0. Technical report, Inst. f. Informatik III, Technische Universität Hamburg-Harburg, 1998.

[71] T. Koch. perPlex. World Wide Web. URL http://www.zib.de/koch/perplex.

[72] T. Koch. The final Netlib-LP results. Technical Report 03-05, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustraße 7, D-14195 Berlin-Dahlem, Germany, 2003.

[73] R. Krawczyk. Fehlerabschätzung bei linearer Optimierung. In K. Nickel, editor, *Interval Mathematics*, volume 29 of *Lecture Notes in Computer Science*, pages 215–222. Springer Verlag, Berlin, 1975.

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

[75] L. J. LeBlanc, J. A. Hill, Jr., et al. Nu-kote's Spreadsheet Linear-Programming Models for Optimizing Transportation. *INTERFACES*, 34(2):139–146, 2004. doi:10.1287/inte.1030.0051. URL http://interfaces.journal.informs.org/cgi/content/abstract/34/2/139.

[76] M. S. Lobo, L. Vandenberghe, et al. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284(1–3):193–228, 1998. URL http://www.stanford.edu/~boyd/socp.html.

[77] L. Lovász. A New Linear Programming Algorithm – Better or Worse Than the Simplex Method? *The Mathematical Intelligencer*, 2:141–146, 1980.

[78] K. Maki, T. Wakuda, et al. Development of Shimming System for a Highly Sensitive NMR Spectrometer with a Superconducting Split Magnet. *IEEE TRANSACTIONS ON APPLIED SUPERCONDUCTIVITY*, 18(2):844–847, June 2008. ISSN 1051-8223. doi:10.1109/TASC.2008.921881.

[79] G. Mayer. Result verification for eigenvectors and eigenvalues. In J. Herzberger, editor, *Topics in validated computations. Proceedings of the IMACS-GAMM international workshop, Oldenburg, Germany, 30 August - 3 September 1993*, Stud. Comput. Math. 5, pages 209–276. Elsevier, Amsterdam, 1994.

[80] N. Megiddo. On the complexity of linear programming. In *Advances in Economy Theory, Fifth World Congress*, chapter 6, pages 225–268. Cambridge University Press, 1987.

[81] C. Mészáros. Linear Programming Test Problems. World Wide Web. URL http://www.sztaki.hu/~meszaros/bpmpd.

[82] R. E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1966.

[83] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, 1979.

[84] R. E. Moore. The dawning. *Reliable Computing*, 5(4):423–424, November 1999.

[85] A. S. Nemirovski and D. B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. John Wiley, 1983.

[86] Y. Nesterov and A. S. Nemirovski. *Interior-Point Polynomial Algorithms in Convex Programming*, volume 13 of *Studies in Applied Mathematics*. SIAM, 3600 University City Science Center, Philadelphia, PA 19104-268, 1994. ISBN 978-0-898715-15-6.

[87] netlib. netlib linear programming library. World Wide Web. URL http://www.netlib.org/lp.

[88] A. Neumaier. *Interval Methods for Systems of Equations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1990.

[89] A. Neumaier. *Interval Methods for Systems of Equations*, volume 37 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, UK, 1990. ISBN 0-521-33196-X.

[90] A. Neumaier. *Introduction to Numerical Analysis*. Cambridge University Press, 2001.

[91] A. Neumaier. Complete Search in Continuous Global Optimization and Constraint Satisfaction. *Acta Numerica*, 13:271–369, 2004.

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

[93] A. Neumaier, O. Shcherbina, et al. A comparison of complete global optimization solvers. *Math. Program.*, 103(2):335–356, 2005. ISSN 0025-5610. doi:10.1007/s10107-005-0585-4.

[94] J. Neumann and H. Goldstine. Numerical Inverting of Matrices of High Order. *Bull. Amer. Math. Soc. 53*, pages 1021–1099, 1947.

[95] E. Omer, R. Guetta, et al. Optimal design of an "Energy Tower" power plant. *IEEE TRANSACTIONS ON ENERGY CONVERSION*, 23(1):215–225, March 2008. ISSN 0885-8969. doi:10.1109/TEC.2007.905349.

[96] OptiRisk Systems. FortMP. URL http://www.optirisk-systems.com/products_fortmp.asp.

[97] F. Ordóñez and R. M. Freund. Computational experience and the explanatory value of condition measures for linear optimization. *SIAM J. Optimization*, 14(2):307–333, 2003.

[98] A. L. Pomerantsev and O. Y. Rodionova. Construction of a multivariate calibration by the simple interval calculation method. *JOURNAL OF ANALYTICAL CHEMISTRY*, 61(10):952–966, October 2006. ISSN 1061-9348. doi:10.1134/S1061934806100030.

[99] J. Renegar. Some perturbation theory for linear programming. *Math. Program.*, 65(1):73–91, 1994. ISSN 0025-5610. doi:10.1007/BF01581690.

[100] J. Renegar. Incorporating Condition Measures into the Complexity Theory of Linear Programming. *SIAM Journal on Optimization*, 5(3):506–524, 1995. doi:10.1137/0805026. URL http://link.aip.org/link/?SJE/5/506/1.

[101] J. Renegar. Linear programming, complexity theory and elementary functional analysis. *Math. Program.*, 70(3, Ser. A):279–351, 1995.

[102] R. T. Rockafellar. Lagrange multipliers and optimality. *SIAM Review*, 35(2):183–238, June 1993.

[103] J. B. Rosen and S. Suzuki. Construction of Nonlinear Programming Test Problems. *Communication of ACM*, 8:113, 1965.

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

[105] S. M. Rump. Validated Solution of Large Linear Systems. In R. Albrecht, G. Alefeld, and H. Stetter, editors, *Validation numerics: theory and applications*, volume 9 of *Computing Supplementum*, pages 191–212. Springer, 1993.

[106] S. M. Rump. Verification Methods for Dense and Sparse Systems of Equations. In J. Herzberger, editor, *Topics in Validated Computations — Studies in Computational Mathematics*, pages 63–136. Elsevier, Amsterdam, 1994.

[107] N. Z. Shor. Cut-off method with space extension in convex programming problems. *Cybernetics*, 13(1):94–96, 1977. English translation of [108].

[108] N. Z. Shor. Cut-off method with space extension in convex programming problems (in Russian). *Kibernetika*, 13:94–95, 1977.

[109] R. Skeel. Roundoff Error and the Patriot Missile. *SIAM News*, 25(4):11, July 1992.

[110] G. Tarsy and N. Toda. Floating-Point Computing: A Comedy of Errors? World Wide Web. URL http://developers.sun.com/solaris/articles/fp_errors.html.

[111] M. Tawarmalani and N. V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99:563–591, 2004.

[112] The Mathworks. MATLAB. URL http://www.mathworks.com/products/matlab.

[113] Thomson Scientific. Web of Science. World Wide Web. URL http://newisiknowledge.com.

[114] M. J. Todd. The many facets of linear programming. *Math. Program., Ser. B*, 91(3):417–436, February 2002.

[115] A. M. Turing. Rouding-off Errors in Matrix Processes. *The Quarterly Journal of Mechanics and Applied Mathematics*, 1(1):287–308, 1948.

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

[117] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[118] R. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, 1996.

[119] X.-H. Vu, D. Sam-Haroud, and M.-C. Silaghi. Approximation Techniques for Non-linear Problems with Continuum of Solutions. In R. C. H. Sven Koenig, editor, *Proceedings of The 5th International Symposium on Abstraction, Reformulation and Approximation (SARA'2002)*, volume LNAI 2371 of *Lecture Notes in Computer Science - Lecture Notes in Artificial Intelligence*, pages 224–241. Springer-Verlag, Canada, August 2002. ISBN 3-540-43941-2. © Springer-Verlag.

[120] X.-H. Vu, D. Sam-Haroud, and M.-C. Silaghi. Numerical Constraint Satisfaction Problems with Non-isolated Solutions. In *1st International Workshop on Global Constrained Optimization and Constraint Satisfaction (COCOS'2002)*. COCONUT Consortium, France, October 2002.

[121] R.-S. Wang, Y. Wang, et al. Analysis on multi-domain cooperation for predicting protein-protein interactions. *BMC BIOINFORMATICS*, 8, October 16 2007. ISSN 1471-2105. doi:10.1186/1471-2105-8-391.

[122] J. H. Wilkinson. Error Analysis of Floating-point Computation. *Numerische Mathematik*, 2(1):319–340, December 1960.

[123] R. Wunderling. Soplex. World Wide Web. URL `http://soplex.zib.de`.

[124] S.-T. Xia, F.-W. Fu, and Y. Jiang. On the minimum average distance of binary constant weight codes. *DISCRETE MATHEMATICS*, 308(17):3847–3859, September 6, 2008. ISSN 0012-365X. doi:10.1016/j.disc.2007.07.081.

[125] D. B. Yudin and A. S. Nemirovski. Informational complexity and efficient methods for the solution of convex extremal problems. *Matekon*, 13(2):3–25, 1976. English translation of [126].

[126] D. B. Yudin and A. S. Nemirovski. Informational complexity and efficient methods for the solution of convex extremal problems (in Russian). *Ékonomika i Matematicheskie metody*, 12:357–369, 1976.

Lebenslauf

Persönliche Daten

Name	Christian Keil
Geburtsdatum und -ort	3. März 1978 in Hamburg

Berufserfahrung

März 2004 – Juni 2008	**Wissenschaftlicher Mitarbeiter** Institut für Zuverlässiges Rechnen Technische Universität Hamburg–Harburg
Oktober 2002 – Februar 2003	**Fachpraktikum als Entwickler** I–TO–I GmbH Hamburg

Studium und Ausbildung

Oktober 1998 – Februar 2004	**Diplom Informatik-Ingenieurwesen** Vertiefungsrichtung Wissenschaftliches Rechnen Technische Universität Hamburg–Harburg
1984 – 1997	**Abitur** Friedrich Ebert Gymnasium Hamburg