Delta-Complete Linear Programming Techniques for Satisfiability and Numerical Optimisation

Martin Sidaway

A thesis submitted for the qualification of

Doctor of Philosophy

in the School of Computing at Newcastle University.

June 2024

Abstract

In this thesis, I develop rigorous algorithms for solving linear satisfiability and optimisation problems within a Satisfiability Modulo Theories (SMT) framework. Over the last 20 years, extensive applications for such algorithms have opened up in the formal verification of software and, more recently, of rectified linear-unit (ReLU) neural networks. These networks learn a function that is continuous and piecewise linear over the inputs. Many of the most important safety properties for a ReLU neural network, in addition to the linear segments of the network itself, can be encoded as conjunctions of linear inequalities. This suggests that developing better methods to solve satisfiability problems over conjunctions of linear inequalities ("linear feasibility problems") should be a priority for the neural network verification community.

The most popular, and in many cases the most efficient method for solving linear feasibility problems is the simplex method. There also exists a family of methods known as interior-point methods, which have been found in some cases to outperform the simplex method. In this thesis, two algorithms are developed for the rigorous δ-complete solution of linear feasibility problems. One is based on the simplex method, and the other is based on an interior-point method.

(A solver is considered δ-complete if it is a sound and complete method for solving the δ-decision problem corresponding to an SMT decision problem. The δ-decision problem is the problem of deciding between unsatisfiability of the original SMT decision problem and satisfiability of the δ-relaxed problem, which is the problem in which all arithmetic constraints, expressed as a set of inequalities, have been "relaxed" by adding or subtracting δ on one side or the other, in such a way as to make them easier to satisfy.)

Both algorithms were implemented in software, and the interior-point method was found to be slower by many orders of magnitude. As a result, the majority of this thesis focuses on the simplex-based method, implemented as dLinear4,[1] which is my modified version of the delta-complete SMT solver dReal4.[2]

To maximise the efficiency of the implementation, the simplex algorithm was built on top of an inexact, floating-point solver. This solver is then enclosed in a loop that tries an increasing sequence of floating-point precisions. I prove that the algorithm terminates, as the floating-point solver will eventually identify the exact solution. Correctness (for the δ-decision problem) is ensured by the rational checks that follow the call to the floating-point solver. The time savings of making the method δ-complete are realised as a reduction in the number of calls to the floating-point solver, when an infeasible basis is nevertheless found to yield a δ-satisfying primal solution.

[1] https://github.com/martinjos/dlinear4

[2] https://github.com/dreal/dreal4

My implementation dLinear4 was tested on a wide range of SMT- and LP-derived problem instances. It was found to beat leading SMT solvers such as z3[3] and cvc4[4] on most LP-derived instances, even when an exact solution was sought ($\delta = 0$). Some additional time savings have been found on larger instances when $\delta > 0$.

Finally, a new concept of δ-completeness for full linear programs is developed. This concept demands rigorous bounds confining the optimal objective function value (where applicable) to a range no larger than δ. An algorithm is defined, proven δ-complete according to this new concept (building upon the proof for the feasibility case), and implemented in dLinear4.

[3]https://github.com/Z3Prover/z3

[4]https://cvc4.github.io

Acknowledgements

I would like to thank Dr. Paolo Zuliani, my supervisor, for his continual support and invaluable advice.

I would also like to thank my examiners, Dr. Sadegh Soudjani of Newcastle University, and Dr. Ambros Gleixner of the Zuse Institute Berlin, for their invaluable feedback.

This research made use of the Rocket High Performance Computing service at Newcastle University.

Contents

Chapter 1

Introduction

1.1 Background

1.1.1 ReLU Neural Networks

ReLU[1] neural networks have lately become the most widely-used form of artificial neural network (henceforth "neural network"), and have been finding an ever-expanding range of applications. They learn a function that is continuous and piecewise linear over the inputs. In this piecewise linear function, the number of linear segments is exponential in the number of network layers, often making it prohibitively expensive to rigorously verify global properties of the function, especially in a "deep" network (one with many layers) – which has been found to be the most powerful form of neural network.

In many cases the desired safety property is encoded as a decision problem in which *true* means that the property is violated, and *false* means that it holds. For instance, suppose we have a neural network that is designed to drive a car autonomously. Then we may ask the question, "will the network ever accelerate when the speed is at or above the speed limit?" And let's say, just for the sake of argument, that we want (and expect) the answer to be "no", or in other words "false". If the correct answer is "false" (it won't accelerate), but our solver gives us the answer "true", this is okay – it may make us overly cautious, but no-one will be hurt as a result. However, if the correct answer is "true", but our solver says "false", then we may place undue trust in the network.

In such cases, overapproximation – where false positives are allowed but false negatives are not – can be helpful. For instance, δ-complete methods (for satisfiability problems over conjunctions of inequalities) rule out false negatives – which would correspond to overlooked property violations – but permit false positives, provided that no bound is violated by more than the chosen value δ. This value δ provides a means of tuning the tradeoff between execution time and solution quality. The problem solved by a δ-complete method is known as the *δ-decision problem* – as opposed to the *decision problem*, which is the problem of finding the answer "true" or "false" with no allowance for false positives.

[1]Rectified Linear Unit

1.1.2 Delta-Decidability and Delta-Completeness

The concept of delta-decidability (or δ-decidability), introduced in [3], is concerned with relaxing the strict completeness requirement of a decision procedure for nonlinear real arithmetic sentences with quantifiers, in much the way indicated above for ReLU neural networks. In [4], this concept is applied to the more restricted problem of the satisfiability of quantifier-free Satisfiability Modulo Theories (SMT) [5] formulas. It is important to understand that δ-completeness does not affect the correctness of the algorithms, nor the guarantee of termination: only the nature of the results that may be provided. For instance, a δ-complete algorithm for satisfiability pronounces either unsatisfiability or δ-satisfiability, and in both cases, the result is exact. In the case of δ-satisfiability, it constitutes a mathematically rigorous assertion that a bounded perturbation of the input problem – known as the δ-decision problem – is exactly satisfiable.

1.1.3 Verifying ReLU Neural Networks

A ReLU neural network is a real-vector-valued, continuous, piecewise linear function of real vectors, composed of "rectified linear units", each of which is the composition of a rectifier and a multivariate affine function. Typically, the rectifier itself is denoted ReLU, despite the fact that ReLU stands for "rectified linear unit". Hence, a rectified linear unit with n inputs will typically look like

$$f(x) = \mathrm{ReLU}(a \cdot x + b), \text{ for some } a \in \mathbb{R}^n \text{ and } b \in \mathbb{R},$$

where

$$\mathrm{ReLU}(\chi) = \max(0, \chi), \qquad \chi \in \mathbb{R},$$

and the components of x in $f(x)$ may themselves be the outputs of further rectified linear units.

Verifying the "behaviour" of these "networks" (i.e., how the function responds to its inputs) typically involves composing a satisfiability problem, of which the function itself is a parameter. For instance, we may ask a solver to prove (or, equivalently, to disprove) the statement

$$\exists x : f(x) \geqslant \zeta, \qquad \ell \leqslant x \leqslant u, \tag{1.1}$$

where $\ell, u \in \mathbb{R}^n$ bound some hyperrectangular region of the input space that we are interested in, and $\zeta \in \mathbb{R}$ is a value that we (usually) do *not* want the output to exceed over this region. Naturally, since $f(x)$ is piecewise linear, (1.1) will contain subproblems that have only linear inequalities.

1.1.4 Linear Programming

The satisfiability problem for a system of linear inequalities, such as $Ax \geqslant b$ where $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, is most commonly solved using a method for linear programming (LP; also known as linear optimisation), which is in complexity class $\mathbf{P}$ – usually the simplex algorithm, but sometimes an interior-point method. This is done by transforming the satisfiability problem $\exists x : Ax \geqslant b$ into an equivalent linear optimisation problem (usually called a *linear program*). For a more general SMT formula using the theory of quantifier-free linear real arithmetic (QF_LRA), the satisfiability problem can be solved by using a SAT solver to suggest conjunctions of theory literals, and feeding the equivalent optimisation problems into an LP solver, using any negative results to guide the SAT solver.

Linear programming is a large and well-studied topic, but although academic analyses of the method have usually assumed exact arithmetic, solvers, including those arising from academia, have tended to provide no formal guarantees, due to the use of inexact (floating-point) arithmetic, and lack of any provision to rigorously detect incorrect solutions, or to improve upon a solution, no matter how close to optimal, found to be incorrect. The fact that the results have nevertheless usually proven to be adequate for academic (and indeed industrial) purposes is probably a consequence of the fact that solution quality in LP is mostly determined by finding the correct basis. Provided this can be found (which in most practical problems it can, even using fixed-precision floating-point arithmetic), the only error will arise from a single basis inversion and subsequent matrix-vector multiplication. However, without any guarantee of finding the correct basis, the error can in principle be arbitrary. Exact solvers have therefore found new interest with the development and popularisation of simplex solvers for SMT [6], a domain in which no error can be tolerated, although they have often tended to be very slow, due to the over-reliance on rational arithmetic.

1.1.5 Related Work

An example of a state-of-the-art SMT solver that uses the simplex algorithm with exact rational arithmetic for solving quantifier-free linear real arithmetic (QF_LRA) problems is Microsoft's z3 [7]. Another high-performance SMT solver is CVC4 [8], from Stanford University; while this solver uses exact rational simplex by default, it can be configured to accelerate the solving of QF_LRA conjunctions by initially attempting to solve them using a floating-point solver, and then using the result to initialise an exact rational solver—which can often save a significant amount of time.

QSopt-ex [9] is not an SMT solver, but it is notable in that it solves linear programs exactly by applying a floating-point solver many times with successively increasing precisions, using exact rational arithmetic only to derive a solution from the resulting basis and check its correctness. This approach scales better than falling back on a rational solver, because as problems get larger and more complex, the usefulness of the results found by any given fixed choice of floating-point precision diminishes, meaning that running time comes to be dominated by the rational solver. Unlike floating-point, rational arithmetic has a non-constant operation cost that typically grows throughout the course of the algorithm.

In [10], the QSopt-ex approach to exact LP is made more efficient using iterative refinement, and given a formal grounding with a proof that the algorithm completes in oracle-polynomial time under certain conditions. However, these conditions – specifically, that the "limited precision LP oracle" (floating-point LP solver) must always be able to "find a solution with residual errors bounded by a fixed constant" – can not easily be guaranteed in practice, as noted in that paper itself.

Another approach to the exact LP problem is that described in [11] and used in the software packages Lurupa and VSDP. This approach involves computing rigorous error bounds on the optimal objective function value. These solvers have the advantage of accepting interval input data, allowing them to model real-world uncertainty in the input. However, the principal problem with this approach is that it does not suggest any general, scalable way of improving a bad solution. It is stated in [11] that "nothing is assumed about the quality" of the "approximate optimal solution" required in order to derive the error bounds. However, clearly, the quality of the error bounds will be dependent on the quality of this optimal solution – otherwise, we could simply use a random guess! Indeed, the algorithms presented in that paper simply give up (returning infinite bounds) if the approximate solver fails to provide a usable solution.

1.2 Fundamental Preliminaries

These preliminaries are simple, but fundamental to many parts of this thesis, so they are introduced here before going any further.

1.2.1 Floating-point arithmetic model

Most modern floating-point systems (notably IEEE 754 and MPFR, but not GMP) are designed in such a way that for every binary operation $y := x_1 \text{ op } x_2$, the floating point result y is such that

$$\forall p \in \{1, -1\} : \quad \exists z \in \mathbb{Q} : \quad y = (1+z)^p (x_1 \text{ op } x_2), \quad |z| \leqslant \epsilon, \tag{1.2}$$

and ϵ is the *unit roundoff* (sometimes called *machine epsilon*), which is inversely exponential in the number of bits of precision. The value of $p \in \{1, -1\}$ may be chosen arbitrarily according to convenience.

In this thesis, I abbreviate this as $y := \mathbf{fl}_\epsilon(x_1 \text{ op } x_2)$ where the subscript ϵ indicates the unit roundoff, and hence the precision used. However, I extend this notation such that arbitrary linear algebra expressions are permitted within $\mathbf{fl}_\epsilon(\cdot)$, and it is to be understood that each scalar operation underlying the expression is to be carried out in that same precision, in an arbitrary order. Furthermore, where I have a matrix B that is not directly stored, but whose decomposition is stored, an expression such as $b := \mathbf{fl}_\epsilon(B^{-1}a)$ is to be understood as signifying the computation of b from a using the appropriate solving procedure(s), carried out entirely in the indicated precision.

1.2.2 Big O notation

In this thesis, I make frequent use of big O notation to express the rate of decay of a particular value as the unit roundoff, ϵ, goes to zero. That is, when I say that $f(\epsilon) = O(g(\epsilon))$ as $\epsilon \to 0$, I mean that there exist positive real numbers ϵ_0 and κ such that

$$\forall \epsilon \in [0, \epsilon_0] : |f(\epsilon)| \leqslant \kappa g(\epsilon). \tag{1.3}$$

In most cases, $g(\epsilon)$ will in fact be either 1, ϵ, or some power of ϵ.

Note that both f and g may have other parameters, such as the problem data, over which (1.3) can be considered to be universally quantified. Also, equations may be freely rearranged and big O terms combined, so that, for instance $f_1(\epsilon) + O(g_1(\epsilon)) = f_2(\epsilon) + O(g_2(\epsilon))$ is equivalent to $f_1(\epsilon) - f_2(\epsilon) = O(g_1(\epsilon) + g_2(\epsilon))$. Finally, in some cases, if it is likely that $f(\epsilon) \ll g(\epsilon)$ but this does not on its own provide a rigorous bound, I may write something of the form $f(\epsilon) \leqslant g(\epsilon) + O(h(\epsilon))$.

1.3 Overall Thesis Summary

This thesis builds upon the QSopt-ex approach, showing how satisfiability can be relaxed in favor of δ-satisfiability, proving the δ-completeness of the resulting algorithm, introducing my implementation of these ideas, and demonstrating the utility and effectiveness of the δ-complete algorithm on real-world problems. Given a small enough δ, my algorithm is guaranteed to find an exact solution to any linear program, and for larger values of δ, the running time and solution quality correlate inversely with δ.

To maximise the efficiency of the implementation, the simplex algorithm component was built on top of an inexact, floating-point solver. Rather than treating this inexact solver as a black box, and making assumptions about its properties, the precise algorithm used by the solver is investigated, and its properties are explored. For this purpose, it is found essential to make use of an inexact algorithm that uses a "partial pivoting" simplex method, as recommended by Bartels and Golub in 1969–1971 [12, 1]. This enables the errors produced by the algorithm to be bounded.

Although these bounds are usually not known (and in general are prohibitively expensive to compute), they are shown to be $O(\epsilon)$ for fixed problem data, in terms of the unit roundoff ϵ. Bland's rule is also used in order to guarantee termination once these errors are sufficiently small, and there is an iteration limit to prevent looping in case Bland's rule fails (a condition that is proven not to affect the correctness or completeness of the algorithm). (In real-world implementations, Bland's rule may be replaced by a more efficient heuristic rule, provided that Bland's rule or something equivalent is used as a fallback.)

It is hence shown that for any given linear feasibility problem, there exists some finite precision and some tolerance value for which the floating-point algorithm finds a *basis*, or subset of the columns of the input matrix, that enables an exactly correct solution (or an exact proof of infeasibility) to be derived. This is extended to all possible input problems by enclosing the floating-point solver in a loop that tries an increasing sequence of precisions, and a decreasing sequence of tolerance values.

1.3.1 Scope of Problem

We consider formulas of the form

$$\phi = t_1(x_1, \ldots, x_n) = 0 \wedge \ldots \wedge t_m(x_1, \ldots, x_n) = 0 \wedge x_1 \geqslant 0 \wedge \ldots \wedge x_n \geqslant 0, \tag{1.4}$$

where

$$t_i(x_1, \ldots, x_n) = A_{i1}x_1 + \ldots + A_{in}x_n - b_i, \quad 1 \leqslant i \leqslant m, \tag{1.5}$$

and all constants A_{ij} and b_i and variables x_j are in $\mathbb{Q}$.

The satisfiability problem for the formula ϕ may be written in standard linear algebra notation as

$$\exists x \in \mathbb{Q}^n : Ax = b,\ x \geqslant 0, \tag{1.6}$$

where $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$. Here, the comma implies $\wedge$ (and), the operators $=$ and $\geqslant$ are each to be understood as an elementwise conjunction over the elements of their vector operands, and 0 denotes a zero vector of the required size. The relationship between A, b and scalars A_{ij}, b_i from ϕ should be clear.

I have chosen this problem, rather than the simpler $Ax \geqslant b$, because it is more commonly used in the linear programming literature. Note that the simpler problem is equivalent to $Ax - t = b$, $t \geqslant 0$, which can be rewritten as $Ax_+ - Ax_- - t = b$, $x_+, x_-, t \geqslant 0$, and then, by renaming variables, as (1.6). All of these are considered linear programming problems, because they are generally solved using linear optimisation.

1.4 Delta-Complete Decision Procedures

We have now established enough context to introduce the concept of a delta-complete decision procedure, on which my thesis obviously depends a great deal. Here, I give only the definitions necessary for my analysis. Further definitions and theory are available in [3] and [4].

Definition 1.1. *If ϕ is a Boolean formula in which every atomic formula is of one the forms $t = 0$ or $t \geqslant 0$ where t is an arbitrary term, then the δ-weakening of ϕ, denoted $\phi^{-\delta}$, is, for any $\delta > 0$, the formula obtained by replacing every atomic formula $t = 0$ with $|t| \leqslant \delta$, and every atomic formula $t \geqslant 0$ with $t \geqslant -\delta$ (or, equivalently, $t + \delta \geqslant 0$).*

Clearly, if ϕ is defined as in (1.4), then $\phi^{-\delta}$ can be written as

$$\phi^{-\delta} = |t_1(x_1, \ldots, x_n)| \leqslant \delta \wedge \ldots \wedge |t_m(x_1, \ldots, x_n)| \leqslant \delta \wedge x_1 \geqslant -\delta \wedge \ldots \wedge x_n \geqslant -\delta, \quad (1.7)$$

and its satisfiability problem can be written in linear algebra notation as

$$\exists x \in \mathbb{Q}^n : |Ax - b| \leqslant \delta \mathbb{1}_m,\ x \geqslant -\delta \mathbb{1}_n, \quad (1.8)$$

or, more straightforwardly, but also more verbosely,

$$\exists x \in \mathbb{Q}^n : Ax \geqslant b - \delta \mathbb{1}_m,\ Ax \leqslant b + \delta \mathbb{1}_m,\ x \geqslant -\delta \mathbb{1}_n, \quad (1.9)$$

where $\mathbb{1}_\sigma = [1, \ldots, 1]^\top$ denotes a vector of ones of size σ.

Definition 1.2. *A δ-complete decision procedure for satisfiability over a class of formulas $\mathcal{F}$ is an algorithm that, if provided as input any formula $\phi \in \mathcal{F}$, terminates and outputs (non-deterministically) exactly one of the following symbols:*

1. unsat *only if ϕ is unsatisfiable*

2. δ-sat *only if $\phi^{-\delta}$ (the δ-weakening of ϕ) is satisfiable*

A δ-complete decision procedure for satisfiability over (1.4) must therefore be able to provide either a negative answer to (1.6), or a positive answer to (1.8). Note that this is non-deterministic, because both conditions can simultaneously be true.

1.4.1 Difference between the Nonlinear and Linear Cases

δ-complete (or delta-complete) decision procedures have also been used for non-linear real arithmetic problems augmented with arbitrary continuous real functions (such as the sine function). These approaches tend to involve interval arithmetic. This can handle arbitrary functions, but often suffers from issues with unchecked growth of the overapproximation error (which can be avoided in the linear case, as shown in this thesis) [3, 4, 13, 14]. The linear case also permits methods that are much more efficient than interval arithmetic.

Also note that any complete method is, by definition, trivially delta-complete. In that sense, delta-completeness of a linear problem is unavoidable when using e.g. a rational simplex solver (because completeness is then unavoidable). The challenge in the linear case is to find a method that is delta-complete without being complete, in a way that provides a meaningful advantage over the complete method.

For general non-linear real arithmetic (NRA) problems augmented with arbitrary continuous real functions (such as the sine function), a complete solver is impossible, because the result is uncomputable [15, p. 45]. For linear real arithmetic (with no extensions), a complete solver is trivial (e.g. using rational arithmetic), but can take an unacceptably long time for extremely large problems. There are of course optimisations that can offer an improvement over the simplex algorithm with pure rational arithmetic, without sacrificing the exactness of the solution (e.g. `SoPlex` [16, 17], `QSopt_ex` [9, 18]). However, if a problem is still too large to solve to completion (for a given application), even with these optimisations, then it may be helpful to have a delta-complete solver.

1.5 Aim and Objectives

The aim of this work is *to devise novel, numerically rigorous methods of solving linear satisfiability (SMT) and optimisation problems partially, but with delta-completeness guarantees.*

To that end, the following objectives were identified:

- investigate numerically rigorous techniques for solving linear programming problems for both SMT and optimisation.
- develop algorithms that relax the completeness of these algorithms in favour of delta-completeness (for SMT) or some equivalent concept (for optimisation), providing results that are still useful, but with a time saving when full completeness is not required.
- implement the developed algorithms in software, and apply them to the many available linear SMT and optimisation benchmark problems, comparing their performance against the performance of the equivalent complete algorithms.

1.6 Thesis Outline

- **Chapter 2** presents a version of the full-Newton step algorithm for interior-point linear optimisation, restricted to solving satisfiability problems, that is delta-complete for those problems. The proof of delta-completeness of that algorithm is also presented.
- **Chapter 3** presents an in-depth analysis of the guarantees provided by the Bartels–Golub simplex algorithm, with several important updates relative to Bartels' own analysis, concluding with a set of bounds that will be important in Chapter 4.

- **Chapter 4** presents a version of the simplex algorithm, restricted to solving satisfiability problems, that is delta-complete for those problems. The proof of delta-completeness of that algorithm, making use of the results of Chapter 3, is also presented.

- **Chapter 5** introduces a new concept of delta-completeness applicable to the general linear programming problem for optimisation. An algorithm conforming to this concept is presented, and proven to be delta-complete in accordance with this new concept.

- **Chapter 6** describes the developed software tool `dLinear`, in which the algorithms presented in Chapters 4 and 5 are implemented. An experimental implementation of the algorithm of Chapter 2 (`dcipm_simple`) is also described.

- **Chapter 7** demonstrates the performance of `dLinear` on a wide variety of problems drawn from the LP and SMT communities. This is compared against that of theorem provers `Z3` and `CVC`, and exact linear programming solvers `SoPlex` and `QSopt_ex`.

- **Chapter 8** contains concluding remarks and directions for further work.

Chapter 2

δ-Complete Interior-Point Method for Satisfiability

This chapter presents a version of the full-Newton step algorithm for interior-point linear optimisation, restricted to solving satisfiability problems, that is delta-complete for those problems. A proof of delta-completeness of the algorithm is also presented.

Many useful results on interior-point methods from [2] have been collected into Appendix A. In this chapter, when a result from [2] is required, the text will refer primarily to the appropriate parts of Appendix A. Citations for the corresponding parts of [2] can be found in Appendix A.

2.1 Interior-Point System

First we must embed (1.6) into an equisatisfiable inequality-form problem. This can be achieved by rewriting $Ax = b$ as $Ax \geqslant b$, $Ax \leqslant b$. Note that the δ-weakening of this form, using the original definition, is equivalent to (1.8), which is the δ-weakening of (1.6). This is important because it means that a δ-complete algorithm for the new problem will also be a δ-complete algorithm for (1.6). By redefining A and b, we can then rewrite the problem as:

$$\exists x \in \mathbb{Q}^n : Ax \geqslant b,\ x \geqslant 0, \tag{2.1}$$

and the δ-weakening as:

$$\exists x \in \mathbb{Q}^n : Ax \geqslant b - \delta\mathbb{1},\ x \geqslant -\delta\mathbb{1}. \tag{2.2}$$

Next, we reformulate this as a linear optimisation problem. As we have no objective function to optimise, but the method nonetheless calls for one, we simply use the constant function $0^\top x$. This gives:

$$\min\{0^\top x : Ax \geqslant b, x \geqslant 0\}. \tag{2.3}$$

That is, we want to find the minimum value (and, by convention, the corresponding argument x) contained in the set consisting of $0^\top x$ for every x such that $Ax \geqslant b$ and $x \geqslant 0$. Clearly, unless the conditions $Ax \geqslant b$, $x \geqslant 0$ are unsatisfiable (in which case the set is empty), this set is equal to $\{0\}$, and the minimum value is 0. However, the problem (2.3), when given to an interior-point algorithm (which is iterative and convergent), causes the iterates produced by the solver to converge in a way that is useful to us.

Every linear optimisation (LO) problem has a corresponding "dual" problem that is also an LO problem. The dual of (2.3) is:

$$\max\left\{b^\top y : A^\top y \leqslant 0, y \geqslant 0\right\}. \tag{2.4}$$

In this context, (2.3) is referred to as the primal problem. By Lemma R-I.1 (see Appendix A.1), we have $b^\top y \leqslant 0^\top x$ for all primal-feasible x and dual-feasible y (where "feasible" means that it satisfies the relevant set-membership conditions). In other words, $b^\top y \leqslant 0$ for all dual-feasible y. Furthermore, by Theorem R-I.26 (Appendix A.1), (2.4) has a finite solution y^* if and only if (2.3) has a finite solution x^*, in which case $b^\top y^* = 0^\top x^*$. That is, for any finite solution y^* of (2.4), $b^\top y^* = 0$.

In order to use (2.3) with an interior-point algorithm, we must first transform it into a form that has the following properties:

1. It must be feasible (i.e. the set that we are optimising over must be non-empty), even if (2.3) is not.

2. Infeasibility of (2.3) or (2.4) must be indicated in some way by the result.

3. It must be equal to its own dual problem ("self-dual"). (This is not a hard requirement, but it simplifies the analysis, and does not sacrifice generality.)

    (a) In order for the problem to be self-dual, its matrix must be skew-symmetric. This is also a useful property in and of itself.

4. It must satisfy the interior-point condition (IPC), which stipulates that there must be some feasible vector that strictly satisfies all inequalities in the feasibility conditions.

The optimality conditions for (2.3) and (2.4) are (see (R-2.5) in Appendix A.2):

$$\begin{aligned} Ax &\geqslant b, & x &\geqslant 0, \\ -A^\top y &\geqslant 0, & y &\geqslant 0, \\ b^\top y &\geqslant 0. \end{aligned} \tag{2.5}$$

The condition $b^\top y \geqslant 0$, when combined with the condition $b^\top y \leqslant 0$, which, as we have already noted, is satisfied by any dual-feasible y, gives us the dual optimality condition $b^\top y = 0$. Meanwhile, clearly, any primal-feasible x is also primal-optimal.

As in Appendix A.3 (and [2, p. 20]), we then embed this into the homogeneous skew-symmetric inequality system, which is equisatisfiable:

$$\bar{M}\bar{z} = \begin{bmatrix} 0_{m\times m} & A & -b \\ -A^\top & 0_{n\times n} & 0_n \\ b^\top & 0_n^\top & 0 \end{bmatrix} \begin{bmatrix} y \\ x \\ \kappa \end{bmatrix} \geqslant \begin{bmatrix} 0_m \\ 0_n \\ 0 \end{bmatrix},\ x \geqslant 0,\ y \geqslant 0,\ \kappa > 0. \tag{2.6}$$

Theorem 2.1. *Systems* (2.5) *and* (2.6) *are equisatisfiable.*

Proof. Apply Theorem R-I.5 (Appendix A.4). Refer to (R-2.5), (R-2.7) and (R-2.8) (Appendices A.2 and A.3), and consider the case where $c = 0$. Note that (2.6) incorporates $\kappa > 0$. □

We then embed (2.6) into the IPC (interior-point condition) system (the same as (R-2.16) in Appendix A.6):

$$\min\{q^\top z : Mz \geqslant -q, z \geqslant 0\}, \tag{2.7}$$

where

$$M = \begin{bmatrix} \bar{M} & r \\ -r^\top & 0 \end{bmatrix},\ z = \begin{bmatrix} \bar{z} \\ \vartheta \end{bmatrix},\ q = \begin{bmatrix} 0_{n-1} \\ n \end{bmatrix},\ r = e - \bar{M}e, \tag{2.8}$$

where $M \in \mathbb{Q}^{n\times n}$, etc.

Theorem 2.2. *From any optimal solution to* (2.7) *such that* $\kappa > 0$*, we can derive a solution to* (2.6)*, and vice-versa.*

Proof. Apply Theorem R-I.5 (Appendix A.4). □

Note that e is feasible for (2.7), with slack $Me + q = e$ (using the identity $e^\top \bar{M}e = 0$, which is a consequence of the skew-symmetry of $\bar{M}$). This satisfies the most important precondition of the interior-point algorithm (existence of a positive feasible vector with positive slack; known as the interior-point condition or IPC).

2.1.1 The unsatisfiable case

Definition 2.3. *A strictly complementary solution z to* (2.7) *is one in which $Zs(z) = 0$ (or, equivalently, $zs(z) = 0$) and $z + s(z) > 0$, where $Z = \operatorname{diag}(z)$ and $s(z) = Mz + q$.*

Note that the condition $Zs(z) = 0$ where $Z = \operatorname{diag}(z)$ can be restated as $zs(z) = 0$, if $zs(z)$ denotes the Hadamard or elementwise product of the column vectors z and $s(z)$.

Lemma 2.4. *Any optimal solution to* (2.7) *is complementary – i.e. it has the "complementarity" or complementary slackness property $zs(z) = 0$ where $s(z) = Mz + q$.*

Proof. As $z = 0$ is feasible for (2.7) and $q \geqslant 0$, we know that the optimal value is 0, so that any optimal solution will have $q^\top z = 0$. From equation (R-2.23) (Appendix A.5), we have $q^\top z = z^\top s(z)$. Now, from (2.7) we clearly have $z \geqslant 0$ and $s(z) \geqslant 0$. So $z^\top s(z) = 0$ implies $zs(z) = 0$. □

Theorem 2.5. *The problem (2.7) has a strictly complementary optimal solution.*

Proof. Apply Theorem R-I.20 (Appendix A.7). □

Definition 2.6 (see Appendix A.7)**.** *The optimal partition of (2.7) is defined as*

$$\begin{aligned} B &:= \{i : \exists z : z_i > 0, \text{ and } z \text{ is optimal for } (2.7)\}, \\ N &:= \{i : \exists z : s_i(z) > 0, \text{ and } z \text{ is optimal for } (2.7)\}. \end{aligned}$$

Theorem 2.7. *If (B, N) is the optimal partition of (2.7), then for every index $i \in 1, \ldots, n$, either $i \in B$ and $i \notin N$, or $i \in N$ and $i \notin B$ – that is, (B, N) is a partition of the index set of a solution vector z to (2.7).*

Proof. This follows from combining Theorem 2.5 with Corollary R-I.11 (Appendix A.7) (which establishes that B and N are disjoint). □

Lemma 2.8. *For every strictly complementary optimal solution z of (2.7),*

$$\begin{aligned} z_i > 0,\ s_i(z) = 0, \text{ where } i \in B, \text{ and} \\ z_i = 0,\ s_i(z) > 0, \text{ where } i \in N. \end{aligned}$$

where (B, N) is the optimal partition of (2.7).

Proof. Optimality of the solution requires that either $z_i = 0$ or $s_i(z) = 0$ for all i in the index set of z and $s(z)$, so that we can have $z^\top s(z) = 0$ (along with feasibility). Strict complementarity then requires, for each i, either $z_i > 0$ or $s_i(z) > 0$, so that we can have $z_i + s_i(z) > 0$. We must therefore have $z_i > 0$ if $i \in B$ and $s_i(z) > 0$ if $i \in N$, as the opposite would contradict Definition 2.6. Theorem 2.7 ensures that this works correctly for all i. □

Theorem 2.9. *κ is zero in the optimal partition of (2.7) if and only if (2.6) is unsatisfiable.*

Proof. By Lemma R-I.10 (Appendix A.6), if κ is zero in the optimal partition, then it is zero in every optimal solution of (2.7). However, if (2.6) is satisfiable, then by Theorem 2.2 there is an optimal solution to (2.7) such that $\kappa > 0$.

For the converse, observe that if $\kappa > 0$ in the optimal partition, then by Theorems 2.5 and 2.2, (2.6) is satisfiable. Therefore, by modus tollens, if (2.6) is unsatisfiable, κ is zero in the optimal partition. □

2.2 Full-Newton Step Algorithm

System (2.7) can be partially solved using the "full-Newton step algorithm" (Algorithm A.1 in Appendix A.8). This algorithm terminates after at most $\lceil\theta^{-1}\log(n\epsilon^{-1})\rceil$ iterations, where θ is the *barrier update parameter*[1] and ϵ is the *accuracy parameter* (Lemma R-I.36, Appendix A.9). The returned partial solution satisfies $z > 0$ (Theorem R-I.16, Appendix A.10), and is feasible for (2.7), but clearly not optimal (because $\vartheta > 0$).

Algorithm 2.1 on p. 17 is essentially the same algorithm, but with the addition of lines 2–3 and 10–15.

The assignment of the slack vector s, which is left implicit in Algorithm A.1, is performed explicitly on lines 4 and 9 of Algorithm 2.1. Similarly, the computation of the Newton direction Δz is now performed explicitly on line 7 (note that $Z = \operatorname{diag}(z)$ and $S = \operatorname{diag}(s)$). These are the same computations as those implied in Algorithm A.1. And the input matrix M is analogous, and similar in form, to that used implicitly in Algorithm A.1, as is the output primal solution z.

The only significant differences, then, are in the input and output precision δ, the output result symbol (and the manner in which it and the output δ are determined), the specific form of the parameters ϵ, θ (which in Algorithm A.1 are only given bounds), and the additional termination condition on line 10. Or, in summary, the precision δ, lines 2–3, and lines 10–15.

The need to input a precision δ is obvious: this is how the algorithm knows when to terminate early on lines 10–11, giving a result of δ-sat, as proven to be correct in Theorem 2.10. The particular choice of the ϵ parameter on line 2 is made in order to ensure that the main loop will not otherwise terminate until the iterates are such that the optimal partition can be found, meaning that the solution can be decided exactly. This is implemented in lines 12–15, and made use of in Theorem 2.10. Finally, the choice of θ on line 3 enables us to put a bound on the number of iterations required, as in Theorem R-I.37.

2.3 Proof of δ-Completeness

Theorem 2.10. *Algorithm 2.1 satisfies Definition 1.2: it is a δ-complete decision procedure for satisfiability over* (1.4).

Proof. First, we note that the algorithm terminates. This is because at each iteration of the loop starting on line 5, μ is decreased by a factor of $(1-\theta)$, which will eventually lead to the condition $n\mu < \epsilon$. This means that one of the return statements on lines 11, 13 and 15 will eventually execute.

For any partial solution $z = (x, y, \kappa, \vartheta)$ returned by Algorithm 2.1 (p. 17), including, in particular, that returned on line 11, the strict feasibility of z for (2.7) implies (see also

[1]This is *not* the same as ϑ, which is the so-called *normalising variable* of system (2.7).

system (R-4.16) (Appendix A.14)):

$$A\left(\frac{x}{\kappa}\right) \geqslant b - \frac{\vartheta}{\kappa}(\bar{b}) \geqslant b - \frac{\vartheta}{\kappa}\|\bar{b}\|_\infty e, \qquad \frac{x}{\kappa} \geqslant 0, \tag{2.9}$$

where $\bar{b} = b + e - Ae$ (and obviously $\kappa > 0$). Using (2.2), we see that the solution proves that the problem is δ-satisfiable for $\delta = \frac{\vartheta}{\kappa}\|\bar{b}\|_\infty$. As $\frac{\vartheta}{\kappa}\|\bar{b}\|_\infty$ is less than or equal to the input δ (due to line 10), we indeed have a valid assertion of δ-*sat* on line 11.

Otherwise, if line 12 is reached, we know that z is an ϵ-solution with ϵ as given on line 2. According to Theorem R-I.42 (Appendix A.11), the condition number σ_{SP} of M satisfies $1/(\prod_{j=1}^{n}\|M_j\|_2) \leqslant \sigma_{SP}$. This means that we have $\epsilon \leqslant \sigma_{SP}^2/(4n)$. As part of the proof of Theorem R-I.47 (Appendix A.12), it is shown that an ϵ-solution with this ϵ (or smaller) is sufficient to ensure complete separation between small variables and large variables, meaning that the condition on line 12 is sufficient to decide the satisfiability problem. This provides either a valid assertion of *unsat*, or a valid assertion of δ-*sat* with $\delta = 0$, which is equivalent to the statement that the original problem is satisfiable. □

Note that if (2.1) is satisfiable, then $\lim_{\mu\downarrow 0}\frac{\vartheta}{\kappa} = 0$ (Theorem R-I.58, Appendix A.13). This makes it likely that the algorithm will terminate on line 11 where possible.

If δ-satisfiability (or satisfiability) is concluded, then the result can be trivially verified by computing $A(x/\kappa)$. If unsatisfiability (or satisfiability) is concluded, then, using the returned $z = (x, y, \kappa, \vartheta) > 0$, we can verify the result by checking that z is primal feasible for (2.7), that $\max(zs)/\min(zs) \leqslant 4$, that $\vartheta \leqslant 1/(4n^2(\prod_{j=1}^{n}\|M_j\|_2)^2)$ (note: n^2 here rather than n), and that we have the appropriate relationship between κ and s_κ where $s_\kappa = s_{n-1}$ and $s := Mz + q$ with q defined as in (2.8).

Algorithm 2.1: δ-complete full-Newton step algorithm (derived from Algorithm A.1 (Appendix A.8)).

```
input  : Problem matrix $M \in \mathbb{Q}^{n\times n}$ as in (2.8) and (2.6), decomposable into
         $(A, b, r)$;
         Target precision $\delta \in \mathbb{Q}$.
output: Result symbol $\in \{\delta\text{-sat}, \text{unsat}\}$;
         Primal solution $z \in \mathbb{Q}^n$, decomposable into $(x, y, \kappa, \vartheta)$;
             Homogenising variable $\kappa \in \mathbb{Q}$;
             Normalising variable $\vartheta \in \mathbb{Q}$;
         Actual precision $\delta \in \mathbb{Q}$ (where applicable).
1  begin
2      $\epsilon := 1/(4n(\prod_{j=1}^{n} \|M_j\|_2)^2)$ where $M_j$ is column $j$ of $M$ ;  // Accuracy parameter $\epsilon$
3      $\theta := 1/\sqrt(2n)$ ;                                  // Barrier update parameter $\theta$
4      $z := e$; $s := e$; $\mu := 1$ ;                  // Initial point: the $\mu$-center for $\mu = 1$
5      while $n\mu \geqslant \epsilon$ do
6          $\mu := (1-\theta)\mu$ ;                       // Update barrier parameter $\mu$ using $\theta$
7          $\Delta z := (S + ZM)^{-1}(\mu e - zs)$ ;                 // Compute Newton direction
8          $z := z + \Delta z$ ;                                      // Perform full Newton step
9          $s := s + M\Delta z$;
10         if $\frac{\vartheta}{\kappa}\|\bar{b}\|_\infty \leqslant \delta$ where $\bar{b} = b + e - Ae$ then
11             return $\delta$-sat$(z,\ \delta := \frac{\vartheta}{\kappa}\|\bar{b}\|_\infty)$ ;     // Early termination - $\delta$-sat proven
12     if $\kappa > s_\kappa$ then
13         return $\delta$-sat$(z,\ \delta := 0)$ ;              // Satisfiability proven rigorously
14     else
15         return unsat$(z)$ ;                           // Unsatisfiability proven rigorously
```

Chapter 3

The Bartels–Golub Simplex Algorithm

Critical to the δ-complete simplex algorithm in Chapter 4 is a floating-point version of the simplex algorithm with bounded errors. This property of having bounded errors is sometimes called "stability". A naive floating-point implementation of simplex has unbounded errors, because of potential division by arbitrarily small values during Gaussian elimination.

The Bartels–Golub algorithm [12, 1], even after almost 50 years, is still regarded as an important milestone in the development of simplex method implementations. One of its successors, the Forrest–Tomlin algorithm [19], is widely used in modern simplex implementations. However, it has been noted that Forrest–Tomlin lacks the stability of Bartels–Golub. In particular, it omits the pivoting step which is the key to the Bartels–Golub algorithm's stability [20]. This means, simply put, that there is no a-priori bound on the relative error of the Forrest–Tomlin algorithm applied to an arbitrary LP. By contrast, Bartels–Golub provides this a-priori bound. Other authors have suggested improvements to Bartels–Golub that do not sacrifice this property. However, the Bartels–Golub algorithm itself remains a key reference point for stable simplex implementations.

An error analysis of the Bartels–Golub algorithm was published by Bartels [1]. Unfortunately, there are a number of issues with that paper that impede comprehension and progress. For one thing, because of its age, the error analysis contains complications that were necessitated by obsolete hardware, and fails to make use of important results and conventions that have simplified the field of error analysis since its publication. For another, it appears to contain a number of actual errors, and at least one unnecessarily complex bound, even taking into account the conventions of its day.

This thesis is not the right place for a full rewrite of Bartels' analysis. However, in this chapter, I provide an overview of the changes to [1] that I believe are necessary (or desirable), including the key results that are used in this thesis (given as big-O bounds).

In this chapter, I make frequent references to specific equations in [1]. To avoid excessive clutter, I do not explicitly reference [1] each time. Instead, I change the reference

(X.XX) from [1] by prefixing with B-, giving (B-X.XX). For example, (5.14) from [1] becomes (B-5.14). This makes them easy to distinguish from equation references internal to this thesis, which are written simply as, for instance, (3.10).

3.1 Introduction to the Bartels–Golub Algorithm

The Bartels–Golub algorithm is an algorithm for solving linear programming problems in $O(n^2)$ time with bounded error. As a simplex-based method, it is based on performing repeated *pivots* (changes of basis) until a basis is found in which all basic (non-derived) variables may be set to their bounds to yield an optimal solution. This is possible because the solution to an LP is always at a corner point; selecting variables to join the basis is akin to selecting a set of planes from the set comprising a polyhedron (such that the intersection of all selected planes is a corner point of the polyhedron).

This can seem, intuitively, like a trivial problem (and an exact $O(n^3)$ implementation is indeed trivial), except that problems typically have thousands if not millions of dimensions, making the performance of an exact $O(n^3)$ algorithm unacceptable. The wide range of applications, and pivotal role that this algorithm can play (no pun intended), justify intensive study of techniques for its optimisation.

Solving in $O(n^2)$ requires, first of all, for the current set of variable assignments to be stored and updated *implicitly*, by storing and updating a triangular decomposition of the basis matrix that can be used to derive the non-basic variables (and explicitly storing the values of basic variables only). Secondly, for $O(n^2)$ performance, the basis matrix decomposition must be updated by minimal modifications to its predecessor (from prior to the pivot). And of course $O(n^2)$ performance is meaningless if arithmetic operations (normally considered to be constant-time) become unacceptably slow, so floating-point (inexact) arithmetic must be used. If this is not done carefully, it leads to the rapid accumulation of errors in the basis matrix representation, which may cause the algorithm to fail to identify a correct basis. The Bartels–Golub method mitigates this problem by putting strict bounds on the accumulation of error at each pivot (but does not, in itself, guarantee finding a correct basis).

In Richard Bartels' 1971 paper [1], which provides the main source of detailed information about the original Bartels–Golub method, it is called *the Hessenberg-LU Implementation of the Simplex Method*. This is because a key feature of this method is that the triangular ("LU") decomposition of the basis matrix is updated via the temporary transformation of the upper-triangular matrix U to a Hessenberg matrix (that is to say, a matrix that is *almost* upper-triangular, with zeros below the first subdiagonal (rather than the main diagonal)).

More specifically, [1, p. 421] uses

$$\Pi B = LU \tag{B-5.1}$$

as the decomposition of the basis matrix B, where L is lower-triangular, U is upper-triangular, and Π is a permutation matrix (enabling the rows of B to be rearranged). It is also explained there that a linear system of the form $Bv = q$ may be solved (for v) via $Lt = \Pi q$ and $Uv = t$ (using trivial substitution due to the triangular structures of L and U).

On [1, p. 416], an algorithm, (B-2.5), is given for performing a simplex pivot. It is clear that in this algorithm, all of the computationally intensive steps are of the form $Bv = q$ or $B^\top v = q$ (which is also easy to solve given a triangular decomposition). This algorithm is given as the key component of a solution to the following problem ([1, p. 415]):

$$\begin{aligned}&\text{maximize the } \textit{objective function}\text{: } z = c^\top x,\\ &\text{subject to the } \textit{constraints}\text{: } Ax = b \text{ and } x \geqslant 0.\end{aligned} \tag{B-2.1}$$

Note that the *constraints* are the same as the conditions of my overall *satisfiability problem* (1.6) in Chapter 1. As noted there, conditions of the form $Ax \geqslant b$ may easily be converted into this form, and even a problem (such as (1.6)) with no objective function must be solved using linear optimisation – essentially, you get the objective function "for free" (more accurately, satisfying the constraints requires the introduction of a temporary objective function).

3.1.1 Further concepts and notation for the Bartels–Golub algorithm

All of the equations in this subsection are copied from [1].

This (from [1, p. 421]) is simply a decomposition of the basis matrix B into its columns, each of which is a column of the original constraint matrix A from (B-2.1). Notice that given A, the index vector $\nu \in \mathbb{N}^m$ (using the Greek letter *nu*) completely specifies the basis.

$$B = [A_{\nu_1}, \ldots, A_{\nu_m}]. \tag{B-5.4}$$

An odd feature of [1] is that ν, which maps from column indices of B onto column indices of A, appears to be local in scope: when it says on p. 422 that $B^{(2)}$ *is to be formed from* $B^{(1)}$ *in the same manner in which* $B^{(1)}$ *was formed from* B, this implies that (B-5.4) is to be applied to $B^{(1)}$ as it previously was to B, effectively redefining ν. Any other interpretation results in difficulties, because if ν is invariant, we must then convert between the "original basis" indices used to index ν and the "current basis" indices used to index U (or $U^{(1)}$, etc., in subsequent pivots).

This (also from [1, p. 421]) shows how the second basis matrix, after the first pivot, is simply the result of removing column ν_{r_1} of A from column r_1 of B, and appending column ν_{s_1} of A.

$$B^{(1)} = [A_{\nu_1}, \ldots, A_{\nu_{r_1-1}}, A_{\nu_{r_1+1}}, \ldots, A_{\nu_m}, A_{s_1}]. \tag{B-5.6}$$

A superscript with parentheses is used throughout [1] to indicate the pivot number (except on r and s, where plain subscripts are used).

The following equation (from [1, p. 422]) shows the result of applying the above "pivot" operation via a modification to the matrix U from the triangular decomposition $\Pi B = LU$ (as in (B-5.1)), yielding a new triangular-Hessenberg decomposition $\Pi B^{(1)} = LH^{(1)}$.

$$\begin{aligned} L^{-1}\Pi B^{(1)} &= [U_1, \dots, U_{r_1-1}, U_{r_1+1}, \dots, U_m, L^{-1}\Pi A_{s_1}] \\ &= H^{(1)}. \end{aligned} \tag{B-5.7}$$

Notice that the nature of matrix multiplication (along with the fact that Π simply permutes the rows of B) means that the rearrangement of the columns of B can be expressed simply by the same rearrangement of the columns of U. The appended column is also very straightforward, because we can use the fact that a left-applied matrix (here, $L^{-1}\Pi$) modifies the columns of its matrix operand independently. The result is assigned the symbol $H^{(1)}$, because this is the Hessenberg matrix resulting from the first pivot.

The following equation (also from [1, p. 422]) expresses the re-triangularisation of the upper Hessenberg matrix $H^{(1)}$ to give a new upper-triangular matrix $U^{(1)}$. The $\Pi_j^{(1)}$ are trivial permutation matrices, while the $\Gamma_j^{(1)}$ are trivial lower-triangular matrices. The index j ranges from r_1 to $m-1$ because r_1 is the column of B that was removed, and $m-1$ is the penultimate column of B (column m needs no adjustment because it is entirely in the upper triangular part of the matrix).

$$U^{(1)} = \Gamma_{m-1}^{(1)}\Pi_{m-1}^{(1)} \dots \Gamma_{r_1}^{(1)}\Pi_{r_1}^{(1)} H^{(1)}. \tag{B-5.9}$$

This (also from [1, p. 422]) simply gives the name $C^{(1)}$ to the inverse of the operation applied above to $H^{(1)}$ to give $U^{(1)}$. In other words, $C^{(1)}U^{(1)} = H^{(1)}$. However, it also generalises this to pivot i. Note that the permutation matrices $\Pi_j^{(i)}$ are self-inverse (because they only perform a single swap).

$$C^{(i)} = \Pi_{r_i}^{(i)}\Gamma_{r_i}^{(i)^{-1}} \dots \Pi_{m-1}^{(i)}\Gamma_{m-1}^{(i)^{-1}}. \tag{B-5.12}$$

It is not explicitly stated in [1], but once we have the decomposition $\Pi B^{(1)} = LC^{(1)}U^{(1)}$, we can then pivot $B^{(1)}$ via similar manipulation of $U^{(1)}$ to that shown in (B-5.7), yielding $\Pi B^{(2)} = LC^{(1)}H^{(2)}$ (note that $C^{(1)}$ remains intact, alongside L). We may then again apply an operation similar to (B-5.9), yielding $C^{(2)}U^{(2)} = H^{(2)}$, as above. Indeed, this can be generalised to give $C^{(k)}U^{(k)} = H^{(k)}$ for any pivot k.

Ultimately, at iteration k, we will have $\Pi B^{(k)} = LC^{(1)}C^{(2)} \dots C^{(k)}U^{(k)}$, which we may abbreviate to $\Pi B^{(k)} = LG^{(k)}U^{(k)}$, using:

$$G^{(k)} = C^{(1)} \dots C^{(k)}. \tag{B-5.13}$$

3.1.2 The basics of the error analysis

At this point, we move into chapter 6 of [1]. This chapter is concerned with error analysis, so error terms, prefixed by δ, are now introduced into the equations from chapter 5 of [1] (and our Section 3.1.1). Consideration of the initial row permutation matrix Π is also omitted (as it has no bearing on the analysis), meaning that our $\Pi B^{(k)} = LG^{(k)}U^{(k)}$ becomes:

$$B^{(k)} + \delta B^{(k)} = LG^{(k)}U^{(k)}, \tag{B-6.3}$$

which defines $\delta B^{(k)}$ simply as the difference between the value of $B^{(k)}$ and the true value of $LG^{(k)}U^{(k)}$, as if computed exactly but using the inexact stored values of L, $G^{(k)}$ and $U^{(k)}$. (Note that it would not make sense to consider error introduced by multiplying these three matrices, as that computation is never actually performed.)

Conceptually, $\delta B^{(k)}$ is the answer to the question "even if we were to solve *exactly* using L, $G^{(k)}$ and $U^{(k)}$, it would not be the same as solving *exactly* using $B^{(k)}$ – so how can we model the difference?". The answer is that it would indeed be the same as solving exactly using $B^{(k)} + \delta B^{(k)}$.

Solving $B^{(k)}v = q$ for v (or, more accurately, solving $(B^{(k)} + \delta B^{(k)})v = q$ for v) is performed via each component of the decomposition of $B^{(k)} + \delta B^{(k)}$ given above – first L, then $G^{(k)}$, then $U^{(k)}$, each introducing a new error term, as made explicit in the following three equations:

$$(L + \delta L)t = q, \tag{B-6.4}$$
$$(G^{(k)} + \delta G^{(k)})w = t, \tag{B-6.5}$$
$$(U^{(k)} + \delta U^{(k)})v = w. \tag{B-6.6}$$

Substitution and expansion of (B-6.4), (B-6.5) and (B-6.6), followed by substitution of (B-6.3) to eliminate the term $LG^{(k)}U^{(k)}$, and then collection of all matrix terms apart from $B^{(k)}$ into a single combined error term $\mathcal{E}^{(k)}$, yields:

$$(B^{(k)} + \mathcal{E}^{(k)})v = q, \tag{B-6.7}$$

where:

$$\begin{aligned}\mathcal{E}^{(k)} = \delta B^{(k)} + LG^{(k)}\delta U^{(k)} + L\delta G^{(k)}U^{(k)} + \delta LG^{(k)}U^{(k)} + L\delta G^{(k)}\delta U^{(k)}\\ + \delta LG^{(k)}\delta U^{(k)} + \delta L\delta G^{(k)}U^{(k)} + \delta L\delta G^{(k)}\delta U^{(k)}.\end{aligned} \tag{B-6.8}$$

3.1.3 Error analysis of the retriangularisation factors

Solving $(G^{(k)} + \delta G^{(k)})w = t$ is not done in a single step, because $G^{(k)}$ is not stored directly – in fact, not even the $C^{(1)} \ldots C^{(k)}$ of which $G^{(k)}$ is composed are stored directly – instead, we must look at both (B-5.13) and (B-5.12) to find the lower-triangular factors ${\Gamma_j^{(i)}}^{-1}$, and permutation matrices $\Pi_j^{(i)}$, which *are* stored and used directly.

In [1], it is shown that $\delta G^{(k)}$ may be modeled as:

$$\delta G^{(k)} = \tilde{C}^{(1)} \dots \tilde{C}^{(k)} - C^{(1)} \dots C^{(k)}, \tag{B-6.27}$$

(noting that $C^{(1)} \dots C^{(k)}$ is simply the expansion of $G^{(k)}$), where

$$\tilde{C}^{(i)} = \Pi^{(i)}_{r_i}[\Gamma^{(i)^{-1}}_{r_i} - \delta\Gamma^{(i)^{-1}}_{r_i}] \dots \Pi^{(i)}_{m-1}[\Gamma^{(i)^{-1}}_{m-1} + \delta\Gamma^{(i)^{-1}}_{m-1}]. \tag{B-6.28}$$

Note that there is no need to introduce error terms for the $\Pi^{(i)}_j$ factors, as they are permutation matrices – in other words, they are themselves simply permutations of the identity matrix, and as such, solving with them will be exact. As such, all of the error implicit in $\delta G^{(k)}$ comes from solving with the $\Gamma^{(i)^{-1}}_j$ factors. Also note that the alternation of $-$ and $+$ operators in (B-6.28) may safely be ignored: the error terms are used for their magnitude only.

It is then shown that the permutation factors, being self-inverse, can be rearranged in a useful way, giving new expansions for $C^{(i)}$ and $\tilde{C}^{(i)}$, as follows (compare (B-5.12) and (B-6.28)):

$$C^{(i)} = [\Pi^{(i)}_{r_i} \dots \Pi^{(i)}_{m-1}]\Omega^{(i)}_{r_i} \dots \Omega^{(i)}_{m-1} \tag{B-6.34}$$

$$\tilde{C}^{(i)} = [\Pi^{(i)}_{r_i} \dots \Pi^{(i)}_{m-1}][\Omega^{(i)}_{r_i} + \delta\Omega^{(i)}_{r_i}] \dots [\Omega^{(i)}_{m-1} + \delta\Omega^{(i)}_{m-1}] \tag{B-6.35}$$

where

$$\begin{aligned} \Omega^{(i)}_{m-1} &= \Gamma^{(i)^{-1}}_{m-1} \\ \Omega^{(i)}_{m-2} &= \Pi^{(i)}_{m-1}\Gamma^{(i)^{-1}}_{m-2}\Pi^{(i)}_{m-1} \\ \vdots \quad & \quad \quad \vdots \\ \Omega^{(i)}_{r_i} &= \Pi^{(i)}_{m-1}\Pi^{(i)}_{m-2} \dots \Pi^{(i)}_{r_i+1}\Gamma^{(i)^{-1}}_{r_i}\Pi^{(i)}_{r_i+1} \dots \Pi^{(i)}_{m-1} \end{aligned} \tag{B-6.36}$$

"and similarly for $\Omega^{(i)}_j + \delta\Omega^{(i)}_j$ ($j = r_i, \dots, m-1$)"
(this is not explicitly shown in [1] – instead, the reader is left to imagine a version of (B-6.36) with $(\Gamma^{(i)^{-1}}_{r_i} + \delta\Gamma^{(i)^{-1}}_{r_i})$ in place of $\Gamma^{(i)^{-1}}_{r_i}$, etc., and also $\Omega^{(i)}_{r_i} + \delta\Omega^{(i)}_{r_i}$ in place of $\Omega^{(i)}_{r_i}$, etc., but with the $\Pi^{(i)}_j$ factors of course unchanged).

This rearrangement of the permutation factors is useful, because each of the $\Omega^{(i)}_j$ (and, by extension, $\Omega^{(i)}_j + \delta\Omega^{(i)}_j$) factors is lower triangular, meaning that their product is also lower triangular. This means that the expansions given by (B-6.34) and (B-6.35) consist of a single permutation matrix followed by a single lower triangular matrix.

3.1.4 Key elements in the error analysis of the basis decomposition

Each matrix $C^{(i)^{-1}}$, introduced (implicitly) by (B-5.12) (which gives the inverse, $C^{(i)}$), whose purpose is to retriangularise the post-pivot upper Hessenberg matrix $H^{(i)}$ (yielding $U^{(i)}$), is composed of a series of elementary permutation (optional row swap) and lower-triangular (elimination) matrices that descend from the lowest element in the pivot column (column r_i) of $H^{(i)}$, eliminating each subdiagonal element, proceeding downwards and rightwards, until the whole matrix is upper triangular.

One of the implications of this is that each elimination (and permutation) only affects columns to the right of (and including) the column of each respective subdiagonal element to be eliminated. Since the eliminations are performed sequentially downwards and rightwards, the first elimination applies to all columns from the pivot column onwards, the second applies to all columns from the *next* column onwards, and so on. The final elimination applies to the last *two* columns, since the final column needs no elimination of its own.

For algorithm comprehension, $C^{(i)^{-1}}$ is all that is required here. However, for error analysis, we must restrict $C^{(i)^{-1}}$ to perform only the operations necessary for each column. The result of these restrictions are given (in inverse form, like $C^{(i)}$ – and for pivot number $i = k+1$) by:

$$\begin{aligned} \Phi_j^{(k+1)} &= [\Gamma_j^{(k+1)}\Pi_j^{(k+1)} \dots \Gamma_{r_k}^{(k+1)}\Pi_{r_k}^{(k+1)}]^{-1} \quad \text{for} \quad j = r_k, \dots, m-1; \\ \Phi_m^{(k+1)} &= \Phi_{m-1}^{(k+1)}. \end{aligned} \tag{B-6.74}$$

The lower subscript j indicates the column of $H^{(k+1)}$ to which $\Phi_j^{(k+1)^{-1}}$ performs the same operation as $C^{(k+1)^{-1}}$. Note that in fact $\Phi_m^{(k+1)} = \Phi_{m-1}^{(k+1)} = C^{(k+1)}$, since the final two columns of $H^{(k+1)}$ require all eliminations to be applied.

Given (B-6.74), and the knowledge that $C^{(k+1)}$ does not touch columns $1, \dots, r_k - 1$ of $H^{(k+1)}$ (see explanation above), we can model the error in the derivation of each column of $U^{(k+1)}$ from $H^{(k+1)}$ using:

$$U_i^{(k+1)} = H_i^{(k+1)} \quad \text{for} \quad i = 1, \dots, r_k - 1, \tag{B-6.71}$$

and:

$$H_j^{(k+1)} = [\Phi_j^{(k+1)} + \delta\Phi_j^{(k+1)}]U_j^{(k+1)} \quad \text{for} \quad j = r_k, \dots, m. \tag{B-6.75}$$

And then a matrix $\psi^{(k+1)}$ may be introduced, with columns obeying

$$\psi_1^{(k+1)} = \dots = \psi_{r_k-1}^{(k+1)} = 0, \tag{B-6.77}$$

$$\psi_j^{(k+1)} = \delta\Phi_j^{(k+1)}U_j^{(k+1)}, \quad \text{for} \quad j \geqslant r_k. \tag{B-6.78}$$

Since $\Phi_j^{(k+1)}U_j^{(k+1)} = \Phi_m^{(k+1)}U_j^{(k+1)}$ for all j (noting that $\Phi_m^{(k+1)} = C^{(k+1)}$), we may then rewrite (B-6.71) and (B-6.75) as

$$H^{(k+1)} = \Phi_m^{(k+1)} U^{(k+1)} + \psi^{(k+1)}. \tag{B-6.76}$$

3.2 Off-By-One Error in Bartels' Error Analysis [1]

I feel that it is important to draw attention to a simple off-by-one error in Bartels' error analysis. This does not significantly affect the correctness of his analysis, nor mine. I have chosen to go into so much detail here mainly because failing to do so could impede comprehension. And I feel that it is fair to do so, because either I am correct, or else there is a flaw in my own comprehension of Bartels' analysis, which is something that would be important to flag up for future researchers' attention.

In the inductive case of the bounding of $\delta B^{(k)}$ (note that Bartels uses δ as a symbol prefix, just as I use λ in this thesis; hence, δB is to be read as one symbol), there appears to be a mistake in the use of indices. This can easily be seen by a comparison of the first (partial) sentence on p. 422 of [1]:

> let $B^{(1)}$ be obtained from B by dropping the r_1-th column, shifting all subsequent columns one position to the left, and appending A_{s_1} on the right. ([1], p. 422)

with the first sentence on p. 430:

> Let $B^{(k+1)}$ be formed by dropping column $B^{(k)}_{r_k}$, shifting the subsequent columns of $B^{(k)}$ left one place, and appending column A_{s_k}. ([1], p. 430)

In the base case, we have $k = 0$. Plugging $k = 0$ into the above quotation from p. 430, we see that $B^{(1)}$ is formed from $B^{(0)}$, just as on p. 422. (B from p. 422 clearly corresponds to $B^{(0)}$ from p. 430.) However, the leaving and entering column indices are then r_0 and s_0, respectively, while on p. 422, they are r_1 and s_1, respectively.

Since r_0 and s_0 are never explicitly mentioned in [1], while r_1 and s_1 are mentioned in many places (for instance, in (B-5.14)), and the use of r_k and s_k in the base case of $k = 0$ causes various other contradictions throughout the proof, it seems likely that these should instead be r_{k+1} and s_{k+1}. In fact, it is hard to make sense of the analysis any other way.

For instance,

$$C^{(i)} = \Pi^{(i)}_{r_i} \Gamma^{(i)^{-1}}_{r_i} \dots \Pi^{(i)}_{m-1} \Gamma^{(i)^{-1}}_{m-1}, \tag{B-5.12}$$

([1], p. 422)

so that

$$C^{(k+1)} = \Pi^{(k+1)}_{r_{k+1}} \Gamma^{(k+1)^{-1}}_{r_{k+1}} \dots \Pi^{(k+1)}_{m-1} \Gamma^{(k+1)^{-1}}_{m-1}. \tag{3.1}$$

However,

$$\begin{aligned} \Phi_j^{(k+1)} &= [\Gamma_j^{(k+1)}\Pi_j^{(k+1)} \ldots \Gamma_{r_k}^{(k+1)}\Pi_{r_k}^{(k+1)}]^{-1} \quad \text{for} \quad j = r_k, \ldots, m-1, \\ \Phi_m^{(k+1)} &= \Phi_{m-1}^{(k+1)}, \end{aligned} \tag{B-6.74}$$

([1], p. 430)

and letting $j = m - 1$, and distributing the inversion across the matrix factors of $\Phi_{m-1}^{(k+1)}$ gives

$$\Phi_m^{(k+1)} = \Phi_{m-1}^{(k+1)} = \Pi_{r_k}^{(k+1)}\Gamma_{r_k}^{(k+1)^{-1}} \ldots \Pi_{m-1}^{(k+1)}\Gamma_{m-1}^{(k+1)^{-1}}, \tag{3.2}$$

(the $\Pi_\cdot^{(\cdot)}$ factors being elementary permutation matrices, which are symmetric and orthogonal, and therefore self-inverse). Note that the right-hand side of (3.2) would be the same as that of (3.1), except for the fact that r_{k+1} is replaced by r_k in the subscript of the leftmost factors. Which means that, according to (B-5.12) and (B-6.74),

$$\Phi_m^{(k+1)} \not\equiv C^{(k+1)}. \tag{3.3}$$

However, this leads to an apparent contradiction. Taking the following equation:

$$\begin{aligned} B^{(k+1)} + \Delta B^{(k+1)} &= LG^{(k)}H^{(k+1)} \\ &= LG^{(k)}[\Phi_m^{(k+1)}U^{(k+1)} + \psi^{(k+1)}] \\ &= LG^{(k+1)}U^{(k+1)} + LG^{(k)}\psi^{(k+1)}, \end{aligned} \tag{B-6.79}$$

([1], p. 431)

and subtracting $LG^{(k)}\psi^{(k+1)}$ gives

$$LG^{(k)}\Phi_m^{(k+1)}U^{(k+1)} = LG^{(k+1)}U^{(k+1)}. \tag{3.4}$$

But we are also given

$$G^{(k+1)} = G^{(k)}C^{(k+1)}, \tag{B-6.70}$$

([1], p. 430)

and substituting this into (3.4) gives

$$LG^{(k)}\Phi_m^{(k+1)}U^{(k+1)} = LG^{(k)}C^{(k+1)}U^{(k+1)}, \tag{3.5}$$

which can not be guaranteed in general unless $\Phi_m^{(k+1)} \equiv C^{(k+1)}$, contradicting (3.3).

3.3 Updates to Bartels' Floating-Point Arithmetic Model

In Section 4 of [1], Bartels bounds the errors in atomic floating-point computations. There are three things about this analysis that can be updated:

1. There is no need for separate error factors for the terms in an addition or subtraction, because all modern floating-point systems use a guard digit.

2. Each error factor in every atomic operation may optionally be inverted. This can simplify certain parts of the analysis.

3. Modern practice avoids the use of mixed-precision operations (calculations in one precision followed by rounding to another, say)—at least, on a conceptual level. A single unit roundoff value is therefore sufficient, except where my algorithm explicitly changes the precision.

From these considerations, we arrive at the arithmetic model in Section 1.2.1.

3.4 Updated Analysis for Triangular Systems

For the errors arising from the process of triangular substitution, as well as the errors contained in the triangular factors themselves at the beginning of the first iteration, Bartels cites a 1967 paper by Bruce A. Chartres and James C. Geuder. This analysis may be entirely replaced using modern sources; for instance, from [21, pp. 106–107]:

$$(L + \delta L)t = q, \qquad |\delta L| \leqslant O(\epsilon)|L|, \tag{3.6}$$
$$(U^{(k)} + \delta U^{(k)})v = w, \qquad |\delta U^{(k)}| \leqslant O(\epsilon)|U^{(k)}|. \tag{3.7}$$

And for the error in the triangular factors at the beginning of the first iteration, [22, p. 164] provides:

$$B^{(0)} + \delta B^{(0)} = LU^{(0)}, \qquad |\delta B^{(0)}| \leqslant O(\epsilon)|L||U^{(0)}|. \tag{3.8}$$

As L does not change during the algorithm, its norm may be treated, for my purposes, as an arbitrary but fixed constant – that is, $\|L\|_1 = O(1)$. In any case, if partial pivoting is used in the derivation of L, so that we can apply (C.2) from Appendix C.1, then we have $|L| \leqslant (1+\epsilon)\mathcal{L}$, where $\mathcal{L}$ is the lower-triangular matrix of ones. (In practice, partial pivoting ensures that $|L| \leqslant \mathcal{L}$ even when $\epsilon > 0$; however, it is not possible to prove this using only the floating-point model of Section 1.2.1.) In [1], (B-6.82), which is derived from (B-6.68), implies $\|L\|_1 \leqslant m$, which in turn implies that $|L| \leqslant \mathcal{L}$; it may therefore be preferable to use $\|L\|_1 \leqslant m(1+\epsilon)$, and modify (B-6.82) accordingly.

3.5 Updated Bound for the Product Sub-Factors $\tilde{C}^{(i)}$

Using the floating point model of Section 1.2.1, we can bound the components $|\xi_j^{(i)}|$ and $|\eta_j^{(i)}|$ of the matrix $|\delta\Omega_j^{(i)}|$ by considering, as on p. 425 of [1], the effect of an elementary row operation matrix Γ in floating-point arithmetic (i.e. $d = \mathbf{fl}_\epsilon(\Gamma a)$). If the pivot row is ν, so that the elimination is performed on row $\nu+1$ of the upper Hessenberg matrix, this gives:

$$d_i = a_i, \quad i \neq \nu + 1, \tag{3.9}$$

$$(1+\sigma)d_{\nu+1} = a_{\nu+1} - (1+\tau)ga_\nu, \tag{3.10}$$

where $|\sigma|, |\tau| \leqslant \epsilon$. Here, we have used the inverse form for the error in the subtraction.

Using $d_\nu = a_\nu$, (3.10) may be rewritten as

$$a_{\nu+1} = (d_{\nu+1} + gd_\nu) + (\sigma d_{\nu+1} + \tau g d_\nu). \tag{3.11}$$

Careful inspection of the derivation of $\delta\Omega_j^{(i)}$ from $\delta\Gamma_j^{(i)^{-1}}$ reveals that $\xi_j^{(i)}$ and $\eta_j^{(i)}$ are the coefficients of d_ν and $d_{\nu+1}$ respectively in the above error term $(\sigma d_{\nu+1} + \tau g d_\nu)$ (adapted for $\Gamma_j^{(i)}$)—hence, $\xi_j^{(i)} = \tau g_j^{(i)}$ and $\eta_j^{(i)} = \sigma$, for some $|\sigma|, |\tau| \leqslant \epsilon$. Since $|g_j^{(i)}| \leqslant 1$, as stated on p. 426 of [1], we also therefore have $|\xi_j^{(i)}|, |\eta_j^{(i)}| \leqslant \epsilon$.

This means that the non-zero elements of $(\Omega_j^{(i)} + \delta\Omega_j^{(i)})$ are bounded in magnitude by $(1+\epsilon)$. Hence, by the same logic that is used to derive the bound given in [1], we obtain

$$|\tilde{C}^{(i)}| \leqslant (1+\epsilon)^{m-1}\mathcal{I}, \tag{3.12}$$

where $\mathcal{I}$ is the matrix whose elements are all 1. Note that this differs from the bound in [1, p. 427] only in the replacement of $3\epsilon_1/(1-\epsilon_1)$ by ϵ.

3.6 Updated Bound for the Product Sub-Factor Errors $\tilde{C}^{(i)} - C^{(i)}$

On p. 428 of [1], it is stated that the matrix in (B-6.50), which is the product of the diagrammed matrix and $\mathcal{L}$ (the lower-triangular matrix of ones), is elementwise bounded by $2\Big(|\xi_j^{(i)}| + |\eta_j^{(i)}|\Big)\mathcal{L}$. In fact, it is quite easy to see that it is in fact bounded by $\Big(|\xi_j^{(i)}| + |\eta_j^{(i)}|\Big)\mathcal{L}$. Furthermore, when the product of any number of these diagrammed matrices (in order of increasing j) is taken, it is easy to show that $j < \rho_j^{(i)}$ ensures that column j of all but the last such matrix makes no contribution to the result. Thus, for instance,

$$\mathcal{L}|\delta\Omega_{j_1}^{(i)}|\mathcal{L}|\delta\Omega_{j_2}^{(i)}|\mathcal{L} \leqslant |\eta_{j_1}^{(i)}|\Big(|\xi_{j_2}^{(i)}| + |\eta_{j_2}^{(i)}|\Big)\mathcal{L}, \tag{3.13}$$

and in general,

$$\mathcal{L}|\delta\Omega_{j_1}^{(i)}|\cdots\mathcal{L}|\delta\Omega_{j_N}^{(i)}|\mathcal{L} \leqslant \left(\prod_{k=1}^{N-1}|\eta_{j_k}^{(i)}|\right)\left(|\xi_{j_N}^{(i)}|+|\eta_{j_N}^{(i)}|\right)\mathcal{L} \leqslant 2\epsilon^N\mathcal{L}, \tag{3.14}$$

where each $|\xi_{j_k}^{(i)}|, |\eta_{j_k}^{(i)}| \leqslant \epsilon$ for $1 \leqslant k \leqslant N$.

Hence, using (B-6.48) from [1] (and taking into account the final row permutation factors applied to turn this into $C^{(i)} - \tilde{C}^{(i)}$) we obtain

$$|\tilde{C}^{(i)} - C^{(i)}| \leqslant 2\left[(m-1)\epsilon + \binom{m-1}{2}\epsilon^2 + \ldots + \epsilon^{m-1}\right]\mathcal{I} = 2((1+\epsilon)^{m-1}-1)\mathcal{I} = O(\epsilon)\mathcal{I}, \tag{3.15}$$

where $\mathcal{I}$ is the full matrix of ones. Note that this differs from the bound in [1, p. 429] only in the replacement of τ by $2((1+\epsilon)^{m-1}-1)$ (which is $O(\epsilon)$).

3.7 Updated Bound for the Product Factor Error $\delta G^{(k)}$

In [1, p. 429], it is stated that $|\delta G^{(k)}| \leqslant \tau E^{k-1}\mathcal{I}^{k-1}$ where $E = 1+3\epsilon_1/(1-\epsilon_1)$, and $\mathcal{I}$ is the matrix of ones. However, this appears to be a mistake. In fact, if (B-6.27) and (B-6.29) are used, along with (B-6.43), (B-6.47) and (B-6.56), to derive this bound, we first of all see that the expansion of (B-6.27) has k terms, each of which has one factor bounded by (B-6.56) and $k-1$ factors bounded by either (B-6.43) or (B-6.47). Hence, each of the k terms is bounded by a product of $\tau\mathcal{I}^k$ and between 0 and $k-1$ factors of E^{m-1}. One possible bound would therefore be $\tau(1 + E^{m-1} + \ldots + E^{(k-1)(m-1)})\mathcal{I}^k$, for example. Since $E \geqslant 1$, another possibility would be $\tau k E^{(k-1)(m-1)}\mathcal{I}^k$. It is hard to account for this discrepancy.

In Sections 3.5 and 3.6, I have shown that (assuming that a single precision is used for all computations, and that $[m-1]\epsilon < 1$) it is possible to replace E with $1+\epsilon$, and τ with $O(\epsilon)$. We therefore arrive at the following bound:

$$|\delta G^{(k)}| \leqslant O(\epsilon)k(1+\epsilon)^{(k-1)(m-1)}\mathcal{I}^k = O(\epsilon)\mathcal{I}, \tag{3.16}$$

making use of $k = O(1)$ (so boundedness of k is required).

3.8 Overall Bound for the Upper Triangular Factor $U^{(k)}$

As $U^{(k)}$ is derived from $H^{(k)}$ using a procedure that is equivalent to partial pivoting, while $H^{(k)}$ is upper Hessenberg, we may use the bound (C.4) from Appendix C.3, similar to (B-6.85) in [1]. Similarly, assuming that $U^{(0)}$ is derived from a subset of the columns of A

using partial pivoting, we can use the bound (C.3) from Appendix C.2, similar to (B-6.84) in [1] (although Bartels does not state that $U^{(0)}$ is derived using partial pivoting).

For $U^{(0)}$, therefore, using (C.3) and maximising over j we get a bound of

$$|U_{ij}^{(0)}| \leqslant (2(1+\epsilon)^3)^{m-1} \max_{p,q} |A_{pq}| = O(1), \qquad \text{for all } i, j. \tag{3.17}$$

We now turn our attention to (B-6.87) in [1]. It is obvious that $\|A_{s_k}\|_1 \leqslant m \max_{i,j} |a_{i,j}|$ (noting that s_{k-1} must be replaced by s_k, as discussed in Section 3.2).

It can be shown that $\|(G^{(k)} + \delta G^{(k)})^{-1}\|_1 = O(1)$, but the proof is somewhat more involved. I sketch it here.

Lemma 3.1.

$$\|(G^{(k)} + \delta G^{(k)})^{-1}\|_1 = O(1). \tag{3.18}$$

Proof. (Sketch.) Using (B-5.13) and (B-6.27) from [1], we have $G^{(k)}+\delta G^{(k)} = \tilde{C}^{(1)} \cdots \tilde{C}^{(k)}$. Expanding $[\tilde{C}^{(i)}]^{-1}$ using (B-6.35), we get some $\Pi_j^{(i)}$ factors, each with norm 1, and some $[\Omega_j^{(i)} + \delta\Omega_j^{(i)}]^{-1}$ factors, in strictly decreasing order of j. These factors have the same pattern of non-zeros as $\Omega_j^{(i)} + \delta\Omega_j^{(i)}$, and although they are multiplied in the reverse order, we obtain the bound by a procedure similar to that used to derive (B-6.46). $\square$

This leaves us only to show that $\|(L + \delta L)^{-1}\| \leqslant 1$ (approximately or otherwise). In fact, I have so far not found a way to do this, but if the initial basis is decomposed using partial pivoting, with the elements of $|L|$ bounded exactly by 1, then it is possible to show that $\|L^{-1}\| \leqslant 2^{m-1}$ [22, p. 148] (and this applies for both the ℓ_1 and the ℓ_∞ norm). If, on the other hand, we introduce the substitution error δL, and also allow the elements of $|L|$ to be as large as $1+\epsilon$ (as discussed earlier), then we have $\|(L+\delta L)^{-1}\| = O(1)$ as $\epsilon \to 0$.

It is then possible to inductively prove a closed-form bound for $|U^{(k)}|$ where $k > 0$:

Lemma 3.2. *Assuming $k = O(1)$,*

$$|U_{ij}^{(k)}| = O(1). \tag{3.19}$$

Proof. For $k = 0$, see (3.17). For $k > 0$, use (C.4) from Appendix C.3 with $A^0 = H^{(k)}$, $A^{(2(n-1))} = U^{(k)}$, and $n = m$, and take the maximum over j. This yields

$$|U_{ij}^{(k)}| \leqslant m(1+\epsilon)^{3(m-1)} \max_{p,q} |H_{pq}^{(k)}|. \tag{3.20}$$

From (B-6.86) in [1], we have

$$\max_{p,q} |H_{pq}^{(k)}| \leqslant \max\{\max_{p,q} |U_{pq}^{(k-1)}|, \|H_m^{(k)}\|_1\}, \tag{3.21}$$

where, as mentioned above,

$$\begin{aligned}\|H_m^{(k)}\|_1 &= \|(G^{(k)} + \delta G^{(k)})^{-1}(L + \delta L)^{-1} A_{s_k}\|_1 \\ &\leqslant O(1) m \max_{i,j} |a_{i,j}| = O(1).\end{aligned} \tag{3.22}$$

Now, consider $k = 1$. In this case, using (3.17), we immediately have

$$\max_{p,q} |H_{pq}^{(1)}| \leqslant \max\{\max_{p,q} |U_{pq}^{(0)}|, \|H_m^{(1)}\|_1\} = O(1). \tag{3.23}$$

Now, we propose the hypothesis

$$\max_{p,q} |H_{pq}^{(k)}| = O(m^k). \tag{3.24}$$

Note that we do not use O(1). Even though this may seem to work, it fails to take into account the fact that each recursive application of (3.20) and (3.21) may contribute an additional factor of $m(1+\epsilon)^{3(m-1)} = O(m)$.

Next, consider the inductive case for $k+1$:

$$\begin{aligned}\max_{p,q} |H_{pq}^{(k+1)}| &\leqslant \max\{ \max_{p,q} |U_{pq}^{(k)}|, \|H_m^{(k+1)}\|_1\} \\ &\leqslant \max\{m(1+\epsilon)^{3(m-1)} O(m^k), O(1)\} = O(m^{k+1}),\end{aligned}$$

which does indeed prove the hypothesis for all $k \geqslant 1$.

Using $k = O(1)$ and hence $m^k = O(1)$, and substituting into (3.20), we obtain the bound in the lemma. □

3.9 Updated Bound for the Basis Matrix Error $\delta B^{(k)}$

I have addressed the bounding of $B^{(0)}$ in Section 3.4. We can now note that, provided that partial pivoting is used for the initial basis factorisation, using (C.2) from Appendix C.1 and (3.17) from Section 3.8, the initial basis error bound (3.8) becomes

$$|\delta B_{ij}^{(0)}| \leqslant O(\epsilon) \max_{pq} |L_{pq}| \max_{pq} |U_{pq}^{(0)}| = O(\epsilon). \tag{3.25}$$

For the inductive case, I can first make a few comments about Bartels' analysis. First of all, a comparison of (B-6.81) and (B-6.80) from [1] shows that $\|G^{(k+1)}\|_1$ in (B-6.81) may in fact be replaced by $\|G^{(k)}\|_1$. Consideration of (B-6.79) confirms that this is correct.

From (B-5.13), we see that $G^{(k)} = C^{(1)} \cdots C^{(k)}$. This means that we can use the bound (B-6.43) on $|C^{(i)}|$ to derive a bound on $|G^{(k)}|$, and hence its norm. (B-6.82), which is derived from (B-6.68), implies

$$\|G^{(k)}\|_1 \leqslant m^k = O(1), \tag{3.26}$$

making use of $k = O(1)$ (so boundedness of k is required). This is indeed the expected bound.

It may be helpful to realise that there appears to be no advantage to the use of any of the $\Phi_j^{(k+1)}$ with $j < m$. The bounds derived in [1] would be the same even if all instances of $\Phi_j^{(k+1)}$ were replaced by $\Phi_m^{(k+1)}$. Indeed, in view of the identity $\Phi_m^{(k+1)} = C^{(k+1)}$ mentioned in Section 3.2, it might be better to replace them with $C^{(k+1)}$. However, in this case, it is important to remember that each column in the solution of (B-6.75) from [1] requires its own individual error matrix $\delta\Phi_j^{(k+1)}$, and so we may, for instance, call these error matrices $\delta C_j^{(k+1)}$, but we may not remove the index j (although no such index is required on $C^{(k+1)}$). In fact, this change of notation is helpful, because it makes it easy to see the relationship between $\delta C_j^{(i)}$ here and $\tilde{C}^{(i)} - C^{(i)}$ from the analysis for $\delta G^{(k)}$.

(B-6.83) in [1] depends on $\|\delta\Phi_j^{(k+1)}\|_1 \leqslant m$ (or indeed $\|\delta C_j^{(k+1)}\|_1 \leqslant m$). Given that $\delta\Phi_j^{(k+1)}$ (or $\delta C_j^{(k+1)}$) has the same error bound as $\tilde{C}^{(k+1)} - C^{(k+1)}$, which is given by (B-6.56), this effectively depends on $\tau \leqslant 1$. In Section 3.6, I have shown how τ can here be replaced by $O(\epsilon)$. Hence, this requirement is met as $\epsilon \to 0$.

Now, we proceed to inductively derive a closed-form expression for a bound on $\delta B^{(k)}$ (which is equally valid for the ℓ_1 or the ℓ_∞ norm):

Lemma 3.3. *Assuming* $k = O(1)$,

$$\|\delta B^{(k)}\| = O(\epsilon). \tag{3.27}$$

Proof. For $k = 0$, see (3.25). For $k > 0$, first, we must consider $\|\delta A_{s_k}\|$. Using (B-6.68) from [1] (but remembering that erroneously has s_k in place of s_{k+1}, as explained in Section 3.2), and assuming that L is derived using partial pivoting, we have, for $k \geqslant 1$,

$$\begin{aligned}\|\delta A_{s_k}\| &\leqslant (\|L\|\|\delta G^{(k-1)}\| + \|G^{(k-1)}\|\|\delta L\| + \|\delta G^{(k-1)}\|\|\delta L\|)\|H_m^{(k)}\| \\ &\leqslant \left[O(1)O(\epsilon) + O(1)O(\epsilon) + O(\epsilon^2)\right]O(1) = O(\epsilon).\end{aligned} \tag{3.28}$$

And next, $\|\psi^{(k)}\|$. As mentioned above, $\|\delta\Phi_j^{(k)}\| = O(\epsilon)$. This gives, for $k \geqslant 1$, in place of (B-6.83) in [1],

$$\|\psi^{(k)}\| \leqslant \max_j \|\delta\Phi_j^{(k)}\|\|U_j^{(k)}\| = O(\epsilon). \tag{3.29}$$

We propose an inductive hypothesis of $\|\delta B^{(k)}\| = O((k+1)\epsilon)$ (noting that we already have this result for $k = 0$) and then prove it for $k \geqslant 1$. Note that we do not use $O(\epsilon)$, even though this may seem to work. This is because each recursive application of (B-6.81) contributes an additional term of $O(\epsilon)$, as can be seen in the derivation below.

Again assuming partial pivoting for the initial basis, we have (noting that I use s_{k+1} in place of s_k, as explained in Section 3.2, and $G^{(k)}$ in place of $G^{(k+1)}$, as explained earlier in this section):

$$\begin{aligned}\|\delta B^{(k+1)}\| &\leqslant \max\{\|\delta B^{(k)}\|, \|\delta A_{s_{k+1}}\|\} + \|L\|\|G^{(k)}\|\|\psi^{(k+1)}\| \\ &\leqslant \max\{O((k+1)\epsilon), O(\epsilon)\} + O(1)O(1)O(\epsilon) = O((k+2)\epsilon).\end{aligned}$$

Note that where $k > 0$, we can discard a redundant ϵ to obtain $\|\delta B^{(k)}\| = O(k\epsilon)$, following which the boundedness of k gives the result in the lemma. □

3.10 Overall Error Bound

The Bartels–Golub algorithm maintains an invertible decomposition B of the current basis matrix $A_{\mathcal{B}}$ at every iteration of the simplex algorithm. This can be used to derive a column of the updated tableau matrix ($B^{-1}A_s$ where A_s is the corresponding column in the initial tableau), the updated right-hand-side vector ($B^{-1}b$ where b is the initial right-hand-side), and the updated objective function coefficients ($B^{-\top}\gamma$, where $\gamma = c_{\mathcal{B}}$, c is the initial objective function and $\mathcal{B}$ is the current basis). This is enough information to carry out an iteration of the simplex method, and update the decomposition for the next iteration.

To understand the conditions under which the algorithm can make an incorrect pivoting decision, we need bounds on the errors in these three vectors. However, the details of how to derive these bounds are covered in Section 4.6. The only part that must be addressed here is the bound on the basis matrix total error, which Bartels calls $\mathcal{E}^{(k)}$, and which in the main part of this thesis I have called $\Delta M_{\mathcal{B}}$. In this section, I use Bartels' notation, to make it easier to follow this discussion alongside Bartels'. Hence, I consider $\mathcal{E}^{(k)}$, defined by:

$$(B^{(k)} + \mathcal{E}^{(k)})\hat{v} = \hat{q}, \tag{3.30}$$

which is functionally the same as (4.7) in Section 4.5.3 of this thesis. The hats on $\hat{v}$ and $\hat{q}$ are my innovation, to show that they are approximate quantities ($\hat{q}$ is approximate because the inputs to my algorithm are rounded; $\hat{v}$ is approximate both because it is derived from $\hat{q}$ and because of the solving error $\mathcal{E}^{(k)}$).

In equation (4.8), I give an expansion of $\Delta M_{\mathcal{B}}$ in terms of $\lambda M_{\mathcal{B}}$, λB, G, U, λG and λU (in which I choose to omit the iteration indices k). This is similar to Bartels expansion of $\mathcal{E}^{(k)}$, except that I use λ in place of δ, and assume that the initial lower-triangular factor L is the identity matrix, meaning that its substitution error λL will be zero, while Bartels does not seek, as I do, to account for the error $\lambda M_{\mathcal{B}}$ in the input data itself. This accounts for the differences between my expansion in (4.8), and Bartels', which is:

$$\begin{aligned}\mathcal{E}^{(k)} = \delta B^{(k)} &+ LG^{(k)}\delta U^{(k)} + L\delta G^{(k)}U^{(k)} + \delta LG^{(k)}U^{(k)} + L\delta G^{(k)}\delta U^{(k)} \\ &+ \delta LG^{(k)}\delta U^{(k)} + \delta L\delta G^{(k)}U^{(k)} + \delta L\delta G^{(k)}\delta U^{(k)}.\end{aligned} \tag{3.31}$$

3.11 Conclusion

For my purposes, the most important thing to derive from Bartels' paper [1] is what I state in Lemma 4.4: namely, that $\|\Delta M_{\mathcal{B}}\|_\infty = O(\epsilon)$, or, using Bartels' notation and methodology, $\|\mathcal{E}^{(k)}\|_1 = O(\epsilon)$ (since $\lambda M_{\mathcal{B}}$ obeys (4.4), so that $\|\lambda M_{\mathcal{B}}\| = O(\epsilon)$, this term in $\Delta M_{\mathcal{B}}$ can safely be ignored). The norm used, whether ℓ_1 or ℓ_∞, is of no consequence, since this can only affect the order of magnitude by a constant factor. I therefore omit the norm suffix when bounding matrix norms. Since both of these norms are sub-multiplicative, the overall bound on $\|\mathcal{E}^{(k)}\|$ is easily derived from the subsidiary norms.

In this chapter, assuming that partial pivoting is used in the initial basis factorisation, I have shown that, for all $k \geqslant 0$:

Reference	**Norm bound (ℓ_1 or ℓ_∞)**
Appendix C.1, (C.2)	$\|L\| = O(1)$ as $\epsilon \to 0$.
§ 3.4, (3.6)	$\|\delta L\| = O(\epsilon)$ as $\epsilon \to 0$.
§ 3.9, (3.26)	$\|G^{(k)}\| = O(1)$ as $\epsilon \to 0$ (for bounded k).
§ 3.7, (3.16)	$\|\delta G_{ij}^{(k)}\| = O(\epsilon)$ as $\epsilon \to 0$ (for bounded k).
§ 3.8, (3.19) (Lemma 3.2)	$\|U^{(k)}\| = O(1)$ as $\epsilon \to 0$ (for bounded k).
§ 3.4, (3.7)	$\|\delta U^{(k)}\| = O(\epsilon)$ as $\epsilon \to 0$ (for bounded k).
§ 3.9, (3.27) (Lemma 3.3)	$\|\delta B^{(k)}\| = O(\epsilon)$ as $\epsilon \to 0$ (for bounded k).

We can then see that for bounded $k \geqslant 0$, each term in (3.31) has at least one factor whose norm is $O(\epsilon)$, with the remaining factors having norms that are $O(1)$, showing that $\|\mathcal{E}^{(k)}\| = O(\epsilon)$ as $\epsilon \to 0$.

As indicated in Section 3.10, this also demonstrates that $\|\Delta M_{\mathcal{B}}\| = O(\epsilon)$ as $\epsilon \to 0$ when k is bounded (using $\Delta M_{\mathcal{B}}$ as defined by (4.7), along with (4.4) to bound $\|\lambda M_{\mathcal{B}}\|$).

Chapter 4

δ-Complete Simplex Method for Satisfiability

This chapter presents a version of the simplex algorithm, restricted to solving satisfiability problems, that is delta-complete for those problems. A proof of delta-completeness of the algorithm is also presented, making use of the results of Chapter 3.

4.1 Simplex

4.1.1 From satisfiability to optimisation

A linear satisfiability problem of the form (1.6) can be reformulated using a linear optimisation problem (i.e. a linear program). Specifically, that expression is equivalent to:

$$\min_{t,s\in\mathbb{Q}^m,\, x\in\mathbb{Q}^n} \left\{ \sum_{i=1}^{m} (t_i + s_i) \text{ such that } Ax + t - s = b,\ x \geqslant 0,\ t \geqslant 0,\ s \geqslant 0 \right\} = 0. \tag{4.1}$$

By letting $M = [I_m, -I_m, A]$ and $c = [1, \dots, 0, \dots]$ ($2m$ ones followed by n zeros), this can again be rewritten as:

$$\min_{z\in\mathbb{Q}^{2m+n}} \left\{ c^\top z \text{ such that } Mz = b,\ z \geqslant 0 \right\} = 0, \tag{4.2}$$

and, for convenience, I define $\bar{n} := 2m + n$, so that $M \in \mathbb{Q}^{m\times\bar{n}}$ and $c \in \mathbb{Q}^{\bar{n}}$. The solution, $z \in \mathbb{Q}^{\bar{n}}$, is decomposable into $[t, s, x]$, with these corresponding to the vectors in (4.1).

We say that a linear program is feasible if it has a non-empty domain, and that a candidate vector is feasible if it is a member of the domain of the linear program. We call $c^\top z$ the objective function.

The formulation (4.1) is always feasible – for any A, b, we can form an initial feasible assignment of (for instance) $x = 0$, $t = \max(b, 0)$, and $s = \max(-b, 0)$, where max is applied elementwise.

4.1.2 Basic solutions

If the linear program in (4.2) has an optimal solution (i.e. z), then at least one of its optimal solutions is *basic*. This means that it can be described by a *basis* consisting of a sequence $M_{\mathcal{B}}$ of m linearly independent columns of M ($\mathcal{B}$ being the sequence of indices of these columns, and $\mathcal{N} = \{1, \ldots, \bar{n}\} \setminus \mathcal{B}$), such that $M_{\mathcal{B}} z_{\mathcal{B}} = b$ and $z_{\mathcal{N}} = 0$. Since $M_{\mathcal{B}}$ is square and has linearly independent columns, it is invertible, meaning that $z_{\mathcal{B}}$ is uniquely determined. A point z that can be described by a basis in this way is called a *basic solution* (a term that, on its own, does not connotate optimality, nor even feasibility). Clearly, a basic solution z is feasible if and only if $z_{\mathcal{B}} \geqslant 0$.

The simplex algorithm, in its simplest incarnation, requires a feasible basic solution as input. It proceeds iteratively through a series of *pivots*, in each of which a column of $M_{\mathcal{B}}$ is exchanged for a column from $M_{\mathcal{N}}$ in a way that preserves feasibility of the basic solution. When it is no longer possible to do so, the algorithm terminates. In general, it will then conclude either optimality or unboundedness, depending on the reason for termination. However, in my case, since (4.1) is bounded (and since the number of possible bases is finite), an optimal solution will always be found, provided that the algorithm does not loop infinitely. To prevent looping, I use Bland's rule [23] to choose the pivot.

If (4.2) is derived from (4.1), then an initial feasible basic solution can be found by setting x, t and s as described previously, and then choosing the columns from M corresponding to the non-zero values in t and s. Additional columns may then be chosen arbitrarily (but preserving linear independence) in order to make up a total of m linearly independent columns.

4.1.3 Duality

Every linear program, such as that in (4.2), has a corresponding *dual* problem (of which the original problem is called the *primal*), which is also a linear program. In this case, equating the dual problem to zero gives

$$\max_{y \in \mathbb{Q}^m} \left\{ b^\top y \text{ such that } M^\top y + r = c,\ r \geqslant 0 \right\} = 0. \tag{4.3}$$

This statement is equivalent to (4.2) because of the following theorem (whose proof I omit, because this is a well-known result – see Section 4.1.4 below).

Theorem 4.1 (Strong duality theorem)**.** *If z is optimal for the LO problem in* (4.2) *and y is optimal for its dual problem shown in* (4.3)*, then the objective function values are related by $b^\top y = c^\top z$.*

Another useful property of (4.2) and (4.3) is the weak duality theorem (whose proof is trivial).

Theorem 4.2 (Weak duality theorem)**.** *If z is feasible for the LO problem in (4.2) and y is feasible for its dual problem shown in (4.3), then the objective function values are related by $b^\top y \leqslant c^\top z$.*

This is especially useful for my purposes, as it means that any y feasible for (4.3) such that $b^\top y > 0$, even if non-optimal, provides a positive lower bound on the objective function value of (4.1), and, thereby, a proof of unsatisfiability of (1.6).

The dual has the property that if the basis $\mathcal{B}$ produces an optimal basic solution for the LO problem in (4.2), then a basis of (in terms of variables) $r_\mathcal{N}$ and y (with whatever ordering we choose) will produce an optimal basic solution for the dual LO problem in (4.3). This means setting the non-basic variables $r_\mathcal{B} = 0$, which allows y to be derived from $M_\mathcal{B}^\top y = c_\mathcal{B}$, following which it is trivial to derive $r_\mathcal{N}$ (if required). Since $M_\mathcal{B}$ is square and invertible, the same can be said of $M_\mathcal{B}^\top$ (indeed, it may be helpful to note that $(M_\mathcal{B}^\top)^{-1} = (M_\mathcal{B}^{-1})^\top$).

4.1.4 Known results about the simplex algorithm

For some well-known results about the simplex algorithm, I refer to [24]. In particular, invertibility and primal feasibility of the basis are preserved throughout [24, p. 68]. If the algorithm returns a result of *optimal*, then the primal basic solution is indeed optimal [24, p. 67], and the dual basic solution is also feasible and optimal [24, p. 88]. If it returns *unbounded* then the problem is indeed unbounded [24, p. 68]. And if Bland's pivot rule is used, then it terminates [24, p. 73] (or indeed [23]). Note that [24] uses a different formulation of the simplex algorithm, so some of the proofs are not applicable without modification to the form used here.

4.2 δ-Complete Algorithm

In general, to find an exact answer to (1.6), we must use either exact or interval arithmetic. For instance, if the solution set is a point, then in order to certify the result, we need to find this point, which may be an arbitrary rational vector. Since the (finite) floating-point numbers are dyadic, and the dyadic numbers are a strict subset of the rationals, there exist rational vectors that cannot be found by any algorithm that exclusively uses (non-interval) floating-point arithmetic.

The simplex algorithm with approximate (floating-point) arithmetic has been used very successfully to accelerate the exact (rational) simplex algorithm. The main reason for this success is that the simplex algorithm typically makes many unnecessary (or exploratory) pivots on its way to the final basis. Once an optimal basis is found, it can be checked in rational arithmetic by making a sequence of no more than m pivots (known as *factorising* the basis). However, the more difficult the problem, the less effective this will be, because the returned basis will be further away from the nearest usable basis, requiring expensive rational optimisation. There is no bound on the precision that may be needed to find a usable basis for a given problem. Hence, it makes sense to use variable-precision floating-point, increasing the precision until such a basis is found.

The δ-completeness of Algorithm 4.1 depends on the details of the simplex algorithm used on line 8. In particular, correctness can be ensured without examining the details of this step, but termination cannot. Without a proof of termination, the algorithm is not δ-complete. For this reason, it is necessary to provide the details of the simplex algorithm assumed. These details are given in Algorithm 4.2.

Algorithm 4.2 takes as input a floating-point precision with unit roundoff ϵ, and a tolerance value τ. If, instead of a floating-point precision, we are able to choose for rational arithmetic to be used (effectively setting $\epsilon = 0$), and we additionally set $\tau = 0$, then $\mathbf{fl}_\epsilon(\cdot)$ leaves its parameter unchanged, and the algorithm becomes equivalent to the usual exact simplex algorithm. I refer to this as *the exact version* of Algorithm 4.2. In this case, we can make use of the many known results about the simplex algorithm, and in particular those mentioned in Section 4.1.4.

4.3 Floating-Point Exceptions

There are three ways that the model of floating-point arithmetic given in Section 1.2.1 could, in principle, break down during Algorithm 4.2. These are overflow, underflow, and division by zero. Division by zero is prevented by the use of the condition $\hat{d}_i > 0$ in line 17.

Overflow occurs when the result is larger in magnitude than the largest representable magnitude; similarly, underflow occurs when the result of $(x_1 \text{ op } x_2)$ is non-zero but so close to zero that there is no choice of z (satisfying $|z| \leqslant \epsilon$) that makes $(1 + z)(x_1 \text{ op } x_2)$ exactly representable. Both of these are caused by the limited precision of the exponent field.

Note that so-called *denormal* (unnormalised) numbers do not address the underflow problem, as they in fact only reduce the cutoff point, at the cost of a large increase in ϵ. For present purposes, I therefore ignore the existence of denormal numbers. In a real implementation, either denormals must be disabled or trapped, or ϵ must be adjusted accordingly.

Addressing overflow is very difficult without an arbitrary-precision exponent representation, which is currently rare in arbitrary-precision floating-point implementations. I consider this problem to be outside of the scope of this thesis: I assume that all operations (except division by zero) obey (1.2) exactly.

4.4 Proof of δ-Completeness: Correctness

Lemma 4.3. *Algorithm 4.1 returns only correct results.*

Algorithm 4.1: δ-complete simplex algorithm

```
input  : Matrix $A \in \mathbb{Q}^{m \times n}$ and vector $b \in \mathbb{Q}^m$ as in (4.1);
         Infeasibility threshold $\delta \in \mathbb{Q}$.
output: Result symbol $\in \{\delta\text{-sat}, \text{unsat}\}$;
         Final basis $\mathcal{B}$;
         Final primal and dual iterate $(t, s, x, y)$;
         Actual max infeasibility $\delta \in \mathbb{Q}$ (where applicable).
Data: Infinite sequence of precisions such that the unit roundoffs
      $\{\epsilon_0, \epsilon_1, \epsilon_2, \ldots\} \subseteq \mathbb{Q}$ converge to zero
Data: Infinite sequence of tolerances $\{\tau_0, \tau_1, \tau_2, \ldots\} \subseteq \mathbb{Q}$ that converge to zero
1  begin
2      $M := [I, -I, A]$; // Compound matrix as in Section 4.1.1
3      Set the objective coefficient vector: $c \in \mathbb{Q}^{\bar{n}}$ where $\bar{n} = 2m + n$ so that $c_i = 1$
         where $i \leqslant 2m$, and $c_i = 0$ otherwise;
4      Set the initial basis $\mathcal{B}^I = [\mathcal{B}^I_1, \mathcal{B}^I_2, \ldots, \mathcal{B}^I_m]$ such that for each $1 \leqslant i \leqslant m$, we
         have $\mathcal{B}^I_i = i$ if $b_i > 0$ and $\mathcal{B}^I_i = m + i$ if $b_i \leqslant 0$;
5      for $k := 0$ to $+\infty$ do
6          for all $(\epsilon_i, \tau_j) \in \mathbb{N} \times \mathbb{N}$ such that $i + j = k$ do
7              try
8                  Run the simplex algorithm (Algorithm 4.2) with matrix $M$, vectors
                     $b$ and $c$, initial basis $\mathcal{B} := \mathcal{B}^I$, the floating-point precision with unit
                     roundoff $\epsilon_i$, and the tolerance $\tau_j$, yielding a new basis $\mathcal{B}$ for $M$;
9              catch Simplex returns status other than optimal
10                 Go to next precision;
11             try
12                 Factorise the basis matrix $M_{\mathcal{B}}$ in rational arithmetic, yielding the
                     inverse basis matrix $M_{\mathcal{B}}^{-1}$;
13             catch Basis is non-invertible
14                 Go to next precision;
15             $y := M_{\mathcal{B}}^{-\top} c_{\mathcal{B}}$;
16             $z_{\mathcal{N}} := 0$;
17             $z_{\mathcal{B}} := M_{\mathcal{B}}^{-1} b$; // Now $z = [t, s, x]$
18             if $b^\top y > 0$ and $y$ is feasible for (4.3) then
19                 return unsat;
20             if $x \geqslant 0$ and $|t - s| \leqslant \delta \mathbb{1}_m$ then
21                 return $\delta$-sat($\delta := \max(0, \max_{i=1}^m (t_i - s_i))$);
```

Algorithm 4.2: Floating-point primal simplex algorithm, using the Bartels–Golub update [1] and Bland's pivot rule [23]

```
input  : Matrix $M \in \mathbb{Q}^{m \times \bar{n}}$, and vectors $b \in \mathbb{Q}^m$ and $c \in \mathbb{Q}^{\bar{n}}$, as in (4.2);
         Initial basis $\mathcal{B}$;
         Floating-point precision with unit roundoff $\epsilon \in \mathbb{Q}$;
         Tolerance $\tau \in \mathbb{Q}$.
output: Status symbol $\in \{\text{optimal}, \text{unbounded}\}$;
         Final basis $\mathcal{B}$;
         Approximate final objective function value $\hat{\omega} \in \mathbb{Q}$ (where applicable);
         Last entering variable index $j$ (where applicable).
1  begin
2      $\hat{M} := \mathbf{fl}_\epsilon(M)$; // Round inputs
3      $\hat{b} := \mathbf{fl}_\epsilon(b)$;
4      $\hat{c} := \mathbf{fl}_\epsilon(c)$;
5      Compute the initial triangular basis factorisation $L, U$ of basis matrix $\hat{M}_\mathcal{B}$
         using Gaussian elimination (which involves constructing the row
         permutation matrix $\mathit{\Pi}$), set $G := I$, and let $B$ alias $\mathit{\Pi}^{-1}LGU$;
6      for $k := 1$ to $\bar{n}!/(\bar{n} - m)!$ do
7          $\hat{z}_\mathcal{B} := \mathbf{fl}_\epsilon(B^{-1}\hat{b})$; // Primal solution for $B \equiv \mathit{\Pi}^{-1}LGU \approx \hat{M}_\mathcal{B}$
8          $\hat{z}_\mathcal{N} := 0$;
9          $\hat{y} := \mathbf{fl}_\epsilon(B^{-\top}\hat{c}_\mathcal{B})$; // Dual solution
10         $\hat{r} := \mathbf{fl}_\epsilon(\hat{c} - \hat{M}^\top\hat{y})$; // Reduced costs
11         if $\hat{r} \geqslant -\tau\mathbb{1}_{\bar{n}}$ then
12             return *optimal*, $\hat{\omega} := \hat{c}_\mathcal{B}^\top\hat{z}_\mathcal{B}$;
13         $j := \min\{j \in \{1, \ldots, \bar{n}\} : \hat{r}_j < -\tau\}$ // Choose entering variable
14         $\hat{d} := \mathbf{fl}_\epsilon(B^{-1}\hat{M}_{\cdot j})$; // Entering variable coefficient vector
15         if $\hat{d} \leqslant \tau\mathbb{1}_m$ then
16             return *unbounded*;
17         Compute the candidate updates $\hat{u}_i := \mathbf{fl}_\epsilon(\hat{z}_{\mathcal{B}_i}/\hat{d}_i)$ where $\hat{d}_i > 0$ and $\hat{u}_i := 0$
             otherwise, for $1 \leqslant i \leqslant m$;
           // Choose leaving variable (note that $i$ will be an index into $\mathcal{B}$ due to
              the ordering of $\hat{d}$ and $\hat{u}$):
18         $i := \min\{i \in \{1, \ldots, m\} : \hat{d}_i > \tau \wedge \hat{u}_i \leqslant \min_{i=1}^m\{\hat{u}_i : \hat{d}_i > \tau\} + \tau\}$;
19         $\mathcal{B} := [\mathcal{B}_1, \ldots, \mathcal{B}_{i-1}, \mathcal{B}_{i+1}, \ldots, \mathcal{B}_m, j]$;
20         Update the basis factorisation $L, G, U$ using the Bartels–Golub algorithm
             [1] such that $\hat{M}_\mathcal{B} \approx \mathit{\Pi}^{-1}LGU$, in floating-point arithmetic with unit
             roundoff $\epsilon$, letting $B$ again alias $\mathit{\Pi}^{-1}LGU$ following the update. (Note
             that $\mathit{\Pi}$ and $L$ do not change.)
```

Proof. If the algorithm returns on line 19, then the weak duality theorem (Theorem 4.2) ensures that a feasible y such that $b^\top y > 0$ can exist only if the optimal value of (4.1) is positive, in which case (1.6) is indeed unsatisfiable.

If the algorithm returns on line 21, then using $|t - s| \leqslant \delta \mathbb{1}_m$ and $Ax + t - s = b$ (from (4.1)), we have $|Ax - b| \leqslant \delta \mathbb{1}_m$. Since we also have $x \geqslant 0$, the condition (1.8) (i.e. the δ-weakening of (1.6)) is certified by x. $\square$

4.5 Proof of δ-Completeness: General Error Bounds

In order to prove that Algorithm 4.1 terminates, I must first establish some error bounds that are used in the proof. In Section 4.7, I use these bounds to prove that with a high enough finite precision (and appropriate choice of the tolerance parameter τ), Algorithm 4.2 behaves the same as the exact version of the algorithm (where, effectively, $\epsilon = 0$ and $\tau = 0$) at every stage. With the use of Bland's rule, and an iteration cap to prevent cycling when the precision is too low to allow Bland's rule to work properly, this is enough to establish, as I do in Section 4.8, that Algorithm 4.2 eventually terminates with an appropriate basis if the precision is high enough, or otherwise at least terminates, allowing Algorithm 4.1 to try higher precisions and ultimately terminate.

A consequence of this is that if δ is set to 0, then Algorithm 4.1 is complete.

4.5.1 Iteration limit

In Algorithm 4.2, the number of iterations is explicitly limited to $\bar{n}!/(\bar{n} - m)!$, in order to ensure that it terminates regardless of the precision. This is the number of possible sequences of m columns of the matrix M, which has $\bar{n}$ columns. I use the number of sequences rather than the number of sets, because while Bland's rule guarantees that there are no cycles, and hence the algorithm can never return to the same state, it does not explicitly guarantee that it will not return to a basis permutation [23]. The number of sequences is safe to use because, in an exact implementation, the sequence of basic variables describes the full state of a simplex algorithm iteration.

For the inexact (floating-point) case, I first seek to prove that there is some $\epsilon > 0$ and $\tau \geqslant 0$ such that the algorithm behaves correctly in every iteration where $k \leqslant \bar{n}!/(\bar{n} - m)!$. This guarantees the correct operation of Bland's rule, which ensures that my $\bar{n}!/(\bar{n} - m)!$ iteration limit is sufficient to visit every possible sequence of basic variables, and hence terminate correctly. A convenient consequence of this is that we can regard k as a bounded quantity—i.e. $k = O(1)$.

4.5.2 Initial rounding error

The parameters M, b, and c may be arbitrary rationals, and hence are subjected to an initial rounding step in lines 2 to 4 of Algorithm 4.2 to convert them into dyadic numbers with the appropriate precision. After rounding, the corresponding vectors $\hat{M}$, $\hat{b}$, and $\hat{c}$ will be such that

$$\hat{M} = M + \lambda M, \qquad |\lambda M| \leqslant \epsilon |M|, \tag{4.4}$$

$$\hat{b} = b + \lambda b, \qquad |\lambda b| \leqslant \epsilon |b|, \tag{4.5}$$

$$\hat{c} = c + \lambda c, \qquad |\lambda c| \leqslant \epsilon |c|. \tag{4.6}$$

4.5.3 Basis and substitution error

At each iteration of the algorithm, the inverse basis matrix $M_{\mathcal{B}}^{-1}$ is used three times. Conceptually, this matrix, when applied to a column of the initial tableau $[M|b]$, converts it to its current "pivoted" form. This is more efficient than maintaining the complete tableau at each iteration; it also allows (if the matrix is suitably factorised) for improved accuracy (and this turns out to be critical for proving termination). The inverse basis matrix can also be used to derive the current dual vector y efficiently. These uses are illustrated on lines 7, 9, and 14 of Algorithm 4.2.

To this end, during the course of the algorithm, the current basis matrix is factorised (as specified in [1]) into a permutation matrix Π, a lower unit triangular matrix L, an upper triangular matrix U, and a product of update factors G, such that $B \equiv \Pi^{-1}LGU$ is the current invertible approximation of the basis matrix $M_{\mathcal{B}}$. In a real-world implementation, L and U would be refactorised periodically, effectively absorbing the factors from G – to keep the proofs simple, this is not considered here. This means that L is constant, because the factorisation updates only modify G and U.

When Algorithm 4.2 is invoked at line 8 of Algorithm 4.1, the choice of basis means that no pivoting is required, so Π is the identity matrix, and Gaussian elimination at line 5 produces an initial basis factorisation in which L is the identity matrix, and U is a diagonal matrix whose diagonal elements are 1 or -1, depending on whether the corresponding element of b is positive. This significantly simplifies the analysis, enabling us to ignore L and Π, and even the initial Gaussian elimination step itself (which can be replaced in this case by a much simpler procedure). However, in Chapter 5, Algorithm 4.2 will be invoked with more general settings of the parameters.

In a general iteration of Algorithm 4.2, there will be four sources of error: rounding error $\lambda M = \hat{M} - M$ (which is constant throughout, but recall that in each iteration the sequence of columns considered - denoted by $\mathcal{B}$ - is different), factorisation error $\lambda B = B - \hat{M}_{\mathcal{B}}$, error λU from backward substitution of U (or forward substitution of $U^{\top}$), and error λG from applying the factors in G.

If $\Pi = L = I$, then $B = GU$, and when solving $B\hat{v} = \hat{q}$ (i.e. $\hat{v} = B^{-1}\hat{q}$) we will have

$$(M_{\mathcal{B}} + \Delta M_{\mathcal{B}})\hat{v} = (G + \lambda G)(U + \lambda U)\hat{v} = \hat{q}, \tag{4.7}$$

hence,

$$\begin{aligned}\Delta M_{\mathcal{B}} &= (G + \lambda G)(U + \lambda U) - M_{\mathcal{B}} \\ &= B + \lambda GU + G\lambda U + \lambda G\lambda U + \lambda M_{\mathcal{B}} - \hat{M}_{\mathcal{B}} \\ &= \lambda M_{\mathcal{B}} + \lambda B + G\lambda U + \lambda GU + \lambda G\lambda U.\end{aligned} \tag{4.8}$$

Similarly, when solving $B^\top \hat{v} = \hat{q}$ (i.e. $\hat{v} = B^{-\top}\hat{q}$) we will have

$$(M_{\mathcal{B}}^\top + \Delta M_{\mathcal{B}}^\top)\hat{v} = (U^\top + \lambda U^\top)(G^\top + \lambda G^\top)\hat{v} = \hat{q}, \tag{4.9}$$

and then $\Delta M_{\mathcal{B}}$ again obeys (4.8).

Lemma 4.4. *The basis matrix total error $\Delta M_{\mathcal{B}}$, as specified in (4.8), satisfies*

$$\|\Delta M_{\mathcal{B}}\|_\infty = O(\epsilon) \text{ as } \epsilon \to 0 \tag{4.10}$$

at any iteration of Algorithm 4.2, for fixed values of M, b, c.

Proof. See Chapter 3, Section 3.11 (Conclusion). As noted in Section 4.5.1, the iteration limit in Algorithm 4.2 ensures that $k = O(1)$. Note also that this bound applies irrespective of the current basis $\mathcal{B}$, as the derivation makes no assumptions about the basis, of which each problem has only a finite number. □

4.6 Proof of δ-Completeness: Iterate Vector Error Bounds

In this section, I present (for instance) (4.7) as $\hat{v} := \mathbf{fl}_\epsilon(M_{\mathcal{B}}^{-1}\hat{q})$ – even though Algorithm 4.2 actually calls for B (the factorisation of $\hat{M}_{\mathcal{B}}$) instead of $M_{\mathcal{B}}$. The first thing to note about this is that $\mathbf{fl}_\epsilon(\cdot)$ implies the conversion of all inputs to floating-point with precision ϵ, and the use of the most appropriate algorithm for the computation, so that in fact it makes no difference which notation is used. The second is that changing (4.7) to use $\hat{M}_{\mathcal{B}}$ instead of $M_{\mathcal{B}}$ wouldn't make any practical difference either, because it would simply move $\lambda M_{\mathcal{B}}$ out of $\Delta M_{\mathcal{B}}$ and (loosely speaking) into $\hat{M}_{\mathcal{B}}$. I therefore prefer to use the $M_{\mathcal{B}}$ form in all cases, both to reduce visual clutter, and to make it easy to trace the errors back to the algorithm inputs.

4.6.1 Primal, dual and entering coefficient vector total error

Lemma 4.5. *If $\hat{q} = q + \lambda q$ and we define $v := M_{\mathcal{B}}^{-1}q$, and $\hat{v} := \mathbf{fl}_\epsilon(M_{\mathcal{B}}^{-1}\hat{q})$ using (4.7), and $\Delta v := \hat{v} - v$, and if $\|q\|_\infty = O(1)$ and $\|\lambda q\|_\infty = O(\epsilon)$, then*

$$\|\Delta v\|_\infty = O(\epsilon) \text{ as } \epsilon \to 0,$$

for fixed $M, b, c,$ and $\mathcal{B}$.

Proof. Using the invertibility of $M_{\mathcal{B}}$, and $\hat{q} = q + \lambda q$, (4.7) can be rearranged to:

$$M_{\mathcal{B}}(\hat{v} + M_{\mathcal{B}}^{-1}(\Delta M_{\mathcal{B}}\hat{v} - \lambda q)) = q, \tag{4.11}$$

therefore, using $M_{\mathcal{B}}v = q$, we see that $v = \hat{v} + M_{\mathcal{B}}^{-1}(\Delta M_{\mathcal{B}}\hat{v} - \lambda q)$, and therefore,

$$\Delta v = \hat{v} - v = M_{\mathcal{B}}^{-1}(\lambda q - \Delta M_{\mathcal{B}}\hat{v}).$$

Substituting $\hat{v} = v + \Delta v$ into the RHS gives

$$\Delta v = M_{\mathcal{B}}^{-1}(\lambda q - \Delta M_{\mathcal{B}}v) - M_{\mathcal{B}}^{-1}\Delta M_{\mathcal{B}}\Delta v,$$

which rearranges to

$$(I + M_{\mathcal{B}}^{-1}\Delta M_{\mathcal{B}})\Delta v = M_{\mathcal{B}}^{-1}(\lambda q - \Delta M_{\mathcal{B}}v).$$

Now, $M_{\mathcal{B}}^{-1}$ is constant, so if the elements of $|\Delta M_{\mathcal{B}}|$ go to zero as $\epsilon \to 0$, then as they do so $J := I + M_{\mathcal{B}}^{-1}\Delta M_{\mathcal{B}}$ approaches the identity. Beyond a certain point, we will always have $|J_{ii}| > \sum_{j \neq i} |J_{ij}|$ for every i, and then J will be invertible. Hence,

$$\Delta v = (I + M_{\mathcal{B}}^{-1}\Delta M_{\mathcal{B}})^{-1} M_{\mathcal{B}}^{-1}(\lambda q - \Delta M_{\mathcal{B}}v), \text{ as } \epsilon \to 0.$$

Furthermore, if we let $K := M_{\mathcal{B}}^{-1}\Delta M_{\mathcal{B}}$, then if $\|K\|_\infty < 1$, we have the convergent Neumann series expansion [25, p. 191]:

$$(I + K)^{-1} = I - K + K^2 - K^3 + \dots$$

Indeed, using Lemma 4.4, we have $\left\|M_{\mathcal{B}}^{-1}\Delta M_{\mathcal{B}}\right\|_\infty = O(\epsilon)$, for bounded k. Using the squareness of K and the submultiplicativity of the induced matrix norm, we obtain the following convergent power series:

$$\left\|(I + K)^{-1}\right\|_\infty \leqslant \|I\|_\infty + \|K\|_\infty + \left\|K^2\right\|_\infty + \dots \leqslant 1 + \|K\|_\infty + \|K\|_\infty^2 + \dots = O(1).$$

Hence, as we approach the limit, $\left\|(I + M_{\mathcal{B}}^{-1}\Delta M_{\mathcal{B}})^{-1}\right\|_\infty = O(1)$. This gives $\|\Delta v\|_\infty = O(\|\lambda q\|_\infty + \|\Delta M_{\mathcal{B}}\|_\infty \|v\|_\infty)$. Hence, using $\|\lambda q\|_\infty = O(\epsilon)$ and Lemma 4.4, and $\|v\|_\infty \leqslant \left\|M_{\mathcal{B}}^{-1}\right\|_\infty \|q\|_\infty = O(1)$, we arrive at the result in this lemma. □

Corollary 4.6. *If $\hat{M}, \hat{b}, \hat{c}$ obey (4.4), (4.5) and (4.6), and $y = M_{\mathcal{B}}^{-\top} c_{\mathcal{B}}$, $z_{\mathcal{B}} = M_{\mathcal{B}}^{-1} b$, and $d = M_{\mathcal{B}}^{-1} M_{\cdot j}$, and similarly $\hat{y} = fl_\epsilon(M_{\mathcal{B}}^{-\top}\hat{c}_{\mathcal{B}})$, $\hat{z}_{\mathcal{B}} = fl_\epsilon(M_{\mathcal{B}}^{-1}\hat{b})$, and $\hat{d} = fl_\epsilon(M_{\mathcal{B}}^{-1}\hat{M}_{\cdot j})$ using (4.7) and (4.9), and we define $\Delta y := \hat{y} - y$, $\Delta z_{\mathcal{B}} := \hat{z}_{\mathcal{B}} - z_{\mathcal{B}}$, and $\Delta d := \hat{d} - d$, then, as $\epsilon \to 0$,*

$$\|\Delta y\|_\infty = O(\epsilon), \quad \|\Delta z_{\mathcal{B}}\|_\infty = O(\epsilon), \quad \text{and} \quad \|\Delta d\|_\infty = O(\epsilon),$$

for fixed $M, b, c,$ and $\mathcal{B}$.

Proof. Apply Lemma 4.5. In the case of Δy, use $M_{\mathcal{B}}^\top$, $\Delta M_{\mathcal{B}}^\top$, and (4.9) in place of $M_{\mathcal{B}}$, $\Delta M_{\mathcal{B}}$, and (4.7), respectively. □

4.6.2 Reduced cost vector total error

Lemma 4.7. *If $\hat{r} = \mathit{fl}_\epsilon(\hat{c} - \hat{M}^\top \hat{y})$, and $\hat{M}$ and $\hat{c}$ obey (4.4) and (4.6), and $\Delta y = \hat{y} - y$ where $y = M_{\mathcal{B}}^{-\top} c_{\mathcal{B}}$, and $\hat{y} = \mathit{fl}_\epsilon(M_{\mathcal{B}}^{-\top} \hat{c}_{\mathcal{B}})$ using (4.9), and we define $\Delta r := \hat{r} - r$ where $r = c - M^\top y$, then*

$$\|\Delta r\|_\infty = O(\epsilon) \text{ as } \epsilon \to 0, \tag{4.12}$$

for fixed $M, b, c,$ and $\mathcal{B}$.

Proof. We have $\hat{r}_j = \mathbf{fl}_\epsilon(\hat{c}_j - \hat{M}_{\cdot j}^\top \hat{y})$ for each $1 \leqslant j \leqslant \bar{n}$. For error estimation purposes, this can be viewed as an inner product between two vectors of length $m+1$. Hence, following the analysis in [26],

$$\exists \phi_j : |\phi_j| \leqslant (1+\epsilon)^{m+1} - 1, \quad \hat{r}_j = (1+\phi_j)(\hat{c}_j - \hat{M}_{\cdot j}^\top \hat{y}), \qquad 1 \leqslant j \leqslant \bar{n},$$

noting that we have in total m additions, and each term has at most one multiplication.

Now, the total error Δr obeys

$$\begin{aligned}
\Delta r_j &= \hat{r}_j - r_j \\
&= (1+\phi_j)(\hat{c}_j - \hat{M}_{\cdot j}^\top \hat{y}) - (c_j - M_{\cdot j}^\top y) \\
&= (1+\phi_j)(c_j + \lambda c_j - (M_{\cdot j} + \lambda M_{\cdot j})^\top (y + \Delta y)) - (c_j - M_{\cdot j}^\top y) \\
&= \phi_j c_j - \phi_j M_{\cdot j}^\top y + (1+\phi_j)(\lambda c_j - \lambda M_{\cdot j}^\top y - M_{\cdot j}^\top \Delta y - \lambda M_{\cdot j}^\top \Delta y) \\
&= \phi_j c_j + (1+\phi_j)\lambda c_j - (\phi_j M_{\cdot j}^\top + (1+\phi_j)\lambda M_{\cdot j}^\top) y - (1+\phi_j)(M_{\cdot j}^\top + \lambda M_{\cdot j}^\top)\Delta y,
\end{aligned} \tag{4.13}$$

which gives

$$\|\Delta r\|_\infty \leqslant ((1+\epsilon)^{m+2} - 1)(\|c\|_\infty + \left\|M^\top\right\|_\infty \|y\|_\infty) + (1+\epsilon)^{m+2} \left\|M^\top\right\|_\infty \|\Delta y\|_\infty. \tag{4.14}$$

Note that $y = M_{\mathcal{B}}^{-\top} c_{\mathcal{B}}$ depends only on $M, c,$ and $\mathcal{B}$, and so can be considered constant for the purposes of this lemma (so that $\|y\|_\infty = O(1)$).

Finally, therefore, $\|\Delta y\|_\infty = O(\epsilon)$ from Corollary 4.6 yields the lemma. □

4.6.3 Candidate update vector total error

Lemma 4.8. *If $\hat{u}_i = \mathit{fl}_\epsilon(\hat{z}_{\mathcal{B}_i} / \hat{d}_i)$ wherever $\hat{d}_i > 0$, and $\hat{u}_i = 0$ otherwise, for $1 \leqslant i \leqslant m$, and similarly $u_i = z_{\mathcal{B}_i} / d_i$ wherever $d_i > 0$ and $u_i = 0$ otherwise, and $\hat{z}_{\mathcal{B}} = \mathit{fl}_\epsilon(M_{\mathcal{B}}^{-1} \hat{b})$ and $\hat{d} = \mathit{fl}_\epsilon(M_{\mathcal{B}}^{-1} \hat{M}_{\cdot j})$ using (4.7), and $\hat{M}$ and $\hat{b}$ obey (4.4) and (4.5), and we define*

$\Delta u := \hat{u} - u$, then

$$\|\Delta u\|_\infty = O(\epsilon) \text{ as } \epsilon \to 0$$

for fixed M, b, c, and $\mathcal{B}$.

Proof. In Section 4.7.2, it is shown that if ϵ is small enough, then $\hat{d}_i > 0$ if and only if $d_i > 0$; hence, for the purposes of determining limiting behaviour as $\epsilon \to 0$, we can take this property for granted.

Where $\hat{d}_i > 0$, then, and defining the vector η such that $|\eta_i| \leqslant \epsilon$ for $1 \leqslant i \leqslant m$, we have an error

$$\begin{aligned}\Delta u_i = \hat{u}_i - u_i &= (1+\eta_i)\frac{z_{\mathcal{B}_i} + \Delta z_{\mathcal{B}_i}}{d_i + \Delta d_i} - z_{\mathcal{B}_i}/d_i \\ &= \frac{(1+\eta_i)(z_{\mathcal{B}_i} + \Delta z_{\mathcal{B}_i})d_i - (d_i + \Delta d_i)z_{\mathcal{B}_i}}{(d_i + \Delta d_i)d_i} \\ &= \frac{(\eta_i d_i - \Delta d_i)z_{\mathcal{B}_i} + (1+\eta_i)\Delta z_{\mathcal{B}_i} d_i}{(d_i + \Delta d_i)d_i},\end{aligned} \tag{4.15}$$

and the denominator is positive (assuming that ϵ is small enough for the condition of Section 4.7.2).

In Section 4.7.2, it is shown that there exists a minimum positive value of d_i across all bases $\mathcal{B}$ and all $1 \leqslant i \leqslant m$, depending only on the inputs M and (via j) c, and I denote this by d^+_{min}. Additionally, using Corollary 4.6, we have $\|\Delta d\|_\infty = O(\epsilon)$. From this, we see that $\|1/((d_i + \Delta d_i)d_i)\|_\infty \leqslant \left\|1/((d^+_{min} - \epsilon)d^+_{min})\right\|_\infty = O(1)$ as $\epsilon \to 0$. This allows us to bound $\|\Delta u\|_\infty$ like this (using ηd to denote the Hadamard or elementwise product between column vectors η and d, and similar for the products involving $z_\mathcal{B}$ and $\Delta z_\mathcal{B}$):

$$\|\Delta u\|_\infty \leqslant O(\|(\eta d - \Delta d)z_\mathcal{B} + (\mathbb{1}_m + \eta)\Delta z_\mathcal{B} d\|_\infty).$$

Now, clearly, this implies that

$$\|\Delta u\|_\infty \leqslant m(\|\eta\|_\infty \|d\|_\infty + \|\Delta d\|_\infty)\|z_\mathcal{B}\|_\infty + m(1 + \|\eta\|_\infty)\|\Delta z_\mathcal{B}\|_\infty \|d\|_\infty, \tag{4.16}$$

and using the fact that d and $z_\mathcal{B}$ are determined entirely by the inputs and the current basis, and are hence of norm $O(1)$, and again using Corollary 4.6 to give us $\|\Delta z_\mathcal{B}\|_\infty = O(\epsilon)$, we obtain the lemma. $\square$

4.6.4 Error bounds for Algorithm 4.2

Finally, we can combine Corollary 4.6, Lemma 4.7, and Lemma 4.8 into a single lemma, showing how the errors in all of the important quantities in a given iteration of the algorithm go to zero as the precision is increased (provided that no errors have been made prior to this iteration).

Lemma 4.9. *For an arbitrary iteration of Algorithm 4.2 (with $k \leqslant \bar{n}!/(\bar{n}-m)!$), we have:*

$$\|\Delta z_{\mathcal{B}}\|_\infty = O(\epsilon), \text{ as } \epsilon \to 0, \qquad \text{where } \Delta z_{\mathcal{B}} = \hat{z}_{\mathcal{B}} - z_{\mathcal{B}}, \tag{4.17}$$
$$\|\Delta y\|_\infty = O(\epsilon), \text{ as } \epsilon \to 0, \qquad \text{where } \Delta y = \hat{y} - y, \tag{4.18}$$
$$\|\Delta r\|_\infty = O(\epsilon), \text{ as } \epsilon \to 0, \qquad \text{where } \Delta r = \hat{r} - r, \tag{4.19}$$
$$\|\Delta d\|_\infty = O(\epsilon), \text{ as } \epsilon \to 0, \qquad \text{where } \Delta d = \hat{d} - d, \tag{4.20}$$
$$\|\Delta u\|_\infty = O(\epsilon), \text{ as } \epsilon \to 0, \qquad \text{where } \Delta u = \hat{u} - u. \tag{4.21}$$

for fixed $M, b, c, \mathcal{B}$, where $\hat{z}_{\mathcal{B}}, \hat{y}, \hat{r}, \hat{d}, \hat{u}$ come from the version of the algorithm with some precision having unit roundoff ϵ and with tolerance τ, and $z_{\mathcal{B}}, y, r, d, u$ come from the exact version of the algorithm (see Section 4.2), provided that all of the decisions made by the respective algorithms have (in effect) been the same up to the point where each respective variable is calculated. In the case of $z_{\mathcal{B}}, y, r$ this means only that they must have entered iteration k with the same basis. In the case of d, u, they must also have chosen the same entering variable j at line 13.

Proof. Apply Corollary 4.6, Lemma 4.7, and Lemma 4.8. □

4.7 Proof of δ-Completeness: Correct Behaviour

Here, I aim to show that there is a finite precision with unit roundoff ϵ^*, and a tolerance τ^*, such that the simplex algorithm with any precision having unit roundoff $\epsilon \leqslant \epsilon^*$ and any tolerance $\tau \leqslant \tau^*$ makes the correct decisions at every stage (meaning the same sequence of decisions as the exact version of the algorithm – see Section 4.2) – specifically, at lines 11 (optimality determination), 13 (entering variable choice), 15 (unboundedness determination), and 18 (leaving variable choice) – in every iteration.

4.7.1 Correct optimality determination & entering variable choice

This lemma asserts that Algorithm 4.2 can be made to determine optimality correctly, and also choose the entering variable correctly, by making the tolerance low enough and, for that tolerance, making the precision high enough.

Lemma 4.10. *For each M, b, c, there exists some tolerance τ^* and some precision with unit roundoff ϵ^* such that, in any iteration $k \leqslant \bar{n}!/(\bar{n} - m)!$, Algorithm 4.2 with this tolerance and precision behaves the same at lines 11 (optimality determination) and 13 (entering variable choice) as the exact version of the algorithm, provided that both versions have by this point reached the same basis $\mathcal{B}$. Furthermore, this condition applies for any $0 < \tau \leqslant \tau^*$ (but potentially not with the same value of ϵ), and for any such τ, there exists a corresponding precision with unit roundoff $\epsilon^\dagger$ such that for any precision with unit roundoff $\epsilon \leqslant \epsilon^\dagger$, the condition applies for (τ, ϵ).*

Proof. Let $\hat{r}$ be the reduced cost vector computed at some iteration $k \leqslant \bar{n}!/(\bar{n}-m)!$ of the algorithm given tolerance τ and some precision with unit roundoff ϵ, while r is the reduced cost vector computed at iteration k of the algorithm given tolerance 0 and rational arithmetic ($\epsilon = 0$).

Define $\Delta r := \hat{r} - r$, so that for $1 \leqslant j \leqslant \bar{n}$, we want to show that we can ensure that $r_j + \Delta r_j + \tau \geqslant 0$ if and only if $r_j \geqslant 0$ – then the behaviour at lines 11 (optimality determination) and 13 (entering variable choice) will be the same.

It should be clear that the condition $\|\Delta r\|_\infty \leqslant \tau$ is sufficient in order to ensure that $r_j \geqslant 0$ implies $r_j + \Delta r_j + \tau \geqslant 0$, for all j. The next thing is to show that $r_j < 0$ implies $r_j + \Delta r_j + \tau < 0$. Clearly, if $\|\Delta r\|_\infty \leqslant \tau$, and $r_j < 0$ implies $r_j < -2\tau$, then this condition is satisfied.

Now, $r = c - M^\top y = c - M^\top M_{\mathcal{B}}^{-\top} c_{\mathcal{B}}$, so r_j is a deterministic function of M, c, and the basis $\mathcal{B}$ (of which there are finitely many). This means that across all bases $\mathcal{B}$ having at least one negative value of r_j, and all $1 \leqslant j \leqslant \bar{n}$, there exists a maximum (i.e. least negative) negative value of r_j – let us call this value r^-_{max} (and let this be $-\infty$ if r_j is always nonnegative for every basis and all j).

Clearly, the condition $r^-_{max} < -2\tau$ ensures that $r_j < 0$ will imply $r_j < -2\tau$ across all bases $\mathcal{B}$ and all j. Hence, the conditions $\|\Delta r\|_\infty \leqslant \tau$ and $\tau < -\frac{1}{2} r^-_{max}$, taken together, imply that $r_j \geqslant 0$ if and only if $r_j + \Delta r_j + \tau \geqslant 0$, for all j. Using Lemma 4.7, we see that for any given basis, $\|\Delta r\|_\infty = O(\epsilon)$, and hence for any of the finite number of possible bases, there exists ϵ such that $\|\Delta r\|_\infty \leqslant \tau$ for any given value of $\tau > 0$. If we take the minimum such ϵ across all bases $\mathcal{B}$, then we will have $\|\Delta r\|_\infty \leqslant \tau$ regardless of the basis. Meanwhile, $-\frac{1}{2} r^-_{max}$ is a positive constant. Hence, we can choose any $0 < \tau < -\frac{1}{2} r^-_{max}$, and satisfy both conditions by also choosing ϵ appropriately for the chosen τ.

Furthermore, if τ^* is any value of τ obeying $0 < \tau < -\frac{1}{2} r^-_{max}$, then it is clear that any $0 < \tau < \tau^*$ will also obey this condition. Similarly, if ϵ^* is a value of ϵ such that regardless of the basis, $\|\Delta r\|_\infty \leqslant \kappa\epsilon^* \leqslant \tau$, where $\kappa\epsilon^*$ is an instantiation of $O(\epsilon)$, then any $0 < \epsilon < \epsilon^*$ also obeys this condition (noting that ϵ is effectively a parameter of $\|\Delta r\|_\infty$). □

4.7.2 Correct unboundedness determination

This lemma asserts that Algorithm 4.2 can be made to determine unboundedness correctly, by making the tolerance low enough and, for that tolerance, making the precision high enough.

Lemma 4.11. *For each M, b, c, there exists some tolerance τ^* and some precision with unit roundoff ϵ^* such that, in any iteration $k \leqslant \bar{n}!/(\bar{n}-m)!$, Algorithm 4.2 with this tolerance and precision agrees with the exact version of the algorithm on the value of each element of the Boolean conjunction $\hat{d} \leqslant \tau \mathbb{1}_m$ (as in line 15 – unboundedness determination), provided that both versions have by this point reached the same basis $\mathcal{B}$, and selected the same entering variable j. Furthermore, this condition applies for any $0 < \tau \leqslant \tau^*$ (but* *potentially not with the same value of ϵ), and for any such τ, there exists a corresponding precision with unit roundoff $\epsilon^\dagger$ such that for any $\epsilon \leqslant \epsilon^\dagger$, the condition applies for (τ, ϵ).*

Proof. Let $\hat{d}$ be the entering variable coefficient vector computed at some iteration $k \leqslant \bar{n}!/(\bar{n}-m)!$ of the algorithm given tolerance τ and some precision with unit roundoff ϵ, while d is the equivalent vector computed at iteration k of the algorithm given tolerance 0 and rational arithmetic ($\epsilon = 0$). Define $\Delta d := \hat{d} - d$. We want to show that we can ensure that $d_i + \Delta d_i - \tau \leqslant 0$ if and only if $d_i \leqslant 0$, for all $1 \leqslant i \leqslant m$.

First, if we have $\|\Delta d\|_\infty \leqslant \tau$, then this ensures that $d_i \leqslant 0$ implies $d_i + \Delta d_i - \tau \leqslant 0$. If we additionally have $d_i > 0 \to d_i > 2\tau$, then we additionally conclude that $d_i > 0$ implies $d_i > \tau - \Delta d_i$, proving the required reverse implication.

Now, $d = M_{\mathcal{B}}^{-1} M_{\cdot j}$, so d_i is a deterministic function of M, $\mathcal{B}$, and j (noting that j is in the exact rational case determined by M, c and $\mathcal{B}$), meaning that across all bases $\mathcal{B}$ (of which there are finitely many) and $1 \leqslant i \leqslant m$, there exists a minimum positive value of d_i (unless d_i is always nonpositive, in which case correct behaviour is ensured by $\|\Delta d\|_\infty \leqslant \tau$ alone) and I denote this value by d^+_{min} (and let this be $+\infty$ if d_i is always nonpositive for every basis and all i).

Using Corollary 4.6, we see that for any given basis, $\|\Delta d\|_\infty = O(\epsilon)$, and hence for any of the finite number of possible bases, there exists ϵ such that $\|\Delta d\|_\infty \leqslant \tau$ for any given value of $\tau > 0$. If we take the minimum such ϵ across all bases $\mathcal{B}$, then we will have $\|\Delta d\|_\infty \leqslant \tau$ regardless of the basis. Meanwhile, d^+_{min} is a positive constant. Hence, we can choose any $0 < \tau < \frac{1}{2} d^+_{min}$, giving $d_i > 0 \to d_i > 2\tau$, and additionally satisfy $\|\Delta d\|_\infty \leqslant \tau$ simply by choosing a small enough value for ϵ.

Furthermore, if τ^* is any value of τ obeying $0 < \tau < \frac{1}{2} d^+_{min}$, then it is clear that any $0 < \tau < \tau^*$ will also obey this condition. Similarly, if ϵ^* is a value of ϵ such that regardless of the basis, $\|\Delta d\|_\infty \leqslant \kappa\epsilon^* \leqslant \tau$, where $\kappa\epsilon^*$ is an instantiation of $O(\epsilon)$, then any $0 < \epsilon < \epsilon^*$ also obeys this condition (noting that ϵ is effectively a parameter of $\|\Delta d\|_\infty$). □

Corollary 4.12. *For each M, b, c, there exists some tolerance τ^* and some precision with unit roundoff ϵ^* such that, in any iteration $k \leqslant \bar{n}!/(\bar{n}-m)!$, Algorithm 4.2 with this tolerance and precision behaves the same at line 15 (unboundedness determination) as the exact version of the algorithm, provided that both versions have by this point reached the same basis $\mathcal{B}$, and selected the same entering variable j. Furthermore, this condition applies for any $0 < \tau \leqslant \tau^*$ (but potentially not with the same value of ϵ), and for any such τ, there exists a corresponding precision with unit roundoff $\epsilon^\dagger$ such that for any $\epsilon \leqslant \epsilon^\dagger$, the condition applies for (τ, ϵ).*

Proof. This follows trivially from Lemma 4.11. □

4.7.3 Correct leaving variable choice

This lemma asserts that Algorithm 4.2 can be made to choose the leaving variable correctly, by making the tolerance low enough and, for that tolerance, making the precision high enough.

Lemma 4.13. *For each M, b, c, there exists some tolerance τ^* and some precision with unit roundoff ϵ^* such that, in any iteration $k \leqslant \bar{n}!/(\bar{n}-m)!$, Algorithm 4.2 with this tolerance and precision behaves the same at line 18 (leaving variable choice) as the exact version of the algorithm, provided that both versions have by this point reached the same basis $\mathcal{B}$, selected the same entering variable j, agree on the value of each element of the Boolean conjunction $\hat{d} \leqslant \tau \mathbb{1}_m$ (as in line 15 – unboundedness determination), and reach line 18. Furthermore, this condition applies for any $0 < \tau \leqslant \tau^*$ (but potentially not with the same value of ϵ), and for any such τ, there exists a corresponding precision with unit roundoff $\epsilon^\dagger$ such that for any $\epsilon \leqslant \epsilon^\dagger$, the condition applies for (τ, ϵ).*

Proof. Let $\hat{u}$ be the candidate update vector (and $\hat{d}$ the entering variable coefficient vector) computed at some iteration $k \leqslant \bar{n}!/(\bar{n}-m)!$ of the algorithm given tolerance τ and some precision with unit roundoff ϵ, while u (respectively, d) is the equivalent vector computed at iteration k of the algorithm given tolerance 0 and rational arithmetic ($\epsilon = 0$). Define $\Delta u := \hat{u} - u$ and $\Delta d := \hat{d} - d$.

We want to show that we can ensure that at each iteration,

$$u_i + \Delta u_i \leqslant \min\{u_i + \Delta u_i : d_i > 0\} + \tau \iff u_i = \min\{u_i : d_i > 0\}.$$

Note that, as we assume here that both versions of the algorithm agree on the value of each element of the Boolean conjunction $\hat{d} \leqslant \tau \mathbb{1}_m$ – so that $d_i > 0$ if and only if $\hat{d}_i > \tau$, for all $1 \leqslant i \leqslant m$ – we can replace the condition $\hat{d}_i > \tau$ with $d_i > 0$ (which I have done here). Since both versions of the algorithm reach line 18, it is clear that the referenced minima exist.

Consider the sequence $\{\alpha_1, \alpha_2, \ldots, \alpha_\theta\}$ of distinct values of u_i (over $1 \leqslant i \leqslant m$ where $d_i > 0$), arranged in ascending order. Either $\theta = 1$, in which case all candidates have the minimum value (and we may set $\beta := +\infty$), or $\alpha_2 - \alpha_1 > 0$, in which case we may define $\beta := \alpha_2 - \alpha_1$. In either case,

$$2\|\Delta u\|_\infty \leqslant \tau < \beta - 2\|\Delta u\|_\infty \tag{4.22}$$

is sufficient to ensure correct classification of all components, as shall be proven below. Since $\alpha_1 = \min\{u_i : d_i > 0\}$, we have

$$\alpha_1 - \|\Delta u\|_\infty \leqslant \min\{u_i + \Delta u_i : d_i > 0\} \leqslant \alpha_1 + \|\Delta u\|_\infty. \tag{4.23}$$

Now, if $u_i = \alpha_1$, then $u_i + \Delta u_i \leqslant \alpha_1 + \|\Delta u\|_\infty$; hence, using the left-hand sides of (4.22) and (4.23),

$$\min\{u_i + \Delta u_i : d_i > 0\} + \tau \geqslant \alpha_1 + \|\Delta u\|_\infty \geqslant u_i + \Delta u_i, \tag{4.24}$$

correctly classifying the component as α_1. For the other possibility, suppose $u_i \geqslant \alpha_2$ (so that it may be α_2, or α_3, or so on). Noting that $u_i + \Delta u_i \geqslant \alpha_2 - \|\Delta u\|_\infty = \alpha_1 + \beta - \|\Delta u\|_\infty$, and using the right-hand sides of (4.22) and (4.23),

$$\min\{u_i + \Delta u_i : d_i > 0\} + \tau < \alpha_1 + \beta - \|\Delta u\|_\infty \leqslant u_i + \Delta u_i, \tag{4.25}$$

ensuring that the component is not classified as α_1.

Since $u_i = z_{\mathcal{B}_i}/d_i = (M_{\mathcal{B}}^{-1}b)_i/(M_{\mathcal{B}}^{-1}M_{\cdot j})_i$ where $d_i > 0$ and $u_i = 0$ otherwise, we see that u, and hence β, is a deterministic function of M, b, $\mathcal{B}$, and (via j) c. Let $u_{min}^{1\sim2}$ be equal to the smallest positive value of β produced in this way across all bases $\mathcal{B}$ that reach line 18 (leaving variable choice) with more than one distinct value of u_i over $1 \leqslant i \leqslant m$ where $d_i > 0$, and $u_{min}^{1\sim2} := +\infty$ if none such exists. Suppose that we have found some value of τ satisfying $0 < \tau < u_{min}^{1\sim2}$, giving $\tau < \beta$ for every possible β (and also ensuring that $u_{min}^{1\sim2} - \tau > 0$). Then $\|\Delta u\|_\infty < \min\{\tau, u_{min}^{1\sim2} - \tau\}/2$, which is equivalent to $2\|\Delta u\|_\infty < \tau < u_{min}^{1\sim2} - 2\|\Delta u\|_\infty$, ensures that (4.22) holds regardless of the basis $\mathcal{B}$, and hence all components are correctly classified.

Finally, using Lemma 4.8, we see that for any given basis, $\|\Delta u\|_\infty = O(\epsilon)$, and hence for any of the finite number of possible bases, there exists ϵ such that $\|\Delta u\|_\infty < \min\{\tau, u_{min}^{1\sim2} - \tau\}/2$ for any given value of $0 < \tau < u_{min}^{1\sim2}$. If we take the minimum such ϵ across all bases $\mathcal{B}$, then we will have $\|\Delta u\|_\infty < \min\{\tau, u_{min}^{1\sim2} - \tau\}/2$ regardless of the basis.

Furthermore, if τ^* is any value of τ obeying $0 < \tau < u_{min}^{1\sim2}$, then it is clear that any $0 < \tau < \tau^*$ will also obey this condition. Similarly, if ϵ^* is a value of ϵ such that regardless of the basis, $\|\Delta u\|_\infty \leqslant \kappa\epsilon^* < \min\{\tau, u_{min}^{1\sim2} - \tau\}/2$, where $\kappa\epsilon^*$ is an instantiation of $O(\epsilon)$, then any $0 < \epsilon < \epsilon^*$ also obeys this condition (noting that ϵ is effectively a parameter of $\|\Delta u\|_\infty$). □

4.8 Proof of δ-Completeness: Proof of Termination, and Conclusion

As determined in Section 4.7, the following conditions are sufficient for correct behaviour of the simplex algorithm, at any iteration $k \leqslant \bar{n}!/(\bar{n} - m)!$:

1. $\tau < -r_{max}^{-}/2$
2. $\|\Delta r\|_\infty \leqslant \tau$
3. $\tau < d_{min}^{+}/2$
4. $\|\Delta d\|_\infty \leqslant \tau$
5. $\tau < u_{min}^{1\sim2}$
6. $\|\Delta u\|_\infty < \min\{\tau, u_{min}^{1\sim2} - \tau\}/2$

Along with the following note about $-r_{max}^{-}/2$, $d_{min}^{+}/2$, and $u_{min}^{1\sim2}$, the problem is therefore reduced to showing that $\|\Delta r\|_\infty$, $\|\Delta d\|_\infty$, and $\|\Delta u\|_\infty$ go to 0 as $\epsilon \to 0$.

We do not compute the values $-r^-_{max}/2$, $d^+_{min}/2$, and $u^{1\sim 2}_{min}$, but they exist, are determined entirely by the input data M, b and c, and are known to be positive (although they may be infinite). The definitions of these "latent values" are:

- r^-_{max} is the maximum (i.e. least negative) negative value across all bases $\mathcal{B}$ and all $1 \leqslant j \leqslant \bar{n}$ of the exact reduced cost $r_j = c_j - (M_{\cdot j})^\top M_{\mathcal{B}}^{-\top} c_{\mathcal{B}}$ that would be computed by the exact version of Algorithm 4.2 for the given basis.

- d^+_{min} is the minimum (i.e. least positive) positive value across all bases $\mathcal{B}$ and all $1 \leqslant i \leqslant m$ of the exact entering variable coefficient $d_i = (M_{\mathcal{B}}^{-1})_{i\cdot} M_{\cdot j}$ that would be computed by the exact version of Algorithm 4.2 for the given basis (given the entering variable j chosen by this version of the algorithm for this basis, where applicable).

- $u^{1\sim 2}_{min}$ is the minimum (positive) value across all bases $\mathcal{B}$ (if applicable, or else $+\infty$) of the absolute difference between the smallest and second-smallest distinct exact values of $u_i = z_{\mathcal{B}_i}/d_i$ (where $d_i > 0$), among those (across $1 \leqslant i \leqslant m$) that would be computed by the exact version of Algorithm 4.2 for the given basis (given the primal basic solution $z_{\mathcal{B}}$ and entering variable coefficient vector d computed by this version of the algorithm for this basis, where applicable).

Now, clearly, there exists a value τ^* such that any $0 < \tau^\dagger \leqslant \tau^*$ satisfies conditions 1, 3, and 5. For any such value $\tau^\dagger$, Lemma 4.9 tells us that it will then be possible to find some $\epsilon > 0$ that is small enough to enable $\|\Delta r\|_\infty$, $\|\Delta d\|_\infty$, and $\|\Delta u\|_\infty$ to satisfy conditions 2, 4, and 6 with $\tau := \tau^\dagger$.

Lemma 4.14. *For each M, b, c, and initial basis $\mathcal{B}$, there exists a tolerance value τ^* such that for any $\tau \leqslant \tau^*$, there exists a precision with unit roundoff ϵ^* such that for any precision with smaller unit roundoff $\epsilon \leqslant \epsilon^*$, Algorithm 4.2 with tolerance τ and a precision with unit roundoff ϵ behaves the same as the exact algorithm (where $\tau = 0$ and $\epsilon = 0$) at every part of every iteration, thereby giving the same results (status and basis).*

Proof. Clearly, for each M, b, c, there exists some pairing (τ^*, ϵ^*) satisfying all four of Lemmas 4.10, 4.11 and 4.13 and Corollary 4.12. First, we find values of τ capable of satisfying each lemma, and take the minimum, which we may call τ^*. Then, we find precisions with unit roundoffs ϵ such that (τ^*, ϵ) satisfies each lemma, and take the precision with the minimum such ϵ, which we may call ϵ^*. The resulting pairing (τ^*, ϵ^*) then satisfies all four lemmas.

This parameter choice ensures (via Lemma 4.10) identical behaviour at lines 11 (optimality determination) and 13 (entering variable choice) in each iteration, provided that the basis is the same at the start of the iteration. Since the entering variable will be the same, we then have identical behaviour (via Corollary 4.12) at line 15 (unboundedness determination), and (via Lemma 4.11) agreement on the value of each element of the Boolean conjunction $\hat{d} \leqslant \tau \mathbb{1}_m$. We then have all of the preconditions necessary for

Lemma 4.13 to ensure identical behaviour at line 18 (leaving variable choice), should it be reached.

This is immediately sufficient for behaviour to be identical during the first iteration, provided that the initial basis is the same. Since this identical behaviour property guarantees that the bases being equal at the start of an iteration means that they will also be equal at the end of the iteration, we can see that by induction, the entire sequence of bases encountered, as well as behaviour at every part of every iteration, will be the same.

Clearly, for any τ^* satisfying the above conditions, any $0 < \tau \leqslant \tau^*$ will similarly satisfy them. Similarly, for any τ and ϵ^* satisfying the above conditions, τ paired with any $0 < \epsilon \leqslant \epsilon^*$ will also satisfy them. As the final basis is a function only of the pivoting (and termination) decisions made during the algorithm, we find that both algorithms give the same results. □

Lemma 4.15. *Algorithm 4.1 always terminates in finite time.*

Proof. When Algorithm 4.2 is called at line 8 of Algorithm 4.1, M, b, c are passed in, along with the initial feasible basis $\mathcal{B}^I$, and the simplex algorithm proceeds as for (4.2). The objective function $c^\top z$ is a positive sum of non-negative variables, meaning that the minimisation is bounded below by 0. Referring to Section 4.1.4, this ensures that the exact version of Algorithm 4.2 (as defined in Section 4.2) would not return a result of *unbounded*. Indeed, in the exact case, the algorithm would terminate before reaching the iteration cap (because of Bland's rule), the result would be *optimal*, the returned basis $M_\mathcal{B}$ would be invertible, and the corresponding basic solutions (both the primal solution z and the dual solution y) would in fact be feasible and optimal (for (4.2) and (4.3), respectively). By the strong duality theorem (Theorem 4.1), we would have $c^\top z = b^\top y$, where $y = M_\mathcal{B}^{-\top} c_\mathcal{B}$, $z_\mathcal{B} = M_\mathcal{B}^{-1} b$, and $z_\mathcal{N} = 0$.

Now, suppose that $b^\top y \leqslant 0$. Then, naturally, $c^\top z \leqslant 0$, so that, in fact, using the construction of c and the non-negativity of z (from (4.2)), we would have $t = 0$ and $s = 0$ (using the decomposition $z = [t, s, x]$ as in Section 4.1.1). This gives $t - s = 0$, while we also have $x \geqslant 0$ (again from non-negativity of z), meaning that the condition at line 20 is met, so that Algorithm 4.1 would terminate at line 21.

Otherwise, $b^\top y > 0$ with y dual feasible, meeting the condition at line 18, so that Algorithm 4.1 would terminate at line 19.

Recall that this is for the hypothetical case in which it invoked the exact version of Algorithm 4.2. Using Lemma 4.14, this means that Algorithm 4.1 also terminates, in the current iteration, if, for the computed values $M, b, c, \mathcal{B}^I$ (which are functions only of the inputs A, b), the tolerance τ and (for this tolerance) unit roundoff ϵ are sufficiently small. This, in turn, means that by the countability of $\mathbb{N} \times \mathbb{N}$, Algorithm 4.1 always terminates after a finite number of calls to Algorithm 4.2 (whose iteration count is explicitly bounded), and hence in finite time. □

Theorem 4.16. *Algorithm 4.1 is a δ-complete decision procedure for satisfiability over the class of formulas defined by* (1.4).

Proof. Due to Lemma 4.15, Algorithm 4.1 always terminates in finite time. This result is either *unsat* or δ-*sat*. If it is *unsat*, then by Lemma 4.3, the input formula is indeed unsatisfiable. Similarly, if it is δ-*sat*, then by Lemma 4.3, the δ-weakening of the input formula is satisfiable. Algorithm 4.1 hence satisfies Definition 1.2 for the class of formulas that it accepts as input.

Algorithm 4.1 accepts as input a problem in the form (4.1), which is derived from (1.4) by a chain of equivalences. □

Chapter 5

δ-Complete General Linear Programming Method

The previous two chapters have focused on using linear programming (LP) techniques to solve satisfiability problems. However, linear programming is a more general problem that has a linear objective function in addition to the linear constraints. The solution must be both feasible, which means that it satisfies the linear constraints, and optimal, which means that it optimises the objective function. This chapter extends the concept of δ-completeness to the solution of general LP problems, and presents an algorithm that is proven to be δ-complete (using this new definition) over the set of all possible LP problems.

5.1 The Top-Down View

In Section 1.4, the concept of a δ-complete decision procedure is defined for decision problems over Boolean formulas where every proposition is an atomic formula of the form $t \geqslant 0$, where t is an arbitrary arithmetic term.

In Section 4.1.1, the decision problem for the case where each term t is linear is formulated in (4.2) using standard linear algebra notation. In this case, because we are solving a satisfiability problem, M and c must have a particular value (with respect to the original problem matrix A) so that the sum of infeasibilities is minimised. Also, the expression for the minimum is set equal to zero, converting the optimisation problem into the required decision problem.

Here, we consider the more general problem where M and c may be arbitrary, and we are not only testing for equality of the minimum to a particular value, but asking for the value of that minimum, whatever it may be. Although this seems like a more difficult problem, it is in fact solved using the same procedure.

The problem considered is therefore

$$\min_{x \in \mathbb{Q}^n} \{ c^\top x \text{ such that } Ax = b,\, x \geqslant 0 \}, \tag{5.1}$$

where, as is conventional in the literature, I have used A and x, in place of the M and z of Section 4.1.1, with $A \in \mathbb{Q}^{m \times n}$. Note also that the feasibility conditions here are the same as the satisfiability problem (1.6).

We now require a new definition of δ-completeness, because, as mentioned above, the previous definition only covers decision procedures. For the satisfiability component of (5.1) (usually called feasibility), namely the conditions $Ax = b$, $x \geqslant 0$, it would at first seem that we can simply relax them as in (1.8). However, this will in fact prove to be unhelpful.

For the optimality component, the most logical and helpful thing to do, from the end-user perspective, is to provide some sort of guarantee about the optimal value. Unfortunately, it is not possible to do this by simply applying the same logic about δ-feasibility to the dual feasibility conditions, because there is no direct relationship between these conditions and the distance to the optimal value.

So we must instead look directly at the optimal value. If we can bound it rigorously to an interval of size no greater than δ, and the feasibility conditions are also δ-satisfied by a given vector x attaining an objective value within this range, then it seems that we can meaningfully say that the found approximate solution is a δ-satisfying assignment for the optimisation problem, which we may then declare δ-satisfiable. If an algorithm, given any linear program in the form (5.1), either finds a δ-satisfying assignment within finite time, or else declares the problem either infeasible or unbounded, then we may say that it is a δ-complete algorithm for linear programming.

5.2 The Practical View

Since linear programming is typically solved by a method that finds an individual optimal assignment to the feasibility constraints, it makes sense for the definition to assume that the algorithm will return such an assignment when successful. In fact, it turns out that in order to provide a δ-enclosure on the optimal value, which we need in order to guarantee that our assignment gets us within δ of the optimal value, we in fact need two assignments: one to give the lower bound (which, assuming that we are minimising, must be rigorously dual-feasible, but may be arbitrarily primal-infeasible), and one to give the upper bound (which must be rigorously primal-feasible, but may be arbitrarily dual-infeasible).

It therefore looks like our new definition of a δ-complete algorithm for LP need not mention δ-satisfaction of the feasibility conditions at all: exact feasibility, either primal or dual (but not necessarily both) is in fact required. Note that if a given basis is both exactly primal feasible and exactly dual feasible, then it is in fact exactly optimal – we therefore obtain efficiencies by the use of a δ-complete algorithm only in the case where our algorithm obtains two separate bases, one of which is primal but not dual feasible, and the other of which is dual but not primal feasible, before it obtains an exactly optimal basis. This may seem unlikely, but it is possible, and it may be more likely to occur in especially large and complex problems.

Definition 5.1. *A δ-complete algorithm for linear programming is an algorithm that, when given any linear programming instance (in a form dependent on the design of the algorithm, but with objective function $c^\top x$) terminates and outputs exactly one of the following symbols:*

1. δ-optimal *only if there exist assignments x and y to the primal and dual optimisation variables (respectively) such that x is exactly primal feasible, y is exactly dual feasible, and the* duality gap $|c^\top x - b^\top y| \leqslant \delta$*;*

2. infeasible *only if the LP is infeasible;*

3. unbounded *only if the LP is unbounded.*

Note that unlike in the satisfiability case, these possibilities are mutually exclusive: if x exists, then the problem is feasible, and if y exists, then it is bounded. Also note that we may have $c^\top x = b^\top y$ (which is guaranteed if x and y may belong to the same basis), in which case the solution is exactly optimal.

To interpret the δ-optimal case, we must recall that $c^\top x$ is the primal objective function of an exactly primal-feasible point, which means that by the duality theorems, it serves as a rigorous upper bound on the objective function (that is being minimised). Similarly, $b^\top y$ is the dual objective function of an exactly dual-feasible point, which means that again by the duality theorems, it serves as a rigorous lower bound on the objective function. In this case, therefore, we have established an enclosure of size at most δ on the true optimum value.

It is easy to see how this definition could be generalised to other types of optimisation problem and solution method, provided that the method can determine infeasibility and unboundedness, and has some concept of duality gap: the *infeasible* and *unbounded* cases remain the same (but with *LP* replaced by the appropriate problem type), while the *δ-optimal* case requires simply that the problem is determined to be neither infeasible nor unbounded (and hence both primal and dual feasible), and that the duality gap is determined to be bounded by δ. If the method provides either a primal or a dual objective function iterate, then this, in combination with the corresponding duality gap, yields the desired enclosure on the true objective function value. However, these values must be rigorously computed, just as the primal and dual objective function values are in Algorithm 5.1 prior to determining a result of δ-optimal.

5.3 Proof of δ-Completeness

The pseudocode for the δ-complete general LP algorithm is given in Algorithm 5.1. First, we show correctness of the algorithm (including that its determination of a δ-optimal result is always correct). Then, we will show that the algorithm terminates. Together, this will prove that the algorithm is δ-complete according to Definition 5.1.

Algorithm 5.1: δ-complete full-LP simplex algorithm

```
input : Matrix $A \in \mathbb{Q}^{m \times n}$ (with full row rank, which implies that $m \leqslant n$) and vectors $b \in \mathbb{Q}^m$ and $c \in \mathbb{Q}^n$ as in (5.1);
        Optimality threshold $\delta \in \mathbb{Q}$.
output: Result symbol $\in \{\delta\text{-optimal}, \text{infeasible}, \text{unbounded}\}$;
        Final bases $\mathcal{B}^L$ and $\mathcal{B}^U$;
        Actual optimality bound $\delta \in \mathbb{Q}$ (where applicable).
begin
    $\bar{n} := 2m + n$;
    $M := [I_m,\ -I_m,\ A]$; // Feasibility LP
    Set the feasibility LP objective coefficient vector: $\bar{c} \in \mathbb{Q}^{\bar{n}}$ so that $\bar{c}_i = 1$ where $i \leqslant 2m$, and $\bar{c}_i = 0$ otherwise;
    Set the initial feasibility LP basis $\mathcal{B}^F = [\mathcal{B}^F_1, \mathcal{B}^F_2, \ldots, \mathcal{B}^F_m]$ such that for each $1 \leqslant i \leqslant m$, we have $\mathcal{B}^F_i = i$ if $b_i > 0$ and $\mathcal{B}^F_i = m + i$ if $b_i \leqslant 0$;
    for $k := 0$ to $+\infty$ do
        for all $(\epsilon_i, \tau_j) \in \mathbb{N} \times \mathbb{N}$ such that $i + j = k$ do
            try
                Solve the feasibility problem $Ax = b$, $x \geqslant 0$ approximately using Algorithm 4.2 with $M$, $b$, $c := \bar{c}$, initial basis $\mathcal{B} := \mathcal{B}^F$, the floating-point precision with unit roundoff $\epsilon_i$, and the tolerance $\tau_j$, yielding a new basis $\mathcal{B}$ for $M$ and approximate final objective function value $\hat{\omega}$;
            catch Simplex returns status other than optimal
                // Incorrect result; feasibility LP is always feasible and bounded
                Go to next precision;
            if $\hat{\omega} > \tau_j$ then
                // Indicates infeasibility of original LP
                try
                    Factorise the basis matrix $M_{\mathcal{B}}$ in rational arithmetic, yielding the inverse basis matrix $M_{\mathcal{B}}^{-1}$;
                catch Basis is non-invertible
                    Go to next precision;
                $z^F_{\mathcal{B}} := M_{\mathcal{B}}^{-1} b$;
                $\omega^F := \bar{c}_{\mathcal{B}}^\top z^F_{\mathcal{B}}$;
                $y := M_{\mathcal{B}}^{-\top} \bar{c}_{\mathcal{B}}$;
                $r := \bar{c} - M^\top y$;
                if $z^F_{\mathcal{B}} \geqslant 0$ and $r \geqslant 0$ and $\omega^F > 0$ then
                    return infeasible;
            if the returned basis contains indices below $2m + 1$ then
                Replace these with arbitrary non-basic indices of at least $2m + 1$ and not already in the basis (recall that $m \leqslant n$);
            Solve the optimality problem (i.e. input LP) approximately using Algorithm 4.2 with $M := A$, $b$, $c$, initial basis $\mathcal{B} := \mathcal{B}_{2m+1,\ldots,2m+n}$, the floating-point precision with unit roundoff $\epsilon_i$, and the tolerance $\tau_j$, yielding a status (optimal or unbounded), a new basis $\mathcal{B}$ for $A$, and (if unbounded) the last chosen entering variable index $p$;
            try
                Factorise the basis matrix $A_{\mathcal{B}}$ in rational arithmetic, yielding the inverse basis matrix $A_{\mathcal{B}}^{-1}$;
            catch Basis is non-invertible
                Go to next precision;
            $x_{\mathcal{B}} := A_{\mathcal{B}}^{-1} b$;
            if $x_{\mathcal{B}} \geqslant 0$ then
                // Primal feasible; upper bound valid
                $\omega^U := c_{\mathcal{B}}^\top x_{\mathcal{B}}$;
                $\mathcal{B}^U := \mathcal{B}$;
            $y := A_{\mathcal{B}}^{-\top} c_{\mathcal{B}}$;
            $r := c - A^\top y$;
            if $r \geqslant 0$ then
                // Dual feasible; lower bound valid
                $\omega^L := b^\top y$;
                $\mathcal{B}^L := \mathcal{B}$;
            else if $r_p < 0$ and $x_{\mathcal{B}} \geqslant 0$ and the returned status is unbounded then
                $d := A_{\mathcal{B}}^{-1} A_{\cdot p}$;
                if $d \leqslant 0$ then
                    return unbounded;
            if $\omega^U$ and $\omega^L$ are both set, and $\omega^U - \omega^L \leqslant \delta$ then
                return $\delta$-optimal($\delta := \omega^U - \omega^L$);
```

Lemma 5.2 (Correctness)**.** *Algorithm 5.1 returns only correct results (using the definition of δ-optimality from Definition 5.1).*

Proof. If the algorithm returns *infeasible* at line 22, then we have found an exactly primal and dual feasible vector $z_{\mathcal{B}}^F$ that causes the objective function $\bar{c}_{\mathcal{B}}^{\top} z_{\mathcal{B}}^F$ of the feasibility LP to be positive. Primal and dual feasibility means that this objective function value is optimal. Since the objective function is the sum of infeasibilities of the original LP, and it was being minimised, this means that a sum of infeasibilities of 0 is unattainable; hence, the original LP is infeasible, as claimed.

If the algorithm returns *unbounded* at line 42, then there exists an exactly primal-feasible basis $\mathcal{B}$ for A (since $x_{\mathcal{B}} \geqslant 0$) such that applying the inverse basis matrix $A_{\mathcal{B}}^{-1}$ (which yields the tableau for that basis) transforms a column $A_{\cdot p}$ of the original tableau to a vector $A_{\mathcal{B}}^{-1} A_{\cdot p}$ that is entirely nonpositive. The variable corresponding to such a tableau column (which must of course be non-basic, as otherwise it would be a unit vector after the transformation) can be made arbitrarily positive without breaking primal feasibility, since any increase can be balanced in each row by increasing that row's basic variable – which appears in that row only, and with a coefficient of 1. Since, ignoring the constraint matrix, all variables are bounded only by their lower bound of zero, none of these bounds is broken either. As the transformed objective function coefficient for this variable, r_p, is negative, and the basic variables by construction do not appear in the transformed objective function, this enables the objective function to be made arbitrarily negative without breaking feasibility. Hence, since the transformed LP is by construction equivalent to the original, the LP is unbounded.

If the algorithm returns δ-*optimal* at line 44, then there exists an exactly primal-feasible basis $\mathcal{B}^U$ with corresponding primal variables $x_{\mathcal{B}^U}$ (primal feasibility being ensured by $x_{\mathcal{B}} \geqslant 0$ and the basic construction of $x_{\mathcal{B}}$) and primal objective function value $\omega^U = c_{\mathcal{B}^U}^{\top} x_{\mathcal{B}^U}$, and an exactly dual-feasible basis $\mathcal{B}^L$ with corresponding dual variables y (dual feasibility being ensured by $r = c - A^{\top} y \geqslant 0$ and the basic construction of y) and dual objective function value $\omega^L = b^{\top} y$ such that $\omega^U - \omega^L \leqslant \delta$, which satisfies the definition of δ-optimality from Definition 5.1. (Note that since $\omega^U - \omega^L > 0$ is ensured by weak duality, it is not necessary to take the absolute value.) $\square$

What we have shown so far is that the algorithm can only return a correct result. That is all very well, but how do we know that it will ever encounter a correct result? For this, we must know a little more about the error bounds of various quantities produced by the algorithm. Fortunately, the algorithm is very similar to that of Chapter 4, meaning that only one new error lemma is needed:

Lemma 5.3 (Objective function value total error)**.** *If $\hat{\omega} = \hat{c}_{\mathcal{B}}^{\top} \hat{z}_{\mathcal{B}}$, and $\hat{z}_{\mathcal{B}} = \mathit{fl}_{\epsilon}(M_{\mathcal{B}}^{-1} \hat{b})$ using (4.7), and $\hat{M}, \hat{b}, \hat{c}$ obey (4.4), (4.5) and (4.6), and we define $\Delta\omega := \hat{\omega} - \omega$ where $\omega = c_{\mathcal{B}}^{\top} z_{\mathcal{B}}$ and $z_{\mathcal{B}} = M_{\mathcal{B}}^{-1} b$, and $\Delta z_{\mathcal{B}} := \hat{z}_{\mathcal{B}} - z_{\mathcal{B}}$, then*

$$|\Delta\omega| = O(\epsilon) \text{ as } \epsilon \to 0, \tag{5.2}$$

for fixed M, b, c, and $\mathcal{B}$.

Proof. Expanding the definition, we have

$$\begin{aligned}\Delta\omega = \hat{\omega} - \omega = \hat{c}_{\mathcal{B}}^{\top}\hat{z}_{\mathcal{B}} - c_{\mathcal{B}}^{\top}z_{\mathcal{B}} &= (c_{\mathcal{B}} + \lambda c_{\mathcal{B}})^{\top}(z_{\mathcal{B}} + \Delta z_{\mathcal{B}}) - c_{\mathcal{B}}^{\top}z_{\mathcal{B}} \\ &= c_{\mathcal{B}}^{\top}\Delta z_{\mathcal{B}} + \lambda c_{\mathcal{B}}^{\top}z_{\mathcal{B}} + \lambda c_{\mathcal{B}}^{\top}\Delta z_{\mathcal{B}}.\end{aligned} \tag{5.3}$$

Using (4.6) and Corollary 4.6, we have $|\lambda c| := |\hat{c} - c| \leqslant \epsilon|c|$ and hence $\|\lambda c\|_{\infty} = O(\epsilon)$, and $\|\Delta z_{\mathcal{B}}\|_{\infty} := \|\hat{z}_{\mathcal{B}} - z_{\mathcal{B}}\|_{\infty} = O(\epsilon)$. Additionally using the fact that $\|z_{\mathcal{B}}\|_{\infty} = \left\|M_{\mathcal{B}}^{-1}b\right\|_{\infty} = O(1)$, this gives

$$\begin{aligned}|\Delta\omega| &\leqslant m\|c_{\mathcal{B}}\|_{\infty}\|\Delta z_{\mathcal{B}}\|_{\infty} + m\|\lambda c_{\mathcal{B}}\|_{\infty}\|z_{\mathcal{B}}\|_{\infty} + m\|\lambda c_{\mathcal{B}}\|_{\infty}\|\Delta z_{\mathcal{B}}\|_{\infty} \\ &= O(1)O(\epsilon) + O(\epsilon)O(1) + O(\epsilon)O(\epsilon) = O(\epsilon).\end{aligned} \tag{5.4}$$

□

And along with this single new error lemma, we need only a single new correctness of behaviour lemma. Note that unlike Lemma 5.2, this makes an assertion not about the results that can be returned by the algorithm, but about its inner workings.

Lemma 5.4 (Correct infeasibility determination)**.** *For each A, b, c, there exists some tolerance τ^* and some precision with unit roundoff ϵ^* such that, provided that the floating-point simplex algorithm (Algorithm 4.2) with this tolerance and precision as called at line 9 of Algorithm 5.1 returns the result* optimal, *Algorithm 5.1 with this tolerance and precision then proceeds to behave exactly the same at line 12 as the exact version of Algorithm 5.1, in which $\tau = 0$ and rational arithmetic is used (so that, effectively $\epsilon = 0$). Furthermore, this condition applies for any $0 < \tau \leqslant \tau^*$ (but potentially not with the same value of ϵ), and for any such τ, there exists a corresponding precision with unit roundoff $\epsilon^{\dagger}$ such that for any precision with unit roundoff $\epsilon \leqslant \epsilon^{\dagger}$, the condition applies for (τ, ϵ).*

Proof. The return of *optimal* ensures that $\hat{\omega}$ is defined and line 12 of Algorithm 5.1 is reached. There is "correct" behaviour (identical to the exact version) at line 12 if and only if

$$\hat{\omega} > \tau \iff \omega > 0. \tag{5.5}$$

Defining $\Delta\omega := \hat{\omega} - \omega$, this may be rewritten

$$\omega > \tau - \Delta\omega \iff \omega > 0. \tag{5.6}$$

Clearly, if $\tau - \Delta\omega \geqslant 0$, then the forward implication is satisfied. From Lemma 5.3, we have $|\Delta\omega| = O(\epsilon)$ as $\epsilon \to 0$, for any given M, b, c and $\mathcal{B}$. This means that for any $\tau > 0$ and valid basis $\mathcal{B}$, there exists some $\epsilon^* > 0$ such that for any $0 < \epsilon \leqslant \epsilon^*$, we have $|\Delta\omega| \leqslant \tau$ when $\hat{\omega}$ is calculated using floating-point arithmetic with unit roundoff ϵ, and hence this forward implication. If we take the minimum such value of ϵ^* across all of the finite number of bases $\mathcal{B}$ for M, b, c, then we acquire a value ϵ^* (corresponding to the given τ) such that any $0 < \epsilon \leqslant \epsilon^*$ satisfies the forward implication regardless of basis.

Conversely, suppose that we define ω^+_{min} to be the minimum positive value of ω across all of the finite number of possible bases $\mathcal{B}$ (and $+\infty$ if no such value exists), noting that for each basis the value $\omega = c_{\mathcal{B}}^\top z_{\mathcal{B}} = c_{\mathcal{B}}^\top M_{\mathcal{B}}^{-1} b$ depends only on the basis itself and the fixed input values M, b, c. [Although we do not compute it, this "latent" value, like those described in Section 4.8, exists, is determined entirely by the input data M, b, c, and is known to be positive (although it may be infinite).] Then the reverse implication is satisfied if we have $\tau - \Delta\omega < \omega^+_{min}$. If $|\Delta\omega| \leqslant \tau$, then this requirement becomes $\tau < \omega^+_{min}/2$. Clearly, for any $\tau^* > 0$ satisfying this condition, it is also satisfied by any $0 < \tau < \tau^*$. □

Next, we can combine Lemma 5.4 with the lemmas of Chapter 4 to derive our desired termination lemma.

Lemma 5.5 (Termination)**.** *Algorithm 5.1 always terminates in finite time.*

Proof. Up to and including line 11, Algorithm 5.1 is functionally identical to the beginning portion of Algorithm 4.1, up to and including line 10 of that algorithm. This includes the initialisation of both algorithms, which sets up the exact same feasibility LP, the two nested loops, which are identically quantified in both algorithms, and the initial try-catch block inside the inner loop, which runs the floating-point algorithm in exactly the same way and responds identically to a non-optimal return value.

This means that just as stated in Lemma 4.15, the simplex algorithm at this point proceeds as for (4.2), and in the "exact case", in which exact arithmetic is used throughout the algorithm (including both calls to Algorithm 4.2), so that $\epsilon_i = 0$, and also $\tau_j = 0$, we can use the known results of Section 4.1.4. So again, the minimisation is bounded below by 0, so a result of *unbounded* would be spurious and can safely be ignored. Again, in the exact case, the algorithm would terminate before reaching the iteration cap (because of Bland's rule), the result would be *optimal*, the returned basis $M_{\mathcal{B}}$ would be invertible, and the corresponding basic solutions (both the primal solution z and the dual solution y) would in fact be feasible and optimal (for (4.2) and (4.3), respectively).

At this point, however, we face something new: in line 12 of Algorithm 5.1, there is a comparison between $\hat{\omega}$, which is an inexact value returned by Algorithm 4.2 and τ_j. However, in the exact case mentioned above, $\hat{\omega}$ is the exact optimal objective value of the feasibility LP, and $\tau_j = 0$. This means that the `if` block is entered if and only if the true objective value, $\bar{c}_{\mathcal{B}}^\top z_{\mathcal{B}}^F$, is greater than zero. Hence, the recomputations in lines 17–18 will in this case yield $\omega^F > 0$. Along with the guarantee of primal and dual feasibility mentioned above, this ensures that the algorithm would terminate at line 22.

Otherwise, in the exact case, if $\hat{\omega} \leqslant 0$, we know that the true optimal objective value is indeed 0 (meaning that the original LP is feasible - as shown in Section 4.1.1), and as mentioned above, the returned basis $M_{\mathcal{B}}$ is invertible. This means, for one thing, that any indices below $2m+1$, which correspond to variables that occur in the feasibility objective function with coefficient 1, must have variable values of 0, and hence can be dropped from the basis in line 24. Adding arbitrary indices of at least $2m+1$ is then necessary to maintain a square basis matrix, and adds additional columns of A to the basis. The full row-rank of A guarantees that this can be done (if necessary), and yields primal feasibility of the basis for the original LP [24, p. 71]. If no such change is required, then the basis is already primal feasible for the original LP, after the index changes in line 25.

In the exact case, the second call to Algorithm 4.2 at line 25 also finds the exact result, this time to the input LP, (5.1). The final basis is again invertible and primal feasible (these conditions are always guaranteed by exact primal simplex, but dual feasibility is not, in general), and the result (*optimal*/*unbounded*) is correct. This means that the factorisation on line 27 succeeds, the recomputed $x_{\mathcal{B}}$ is found to be nonnegative, and ω^U is set on line 32.

If the result is *optimal*, then the basis is in fact dual feasible, so the recomputed r is found to be nonnegative, ω^L is set on line 37, and (due to the strong duality theorem, and because ω^U and ω^L come from the same basis) the difference $\omega^U - \omega^L$ is zero, and the algorithm terminates at line 44.

If the result is *unbounded*, then the recomputed final entering variable's reduced cost, r_p, is found to be negative, the recomputed candidate update vector d is found to be nonpositive, and the algorithm terminates at line 42.

Recall that this is in the hypothetical case in which rational arithmetic is used throughout both algorithms, and $\tau_j = 0$. In an actual iteration of the inner loop in Algorithm 5.1, floating-point arithmetic with unit roundoff $\epsilon_i > 0$ is used within both calls to Algorithm 4.2, and for the comparison on line 12 (which is between two floating-point values with this precision), and $\tau_j > 0$. Due to Lemma 4.14, for each of the two calls to Algorithm 4.2, of there exists a tolerance value τ^* such that for any $0 < \tau < \tau^*$, there exists a precision with unit roundoff ϵ^* such that for any $0 < \epsilon < \epsilon^*$, the algorithm with this tolerance and a precision with this unit roundoff behaves the same as the exact algorithm, giving the same results. In addition, due to Lemma 5.4, the same can be said about the behaviour of the comparison on line 12.

There is a complication here caused by the fact that the second call to Algorithm 4.2 is given a basis that depends on the outcome of the first. However, we can overcome this by noting that there are a finite number of possible bases for each of the problems passed to Algorithm 4.2. This means that we can minimise this τ^* across all bases for the second problem (and the initial basis for the first problem, and the comparison mentioned above), giving us a value that is guaranteed to work for all three, regardless of the starting basis used for the second call. Then, for any $\tau^\dagger > 0$ less than or equal to this minimum τ^*, Lemmas 4.14 and 5.4 assert the existence of a corresponding ϵ^*, for both LP problems (and the intervening comparison), regardless of basis. So just as with τ^*, we can minimise ϵ^* across all bases for the second problem, and the initial basis for the first problem (and the comparison). Any pair $(\tau^\dagger, \epsilon^\dagger)$, where $\epsilon^\dagger \leqslant \epsilon^*$, is guaranteed to ensure correct behaviour of Algorithm 4.2 in both cases, and at line 12 of Algorithm 5.1, meaning identical behaviour throughout the current iteration of Algorithm 5.1, guaranteeing termination, as noted above.

This means that by the countability of $\mathbb{N} \times \mathbb{N}$, Algorithm 4.1 always terminates after a finite number of iterations, and hence a finite number of calls to Algorithm 4.2 (whose iteration count is explicitly bounded), and hence in finite time. □

And finally, we can put together correctness and termination to prove δ-completeness.

Theorem 5.6 (δ-Completeness)**.** *Algorithm 5.1 is a δ-complete algorithm for linear programming (according to Definition 5.1).*

Proof. Due to Lemma 5.5, Algorithm 5.1 always terminates in finite time. The result is either *infeasible*, *unbounded*, or *δ-optimal*. If it is *infeasible* or *unbounded*, then by Lemma 5.2, the input LP is indeed infeasible or unbounded, respectively. On the other hand, if it is *δ-optimal*, then Lemma 5.2 states that the LP satisfies the definition of δ-optimality from Definition 5.1. Algorithm 5.1 hence satisfies Definition 5.1. □

Chapter 6

Implementation

6.1 Interior-Point Algorithm

The δ-complete full-Newton step algorithm of Algorithm 2.1 was implemented straightforwardly in a project called `dcipm_simple`. For this, I used the dense matrix functionality in Dlib,[1] which I extended to work with the `mpq_class` (rational) type from GMP[2] in place of the native floating-point types.

In order to certify the result, it is necessary to obtain some prior overestimate of the number of iterations required to achieve an exact solution. As in the pseudocode of Algorithm 2.1, I use a very crude overapproximation based on the product of ℓ_2 norms of the columns of the fully assembled problem matrix. This is likely to result in many more iterations than necessary, except in the case of an early exit due to δ-sat.

Making matters worse, the use of rational arithmetic causes each iteration to be more expensive than the last. There has been some research into the use of floating-point arithmetic for interior-point methods, but the conditions for convergence to an exact solution tend to be vague and problem-dependent [27].

6.2 Simplex for Satisfiability

The δ-complete satisfiability simplex algorithm of Algorithm 4.1 was implemented in two variants, both as driver routines within my modified version of `QSopt_ex`:[3]

1. As `QSdelta_solver`, which uses phase I (feasibility) simplex to solve the problem, which should be provided as an ordinary (otherwise arbitrary) LP with its objective function set to zero;

2. As `QSexact_delta_solver`, which uses phase II (optimality) simplex to solve the problem, which should be provided as a feasible and bounded LP whose objective function is set to the sum of infeasibilities of the input problem.

[1]`http://dlib.net/`
[2]`https://gmplib.org/`
[3]`https://github.com/martinjos/qsopt-ex`

Note that despite the naming, both routines compute exactly δ-complete results.

In both cases, the iteration over tolerances is omitted; a single tolerance is used throughout. This is because of limitations in `QSopt_ex`. Because of the way the algorithm is designed, this can only affect the termination property (not correctness).

In order to support SMT-LIB input files, and arbitrary Boolean combinations of linear inequalities, the δ-complete SMT solver `dReal4`[4] was heavily modified such that the core ICP solver routine was replaced by my modified version of `QSopt_ex` (either `QSdelta_solver` or `QSexact_delta_solver`, depending on command-line options). The resulting program is called `dLinear4`.[5]

6.3 Simplex for Linear Programming

The δ-complete full-LP simplex algorithm of Algorithm 5.1 was implemented as a third driver routine (`QSdelta_full_solver`) within my modified version of `QSopt_ex`. As in the satisfiability case, only a single tolerance is used, and the same comments apply.

To support SMT-LIB input files with arbitrary Boolean combinations of linear inequalities, this routine was also integrated into `dLinear4`. In order to activate `QSdelta_full_solver`, the solver must be provided an input file with one of the optimisation extension commands supported by `dReal4` (`maximize` or `minimize`), which is used to set the objective function and sense before calling `check-sat`. Instead of `delta-sat` or `unsat`, this will now output `delta-optimal`, `infeasible`, or `unbounded`.

If the Boolean combination of constraints asserted in the SMT-LIB input file encodes more than one linear program (for instance, if it is a disjunction of conjunctions of linear inequalities), then each will be solved in turn (with the same objective function). If any one is found to be unbounded, then the whole problem is of course declared `unbounded`. If all are found to be infeasible, then the whole problem is declared `infeasible`. Otherwise, each feasible LP in the combination will yield an upper and lower bound on the objective function, and in each case these will differ by at most the chosen δ.

A suitable upper and lower bound for the optimal objective value over the entire domain is found (in a minimisation problem) by independently taking the minimum of the upper and lower bounds across all of the feasible LPs. For the lower bound, the reasoning is obvious: the true minimum objective value may be as low as the lowest lower bound across all LPs. For the upper bound, it is only slightly less obvious: the true minimum objective value can never be higher than the lowest upper bound across all LPs. In a maximisation problem, the same reasoning is applied, except with both the lower and upper bound independently maximised, rather than minimised, across all of the LPs. It is simple to verify that in either case, the final upper and lower bound will differ by at most δ.

[4]https://github.com/dreal/dreal4

[5]https://github.com/martinjos/dlinear4

Chapter 7

Evaluation

7.1 δ-Complete Interior-Point Method

As detailed in Section 6.1, I wrote an experimental (and fairly naive) implementation of Algorithm 2.1, which I called `dcipm_simple`. As the properties of this algorithm, at least as discussed in this thesis, depend on the use of exact arithmetic, I used GMP rationals for all numerical operations. The resulting implementation took 3 minutes 29 seconds to solve a single 3x3 instance with $\delta = 1/32$ (giving, in this case, an exact result). I concluded that it would not be feasible to use interior-point methods without acceleration by inexact (i.e. finite-precision) arithmetic.

In [27], it was noted that interior-point algorithms can still converge under certain conditions even when finite-precision arithmetic is used. This would be an interesting avenue for further research into δ-complete interior-point methods. However, in view of the extreme complexity of interior-point methods, and the relative maturity and ubiquity of simplex methods, I decided that the time constraints of my project made it more sensible and profitable to focus on simplex for the time being.

7.2 δ-Complete Simplex Methods

Algorithms 4.1 and 5.1 have been implemented in C++ in my fork of the software package `QSopt_ex` ("QSopt Exact"), which I shall here call `QSopt_dc` ("QSopt Delta-Complete"). Note that although a δ-complete solution is not necessarily an exact solution to the original problem, `QSopt_dc` does find exact solutions to the δ-relaxations of the input problems.

Just like `QSopt_ex`, `QSopt_dc` can be invoked directly using a command-line tool called `esolver`, in which case LP problems in `.mps` or `.lp` format may be processed. Alternatively, `dLinear4`, which is my heavily modified fork of `dReal4`, incorporates `QSopt_dc` into an SMT-style framework, allowing more complex queries in `.smt2` format (with arbitrary Boolean connectives) to be processed, with or without a linear objective function (i.e. either LP or pure linear satisfiability).

Although the algorithms as described in this thesis use Bland's rule as the simplest way to enable theoretical termination properties to be derived, the implementations do not. This follows standard practice in the field of linear programming, where it is recognised that cycling (even in the presence of degeneracy) almost never occurs in real-world problems, and Bland's rule (in the pure form used for didactic purposes) is far too costly to be justified. In the absence of degeneracy-induced cycling (and memory-permitting), the termination property (Lemma 4.15) still holds for this implementation. A more sophisticated implementation might combine an efficient pivoting strategy with Bland's rule as a backstop.

Most tests were carried out on the Rocket HPC Service at Newcastle University, using a single core on an Intel Xeon E5-2699 v4 CPU with 128 GB memory shared between 44 cores. Tests requiring a large amount of memory were instead carried out on the Newcastle University School of Computing's own (virtually disused) HPC system, using a single core on an Intel Xeon E5-2690 v2 CPU with 512 GB of memory shared between 24 cores.

The SMT-LIB benchmarks were generally found to be less than ideal for testing my approach, because they typically equate to a large disjunction of relatively small linear conjunctions, while my solver is optimised for large conjunctions. More suitable problems were found within the linear programming community. My solvers were tested on both types of problem.

7.3 The Sloane–Stufken Problem

The Sloane–Stufken problem is a family of LPs introduced in [28] and used for benchmarking in [9, 18]. The generated LPs are dense and feasible, and their dimensions grow rapidly with the problem parameters. Many are incorrectly found infeasible by the commercial CPLEX simplex and log-barrier solvers [9, 18].

In this thesis, I use the Sloane–Stufken LPs as generated by the program `eg_sloan` from QSopt_ex [18]. However, as my solver is a satisfiability solver, the objective functions were removed (including for the `esolver` tests).

In Table 7.5, `cvc4_apx` refers to CVC4 with the `--use-approx` command-line option, which causes it to use a double-precision LP solver to bootstrap its exact rational solver, which produces the final result. Normally, CVC4 uses a rational solver only, and these timings are given in the `cvc4` column. CVC4 was compiled in competition mode with GPL dependencies enabled, and all relevant best-performance options turned on.

`esolver` indicates the original QSopt_ex solver of that name.

`dlin` refers to my solver dlinear2, `d0` and `dmax` indicate a delta parameter of 0 and the maximum tested, respectively, and `pI` indicates the *phase I* version of dlinear2, which uses phase I simplex (i.e. the solver's built-in feasibility detection) instead of the usual phase II. These can effectively be thought of as two separate solvers, although they share some code, and both implement the same algorithmic concept.

The `bits` columns indicate the final floating-point precision used by the solver before termination. This is present only for the esolver and dlinear2 tests, as only these solvers use variable-precision floating-point.

7.3.1 Comparison with third-party solvers

First of all, there is, on most instances, a clear division between CVC4 and Z3 on the one hand, and the better-performing QSopt_ex-based solvers esolver and dlinear2 on the other. The columns `dlin_d0` and `dlin_pI_d0` are directly comparable with the other solvers, as this parameter setting (delta=0) requires dlinear2 to find a complete (as opposed to δ-complete) solution.

The times and bits in the esolver columns seem to be inconsistent with those reported in [18, p. 30]. Even with the original problem files and the original precompiled binary, I was unable to reproduce Espinoza's results. I put this down to differences in computer architecture.

Between CVC with/without `--use-approx` and Z3, the optimal solver seems to be highly problem-dependent.

The variations among `dlin_d0` and `dlin_pI_d0` are entirely the result of the different (but equivalent) strategy that the *phase I* version of dlinear2 uses for determining satisfiability. Specifically, the *phase I* version uses QSopt_ex's built-in feasibility determination support (so-called *phase I simplex*), while the *regular* version uses the normal optimiser (*phase II simplex*) with an artificial objective function (which makes it equivalent to phase I simplex). Surprisingly, the *regular* version outperformed the *phase I* version on almost all of the instances from [18]. However, on the new instances that I tested (those with $s1 = 40$, $s2 = 80$, and $k1 > 10$), the *phase I* version was the best-performing by far. These differences may be accounted for by variations in the solver implementations for phase I and phase II simplex.

Daniel Espinoza's esolver mostly outperforms dlinear2 with delta=0 on the instances from [18], although there are a couple of exceptions. There are a number of things that could account for this: dlinear2 has a more verbose input format which is more expensive to parse, and it uses QSopt_ex's public API to input the data, whereas QSopt_ex's built-in MPS parser uses an internal interface. dlinear2 also uses an extensively modified version of QSopt_ex which does a slightly different set of rational checks. However, on the new instances ($s1 = 40$, $s2 = 80$, and $k1 > 10$), the *phase I* version of dlinear2 with delta=0 outperformed esolver – indeed, it was the best-performing complete solver on all of these instances.

7.3.2 Maximum deltas / minimum precisions

The final four columns of Table 7.5 show the performance of dlinear2 on the highest tested deltas – leading, naturally, to termination at the lowest precision. (This delta varied by problem.) For the *regular* version of dlinear2 (`dlin_dmax`), this led to termination, in every case, after only the double-precision solve (and rational check), in most cases "outperforming" all of the other solvers. Of course, with *delta* > 0, dlinear2 may give a result of delta-sat, which, although rigorous, is not equivalent to (indeed, is weaker than) the results found by the other solvers. However, this is also an advantage of the delta-complete approach: the ability to terminate once a result is "close enough", while still deriving a rigorous assertion about the problem (specifically, that a perturbation to every bound of magnitude no more than delta leads to a satisfiable problem).

It is notable that the *phase I* version of dlinear2 seems much less likely to terminate with double-precision. This can perhaps be explained by the fact that the QSopt_ex library's phase I simplex solver, as used by the *phase I* version, uses a fixed tolerance to determine feasibility (which is then used to determine satisfiability of the input problem), whereas when phase II simplex is used, as in the *regular* version, determining satisfiability is entirely the responsibility of the caller. Note that this fixed tolerance does not affect the properties of Algorithm 4.1, as it effectively just imposes an artificial upper bound on the value of τ required to guarantee termination, forcing it in some cases to call Algorithm 4.2 some finite number of additional times.

7.3.3 Comparison of deltas

dlinear2 was tested on each problem with a wide variety of deltas. I have chosen to focus here on the *phase I* version of dlinear2, as this version more often terminates at intermediate precisions, making it more useful as a delta-complete solver.

Table 7.1 shows the performance of the *phase I* version of dlinear2 on all tested instances that could be made to halt at a range of different precisions. All quoted timings are the full running times for Algorithm 4.1. The first was present in [18]; the rest were derived from this instance by adding the same number to $k1$ and $k2$ (increasing both the number of variables and the number of constraints quadratically) and double this number to t (which has the effect of maintaining a problem with exactly 2 inequality constraints).

On all of these derived problems, the *phase I* version of dlinear2 was the most efficient solver, as shown in Table 7.5. In addition, in Table 7.1 it can be seen that there is a clear negative correlation between delta and running time. Although the deltas here are probably too large to be of any practical use, this suggests that there may be problems for which my algorithm provides a useful way to perform a breadth-first search of the solution space, expending the minimal amount of time and energy to find an adequately precise solution. This could be especially helpful for tackling much larger problems.

Table 7.1: Sloane–Stufken LP running times (seconds) and maximum deltas – delta-complete solver; all precisions (3+-precision instances)

Parameters					**288 bits**		**192 bits**		**128 bits**		**double**	
s1	**k1**	**s2**	**k2**	**t**	**delta**	**time**	**delta**	**time**	**delta**	**time**	**delta**	**time**
40	10	80	10	19			0–10^3	2.13	10^4–10^{11}	1.26	10^{12}	0.20
40	15	80	15	29	0–10^3	38.68	10^4–10^7	26.72	10^8–10^{60}	15.01		
40	16	80	16	31	0–10^5	45.44	10^6–10^8	27.85	10^9–10^{10}	9.12	10^{11}	1.74
40	17	80	17	33	0–10^6	79.17	10^7–10^{20}	51.43	10^{21}–10^{308}	26.97		
40	18	80	18	35	0–10^7	86.54	10^8–10^9	46.77	10^{10}	13.91	10^{11}	3.11
40	19	80	19	37	0–10^8	548.53	10^9–10^{24}	488.42	10^{25}–10^{107}	436.22		

7.4 Infeasible Instances

All of the problems in Section 7.3 are dense, and satisfiable (i.e. the LPs are feasible). In this section, I discuss my findings for some sparse, unsatisfiable problems (infeasible LPs) from the popular *netlib* set.[1] I singled out four problems in particular, because these were the problems that could be induced to return a result of *delta-sat*.

I set a timeout of just 5 minutes (i.e. 300 seconds), as these are relatively small instances by today's standards. As can be seen in Table 7.2, Z3 and CVC4 without `--use-approx` timed out on three of the instances. CVC4 with `--use-approx` didn't time out, but still took longer than any of the QSopt_ex-based solvers on those three instances.

`esolver` is (again) Daniel Espinoza's QSopt_ex solver. `dlinear2 (unsat)` gives the timing corresponding to the deltas in row `delta (unsat)`, and `dlinear2 (delta-sat)` gives that corresponding to the deltas in row `delta (delta-sat)`. `unsat` signifies a run in which the solver was driven to give an exact result, while `delta-sat` signifies a run in which it halted on one of the preceding (lower) precisions.

dlinear2 was outperformed by esolver in all cases where the solver proceeded to find an exact result, which can again largely be explained by differences in the parser and the precise details of the solving routine. However, it "outperformed" esolver when the required precision was sufficiently low (i.e. when delta was sufficiently large). For cplex2, qual and vol1, the deltas may be small enough to make a delta-sat result acceptable. This again suggests the potential of Algorithm 4.1 for larger problems where the solving times may be much more significant.

[1] http://www.netlib.org/lp/

Table 7.2: Netlib infeasible timings (seconds)

	gran	**cplex2**	**qual**	**vol1**
z3	0.078	timeout	timeout	timeout
cvc4	0.946	timeout	timeout	timeout
cvc4 --use-approx	0.965	1.409	0.826	1.678
esolver	2.679	0.094	0.191	0.215
dlinear2 (unsat)	14.821	0.172	0.399	0.377
dlinear2 (delta-sat)	0.660	0.032	0.077	0.051
delta (unsat)	0–22	0–$10^{-6.7}$	0–0.015	0–0.03
delta (delta-sat)	23+	$10^{-6.523}$+	0.02+	0.04+

Table 7.3: Full LP. Qprec = QSopt_ex precision.

Qprec	**#inst**	**QSopt_ex**		**QSopt_dc**			**SoPlex$_{\text{fac}}$**		
		#iter	**time**	**#iter**	**time**	**Δt**	**#iter**	**time**	**Δt**
Any	564	3688.7	8.3	3688.7	12.9	1.55	4725.1	6.8	0.82
64-bit	498	3307.2	7.1	3307.2	11.6	1.63	4434.0	6.7	0.95
128-bit	63	8532.0	25.0	8532.0	27.9	1.12	7825.4	7.3	0.29
192-bit	3	5142.4	21.0	5142.4	25.4	1.21	4297.6	5.8	0.27

Table 7.4: Repeated SoPlex results.

Test set	**#inst**	**SoPlex$_{\text{fac}}$**				**SoPlex$_{\text{rec}}$**				
		#ref	**#fac**	**t$_{\text{fac}}$**	**t**	**#ref**	**#rec**	**t$_{\text{rec}}$**	**t**	**Δt**
All	1144	1.9	0.95	0.18	3.1	69.8	6.70	1.13	5.3	1.69
[1, 7200]	592	2.0	0.97	0.36	9.4	132.9	10.59	2.72	20.4	2.16
[10, 7200]	316	2.0	0.98	0.68	24.7	234.8	14.92	7.03	85.3	3.45
[100, 7200]	148	2.0	0.98	1.39	41.7	399.6	19.98	18.40	276.9	6.64

Table 7.5: Sloane–Stufken LP running times (seconds) and precision bits – all solvers

Parameters					**cvc4**	**cvc4_apx**	**z3**	**esolver**		**dlin_d0**		**dlin_pI_d0**		**dlin_dmax**		**dlin_pI_dmax**	
s1	**k1**	**s2**	**k2**	**t**	**time**	**time**	**time**	**time**	**bits**	**time**	**bits**	**time**	**bits**	**time**	**bits**	**time**	**bits**
2	100	3	2	4	*	31.65	*	0.90	128	6.33	128	2.66	128	1.42	dbl	1.64	dbl
3	18	5	18	35	16983.45	8255.55	52079.51	84.01	128	141.41	128	44.98	128	2.01	dbl	12.25	dbl
3	20	5	20	35	*	46625.52	*	198.03	128	323.97	128	235.04	128	4.43	dbl	81.56	dbl
5	18	7	18	35	5475.33	18079.95	16792.50	76.16	128	141.33	128	218.40	128	2.57	dbl	218.92	128
11	20	13	20	30	4976.15	*	7109.50	†	128	2485.74	192	2726.95	192	5.07	dbl	2726.95	192
17	20	19	20	10	11580.10	795.24	177.26	9.95	192	62.29	192	73.34	192	5.26	dbl	6.09	dbl
19	20	23	20	10	16026.04	*	215.84	9.58	192	54.53	192	74.65	192	5.23	dbl	5.87	dbl
31	20	37	20	10	2677.64	109.24	105.94	15.58	288	95.84	288	108.31	288	5.36	dbl	6.55	dbl
40	10	80	10	19	9.76	9.12	11.19	1.95	128	2.49	128	2.04	192	0.13	dbl	0.20	dbl
40	11	80	11	21	20.47	29.16	24.58	4.84	192	3.67	128	3.77	192	0.26	dbl	0.30	dbl
40	12	80	12	23	44.18	29.84	37.54	15.60	192	13.80	192	7.72	192	0.29	dbl	4.94	128
40	13	80	13	25	96.84	67.31	70.35	24.84	192	31.50	192	12.72	192	0.43	dbl	8.02	128
40	14	80	14	27	180.35	2086.27	131.27	39.43	192	50.25	192	17.91	192	0.63	dbl	10.42	128
40	15	80	15	29	324.85	187.08	217.37	68.24	192	135.78	192	39.08	288	0.94	dbl	14.92	128
40	16	80	16	31	599.63	10877.56	357.84	143.27	192	367.44	192	45.58	288	1.32	dbl	1.74	dbl
40	17	80	17	33	891.23	12804.80	466.80	220.79	288	360.96	192	80.21	288	2.02	dbl	26.93	128
40	18	80	18	35	1672.97	*	885.13	527.66	288	632.75	288	87.03	288	2.79	dbl	3.11	dbl
40	19	80	19	37	2556.60	966.18	2322.77	752.82	288	1937.63	288	553.44	288	4.22	dbl	433.52	128
40	20	80	20	39	3706.65	*	4503.21	1253.26	288	2291.99	288	153.08	288	5.41	dbl	6.33	dbl
43	20	47	20	10	2182.81	370.06	143.71	21.26	288	119.71	288	121.55	288	5.49	dbl	6.89	dbl
61	20	73	20	10	2589.18	472.60	139.95	13.11	288	103.35	288	145.15	288	5.53	dbl	6.16	dbl
200	20	250	20	10	1200.98	1079.33	165.14	21.95	432	134.67	432	291.80	432	5.84	dbl	6.33	dbl

* Command timed out (> 16 hours) † Command exited with error (floating-point exception)

Chapter 8

Conclusion

In this thesis, three new δ-complete algorithms for linear problems were developed, and proven to be δ-complete. The first two algorithms were concerned with pure linear satisfiability problems, which are also known as linear feasibility problems within the optimisation community. The first was based on interior-point methods, while the second was based on the simplex algorithm.

The third algorithm was concerned with general linear optimisation problems, and was also based on the simplex algorithm. For this algorithm, since δ-completeness has up to now only been defined for satisfiability algorithms, a new concept of δ-completeness for linear optimisation algorithms was developed. This could potentially be extended to other types of optimisation algorithm in future.

All three algorithms were then implemented. The simplex-based algorithm implementations show promise, because their timings were clearly able to come close to the state-of-the-art on some problems.

However, I speculate that something could perhaps prove more useful than the implementation or the evaluation are the theoretical proofs, which could perhaps provide insights into the nature of linear optimisation itself.

8.1 Further Work

In this section, I present a number of directions for further work that would build upon the work presented in my thesis, along with my present state of thinking about these ideas.

8.1.1 LPs with irrational coefficients

My work is largely concerned with overapproximations. For instance, at each stage of my δ-complete simplex algorithm (Algorithm 4.1), a particular floating-point precision and tolerance are chosen – in essence, deciding on the level of overapproximation. And in subsequent steps, the level of overapproximation is reduced, until a sufficiently precise solution is obtained.

Irrational coefficients may be handled in a similar way, selecting rational coefficients that approximate the irrationals, in such a way as to make the problem an overapproximation. And then, in subsequent steps, new coefficients may be chosen – getting progressively closer to the irrational values, but always overapproximating. It would of course be natural to always use the closest floating-point values, at the current precision, to the irrational values, on whichever side is necessary to make the problem an overapproximation.

For instance, suppose we have a constraint $2a + \pi b \geqslant c$. Provided that (without loss of generality) we are using a method that uses only nonnegative variables, we can overapproximate this constraint with $2a + pb \geqslant c$, where the new constant coefficient $p \in \mathbb{Q} > \pi$. If we get a result of unsat, then, since $b \geqslant 0$, we know that $2a + pb \geqslant 2a + \pi b \geqslant c$, and hence the problem with π in place of p is also unsatisfiable, since it has a tighter constraint.

If, on the other hand, we get a result of δ-sat, then we have $2a + pb - c \geqslant -\delta$. However, in order to declare δ-sat, we need $2a + \pi b - c \geqslant -\delta$, which is a tighter constraint, since $pb \geqslant \pi b$. So we have two cases: either we are able to verify (perhaps using some arbitrarily chosen $p' < \pi$) that the tighter constraint holds, or we are not, in which case we proceed to the next precision/tolerance combination.

(Note that adapting this for Algorithm 4.1 requires the addition of a slack variable.)

The thorniest question here is whether this affects termination. This would require further work – but my intuitive sense is that it should not, since we are always overapproximating by δ, and hence there should always be an overlap between the cases that can trigger unsat and the cases that can trigger δ-sat, regardless of any irrational coefficients.

8.1.2 LPs with interval coefficients

Sometimes, there may be bounded measurement error in the problem coefficients, and then it may be felt necessary to encode the uncertainties using interval coefficients. In such a case, there is usually some unknown true value of the coefficient that lies within the given interval. Such intervals should be handled similarly to irrational coefficients. The basic idea is that you use one side of the interval for unsat and the other side for δ-sat. The side to use for unsat is the side that gives you the slacker constraint, since this makes the unsat a stronger statement (which hence implies unsat for any value within the interval). Similarly to the case of irrationals, this is the value you feed into the simplex solver (as a scalar floating-point value in the target precision, "widened" if necessary in the direction of greater slackness). And the side to use for δ-sat is the side that gives you a tighter constraint – making the δ-sat valid for any value within the interval (because it is then valid even for the tightest version of the constraint). Again, similarly to the case of irrationals, this can be checked after you have found a putative solution, and so this value does not need to be passed to the FP solver.

Termination would again need further attention. However, perhaps surprisingly, my intuition is that, in this case, it may not be possible to find a general proof – it may be that certain interval-based problems can not be solved if the delta is too small relative to the sizes of the intervals (but in a problem-dependent way).

8.1.3 Neural networks using alternative activation functions

In the introduction, I mentioned that my methods could be used to answer questions about neural networks with the ReLU activation function. Something to note here is that my work doesn't apply directly even to ReLU neural networks: it only applies to linearisations of them. As such, it could be applied to any network that can be linearised – albeit that a linearisation of a sigmoid neural network will only be an approximation, whereas a linearisation of a ReLU neural network can be exact over a given subset of the input space, due to the network's piecewise linearity. In the case of the sigmoid, it is likely not only that the linearisation will be inexact, but also that the subsets of the input space over which a given linearisation is sufficiently close (for any given application) will be smaller than in the ReLU case. However, it would be interesting to do a comparative analysis of the two types of network and their amenability to this sort of investigation technique.

Appendix A

Important Interior-Point Results from [2]

All results, equations and definitions in this chapter are copied verbatim from [2], and are used (directly or indirectly) in Chapter 2. To reduce confusion, I have labelled equations from [2] with (R-X.XX), where that book's label for them is (X.XX). I have also prefixed the numbers of theorems, definitions, etc. from [2] with "R-", turning Lemma I.1 into Lemma R-I.1, etc. I have not included proofs unless the text of the proof is required for my analysis.

After their introduction, all constants and variables may be assumed to retain their identities up to the end of this chapter.

A.1 Analytical Subject and Duality

The following two optimisation problems from [2, p. 18] form the main analytical subject of the book. They are also direct dependencies of Lemma R-I.1. It is assumed that $A \in \mathbb{R}^{m\times n}$, $c, x \in \mathbb{R}^n$, and $b, y \in \mathbb{R}^m$. A, b, c are constants while x, y are variables. (Note that this also implicitly introduces constants $m, n \in \mathbb{N}$.)

$$\text{(P)} \qquad \min\{c^\top x : Ax \geqslant b, x \geqslant 0\}, \qquad \text{(R-2.1)}$$

$$\text{(D)} \qquad \max\{b^\top y : A^\top y \leqslant c, y \geqslant 0\}. \qquad \text{(R-2.2)}$$

It is asserted that *(P)* is in *canonical form*. However, it is made clear that *(D)*, which is the *dual* of *(P)*, is *not* in canonical form.

Now we have weak and strong duality, respectively:

Lemma R-I.1 (Weak duality – from [2, p. 18]). *Let x be feasible for (P) and y for (D). Then*

$$b^\top y \leqslant c^\top x.$$

Note that *"x is feasible for (P)"* means that x is a member of the set of which (P) takes the minimum, and likewise for y and (D).

Theorem R-I.26 (Strong duality theorem, from [2, p. 39])**.** *For an LO problem (P) in canonical form and its dual problem (D) we have the following two alternatives:*

(i) Both (P) and (D) are solvable and there exist (strictly complementary) optimal solutions x for (P) and y for (D) such that $c^\top x = b^\top y$.

(ii) Neither (P) nor (D) is solvable.

Strictly complementary is defined as follows:

Definition R-I.18 (from [2, p. 35])**.** *Two nonnegative vectors a and b in $\mathbb{R}^n$ are said to be complementary vectors if $ab = 0$. If moreover $a + b > 0$ then a and b are called strictly complementary vectors.*

Note that ab is, by convention, the vector whose entries are obtained by multiplying u and v componentwise, as in Appendix A.6.

A.2 From Optimisation to Inequalities

The duality gap is defined in [2, p. 19] as follows:

$$c^\top x - b^\top y. \tag{R-2.4}$$

The solvability of the following inequality system from [2, p. 19] is shown there to be necessary and sufficient for (P) and (D) to have optimal solutions x, y with *vanishing duality gap* (i.e. such that $c^\top x - b^\top y = 0$):

$$\begin{aligned} Ax &\geqslant b, & x &\geqslant 0, \\ -A^\top y &\geqslant -c, & y &\geqslant 0, \\ b^\top y - c^\top x &\geqslant 0. \end{aligned} \tag{R-2.5}$$

The importance of the vanishing duality gap is captured by the following corollary (of Lemma R-I.1). Recall the definition of "feasible" introduced in the previous section.

Corollary R-I.2 (from [2, p. 18])**.** *If x is feasible for (P) and y for (D), and $c^\top x = b^\top y$, then x is optimal for (P) and y is optimal for (D).*

"Optimal solution" may seem redundant on its own, to one not versed in the optimisation literature, until you realise that they also speak of so-called "feasible solutions" that are members of the set being optimised, but not optima.

A.3 Skew-Symmetry and Homogeneity

The following definitions from [2, p. 20]:

$$\bar{M} := \begin{bmatrix} 0 & A & -b \\ -A^\top & 0 & c \\ b^\top & -c^\top & 0 \end{bmatrix}, \quad \bar{z} := \begin{bmatrix} y \\ x \\ \kappa \end{bmatrix}, \tag{R-2.7}$$

are used there to show that the problem of finding optimal solutions to (P) and (D) with vanishing duality gap can be reduced to finding a solution of the following inequality system (also from [2, p. 20]):

$$\bar{M}\bar{z} \geqslant 0, \quad \bar{z} \geqslant 0, \quad \kappa > 0. \tag{R-2.8}$$

Notice that the condition on $\kappa \in \mathbb{R}$ (a new variable) is a strict inequality, unlike the others.

I call this system the *homogeneous skew-symmetric inequality system*, because the matrix $\bar{M}$ is skew-symmetric, and the solution space is transformed into a homogeneous coordinate system by the introduction of κ.

A.4 The Interior-Point Condition

The interior-point condition is defined by:

> **Definition R-I.4** (IPC – from [2, p. 20]). *We say that any system of (linear) equalities and (linear) inequalities satisfies the interior-point condition (IPC) if there exists a feasible solution that strictly satisfies all inequality constraints in the system.*

The following three numbered "equations" (and one unnumbered equation) from [2, p. 21] are needed by Theorem R-I.5.

This first one effectively defines M, r, and z. Note that on the same page, it is indicated that, for instance, e_n denotes an all-one vector of length n (etc.). $\vartheta \in \mathbb{R}$ is a new variable; everything else is already defined.

$$M = \begin{bmatrix} \begin{bmatrix} 0 & A & -b \\ -A^\top & 0 & c \\ b^\top & -c^\top & 0 \end{bmatrix} & r \\ -r^\top & 0 \end{bmatrix}, \quad r = \begin{bmatrix} e_m - Ae_n + b \\ e_n + A^\top e_m - c \\ 1 - b^\top e_m + c^\top e_n \end{bmatrix}, \quad z = \begin{bmatrix} y \\ x \\ \kappa \\ \vartheta \end{bmatrix}. \tag{R-2.11}$$

And then we have a couple more definitions, both from [2, p. 21]:

$$\bar{n} = m + n + 2,$$

$$q := \begin{bmatrix} 0_{\bar{n}-1} \\ \bar{n} \end{bmatrix}, \tag{R-2.12}$$

followed by an new system of inequalities, also from [2, p. 21] – I call this the *IPC system*:

$$Mz \geqslant -q, \quad z \geqslant 0. \tag{R-2.13}$$

Finally, the following theorem ties together all the different ways of expressing the core problem up to this point:

Theorem R-I.5 (from [2, p. 22])**.** *The following three statements are equivalent:*

(i) Problems (P) and (D) have optimal solutions with vanishing duality gap;

(ii) If $\bar{M}$ and $\bar{z}$ are given by (R-2.7) then (R-2.8) has a solution;

(iii) If M and z are given by (R-2.11) then (R-2.13) has a solution with $\vartheta = 0$ and $\kappa > 0$.

Moreover, system (R-2.13) satisfies the IPC.

A.5 The Slack Vector

This equation, from [2, p. 22], defines the *slack vector* $s(z)$ in terms of the variable $z \in \mathbb{R}^n$, and constants M, q, all of which we have seen already:

$$s(z) := Mz + q. \tag{R-2.17}$$

Note that it is sometimes written simply as s.

And here is an important identity from [2, p. 24] involving $s(z)$:

$$q^\top z = z^\top(s(z) - Mz) = z^\top s(z) - z^\top Mz = z^\top s(z). \tag{R-2.23}$$

A.6 Self-Duality

This minimisation problem, from [2, p. 22], is called the *self-dual problem*:

$$(\mathrm{SP}) \qquad \min\{q^\top z : Mz \geqslant -q, z \geqslant 0\}. \tag{R-2.16}$$

The following excerpt from [2, p. 25] defines the elementwise product notation used within Lemma R-I.10 (and elsewhere):

> [...] In this lemma, and further on, we use the following notation. To any vector $u \in \mathbb{R}^k$, we associate the diagonal matrix U whose diagonal entries are the elements of u, in the same order. If also $v \in \mathbb{R}^k$, then Uv will be denoted

shortly as uv. Thus uv is a vector whose entries are obtained by multiplying u and v componentwise.

The following lemma is important in proving properties that are shared by *all* optimal solutions of (SP):

Lemma R-I.10 (from [2, p. 25])**.** *Let z^1 and z^2 be feasible for (SP). Then z^1 and z^2 are optimal solutions of (SP) if and only if $z^1 s(z^2) = z^2 s(z^1) = 0$.*

A.7 The Optimal Partition

The sets B and N are defined on [2, p. 24] as follows:

$$B := \{i : z_i > 0, \text{ for some optimal } z\},$$
$$N := \{i : s_i(z) > 0, \text{ for some optimal } z\}.$$

The following is a corollary of Lemma R-I.10:

Corollary R-I.11 (from [2, p. 25])**.** *The sets B and N are disjoint.*

The identity of B, N above as the *optimal partition* is only informally stated within the book, on [2, p. 36]:

> [...] We are going to show that there exists an optimal vector z such that z and $s(z)$ are strictly complementary vectors. Then for every index i, either $z_i > 0$ or $s_i(z) > 0$. This implies that the index sets B and N, introduced in Section 2.5 *[edit: this should read "Section 2.6"]* form a partition of the index set, the so-called optimal partition of *(SP)*.

The theorem (and proof) of the existence of this z is given further down the page, concluding the matter:

Theorem R-I.20 (from [2, p. 36])**.** *(SP) has an optimal solution z^* with $z^* + s(z^*) > 0$.*

A.8 Full-Newton Step Algorithm

Algorithm A.1 is the central algorithm of Part I of [2]. It is a polynomial-time algorithm. Although the complexity bounds are not good enough for a real-world implementation, they are good enough for the purposes of my proofs.

The computation of Δz is omitted – it is given by the following equation from [2, p. 49] (which makes use of both the diagonal matrix rule and the componentwise product rule described in Appendix A.6, as well as the identity of e as an all-one vector of the appropriate size):

$$\Delta z = (S + ZM)^{-1}(\mu e - zs). \tag{R-3.8}$$

Algorithm A.1: Full-Newton step algorithm – from [2, p. 50].

```
input  : An accuracy parameter $\varepsilon > 0$;
         a barrier update parameter $\theta$, $0 < \theta < 1$.
1 begin
2     $z = e$; $\mu := 1$;
3     while $n\mu \geqslant \varepsilon$ do
4         $\mu := (1 - \theta)\mu$;
5         $z := z + \Delta z$;
```

The initialisation and update of s is also omitted from the algorithm pseudocode. The initialisation is given by (R-2.17) – see Appendix A.5.

The following unnumbered equations from [2, p. 49] give updates for both z and s, calling the updated versions z^+ and s^+, which will be important later:

$$\begin{aligned} z^+ &:= z + \Delta z, \\ s^+ &:= s(z^+) = M(z + \Delta z) + q = s + M\Delta z. \end{aligned}$$

The inputs and outputs of the algorithm are also not made fully explicit (but the missing items can easily be inferred).

A.9 Termination

This lemma is refreshingly simple, as it only makes use of algorithm inputs and locals:

Lemma R-I.36 (from [2, p. 51])**.** *After at most*

$$\left\lceil \frac{1}{\theta} \log \frac{n}{\varepsilon} \right\rceil$$

iterations, we have $n\mu \leqslant \varepsilon$.

While it is true that Algorithm A.1 requires $n\mu < \epsilon$ for termination, this does not pose any difficulties, since n is a positive integer, and μ is positive and scaled at each iteration by a factor of $1 - \theta$, which is in $(0, 1)$.

A.10 Proximity Measure

The *variance vector* of z is defined on [2, p. 31] as:

$$v := \sqrt{\frac{zs(z)}{\mu}}. \qquad \text{(R-2.41)}$$

The *proximity measure* is then defined, also on [2, p. 31], as:

$$\delta(z, \mu) := \tfrac{1}{2}\|v - v^{-1}\|. \tag{R-2.42}$$

Finally, the following theorem relates the proximity measure of z^+ (the updated version of z) to that of z:

Theorem R-I.16 (from [2, p. 31])**.** *If $\delta := \delta(z, \mu) < 1$, then the Newton step is strictly feasible, i.e., $z^+ > 0$ and $s^+ > 0$. Moreover,*

$$\delta(z^+, \mu) \leqslant \frac{\delta^2}{\sqrt{2(1 - \delta^2)}}.$$

A.11 Condition Number

The condition number is defined on [2, p. 54] as

$$\sigma_{SP} := \min\{\sigma^z_{SP}, \sigma^s_{SP}\},$$

where (also from [2, p. 54]):

$$\sigma^z_{SP} := \min_{i \in B} \max_{z \in \mathcal{SP}^*} \{z_i\}, \quad \sigma^s_{SP} := \min_{i \in N} \max_{z \in \mathcal{SP}^*} \{s_i(z)\},$$

and where (B, N) is the optimal partition of (SP) (see Appendix A.7), and $\mathcal{SP}^*$ is the set of all optimal solutions of (SP).

Theorem R-I.42 (from [2, p. 56])**.** *The condition number σ_{SP} of (SP) satisfies*

$$\sigma_{SP} \geqslant \frac{1}{\prod_{j=1}^{n} \|M_j\|},$$

where M_j denotes the j-th column of M.

A.12 Iteration Bound

The distance of z to the central path is defined on [2, p. 59] as:

$$\delta_c(z) := \frac{\max(zs(z))}{\min(zs(z))}, \tag{R-3.20}$$

where max and min here operate across the elements of their vector argument, and $z > 0$ and $s(z) > 0$.

The following lemma is a prerequisite of Theorem R-I.47:

Lemma R-I.46 (from [2, p. 61])**.** *Let z be a feasible solution of (SP) such that*

$\delta_c(z) \leqslant \tau$. *If*

$$z^\top s(z) < \frac{{\sigma_{SP}}^2}{\tau n},$$

then the optimal partition of (SP) follows from

$$B = \{i : z_i > s_i(z)\} \text{ and } N = \{i : z_i < s_i(z)\}. \tag{R-3.22}$$

Finally, here is a theorem giving a bound on the number of iterations. I include the proof this time, because it contains an important result that I make use of.

Theorem R-I.47 (from [2, p. 61])**.** *After at most*

$$\left\lceil \sqrt{2n} \log \frac{4n^2}{{\sigma_{SP}}^2} \right\rceil \tag{R-3.23}$$

iterations, the Full-Newton step algorithm yields a feasible (and positive) solution z of (SP) that reveals the optimal partition (B,N) of (SP) according to (R-3.22)*.*

Proof. Let us run the Full-Newton step algorithm with $\varepsilon = \sigma_{SP}^2/(4n)$. Then Theorem R-I.37 states that we obtain a feasible z with $z^\top s(z) \leqslant \sigma_{SP}^2/(4n)$ and $\delta(z,\mu) \leqslant 1/2$. Lemma R-I.44 implies that $\delta_c(z) \leqslant 4$. By Lemma R-I.46, with $\tau = 4$, this z gives a complete separation between the small variables and the large variables. By Theorem R-I.37, the required number of iterations for the given ε is at most

$$\left\lceil \sqrt{2n} \log \frac{4n^2}{{\sigma_{SP}}^2} \right\rceil,$$

which is equal to the bound given in the theorem. Thus the proof is complete.

And, for completeness, here are the additional results used within that proof:

Theorem R-I.37 (from [2, p. 52])**.** *If $\theta = \frac{1}{\sqrt{2n}}$ then the algorithm requires at most*

$$\left\lceil \sqrt{2n} \log \frac{n}{\varepsilon} \right\rceil$$

iterations. The output is a feasible $z > 0$ such that $q^\top z = n\mu \leqslant \varepsilon$ and $\delta(z,\mu) \leqslant \frac{1}{2}$.

Lemma R-I.44 (from [2, p. 59])**.** *If $\delta(z,\mu) \leqslant \frac{1}{2}$ then $\delta_c(z) \leqslant 4$.*

A.13 Large and Small Variables

This lemma establishes that the iterates of variables (within z) whose index is in B of the optimal partition can be lower-bounded, while the iterates of variables whose index is in N can be upper bounded.

Lemma R-I.43 (from [2, p. 57]). *For any positive μ we have*

$$z_i(\mu) \geqslant \frac{\sigma_{SP}}{n}, \quad i \in B, \qquad z_i(\mu) \leqslant \frac{n\mu}{\sigma_{SP}}, \quad i \in N,$$
$$s_i(\mu) \leqslant \frac{n\mu}{\sigma_{SP}}, \quad i \in B, \qquad s_i(\mu) \geqslant \frac{\sigma_{SP}}{n}, \quad i \in N.$$

In [2, p. 58], it is shown that if

$$\mu < \frac{{\sigma_{SP}}^2}{n^2},$$

then the optimal partition may be inferred from the current iterate z, using the bounds from Lemma R-I.43. Needless to say, this is a very important result! For one thing, once the optimal partition (B, N) has been established, we can use the fact that all z_i such that $i \in N$ must be zero in the exact solution.

The following theorem, derived from the aforementioned result, establishes the convergence properties of ϑ/κ^2, and how this can help us to infer whether or not $\kappa = 0$ in the exact solution:

Theorem R-I.58 (from [2, p. 82]). *If κ is a large variable then*

$$\lim_{\mu \downarrow 0} \frac{\vartheta}{\kappa} = \lim_{\mu \downarrow 0} \frac{\vartheta}{\kappa^2} = 0,$$

and if κ is a small variable then

$$\lim_{\mu \downarrow 0} \frac{\vartheta}{\kappa^2} = \infty.$$

Although this theorem is presented in chapter 4, where alternative self-dual embeddings (other than *(SP)*) are considered, its proof (which is only informally stated in [2, pp. 81–82]) depends only on Lemma R-I.43, from the previous chapter, where only *(SP)* is used, and the fact that $\mu = \vartheta$ on the central path. This fact, which is shown on [2, p. 80], depends only on (R-4.11), (R-2.23), (R-2.11), and the definition of q on [2, p. 78]. However, noting that this definition of q is the same as in (R-2.12), and (R-4.11) is the same as (R-2.46), meaning that we can eliminate all dependencies from chapter 4, we see that $\mu = \vartheta$ (on the central path) must be equally valid for *(SP)*.

In any case, it can easily be shown that (SP_2) is in fact a generalisation of (SP) – it may be converted into the latter by setting the per-instance constants $x^0 := e_n$, $y^0 := e_m$, $s^0 := e_n$, and $t^0 := e_m$ – see Appendix A.14.

A.14 Modified Form of the Feasibility Conditions

The following re-expression of the feasibility conditions of (SP_2) is from [2, p. 82]:

$$\begin{array}{rcl} A\tilde{x} & \geqslant & b - \frac{\vartheta}{\kappa}\bar{b} \\ A^\top\tilde{y} & \leqslant & c + \frac{\vartheta}{\kappa}\bar{c} \\ c^\top\tilde{x} - b^\top\tilde{y} & \leqslant & \frac{\vartheta}{\kappa}\beta \\ \kappa(\bar{b}^\top\tilde{x} + \bar{c}^\top\tilde{y} + \beta) & \leqslant & n + m + 2. \end{array} \tag{R-4.16}$$

where $\kappa > 0$ – which is true for any non-terminal iterate of an interior-point algorithm. And where (from [2, pp. 78–79])

$$\begin{aligned} \bar{b} &= t^0 + b - Ax^0, \\ \bar{c} &= s^0 - c + A^\top y^0, \\ \beta &= 1 - b^\top y^0 + c^\top x^0. \end{aligned}$$

where (on [2, p. 78]) x^0, y^0 are defined to be arbitrary (constant) positive vectors of dimension n and m respectively, and positive vectors s^0 and t^0 are defined (also on [2, p. 78]) by

$$x^0 s^0 = e_n, \quad y^0 t^0 = e_m.$$

(SP_2), defined on [2, p. 78] is given by

$$(SP_2) \qquad \min\{q^\top z : Mz + q \geqslant 0, z \geqslant 0\},$$

with (also from [2, p. 78])

$$M := \begin{bmatrix} 0_{mm} & A & -b & \bar{b} \\ -A^\top & 0_{nn} & c & \bar{c} \\ b^\top & -c^\top & 0 & \beta \\ -\bar{b}^\top & -\bar{c}^\top & -\beta & 0 \end{bmatrix}, \quad q := \begin{bmatrix} 0_m \\ 0_n \\ 0 \\ n+m+2 \end{bmatrix},$$

using the above definitions of $\bar{b}, \bar{c}, \beta$.

Since (SP_2) is identical to (SP) apart from the definition of M (see (R-2.16) and (R-2.12)), and the definition of M used for (SP_2) differs from that used for (SP) only in the use of $[\bar{b}\ \bar{c}\ \beta\ 0]^\top$ in place of r (see (R-2.11)), and this vector differs from r only in the use of t^0, x^0, s^0, y^0 in place of e_m, e_n, e_n, e_m, respectively (again see (R-2.11), and compare with the definitions of $\bar{b}, \bar{c}, \beta$ above), we may transform (SP_2) into (SP) simply by setting $x^0 := e_n$ and $y^0 := e_m$ (with s^0 and t^0 following suit due to the their definitions given above).

This means that the results in [2] pertaining to (SP_2) may also be applied to (SP) – including the re-expression of the feasibility conditions in the form (R-4.16).

Appendix B

Alternative Interior-Point Proofs

B.1 Simple Cases

Theorem 2.1. *Systems (2.5) and (2.6) are equisatisfiable.*

Proof. If a solution to (2.6) is found, we have $\kappa > 0$, and so $A(x/\kappa) - b \geqslant 0$, $-A^\top y \geqslant 0$ and $b^\top y \geqslant 0$, so that $(x/\kappa, y)$ satisfies (2.5). Conversely, if (x, y) satisfies (2.5), then $(x, y, 1)$ satisfies (2.6). □

Theorem 2.2. *From any optimal solution to (2.7) such that $\kappa > 0$, we can derive a solution to (2.6), and vice-versa.*

Proof. $z = 0$ is clearly feasible for (2.7), and therefore, because $q \geqslant 0$, optimal. So the optimal value is 0. This means that $z_n = 0$ (i.e. $\vartheta = 0$) in any optimal solution of (2.7), and therefore we can derive $\bar{M}\bar{z} \geqslant 0$. This means that any solution of (2.7) in which $\kappa > 0$ satisfies (2.6).

Conversely, if $\bar{z}$ satisfies (2.6), then either $r^\top \bar{z} \leqslant n$, in which case $(\bar{z}, 0)$ is optimal for (2.7), or else $(\nu\bar{z}, 0)$ is optimal for (2.7) for any $0 < \nu \leqslant n/r^\top \bar{z}$ (and of course $r^\top \bar{z} > n > 0$). The condition $\kappa > 0$, if present, is clearly preserved in either case. □

B.2 The Unsatisfiable Case

According to Theorem I.20 in [2] (p. 36), the problem (2.7) has a strictly complementary solution: this is a solution in which out of every variable and its corresponding dual variable, one is equal to zero and the other is positive (in some contexts, these are referred to as "small" and "large" variables, respectively). Combining this result with Corrolary I.11 in [2] (p. 25), we conclude that for every matching pair of primal and dual variables in a strictly complementary solution, the variable that is zero will in fact be zero in every optimal solution.

In Section 3.3.4 of [2] (pp. 58–62), an algorithm is developed for finding the "optimal partition", which tells us the identities of the zero and positive variables in a strictly complementary solution (noting that any strictly complementary solution will have the same partition), without the need to find an exact solution.

These concepts can be used to prove the following theorem:

Theorem B.1. *A strictly complementary solution to* (2.7) *such that* $\kappa = 0$ *exists (or, equivalently,* κ *is a "small" variable in the optimal partition) if and only if* (2.6) *is unsatisfiable.*

Proof. If (2.5) is unsatisfiable, then a strictly complementary solution to (2.7) will necessarily have $\kappa = 0$ (as otherwise (2.5) would be satisfiable, according to Theorems 2.2 and 2.1). Because of strict complementarity (or because we know the optimal partition), we then know that κ can never be positive in any optimal solution to (2.7). However, any solution to (2.5) would give us a solution to (2.7) with $\kappa > 0$, as proven in Theorems 2.1 and 2.2. □

Appendix C

Elementwise Bounds for the Result of Gaussian Elimination with Partial Pivoting

This appendix is concerned with bounding the elements of the factors of an approximate *LU* decomposition of a matrix $A \in \mathbb{R}^{n\times n}$ where partial pivoting is used. This is commonly expressed in terms of the *growth factor* [22, pp. 165–166], which is defined, for $A = LU$, as

$$\rho_n := \frac{\max_{i,j,k} |A^k_{ij}|}{\max_{i,j} |A_{ij}|}, \tag{C.1}$$

where A^k is the matrix produced after $k-1$ iterations of Gaussian elimination (so that $U = A^n$, there being $n-1$ iterations in total, and we can define $A^1 := A$). Hence, $|U_{ij}| \leqslant \rho_n \max_{p,q} |A_{pq}|$ provides the desired bound. Most discussions of the growth factor fail to mention the effect of the use of floating-point arithmetic; hence, the need for this appendix.

C.1 Partial Pivoting

Partial pivoting means rearranging the rows of the matrix A^k at each iteration of Gaussian elimination so that the pivot element is maximal, preventing the production of large numbers that may cause loss of precision [21, p. 127]. With exact arithmetic, this would ensure that the elements of $|L|$ are bounded by 1. Since L is the matrix of multipliers from Gaussian elimination [21, p. 115], and these are formed by dividing two numbers p/q such that (when using partial pivoting) $|p| \leqslant |q|$ [21, pp. 113–114, 127], we know (because of the way floating-point works) that we must have $|p/q| \leqslant 1$. However, this is not guaranteed by the floating-point model of Section 1.2.1. Using that model, we have $|p/q| \leqslant 1 + \epsilon$; hence,

$$|L_{ij}| \leqslant 1 + \epsilon, \qquad \text{for } i > j, \text{ and hence for all } i, j. \tag{C.2}$$

C.2 Bound for General Case

In the general case, where partial pivoting is used, the growth factor is $\rho_n \leqslant 2^{n-1}$ [22, p. 166]. This is because, in the worst case, each element in each A^k is equal to $a - (p/q)b$ where a, b, p, q are all elements of A^{k-1}, while we know that $p/q \leqslant 1$ by partial pivoting. Hence, at each iteration, the elements are at worst doubled in magnitude. If floating-point arithmetic is used, then we must consider the error from the division, multiplication and subtraction, so that $|a - (p/q)b| \leqslant (1+\epsilon)(|a| + (1+\epsilon)^2|b|) \leqslant 2(1+\epsilon)^3 \max(|a|, |b|)$. Hence, the worst-case multiplier at each iteration is $2(1+\epsilon)^3$ instead of 2, so that we have $\rho_n \leqslant (2(1+\epsilon)^3)^{n-1}$, and

$$|U_{ij}| \leqslant (2(1+\epsilon)^3)^{n-1} \max_{p,q} |A_{pq}|, \qquad \text{for } i \leqslant j, \text{ and hence for all } i, j. \tag{C.3}$$

C.3 Bound for Upper Hessenberg Case

If A is upper Hessenberg, and partial pivoting is used, then the growth factor is $\rho_n \leqslant n$ [22, p. 172]. This is because at each iteration k, the only option is to swap rows k and $k+1$ – the remaining rows in the lower part of A^k being zero in column k. Also, this means that only row k or $k+1$ is modified at iteration k (becoming row $k+1$). This means that every computation $a - (p/q)b$ of an element that is modified in iteration k involves one element that has been modified in some previous iteration (from the former row k), and one that has not been modified in any previous iteration (from the former row $k+1$).

Hence, if floating-point arithmetic is used, while we still have $|a - (p/q)b| \leqslant (1+\epsilon)^3(|a| + |b|)$, we know that either a or b is directly from A, so there is no exponential growth – except for that brought about by the $1+\epsilon$ factors. This clearly leads to a growth factor of $\rho_n \leqslant n(1+\epsilon)^{3(n-1)}$, so that

$$|U_{ij}| \leqslant n(1+\epsilon)^{3(n-1)} \max_{p,q} |A_{pq}|, \qquad \text{for } i \leqslant j, \text{ and hence for all } i, j. \tag{C.4}$$

Appendix D

Full LP: full results table

LP	esolver-do	esolver	soplex
10teams	0.2	0.1	0.1
16_n14	2562.9	2781.2	758.6
22433	0.1	0.0	0.0
23588	0.0	0.0	0.0
25fv47	0.8	0.9	1.0
30_70_45_095_100	22.0	17.8	11.7
30n20b8	0.4	0.4	0.6
50v-10	0.1	0.1	0.1
80bau3b	0.6	0.6	1.1
a1c1s1	0.3	0.3	0.1
aa01	5.9	6.0	3.5
aa03	6.0	6.1	2.6
aa3	4.2	4.0	2.6
aa4	1.2	1.2	0.9
aa5	2.8	3.0	2.7
aa6	1.9	1.9	1.0
acc-tight4	2.5	2.6	2.4
acc-tight5	2.4	2.7	1.7
acc-tight6	2.5	2.4	1.6
adlittle	0.0	0.0	0.0
afiro	0.0	0.0	0.0
aflow30a	0.0	0.0	0.0
aflow40b	0.1	0.1	0.1
agg2	0.0	0.0	0.0
agg3	0.0	0.0	0.0
agg	0.0	0.0	0.0
air02	0.2	0.2	0.2
air03	0.5	0.4	0.4

LP	esolver-do	esolver	soplex
air04	5.5	5.9	3.3
air05	1.2	1.2	1.0
air06	6.2	6.0	2.5
aircraft	0.5	0.5	0.4
aligninq	0.1	0.1	0.1
app1-2	44.0	39.4	16.5
arki001	1.6	1.6	0.4
ash608gpia-3col	18.6	14.1	1.8
atlanta-ip	3161.2	4050.4	60.3
atm20-100	1.8	1.8	0.7
b2c1s1	0.6	0.7	0.1
bab1	63.4	68.7	223.1
bab3	360.9	396.2	8.9
bab5	3.5	3.2	11.4
bal8x12	0.0	0.0	0.0
bandm	0.1	0.1	0.1
bas1lp	2.9	2.6	3.1
baxter	4.9	4.1	12.6
bc1	3.4	3.1	2.9
bc	3.1	3.1	2.7
beaconfd	0.0	0.0	0.0
beasleyC3	0.1	0.1	0.1
bell3a	0.0	0.0	0.0
bell5	0.0	0.0	0.0
berlin_5_8_0	0.1	0.1	0.1
bg512142	0.2	0.3	0.2
biella1	3.2	3.2	4.8
bienst1	0.1	0.1	0.0
bienst2	0.1	0.1	0.0
binkar10_1	0.1	0.1	0.1
bk4x3	0.0	0.0	0.0
blend2	0.0	0.0	0.0
blend	0.0	0.0	0.0
bley_xl1	5187.6	3944.2	16513.9
blp-ar98	0.9	0.8	1.0
blp-ic97	0.5	0.5	0.7
bnatt350	0.1	0.1	0.2
bnatt400	0.2	0.2	0.2
bnl1	0.2	0.2	0.1

LP	**esolver-do**	**esolver**	**soplex**
bnl2	0.5	0.5	0.5
boeing1	0.0	0.0	0.0
boeing2	0.0	0.0	0.0
bore3d	0.0	0.0	0.0
brandy	0.0	0.1	0.0
buildingenergy	1680.4	1638.7	2656.5
cap6000	0.2	0.2	0.4
capri	0.0	0.0	0.0
car4	5.5	5.9	5.6
cari	0.6	0.6	0.6
cdma	42297.4	39670.2	3059.8
cep1	0.1	0.1	0.1
ch	1.3	1.6	2.2
circ10-3	14.9	16.5	21.3
co-100	11.4	10.3	11.4
co5	8.8	9.6	5.4
co9	63.4	69.1	29.3
complex	5.4	4.7	3.0
cont11_l	17.2	12.8	26.8
cont11	10388.2	29800.4	*0.0*
cont1	29487.6	29683.6	7955.2
cont4	10814.1	11169.9	25275.8
core2536-691	10.3	10.3	22.0
core4872-1529	128.6	138.0	137.6
cov1075	0.4	0.1	0.2
cq5	5.9	5.3	4.7
cq9	45.6	41.8	13.9
cr42	0.1	0.1	0.1
cre-a	0.4	0.5	0.2
cre-b	20.9	20.6	7.5
cre-c	0.4	0.4	0.2
cre-d	20.1	22.8	4.5
crew1	0.2	0.2	0.5
csched007	0.1	0.1	0.2
csched008	0.1	0.1	0.1
csched010	0.1	0.1	0.2
cycle	0.3	0.3	0.2
czprob	0.2	0.2	0.2
d10200	0.5	0.5	0.5

LP	**esolver-do**	**esolver**	**soplex**
d20200	2.5	2.3	2.7
d2q06c	58.5	58.8	6.6
d6cube	4.6	4.5	0.4
dano3_3	63.8	55.3	27.2
dano3_4	53.2	57.6	26.2
dano3_5	57.7	55.6	28.4
dano3mip	64.5	60.7	30.5
danoint	0.2	0.2	0.2
datt256	*0.0*	*0.0*	31682.9
dbic1	275.9	227.3	*0.0*
dbir1	6.7	5.9	5.0
dbir2	5.3	6.0	8.0
dc1c	10.4	10.1	7.1
dc1l	69.6	54.6	36.7
dcmulti	0.0	0.0	0.0
de063155	2.1	2.0	0.5
de063157	0.1	1.2	*0.0*
de080285	1.1	1.2	0.2
degen2	0.1	0.1	0.1
degen3	0.7	0.8	0.6
degme	15.9	14.2	31.8
delf000	1.1	2.3	0.7
delf001	0.8	1.6	0.8
delf002	0.8	2.2	0.9
delf003	0.9	2.2	1.0
delf004	1.0	2.4	1.0
delf005	0.8	2.1	1.1
delf006	2.0	4.7	1.2
delf007	1.2	2.9	1.2
delf008	1.3	3.1	1.1
delf009	1.4	3.8	1.2
delf010	1.2	2.7	1.2
delf011	1.2	2.4	1.2
delf012	1.2	2.8	1.2
delf013	1.2	3.1	1.1
delf014	1.0	2.2	1.2
delf015	1.0	2.4	1.0
delf017	1.0	2.5	1.0
delf018	0.9	1.8	1.0

LP	**esolver-do**	**esolver**	**soplex**
delf019	0.9	1.8	1.0
delf020	1.1	2.1	1.0
delf021	1.2	2.2	1.0
delf022	1.0	2.1	0.9
delf023	1.2	2.3	1.5
delf024	1.3	3.3	1.5
delf025	0.9	2.2	1.1
delf026	0.9	2.3	1.2
delf027	0.9	1.9	1.1
delf028	0.8	2.0	1.2
delf029	0.9	2.0	1.2
delf030	1.3	2.4	1.2
delf031	0.8	2.1	0.9
delf032	0.9	2.1	1.2
delf033	0.8	1.9	0.9
delf034	0.9	2.2	1.1
delf035	1.0	2.2	1.0
delf036	0.8	2.2	1.0
deter0	0.3	0.4	0.2
deter1	1.9	1.9	1.2
deter2	3.2	2.9	0.8
deter3	3.1	3.1	1.2
deter4	1.1	1.0	0.4
deter5	1.8	1.9	0.8
deter6	1.3	1.3	0.9
deter7	3.2	2.9	1.0
deter8	0.9	0.9	0.5
df2177	0.8	0.8	0.6
dfl001	85.5	109.0	159.8
dfn-gwin-UUM	0.0	0.0	0.0
dg012142	1.0	1.1	2.0
disctom	4.3	4.1	1.7
disp3	0.1	0.1	0.1
dolom1	21.7	25.9	9.3
ds-big	1168.3	1274.5	9.6
dsbmip	0.1	0.1	0.2
ds	62.5	61.9	33.7
e18	17.4	14.0	33.0
e226	0.0	0.0	0.0

LP	**esolver-do**	**esolver**	**soplex**
egout	0.0	0.0	0.0
eil33-2	0.2	0.2	0.2
eilA101-2	7.8	8.2	39.2
eilB101	0.2	0.2	0.3
enigma	0.0	0.0	0.0
enlight13	0.0	0.0	0.0
enlight14	0.0	0.0	0.0
enlight15	0.0	0.0	0.0
enlight16	0.0	0.0	0.0
enlight9	0.0	0.0	0.0
etamacro	0.3	0.3	0.1
ex1010-pi	31.5	26.4	14.0
ex10	28910.2	28310.3	2285.5
ex3sta1	122.5	127.5	106.5
ex9	4807.5	6173.5	315.6
f2000	176.0	199.8	112.4
farm	0.0	0.0	0.0
fast0507	10.6	10.9	13.6
fffff800	0.0	0.0	0.1
fiball	2.0	2.1	1.6
fiber	0.0	0.0	0.0
finnis	0.0	0.0	0.1
fit1d	0.1	0.1	0.1
fit1p	0.1	0.1	0.2
fit2d	5.1	5.2	2.2
fit2p	5.1	5.0	3.9
fixnet6	0.0	0.0	0.0
flugpl	0.0	0.0	0.0
fome11	190.8	264.8	1927.3
fome12	457.5	560.4	1234.6
fome13	4509.6	4679.1	6210.5
fome20	36.6	49.5	37.8
fome21	112.7	125.5	197.8
forplan	0.0	0.0	0.1
fxm2-16	0.7	0.7	0.8
fxm2-6	0.2	0.2	0.2
fxm3_16	0.5	0.5	72.0
fxm3_6	0.1	0.1	1.8
fxm4_6	0.4	0.4	16.6

LP	**esolver-do**	**esolver**	**soplex**
g200x740i	0.1	0.0	0.0
gams10a	0.0	0.0	0.0
gams30a	0.0	0.0	0.0
ganges	0.1	0.1	0.1
gen1	9340.7	11176.2	157.8
gen2	7501.8	7636.6	5347.6
gen4	*0.0*	*0.0*	7308.3
gen	0.1	0.0	0.0
ge	8.1	8.9	7.4
ger50_17_trans	1.2	1.1	2.9
germanrr	2.8	2.6	3.6
germany50-DBM	0.6	0.6	1.0
gesa2_o	0.1	0.1	0.1
gesa2-o	0.1	0.1	0.1
gesa2	0.1	0.1	0.1
gesa3_o	0.1	0.1	0.1
gesa3	0.1	0.1	0.1
gfrd-pnc	0.1	0.0	0.1
glass4	0.0	0.0	0.0
gmu-35-40	0.1	0.1	0.0
gmu-35-50	0.1	0.1	0.1
gmut-75-50	34.6	32.7	15.1
gmut-77-40	4.2	4.2	2.5
go19	0.3	0.3	0.1
gr4x6	0.0	0.0	0.0
greenbea	3.5	3.6	4.1
greenbeb	3.4	3.3	3.0
grow15	2.3	2.3	0.6
grow22	0.8	0.9	0.7
grow7	0.2	0.2	0.2
gt2	0.0	0.0	0.0
hanoi5	2.3	1.9	1.8
haprp	0.1	0.1	0.1
harp2	0.1	0.1	0.1
ic97_potential	0.0	0.0	0.0
iiasa	0.1	0.1	0.1
iis-100-0-cov	0.3	0.4	0.4
iis-bupa-cov	0.7	0.7	1.2
iis-pima-cov	1.4	1.4	1.2

LP	**esolver-do**	**esolver**	**soplex**
i_n13	88.4	117.7	3693.5
in	18.9	17.1	17.5
israel	0.0	0.0	0.0
ivu06-big	32.2	31.4	66.0
ivu52	272.5	230.0	156.9
janos-us-DDM	0.1	0.0	0.1
jendrec1	5.5	5.8	5.0
k16x240	0.0	0.0	0.0
karted	*0.0*	*0.0*	*0.0*
kb2	0.0	0.0	0.0
ken-07	0.2	0.1	0.2
ken-11	3.3	3.7	4.1
ken-13	43.3	47.9	27.5
ken-18	874.0	1025.1	489.2
kent	1.8	2.0	2.6
khb05250	0.0	0.0	0.0
kl02	0.9	0.7	1.1
kleemin3	0.0	0.0	0.0
kleemin4	0.0	0.0	0.0
kleemin5	0.0	0.0	0.0
kleemin6	0.0	0.0	0.0
kleemin7	0.0	0.0	0.0
kleemin8	0.0	0.0	0.0
l152lav	0.1	0.1	0.1
L1_d10_40	*0.0*	*0.0*	*0.0*
l30	271.1	29597.7	*0.0*
l9	0.2	0.2	0.1
large000	1.6	3.5	1.6
large001	1.0	2.5	2.7
large002	2.3	5.5	2.0
large003	1.9	5.0	1.9
large004	1.6	4.9	2.1
large005	1.4	3.5	1.9
large006	1.7	4.3	1.8
large007	1.7	4.1	1.9
large008	1.7	4.6	1.9
large009	2.3	5.7	2.1
large010	1.8	4.5	1.6
large011	1.6	4.0	1.9

LP	**esolver-do**	**esolver**	**soplex**
large012	1.8	4.3	1.8
large013	1.5	3.6	1.7
large014	1.5	3.7	1.7
large015	1.6	4.1	2.1
large016	1.8	4.6	2.0
large017	1.5	3.5	1.9
large018	1.4	3.1	1.9
large019	1.4	2.6	1.8
large020	1.7	3.4	1.9
large021	1.5	3.4	1.8
large022	1.6	3.9	1.5
large023	1.6	3.6	1.8
large024	1.9	4.3	2.7
large025	2.4	5.8	1.9
large026	1.7	4.5	1.7
large027	1.5	3.2	1.6
large028	1.7	4.3	2.2
large029	1.8	4.4	2.0
large030	1.7	3.8	1.7
large031	2.1	4.7	2.0
large032	3.3	7.0	1.9
large033	1.4	3.3	1.9
large034	2.9	6.6	2.3
large035	3.2	7.3	2.6
large036	2.9	6.2	1.5
lectsched-1-obj	31.4	40.1	4.1
lectsched-1	29.5	30.3	3.8
lectsched-2	9.2	7.0	1.8
lectsched-3	19.5	17.5	3.2
lectsched-4-obj	0.7	0.6	0.6
leo1	0.6	0.6	0.7
leo2	1.3	1.3	4.3
Linf_520c	21481.8	18215.1	*0.0*
liu	0.1	0.1	0.1
lo10	47.7	67.4	13812.6
long15	16359.8	17577.4	5546.3
lotfi	0.0	0.0	0.0
lotsize	0.1	0.1	0.1
lp22	75.6	55.7	70.1

LP	**esolver-do**	**esolver**	**soplex**
lpl1	11754.3	9255.0	5620.1
lpl2	0.2	0.2	0.3
lpl3	1.8	1.7	2.0
lrn	5.3	5.5	4.8
lrsa120	2.8	2.5	6.1
lseu	0.0	0.0	0.0
m100n500k4r1	0.0	0.0	0.0
macrophage	0.1	0.1	0.1
manna81	0.2	0.2	0.2
map06	1364.4	1174.8	106.7
map10	1183.6	1545.7	81.2
map14	1563.7	1194.8	117.3
map18	1143.6	1220.2	62.2
map20	1141.7	1307.8	56.0
markshare1	0.0	0.0	0.0
markshare2	0.0	0.0	0.0
markshare_5_0	0.0	0.0	0.0
maros	0.2	0.3	0.2
maros-r7	710.7	743.5	54.9
mas74	0.0	0.0	0.0
mas76	0.0	0.0	0.0
maxgasflow	0.6	0.6	0.7
mc11	0.1	0.1	0.1
mcf2	0.2	0.2	0.2
mcsched	0.3	0.3	0.2
methanosarcina	3.5	3.5	0.3
mik-250-1-100-1	0.0	0.0	0.0
mine-166-5	0.8	0.8	0.5
mine-90-10	0.7	0.7	0.4
mining	42030.3	36683.9	16.9
misc03	0.0	0.0	0.0
misc06	0.2	0.2	0.1
misc07	0.0	0.0	0.0
mitre	0.3	0.3	0.4
mkc1	0.1	0.1	0.1
mkc	0.1	0.1	0.1
mod008	0.0	0.0	0.0
mod010	0.1	0.1	0.1
mod011	0.4	0.4	0.9

LP	esolver-do	esolver	soplex
mod2	499.1	556.6	274.9
model10	1500.6	1734.6	35.0
model11	4.0	4.0	1.5
model1	0.0	0.1	0.0
model2	0.1	0.2	0.5
model3	2.1	2.2	1.4
model4	4.4	4.7	3.2
model5	4.6	4.7	3.7
model6	21.2	22.0	3.9
model7	181.9	190.9	6.0
model8	0.2	0.1	0.5
model9	6.0	6.4	3.8
modglob	0.0	0.0	0.0
modszk1	0.1	0.1	0.1
momentum1	25.2	21.0	3.3
momentum2	44.0	74.0	41.4
momentum3	2392.9	5060.8	853.9
msc98-ip	1054.9	1065.9	3.6
mspp16	48.4	43.2	92.3
multi	0.0	0.0	0.0
mzzv11	20.7	18.8	46.2
mzzv42z	2.6	2.2	12.9
n15-3	242.5	256.4	161.4
n3-3	0.8	0.7	0.7
n3700	1.1	1.1	1.4
n3701	1.1	1.0	1.4
n3702	1.2	1.1	1.3
n3703	1.2	1.3	1.5
n3704	1.1	1.1	1.2
n3705	1.1	1.0	1.4
n3706	1.1	1.0	1.3
n3707	1.2	1.1	1.4
n3708	1.1	1.1	1.5
n3709	1.1	1.1	1.8
n370a	1.1	1.1	1.9
n370b	1.1	1.1	1.7
n370c	1.1	1.1	1.1
n370d	1.2	1.1	1.5
n370e	1.2	1.1	1.2

LP	**esolver-do**	**esolver**	**soplex**
n3div36	1.3	1.3	1.7
n3seq24	47.0	49.9	22.1
n4-3	0.2	0.2	0.1
n9-3	0.7	0.6	0.4
nag	1.4	1.2	0.4
nb10tb	912.4	841.8	61661.5
nemsafm	0.0	0.0	0.1
nemscem	0.1	0.1	0.1
nemsemm1	5.4	6.7	11.1
nemsemm2	2.6	2.8	3.8
nemspmm1	7.6	8.1	5.4
nemspmm2	28.5	28.7	5.7
nemswrld	662.3	587.7	71.5
neos-1053234	0.1	0.1	0.1
neos-1053591	0.0	0.0	0.0
neos-1056905	0.0	0.0	0.0
neos-1058477	0.1	0.1	0.1
neos-1061020	49.5	55.1	10.0
neos-1062641	0.0	0.3	0.1
neos-1067731	25.8	28.0	1.9
neos-1096528	49.9	38.6	14.3
neos-1109824	0.6	0.5	0.5
neos-1112782	0.1	0.1	0.1
neos-1112787	0.1	0.1	0.1
neos-1120495	0.5	0.3	0.3
neos-1121679	0.0	0.0	0.0
neos-1122047	16.3	16.5	2.0
neos-1126860	9.7	11.2	1.4
neos-1140050	2017.2	2114.5	110.8
neos-1151496	0.2	0.2	0.4
neos-1171448	1.4	1.2	1.2
neos-1171692	0.3	0.3	0.2
neos-1171737	0.3	0.3	0.4
neos-1173026	0.1	0.1	0.0
neos-1200887	0.1	0.1	0.0
neos-1208069	0.3	0.3	0.5
neos-1208135	0.3	0.2	0.4
neos-1211578	0.0	0.0	0.0
neos-1215259	0.7	0.7	0.5

LP	esolver-do	esolver	soplex
neos-1215891	0.8	0.7	1.2
neos-1223462	0.4	0.4	4.2
neos-1224597	0.2	0.2	1.8
neos-1225589	0.0	0.0	0.0
neos-1228986	0.0	0.0	0.0
neos-1281048	0.1	0.1	0.1
neos-1311124	0.1	0.0	0.1
neos-1324574	1.2	1.2	0.6
neos-1330346	0.4	0.4	0.3
neos-1330635	0.1	0.1	0.1
neos-1337307	1.9	1.8	1.7
neos-1337489	0.0	0.0	0.0
neos-1346382	0.0	0.0	0.0
neos-1354092	68.4	70.3	11.2
neos-1367061	147.7	114.7	59.4
neos-1396125	0.5	0.5	0.4
neos13	9.6	9.1	17.5
neos-1407044	323.9	301.9	91.4
neos-1413153	0.1	0.2	0.2
neos-1415183	0.2	0.2	0.2
neos-1417043	9.3	8.1	69.5
neos-1420205	0.0	0.1	0.2
neos-1420546	133.0	123.0	65.7
neos-1420790	0.8	0.9	2.6
neos-1423785	20.4	18.9	9.8
neos-1425699	0.0	0.0	0.0
neos-1426635	0.0	0.0	0.0
neos-1426662	0.1	0.0	0.0
neos-1427181	0.1	0.0	0.1
neos-1427261	0.1	0.1	0.1
neos-1429185	0.0	0.0	0.0
neos-1429212	239.1	337.4	2044.8
neos-1429461	0.0	0.0	0.0
neos-1430701	0.0	0.0	0.0
neos-1430811	300.8	292.9	3166.4
neos-1436709	0.1	0.0	0.0
neos-1436713	0.1	0.1	0.1
neos-1437164	0.1	0.1	0.1
neos-1439395	0.0	0.0	0.0

LP	esolver-do	esolver	soplex
neos-1440225	0.2	0.2	0.1
neos-1440447	0.0	0.0	0.0
neos-1440457	0.1	0.0	0.1
neos-1440460	0.0	0.0	0.0
neos-1441553	0.1	0.1	0.1
neos-1442119	0.1	0.0	0.0
neos-1442657	0.0	0.0	0.0
neos-1445532	0.4	0.4	1.8
neos-1445738	0.5	0.5	3.7
neos-1445743	0.6	0.5	4.0
neos-1445755	0.5	0.5	3.6
neos-1445765	0.5	0.5	3.8
neos-1451294	1.1	1.1	1.4
neos-1456979	0.4	0.4	0.4
neos-1460246	0.0	0.0	0.0
neos-1460265	0.1	0.1	0.1
neos-1460543	0.6	0.6	1.6
neos-1460641	0.2	0.2	0.6
neos-1461051	0.1	0.1	0.2
neos-1464762	0.2	0.2	1.3
neos-1467067	0.0	0.0	0.0
neos-1467371	0.2	0.2	1.2
neos-1467467	0.2	0.2	0.7
neos-1480121	0.0	0.0	0.0
neos-1489999	0.1	0.1	0.1
neos-1516309	0.1	0.1	0.1
neos-1582420	1.0	1.1	0.5
neos-1593097	0.5	0.7	0.8
neos-1595230	0.1	0.1	0.0
neos-1597104	4.6	3.6	2.8
neos-1599274	0.2	0.2	0.2
neos15	0.0	0.0	0.0
neos-1601936	5.1	4.8	4.9
neos-1603512	0.1	0.1	0.1
neos-1603518	0.4	0.4	0.3
neos-1603965	16.8	41.4	255.9
neos-1605061	13.2	10.8	10.9
neos-1605075	10.2	7.4	9.4
neos-1616732	0.1	0.0	0.0

LP	esolver-do	esolver	soplex
neos-1620770	0.5	0.4	0.1
neos-1620807	0.1	0.0	0.0
neos-1622252	0.3	0.3	0.1
neos16	0.1	0.0	0.0
neos18	0.7	0.6	0.2
neos1	251.9	263.5	26.9
neos2	492.6	408.5	54.1
neos3	39832.7	26730.2	13278.9
neos-430149	0.0	0.0	0.0
neos-476283	44.3	38.5	17.4
neos-480878	0.2	0.2	0.3
neos-494568	0.5	0.5	0.4
neos-495307	0.1	0.1	0.3
neos-498623	0.8	0.8	0.9
neos-501453	0.0	0.0	0.0
neos-501474	0.0	0.0	0.0
neos-503737	0.3	0.3	0.5
neos-504674	0.1	0.0	0.0
neos-504815	0.0	0.0	0.0
neos-506422	0.3	0.3	0.2
neos-506428	11.6	7.0	2.6
neos-512201	0.0	0.0	0.0
neos-520729	2.4	1.8	26.8
neos-522351	0.0	0.0	0.0
neos-525149	13.7	12.8	6.4
neos-530627	0.0	0.0	0.0
neos-538867	0.0	0.0	0.0
neos-538916	0.0	0.0	0.0
neos-544324	8.1	9.6	7.4
neos-547911	2.8	3.1	2.5
neos-548047	1.8	1.7	1.8
neos-548251	0.2	0.2	0.1
neos-551991	1.0	1.0	0.7
neos-555001	0.5	0.4	1.0
neos-555298	0.1	0.1	0.2
neos-555343	0.4	0.3	0.8
neos-555424	0.2	0.2	0.3
neos-555694	0.2	0.2	0.2
neos-555771	0.2	0.1	0.2

LP	**esolver-do**	**esolver**	**soplex**
neos-555884	0.2	0.2	0.4
neos-555927	0.1	0.1	0.1
neos-565672	239.7	271.6	615.9
neos-565815	3.5	3.7	3.4
neos-570431	0.2	0.2	0.2
neos-574665	0.1	0.1	0.5
neos-578379	0.2	0.2	28.6
neos-582605	0.1	0.1	0.1
neos-583731	0.0	0.0	0.0
neos-584146	0.1	0.1	0.0
neos-584851	0.0	0.0	0.0
neos-584866	1.7	1.6	3.3
neos-585192	1.6	1.8	0.6
neos-585467	0.5	0.5	0.4
neos-593853	0.1	0.1	0.1
neos-595904	0.2	0.3	0.2
neos-595905	0.1	0.1	0.1
neos-595925	0.1	0.1	0.1
neos-598183	0.1	0.1	0.1
neos-603073	0.1	0.1	0.1
neos-611135	4.0	4.7	8.9
neos-611838	0.5	0.5	0.5
neos-612125	0.6	0.6	0.5
neos-612143	0.7	0.7	0.4
neos-612162	0.5	0.5	0.4
neos-619167	46.1	44.9	1.3
neos-631164	0.0	0.1	0.1
neos-631517	0.0	0.0	0.1
neos-631694	0.2	0.2	3.6
neos-631709	27.1	26.1	302.7
neos-631710	4166.9	3806.7	1408.4
neos-631784	9.0	7.2	81.7
neos-632335	1.5	1.7	6.1
neos-633273	0.8	0.9	4.6
neos-641591	5.1	5.7	8.8
neos-655508	0.3	0.3	0.3
neos-662469	6.0	5.6	11.5
neos-686190	0.2	0.2	0.2
neos-691058	0.9	0.9	1.3

LP	**esolver-do**	**esolver**	**soplex**
neos-691073	1.1	1.1	1.1
neos-693347	1.5	1.4	2.4
neos6	1.4	1.3	3.7
neos-702280	34.8	33.4	28.9
neos-709469	0.0	0.0	0.0
neos-717614	0.2	0.2	0.1
neos-738098	95.9	98.3	54.4
neos-775946	0.6	0.6	0.8
neos-777800	3.4	3.2	0.4
neos-780889	176.4	154.8	158.3
neos-785899	0.1	0.1	0.1
neos-785912	0.1	0.1	0.2
neos-785914	0.1	0.1	0.1
neos-787933	2.0	1.8	52.1
neos788725	0.1	0.1	0.1
neos-791021	0.7	0.6	2.6
neos-796608	0.0	0.0	0.0
neos-799711	11.1	379.1	12.0
neos-799838	3.2	2.6	4.2
neos-801834	0.7	0.7	0.4
neos-803219	0.1	0.1	0.1
neos-803220	0.1	0.1	0.1
neos-806323	0.1	0.1	0.1
neos-807454	0.6	0.6	1.4
neos-807456	0.8	0.8	1.3
neos-807639	0.1	0.1	0.1
neos-807705	0.1	0.1	0.1
neos-808072	1.1	1.0	1.8
neos-808214	0.2	0.2	0.2
neos808444	0.7	0.6	12.5
neos-810286	2.8	2.6	3.9
neos-810326	0.9	0.9	1.2
neos-820146	0.0	0.0	0.0
neos-820157	0.1	0.0	0.1
neos-820879	0.5	0.5	0.6
neos-824661	270.6	341.7	7.5
neos-824695	71.5	66.4	1.3
neos-825075	0.1	0.1	0.1
neos-826224	38.0	32.3	2.2

LP	esolver-do	esolver	soplex
neos-826250	17.6	16.4	0.8
neos-826650	0.5	0.5	1.0
neos-826694	16.2	13.9	4.0
neos-826812	8.4	8.6	2.1
neos-826841	1.0	1.0	0.5
neos-827015	36.7	47.9	16.4
neos-827175	23.8	27.9	4.2
neos-829552	14.4	13.7	5.5
neos-830439	0.0	0.0	0.0
neos-831188	1.3	1.3	1.0
neos-839838	16.5	15.9	1.9
neos-839859	0.7	0.7	0.3
neos-839894	267.4	226.8	52.0
neos-841664	2.8	2.8	0.5
neos-847051	0.4	0.4	0.4
neos-847302	0.4	0.4	0.2
neos-848150	0.1	0.1	0.2
neos-848198	0.1	0.1	2.2
neos-848589	9.5	8.6	8.0
neos-848845	0.4	0.4	1.0
neos-849702	0.3	0.3	0.9
neos-850681	0.7	0.6	4.9
neos-856059	0.5	0.5	0.3
neos858960	0.0	0.0	0.0
neos-859770	4.4	4.2	2.7
neos-860244	1.2	1.3	1.0
neos-860300	1.8	1.8	1.4
neos-862348	0.4	0.4	0.5
neos-863472	0.0	0.0	0.0
neos-872648	397.5	375.9	268.3
neos-873061	496.2	381.5	279.3
neos-876808	1064.7	737.8	110.5
neos-880324	0.0	0.0	0.0
neos-881765	0.0	0.0	0.1
neos-885086	2.1	1.3	1.5
neos-885524	1.3	1.4	7.2
neos-886822	0.2	0.2	0.2
neos-892255	0.4	0.4	0.2
neos-905856	0.1	0.1	0.1

LP	esolver-do	esolver	soplex
neos-906865	0.1	0.1	0.1
neos-911880	0.0	0.0	0.0
neos-911970	0.0	0.0	0.0
neos-912015	0.1	0.2	0.1
neos-912023	0.1	0.1	0.1
neos-913984	0.8	0.8	4.3
neos-914441	7.7	87.3	7.1
neos-916173	0.5	0.6	0.6
neos-916792	0.8	0.9	1.1
neos-930752	4.5	4.5	10.0
neos-931517	2.1	1.9	2.4
neos-931538	4.9	4.0	2.9
neos-932721	5.4	6.2	13.0
neos-932816	22.8	21.3	15.9
neos-933364	0.1	0.1	0.1
neos-933550	0.5	0.5	0.3
neos-933562	8.0	7.5	1.0
neos-933638	45.0	47.8	31.2
neos-933815	0.1	0.1	0.1
neos-933966	29.3	23.6	22.3
neos-934184	0.1	0.1	0.1
neos-934278	64.1	63.2	33.1
neos-934441	65.8	58.3	37.6
neos-934531	1.7	1.5	0.9
neos-935234	31.1	29.0	40.4
neos-935348	29.6	27.7	39.0
neos-935496	3.1	3.2	0.8
neos-935627	35.2	32.7	22.8
neos-935674	4.1	4.2	0.8
neos-935769	11.8	9.7	19.3
neos-936660	17.5	18.2	22.5
neos-937446	20.0	17.5	22.6
neos-937511	17.0	13.0	27.4
neos-937815	47.6	49.9	44.4
neos-941262	15.9	15.7	21.0
neos-941313	167.4	134.8	142.9
neos-941698	0.2	0.2	0.2
neos-941717	0.4	0.4	0.5
neos-941782	0.3	0.3	0.2

LP	**esolver-do**	**esolver**	**soplex**
neos-942323	0.1	0.1	0.1
neos-942830	0.2	0.2	0.2
neos-942886	0.1	0.1	0.0
neos-948126	31.6	25.5	36.4
neos-948268	0.9	0.8	4.0
neos-948346	15.6	15.0	9.2
neos-950242	4.7	4.8	6.3
neos-952987	1.2	1.1	0.9
neos-953928	177.5	161.8	5.1
neos-954925	1153.0	1062.7	1772.5
neos-955215	0.1	0.0	0.1
neos-955800	0.4	0.3	1.1
neos-956971	144.0	136.8	310.8
neos-957143	165.4	163.4	4984.8
neos-957270	1.5	1.2	1.5
neos-957323	87.0	77.1	7.7
neos-957389	2.6	3.6	1.2
neos-960392	2.1	2.3	29.4
neos-983171	17.8	17.9	28.2
neos-984165	28.0	25.9	35.1
neos	1972.6	2204.8	1223.3
nesm	0.2	0.4	0.6
net12	1.6	1.5	3.9
netdiversion	90.7	71.5	41.9
netlarge2	10429.5	9667.3	13.5
netlarge3	32.6	29.0	63.6
newdano	0.1	0.1	0.0
nl	13.4	11.8	5.3
nobel-eu-DBE	0.1	0.1	0.2
noswot	0.0	0.2	0.0
npmv07	104.5	134.4	106.2
ns1111636	43.9	42.7	174.2
ns1116954	1103.1	831.6	57.4
ns1208400	1.6	1.6	7.1
ns1456591	1.6	1.6	1.1
ns1606230	9.9	9.4	6.2
ns1631475	149.3	127.6	954.1
ns1644855	1387.1	1345.0	203.6
ns1663818	22.0	21.5	20.8

LP	**esolver-do**	**esolver**	**soplex**
ns1685374	1462.8	3139.7	1303.0
ns1686196	0.2	0.2	0.3
ns1687037	76398.2	76867.7	*0.0*
ns1688347	0.2	3.3	0.2
ns1688926	2561.1	2567.4	*0.0*
ns1696083	1.3	1.2	1.5
ns1702808	0.1	0.0	0.0
ns1745726	0.3	0.3	0.3
ns1758913	225.3	178.3	1037.4
ns1766074	0.0	0.0	0.0
ns1769397	0.4	0.4	0.5
ns1778858	6.7	6.9	16.0
ns1830653	3.0	2.7	0.8
ns1853823	*0.0*	*0.0*	38431.0
ns1854840	*0.0*	*0.0*	16765.3
ns1856153	0.9	0.9	1.0
ns1904248	627.7	667.4	188.7
ns1905797	2.3	2.4	5.7
ns1905800	0.3	0.3	0.4
ns1952667	0.9	0.9	1.4
ns2017839	467.4	532.2	707.7
ns2081729	0.1	0.0	0.0
ns2118727	769.8	670.6	133.7
ns2122603	117.1	304.0	9.1
ns2124243	292.6	224.5	288.1
ns2137859	11.1	11.8	18.7
ns4-pr3	0.5	0.5	2.0
ns4-pr9	0.4	0.4	1.4
ns894236	31.3	28.0	15.9
ns894244	1345.7	1329.9	26.6
ns894786	289.4	244.3	26.0
ns894788	1.0	1.0	1.3
ns903616	352.0	309.7	47.3
ns930473	4.1	3.8	22.4
nsa	0.1	0.1	0.1
nsct1	4.1	4.9	5.9
nsct2	4.1	3.8	3.5
nsic1	0.0	0.0	0.0
nsic2	0.0	0.0	0.0

LP	**esolver-do**	**esolver**	**soplex**
nsir1	0.6	0.6	0.8
nsir2	0.6	0.6	0.9
nsr8k	1145.7	1148.4	453.3
nsrand-ipx	0.5	0.7	0.5
nu120-pr3	0.5	0.6	0.9
nu60-pr9	0.3	0.3	0.8
nug05	0.0	0.0	0.0
nug06	0.1	0.1	0.1
nug07	0.4	0.4	0.5
nug08-3rd	8685.8	9061.4	7217.9
nug08	1.2	1.2	1.8
nug12	1100.1	1187.5	251.6
nug15	*0.0*	*0.0*	55976.9
nug20	*0.0*	*0.0*	*0.0*
nug30	*0.0*	*0.0*	*0.0*
nw04	2.5	2.4	2.8
nw14	4.5	4.1	3.9
ofi	159.1	160.7	328.4
opm2-z10-s2	1075.3	1006.0	218.5
opm2-z11-s8	3126.1	3541.9	353.0
opm2-z12-s14	1339.0	1147.4	580.0
opm2-z12-s7	3984.3	4224.6	588.2
opm2-z7-s2	46.0	31.9	24.0
opt1217	0.0	0.0	0.0
orna1	1.7	1.9	1.2
orna2	1.6	1.9	1.2
orna3	1.7	2.1	1.2
orna4	2.8	3.1	1.2
orna7	1.7	2.0	1.3
orswq2	0.0	0.0	0.0
osa-07	0.7	0.7	1.1
osa-14	2.7	3.1	2.7
osa-30	7.6	7.4	14.1
osa-60	27.0	26.6	65.4
p0033	0.0	0.0	0.0
p0040	0.0	0.0	0.0
p010	3.8	3.6	5.5
p0201	0.0	0.0	0.0
p0282	0.0	0.0	0.0

LP	**esolver-do**	**esolver**	**soplex**
p0291	0.0	0.0	0.0
p0548	0.0	0.0	0.0
p05	1.0	1.0	1.0
p100x588b	0.0	0.0	0.0
p19	0.0	0.0	0.1
p2756	0.0	0.1	0.1
p2m2p1m1p0n100	0.0	0.0	0.0
p6000	0.1	0.1	0.2
p6b	0.1	0.1	0.1
p80x400b	0.0	0.0	0.0
pcb1000	0.5	0.5	0.3
pcb3000	2.3	2.3	1.6
pds-02	0.1	0.1	0.2
pds-06	1.1	1.0	1.7
pds-100	2975.1	3130.2	9363.8
pds-10	3.6	3.4	4.0
pds-20	29.2	28.5	30.7
pds-30	218.4	172.9	155.3
pds-40	417.1	466.3	199.4
pds-50	1214.4	925.5	2739.7
pds-60	2039.8	1870.4	3927.3
pds-70	2002.6	2901.0	5131.7
pds-80	2809.3	3001.2	9225.0
pds-90	3165.1	3275.8	13481.8
perold	6.3	6.5	1.4
pf2177	1.7	1.3	3.3
pg5_34	0.1	0.1	0.1
pg	0.0	0.1	0.1
pgp2	6.3	4.6	0.3
pigeon-10	0.0	0.0	0.0
pigeon-11	0.1	0.0	0.1
pigeon-12	0.1	0.0	0.1
pigeon-13	0.1	0.1	0.1
pigeon-19	0.2	0.1	0.1
pilot4	0.7	1.0	0.8
pilot87	3822.7	4019.9	361.9
pilot-ja	19.0	19.4	3.2
pilotnov	4.1	4.2	1.1
pilot	122.4	263.1	31.2

LP	**esolver-do**	**esolver**	**soplex**
pilot-we	2.0	2.6	1.5
pk1	0.0	0.0	0.0
pldd000b	1.2	1.2	0.4
pldd001b	1.1	1.2	0.4
pldd002b	1.3	1.3	0.4
pldd003b	1.0	1.0	0.4
pldd004b	1.1	1.2	0.4
pldd005b	0.9	1.0	0.4
pldd006b	1.6	1.6	0.4
pldd007b	1.1	1.1	0.4
pldd008b	1.6	1.7	0.5
pldd009b	1.6	1.6	0.4
pldd010b	1.7	1.8	0.5
pldd011b	1.6	1.6	0.5
pldd012b	1.4	1.4	0.4
pltexpa2-16	0.1	0.1	0.1
pltexpa2-6	0.0	0.0	0.0
pltexpa3_16	4.4	4.0	5.2
pltexpa3_6	0.3	0.3	0.4
pltexpa4_6	204.8	197.1	5.2
pp08aCUTS	0.0	0.0	0.0
pp08a	0.0	0.0	0.0
primagaz	0.5	0.5	0.4
problem	0.0	0.0	0.0
probportfolio	0.0	0.0	0.0
prod1	0.0	0.0	0.0
prod2	0.1	0.0	0.0
progas	11.8	15.5	6.1
protfold	0.7	0.7	1.1
pw-myciel4	0.5	0.4	0.4
qap10	16.5	15.5	17.5
qiulp	0.2	0.5	0.1
qiu	0.2	0.5	0.1
qnet1_o	0.0	0.0	0.1
qnet1	0.0	0.0	0.1
queens-30	647.9	624.0	67.3
r05	1.7	1.6	1.7
r80x800	0.0	0.0	0.0
rail01	1463.5	1709.8	14987.4

LP	esolver-do	esolver	soplex
rail02	*0.0*	*0.0*	*0.0*
rail03	27089.0	29582.6	8.3
rail2586	16.2	13.9	24.3
rail4284	19.2	17.6	17.1
rail507	14.5	12.7	21.5
rail516	3.3	3.9	7.1
rail582	11.0	9.3	10.1
ramos3	980.7	954.4	416.5
ran10x10a	0.0	0.0	0.0
ran10x10b	0.0	0.0	0.0
ran10x10c	0.0	0.0	0.0
ran10x12	0.0	0.0	0.0
ran10x26	0.0	0.0	0.0
ran12x12	0.0	0.0	0.0
ran12x21	0.0	0.0	0.0
ran13x13	0.0	0.0	0.0
ran14x18_1	0.0	0.0	0.0
ran14x18-disj-8	4.1	8.9	0.8
ran14x18.disj-8	4.1	9.1	0.8
ran14x18	0.0	0.0	0.0
ran16x16	0.0	0.0	0.0
ran17x17	0.0	0.0	0.0
ran4x64	0.0	0.0	0.0
ran6x43	0.0	0.0	0.0
ran8x32	0.0	0.0	0.0
rat1	9.4	10.5	4.0
rat5	511.1	524.9	224.8
rat7a	10949.6	11166.3	6503.0
rd-rplusc-21	63.0	65.3	1112.6
reblock166	5.9	4.9	2.3
reblock354	23.5	20.4	15.9
reblock420	83.1	76.2	41.3
reblock67	0.2	0.2	0.2
recipe	0.0	0.0	0.0
refine	0.0	0.0	0.0
rentacar	0.7	0.6	2.1
rgn	0.0	0.0	0.0
rlfddd	1.1	0.9	1.6
rlfdual	4.6	4.1	3.3

LP	**esolver-do**	**esolver**	**soplex**
rlfprim	30.5	34.3	37.6
rlp1	0.0	0.0	0.0
rmatr100-p10	1.9	1.8	1.1
rmatr100-p5	2.4	2.5	4.0
rmatr200-p10	31.6	34.9	25.9
rmatr200-p20	23.9	22.6	13.7
rmatr200-p5	57.6	52.1	33.7
rmine10	239.5	224.7	50.1
rmine14	11185.1	9111.4	3658.8
rmine21	86.5	75.9	17.9
rmine25	22.1	18.5	31.5
rmine6	1.0	1.0	0.4
rocII-4-11	2.2	1.9	1.3
rocII-7-11	4.4	4.8	2.7
rocII-9-11	6.1	5.8	3.3
rococoB10-011000	0.1	0.1	0.9
rococoC10-001000	0.1	0.1	0.2
rococoC11-011100	0.2	0.2	1.5
rococoC12-111000	0.3	0.3	2.2
roll3000	0.3	0.3	0.4
rosen10	1.2	1.2	1.1
rosen1	0.2	0.2	0.2
rosen2	0.5	0.5	0.6
rosen7	0.0	0.1	0.1
rosen8	0.1	0.1	0.2
route	5.5	5.8	1.7
rout	0.0	0.0	0.0
roy	0.0	0.0	0.0
rvb-sub	12.2	11.6	4.3
satellites1-25	4.0	3.5	9.6
satellites2-60-fs	60.9	55.2	580.8
satellites2-60	90.7	76.2	583.0
satellites3-40-fs	322.4	309.9	6315.3
satellites3-40	466.0	430.3	6521.4
sc105	0.0	0.0	0.0
sc205-2r-100	0.1	0.1	0.7
sc205-2r-1600	1.1	1.0	2203.0
sc205-2r-16	0.0	0.0	0.0
sc205-2r-200	0.4	0.4	4.7

LP	**esolver-do**	**esolver**	**soplex**
sc205-2r-27	0.0	0.0	0.0
sc205-2r-32	0.0	0.0	0.0
sc205-2r-400	1.4	1.4	35.7
sc205-2r-4	0.0	0.0	0.0
sc205-2r-50	0.1	0.0	0.1
sc205-2r-64	0.1	0.0	0.2
sc205-2r-800	4.2	4.3	306.9
sc205-2r-8	0.0	0.0	0.0
sc205	0.0	0.0	0.0
sc50a	0.0	0.0	0.0
sc50b	0.0	0.0	0.0
scagr25	0.0	0.0	0.0
scagr7-2b-16	0.0	0.0	0.0
scagr7-2b-4	0.0	0.0	0.0
scagr7-2b-64	2.2	2.5	2.3
scagr7-2c-16	0.0	0.0	0.0
scagr7-2c-4	0.0	0.0	0.0
scagr7-2c-64	0.2	0.2	0.3
scagr7-2r-108	0.4	0.4	0.4
scagr7-2r-16	0.0	0.0	0.0
scagr7-2r-216	1.5	1.6	1.5
scagr7-2r-27	0.1	0.1	0.1
scagr7-2r-32	0.1	0.1	0.1
scagr7-2r-432	8.4	7.7	5.9
scagr7-2r-4	0.0	0.0	0.0
scagr7-2r-54	0.2	0.2	0.2
scagr7-2r-64	0.2	0.2	0.3
scagr7-2r-864	49.0	41.3	25.2
scagr7-2r-8	0.0	0.0	0.0
scagr7	0.0	0.0	0.0
scfxm1-2b-16	0.4	0.4	0.4
scfxm1-2b-4	0.1	0.1	0.1
scfxm1-2b-64	25.7	22.2	15.8
scfxm1-2c-4	0.1	0.1	0.1
scfxm1-2r-128	23.2	19.9	19.6
scfxm1-2r-16	0.4	0.4	0.4
scfxm1-2r-256	76.8	63.6	113.8
scfxm1-2r-27	0.6	0.6	0.7
scfxm1-2r-32	1.1	1.0	1.0

LP	**esolver-do**	**esolver**	**soplex**
scfxm1-2r-4	0.1	0.1	0.1
scfxm1-2r-64	4.7	4.3	4.8
scfxm1-2r-8	0.1	0.2	0.2
scfxm1-2r-96	11.5	12.5	11.2
scfxm1	0.0	0.0	0.0
scfxm2	0.1	0.1	0.1
scfxm3	0.1	0.1	0.2
scorpion	0.0	0.0	0.0
scrs8-2b-16	0.0	0.0	0.0
scrs8-2b-4	0.0	0.0	0.0
scrs8-2b-64	0.1	0.1	0.1
scrs8-2c-16	0.0	0.0	0.0
scrs8-2c-32	0.0	0.0	0.0
scrs8-2c-4	0.0	0.0	0.0
scrs8-2c-64	0.1	0.1	0.1
scrs8-2c-8	0.0	0.0	0.0
scrs8-2r-128	0.1	0.1	0.2
scrs8-2r-16	0.0	0.0	0.0
scrs8-2r-256	0.3	0.3	0.5
scrs8-2r-27	0.0	0.0	0.0
scrs8-2r-32	0.0	0.0	0.0
scrs8-2r-4	0.0	0.0	0.0
scrs8-2r-512	1.0	1.0	1.6
scrs8-2r-64b	0.1	0.1	0.1
scrs8-2r-64	0.1	0.1	0.1
scrs8-2r-8	0.0	0.0	0.0
scrs8	0.1	0.1	0.1
scsd1	0.0	0.1	0.0
scsd6	0.6	0.5	0.1
scsd8-2b-16	0.0	0.1	0.1
scsd8-2b-4	0.0	0.0	0.0
scsd8-2b-64	11.8	11.8	0.7
scsd8-2c-16	0.1	0.1	0.1
scsd8-2c-4	0.0	0.0	0.0
scsd8-2c-64	3.2	3.1	0.9
scsd8-2r-108	2.4	2.2	0.4
scsd8-2r-16	0.1	0.1	0.1
scsd8-2r-216	6.3	5.6	1.0
scsd8-2r-27	0.2	0.2	0.1

LP	**esolver-do**	**esolver**	**soplex**
scsd8-2r-32	0.2	0.2	0.1
scsd8-2r-432	9.8	7.5	1.9
scsd8-2r-4	0.0	0.0	0.0
scsd8-2r-54	0.6	0.6	0.2
scsd8-2r-64	0.7	0.7	0.2
scsd8-2r-8b	0.0	0.0	0.0
scsd8-2r-8	0.0	0.0	0.0
scsd8	0.1	0.1	0.2
sct1	16.5	17.6	9.6
sct32	12.2	11.3	5.4
sct5	24.2	27.5	4.3
sctap1-2b-16	0.0	0.0	0.0
sctap1-2b-4	0.0	0.0	0.0
sctap1-2b-64	1.0	1.0	1.1
sctap1-2c-16	0.0	0.0	0.0
sctap1-2c-4	0.0	0.0	0.0
sctap1-2c-64	0.1	0.1	0.2
sctap1-2r-108	0.2	0.2	0.3
sctap1-2r-16	0.0	0.0	0.1
sctap1-2r-216	0.8	0.7	0.7
sctap1-2r-27	0.1	0.0	0.1
sctap1-2r-32	0.1	0.1	0.1
sctap1-2r-480	2.4	1.9	2.4
sctap1-2r-4	0.0	0.0	0.0
sctap1-2r-54	0.1	0.1	0.1
sctap1-2r-64	0.1	0.1	0.2
sctap1-2r-8b	0.0	0.0	0.0
sctap1-2r-8	0.0	0.0	0.0
sctap1	0.0	0.0	0.0
sctap2	0.1	0.0	0.1
sctap3	0.1	0.0	0.1
seba	0.0	0.0	0.0
self	100.7	21448.8	2987.3
set1ch	0.0	0.0	0.0
set3-10	1.6	1.6	0.2
set3-15	1.6	1.6	0.3
set3-20	1.8	1.7	0.3
seymour-disj-10	30.8	31.2	6.1
seymour.disj-10	28.5	29.1	6.1

LP	esolver-do	esolver	soplex
seymourl	1.6	1.5	0.9
seymour	1.4	1.5	0.8
sgpf5y6	268.4	216.7	1630.5
share1b	0.0	0.0	0.0
share2b	0.0	0.0	0.0
shell	0.0	0.0	0.1
ship04l	0.0	0.1	0.1
ship04s	0.0	0.0	0.0
ship08l	0.1	0.1	0.1
ship08s	0.1	0.1	0.1
ship12l	0.1	0.1	0.2
ship12s	0.1	0.1	0.1
shipsched	9.0	9.3	13.1
shs1023	552.1	731.3	1542.5
siena1	54.9	53.4	32.6
sierra	0.1	0.1	0.1
sing161	24335.6	26224.4	9.8
sing245	14571.9	11173.0	3098.6
sing2	59.3	72.5	112.8
sing359	18031.7	14099.3	12774.3
slptsk	12.5	14.1	10.3
small000	0.1	0.1	0.1
small001	0.1	0.1	0.1
small002	0.1	0.2	0.1
small003	0.1	0.1	0.1
small004	0.1	0.1	0.1
small005	0.1	0.1	0.1
small006	0.1	0.1	0.1
small007	0.1	0.1	0.1
small008	0.1	0.1	0.1
small009	0.1	0.1	0.1
small010	0.1	0.1	0.1
small011	0.1	0.1	0.1
small012	0.1	0.1	0.1
small013	0.1	0.1	0.1
small014	0.1	0.1	0.1
small015	0.1	0.1	0.1
small016	0.1	0.1	0.1
south31	42.0	49.7	27.8

LP	**esolver-do**	**esolver**	**soplex**
sp97ar	2.7	2.3	4.7
sp97ic	0.6	0.6	0.7
sp98ar	2.9	3.0	22.1
sp98ic	1.2	1.3	5.5
sp98ir	0.4	0.4	0.5
spal_004	48.1	48.1	43.2
splan1	94.0	2739.1	*0.0*
square15	16454.4	17657.8	6402.7
stair	3.2	3.5	0.7
standata	0.0	0.0	0.0
standmps	0.0	0.0	0.0
stat96v1	1737.1	24261.9	21729.6
stat96v2	31424.0	24108.9	*0.0*
stat96v3	28409.4	23577.3	19.6
stat96v4	2401.7	18201.8	1884.7
stat96v5	812.3	2686.7	570.5
stein27	0.0	0.0	0.0
stein45	0.0	0.0	0.0
stocfor1	0.0	0.0	0.0
stocfor2	0.2	0.2	0.3
stocfor3	20.2	11.5	15.7
stockholm	174.7	176.9	46.9
stormg2_1000	16565.3	13356.4	21.9
stormG2_1000	7694.6	17698.0	7961.3
stormg2-125	120.9	120.7	100.0
stormg2-27	3.0	3.0	2.2
stormg2-8	0.4	0.4	0.4
stp3d	4240.2	3192.3	7747.1
sts405	4.9	5.5	0.7
sts729	35.7	30.1	3.8
swath	0.2	0.2	0.2
sws	0.7	0.7	1.2
t0331-4l	52.6	45.2	25.5
t1717	35.0	31.0	32.2
t1722	6.1	6.3	9.1
tanglegram1	3.7	3.3	5.6
tanglegram2	0.2	0.2	0.2
Test3	48.0	37.8	9.6
testbig	4.2	4.0	272.4

LP	**esolver-do**	**esolver**	**soplex**
timtab1	0.0	0.0	0.0
timtab2	0.0	0.0	0.0
toll-like	0.1	0.1	0.1
tp-6	23.3	18.8	14.5
tr12-30	0.0	0.0	0.0
transportmoment	6.2	6.4	2.4
triptim1	295.8	287.1	143.4
triptim2	18187.9	15413.8	989.4
triptim3	14237.3	11562.3	509.7
truss	1.8	1.8	3.2
ts-palko	*0.0*	*0.0*	*0.0*
tuff	0.0	0.0	0.0
tw-myciel4	2.4	2.2	2.7
uc-case11	488.4	459.2	99.6
uc-case3	104.9	104.9	159.5
uct-subprob	0.3	0.3	0.4
ulevimin	24.1	28.4	124.4
umts	3.3	3.6	0.8
unitcal_7	4.1	3.6	28.1
us04	1.8	1.6	1.3
usAbbrv-8-25_70	0.4	0.4	0.1
van	20.2	18.4	21.5
vpm1	0.0	0.0	0.0
vpm2	0.0	0.0	0.0
vpphard2	20.8	19.6	49.8
vpphard	27.2	26.2	29.1
vtp-base	0.0	0.0	0.0
wachplan	2.0	1.9	0.6
watson_1	555.2	542.8	1384.4
watson_2	4968.3	6398.2	3276.6
wide15	16362.5	17566.5	7376.6
wnq-n100-mw99-14	123.8	110.7	22.2
wood1p	0.6	0.8	0.7
woodw	0.3	0.3	0.5
world	770.2	949.7	291.1
zed	0.0	0.0	0.0
zib54-UUE	0.2	0.2	0.2

Bibliography

[1] R. H. Bartels, “A stabilization of the simplex method,” *Numerische Mathematik*, vol. 16, pp. 414–434, Feb 1971.

[2] C. Roos, T. Terlaky, and J.-P. Vial, *Interior Point Methods for Linear Optimization.* New York: Springer, 2nd ed., 2006.

[3] S. Gao, J. Avigad, and E. M. Clarke, “Delta-decidability over the reals,” in *2012 27th Annual IEEE Symposium on Logic in Computer Science*, pp. 305–314, June 2012.

[4] S. Gao, J. Avigad, and E. M. Clarke, “δ-complete decision procedures for satisfiability over the reals,” in *Automated Reasoning: 6th International Joint Conference*, pp. 286–300, Springer, 2012.

[5] C. Barrett and C. Tinelli, “Satisfiability modulo theories,” in *Handbook of Satisfiability, volume 185, chapter 26*, pp. 825–885, IOS Press, 2009.

[6] B. Dutertre and L. de Moura, “Integrating Simplex with DPLL(T),” tech. rep., CSL, SRI International, 2006.

[7] L. de Moura and N. Bjørner, “Z3: An efficient SMT solver,” in *Tools and Algorithms for the Construction and Analysis of Systems 2008, European Conferences on Theory and Practice of Software*, pp. 337–340, Springer, 2008.

[8] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli, “CVC4,” in *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV ’11)* (G. Gopalakrishnan and S. Qadeer, eds.), vol. 6806 of *Lecture Notes in Computer Science*, (Snowbird, Utah), pp. 171–177, Springer, July 2011.

[9] D. L. Applegate, W. Cook, S. Dash, and D. G. Espinoza, “Exact solutions to linear programming problems,” *Operations Research Letters*, vol. 35, no. 6, pp. 693–699, 2007.

[10] A. M. Gleixner and D. E. Steffy, “Linear programming using limited-precision oracles,” *Mathematical Programming*, pp. 1436–4646, Nov. 2019.

[11] C. Jansson, “Rigorous lower and upper bounds in linear programming,” *SIAM Journal on Optimization*, vol. 14, no. 3, pp. 914–935, 2004.

[12] R. Bartels and G. Golub, “The simplex method of linear programming using LU decomposition,” *Communications of the ACM*, vol. 12, no. 5, pp. 266–268, 1969.

[13] M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, “Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure,” *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 1, no. 3–4, pp. 209–236, 2006.

[14] C. Herde, *Efficient Solving of Large Arithmetic Constraint Systems with Complex Boolean Structure.* PhD thesis, University of Oldenburg, 2010.

[15] A. Tarski, *A decision method for elementary algebra and geometry.* Berkeley: University of California Press, 2d ed., rev.. ed., 1951.

[16] A. M. Gleixner, D. E. Steffy, and K. Wolter, “Improving the accuracy of linear programming solvers with iterative refinement,” in *Proceedings of ISSAC ’12*, pp. 187–194, 2012.

[17] A. M. Gleixner, D. E. Steffy, and K. Wolter, “Iterative refinement for linear programming,” *INFORMS Journal on Computing*, vol. 28, no. 3, pp. 449–464, 2016.

[18] D. G. Espinoza, *On Linear Programming, Integer Programming and Cutting Planes.* PhD thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, May 2006.

[19] J. Forrest and J. Tomlin, “Updated triangular factors of the basis to maintain sparsity in the product form simplex method,” *Mathematical Programming*, vol. 2, no. 1, pp. 263–278, 1972.

[20] N. Higham, R. Chan, C. Greif, and D. O’Leary, *Commentary on Matrix Factorizations and Applications*, pp. 227–235. United Kingdom: Oxford University Press, 2007.

[21] G. H. Golub and C. F. van Loan, *Matrix Computations.* Johns Hopkins Studies in the Mathematical Sciences, Baltimore: Johns Hopkins University Press, 4th ed., 2013.

[22] N. J. Higham, *Accuracy and Stability of Numerical Algorithms.* Philadelphia: Society for Industrial and Applied Mathematics, 2002.

[23] R. G. Bland, “New finite pivoting rules for the simplex method,” *Mathematics of Operations Research*, vol. 2, no. 2, pp. 103–107, 1977.

[24] J. Matousek, *Understanding and using linear programming.* Universitext, Springer, 2007.

[25] M. Reed and B. Simon, *Functional Analysis*, vol. 1 of *Methods of modern mathematical physics.* New York: Academic Press, rev. and enl. ed., 1980.

[26] J. H. Wilkinson, “Error analysis of floating-point computation,” *Numerische Mathematik*, vol. 2, pp. 319–340, Dec 1960.

[27] S. J. Wright, “Effects of finite-precision arithmetic on interior-point methods for nonlinear programming,” *SIAM Journal on Optimization*, vol. 12, no. 1, pp. 36–78, 2001.

[28] N. Sloane and J. Stufken, “A linear programming bound for orthogonal arrays with mixed levels,” *Journal of Statistical Planning and Inference*, vol. 56, no. 2, pp. 295–305, 1996.