Dissertation

The Dual Simplex Method, Techniques for a fast and stable implementation

von

Dipl. Inform. Achim Koberstein

Schriftliche Arbeit zur Erlangung des akademischen Grades
doctor rerum politicarum (dr. rer. pol.)
im Fach Wirtschaftsinformatik

eingereicht an der
Fakultät für Wirtschaftswissenschaften der
Universität Paderborn

Gutachter:
1. Prof. Dr. Leena Suhl
2. Prof. Dr. Michael Jünger

Paderborn, im November 2005

To my family

Danksagungen

Die vorliegende Arbeit entstand in den vergangenen zweieinhalb Jahren, die ich als Stipendiat der *International Graduate School of Dynamic Intelligent Systems* am *Decision Support & Operations Research (DSOR) Lab* der Universität Paderborn verbrachte. Ich möchte mich an dieser Stelle bei allen Menschen bedanken, die zum Gelingen dieser Arbeit beigetragen haben.

Zuerst gilt mein besonderer und herzlicher Dank meinem "Betreuerteam", der Leiterin des DSOR-Labs Prof. Dr. Leena Suhl und ihrem Mann Prof. Dr. Uwe Suhl von der Freien Universität Berlin. Leena Suhl danke ich für das besondere Vertrauensverhältnis und die ständige fachliche und persönliche Unterstützung. Uwe Suhl erwähnte zu Beginn meiner Promotionszeit eher beiläufig, dass der duale Simplexalgorithmus ein lohnenswertes Thema wäre. Es ist vor allem ihm und seiner vorausgegangenen fünfundzwanzigjährigen Entwicklungsarbeit an dem Optimierungssystem MOPS zu verdanken, dass daraus dann tatsächlich eine Dissertation geworden ist. Die enge und lehrreiche Zusammenarbeit mit ihm hat mir großen Spaß gemacht und wird hoffentlich noch lange andauern.

Ich möchte mich außerdem bei allen Kollegen am DSOR-Lehrstuhl für die gute Zusammenarbeit und die anregende Forschungsatmosphäre bedanken. Besonders meiner Bürokollegin Natalia Kliewer (inzwischen Juniorprofessorin) gebührt mein Dank für ihre freundschaftliche Unterstützung, viele erhellende und erheiternde philosophisch-politische Gespräche über die Monitore hinweg und ihre verständnisvolle Teilnahme an privaten und beruflichen Höhenflügen und Tiefschlägen der letzten drei Jahre.

Ich danke auch der International Graduate School of Dynamic Intelligent Systems, insbesondere dem Leiter Dr. Eckhard Steffen und dem ganzen Organisationsteam für die engagierte Begleitung meiner Arbeit, die weit über die finanzielle Unterstützung hinaus ging. Ich wünsche der IGS eine glänzende Zukunft und ihren jetzigen und künftigen Stipendiaten eine erfolg- und lehrreiche Promotionszeit.

Ich danke allen, die mir bei der Korrektur des Manusskripts geholfen haben, insbesondere Astrid Lukas-Reiß, Sophie Koberstein und Markus Krause.

Schließlich möchte ich mich bei meiner Frau Sophie sowie meinen Eltern, Schwiegereltern und Freunden für ihren Rückhalt und ihre Unterstützung bedanken. Durch euch weiß ich jeden Tag, was wirklich wichtig ist.

Vielen herzlichen Dank!

Paderborn, im Oktober 2005 Achim Koberstein

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In 1947, G.B. Dantzig stated the Linear Programming Problem (LP) and presented the (primal) simplex method[1] to solve it (cf. [18, 19, 21]). Since then many researchers have strived to advance his ideas and made linear programming the most frequently used optimization technique in science and industry. Besides the investigation of the theoretical properties of the simplex method the development of increasingly powerful computer codes has always been a main goal of research in this field. Boosted by the performance leaps of computer hardware, the continuous improvement of its algorithmic and computational techniques is probably the main reason for the success story of the simplex method. Orden [52] and Hoffmann [35] were among the first to report computational results for their codes. In 1952, an LP problem with 48 constraints and 71 variables took about 18 hours and 73 simplex iterations to solve on a SEAC computer, which was the hardware available at that time. The most difficult test problem[2] used in this dissertation has 162,142 constraints and 1,479,833 variables and is solved by our dual simplex code in about 7 hours and about 422,000 simplex iterations on a standard personal computer (see chapter 9). More information about the history of linear programming and LP computer codes can be found in [43] and [51], respectively.

While the primal simplex algorithm was in the center of research interest for decades and subject of countless publications, this was not the case regarding its dual counterpart. After Lemke [42] had presented the dual simplex method in 1954, it was not considered to be a competitive alternative to the primal simplex method for nearly forty years. Commercial LP-systems of the late 1980s like MPSX/370, MPS III or APEX-III featured only rudimentary implementations of it and did not even include dual phase I methods to deal with dual infeasible starting bases (cf. [10]). This eventually changed in the early 1990s mainly due to the contributions of Forrest and Goldfarb [26], who developed a computationally relatively cheap dual version of the steepest edge pricing rule.

During the last decade commercial solvers made great progress in establishing the dual simplex method as a general solver for large-scale LP problems. Nowadays, large scale LP problems can be solved either by an interior point, primal simplex or dual simplex algorithm or a combination of such algorithms. In fact, extensive computational studies indicate, that the overall performance of the dual simplex

[1]The historic circumstances of the early days of linear programming have been documented e.g. by S.I. Gass [28] and A.J. Hoffman [36].

[2]LP-relaxation of the integer problem MUN1_M_D, which is an instance of multi-commodity-flow model used in a bus scheduling application [37].

may be superior to that of the primal simplex algorithm (cf. [10]). In practise, there are often LP-models, for which one of the three methods clearly outperforms the others. For instance, experiments showed, that the test problem mentioned above can only be solved by the dual simplex method. Primal simplex codes do virtually not converge on this problem due to degeneracy and interior point codes fail due to extensive memory consumption.

Besides its relevance for the solution of large scale LP problems, it is long known, that the dual simplex algorithm plays an important role for solving LP problems, where some or all of the variables are constrained to integer values (mixed-integer linear programming problems – MIPs). Virtually all state-of-the-art MIP-solvers are based on a branch-and-bound approach, where dual bounds on the objective function value are computed by successive reoptimization of LP-type subproblems. While the interior-point method is conceptually ineligible to take advantage of a given nearly optimal starting solution, the dual simplex method is particularly well suited for this purpose. The reason is that for most of the branch-and-bound subproblems the last LP-solution stays dual feasible and the execution of a dual phase I method is not necessary. Therefore, the dual simplex method is typically far superior to the primal simplex method in a branch-and-bound framework.

Despite of its success and relevance for future research only few publications in research literature explicitly discuss implementation details of mathematical or computational techniques proposed for the dual simplex algorithm. Furthermore, reported computational results are often produced by out-dated simplex codes, which do not feature indispensable techniques to solve large scale LPs (like a well implemented LU factorization of the basis and a powerful LP preprocessor). Even if the presented ideas look promising, it often remains unclear, how to implement them within a state-of-the-art LP-system. Such techniques are for example:

- An enhanced dual ratio test for dual phase I and phase II. It was described by Fourer [27] in an unpublished rather theoretical note about the dual simplex method[3]. Maros [48, 47] and Kostina [41] published computational results for this technique.
- Pan's dual phase I method. Pan presented his algorithm in [54] and published computational results in [55].
- A method to exploit hypersparsity. In the mid 1980's Gilbert and Peierls [29] published a technique to solve particularly sparse systems of linear equations. In [10] Bixby indicates that this technique contributed to enormous improvements of the Cplex LP-solver. However, he does not disclose implementation details.

The lack of descriptions of implementation details in the research literature has led to a great performance gap between open-source research codes[4] and commercial LP-

[3]Apparently, the idea was published long before by Gabasov [57] in Russian language.

[4]An exception is the LP code, which is being developed in the COIN open-source initiave [44]. However, this code is largely undocumented and no research papers have yet been published about its internals.

systems, which is frequently documented in independent benchmarks of optimization software (cf. [4]).

The goals of this dissertation follow directly from the above discussion. We think, that it is essential for future research in the field of linear and mixed integer programming to dispose of a state-of-the art implementation of the dual simplex algorithm. Therefore, we want

- to develop a dual simplex code, which is competitive to the best existing open-source and commercial LP-systems,
- to identify, advance and document important implementation techniques, which are responsible for the superiority of commercial simplex codes, and
- to conduct meaningful computational studies to evaluate promising mathematical and computational techniques.

Our work is based on the Mathematical OPtimization System (MOPS) (see [65, 66]), which has been deployed in many practical applications for over two decades (see e.g. [69], [67], [62] and [37]) and has continuously been improved in algorithms, software design and implementation. The system started as a pure LP-solver based on a high speed primal simplex algorithm. In its present form MOPS belongs to the few competitive systems in the world to solve large-scale linear and mixed integer programming problems. The algorithms and computational techniques used in MOPS have been documented in numerous scientific publications (cf. section 8.1).

The remainder of this thesis is structured in three major parts. In part I, which comprises the chapters 2 to 4, we give a detailed description of the relevant mathematical algorithms. In chapter 2 we introduce fundamental concepts of linear programming. Chapter 3 gives a detailed derivation of the dual simplex method. The chapter ends with an algorithmic description of an elaborate version of the algorithm, which represents the basis of the dual phase II part of our implementation. In chapter 4 we give an overview of dual phase I algorithms. In particular, we show that there are two mathematically equivalent approaches to minimize the sum of dual infeasibilities and give the first algorithmic description of Pan's method for general LPs with explicit lower and upper bounds.

In part II, we describe computational techniques, which are crucial for the performance and numerical stability of our code. Here, the efficient solution of the required systems of linear equations in chapter 5 plays an important role. We particularly emphasize the exploitation of hypersparsity. Techniques to achieve numerical stability and prevent degeneracy and cycling are presented in chapter 6. We discuss in detail, how to combine Harris' idea to use tolerances to improve numerical stability with the bound flipping ratio test. The shorter chapter 7 is concerned with further important points, e.g. LP preprocessing and the efficient computation of the transformed pivot row.

Part III comprises the chapters 8 and 9. The first describes our implementation of the mathematical algorithms and computational techniques presented in the previous parts. Focal points are efficient data structures, organization of the pricing loop, the dual ratio test and the exploitation of hypersparsity. In the second chapter of this part we evaluate the performance of our code compared to the best commercial and open-source implementations of the dual simplex method on the basis of computational results. Furthermore, we provide a study on the dual phase I and a chronological progress study, which illustrates the impact of the specific implementation techniques on the overall performance of the code during our development process.

Chapter 10 summarizes the contributions of this dissertation and discusses possible directions of future research.

Part I

Fundamental algorithms

Chapter 2

Foundations

2.1 The linear programming problem and its computational forms

A *linear programming problem (LP)* is the problem of minimizing (or maximizing) a linear function subject to a finite number of linear constraints. In matrix notation this definition corresponds to the following *general form* of the LP problem:

$$\begin{aligned} \text{minimize} \quad & c_0 + c^T x && (2.1a)\\ \text{subject to} \quad & L \leq \bar{A}x \leq U && (2.1b)\\ & l \leq x \leq u, && (2.1c) \end{aligned}$$

where $c_0 \in \mathbb{R}$, $c, x \in \mathbb{R}^{\bar{n}}$, $b \in \mathbb{R}^m$, $\bar{A} \in \mathbb{R}^{m \times \bar{n}}$, $l, u \in (\mathbb{R} \cup \{-\infty, +\infty\})^{\bar{n}}$ and $L, U \in (\mathbb{R} \cup \{-\infty, +\infty\})^m$ with $m, \bar{n} \in \mathbb{N}$. We call c the *cost vector* (c_0 is a constant component), x the vector of the *decision variables*, $\bar{A}$ the *constraint matrix*, L and U the (*lower* and *upper*) *range vectors* and l and u the (*lower* and *upper*) bounds. (2.1a) is called *objective function*, (2.1b) the *set of joint constraints* and (2.1c) the *individual bound constraints*. We call a variable x_j with $j \in \{1, \ldots, \bar{n}\}$ *free* if $l_j = -\infty$ and $u_j = +\infty$. We call it *boxed* if $l_j > -\infty$ and $u_j < +\infty$. If $l_j = u_j = a$ for some $a \in \mathbb{R}$ we call it *fixed*.

It is easy to see that any kind of LP problem that may occur in a practical application can be put into general form (2.1). If for $i \in \{1, \ldots, m\}$ we have both $L_i > -\infty$ and $U_i < +\infty$ constraint i is a *range constraint*. If $L_i > -\infty$ and $U_i = +\infty$ or $L_i = -\infty$ and $U_i < +\infty$ constraint i is an *inequality-constraint* ($\geq$ or $\leq$ resp.). If $-\infty < L_i = U_i < +\infty$ it is an *equality-constraint*.

To be approachable by the simplex algorithm LP (2.1) has to be cast into a *computational form*, that fulfills further requirements, i.e., the constraint matrix has to have full row rank and only equality constraints are allowed:

$$\begin{aligned} \min \quad & c^T x && (2.2a)\\ \text{s.t.} \quad & Ax = b && (2.2b)\\ & l \leq x \leq u. && (2.2c) \end{aligned}$$

Here, $c, x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $l, u \in (\mathbb{R} \cup \{-\infty, +\infty\})^n$ and $A \in \mathbb{R}^{m \times n}$ with $rank(A) = m$, $m, n \in \mathbb{N}$ and $m < n$. In this representation we call b the *right hand side (RHS) vector*. In the following we will denote by $\mathcal{J} = \{1, \ldots, n\}$ the set of column indices.

To put an LP in general form (2.1) into one in computational form (2.2) inequality constraints are transformed into equality constraints by introducing *slack-* and *surplus-variables.* In LP-systems (like MOPS) this is usually done in a standardized way by adding a complete identity matrix to the constraint matrix:

$$\min \quad c^T x_S \tag{2.3a}$$

$$\text{s.t.} \quad \left[\, \bar{A} \mid I \,\right] \begin{bmatrix} x_S \\ x_L \end{bmatrix} = 0 \tag{2.3b}$$

$$l \leq \begin{bmatrix} x_S \\ x_L \end{bmatrix} \leq u \tag{2.3c}$$

The variables associated with $\bar{A}$ are called *structural variables* (short: *structurals*), those associated with I are called *logical variables* (short: *logicals*). Ranges L_i and U_i are transferred to individual bounds on logical variables by setting $l_{\bar{n}+i} = -U_i$ and $u_{\bar{n}+i} = -L_i$. Consequently, equality constraints lead to fixed logical variables. Note, that this scheme assures that both the full row rank assumption (because of I) and the requirement that $m < n$ (since $n = \bar{n} + m$) are fulfilled. We call (2.3) the *internal model representation (IMR)* while the general form (2.1) is also called *external model representation (EMR).*

For ease of notation we will mostly use the computational form (2.2) to describe mathematical methods and techniques. We will always assume that it coincides with the IMR (2.3). One obvious computational advantage of the IMR is for instance, that the right hand side vector b is always 0, which means that it vanishes completely in implementation. Nevertheless, we consider b in the algorithmic descriptions.

To examine the theoretical properties of LPs and also for educational purposes, an even simpler yet mathematically equivalent representation is used in LP literature. It is well known that every LP can be converted into the following *standard form*:

$$\min \quad c^T x \tag{2.4a}$$

$$\text{s.t.} \quad Ax \geq b \tag{2.4b}$$

$$x \geq 0. \tag{2.4c}$$

The set $\mathcal{X} = \{x \in \mathbb{R}^n : Ax \geq b, x \geq 0\}$ is called the *feasible region* of the LP (2.4) and $x \in \mathbb{R}^n$ is called *feasible* if $x \in \mathcal{X}$. If for every $M \in \mathbb{R}$ there is an $x \in \mathcal{X}$ such that $c^T x < M$, then (2.4) is called *unbounded.* If $\mathcal{X} = \emptyset$ it is called *infeasible.* Otherwise, an *optimal solution* $x^* \in \mathcal{X}$ exists with objective function value $z^* = c^T x^*$ and $z^* \leq c^T x$ for all $x \in \mathcal{X}$.

The simplex algorithm was originally developed for LPs in standard form (without taking individual bounds into account implicitly). To apply the simplex algorithm to (2.4) it has to be transformed as above to a form which we will call *computational standard form*:

$$\min \quad c^T x \tag{2.5a}$$

$$\text{s.t.} \quad Ax = b \tag{2.5b}$$

$$x \geq 0, \tag{2.5c}$$

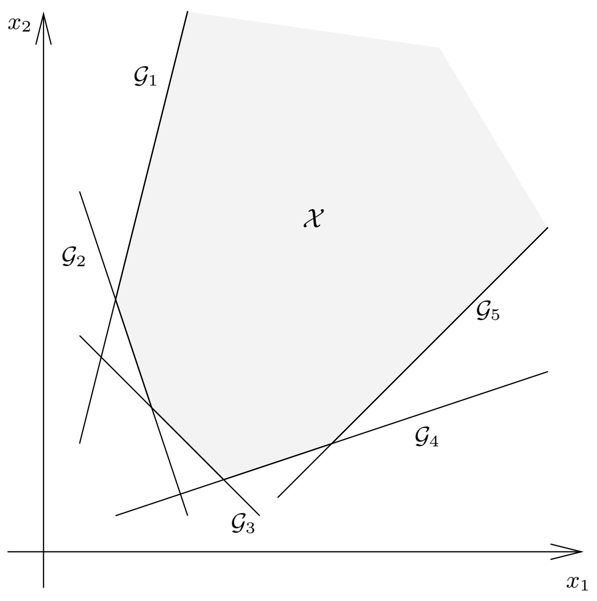


Figure 2.1: A convex polyhedron in $\mathbb{R}^2$.

where $A \in \mathbb{R}^{m \times n}$ has full row rank m and $m < n$. We will use the standard form and its computational variant only rarely since they are of minor importance for practical LP systems.

2.2 Geometry

Each inequality[1] in the set of joint and individual bound constraints can be interpreted geometrically as a *half-space* $\mathcal{H}_i = \{x \in \mathbb{R}^n : a^i x \leq b_i\}$ in the space of the decision variables $\mathbb{R}^n$ with a corresponding *hyperplane* $\mathcal{G}_i = \{x \in \mathbb{R}^n : a^i x = b_i\}$. Therefore, the feasible region $\mathcal{X}$ of any LP problem can be described as the intersection of a finite number of half-spaces, which is called a *polyhedron*. Polyhedra are convex, since a half-space is a convex set, and the intersection of convex sets is convex. A convex polyhedron in $\mathbb{R}^2$ is depicted in figure 2.1.

The extreme points of a polyhedron $\mathcal{X}$, that cannot be written as a nontrivial linear combination of other points in $\mathcal{X}$, are called *vertices*. They lie in the intersection of at most n hyperplanes, that define $\mathcal{X}$. If more than n hyperplanes intersect at a vertex, it is called *degenerate*. If the line emanating from a vertex along the intersection of at least two hyperplanes is not bounded by another vertex, it is an *extreme ray* of the polyhedron. Many results on polyhedra are known in the theory of convex sets. The *representation theorem* says, that every nonempty polyhedron $\mathcal{X}$ can be represented as the union of the convex hull of a finite set of vertices $\mathcal{S} = \{s_1, \ldots, s_k\} \subseteq \mathcal{X}$ and the cone defined by a finite set of extreme directions $\mathcal{R} = \{r_1, \ldots, r_q\} \subset \mathbb{R}^n$. Furthermore, every point $x \in \mathcal{X}$ can be written as the sum of a convex linear combination of the vertices and a linear combination of the extreme directions:

[1]an equality can be transformed into one (if it includes a logical variable) or two inequalities

$$x = \sum_{i=1}^{k} \lambda_i s_i + \sum_{j=1}^{q} \mu_j r_j, \quad \text{with} \quad \sum_{i=1}^{k} \lambda_i = 1,\ \lambda_i \geq 0\,\forall i \quad \text{and} \quad \mu_j \geq 0\,\forall j. \tag{2.6}$$

With the help of this theorem one can proof a fundamental property of LP problems: for every LP problem exactly one of the following three *solution states* is true (see e.g. [46], p.25 for proof):

- The LP is *infeasible.* No feasible solution exists ($\mathcal{X} = \emptyset$).

- The LP is *unbounded.* There is a vertex $s_i \in \mathcal{S}$, an extreme direction $r_j \in \mathcal{R}$ such that for every $M \in \mathbb{R}$ we can find a value $\mu \geq 0$ such that $x^* = s_i + \mu r_j \in \mathcal{X}$ and $c^T x^* < M$.

- The LP has an *optimal solution.* There is an optimal vertex $x^* \in \mathcal{S}$ such that $c^T x^* \leq c^T x$ for all $x \in \mathcal{X}$.

Accordingly, if an LP problem has an optimal solution, it suffices to search the (finite number of) vertices of the feasible region until a vertex with the optimal solution value is found. From the two possible definitions of a vertex two different algebraic representations can be derived. Defining a vertex as the intersection of hyperplanes leads to the concept of a *row basis.* We will not further pursue this concept, since in most cases it is computationally inferior to the concept of a *column basis*, which follows from the vertex definition via nontrivial convex combinations.

Suppose, we have an LP given in computational standard form and $\mathcal{X} = \{x \in \mathbb{R}^n : Ax = b, x \geq 0\}$. A point $x \in \mathcal{X}$ is a vertex if and only if the column vectors of A, that are associated with strictly positive entries in x, are linearly independent (see e.g. [16] , p.9 for proof). As a direct consequence x can have at most m positive entries. If x has strictly less than m positive entries, we can always expand the number of linearly independent columns to m by adding columns associated with zero positions of x, since A has full row rank (in this case, the vertex is degenerate). Therefore, every vertex of the feasible region has at least one set of m linearly independent columns of A associated with it. The set of indices of these columns is called a *(column) basis.* Consequently, degenerate vertices have several bases.

The fundamental idea of the simplex algorithm is to search the finite number of bases until a basis is found that belongs to the optimal vertex. Since there are $\binom{m}{n}$ bases, this might take an exponential number of steps, which is in fact the theoretical bound for the runtime of the method. In practise, it has surprisingly turned out to perform much better (typically $c * m$ number of steps, where c is a small constant). It has been learned, that the performance heavily depends on sophisticated search strategies.

2.3 LP Duality

The basic idea of LP duality can be seen best by considering an LP in standard form:

$$\begin{aligned} z^* = \min \quad & c^T x \\ \text{s.t.} \quad & Ax \geq b \\ & x \geq 0. \end{aligned} \tag{2.7}$$

If we multiply every constraint $i \in \{1, \dots, m\}$ with a nonnegative number y_i such that $y^T a_j \leq c_j$ for every $j \in \{1, \dots, n\}$, then $y^T b$ is a lower bound on the optimal objective function value z^*, since $y^T b \leq y^T Ax \leq c^T x$ (the first relation is true, because $y \geq 0$, the second is true, because $x \geq 0$ and $y^T a_j \leq c_j$). This property is called *weak duality.* Since it is satisfied by every y satisfying the restrictions above we can find the best lower bound by solving the LP:

$$\begin{aligned} Z^* = \max \quad & b^T y \\ \text{s.t.} \quad & A^T y \leq c \\ & y \geq 0. \end{aligned} \tag{2.8}$$

This LP is called the *dual LP* (or short: the *dual*) of the LP (2.7). Both LPs are called a *primal-dual pair.* Since any LP problem can be transformed into standard form, we can associate a dual problem to it in the above sense.

Consider a primal constraint $a^i x \leq b_i$. By multiplying by -1, we get $-a^i \geq -b_i$, which is a constraint in standard form. In the dual program, a dual multiplier $y_i \geq 0$ gets assigned to it and $-b_i$ becomes its coefficient in the dual objective function, while $-(a^i)T$ becomes its coefficient (column) in the dual constraint matrix. This is equivalent to assigning a dual multiplier $y_i \leq 0$ right away and keeping b_i and $(a^i)^T$ as coefficients in the dual objective function and constraint matrix respectively. Primal equality constraints $a^i x = b_i$ can be replaced by two inequality constraints $a^i x \geq b_i$ and $a^i \leq b_i$. In the dual program two dual variables $y_i^1 \geq 0$ and $y_i^2 \leq 0$ go with them, which have identical coefficients b_i and $(a^i)^T$ in the dual objective function and constraint matrix respectively. Hence, the coefficients can be factored out giving $b_i(y_i^1 + y_i^2)$ and $(a^i)^T(y_i^1 + y_i^2)$ respectively. The sum $y_i^1 + y_i^2$ can assume any value in $\mathbb{R}$, so it can be substituted by a free dual variable y_i. In a similar manner we can derive further transformation rules for different types of primal variables. We summarize them in table 2.1.

Now, reconsider an LP in computational form and suppose in the first instance, that all individual bounds are finite, i.e., $l_j > -\infty$ and $u_j < \infty$ for all $j \in \mathcal{J}$:

$$\min \quad c^T x \tag{2.9a}$$
$$\text{s.t.} \quad Ax = b \tag{2.9b}$$
$$l \leq x \leq u. \tag{2.9c}$$

Then, we can treat the individual bounds (2.9c) like constraints and introduce dual variables $y \in \mathbb{R}^m$ for the constraints (2.9b), $v \in \mathbb{R}^n$ for the lower bound constraints and $w \in \mathbb{R}^n$ for the upper bound constraints in (2.9c). The constraints (2.9b) can be

primal	dual
$a^i x \geq b_i$	$y_i \geq 0$
$a^i x \leq b_i$	$y_i \leq 0$
$a^i x = b_i$	y_i free
$x \geq 0$	$a_j^T y \leq c_j$
$x \leq 0$	$a_j^T y \geq c_j$
x free	$a_j^T y = c_j$

Table 2.1: Primal-dual transformation rules.

dualized by using the third transformation rule, the lower bounds by using the first and the upper bounds by using the second rule. This leads to the following dual for LP (2.9):

$$\begin{aligned} \max \quad & b^T y + l^T v + u^T w && \text{(2.10a)} \\ \text{s.t.} \quad & A^T y + v + w = c && \text{(2.10b)} \\ & v \geq 0 && \text{(2.10c)} \\ & w \leq 0 && \text{(2.10d)} \end{aligned}$$

We call v_j and w_j *dual slack variables* or *dual logicals*. By setting $d = v+w = c-A^T y$ we see, that the sum of the dual slacks coincides with the *reduced cost vector* d. We will come back to this correspondence in the next section.

If we permit individual bounds to take infinite value in the primal LP, we have to be more careful in the dual LP. In fact, for infinite bounds no dual variables need to be introduced. In our notation we keep the indexing of the dual variables from 1 to n, but all v_j, w_j for which $l_j = -\infty$, $u_j = \infty$, respectively, vanish from the dual program:

$$\begin{aligned} \max \quad & b^T y + \sum_{\{j\,:\,l_j > -\infty\}} l_j v_j + \sum_{\{j\,:\,u_j < \infty\}} u_j w_j && && \text{(2.11a)} \\ \text{s.t.} \quad & a_j^T y = c_j && \text{if } l_j = -\infty \text{ and } u_j = \infty && \text{(2.11b)} \\ & a_j^T y + v_j = c_j && \text{if } l_j > -\infty \text{ and } u_j = \infty && \text{(2.11c)} \\ & a_j^T y + w_j = c_j && \text{if } l_j = -\infty \text{ and } u_j < \infty && \text{(2.11d)} \\ & a_j^T y + v_j + w_j = c_j && \text{if } l_j > -\infty \text{ and } u_j < \infty && \text{(2.11e)} \\ & v_j \geq 0 && \text{if } l_j > -\infty && \text{(2.11f)} \\ & w_j \leq 0 && \text{if } u_j < -\infty && \text{(2.11g)} \end{aligned}$$

In the following sections we will use (2.11) to illustrate the mathematical concepts of the dual simplex method (DSX). The DSX solves an LP given in computational form (2.2), or rather in IMR (2.3), by implicitly applying the primal simplex to its dual.

Consider LP (2.9) with permissible infinite individual bounds. Any vector $x \in \mathbb{R}^n$ that satisfies (2.9b) is called a *primal solution*. If a primal solution x lies within the individual bounds (2.9c) it is called a *primal feasible solution.* If no primal feasible solution exists, (2.9) is said to be *primal infeasible.* Otherwise, it is *primal feasible.* If for every $M \in \mathbb{R}$ there is a primal feasible solution x such that $c^T x < M$ then (2.9) is *primal unbounded.*

Accordingly, any vector $(y^T, v^T, w^T)^T \in \mathbb{R}^{m+2n}$ that satisfies the dual constraints (2.11b) – (2.11e) is called a *dual solution.* If a dual solution additionally satisfies constraints (2.11f) and (2.11g), it is called a *dual feasible solution.* If no dual feasible solution exists, (2.9) is said to be *dual infeasible.* Otherwise it is *dual feasible.* If for every $M \in \mathbb{R}$ there is a dual feasible solution $(y^T, v^T, w^T)^T$ such that $b^T y + \sum_{\{j\,:\,l_j > -\infty\}} l_j v_j + \sum_{\{j\,:\,u_j < \infty\}} u_j w_j > M$, then (2.9) is *dual unbounded.*

From weak duality, we can directly conclude that an LP must be primal infeasible if it is dual unbounded. Likewise, if it is primal unbounded, it must be dual infeasible. The converse need not be true, there are LPs that are both primal and dual infeasible. In the next sections we will show that, if a primal optimal solution exists then there is also a dual optimal solution with the same objective function value. This property is called *strong duality.*

2.4 Basic solutions, feasibility, degeneracy and optimality

As we have seen in section 2.2, a *basis* $\mathcal{B} = \{k_1, \dots, k_m\}$ is an ordered subset of the set of column indices $\mathcal{J} = \{1, \dots, n\}$, such that the submatrix $A_\mathcal{B}$ of the constraint matrix A is nonsingular. For convenience we will denote the basis matrix by B, i.e., $B = A_\mathcal{B} = (a_{k_1}, \dots, a_{k_m}) \in \mathbb{R}^{m \times m}$ and refer to the i-th element of the basis by the notation $\mathcal{B}(i)$, i.e., $\mathcal{B}(i) = k_i$. The set of nonbasic column indices is denoted by $\mathcal{N} = \mathcal{J} \setminus \mathcal{B}$. A variable x_j is called *basic* if $j \in \mathcal{B}$ and *nonbasic* if $j \in \mathcal{N}$. By permuting the columns of the constraint matrix and the entries of x, c we can write $A = (B, A_\mathcal{N})$, $x = \binom{x_\mathcal{B}}{x_\mathcal{N}}$ and $c = \binom{c_\mathcal{B}}{c_\mathcal{N}}$. Accordingly, (2.9) can be expressed as

$$Bx_\mathcal{B} + A_\mathcal{N} x_\mathcal{N} = b. \tag{2.12}$$

Since B is nonsingular (and hence, its inverse B^{-1} exists,) we obtain

$$x_\mathcal{B} = B^{-1}(b - A_\mathcal{N} x_\mathcal{N}). \tag{2.13}$$

This equation shows that the nonbasic variables $x_\mathcal{N}$ uniquely determine the values of the nonbasic variables $x_\mathcal{B}$. In this sense, we can think of the basic variables as dependent and the nonbasic variables as independent.

A primal solution x is called *basic*, if every nonbasic variable is at one of its finite bounds or takes a predefined constant value $a \in \mathbb{R}$ in the case that no such bound exists (= free variable). If x is basic and primal feasible we call it a *primal feasible basic solution* and $\mathcal{B}$ a *primal feasible basis.* Since the $n - m$ nonbasic variables of a primal basic solution are feasible by definition, primal feasibility can be checked by looking only at the m basic variables and testing, if they are within their bounds:

$$l_{\mathcal{B}} \leq x_{\mathcal{B}} \leq u_{\mathcal{B}} \tag{2.14}$$

If at least one basic variable hits one of its finite bounds exactly, x and $\mathcal{B}$ are said to be *primal degenerate*. The number of components of $x_{\mathcal{B}}$, for which this is true, is called the *degree of primal degeneracy*.

Considering the dual LP (2.11) we can partition the constraints (2.11b) – (2.11e) into basic (if $j \in \mathcal{B}$) and nonbasic (if $j \in \mathcal{N}$) ones. A dual solution $(y^T, v^T, w^T)^T$ is called *basic* if

$$\begin{aligned} & B^T y = c_{\mathcal{B}} \\ \Leftrightarrow \quad & y = (B^T)^{-1} c_B \end{aligned} \tag{2.15a}$$

and

$$\begin{aligned}
v_j &= c_j - a_j^T y & & & &\text{if } l_j > -\infty \text{ and } u_j = \infty, & (2.15b)\\
w_j &= c_j - a_j^T y & & & &\text{if } l_j = -\infty \text{ and } u_j < \infty, & (2.15c)\\
v_j &= c_j - a_j^T y & &\text{and} & w_j = 0 \quad &\text{if } l_j > -\infty \text{ and } u_j < \infty \text{ and } x_j = l_j, & (2.15d)\\
w_j &= c_j - a_j^T y & &\text{and} & v_j = 0 \quad &\text{if } l_j > -\infty \text{ and } u_j < \infty \text{ and } x_j = u_j. & (2.15e)
\end{aligned}$$

In the following, we denote the vector of the *reduced costs* by

$$d = c - A^T y. \tag{2.16}$$

Now, we see that for a dual basic solution, we get

$$\begin{aligned}
v_j &= d_j & & & &\text{if } l_j > -\infty \text{ and } u_j = \infty, & (2.17a)\\
w_j &= d_j & & & &\text{if } l_j = -\infty \text{ and } u_j < \infty, & (2.17b)\\
v_j &= d_j & &\text{and} & w_j = 0 \quad &\text{if } l_j > -\infty \text{ and } u_j < \infty \text{ and } x_j = l_j, & (2.17c)\\
w_j &= d_j & &\text{and} & v_j = 0 \quad &\text{if } l_j > -\infty \text{ and } u_j < \infty \text{ and } x_j = u_j. & (2.17d)
\end{aligned}$$

A dual basic solution is called *feasible*, if the constraints (2.11b), (2.11f) and (2.11g) are satisfied for all $j \in \mathcal{J}$. Note, that dual basic constraints are satisfied by definition (those of type (2.11b) because of 2.15a, those of type (2.11f) and (2.11g), because $d_j = c_j - a_j^T y = c_j - a_j^T (B^T)^{-1} c_{\mathcal{B}} = c_j - e_j^T c_{\mathcal{B}} = c_j - c_j = 0$ for $j \in \mathcal{B}$). The same is true for constraints that are associated with fixed primal variables ($l_j = u_j$), because we can choose between (2.17c) and (2.17d) depending on the sign of d_j (if $d_j \geq 0$, we choose (2.17c), and (2.17d) o.w.).

Therefore, to check dual feasibility, we only need to consider those $j \in \mathcal{N}$, where x_j is neither basic nor fixed. In table 2.2 we summarize the dual feasibility conditions in terms of the corresponding primal basic solution x and the reduced cost vector d.

If for at least one nonbasic variable we have $d_j = 0$, $(y^T, v^T, w^T)^T$ and $\mathcal{B}$ are called *dual degenerate*. The number of zero entries in $d_{\mathcal{N}}$ is called the *degree of dual degeneracy*.

Now, suppose we are given a basis $\mathcal{B}$ with a primal basic solution x and a dual

status of x_j	dual feasible if
basic	true
nonbasic fixed	true
nonbasic at lower bound ($x_j = l_j$)	$d_j \geq 0$
nonbasic at upper bound ($x_j = u_j$)	$d_j \leq 0$
nonbasic free ($x_j = 0$)	$d_j = 0$

Table 2.2: Dual feasibility conditions.

feasible basic solution $(y^T, v^T, w^T)^T$. Replacing x in the primal objective function, we get the following:

$$\begin{aligned}
c^T x &= c_{\mathcal{B}}^T x_{\mathcal{B}} + c_{\mathcal{N}}^T x_{\mathcal{N}} \\
&= c_{\mathcal{B}}^T (B^{-1} b - B^{-1} A_{\mathcal{N}} x_{\mathcal{N}}) + c_{\mathcal{N}}^T x_{\mathcal{N}} && \text{by (2.13)} \\
&= y^T b - y^T A_{\mathcal{N}} x_{\mathcal{N}} + c_{\mathcal{N}}^T x_{\mathcal{N}} && \text{by (2.15a)} \\
&= y^T b + (c_{\mathcal{N}} - y^T A_{\mathcal{N}})^T x_{\mathcal{N}} \\
&= y^T b + d_{\mathcal{N}}^T x_{\mathcal{N}} && \text{by (2.16)} \\
&= y^T b + l^T v + u^T w && \text{by table 2.2}
\end{aligned} \tag{2.18}$$

This means, that the primal and the dual objective function value is equal in this case. Therefore, if x is also primal feasible, then it is optimal due to weak duality. Hence, being both primal and dual feasible is a sufficient condition for a basis $\mathcal{B}$ to be optimal. Vice versa, if we have an optimal primal basic solution, we can construct a dual feasible basic solution, which is optimal for the dual LP.

From polyhedral theory we know, that if an LP has a primal feasible solution, then it also has a primal feasible basic solution and if it has an optimal solution it also has an optimal basic solution (both facts follow from the *representation theorem*, see e.g. [46] for proof). The second fact together with (2.18) proofs *strong duality*: if an LP has an optimal solution, then its dual also has an optimal solution and both optimal solution values are equal.

Chapter 3

The Dual Simplex Method

3.1 The Revised Dual Simplex Algorithm

3.1.1 Basic idea

Simplex type algorithms search the space of basic solutions in a greedy way, until either infeasibility or unboundedness is detected or an optimal basic solution is found. While the primal simplex algorithm maintains primal feasibility and stops when dual feasibility is established, the dual simplex algorithm starts with a dual feasible basis and works towards primal feasibility.

Algorithm 1: Basic steps of the dual simplex method.

```
          Input: LP in computational form (2.2), dual feasible basis $\mathcal{B}$
          Output: Optimal basis $\mathcal{B}$ or proof that LP is dual unbounded.
(Step 1)  Pricing
          Find a leaving variable $p \in \mathcal{B}$, that is primal infeasible. If no such $p$
          exists, then $\mathcal{B}$ is optimal $\rightarrow$ exit.
(Step 2)  Ratio Test
          Find an entering variable $q \in \mathcal{N}$, such $(\mathcal{B} \setminus p) \cup q$ is a again dual
          feasible basis. If no such $q$ exists, then the LP is dual unbounded $\rightarrow$
          exit.
(Step 3)  Basis change
          Set $\mathcal{B} \leftarrow (\mathcal{B} \setminus p) \cup q$ and $\mathcal{N} \leftarrow (\mathcal{N} \setminus q) \cup p$.
          Update problem data.
          Go to step 1.
```

Algorithm 1 shows the basic steps of the dual simplex method. In each iteration it moves from the current dual feasible basis to a neighboring basis by exchanging a variable in $\mathcal{B}$ by a variable in $\mathcal{N}$. Since we want to reduce primal infeasibility, a primal infeasible basic variable is chosen to leave the basis in step 1 and is made nonbasic (hence, primal feasible) by setting it to one of its finite bounds (free variables are not eligible here). In section 3.1.3 we will see, that this selection also ensures a nondecreasing dual objective function value. If no primal infeasible variable exist, we know, that the current basis must be optimal, because it is primal and dual feasible. In step 1 a nonbasic variable is selected to enter the basis, such that the new basis is again dual feasible. We will see in section 3.1.4, that the LP is dual unbounded (i.e. primal infeasible), if no entering variable can be determined. Finally, all necessary update operations associated with the basis change are carried out in step 1.

To be able to apply algorithm 1 to some given LP we are left with the task to obtain a dual feasible basis to start with. There are a variety of methods to do this, which we will discuss in chapter 4. At first, we will describe in greater detail what we mean by neighboring solution, dual pricing, ratio test and basis change.

3.1.2 Neighboring solutions

Let $\mathcal{B} = \{k_1, \ldots, k_m\}$ be a dual feasible basis, $(y^T, v^T, w^T)^T$ a corresponding dual feasible basic solution and x a corresponding primal basic solution. In this situation we know, that the dual constraints (2.11c) – (2.11e) hold at equality for all $j \in \mathcal{B}$ (we do not consider constraint (2.11b) here, because free variables are not eligible to leave the basis) and that the corresponding dual logical variables are equal to zero. If variable $p = \mathcal{B}(r)$ leaves the bases, then the r-th dual basic constraint may (but need not) change from equality to inequality, while all other dual constraints associated with basic variables keep holding at equality. Denoting the new dual solution by $(\bar{y}^T, \bar{v}^T, \bar{w}^T)^T$ and the change in the r-th dual basic constraint by $t \in \mathbb{R}$, such that

$$t = a_p^T \bar{y} - a_p^T y \tag{3.1}$$

we get

$$a_p^T \bar{y} - t = a_p^T y \tag{3.2a}$$

and

$$a_j^T \bar{y} = a_j^T y \quad \text{for all } j \in \mathcal{B} \setminus \{p\}, \tag{3.2b}$$

which can be written concisely as

$$B^T \bar{y} - e_r t = B^T y. \tag{3.3}$$

From equation 3.3 we get

$$\begin{aligned} \bar{y} &= y + (B^T)^{-1} e_r t \\ &= y + \rho_r t, \end{aligned} \tag{3.4}$$

where $\rho_r = (B^T)^{-1} e_r$ denotes the r-th column of $(B^T)^{-1}$, as an update formula for $\bar{y}$ and

$$\bar{d} = c - A^T \bar{y}$$

$$\begin{aligned} &= c - A^T(y + \rho_r t) \\ &= d - A^T(B^T)^{-1} e_r\, t \\ &= d - \alpha^r\, t, \end{aligned} \tag{3.5}$$

where $\alpha^r = e_r^T B^{-1} A$ denotes the r-th row of the transformed constraint matrix $B^{-1}A$, as an update formula for d.

Obviously, $\alpha_j^r = e_r^T B^{-1} a_j = e_r^T e_j = 0$ for all $j \in \mathcal{B} \setminus \{p\}$ and $\alpha_p^r = e_r^T B^{-1} a_p = e_r^T e_r = 1$. Hence, (3.5) can be stated more precisely as

$$\bar{d}_j = d_j = 0 \qquad \text{for all } j \in \mathcal{B} \setminus \{p\}, \tag{3.6a}$$
$$\bar{d}_p = -t \qquad \text{and} \tag{3.6b}$$
$$\bar{d}_j = d_j - \alpha_j^r\, t \quad \text{for all } j \in \mathcal{N}. \tag{3.6c}$$

The change in the r-th dual basic constraint has to be compensated by the corresponding dual logical variable(s). We say, that the constraint is relaxed in a *feasible direction*, if the dual logicals stay feasible. If it is a constraint of type (2.11c), we get $\bar{v}_p = -t$, hence we need $t \le 0$. If it is a constraint of type (2.11d), we get $\bar{w}_p = -t$, hence we need $t \ge 0$. If it is a constraint of type (2.11e), both directions are feasible: we set $\bar{v}_p = -t, \bar{w}_p = 0$ if $t \le 0$ and $\bar{v}_p = 0, \bar{w}_p = -t$ if $t \ge 0$.

In the next two sections, we will clarify, how to choose p and t, such that the dual objective function improves and $(\bar{y}^T, \bar{v}^T, \bar{w}^T)^T$ is again a dual feasible basic solution.

3.1.3 Pricing

If we relax a dual basic constraint p in a dual feasible direction with the restriction, that no $\bar{d}_j$ with $j \in \mathcal{N}$ changes sign (or leaves zero), we get the following dual objective function value $\bar{Z}$ for the new dual solution $(\bar{y}^T, \bar{v}^T, \bar{w}^T)^T$:

$$\begin{aligned} \bar{Z} &= b^T\bar{y} + \sum_{\substack{j \in \mathcal{J} \\ l_j > -\infty}} l_j \bar{v}_j + \sum_{\substack{j \in \mathcal{J} \\ u_j < \infty}} u_j \bar{w}_j \\ &= b^T\bar{y} + \sum_{\substack{j \in \mathcal{N} \\ d_j \ge 0}} l_j \bar{d}_j + \sum_{\substack{j \in \mathcal{N} \\ d_j \le 0}} u_j \bar{d}_j - t u_p^{\pm}, \quad \text{where } u_p^{\pm} = \begin{cases} l_p & \text{if } t \le 0 \\ u_p & \text{if } t \ge 0 \end{cases} \\ &= b^T(y + t e_r^T B^{-1}) + \sum_{\substack{j \in \mathcal{N} \\ d_j \ge 0}} l_j (d_j - t e_r^T B^{-1} a_j) + \sum_{\substack{j \in \mathcal{N} \\ d_j \le 0}} u_j (d_j - t e_r^T B^{-1} a_j) - t u_p^{\pm} \\ &= Z + t e_r^T B^{-1} b - \sum_{\substack{j \in \mathcal{N} \\ d_j \ge 0}} t e_r^T B^{-1} a_j l_j - \sum_{\substack{j \in \mathcal{N} \\ d_j \le 0}} t e_r^T B^{-1} a_j u_j - t u_p^{\pm} \\ &= Z + t e_r^T B^{-1}(b - A_{\mathcal{N}} x_{\mathcal{N}}) - t u_p^{\pm} \\ &= Z + t e_r^T x_{\mathcal{B}} - t u_p^{\pm} \\ &= Z + t(x_p - u_p^{\pm}) \end{aligned}$$

$$= Z + \Delta Z, \quad \text{where } \Delta Z = \begin{cases} t(x_p - l_p) & \text{if } t \le 0 \\ t(x_p - u_p) & \text{if } t \ge 0. \end{cases} \tag{3.7}$$

Now we can determine p and the sign of t such that ΔZ is positive, since we want the dual objective function to increase (maximization). We select p such that either $x_p < l_p$ and $t \le 0$ or $x_p > u_p$ and $t \ge 0$. In both cases the leaving variable can be set to a finite bound, such that it does not violate the dual feasibility conditions. If $t \le 0$, $x_p = l_p$ is dual feasible, since $d_p = -t \ge 0$. The same is true for the case $t \ge 0$: $x_p = u_p$ is dual feasible, since $d_p = -t \le 0$. In the next section we will see, how dual feasibility is maintained for the remaining nonbasic variables.

Note, that x_p is eligible to leave the basis only if it is primal infeasible. If there is no primal infeasible basic variable left, no further improvement in the dual objective function can be accomplished. The decision which of the primal infeasible variables to select as leaving variable has great impact on the number of total iterations of the method. As in the primal simplex method, simple rules (like choosing the variable with the greatest infeasibility) have turned out to be inefficient for the dual simplex algorithm. We will discuss suitable pricing rules in section 3.3. In section 8.2.2 we will describe further computational aspects that have to be taken into account for an efficient implementation of the pricing step.

3.1.4 Ratio test

In the previous section we saw that the dual logical of the relaxed basic constraint always remains dual feasible. Since all other basic constraints stay at equality, we only have to consider the nonbasic dual logicals to fulfill the dual feasibility conditions (see table 2.2). Furthermore, we can neglect dual logicals that are associated with fixed primal variables because they can never go dual infeasible.

As t moves away from zero, we know, that the nonbasic dual logicals evolve according to equation (3.6c):

$$\bar{d}_j = d_j - \alpha_j^r t \qquad \text{for all } j \in \mathcal{N} \tag{3.8}$$

If $x_j = l_j$, dual feasibility is preserved as long as

$$\begin{aligned} & \bar{d}_j \ge 0 \\ \Leftrightarrow \quad & d_j - \alpha_j^r t \ge 0 \\ \Leftrightarrow \quad & t \ge \frac{d_j}{\alpha_j^r} \quad \text{if } \alpha_j^r < 0 \qquad \text{and} \qquad t \le \frac{d_j}{\alpha_j^r} \quad \text{if } \alpha_j^r > 0. \end{aligned} \tag{3.9}$$

If $x_j = u_j$, dual feasibility is preserved as long as

$$\begin{aligned} & \bar{d}_j \le 0 \\ \Leftrightarrow \quad & d_j - \alpha_j^r t \le 0 \\ \Leftrightarrow \quad & t \ge \frac{d_j}{\alpha_j^r} \quad \text{if } \alpha_j^r > 0 \qquad \text{and} \qquad t \le \frac{d_j}{\alpha_j^r} \quad \text{if } \alpha_j^r < 0. \end{aligned} \tag{3.10}$$

If x_j is free, dual feasibility is preserved as long as

$$\begin{aligned} &\bar{d}_j = 0 \\ \Leftrightarrow\quad & d_j - \alpha_j^r t = 0 \\ \Leftrightarrow\quad & t = \frac{d_j}{\alpha_j^r} = 0 \quad \text{since } d_j = 0. \end{aligned} \tag{3.11}$$

When a nonbasic constraint j becomes tight (i.e. $\bar{d}_j = 0$), it gets eligible to replace the relaxed constraint p in the set of basic constraints. The constraints that become tight first as t is increased (or decreased) define a bound θ^D on t, such that dual feasibility is not violated. Among them, we select a constraint q to enter the basis. The associated primal variable x_q is called *entering variable* and θ^D is called *dual step length.* We call a value of t, at which a constraint becomes tight, a *breakpoint.* If no constraint becomes tight, no breakpoint exists and the problem is dual unbounded.

If $t \geq 0$ is required, we can denote the set of subscripts, which are associated with positive breakpoints, by

$$\mathcal{F}^+ = \{j \,:\, j \in \mathcal{N}, x_j \textit{ free} \text{ or}(x_j = l_j \text{ and } \alpha_j^r > 0) \text{ or } (x_j = u_j \text{ and } \alpha_j^r < 0)\}. \tag{3.12}$$

If $\mathcal{F}^+ = \emptyset$, then there exists no bound on t and the problem is dual unbounded. Otherwise, q and θ^D are determined[1] by

$$q \in \arg\min_{j \in \mathcal{F}^+} \left\{ \frac{d_j}{\alpha_j^r} \right\} \quad \text{and} \quad \theta^D = \frac{d_q}{\alpha_j^q}. \tag{3.13}$$

If $t \leq 0$ is required, we can denote the set of subscripts, which are associated with breakpoints, by

$$\mathcal{F}^- = \{j \,:\, j \in \mathcal{N}, x_j \textit{ free} \text{ or}(x_j = l_j \text{ and } \alpha_j^r < 0) \text{ or } (x_j = u_j \text{ and } \alpha_j^r > 0)\}. \tag{3.14}$$

As above, if $\mathcal{F}^- = \emptyset$, the problem is dual unbounded. Otherwise, q and θ^D are determined[2] by

$$q \in \arg\max_{j \in \mathcal{F}^-} \left\{ \frac{d_j}{\alpha_j^r} \right\} \quad \text{and} \quad \theta^D = \frac{d_q}{\alpha_j^q}. \tag{3.15}$$

To ease the algorithmic flow we can get rid of the negative case ($t \leq 0$) by setting $\tilde{\alpha}_j^r = \alpha_j^r$ if $t \geq 0$ and $\tilde{\alpha}_j^r = -\alpha_j^r$ if $t \leq 0$. Then, we can use only equations (3.12) and (3.13) with $\tilde{\alpha}$ instead of α to determine q and θ^D (see step 2 in algorithm 2).

Note, that there can be more then one choice for q in (3.13) and (3.15), since there might be several breakpoints with the same minimal (or maximal) value. This is the case for example, if the current basis is dual degenerate by a degree greater than one (more than one nonbasic d_j is zero). In that situation, we can choose the one among them, that has favorable computational properties. If free nonbasic variables exist, then the basis must be dual degenerate. Therefore, they are always eligible to enter

[1]In the absence of degeneracy the minimum in equation 3.13 is unique.

[2]In the absence of degeneracy the maximum in equation 3.15 is unique.

the basis right away.

3.1.5 Basis change

In the pricing step and the dual ratio test we have determined a variable that enters and one that leaves the basis. In the basis change step we update all the vectors which are required to start the next iteration with the new basis, i.e., the dual basic solution, the primal basic solution, the objective function value and the representation of the basis (and its inverse).

As we have seen before, the dual logicals v and w coincide with the reduced cost vector d. Therefore, the dual basic solution can be represented by $(y^T, d^T)^T$. In section 3.1.2 we have already derived update formulae for y and d. Using the dual step length θ^D for t, we get

$$\bar{y} = y + \theta^D \rho_r, \qquad \text{where } \rho_r = (B^T)^{-1} e_r, \tag{3.16a}$$

for $\bar{y}$ and

$$\begin{aligned}
\bar{d}_p &= -\theta^D, && (3.16b)\\
\bar{d}_q &= 0, && (3.16c)\\
\bar{d}_j &= d_j = 0 \qquad \text{for all } j \in \mathcal{B} \setminus \{p\} && (3.16d)\\
\bar{d}_j &= d_j - \theta^D \alpha_j^r \qquad \text{for all } j \in \mathcal{N} \setminus \{q\} && (3.16e)
\end{aligned}$$

for $\bar{d}$. Note, that it is not necessary though to update both vectors y and d. In most presentations of the dual simplex method (as in ours so far), only an update of d is needed. We can as well formulate a version of the algorithm, in which only y is updated. In this case, those entries of $d_{\mathcal{N}}$, which are needed for the ratio test, must be computed in place from their definition.

Considering the primal basic solution x, we know that the leaving variable x_p goes to one of its finite bounds and that the variable x_q might leave its bound while entering the basis. The displacement of x_q is denoted by θ^P and called *primal step length.* Hence, we get for the primal variables in $\mathcal{N} \cup \{p\}$:

$$\begin{aligned}
\bar{x}_p &= \begin{cases} l_p & \text{if } x_p < l_p \\ u_p & \text{if } x_p > u_p, \end{cases} && (3.17a)\\
\bar{x}_q &= x_q + \theta^P \qquad \text{and} && (3.17b)\\
\bar{x}_j &= x_j \qquad \text{for all } j \in \mathcal{N} \setminus \{q\} && (3.17c)
\end{aligned}$$

Let $j = \mathcal{B}(i)$ and $\rho_i = (B^T)^{-1} e_i$. Then we can derive an update formula for basic primal variables $j \in \mathcal{B} \setminus \{p\}$ from equation (2.13):

$$\bar{x}_j = \rho_i (b - A_{\mathcal{N}} \bar{x}_{\mathcal{N}})$$

$$\begin{aligned}
&= \rho_i b - \alpha^i_{\mathcal{N}} \bar{x}_{\mathcal{N}} \\
&= \rho_i b - \sum_{j \in \mathcal{N} \setminus \{q\}} \alpha^i_j x_j - \alpha^i_q (x_q + \theta^P) \\
&= x_j - \theta^P \alpha^i_q \qquad (3.17\text{d})
\end{aligned}$$

Finally, we can compute the primal step length θ^P by using (3.17d) for $\bar{x}_p$:

$$\begin{aligned}
&\bar{x}_p = x_p - \theta^P \alpha^p_q \\
\Leftrightarrow \quad &\theta^P = \frac{x_p - l_p}{\alpha^p_q} \quad \text{if } x_p < l_p \quad \text{or} \quad \theta^P = \frac{x_p - u_p}{\alpha^p_q} \quad \text{if } x_p > u_p.
\end{aligned} \tag{3.18}$$

To represent the basis and its inverse in a suitable form is one of the most important computational aspects of the (dual) simplex method. We will leave the computational techniques which are used to maintain these representations and to solve the required systems of linear equations for chapter 5. For now, we will only investigate how the basic submatrix B and its inverse alter in a basis change. We start by restating the obvious update of the index sets $\mathcal{B}$ and $\mathcal{N}$ (which in a computational context are called *basis heading*):

$$\begin{aligned}
\bar{\mathcal{B}} &= (\mathcal{B} \setminus p) \cup q \\
\bar{\mathcal{N}} &= (\mathcal{N} \setminus q) \cup p.
\end{aligned} \tag{3.19}$$

The new basis matrix $\bar{B}$ emerges from the old basis matrix B by substituting its r-th column by the entering column a_q, where $p = \mathcal{B}(r)$:

$$\begin{aligned}
\bar{B} &= B - B e_r e_r^T + a_q e_r^T \\
&= B + (a_q - B e_r)\, e_r^T \\
&= B \left(I + B^{-1} (a_q - B e_r)\, e_r^T \right) \\
&= B \left(I + (\alpha_q - e_r)\, e_r^T \right),
\end{aligned} \tag{3.20}$$

where $\alpha_q = B^{-1} a_q$. Let F be defined as

$$F = I + (\alpha_q - e_r)\, e_r^T. \tag{3.21}$$

Hence, F corresponds to an identity matrix, where the r-th column is replaced by α_q:

$$F = \begin{pmatrix} 1 & & \alpha^1_q & & \\ & \ddots & \vdots & & \\ & & \alpha^r_q & & \\ & & \vdots & \ddots & \\ & & \alpha^m_q & & 1 \end{pmatrix}. \tag{3.22}$$

With equations (3.20) and (3.21) the basis matrix $\bar{B}$ can be displayed as follows:

$$\bar{B} = BF. \tag{3.23}$$

Let $E = F^{-1}$, then we get for $\bar{B}^{-1}$:

$$\bar{B}^{-1} = EB^{-1}. \tag{3.24}$$

The inverse of the new basis matrix is the result of the old basis inverse multiplied with matrix E. It is easy to verify that E exists if and only if $\alpha_q^r \neq 0$ and then has the form

$$E = \begin{pmatrix} 1 & & \eta_1 & & \\ & \ddots & \vdots & & \\ & & \eta_r & & \\ & & \vdots & \ddots & \\ & & \eta_m & & 1 \end{pmatrix}, \tag{3.25}$$

where

$$\eta_r = \frac{1}{\alpha_q^r} \qquad \text{and} \qquad \eta_i = -\frac{\alpha_q^i}{\alpha_q^r} \quad \text{for } i = 1, \ldots, m,\ i \neq r. \tag{3.26}$$

A matrix of this type is called an *elementary transformation matrix (ETM)* or short: *eta-matrix*. Let ρ^i, $i = 1, \ldots, m$, be the rows of B^{-1}. Then, we get from equation (3.24) and the entries of η in (3.26) for the rows $\bar{\rho}^i$ of $\bar{B}^{-1}$:

$$\bar{\rho}^r = \frac{1}{\alpha_q^r}\,\rho^r \qquad \text{and} \tag{3.27}$$

$$\bar{\rho}^i = \rho^i - \frac{\alpha_q^i}{\alpha_q^r}\,\rho^r \qquad \text{for } i = 1, \ldots, m,\ i \neq r. \tag{3.28}$$

Equation 3.24 is the basis for the so called *PFI update* (where *PFI* stands for *product form of the inverse*). The idea is to represent the basis inverse as a product of eta-matrices in order to efficiently solve the linear systems efficiently required in the simplex method. This can be done by initializing the solution vector by the right-hand-side of the system and the successively multiply it with the eta-matrices. Depending on wether the system consists of the basis matrix or the transpose of the basis matrix this procedure is performed from head to tail (called *forward transition – FTran*) of the eta-sequence or from tail to head (called *backward transition – BTran*), respectively. The sequence of eta-matrices can be stored in a very compact way since only the nonzero elements in the r-column have to be remembered. However, the PFI-update was replaced by the so called LU-update, which turned out to allow for an even sparser representation and also proved to be numerically more stable. LU factorization and update techniques will be described in great detail in chapter 5.

3.1.6 Algorithmic descriptions

The revised dual simplex method with simple ratio test and update of d is summarized in algorithm 2. Algorithm 3 is a variant where y is updated instead of d. Although the first version seems to be clearly more efficient on the first sight, the situation actually depends on sparsity characteristics of the problem instance and the implementation of the steps *Pivot Row* and *Ratio Test*.

3.2 The Bound Flipping Ratio Test

The simple dual ratio test described in section 3.1.4 selects the entering index q among those nonbasic positions that would become dual infeasible if the displacement t of the r-th dual basic constraint was further increased (or decreased resp.). The *bound flipping ratio test* (which is sometimes also called *generalized ratio test* or *long step rule*) is based on the observation, that a boxed nonbasic variable x_j can be kept dual feasible even if its reduced cost value $\bar{d}_j$ switches sign by setting it to its opposite bound (see table 2.2). This means, that we may further increase the dual step length and pass by breakpoints in the ratio test which are associated with boxed primal variables as long as the dual objective function keeps improving. It can be seen that the rate of improvement decreases with every bound flip. When it drops below zero no further improvement can be made and the entering variable can be selected from the current set of bounding breakpoints (all of which have the same value).

According to Kostina [41] the basic idea of the bound flipping ratio test has been first published in the Russian OR community by Gabasov, Kirillova and Kostyukova [57] as early as 1979. In western OR literature mostly Fourer [27] is cited to be the first one to publish it. In our description we follow Fourer and Maros [47].

To describe the bound flipping ratio test precisely let us consider the case where $x_p > u_p$ and $t > 0$. In equation (3.7) we see, that the slope of the dual objective function with respect to t is given by

$$\delta_1 = x_p - u_p \tag{3.29}$$

as long as no d_j changes sign, i.e,

$$0 \leq t \leq \theta_1^D \qquad \text{with} \quad q_1 \in \arg\min_{j \in \mathcal{Q}_1^+} \left\{ \frac{d_j}{\alpha_j^r} \right\}, \ \theta_1^D = \frac{d_{q_1}}{\alpha_{q_1}^r} \text{ and } \mathcal{Q}_1^+ = \mathcal{F}^+. \tag{3.30}$$

For $t = \theta_1^D$ we have $\bar{d}_{q_1} = d_{q_1} - t\alpha_{q_1}^r = 0$ and q_1 is eligible to enter the basis. The change ΔZ_1 of the dual objective function up to this point is

$$\Delta Z_1 = \theta_1^D \delta_1^D. \tag{3.31}$$

If t is further increased, such that $t > \theta_1^D$, then we get two cases dependent on the sign of $\alpha_{q_1}^r$.

- If $\alpha_{q_1} > 0$, then $\bar{d}_{q_1}$ becomes negative and goes dual infeasible since in this

Algorithm 2: Dual simplex method with simple ratio test and update of d.

```
Input: LP in computational form (2.2), dual feasible basis
       $\mathcal{B} = \{k_1, \dots, k_m\}$, primal nonbasic solution vector $x_\mathcal{N}$
Output: Optimal basis $\mathcal{B}$ with optimal primal basic solution $x$ and
        optimal dual basic solution $(y^T, d^T)^T$ or proof that LP is
        dual unbounded.

(Step 1) Initialization
         Compute $\tilde{b} = b - A_\mathcal{N} x_\mathcal{N}$ and solve $B x_\mathcal{B} = \tilde{b}$ for $x_\mathcal{B}$.
         Solve $B^T y = c_\mathcal{B}$ for $y$ and compute $d_\mathcal{N} = c_\mathcal{N} - A_\mathcal{N}^T y$.
         Compute $Z = c^T x$.

(Step 2) Pricing
         If $l_\mathcal{B} \leq x_\mathcal{B} \leq u_\mathcal{B}$ then terminate: $\mathcal{B}$ is an optimal basis.
         Else select $p = \mathcal{B}(r) \in \mathcal{B}$ with either $x_p < l_p$ or $x_p > u_p$.
         If $x_p < l_p$ set $\delta = x_p - l_p$, if $x_p > u_p$ set $\delta = x_p - u_p$.

(Step 3) BTran
         Solve $B^T \rho_r = e_r$ for $\rho_r$.

(Step 4) Pivot row
         Compute $\alpha^r = A_\mathcal{N}^T \rho_r$.

(Step 5) Ratio Test
         If $x_p < l_p$ set $\tilde{\alpha}^r = -\alpha^r$, if $x_p > u_p$ set $\tilde{\alpha}^r = \alpha^r$.
         Let $\mathcal{F} = \{j \,:\, j \in \mathcal{N}, x_j \textit{ free}$ or $(x_j = l_j$ and $\tilde{\alpha}_j^r > 0)$ or
                                       $(x_j = u_j$ and $\tilde{\alpha}_j^r < 0)\}$.
         If $\mathcal{F} = \emptyset$ then terminate: the LP is dual unbounded.
         Else determine $q \in \arg\min_{j \in \mathcal{F}} \left\{ \frac{d_j}{\tilde{\alpha}_j^r} \right\}$ and $\theta^D = \frac{d_q}{\alpha_q^r}$.

(Step 6) FTran
         Solve $B \alpha_q = a_q$ for $\alpha_q$.

(Step 7) Basis change and update
         Set $Z \leftarrow Z + \theta^D \delta$.
         Set $d_p \leftarrow -\theta^D$, $d_q \leftarrow 0$ and $d_j \leftarrow d_j - \theta^D \alpha_j^r$ for all $j \in \mathcal{N} \setminus \{q\}$.
         Compute $\theta^P = \frac{\delta}{\alpha_q^p}$ and set $x_\mathcal{B} \leftarrow x_\mathcal{B} - \theta^P \alpha_q$ and $x_q \leftarrow x_q + \theta^P$.
         Set $\mathcal{B} \leftarrow (\mathcal{B} \setminus p) \cup q$ and $\mathcal{N} \leftarrow (\mathcal{N} \setminus q) \cup p$.
         Update $B$.

         Go to step 2.
```

Algorithm 3: Dual simplex method with simple ratio test and update of y.

```
Input: LP in computational form (2.2), dual feasible basis
       $\mathcal{B} = \{k_1, \ldots, k_m\}$, primal nonbasic solution vector $x_\mathcal{N}$
Output: Optimal basis $\mathcal{B}$ with optimal primal basic solution $x$ and
        optimal dual basic solution $(y^T, d^T)^T$ or proof that LP is
        dual unbounded.

(Step 1) Initialization
         Compute $\tilde{b} = b - A_\mathcal{N} x_\mathcal{N}$ and solve $B x_\mathcal{B} = \tilde{b}$ for $x_\mathcal{B}$.
         Solve $B^T y = c_\mathcal{B}$ for $y$.
         Compute $Z = c^T x$.

(Step 2) Pricing
         If $l_\mathcal{B} \leq x_\mathcal{B} \leq u_\mathcal{B}$ then terminate: $\mathcal{B}$ is an optimal basis.
         Else select $p = k_r \in \mathcal{B}$ with either $x_p < l_p$ or $x_p > u_p$.
         If $x_p < l_p$ set $\delta = x_p - l_p$, if $x_p > u_p$ set $\delta = x_p - u_p$.

(Step 3) BTran
         Solve $B^T \rho_r = e_r$ for $\rho_r$.

(Step 4) Pivot row
         Compute $\alpha^r = A_\mathcal{N}^T \rho_r$.

(Step 5) Ratio Test
         If $x_p < l_p$ set $\tilde{\alpha}^r = -\alpha^r$, if $x_p > u_p$ set $\tilde{\alpha}^r = \alpha^r$.
         Let $\mathcal{F} = \{j : j \in \mathcal{N}, x_j \textit{ free}$ or $(x_j = l_j$ and $\tilde{\alpha}_j^r > 0)$ or
                                        $(x_j = u_j$ and $\tilde{\alpha}_j^r < 0)\}$.
         If $\mathcal{F} = \emptyset$ then terminate: the LP is dual unbounded.
         Else determine $q \in \arg\min_{j \in \mathcal{F}} \left\{ \frac{c_j - y^T a_j}{\tilde{\alpha}_j^r} \right\}$ and $\theta^D = \frac{c_q - y^T a_q}{\alpha_q^r}$.

(Step 6) FTran
         Solve $B \alpha_q = a_q$ for $\alpha_q$.

(Step 7) Basis change and update
         Set $Z \leftarrow Z + \theta^D \delta$.
         Set $y \leftarrow y + \theta^D \rho_r$.
         Compute $\theta^P = \frac{\delta}{\alpha_q^p}$ and set $x_\mathcal{B} \leftarrow x_\mathcal{B} - \theta^P \alpha_q$ and $x_q \leftarrow x_q + \theta^P$.
         Set $\mathcal{B} \leftarrow (\mathcal{B} \setminus p) \cup q$ and $\mathcal{N} \leftarrow (\mathcal{N} \setminus q) \cup p$.
         Update $B$.

         Go to step 2.
```

case $x_{q_1} = l_{q_1}$ (otherwise q_1 would not define a breakpoint). If x_{q_1} is a boxed variable though, it can be kept dual feasible by setting it to u_{q_1}.

- If $\alpha_{q_1} < 0$, then $\bar{d}_{q_1}$ becomes positive and goes dual infeasible since in this case $x_{q_1} = u_{q_1}$ (otherwise q_1 would not define a breakpoint). If x_{q_1} is a boxed variable though, it can be kept dual feasible by setting it to l_{q_1}.

Since a bound flip of a nonbasic primal variable x_{q_1} effects the values of the primal basic variables and especially x_p, also the slope of the dual objective function changes. For $x_{\mathcal{B}}$ we have

$$\begin{aligned} x_{\mathcal{B}} &= B^{-1}(b - A_{\mathcal{N}}x_{\mathcal{N}}) \\ &= B^{-1}b - B^{-1}A_{\mathcal{N}}x_{\mathcal{N}} \\ &= B^{-1}b - \sum_{j\in\mathcal{N}} (B^{-1}a_j)x_j \\ &= B^{-1}b - \sum_{j\in\mathcal{N}} \alpha_j x_j. \end{aligned} \tag{3.32}$$

Consequently, if $\alpha_{q_1} > 0$ and x_{q_1} is set from l_{q_1} to u_{q_1}, x_p changes by an amount of $-\alpha^r_{q_1}u_{q_1} + \alpha^r_{q_1}l_{q_1}$ and we get for the slope beyond δ^D_1:

$$\begin{aligned} \delta_2 &= x_p - \alpha^r_{q_1}u_{q_1} + \alpha^r_{q_1}l_{q_1} - u_p \\ &= x_p - (u_{q_1} - l_{q_1})\alpha^r_{q_1} - u_p \\ &= \delta_1 - (u_{q_1} - l_{q_1})\alpha^r_{q_1}. \end{aligned} \tag{3.33}$$

If $\alpha_{q_1} < 0$ and x_{q_1} is set from u_{q_1} to l_{q_1}, x_p changes by an amount of $\alpha^r_{q_1}u_{q_1} - \alpha^r_{q_1}l_{q_1}$ and we get for the slope beyond δ^D_1:

$$\begin{aligned} \delta_2 &= x_p + \alpha^r_{q_1}u_{q_1} - \alpha^r_{q_1}l_{q_1} - u_p \\ &= x_p + (u_{q_1} - l_{q_1})\alpha^r_{q_1} - u_p \\ &= \delta_1 + (u_{q_1} - l_{q_1})\alpha^r_{q_1}. \end{aligned} \tag{3.34}$$

Hence, in both cases the slope decreases by $(u_{q_1} - l_{q_1})|\alpha^r_{q_1}|$. If the new slope δ_2 is still positive, it is worthwhile to increase t beyond θ^D_1, until the next breakpoint θ^D_2 with

$$\theta^D_2 = \frac{d_{q_2}}{\alpha^r_{q_2}} \quad \text{with} \quad q_2 \in \arg\min_{j\in\mathcal{Q}^+_2} \left\{\frac{d_j}{\alpha^r_j}\right\} \quad \text{and } \mathcal{Q}^+_2 = \mathcal{Q}^+_1 \setminus \{q_1\}. \tag{3.35}$$

is reached. The change of the dual objective function as t moves from θ^D_1 to θ^D_2 is

$$\Delta Z_2 = (\theta^D_2 - \theta^D_1)\delta_2. \tag{3.36}$$

This procedure can be iterated until either the slope becomes negative or dual unboundedness is detected in iteration i if $\mathcal{Q}^+_i = \emptyset$. In the framework of the dual simplex algorithm these iterations are sometimes called *mini iterations*. As in the simple ratio test the case for $t < 0$ is symmetric and can be transformed to the case

$t > 0$ by switching the sign of α^r. Only the slope has to be initialized differently by setting $\delta_1 = |x_p - l_p|$.

One of the features of the BFRT is, that under certain conditions it is possible to perform an improving basis change even if the current basis is dual degenerate. To see this, suppose that the first k breakpoints $\theta_i^D, i = 1, \dots, k$ are zero (hence, $d_{q_i} = 0$ for $i = 1, \dots, k$) and $\theta_{k+1}^D > 0$. Then the eventual step length $\theta^D = d_q/\alpha_q^r$ can be strictly positive if and only if after k mini iterations the slope δ_k is still positive, i.e.

$$\delta_k = \delta_1 - \sum_{i=1}^{k} (u_{q_i} - l_{q_i})|\alpha_{q_i}^r| > 0 \tag{3.37}$$

Obviously, it depends on the magnitude of the initial primal infeasibility δ_1 and the respective entries in the pivot row $\alpha_{q_i}^r$ as well as on the distances between the individual bounds wether or not this condition can be met. It turns out however, that especially on the LP-relaxations of combinatorial optimization problems, which are often highly degenerate, the BFRT works very well. This is why the implementation of the BFRT is particularly important in dual simplex codes, which are supposed to be integrated into a branch-and-bound based MIP-code like MOPS.

When an entering variable $q = q_k$ has eventually been determined after k iterations, we have to update the primal basic variables $x_\mathcal{B}$ according to the bound flips in $x_\mathcal{N}$. Let $\mathcal{T} = \{q_1, \dots, q_k\}$ be the set of indices of all of the nonbasic variables, which are to be set to their opposite bound, $\mathcal{T}^+ = \{j \in \mathcal{T} \,|\, \alpha_j^r > 0\}$ the set of indices of those nonbasic variables, which switch from lower to upper bound and $\mathcal{T}^- = \{j \in \mathcal{T} \,|\, \alpha_j^r < 0\}$ the set of indices of those nonbasic variables, which switch from upper to lower bound. Then, according to equation 3.32, $x_\mathcal{B}$ can be updated in the following way:

$$\begin{aligned}
\bar{x}_\mathcal{B} &= x_\mathcal{B} - \sum_{j\in\mathcal{T}^+} (u_j - l_j)\alpha_j - \sum_{j\in\mathcal{T}^-} (l_j - u_j)\alpha_j \\
&= x_\mathcal{B} - B^{-1}\left(\sum_{j\in\mathcal{T}^+} (u_j - l_j)a_j + \sum_{j\in\mathcal{T}^-} (l_j - u_j)a_j\right) \\
&= x_\mathcal{B} - \Delta x_\mathcal{B},
\end{aligned} \tag{3.38}$$

where $\Delta x_\mathcal{B}$ is the solution vector of the linear system

$$B\Delta x_\mathcal{B} = \tilde{a} \quad \text{with} \quad \tilde{a} = \sum_{j\in\mathcal{T}^+} (u_j - l_j)a_j + \sum_{j\in\mathcal{T}^-} (l_j - u_j)a_j. \tag{3.39}$$

There are different ways to embed the bound flipping ratio test into the revised dual simplex method given in the form of algorithm 2. In both Fourer's and Maros' description the selection of the leaving variable and the update operations for $x_\mathcal{B}$ and Z are integrated in the sense, that the sets $\mathcal{T}^+$ and $\mathcal{T}^-$ as well as the summation of $\tilde{a}$ are done in the main selection loop for q. We will present a version that is motivated by the dual simplex implementation in the COIN LP code [44], where the update operations are strictly separated from the selection of the entering variable.

Even the bound flips are not recorded in the ratio test but during the update of the reduced cost vector d. If an updated entry d_j is detected to be dual infeasible, the corresponding nonbasic primal variable (which must be boxed in this case) is marked for a bound flip and column a_j is added to $\tilde{a}$. After the update of d, system (3.39) is solved and $x_\mathcal{B}$ is updated according to (3.38). The actual bound flips in $x_\mathcal{N}$ are only performed at the very end of a major iteration.

The three parts selection of the entering variable, update of d and $x_\mathcal{B}$ and update of $x_\mathcal{N}$ are described in algorithms 4, 5 and 6 respectively. The update of the objective function value can be done in two different ways. One way is to compute the objective change in the selection loop for q in algorithm 4 by an operation of type (3.36). The other way is to look at the objective function value from a primal perspective. From equation (2.18) we know, that in every iteration of the dual simplex method the primal and the dual objective function value is equal for the current primal and dual solution (provided that v and w are set according to equation (2.17)):

$$c_\mathcal{B}^T x_\mathcal{B} + c_\mathcal{N}^T x_\mathcal{N} = b^T y + l^T v + u^T w. \tag{3.40}$$

Hence, we can as well compute the change in the objective function value by taking into account the change of the primal basic and nonbasic variables caused by the bound flips:

$$\Delta Z = -\sum_{j \in \mathcal{B}} c_j \Delta x_j + \sum_{j \in \mathcal{T}^+} c_j (u_j - l_j) + \sum_{j \in \mathcal{T}^+} c_j (u_j - l_j). \tag{3.41}$$

Thus, also this update operation can be separated from the selection loop for q and moved to algorithm 5.

3.3 Dual steepest edge pricing

In the pricing step of the dual simplex method we have to select a variable to leave the basis among all of those variables which are primal infeasible. In geometrical terms, this corresponds to choosing a search direction along one of the edges of the dual polyhedron emanating from the vertex defined by the current basic solution, which makes an acute[3] angle with the gradient of the dual objective function. The basic idea of *dual steepest edge pricing (DSE)* is to determine the edge direction that forms the most acute angle with the dual gradient and is in this sense *steepest*.

We will follow the description of Forrest and Goldfarb [26], who described the first practical steepest edge variants for the dual simplex method. Surprisingly, they showed in numerical experiments, that their simplest version, which they call *Dual algorithm I* performs best in most cases.

This variant of DSE is derived for a problem in computational standard form (2.5).

[3]since we are maximizing

Algorithm 4: Selection of q with the BRFT.

```
If $x_p < l_p$ set $\tilde{\alpha}^r \leftarrow -\alpha^r$, if $x_p > u_p$ set $\tilde{\alpha}^r \leftarrow \alpha^r$.
Let $\mathcal{Q} \leftarrow \{j : j \in \mathcal{N}, x_j$ free or $(x_j = l_j$ and $\tilde{\alpha}^r_j > 0)$ or
                                    $(x_j = u_j$ and $\tilde{\alpha}^r_j < 0)\}$.
while $\mathcal{Q} \neq \emptyset$ and $\delta \geq 0$ do
    Select $q \in \arg\min_{j \in \mathcal{Q}} \left\{ \frac{d_j}{\tilde{\alpha}^r_j} \right\}$.
    Set $\delta \leftarrow \delta - (u_q - l_q)|\alpha^r_q|$.
    Set $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{q\}$.
end
If $\mathcal{Q} = \emptyset$ then terminate: the LP is dual unbounded.
Set $\theta^D \leftarrow \frac{d_q}{\alpha^r_q}$.
```

Algorithm 5: Update of d and $x_{\mathcal{B}}$ for the BRFT.

```
Set $\tilde{a} \leftarrow 0$, $\mathcal{T} \leftarrow \emptyset$ and $\Delta Z \leftarrow 0$.
forall $j \in \mathcal{N}$ do
    Set $d_j \leftarrow d_j - \theta^D \alpha^r_j$.
    if $l_j \neq u_j$ then
        if $x_j = l_j$ and $d_j < 0$ then
            Set $\mathcal{T} \leftarrow \mathcal{T} \cup \{j\}$.
            Set $\tilde{a} \leftarrow \tilde{a} + (u_j - l_j)a_j$.
            Set $\Delta Z \leftarrow \Delta Z + (u_j - l_j)c_j$.
        else if $x_j = u_j$ and $d_j > 0$ then
            Set $\mathcal{T} \leftarrow \mathcal{T} \cup \{j\}$.
            Set $\tilde{a} \leftarrow \tilde{a} + (l_j - u_j)a_j$.
            Set $\Delta Z \leftarrow \Delta Z + (l_j - u_j)c_j$.
        end
    end
end
Set $d_p \leftarrow -\theta^D$.
if $\mathcal{T} \neq \emptyset$ then
    Solve $B\Delta x_{\mathcal{B}} = \tilde{a}$ for $\Delta x_{\mathcal{B}}$.
    Set $x_{\mathcal{B}} \leftarrow x_{\mathcal{B}} - \Delta x_{\mathcal{B}}$.
    Set $\Delta Z \leftarrow \Delta Z - \sum_{j \in \mathcal{B}} c_j \Delta x_j$.
end
Set $Z \leftarrow Z + \Delta Z$.
```

Algorithm 6: Update of $x_{\mathcal{N}}$ for the BRFT.

```
forall $j \in \mathcal{T}$ do
    if $x_j = l_j$ then
        Set $x_j \leftarrow u_j$.
    else
        Set $x_j \leftarrow l_j$.
    end
end
```

The dual of (2.5) reads:

$$Z^* = \max \quad b^T y \tag{3.42a}$$
$$\text{s.t.} \quad A^T y \leq c. \tag{3.42b}$$

Given a basis $\mathcal{B}$ a dual solution y is *basic* for problem (3.42), if $B^T y = c_{\mathcal{B}}$ and *feasible*, if $A_{\mathcal{N}}^T y \leq c_{\mathcal{N}}$. If the i-th tight basic constraint $j \in \mathcal{B}$ is relaxed, such that $a_j^T \bar{y} + t = c_j$, i.e., $a_j^T y \leq c_j$ for $t \geq 0$, then all other basic constraint stay tight, if $B^T \bar{y} + t e_i = c_{\mathcal{B}}$, which leads to $\bar{y} = y - t\rho_i$ with $\rho_i = B^{-T} e_i$. Hence, $-\rho_i$ is the edge direction associated with the i-th potential basic leaving variable.

Obviously, the gradient of the dual objective function (3.42a) is b. The angle γ between b and $-\rho_i$ is acute, if $-\rho_i^T b = -e_i^T B^{-1} b = -e_i^T x_{\mathcal{B}} = -x_{\mathcal{B}(i)}$ is positive ($x_{\mathcal{B}(i)}$ primal infeasible with $x_{\mathcal{B}(i)} \leq 0$). Furthermore, we have

$$-x_{\mathcal{B}(i)} = -b^T \rho_i = \|b\| \|\rho_i\| \cos\gamma, \tag{3.43}$$

where $\|\cdot\|$ is the Euclidian norm. Since $\|b\|$ is constant for all edge directions, we can determine the leaving variable $p = \mathcal{B}(r)$ with the steepest direction by

$$r \in \underset{i \in \{1,\ldots,m\}}{\arg\max} \left\{ \frac{-x_{\mathcal{B}(i)}}{\|\rho_i\|} \; : \; x_{\mathcal{B}(i)} < 0 \right\}. \tag{3.44}$$

To compute the norms $\|\rho_i\|$ from their definition would require the solution of up to m systems of linear equations (one for each vector ρ_i) and just as many inner products. The actual accomplishment of Forrest and Goldfarb is the presentation of exact update formulas for the squares of these norms, which we will denote by

$$\beta_i = \rho_i^T \rho_i = \|\rho_i\|^2. \tag{3.45}$$

The values β_i are called *dual steepest edge weights*. Using the weights instead of the norms is straightforward, the selection rule (3.44) just has to be modified as follows:

$$r \in \underset{i \in \{1,\ldots,m\}}{\arg\max} \left\{ \frac{(x_{\mathcal{B}(i)})^2}{\beta_i} \; : \; x_{\mathcal{B}(i)} < 0 \right\}. \tag{3.46}$$

From the update formulas (3.27) for the ρ_i we can conclude, that the DSE weights $\bar{\beta}_i$ after the basis change are

$$\bar{\beta}_r = \bar{\rho}_r^T \bar{\rho}_r = \left(\frac{1}{\alpha_q^r} \right)^2 \beta_r \tag{3.47a}$$

and

$$\begin{aligned}\bar{\beta}_i &= \bar{\rho}_i^T \bar{\rho}_i \\ &= \left(\rho_i^T - \frac{\alpha_q^i}{\alpha_q^r} \rho_r^T \right) \left(\rho_i - \frac{\alpha_q^i}{\alpha_q^r} \rho_r \right)\end{aligned}$$

$$= \beta_i - 2\frac{\alpha_q^i}{\alpha_q^r}\rho_r^T\rho_i + \left(\frac{\alpha_q^i}{\alpha_q^r}\right)^2 \beta_r \qquad \text{for } i = 1, \dots, m, i \neq p. \tag{3.47b}$$

Since we do not want to calculate all the ρ_i explicitly during a dual iteration, it is not obvious, how to efficiently compute the term $\rho_r^T\rho_i$. Fortunately, we can write

$$\rho_r^T\rho_i = \rho_i^T\rho_r = e_i^T B^{-1}\rho_r = e_i^T\tau = \tau_i, \tag{3.48}$$

where $\tau = B^{-1}\rho_r$. Hence, it suffices to solve only one additional linear system for τ:

$$B\tau = \rho_r. \tag{3.49}$$

Due to (3.48) equation (3.47b) can be rewritten as

$$\bar{\beta}_i = \beta_i - 2\frac{\alpha_q^i}{\alpha_q^r}\tau_i + \left(\frac{\alpha_q^i}{\alpha_q^r}\right)^2 \beta_r. \tag{3.50}$$

Note that numerical stability can be improved at small additional computational costs by computing β_r explicitly, since ρ_r is available from the computation of the transformed pivot row (see step 2 in algorithm 2).

Besides the geometrical motivation of DSE pricing it can also be viewed as a strategy to reduce or even eliminate misleading scaling effects when using the simple Dantzig pricing rule. To make this clear, suppose the columns of A are scaled by numbers $\frac{1}{s_j} > 0$, such that $\hat{a}_j = \frac{1}{s_j}a_j$. Since $\hat{\rho}_i\hat{a}_{\mathcal{B}(i)} = 1$, this leads to scaled vectors $\hat{\rho}_i = s_{\mathcal{B}(i)}\rho_i$ and hence, to scaled primal basic variables $\hat{x}_{\mathcal{B}(i)} = \hat{\rho}_i b = s_{\mathcal{B}(i)}\rho_i b = s_{\mathcal{B}(i)}x_{\mathcal{B}(i)}$. Consequently, the magnitude of the primal infeasibilities totally depends on the (problem inherent) column scales, which may result in a suboptimal pricing decision by the Dantzig rule. However, dividing the primal basic variables by the Euclidian norms $\|\hat{\rho}_i\|$ yields primal infeasibilities which are independent of the column scales:

$$\tilde{x}_{\mathcal{B}(i)} = \frac{\hat{x}_{\mathcal{B}(i)}}{\|\hat{\rho}_i\|} = \frac{s_{\mathcal{B}(i)}x_{\mathcal{B}(i)}}{\|s_{\mathcal{B}(i)}\rho_i\|} = \frac{x_{\mathcal{B}(i)}}{\|\rho_i\|}. \tag{3.51}$$

In the case of the general computational form (2.2) and its dual (2.11) the edge directions need to be extended by additional components for v and w. This leads to computationally more expensive update formulas for the DSE weights (see [26]). In fact, in this case one more linear system and an inner product have to be solved. It has been shown that for most problems this additional effort does not pay off, i.e., it is not compensated by the additional reduction in the total number of iterations. Therefore, we keep using the DSE weights from above to normalize the primal infeasibilities even if upper and lower bounds are present. Denoting the primal infeasibilities by

$$\delta_i = \begin{cases} x_{\mathcal{B}(i)} - l_{\mathcal{B}(i)} & \text{if } x_{\mathcal{B}(i)} < l_{\mathcal{B}(i)} \\ x_{\mathcal{B}(i)} - u_{\mathcal{B}(i)} & \text{if } x_{\mathcal{B}(i)} > u_{\mathcal{B}(i)} \\ 0 & \text{o.w.} \end{cases} \tag{3.52}$$

the leaving variable is selected by

$$r \in \arg\max_{i\in\{1,\dots,m\}} \left\{ \frac{|\delta_i|^2}{\beta_i} \,:\, \delta_i \neq 0 \right\}. \tag{3.53}$$

This turns the method into a heuristic in the sense, that we cannot guarantee anymore that always the leaving variable associated with the steepest edge direction is chosen. The efficiency and actual performance of DSE pricing depends heavily on implementation details. We give a detailed description of our DSE code in section 8.2.2.

3.4 Elaborated version of the Dual Simplex Algorithm

Algorithm 7 shows a complete mathematical description of the dual simplex method as it is implemented in MOPS. It can be seen as a variant of algorithm 2 enhanced by steepest pricing (described in section 3.3) and the bound flipping ratio test (described in section 3.2).

Note, that in this version of the method, up to four systems of linear equations have to be solved in every iteration:

- $B^T \rho_r = e_r$ in step 7 to compute the transformed pivot row $\alpha^r = \rho_r^T A_{\mathcal{N}}$, which is needed in the ratio test,
- $B\alpha_q = a_q$ in step 7 to compute the transformed pivot column α_q, which is needed to update $x_{\mathcal{B}}$ and B,
- $B\tau = \rho_r$ in step 7 to update the steepest edge weights and
- $B\Delta x_{\mathcal{B}} = \tilde{a}$ in algorithm 5 to update the primal basic variable if bound flips have to be performed.

Besides the computation of the transformed pivot row in step 7 these are the most time consuming operations for a great majority of problems. In chapter 5 we will describe the computational techniques that are deployed to perform them efficiently.

Furthermore, algorithm 7 can only be applied if we have a dual feasible basic solution to start with. If the default starting basis (usually consisting of the identity matrix I associated with the logical variables) is not dual feasible we have to proceed in two phases. In the first phase, we try to obtain a dual feasible basis by so called *dual phase I methods.* A possible outcome of these methods is that no dual feasible solution exists for the given LP problem and we have to terminate (or try to further analyse, wether the problem is also primal infeasible or primal unbounded). If phase I succeeds in obtaining a dual feasible basis we call algorithm 7 as a second phase to prove optimality or dual unboundedness. In this sense, we will call algorithm 7 *dual phase II.* In the following chapter 4 we will describe several dual phase I methods, which we will evaluate and compare computationally in chapter 9.

Algorithm 7: Dual simplex method with dual steepest edge pricing, bound flipping ratio test and update of d.

```
Input: LP in computational form (2.2), dual feasible basis
       $\mathcal{B} = \{k_1, \ldots, k_m\}$, primal nonbasic solution vector $x_\mathcal{N}$
Output: Optimal basis $\mathcal{B}$ with optimal primal basic solution $x$ and
        optimal dual basic solution $(y^T, d^T)^T$ or proof that LP is
        dual unbounded.

(Step 1) Initialization
         Compute $\tilde{b} = b - A_\mathcal{N} x_\mathcal{N}$ and solve $B x_\mathcal{B} = \tilde{b}$ for $x_\mathcal{B}$.
         Solve $B^T y = c_\mathcal{B}$ for $y$ and compute $d_\mathcal{N} = c_\mathcal{N} - A_\mathcal{N}^T y$.
         Initialize $\beta$ (set $\beta^T \leftarrow (1, \ldots, 1)^T$ if $B = I$).
         Compute $Z = c^T x$.
(Step 2) Pricing
         If $l_\mathcal{B} \le x_\mathcal{B} \le u_\mathcal{B}$ then terminate: $\mathcal{B}$ is an optimal basis.
         Else select $p = \mathcal{B}(r)$ by formula (3.53).
         If $x_p < l_p$ set $\delta = x_p - l_p$, if $x_p > u_p$ set $\delta = x_p - u_p$.
(Step 3) BTran
         Solve $B^T \rho_r = e_r$ for $\rho_r$.
(Step 4) Pivot row
         Compute $\alpha^r = A_\mathcal{N}^T \rho_r$.
(Step 5) Ratio Test
         Determine $q$ by algorithm 4.
(Step 6) FTran
         Solve $B \alpha_q = a_q$ for $\alpha_q$.
(Step 7) DSE FTran
         Solve $B \tau = \rho_r$ for $\tau$.
(Step 8) Basis change and update
         Update $d$ by algorithm 5.
         Compute $\theta^P = \frac{\delta}{\alpha_q^p}$ and set $x_\mathcal{B} \leftarrow x_\mathcal{B} - \theta^P \alpha_q$ and $x_q \leftarrow x_q + \theta^P$.
         Update $\beta$ by formulas (3.47a) and (3.50).
         Set $\mathcal{B} \leftarrow (\mathcal{B} \setminus p) \cup q$ and $\mathcal{N} \leftarrow (\mathcal{N} \setminus q) \cup p$.
         Update $B$.
         Flip bounds in $x_\mathcal{N}$ by algorithm 6.
         Set $Z \leftarrow Z + \theta^D \delta$.

         Go to step 7.
```

Chapter 4

Dual Phase I Methods

4.1 Introduction

We refer to an algorithm which produces a dual feasible basis for an arbitrary LP as a dual phase I method. The dual simplex method presented in the previous chapter is called dual phase II in this context. Suppose in the following, that we are given a (start) basis $\mathcal{B}$ and a corresponding primal basic solution x and dual basic solution $(y^T, v^T, w^T)^T$, where v and w are set according to (2.17) with reduced costs d.

4.1.1 Big-M method

The simplest method to obtain a dual feasible basis for LP problems in computational standard form (2.5) (no upper bounds, nonnegative variables only) is to introduce an additional dual slack variable and punish its use in the dual objective function by some sufficiently large cost coefficient M. This is equivalent to adding a constraint of the form $\sum_{j \in \mathcal{J}} x_j < M$ to the primal problem. To apply this method to a problem in computational form (2.2) one could convert it to a problem in standard form by introducing additional variables and constraints. This would increase the size of the problem considerably. The other reason why this method is usually not used in practise is that a high value of M can lead to numerical problems and high iteration counts whereas a too small value might not produce a primal feasible solution. See for example [53] for further details. In the following, we only consider methods, that do not increase the size of the problem.

4.1.2 Dual feasibility correction

In most practical problems, many variables have finite lower and upper bounds. As we have seen in the description of the bound flipping ratio test dual infeasible boxed variables can be made dual feasible by setting them to their respective opposite bound. As in section 3.2 we collect the variables which have to be flipped in two sets

$$\mathcal{T}^+ = \{j \in \mathcal{J} \,:\, x_j = l_j \text{ and } u_j < \infty \text{ and } d_j < 0\} \tag{4.1a}$$

$$\mathcal{T}^- = \{j \in \mathcal{J} \,:\, x_j = u_j \text{ and } l_j > \infty \text{ and } d_j > 0\}, \tag{4.1b}$$

where again the variables in $\mathcal{T}^+$ go from lower to upper bound and the variables in $\mathcal{T}^-$ go from upper to lower bound. Then, the primal basic variables $x_{\mathcal{B}}$ and the dual objective function value Z can be updated according to equations (3.38), (3.39) and (3.41). Algorithmicly, this is equivalent to calling algorithms 5 and 6 with $\theta^D = 0$.

The above procedure is called *dual feasibility correction (DFC)* and can be performed prior to a dual phase I method. Alternatively, we can mark boxed and fixed variables as not eligible during dual phase I, and call DFC prior to dual phase II. In any case this means, that only dual infeasibilities caused by non-boxed variables have to be tackled by dual phase I algorithms, which is usually only a very small fraction of the total number of variables.

4.2 Minimizing the sum of dual infeasibilities

The idea of this approach is to formulate the task of finding a dual feasible basic solution as the subproblem of minimizing the total amount of dual infeasibility. This subproblem can be formulated as an LP problem, which can either be solved directly by the standard dual simplex method (phase II) or by specialized versions of it which exploit its structural characteristics. In section 4.2.1 we present our version of the subproblem approach which is similar to the procedure of Kostina in [41]. In section 4.2.2 we will describe the algorithmic approach following Maros [48]. It differs from the version described by Fourer [27] only in the way the extended ratio test is applied: while Fourer only allows flips from dual infeasible to dual feasible Maros also allows the contrary, both under the precondition that the sum of dual infeasibilities decreases. To put it differently: Fourer's algorithm is monotone both in the sum and in the number of dual infeasibilities, while Maros's algorithm is monotone only in the sum of dual infeasibilities.

4.2.1 Subproblem approach

As discussed in section 4.1.2 we only have to consider dual infeasibilities associated with non-boxed primal variables. Let

$$\mathcal{J}^u = \{j \in \mathcal{J} \,:\, l_j = -\infty \text{ and } u_j < \infty\} \tag{4.2}$$

be the set of non-free variables that go dual infeasible if $d_j > 0$,

$$\mathcal{J}^l = \{j \in \mathcal{J} \,:\, l_j > -\infty \text{ and } u_j = \infty\} \tag{4.3}$$

be the set of non-free variables that go dual infeasible if $d_j < 0$ and

$$\mathcal{J}^f = \{j \in \mathcal{J} \,:\, l_j = -\infty \text{ and } u_j = \infty\} \tag{4.4}$$

be the set of free variables. Now, we can state the problem of finding a basis with a minimal sum of dual infeasibilities as follows:

$$\max \quad Z_0 = \sum_{\substack{j\in\mathcal{J}^l\cup\mathcal{J}^f \\ d_j<0}} d_j - \sum_{\substack{j\in\mathcal{J}^u\cup\mathcal{J}^f \\ d_j>0}} d_j \tag{4.5a}$$

$$\text{s.t.} \quad a_j^T y + d_j = c_j \qquad \text{for all } j \in \mathcal{J}^l \cup \mathcal{J}^u \cup \mathcal{J}^f. \tag{4.5b}$$

Problem (4.5) is equivalent to the following formulation:

$$\max \quad Z_0 = \sum_{j\in\mathcal{J}^l\cup\mathcal{J}^f} w_j - \sum_{j\in\mathcal{J}^u\cup\mathcal{J}^f} v_j \tag{4.6a}$$

$$\text{s.t.} \quad a_j^T y + v_j + w_j = c_j \qquad \text{for all } j \in \mathcal{J}^l \cup \mathcal{J}^u \cup \mathcal{J}^f \tag{4.6b}$$

$$v_j \geq 0 \tag{4.6c}$$

$$w_j \leq 0. \tag{4.6d}$$

The dual of problem (4.6) is

$$\min \quad z_0 = \sum_{j\in\mathcal{J}^l\cup\mathcal{J}^u\cup\mathcal{J}^f} c_j x_j \tag{4.7a}$$

$$\sum_{j\in\mathcal{J}^l\cup\mathcal{J}^u\cup\mathcal{J}^f} a_j x_j = 0 \tag{4.7b}$$

$$-1 \leq x_j \leq 0 \qquad \text{for all } j \in \mathcal{J}^u \tag{4.7c}$$

$$0 \leq x_j \leq 1 \qquad \text{for all } j \in \mathcal{J}^l \tag{4.7d}$$

$$-1 \leq x_j \leq 1 \qquad \text{for all } j \in \mathcal{J}^f. \tag{4.7e}$$

Note, that problem (4.7) is a reduced version of our original problem (2.2) in the sense that it consists of a subset of the original set of columns and that bounds and right-hand-side is changed. Since all variables of problem (4.7) are boxed every given starting basis can be made dual feasible by dual feasibility correction and dual phase II as described in algorithm 7 can directly be applied. If eventually it yields $z_0 = 0$, we have a dual feasible basis for our original problem and can start the dual phase II on problem (2.2). If $z_0 < 0$, the original problem is dual infeasible. The complete approach is summarized in algorithm 8.

Fourer mentions in [27] that for model (4.5) the dual pricing step can be simplified: only those variables need to be considered in the selection process, which will become dual feasible after the basis change. Therefore, in our approach only those variables are eligible to leave the basis, which go to their zero bound (being dual feasible in the original problem).

Furthermore, we actually use a slightly modified version of the artificial problem (4.7) in our implementation. To favor free variables to be pivoted into the basis we give a weight of 10^4 to them in the sum of dual infeasibilities (4.5a). This translates into modified individual bounds in the primal bound constraints (4.7e), which change to

$$-10^4 \leq x_j \leq 10^4 \qquad \text{for all } j \in \mathcal{J}^f. \tag{4.8}$$

Algorithm 8: The 2-phases dual simplex method with subproblem dual phase I.

```
          Input: LP in computational form (2.2).
          Output: Optimal basis $\mathcal{B}$ or proof that LP is dual unbounded.
(Step 1)  Mark boxed and fixed variables as not eligible.
(Step 2)  Change the bounds of the remaining variables and the
          right-hand-side vector according to problem (4.7).
(Step 3)  Start with an initial basis and make it dual feasible by dual
          feasibility correction (call algorithms 5 and 6 with $\theta^D = 0$).
(Step 4)  Execute algorithm 7 on auxiliary problem (4.7).
(Step 5)  If $z_0 < 0$, then original problem (2.2) is dual infeasible, stop.
(Step 6)  If $z_0 = 0$, the current basis is dual feasible for problem (2.2).
(Step 7)  Unmark boxed and fixed variables and restore original bounds and
          right-hand-side.
(Step 8)  Execute algorithm 7 on the original problem (2.2).
```

4.2.2 Algorithmic approach

The goal of this section is to develop a specialized dual simplex type algorithm which solves problem (4.5). Suppose, we are given a start basis $\mathcal{B}$. For easier notation we define two index sets

$$\mathcal{P} = \{j \in \mathcal{J}^u \cup \mathcal{J}^f : d_j > 0\} \tag{4.9}$$

and

$$\mathcal{M} = \{j \in \mathcal{J}^l \cup \mathcal{J}^f : d_j < 0\}, \tag{4.10}$$

which contain those (non-basic) variables that are dual infeasible with positive and negative reduced costs, respectively. We do not need to consider boxed variables, since they can be made dual feasible by feasibility correction (s. section 4.1.2).

Now we can rewrite the (negative) sum of dual infeasibilities as

$$Z_0 = \sum_{j \in \mathcal{M}} d_j - \sum_{j \in \mathcal{P}} d_j. \tag{4.11}$$

When we relax the r-th dual basic constraint (with $\mathcal{B}(r) = p$) by an amount of t the reduced costs change according to equations (3.6). If we select the leaving variable and the direction of relaxation in such a way that it will be dual feasible after the basis change, it will not contribute to the sum of infeasibilities. Hence, as long as no nonbasic d_j changes sign, Z_0 can be expressed as a function of t:

$$Z_0(t) = \sum_{j \in \mathcal{M}} d_j(t) - \sum_{j \in \mathcal{P}} d_j(t)$$

$$\begin{aligned}
&= \sum_{j\in\mathcal{M}} (d_j - t\alpha_j^r) - \sum_{j\in\mathcal{P}} (d_j - t\alpha_j^r) \\
&= Z_0 - t\left(\sum_{j\in\mathcal{M}} \alpha_j^r - \sum_{j\in\mathcal{P}} \alpha_j^r\right) \\
&= Z_0 - tf_r, \qquad (4.12)
\end{aligned}$$

where f_r is the r-the entry of the *dual phase I pricing vector*

$$f = \sum_{j\in\mathcal{M}} \alpha_j - \sum_{j\in\mathcal{P}} \alpha_j = B^{-1}\left(\sum_{j\in\mathcal{M}} a_j - \sum_{j\in\mathcal{P}} a_j\right). \qquad (4.13)$$

Since we want to maximize Z_0 we have to determine r and t such that $-tf_r$ is positive. Furthermore, x_p has be to dual feasible after the basis change, i.e., it has to leave at the right finite bound given the sign of t. Hence,

$$\begin{aligned}
&\text{if} \quad f_r > 0 \quad \text{we need} \quad t < 0 \text{ and } l_{\mathcal{B}(r)} > -\infty \qquad \text{and} \qquad (4.14)\\
&\text{if} \quad f_r < 0 \quad \text{we need} \quad t > 0 \text{ and } u_{\mathcal{B}(r)} < \infty. \qquad (4.15)
\end{aligned}$$

If no such r exists, we can stop. If $Z_0 = 0$, the current basis is dual feasible and we can start dual phase II. If $Z_0 > 0$, the problem is dual infeasible. Otherwise, we choose a leaving index $p = \mathcal{B}(r)$ according to some reasonable pricing rule.

In the next step, we want to determine an entering index. As t is increased (or decreased) from zero a nonbasic dual constraint j gets eligible to enter the basis when it becomes tight, i.e., if

$$\begin{aligned}
d_j(t) = d_j - t\alpha_j^r = 0 \\
\Leftrightarrow \quad t = \frac{d_j}{\alpha_j^r}.
\end{aligned} \qquad (4.16)$$

Suppose for now, that $t > 0$. Then the set of positive breakpoints consists of those nonbasic positions, where d_j and α_j^r have the same sign:

$$\mathcal{F}_0^+ = \{j \in \mathcal{N} \,:\, (d_j \geq 0 \text{ and } \alpha_j^r > 0) \text{ or } (d_j \leq 0 \text{ and } \alpha_j^r < 0)\}. \qquad (4.17)$$

If we choose the dual nonbasic constraint associated with the first (smallest) breakpoint to enter the basis, i.e.,

$$q \in \arg\min_{j\in\mathcal{F}_0^+} \left\{\frac{d_j}{\alpha_j^r}\right\}, \quad \theta^D = \frac{d_q}{\alpha_q^r}, \qquad (4.18)$$

we can assure, that only this constraint possibly changes its feasibility status after the basis change. We refer to the selection of the entering variable according to (4.18) as *simple ratio test* in dual phase I.

If q is dual feasible (neither in $\mathcal{P}$ nor in $\mathcal{M}$), then it stays dual feasible when it enters the basis and the infeasibility sets do not change. In this case, we can conclude from equations (4.13) and (3.24), that f can be updated by

$$\bar{f} = Ef. \tag{4.19}$$

If q is dual infeasible with $q \in \mathcal{P}$, it leaves $\mathcal{P}$, which means, that α_q has to be withdrawn from the sum in (4.13). Hence, in this case the update formula for f reads

$$\bar{f} = E(f + \alpha_q). \tag{4.20}$$

Consequently, if $q \in \mathcal{M}$, it leaves $\mathcal{M}$, and f can be updated by

$$\bar{f} = E(f - \alpha_q). \tag{4.21}$$

Since $E\alpha_q = (0, \dots, 0)^T$, equation (4.19) is true in all of the three cases. It is equivalent to the following update operations:

$$\bar{f}_i = f_i - \frac{f_r}{\alpha_q^r}\alpha_q^i \qquad \text{for } i = 1, \dots, m,\ i \neq r \tag{4.22a}$$

$$\text{and} \quad \bar{f}_r = f_r - \frac{f_r}{\alpha_q^r}. \tag{4.22b}$$

To finish the iteration it remains to update Z_0 by

$$\bar{Z}_0 = Z_0 - \theta^D f_r, \tag{4.23}$$

and the reduced cost vector d and the basis information consisting of the sets $\mathcal{B}$, $\mathcal{N}$ and some representation of B^{-1} as discussed in section 3.1.5. The dual phase I method by minimizing the sum of dual infeasibilities and simple ratio test is summarized in algorithm 9.

As in dual phase II variants of the ratio test have been developed, which do not necessarily select the leaving index corresponding to the first breakpoint. The idea is to pass by breakpoints and increase t as long as the sum of dual infeasibilities decreases. This is the case as long as f_r does not switch sign. At the beginning of the ratio test we have

$$f_r = \sum_{j \in \mathcal{M}} \alpha_j^r - \sum_{j \in \mathcal{P}} \alpha_j^r. \tag{4.24}$$

Every breakpoint $j \in \mathcal{F}_0^+$ falls in one of the following categories:

1. If j is dual infeasible with $j \in \mathcal{P}$ and $j \notin \mathcal{J}^f$, it leaves $\mathcal{P}$ and becomes dual feasible. Thus, we get for $\bar{f}_r$:

$$\bar{f}_r = f_r + \alpha_j^r. \tag{4.25}$$

2. If j is dual infeasible with $j \in \mathcal{M}$ and $j \notin \mathcal{J}^f$, it leaves $\mathcal{M}$ and becomes dual feasible. Thus, we get for $\bar{f}_r$:

$$\bar{f}_r = f_r - \alpha_j^r. \tag{4.26}$$

Algorithm 9: Dual phase 1: Minimizing the sum of dual infeasibilities with simple ratio test.

```
Input: LP in computational form (2.2), start basis
       $\mathcal{B} = \{k_1, \ldots, k_m\}$
Output: Dual feasible basis $\mathcal{B}$ with dual feasible basic solution
        $(y^T, d^T)^T$ or proof that LP is dual infeasible.

(Step 1) Initialization
         Solve $B^T y = c_\mathcal{B}$ for $y$ and compute $d_\mathcal{N} = c_\mathcal{N} - A_\mathcal{N}^T y$.
         Compute $Z_0 = \sum_{j \in \mathcal{M}} d_j + \sum_{j \in \mathcal{P}} d_j$ with $\mathcal{P}$ and $\mathcal{M}$ as in
         equations (4.9) and (4.10).
         Compute $\tilde{a} = \sum_{j \in \mathcal{M}} a_j - \sum_{j \in \mathcal{P}} a_j$ and solve $Bf = \tilde{a}$ for $f$.

(Step 2) Pricing
         Let $\mathcal{H} = \{i \in \{1, \ldots, m\} : (f_i > 0 \text{ and } l_{\mathcal{B}(i)} > -\infty)$ or
                                      $(f_i < 0 \text{ and } l_{\mathcal{B}(i)} < \infty)\}$.
         If $\mathcal{H} = \emptyset$ then terminate:
            If $Z_0 = 0$, $\mathcal{B}$ is a dual feasible basis.
            If $Z_0 > 0$, the problem is dual infeasible.
         Else select $r \in \mathcal{H}$, $p = \mathcal{B}(r)$ according to a reasonable pricing rule.

(Step 3) BTran
         Solve $B^T \rho_r = e_r$ for $\rho_r$.

(Step 4) Pivot row
         Compute $\alpha^r = A_\mathcal{N}^T \rho_r$.

(Step 5) Simple ratio test
         If $f_r > 0$ set $\tilde{\alpha}^r = -\alpha^r$, if $f_r < 0$ set $\tilde{\alpha}^r = \alpha^r$.
         Let $\mathcal{F} = \{j \in \mathcal{N} : (d_j \geq 0 \text{ and } \alpha_j^r > 0) \text{ or } (d_j \leq 0 \text{ and } \alpha_j^r < 0)\}$.
         Determine $q \in \arg\min_{j \in \mathcal{F}} \left\{ \frac{d_j}{\tilde{\alpha}_j^r} \right\}$ and $\theta^D = \frac{d_q}{\alpha_q^r}$.

(Step 6) FTran
         Solve $B\alpha_q = a_q$ for $\alpha_q$.

(Step 7) Basis change and update
         Set $Z_0 \leftarrow Z_0 - \theta^D f$.
         Set $d_p \leftarrow -\theta^D$, $d_q \leftarrow 0$ and $d_j \leftarrow d_j - \theta^D \alpha_j^r$ for all $j \in \mathcal{N} \setminus \{q\}$.
         Set $f_i \leftarrow f_i - \frac{f_r}{\alpha_q^r} \alpha_q^i$ for $i = 1, \ldots, m$, $i \neq r$ and $f_r \leftarrow f_r - \frac{f_r}{\alpha_q^r}$.
         Set $\mathcal{B} \leftarrow (\mathcal{B} \setminus p) \cup q$ and $\mathcal{N} \leftarrow (\mathcal{N} \setminus q) \cup p$.
         Update $B$.

         Go to step 9.
```

3. If j is dual infeasible with $j \in \mathcal{P}$ and $j \in \mathcal{J}^f$, it leaves $\mathcal{P}$ and becomes dual infeasible with $j \in \mathcal{M}$. Thus, we get for $\bar{f}_r$:
$$\bar{f}_r = f_r + 2\alpha_j^r. \tag{4.27}$$

4. If j is dual infeasible with $j \in \mathcal{M}$ and $j \in \mathcal{J}^f$, it leaves $\mathcal{M}$ and becomes dual infeasible with $j \in \mathcal{P}$. Thus, we get for $\bar{f}_r$:
$$\bar{f}_r = f_r - 2\alpha_j^r. \tag{4.28}$$

5. If j is feasible and $\alpha_j^r < 0$, it becomes dual infeasible and joins $\mathcal{P}$. Thus, we get for $\bar{f}_r$:
$$\bar{f}_r = f_r - \alpha_j^r. \tag{4.29}$$

6. If j is feasible and $\alpha_j^r > 0$, it becomes dual infeasible and joins $\mathcal{M}$. Thus, we get for $\bar{f}_r$:
$$\bar{f}_r = f_r + \alpha_j^r. \tag{4.30}$$

Due to the definition of $\mathcal{F}_0^+$ in the cases 2, 4 and 5, α_j^r has to be negative. Hence, when passing a positive breakpoint, f_r changes as follows:

$$\bar{f}_r = f_r + \begin{cases} |\alpha_j^r| & \text{if } j \in \mathcal{J}^u \cup \mathcal{J}^l, \\ |\alpha_j^r| & \text{if } j \in \mathcal{J}^f \text{ and } d_j = 0, \\ 2\,|\alpha_j^r| & \text{if } j \in \mathcal{J}^f \text{ and } d_j \neq 0,. \end{cases} \tag{4.31}$$

Since we assume $t > 0$, the f_r is negative in the beginning. At every breakpoint it increases by a positive value until it eventually turns positive. This means, that the slope of $Z_0(t)$, which is $-f_r$ as shown in equation (4.12), becomes negative. At this point no further improvement of the sum of dual infeasibilities can be achieved and the current breakpoint determines the entering index.

In order to be able to update the phase I pricing vector f we have to keep track of the changes in $\mathcal{P}$ and $\mathcal{M}$. Let $\mathcal{T}_\mathcal{P}^+$, $\mathcal{T}_\mathcal{P}^-$, $\mathcal{T}_\mathcal{M}^+$, $\mathcal{T}_\mathcal{M}^-$ be the sets of indices, which enter and leave $\mathcal{P}$ and $\mathcal{M}$, respectively. Then, we get for $\bar{f}$:

$$\begin{aligned} \bar{f} &= f + \sum_{j\in\mathcal{T}_\mathcal{P}^-} \alpha_j - \sum_{j\in\mathcal{T}_\mathcal{P}^+} \alpha_j + \sum_{j\in\mathcal{T}_\mathcal{M}^+} \alpha_j - \sum_{j\in\mathcal{T}_\mathcal{M}^-} \alpha_j & (4.32) \\ &= f + B^{-1}\left(\sum_{j\in\mathcal{T}_\mathcal{P}^-} a_j - \sum_{j\in\mathcal{T}_\mathcal{P}^+} a_j + \sum_{j\in\mathcal{T}_\mathcal{M}^+} a_j - \sum_{j\in\mathcal{T}_\mathcal{M}^-} a_j\right) & (4.33) \\ &= f + \Delta f, & (4.34) \end{aligned}$$

where Δf is the solution of the linear system

$$B\Delta f = \tilde{a}, \quad \text{with} \quad \tilde{a} = \sum_{j\in\mathcal{T}_\mathcal{P}^-} a_j - \sum_{j\in\mathcal{T}_\mathcal{P}^+} a_j + \sum_{j\in\mathcal{T}_\mathcal{M}^+} a_j - \sum_{j\in\mathcal{T}_\mathcal{M}^-} a_j. \tag{4.35}$$

The extended ratio test for dual phase I is summarized in algorithm 10. For clarity, we left out the update of Z_0. It can replace step 9 in algorithm 9. It is not difficult to see that the resulting algorithm is exactly the same as applying the dual phase II (with bound flipping ratio test and the modified pricing) to the auxiliary problem (4.7) (see algorithm 8). The dual phase I pricing vector f naturally corresponds to the primal basic solution vector $x_{\mathcal{B}}$ in dual phase II. An advantage of the subproblem approach is though, that no new code is needed to implement it besides the dual phase II code. Hence, every enhancement towards more numerical stability and efficiency in the dual phase II code also improves the dual phase I.

4.3 Artificial bounds

The artificial bounds method is a dual version of the composite simplex method which was originally developed for the primal simplex [74]. It can be seen as a one phase approach where dual infeasible nonbasic variables are penalized in the dual objective function by a high cost coefficient M. From the perspective of the primal problem this means that dual infeasible variables are made dual feasible by setting them to artificial bounds.

Suppose, some starting basis $\mathcal{B}$ is given and $\mathcal{N} = \mathcal{J} \setminus \mathcal{B}$. We can make boxed variables in $\mathcal{N}$ dual feasible by dual feasibility correction. If a variable $j \in \mathcal{N}$ is dual infeasible at lower bound and $u_j = \infty$, then an artificial bound $u_j = M$ is introduced and x_j is set to u_j (d_j is not changed by this operation so j is dual feasible now). If a variable $j \in \mathcal{N}$ is dual infeasible at upper bound and $l_j = -\infty$, an artificial bound l_j is introduced and x_j is set to l_j. $\mathcal{B}$ is dual feasible now, so we start dual phase II with one modification: when a variable with an artificial bound enters the basis, its original bounds are restored. If the dual phase II terminates with an optimal solution and no nonbasic variable is at an artificial bound, we are done. If there is a nonbasic variable j at an artificial bound, there are two possibilities: either M was chosen to small to remove all of the dual infeasibilities in the final basis or the problem is primal unbounded. To find out, we increase M (by multiplying with some constant) and start over. If M exceeds a certain threshold, we declare unboundedness.

Our first implementation of the dual simplex was based on this approach. One disadvantage of this method is that we do not know how large to choose M. The right value for M heavily depends on the problem characteristic. If we choose it to small we risk many rounds of increasing and starting over. If we choose it to big we might encounter numerical problems. For this reason we decided not to include this method in our code.

4.4 Cost modification

The basic idea of the cost modification method is similar to that of the artificial bounds method: making the starting basis dual feasible by modifying the problem formulation and restoring the original problem while executing dual phase II. Here, additionally, we may need to deploy the primal simplex method at the end.

Algorithm 10: Extended ratio test in dual phase I.

```
Set $\tilde{a} = 0$, $\theta^D = 0$.
If $f_r > 0$ set $\tilde{\alpha}^r = -\alpha^r$ and $\tilde{f}_r = -f_r$, if $f_r < 0$ set $\tilde{\alpha}^r = \alpha^r$ and $\tilde{f}_r = f_r$.
Let $\mathcal{Q} = \{j \in \mathcal{N} : (d_j \geq 0 \text{ and } \alpha_j^r > 0) \text{ or } (d_j \leq 0 \text{ and } \alpha_j^r < 0)\}$.
while $\mathcal{Q} \neq \emptyset$ and $\tilde{f}_r \leq 0$ do
    Select $q \in \arg\min_{j \in \mathcal{Q}} \left\{ \frac{d_j}{\tilde{\alpha}_j^r} \right\}$.
    Set $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{q\}$.
    if $j \in \mathcal{J}^f$ and $d_q \neq 0$ then
        $\tilde{f}_r \leftarrow \tilde{f}_r + 2|\alpha_q^r|$.
    else
        $\tilde{f}_r \leftarrow \tilde{f}_r + |\alpha_q^r|$.
    end
    if $\mathcal{Q} \neq \emptyset$ and $\tilde{f}_r \leq 0$ then
        if $d_q > 0$ then
            if $q \in \mathcal{J}^f$ then
                $\tilde{a} \leftarrow \tilde{a} + 2a_q$.
            else
                $\tilde{a} \leftarrow \tilde{a} + a_q$.
            end
        else if $d_q < 0$ then
            if $q \in \mathcal{J}^f$ then
                $\tilde{a} \leftarrow \tilde{a} - 2a_q$.
            else
                $\tilde{a} \leftarrow \tilde{a} - a_q$.
            end
        else
            if $\alpha_q^r > 0$ then
                $\tilde{a} \leftarrow \tilde{a} + a_q$.
            else
                $\tilde{a} \leftarrow \tilde{a} - a_q$.
            end
        end
    end
end
Set $\theta^D \leftarrow \frac{d_q}{\alpha_q^r}$.
Solve $B\Delta f = \tilde{a}$ for $\Delta f$ and set $f \leftarrow f + \Delta f$.
```

Given a starting basis $\mathcal{B}$ with $\mathcal{N} = \mathcal{J} \setminus \mathcal{B}$, boxed variables are made dual feasible by flipping bounds. Then, for each remaining dual infeasible variable $j \in \mathcal{N}$ we shift its cost coefficient by $-d_j$, i.e, we set

$$\tilde{c}_j = c_j - d_j. \tag{4.36}$$

This leads to a new reduced cost value

$$\tilde{d}_j = \tilde{c}_j - a_j^T y = c_j - d_j - a_j^T y = 0, \tag{4.37}$$

which makes j dual feasible. Note, that by each cost shifting at the start of the method an additional degenerate nonbasic position is created. Therefore, in order to reduce the danger of stalling during the further course of the algorithm, we perturb the cost values by a small margin ϵ , which is randomly generated in the interval $[10^6, 10^5]$. However, as our computational results will show, this procedure can only lessen the effects of the inherent additional degeneracy caused by this method, but not completely prevent them. When the dual phase II terminates with a primal feasible basis, the original costs are restored. If the basis goes dual infeasible by this operation, we switch to the primal simplex method (primal phase II).

This method is implemented in the LP code SoPlex [6], which was developed by Wunderling [75]. He reports that it yields good iteration counts and that for many problems it is not necessary to call the primal method at the end. This was not confirmed in our numerical tests though, where it was outperformed by the other methods.

4.5 Pan's method

Pan proposed his method in [54] and further examined it computationally in [55]. The basic idea is to remove at least one dual infeasibility at every iteration without giving any guarantee that no new infeasibilities are created. This risk is minimized only by a proper, geometrically motivated selection of the leaving variable. We give the first description of the method for general linear programs (containing upper bounds) in algorithm 11. Before calling Pan's method we make boxed variables dual feasible by flipping bounds and do not consider them anymore in the further course of the algorithm.

Note, that no ratio test is performed in this method, so we can guarantee neither a monotone reduction of the sum of dual infeasibilities nor of the number of dual infeasibilities. Also, there is no proof of convergence for this method. However, in our computational experiments it converges with a low iteration count on the vast majority of the tested instances. In that sense we can confirm Pan's results in [55]. On very few, mostly numerically difficult, instances it did not converge in an acceptable number of iterations. In that case, we switch to one of the other methods.

Algorithm 11: Dual phase 1: Pan's method.

```
          Input: LP in computational form (2.2), start basis
                 $\mathcal{B} = \{k_1, \dots, k_m\}$
          Output: Dual feasible basis $\mathcal{B}$ with dual feasible basic solution
                  $(y^T, d^T)^T$ or proof that LP is dual infeasible.
(Step 1)  Initialization
          Solve $B^T y = c_{\mathcal{B}}$ for $y$ and compute $d_{\mathcal{N}} = c_{\mathcal{N}} - A_{\mathcal{N}}^T y$.
(Step 2)  Select entering variable
          If $\mathcal{P} \cup \mathcal{M} = \emptyset$ then terminate: basis is dual feasible.
          Else: select entering variable $q \in \arg\max_{j \in \mathcal{P} \cup \mathcal{M}} |d_j|$.
(Step 3)  FTran
          Solve $B\alpha_q = a_q$ for $\alpha_q$.
(Step 4)  Select leaving variable
          Let $\mathcal{H} = \{i \in \{1, \dots, m\} : (l_{\mathcal{B}(i)} > -\infty$ and $\alpha_q^i < 0)$ or
                                      $(u_{\mathcal{B}(i)} < \infty$ and $\alpha_q^i > 0)\}$.
          If $\mathcal{H} = \emptyset$ then terminate: the problem is dual infeasible.
          Else: select leaving variable $p = \mathcal{B}(r)$ with $r \in \arg\max_{i \in \mathcal{H}} |\alpha_q^i|$.
(Step 5)  Basis change and update
          Set $d_p \leftarrow -\theta^D$, $d_q \leftarrow 0$ and $d_j \leftarrow d_j - \theta^D \alpha_j^r$ for all $j \in \mathcal{N} \setminus \{q\}$.
          Set $\mathcal{B} \leftarrow (\mathcal{B} \setminus p) \cup q$ and $\mathcal{N} \leftarrow (\mathcal{N} \setminus q) \cup p$.
          Update $B$.

          Go to step 9.
```

Part II

Computational techniques

Chapter 5

Solving Systems of Linear Equations

5.1 Introduction

The overall performance of simplex type algorithms heavily depends on the efficient solution of systems of linear equations. In fact, four of these systems have to be solved in every iteration of the elaborated dual simplex as presented in algorithm 7 (cf. section 3.4). We can distinguish two types of them. Those, which are defined by the basis matrix B, we call

FTran: $$B\alpha = a, \tag{5.1}$$

and those, which are defined by its transpose B^T, we call

BTran: $$B^T\pi = h. \tag{5.2}$$

Efficient solution methods have to take into account the special characteristics of these systems and of their application within the (dual) simplex method.

- In general, the basis matrix has no special structure, is neither symmetric nor positive definite.
- Especially during the first simplex iterations the basis matrix usually contains many unit vectors stemming from logical variables.
- The columns of the structural part of the constraint matrix normally contain very few nonzero elements. Therefore, the basis matrix stays sparse even if the basis contains mostly structural variables.
- In every simplex iteration the basis matrix changes in only one column.

Due to these properties many classical solution methods for linear systems are not suitable in the context of the simplex method. Conjugate gradient and other *iterative* methods (see e.g. [7]) work well only for symmetric positive definite systems (or those, which can be effectively preconditioned to gain this property). Some *direct* methods based on Givens-rotations or Householder-reflextions (see e.g. [31]) have favorable numerical features but are computationally too expensive.

In the framework of the simplex method other direct methods based on either the *product form of the inverse (PFI)*, the *LU-decomposition* or combinations of the two have been applied successfully. Historically, the PFI was developed shortly after the introduction of the simplex method (cf. [20]) and made it possible to solve LP problems, which were considered large at that time. Later it was superseded by the LU-decomposition, which turned out to allow for a better exploitation of sparsity during the solution process [13].

In the following we will outline the basic ideas of the PFI and LU-decomposition. In the remainder of this chapter we will focus on the latter and give detailed descriptions of the solution algorithms which are implemented in our code.

5.1.1 Product form of the inverse

The foundation for the PFI has been already laid out in section 3.1.5. The new basis inverse $\bar{B}^{-1}$ after a usual simplex basis change can be constructed by

$$\bar{B}^{-1} = EB^{-1}, \tag{5.3}$$

where B^{-1} is the old basis inverse and E is defined according to (3.25). The idea of the PFI is to totally avoid the explicit representation of B^{-1} and replace it by a sequence of eta-matrices, which is used to solve the required linear systems.

For this purpose an initial set of eta-matrices has to be derived prior to the start of the (dual) simplex algorithm, which represents the inverse of the start basis matrix B. If $B = I$ we are done, since $I^{-1} = I$. Otherwise, the structural columns of B can be successively transformed into unit vectors by Gaussian elimination[1] (cf. [32, p. 94ff.]). Every elimination step yields an eta-matrix, such that finally

$$E_s E_{s-1} \cdots E_1 B = I, \tag{5.4}$$

where s with $s \leq m$ is the number of non-unit vectors in B. Hence, the inverse of B can be finally represented as

$$B^{-1} = E_s E_{s-1} \ldots E_1. \tag{5.5}$$

This procedure is called *basis inversion.* Both numerical stability and the amount of non-zero elements, which is created in the eta-matrices (called *fill-in*), heavily depend on the choice of the pivot element in the elimination process. For deeper insights into the details of the PFI inversion procedure we refer to [46, p. 122ff.], [72] and [34].

Given the sequence of $k \leq s + i - 1$ eta-matrices in the i-th iteration after the basis inversion the system (5.1) can be rewritten as

$$\alpha = B^{-1}a = E_k E_{k-1} \cdots E_1 a. \tag{5.6}$$

Equation (5.6) is solved recursively from right to left, the eta-matrices are applied in the order of their generation. Therefore, this operation is called *forward transfor-*

[1] Every elimination step is equivalent to a simplex basis change starting from an identity matrix and working towards the basis matrix B by choosing a leaving column from I and an entering column from B.

mation (FTran). Note, that an eta-matrix with pivot column r can be skipped, if the r-th entry in the current intermediate result is zero.

Likewise, system (5.2) can be written as

$$\pi^T = h^T B^{-1} = h^T E_k E_{k-1} \cdots E_1 \tag{5.7}$$

and solved by applying the eta-matrices from left to right, i.e., in the reversed order of their generation. Therefore, this operation is called *backward transformation (BTran).* Here, the exploitation of sparsity as in FTran is not possible. However, in every intermediate result the number of non-zeros can at most increase by one, since only the r-th entry of the result vector changes if r is the position of the eta-column.

With the number of eta-matrices numerical inaccuracies, storage requirements and computational effort in FTran and BTran increase in every simplex iteration. Therefore, a *reinversion* of the basis is triggered either when a certain number of iterations since the last inversion has passed or numerical or storage problems occur.

5.1.2 LU decomposition

Nowadays, virtually all competitive implementations of the simplex method use an *LU-decomposition* of the basis matrix B, which goes back to the *elimination form of the inverse* by Markowitz [45]. The idea is to represent the basis matrix B as a product of an upper triangular matrix U and a lower triangular matrix L[2]. The generation process of this representation is called *LU-factorization.* The solution of the required linear systems can then be efficiently conducted by successive forward and backward substitutions.

In the LU-factorization the basis B is transformed into an upper triangular matrix U by Gaussian elimination[3]. In the i-th iteration of the elimination process the element in the upper left corner of the matrix, which has not been transformed up to this point (called *active submatrix*), is chosen as pivot element. Then a lower triangular eta-matrix L^i is generated, such that

$$B_{i+1} = L^i \cdots L^1 B. \tag{5.8}$$

After m steps $B_{m+1} = U$ is upper triangular and

$$L^m \cdots L^1 B = U. \tag{5.9}$$

This form constitutes the bases for the solution of the required linear systems and the LU-update. Defining $L^{-1} = L^m \cdots L^1$ and $L = (L^1)^{-1} \cdots (L^m)^{-1}$ we can represent B as the product of the triangular matrices L and U,

$$B = LU, \tag{5.10}$$

[2]Hence, the LU-decomposition is a representation of the basis matrix, while the PFI is a representation of the basis inverse.

[3]This is similar to the elimination process in the PFI-inversion, where B is transformed into the identity matrix I.

which we will refer to as the *LU-factorization* of B. Given (5.10) we can rewrite system (5.1) as

$$LU\alpha = a. \tag{5.11}$$

By substituting $U\alpha$ by $\bar{\alpha}$ we can solve (5.11) in two independent steps. In a first step we solve

FTranL: $$L\bar{\alpha} = a \quad \Leftrightarrow \quad \bar{\alpha} = L^{-1}a \tag{5.12}$$

by applying the eta-matrices produced during LU-factorization to a, then we solve

FTranU: $$U\alpha = \bar{\alpha} \tag{5.13}$$

by backward substitution. The same can be done for system (5.2) with

$$U^T L^T \pi = h \tag{5.14}$$

by substituting $L^T\pi$ by $\bar{\pi}$ and solving

BTranU: $$U^T\bar{\pi} = h \tag{5.15}$$

for $\bar{\pi}$ followed by computing

BTranL: $$L^T\pi = \bar{\pi} \quad \Leftrightarrow \quad \bar{\alpha} = (L^T)^{-1}a \tag{5.16}$$

for π.

5.2 LU factorization

Numerical stability and the amount of fill-in during the Gaussian elimination procedure depend heavily on the choice of the pivot elements during the factorization procedure. To be free to choose a pivot element, which is advantageous with respect to these two goals, row and column permutations are applied to B in every iteration. The result can be represented in the form

$$LU = PBQ, \tag{5.17}$$

where L and U are lower and upper triangular matrices, respectively, P is a row- and Q is a column permutation matrix. By defining permuted versions $\tilde{L} = P^{-1}LP$ and $\tilde{U} = P^{-1}UQ^{-1}$ of L and U equation (5.17) transforms to

$$P\tilde{L}P^{-1}P\tilde{U}Q = PBQ \tag{5.18}$$

$$\Leftrightarrow \quad \tilde{L}\tilde{U} = B \tag{5.19}$$

$$\Leftrightarrow \quad \tilde{L}^{-1}B = \tilde{U} \tag{5.20}$$

$$\Leftrightarrow \quad \tilde{L}^{-1}B = P^{-1}UQ^{-1}. \tag{5.21}$$

Algorithm 12: LU-factorization.

```
          Input: Let $\tilde{U} = B$, $P = Q = 0$, $\mathcal{P} = \{1, \dots, m\}$, $\mathcal{Q} = \{1, \dots, m\}$
                 and $\tilde{L}^i = I$ for $i = 1, \dots, m$.
          for $k = 1$ to $m$ do
(Step 1)      (Pivot-Selection)
              Choose pivot element $\tilde{u}^p_q \neq 0$ with $p \in \mathcal{P}$ and $q \in \mathcal{Q}$.
(Step 2)      (Permutation)
              Set $P_{p,k} \leftarrow 1$ and $Q_{k,q} \leftarrow 1$.
(Step 3)      (Reduce active submatrix)
              Set $\mathcal{P} \leftarrow \mathcal{P} \setminus \{p\}$.
              Set $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{q\}$.
(Step 4)      (Elimination)
              forall $i \in \{i' \in \mathcal{P} : \tilde{u}^{i'}_q \neq 0\}$ do
                  Set $\tilde{L}^k_{i,p} \leftarrow -\frac{\tilde{u}^i_q}{\tilde{u}^p_q}$.
                  Set $\tilde{u}^i_q = 0$.
                  forall $j \in \{j' \in \mathcal{Q} : \tilde{u}^p_{j'} \neq 0\}$ do
                      Set $\tilde{u}^i_j = \tilde{u}^i_j + \tilde{L}^k_{i,p} * \tilde{u}^p_j$.
                  end
              end
          end
```

Equations (5.19) to (5.21) can now be used to split each of the linear systems (5.1) and (5.2) into efficiently solvable parts (cf. section 5.4.2).

Algorithm 12 shows how the elimination process can be organized on a conceptual level. It obtains $\tilde{L}^{-1}$ as a product of column-eta-matrices $\tilde{L}^i$. Note, that if the pivot column of the active submatrix is a column singleton (the set $\{i' \in \mathcal{P} : \tilde{u}^{i'}_q \neq 0\} = \emptyset$), then no eta-matrix is generated in this iteration. Hence, algorithm 12 ends up with an LU-factorization of the form

$$\tilde{L}^s \cdots \tilde{L}^1 B = \tilde{U} = P^{-1} U Q^{-1}, \tag{5.22}$$

where s is the number of non-trivial eta-matrices and $\tilde{L}^{-1} = \tilde{L}^s \cdots \tilde{L}^1$. In equation (5.22) we can eliminate one permutation by multiplying with the term QP from the right, which gives us:

$$\tilde{L}^s \cdots \tilde{L}^1 BQP = \tilde{U}QP = P^{-1}UP. \tag{5.23}$$

Note, that BQP and $\tilde{U}QP$ emerge from B and $\tilde{U}$, respectively, by column permutation. Consequently, in our implementation we perform a physical columnwise reordering of $\tilde{U}$ after every refactorization. Since B is not explicitly stored it suffices to adapt the mapping of the column indices of the basis matrix to the indices of the basic variables (*basis heading*) accordingly. Thus, with

$$B \leftarrow BQP \quad \text{and} \tag{5.24}$$

$$\tilde{U} \leftarrow \tilde{U}QP \tag{5.25}$$

we get as the final output of the refactorization procedure the following representation of the LU-factorization:

$$\tilde{L}^s \cdots \tilde{L}^1 B = \tilde{U} = P^{-1}UP. \tag{5.26}$$

We proceed with solving the required linear systems based on that new ordering, since for the simplex method the order of the basic variables does not matter. For further important details concerning pivot selection strategies and implementation we confer to [68].

5.3 LU update

Both methods, PFI and LU-decomposition, were first used in combination (LU-decomposition in reinversion and PFI during the simplex iterations). Later, an updating method for the LU-representation was presented by Bartels and Golub [8], which was the basis for several variants developed by Forrest and Tomlin [24], Saunders [60, 61], Reid [58] and Suhl and Suhl [63]. After a short introduction we will describe and analyze the two related methods of Forrest/Tomlin and Suhl/Suhl. The latter is the method, which is actually implemented in our code.

Recalling equation (5.19) the basis matrix can be represented as

$$B = \tilde{L}\tilde{U}, \tag{5.27}$$

where $\tilde{L} = P^{-1}LP$ and $\tilde{U} = P^{-1}UP$ (cf. equation 5.26) are permuted lower and upper triangular matrices, respectively. In each iteration of the (dual) simplex method the entering column a_q replaces the leaving column a_p at position r in the basis matrix (cf. equation (3.20)). Hence, for the new basis $\bar{B}$, we get

$$\begin{aligned} \bar{B} &= B - Be_r e_r^T + a_q e_r^T \\ &= B + (a_q - Be_r)e_r^T \\ &= \tilde{L}\tilde{U} + (a_q - \tilde{L}\tilde{U}e_r)e_r^T. \end{aligned} \tag{5.28}$$

Multiplying with $\tilde{L}^{-1}$ leads to

$$\tilde{L}^{-1}\bar{B} = \tilde{U} + (\tilde{L}^{-1}a_q - \tilde{U}e_r)e_r^T = \tilde{V}. \tag{5.29}$$

Equation (5.29) shows, that in $\tilde{U}$ the r-th column is replaced by the vector $\bar{\alpha} = \tilde{L}^{-1}a_q$. We denote the result by $\tilde{V}$. The vector $\bar{\alpha}$ is called *permuted spike* and comes as the intermediate FTranL result of the FTran operation, which computes the transformed pivot column α in the dual simplex algorithm (cf. step 7 in algorithm 7).

To see, how the upper triangular matrix U changes, we further manipulate equation (5.29) in order to remove the permutation from the right hand side. Here, we suppose, that the r-th row in I is the t-th row in P, i.e. $Pe_r = e_t$ and $e_r^T P^{-1} = e_t^T$. Multiplying (5.29) with P from the left and with P^{-1} from the right yields

$$\begin{aligned} P\tilde{L}^{-1}\bar{B}P^{-1} &= U + (P\tilde{L}^{-1}a_q - UPe_r)e_r^T P^{-1} \\ &= U + (P\tilde{L}^{-1}a_q - Ue_t)e_t^T = V. \end{aligned} \tag{5.30}$$

Consequently, the resulting matrix V differs from U only in the t-th column, which contains the *spike* $\bar{\alpha}' = P\tilde{L}^{-1}a_q$. The term spike becomes clear by looking at the shape of the matrix V:

$$V = \begin{pmatrix} u_1^1 & \cdots & \bar{\alpha}_1' & \cdots & u_m^1 \\ & \ddots & \vdots & & \vdots \\ & & \bar{\alpha}_t' & \cdots & u_m^t \\ & & \vdots & \ddots & \vdots \\ & & \bar{\alpha}_m' & & u_m^m \end{pmatrix}. \tag{5.31}$$

Although in implementation we have to deal with $\tilde{V}$, LU-update methods are usually described in terms of V for easier conceivability. All of them restore upper triangularity by factorizing V, after applying further permutations to it to minimize the generation of additional fill-in and to improve numerical stability. Only Forrest/Tomlin like methods preserve the symmetric permutation of U.

5.3.1 Forrest/Tomlin update

In thw update method presented by J. J. H. Forrest and J. A. Tomlin in [24] the spike (the t-th column of V) is permuted to the last column position as in the earlier method of Bartels and Golub [8]. But additionally, also the t-th row is permuted to the last row position. This is done by a permutation matrix $R = I_S$ with $S = (1, \dots, t-1, t+1, \dots, m, t)$ and its inverse R^T:

$$R = \begin{array}{c} \begin{array}{cccccc} & & & t & & \\ & & & \downarrow & & \end{array} \\ \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & & 1 & \\ & & & & & \ddots & \\ & & & & & & 1 \\ & & & 1 & & & \end{pmatrix} \end{array}, \quad R^T = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & & & 1 \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{pmatrix} \begin{array}{c} \\ \\ \\ \leftarrow t \\ \\ \\ \\ \end{array} \tag{5.32}$$

The resulting matrix RVR^T violates upper triangularity only in its last row (cf. 5.1). This row contains the vector $[0, \dots, 0, u_{t+1}^t, \dots, u_m^t, \bar{\alpha}_t']$, which has to be eliminated (except the diagonal element $\bar{\alpha}_t'$) to obtain the upper triangular matrix $\bar{U}$. In the update method of Forrest and Tomlin the pivot elements are chosen strictly down the diagonal during the elimination process, so no further permutations are necessary. Therefore, no fill-in can occur in $\bar{U}$, however, numerical stability is completely disregarded.

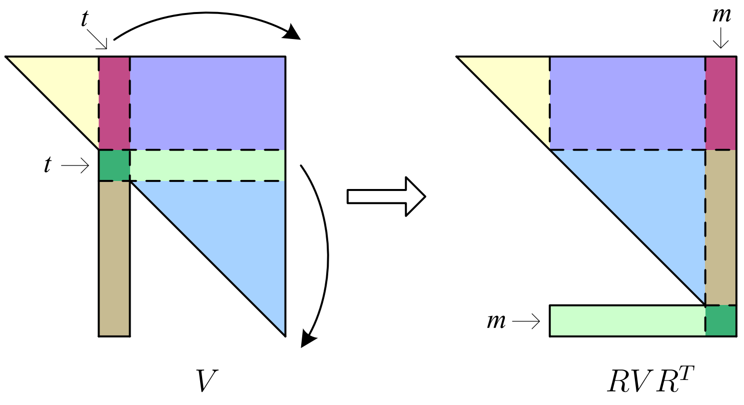


Figure 5.1: Forrest/Tomlin update

During the elimination of the last row eta matrices $\hat{L}_j$ $(t+1 \leq j \leq m)$ arise, each with exactly one off-diagonal element μ_j in the last row and column $j-1$. The product of the eta-matrices $\hat{L}_{t+1}, \ldots, \hat{L}_m$ can be summarized in a single eta-matrix $\hat{L}$ with one row eta-vector:

$$\hat{L} = \hat{L}_m \cdots \hat{L}_{t+1} = \begin{array}{c} \begin{array}{ccccc} & \overset{t}{\downarrow} & & \overset{m-1}{\downarrow} & \end{array} \\ \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & \ddots & & \\ & & & & 1 & \\ & & \mu_{t+1} & \cdots & \mu_m & 1 \end{pmatrix} \end{array} . \tag{5.33}$$

It is not difficult to verify that the multipliers μ_j $(t+1 \leq j \leq m)$ can be determined by solving the following linear system:

$$\begin{pmatrix} u_{t+1}^{t+1} & & \\ \vdots & \ddots & \\ u_m^{t+1} & \cdots & u_m^m \end{pmatrix} \begin{pmatrix} \mu_{t+1} \\ \vdots \\ \mu_m \end{pmatrix} = \begin{pmatrix} -u_{t+1}^t \\ \vdots \\ -u_m^t \end{pmatrix} . \tag{5.34}$$

The diagonal element $\bar{\alpha}'_t$ of V is situated in the lower right corner of $\bar{U}$ and must be updated to

$$\bar{u}_m^m = \sum_{i=t+1}^{m} \mu_i \bar{\alpha}'_i + \bar{\alpha}'_t. \tag{5.35}$$

The result of the update procedure is the factorization of RVR^T:

$$\hat{L}RVR^T = \bar{U}. \tag{5.36}$$

By replacing V by its definition (eq. (5.30)) and defining

$$\bar{P} = RP \qquad \text{and} \tag{5.37}$$
$$\bar{L} = \bar{P}^{-1}\hat{L}\bar{P} \tag{5.38}$$

we can derive a new factorization of $\bar{B}$:

$$\hat{L}RP\tilde{L}^{-1}\bar{B}P^{-1}R^{-1} = \bar{U} \tag{5.39}$$
$$\Leftrightarrow \qquad \hat{L}\bar{P}\tilde{L}^{-1}\bar{B}\bar{P}^{-1} = \bar{U} \tag{5.40}$$
$$\Leftrightarrow \qquad \bar{P}\bar{L}\tilde{L}^{-1}\bar{B}\bar{P}^{-1} = \bar{U} \tag{5.41}$$
$$\Leftrightarrow \qquad \bar{L}\tilde{L}^{-1}\bar{B} = \bar{P}^{-1}\bar{U}\bar{P}. \tag{5.42}$$

Note, that in each iteration of the (dual) simplex method a rowwise eta-matrix of the type $\bar{L}$ is generated in the LU-update procedure. Obviously, these eta-matrices have to be dealt with separately from the columnwise etas from LU-factorization in FTran and BTran. In section 5.4.2 we will come back to this point in greater detail.

Stability test

As there is no flexibility in the choice of the pivot elements in the Forrest/Tomlin update, numerical stability can not be taken in account during the elimination process. However, it is possible to test numerical accuracy *a posteriori*. The following equation is true after the update (for proof see [24, p. 272f.]):

$$\alpha_q^r = \frac{\bar{u}_m^m}{u_t^t}. \tag{5.43}$$

If the deviation between the current simplex pivot element α_q^r and the right-hand-side of equation (5.43) is greater than some predefined tolerance the operation is judged numerically instable. In this case the current simplex iteration is aborted and a refactorization is triggered to obtain a more stable LU-representation of the basis.

5.3.2 Suhl/Suhl update

The update method of L. M. Suhl and U. H. Suhl [63] is a variant of the Forrest/Tomlin update, which exploits an observation of Saunders (cf. [60, 61]): the t-th column and the t-th row are not permuted to the last position, respectively, but to position l, which is the index of the last nonzero element in the spike $\bar{\alpha}'$. This is done by a permutation matrix $R = I_S$ with $S = (1, \dots, t-1, t+1, \dots, l, t, l+1, \dots, m)$ and its inverse R^T.

Here, the matrix RVR^T differs only in row l from an upper triangular form, which

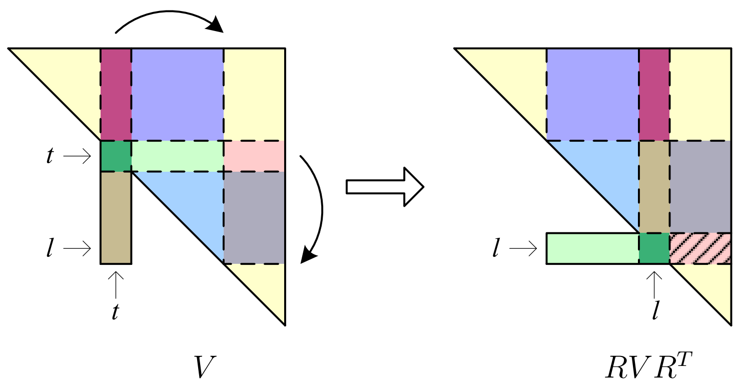


Figure 5.2: Suhl/Suhl update

contains the vector $[0, \dots, 0, u^t_{t+1}, \dots, u^t_l, \bar{\alpha}'_t, u^t_{l+1}, \dots, u^t_m]$ (cf. figure 5.2). The elimination of the elements $u^t_{t+1}, \dots, u^t_l$ is carried out along the lines of the Forrest/Tomlin procedure. Since in this method row t is not permuted to the last position, fill-in can occur in row l of $\bar{U}$ (hatched region in figure 5.2). In general, we can expect less fill-in than with Forrest-Tomlin, since fewer eliminations have to be carried out (the elements $\bar{\alpha}'_t, u^t_{l+1}, \dots, u^t_m$ need not to be eliminated).

During the elimination process eta-matrices $\hat{L}_j$ $(t+1 \le j \le l)$ arise, which can be summarized in a single eta-matrix $\hat{L}$ with a row-eta-vector in row l:

$$\hat{L} = \hat{L}_l \cdots \hat{L}_{t+1} = \begin{array}{c} \\ \left(\begin{array}{cccccccc} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & \ddots & & & & \\ & & & & 1 & & & \\ & & \mu_{t+1} & \cdots & \mu_l & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{array}\right) \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \leftarrow l \\ \\ \\ \end{array} \qquad (5.44)$$

with column markers $\overset{t}{\downarrow}$ over the column of μ_{t+1} and $\overset{l-1}{\downarrow}$ over the column of μ_l.

Similar to the Forrest/Tomlin update the multipiers μ_j $(t+1 \le j \le l)$ can be determined by executing a (reduced) BTranU operation:

$$\begin{pmatrix} u^{t+1}_{t+1} & & \\ \vdots & \ddots & \\ u^{t+1}_l & \cdots & u^l_l \end{pmatrix} \begin{pmatrix} \mu_{t+1} \\ \vdots \\ \mu_l \end{pmatrix} = \begin{pmatrix} -u^t_{t+1} \\ \vdots \\ -u^t_l \end{pmatrix}. \qquad (5.45)$$

For row l of the resulting upper triangular matrix $\bar{U}$ we get:

$$\bar{u}^l_l = \sum_{i=t+1}^{l} \mu_i \bar{\alpha}'_i + \bar{\alpha}'_t \quad \text{and} \quad \bar{u}^l_j = \sum_{i=t+1}^{l} \mu_i u^i_j + u^t_j \quad \text{for} \quad j = l+1, \dots, m. \qquad (5.46)$$

Since we have $\bar{\alpha}'_i = 0$ for $i = l+1, \ldots, m$, equation (5.46) yields the same value for $\bar{u}^l_l$ as equation (5.35) for $\bar{u}^m_m$ in Forrest/Tomlin. Therefore, the stability test described in the previous section can be applied as well in the context of the Suhl/Suhl update. The same is true for the derivation of the new factorization in equations (5.36) to (5.42).

Algorithm 13 shows the LU-update method of Suhl/Suhl in terms of $\tilde{U}$. The sets $\tilde{\mathcal{U}}_j$ and $\tilde{\mathcal{U}}^i$ contain the indices of the nonzero elements of the j-th column and the i-th row of $\tilde{U}$, respectively. The permutations P and P^{-1} are maintained as arrays. In algorithm 13 we use a functional notation, where $P(j) = i$ is equivalent to $P^i_j = 1$ and $P^{-1}(j) = i$ is equivalent to $(P^{-1})^i_j = 1$. Intuitively, P maps indices of $\tilde{U}$ to indices of U while P^{-1} does the opposite direction.

5.4 Exploiting (hyper-)sparsity in FTran, BTran and LU-update

As mentioned before, the constrained matrix A of nearly all practical LP problems is *sparse*, i.e., only a small fraction of its elements is different from zero. In fact, it has been observed, that usually the matrix A contains only 5–10 nonzero elements (short: *nonzeros*) per column independent of the size of the problem. Consequently the same is true for the basis matrix B, which typically leads to sparse factors $\tilde{L}$ and $\tilde{U}$ in LU-factorization. Concerning FTran, BTran and their suboperations, it turns out that it is not only important to exploit the sparsity of $\tilde{L}^{-1}$ and $\tilde{U}$ but also the sparsity of the result vectors $\bar{\alpha}, \alpha, h$ and π. In section 5.4.1 we will describe solution methods for triangular systems exhibiting different degrees of sparsity in their result vectors. In section 5.4.2 we show how these methods can be combined to construct powerful FTran and BTran operations.

5.4.1 Algorithms for sparse and hypersparse triangular systems

To illustrate the solution techniques we will take a closer look at the upper triangular system $Ux = b$[4]:

$$\begin{pmatrix} u^1_1 & \cdots & u^1_i & \cdots & u^1_m \\ & \ddots & \vdots & & \vdots \\ & & u^i_i & \cdots & u^i_m \\ & & & \ddots & \vdots \\ & & & & u^m_m \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_m \end{pmatrix}. \tag{5.47}$$

The vector x can be determined by *backward substitution*:

$$x_i = \frac{1}{u^i_i}\left(b_i - \sum_{j=i+1}^{m} u^i_j x_j\right) \quad \text{for } i = m, m-1, \ldots, 1. \tag{5.48}$$

[4]Here, x and b are just some arbitrary vectors in $\mathbb{R}^m$.

Algorithm 13: LU-Update Suhl/Suhl (in terms of $\tilde{U}$)

```
// remove column $r$ from $\tilde{U}$
forall $i \in \tilde{\mathcal{U}}_r$ do
    $\tilde{u}^i_r \leftarrow 0$
    remove $i$ from $\tilde{\mathcal{U}}_r$ and $r$ from $\tilde{\mathcal{U}}^i$
end
// insert vector $\bar{\alpha}$ into column $r$ of $\tilde{U}$
forall $i \in \mathcal{I}_{\bar{\alpha}}$ do
    $\tilde{u}^i_r \leftarrow \bar{\alpha}_i$
    add $i$ to $\tilde{\mathcal{U}}_r$ and $r$ to $\tilde{\mathcal{U}}^i$
end
// eliminate elements $u^t_{t+1}, \ldots, u^t_l$, calculate multipliers $\mu_{t+1}, \ldots, \mu_l$
for $k = t + 1$ to $l$ do
    $i \leftarrow P^{-1}(k)$
    if $\tilde{u}^r_i \neq 0$ then
        $\bar{L}_{r,i} \leftarrow -\frac{\tilde{u}^r_i}{\tilde{u}^i_i}$
        $\tilde{u}^r_i \leftarrow 0$
        remove $i$ from $\tilde{\mathcal{U}}^r$ and $r$ from $\tilde{\mathcal{U}}_i$
        forall $j \in \tilde{\mathcal{U}}^i \setminus \{i\}$ do
            $tmp \leftarrow \tilde{u}^r_j$
            $\tilde{u}^r_j \leftarrow \tilde{u}^r_j + \bar{L}_{r,i} * \tilde{u}^i_j$
            if $tmp = 0 \wedge u^r_j \neq 0$ then add $r$ to $\tilde{\mathcal{U}}_j$ and $j$ to $\tilde{\mathcal{U}}^r$
            else if $tmp \neq 0 \wedge \tilde{u}^r_j = 0$ then remove $r$ from $\tilde{\mathcal{U}}_j$ and $j$ from $\tilde{\mathcal{U}}^r$
        end
    end
end
// update permutation
for $k = t + 1$ to $l$ do
    $i \leftarrow P^{-1}(k)$
    $P(i) = k - 1$
    $P^{-1}(k - 1) = i$
end
```

Algorithm 14: $Ux = b$ – dense method (for dense b)

```
Input: Let $x = b$.
1 for $i = m$ to 1 step $-1$ do
2     forall $j \in \mathcal{U}^i \setminus \{i\}$ do
3         $x_i \leftarrow x_i - u^i_j * x_j$
4     end
5     $x_i \leftarrow x_i / u^i_i$
6 end
```

Algorithm 15: $Ux = b$ – sparse method (for sparse b)

```
Input: Let $x = b$.
1 for $j = m$ to 1 step $-1$ do
2     if $x_j \neq 0$ then
3         $x_j \leftarrow x_j / u^j_j$
4         forall $i \in \mathcal{U}_j$ do
5             $x_i \leftarrow x_i - u^i_j * x_j$
6         end
7     end
8 end
```

Equation 5.48 directly translates into algorithm 14, which we refer to as *dense method*. $\mathcal{U}^i$ contains the indices of the nonzero elements of the i-th row of U.

Algorithm 14 only exploits the sparsity of U. However, we can utilize the fact that in the summation term of equation 5.48 an entry x_j is multiplied exclusively with entries of the j-th column of U. If we reorganize the computation in a column-wise fashion, we can skip columns of U where the corresponding entry in x is zero. This is done in algorithm 15, which we call *sparse method*. $\mathcal{U}_j$ contains the indices of the nonzero elements of the j-th column of U.

The sparse method is the most efficient solution technique for a sparse right-hand side b and a moderately sparse (5% – 50% nonzeros) result vector x. This is the typical situation for most practical LP problems. Only in cases where the right-hand side vector is already quite dense (> 50% nonzeros) or the required data structures are not available the dense method should be applied.

Exploitation of hyper-sparsity

To our knowledge it was first reported by Bixby [10] that for some LP problems further remarkable improvements in performance can be achieved by exploiting sparsity even more aggressively. This can be done by a technique published by J. R. Gilbert and T. Peierls [29] as early as 1988 in the Linear Algebra community. In a first *symbolical phase* a list and a feasible ordering of the nonzero positions in the result vector is determined. In the subsequent *numerical phase* the sparse solution method (cf. algorithm 15) is performed based on this list.

In the symbolical phase the structure of U is viewed as a direct graph $\mathcal{G}$ with m nodes and edges (j, i) from a node j to a node i if and only if $u^i_j \neq 0$ and $i \neq j$ (cf. figure 5.3). Let $\mathcal{I} = \{i : b_i \neq 0\}$ and $\mathcal{O} = \{i : x_i \neq 0\}$ be the sets of nonzero indices in the input vector b and the output vector x, respectively. Then according to Gilbert and Peierls every node corresponding to an element in $\mathcal{O}$ can be reached from a node corresponding to an element in $\mathcal{I}$ by a path in the graph $\mathcal{G}$. Hence, the elements in $\mathcal{O}$ can be determined by depth first search (DFS) starting from the nodes in $\mathcal{I}$.

$$\begin{pmatrix} \times & \times & & \times & & \\ & \times & \times & & \times & \\ & & \times & & & \times \\ & & & \times & & \\ & & & & \times & \times \\ & & & & & \times \end{pmatrix}$$

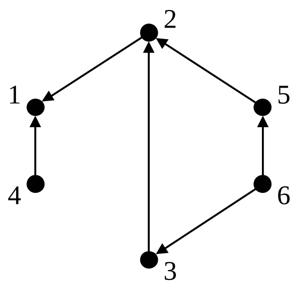


Figure 5.3: An upper triangular matrix an the corresponding nonzero graph $\mathcal{G}$

In the solution procedures presented above the entries x_i of the result vector x are computed in descending order $i = m, m-1, \dots, 1$. However, we cannot guarantee that the DFS on $\mathcal{G}$ finds the elements of $\mathcal{O}$ in exactly this order. Fortunately, the elements of $\mathcal{O}$ can be brought in a valid order efficiently. To compute an entry x_i we need all those entries x_j with $x_j \neq 0$, $u^i_j \neq 0$ and $j > i$. In the i-th row of U we have in column $j > i$ just those nonzero elements for which (j, i) is an edge in $\mathcal{G}$. Thus, we can compute entry x_i as soon as all x_j corresponding to an edge (j, i) have been computed. Hence, the entries of x can be computed in a *topological order* of the elements of $\mathcal{O}$, which can easily be determined during the DFS. Let $\mathcal{O}'$ be an ordered list of the elements in $\mathcal{O}$, which is empty at the beginning of the DFS. Every time the DFS backtracks from node i to a preceding node j, node i is added to the beginning of the list. After the DFS $\mathcal{O}'$ contains a topological ordering of the nonzero positions of x.

In the numerical phase the result vector x can now be determined by traversing $\mathcal{O}'$:

$$x_i = \begin{cases} \dfrac{1}{u^i_i}\left(b_i - \sum\limits_{j \in (\mathcal{U}^i \setminus \{i\}) \cap \mathcal{O}} u^i_j x_j\right) & \text{for } i \in \mathcal{O}', \\ 0 & \text{for } i \notin \mathcal{O}'. \end{cases} \tag{5.49}$$

For the entries x_i with $i \notin \mathcal{O}'$ we can guarantee $x_i = 0$ and $i \notin \mathcal{O}$. The remaining entries x_i with $i \in \mathcal{O}'$ are potential nonzero elements. However, the symbolical phase does not consider cancellation, so some of the elements in $\mathcal{O}$ might as well be zero. Although cancellation occurs very rarely, it should be considered in implementation when a new index vector of the nonzero elements of the result vector x is constructed. Algorithm 16 summarizes the hyper-sparse solution method.

Algorithm 16: $Ux = b$ – hyper-sparse method (for very sparse x)

```
Input: Let $x = b$.
1 Compute $\mathcal{O}'$ in topological order by depth-first-search on column-wise
  representation of $U$.
2 foreach $j \in \mathcal{O}'$ (ordered!) do
3     if $x_j \neq 0$ then
4         $x_j \leftarrow x_j / u_j^j$
5         forall $i \in \mathcal{U}_j$ do
6             $x_i \leftarrow x_i - u_j^i * x_j$
7         end
8     end
9 end
```

5.4.2 FTran and BTran with Suhl/Suhl update

In the following we will briefly discuss how the methods of the previous section can be applied to FTran and BTran operations given a Forrest/Tomlin or Suhl/Suhl update, respectively. In an arbitrary simplex iteration we are given an LU-factorization of the form:

$$\tilde{L}^k \cdots \tilde{L}^{s+1} \tilde{L}^s \cdots \tilde{L}^1 B = P^{-1} U P = \tilde{U}, \tag{5.50}$$

where U is an upper triangular $m \times m$ matrix, P is an $m \times m$ permutation matrix, $\tilde{L}^j$ $(1 \leq j \leq s)$ are permuted lower triangular column-eta-matrices stemming from LU-factorization and $\tilde{L}^j$ $(s+1 \leq j \leq k)$ are permuted row-eta-matrices stemming from LU-update. Furthermore, we know that the product $\tilde{L}^s \cdots \tilde{L}^1$ is a permuted lower triangular $m \times m$ matrix. Given (5.50) we decompose the FTran operation 5.1 into three parts:

FTranL-F: Compute $\bar{\bar{\alpha}} = \tilde{L}^s \cdots \tilde{L}^1 a$. (5.51a)

FTranL-U: Compute $\bar{\alpha} = \tilde{L}^k \cdots \tilde{L}^{s+1} \bar{\bar{\alpha}}$. (5.51b)

FTranU: Solve $P^{-1} U P \alpha = \bar{\alpha}$ for α. (5.51c)

In FTranL-F the column-eta-matrices from LU-factorization are successively multiplied with the input vector a in the order of their generation. An eta-matrix $\tilde{L}^j$ with its eta-column in position j' can be skipped, if the j'-th entry of the intermediate result $\tilde{L}^{j-1} \cdots \tilde{L}^1 a$ is zero. Also a hyper-sparse method can be applied, since the column-eta-vectors can be regarded as a column-wise representation of a (permuted) lower triangular matrix. Both cannot be done in FTranL-U, since the eta-matrices from LU-update are in row-wise format. In FTranU we can apply the algorithms from the previous section if we handle the permutation properly. For the sparse- and hypersparse method, a column-wise representation of U is required, which has to be derived from the row-wise representation after each refactorization. If the FTran operation is called to compute the transformed incoming column, $\bar{\alpha}$ is saved after FTranL-U to be used as spike in LU-update.

Similarly, the BTran operation 5.2 is decomposed into three parts

BTranU: Solve $P^{-1}U^T P\bar{\pi} = h$ for $\bar{\pi}$. (5.52a)

BTranL-U: Compute $\bar{\bar{\pi}} = (\tilde{L}^{s+1})^T \cdots (\tilde{L}^k)^T \bar{\pi}$. (5.52b)

BTranL-F: Compute $\pi = (\tilde{L}^1)^T \cdots (\tilde{L}^s)^T \bar{\bar{\pi}}$. (5.52c)

As in FTranU the slightly modified version of the sparse- and hypersparse solution algorithms 15 and 16 can as well be applied in BTranU. Here, forward substitution has to be performed on a row-wise representation of U (which is equivalent to a columnwise representation of U^T). As before the permutation can be handled efficiently. In BTranL-U the same techniques can be applied (which was not possible in FTranL-U), since the row-eta-vectors represent the non-trivial columns of the transposed eta-matrices $(\tilde{L}^{s+1})^T, \ldots, (\tilde{L}^k)^T$. However, in BTranL-F a row-wise representation of the product $\tilde{L}^s \cdots \tilde{L}^1$ would be required to exploit sparsity and hypersparsity of π. This is not done in our implementation.

Chapter 6

Numerical Stability and Degeneracy

6.1 Introduction

It might seem as a minimum requirement for an implemented numerical algorithm that it terminates after a finite period of time with a correct answer for any problem instance belonging to the class of problems it was designed for. Unfortunately, even if the algorithm can be proved to be correct with respect to these two goals on a theoretical basis, the same cannot be guaranteed for its implementation in a computational environment, which can only handle floating point numbers of finite precision. For the simplex algorithm the situation is even worse: for the commonly used pivot selection rules we can not even theoretically guarantee termination due to the possibility of getting into an infinite cycle of degenerate bases. While this type of *pure cycling* occurs rarely for practical LP problems degeneracy often retards convergence by preventing improvement of the objective function for many consecutive iterations (called *stalling*). Together with numerical inaccuracies the thread of *numerical cycling* arises, where small deteriorations in the objective function may lead into an infinite loop.

For these reasons we have to be more modest and lower our requirements to the following goal: our code should give *sufficiently correct* results for the widest possible range of LP problems or otherwise, fail in a controlled manner. In this chapter we will discuss methods, which have been proposed in the literature and used in other publicly available codes, and present the variants which we incorporated into the MOPS dual simplex code to improve numerical stability, handle inaccuracies and reduce the number of degenerate iterations. But before we come to methods we will characterize the notions of numerical stability and degeneracy in more detail.

6.1.1 Numerical stability

As we mentioned before the representation of floating point numbers as well as all arithmetic operations are of limited precision on commonly used computer technology. On modern PC-environments 64-bit[1] arithmetic leads to a relative numerical error in the order of 10^{-16}. Analyzing the numerical stability of an algorithms means to understand how and under which circumstances these inevitable inaccuracies can grow and accumulate and in the worst case, lead to unusable results. In the follow-

[1]Internally modern processor even work with 80-bit arithmetic. The results are then converted to 64-bit.

ing we will discuss two results from the field of numerical error analysis, which are important in the context of the (dual) simplex algorithm.

The first one is an elementary result from *backward error analysis* of linear systems. Our presentation is similar to [31, p. 50ff.]. Let B be a nonsingular $m \times m$-matrix and consider the linear system

$$Bx = b \tag{6.1}$$

with right-hand-side b and exact solution x. Let ΔB and Δb be perturbations of the problem data B and b such that $x + \Delta x^b$ and $x + \Delta x^B$ are solutions of the perturbed systems

$$B(x + \Delta x^b) = b + \Delta b \tag{6.2}$$

and

$$(B + \Delta B)(x + \Delta x^B) = b, \tag{6.3}$$

respectively. With these notations it is easy to see that the following upper bounds hold for the relative error of the solution vector[2]:

$$\frac{\|\Delta x^b\|}{\|x\|} \leq \|B^{-1}\|\|B\|\frac{\|\Delta b\|}{\|b\|} \tag{6.4}$$

and

$$\frac{\|\Delta x^B\|}{\|x + \Delta x^B\|} \leq \|B^{-1}\|\|B\|\frac{\|\Delta B\|}{\|B\|}. \tag{6.5}$$

In both of the inequalities the term $\|B^{-1}\|\|B\|$ reflects the *maximum* possible change in the *exact* solution of a linear system induced by a change in the data. This quantity is called *condition number* of a nonsingular matrix B and is defined by

$$\text{cond}(b) = \|B^{-1}\|\|B\|. \tag{6.6}$$

From (6.4) and (6.5) we can conclude that for a *well conditioned* matrix B, where $\text{cond}(B) \approx 1$, the relative error in the solution vector x can not be much greater than the inaccuracies in the problem data (possibly resulting from limited floating point precision). On the other hand for an *ill conditioned* matrix B, where $\text{cond}(B) \gg 1$, small errors in the problem data can lead to great changes in the solution of the linear system.

We want to emphasize that the condition number is totally problem dependent and can be determined a priori to its solution (which is very expensive though). In the context of the simplex method often the condition of an optimal basis is taken as a measure for the numerical difficulty of an LP problem. While the simplex method must terminate with an optimal basis, the path it takes to reach this basis greatly depends upon the deployed pivot selection strategies. Often, even problems, which are well conditioned around their optimal solution, feature regions of badly conditioned bases in their space of basic solutions. The goal of numerically stable implementations must be to avoid these regions or leave them as soon as possible when numerical difficulties occurred.

[2]Here we denote by $\|\cdot\|$ the euclidian norm with respect to vectors and the corresponding subordinate norm w.r.t. matrices such that $\|B\| = \sup_{\|x\|=1} \|Bx\|$.

The second long known result from error analysis (proof e.g. in [75, p. 54f.]) shows how the condition might evolve in a simplex basis change. If $\bar{B}$ denotes the new basis matrix, which emanates from the old basis matrix B according to equation (5.28), then the following upper bound on the condition number of the new basis can be derived:

$$\text{cond}(\bar{B}) \leq \frac{1}{|\alpha_q^r|} \cdot (1 + \|\alpha_q\|) \cdot (1 + \|\alpha_q\| + |\alpha_q^r|) \cdot \text{cond}(B) \tag{6.7}$$

The main consequence of inequality (6.7) is that small pivot elements should be avoided to prevent a substantial increase of the condition number in a basis change. In section 6.2.2 we will therefore discuss techniques to avoid the selection of small pivot element in the simple and in the bound flipping dual ratio test. Unfortunately, these techniques cannot guarantee stability under all possible circumstances, so we need further strategies to detect and to handle numerical problems during the solution process. In particular we must minimize if not eliminate the risk of numerical cycling. Such strategies are summarized in section 6.2.3. The remainder of the chapter is devoted to techniques to lessen the detrimental effects of degeneracy.

6.1.2 Degeneracy and cycling

When we talk about degeneracy in this section we always mean *dual* degeneracy as it is defined in section 2.4. In a degenerate iteration of the dual simplex method the entering variable has zero reduced costs, which results in a dual step length of zero and prevents an improvement of the objective function value. Note, that in the bound flipping ratio test, a degenerate basis does not necessarily lead to a degenerate iteration. Within a sequence of degenerate iterations and separated by the same succession of entering and leaving variables the same basis can occur repeatedly and the method can get stuck in an infinite cycle. This phenomenon is called *cycling.* In non-degenerate iterations the method proceeds exclusively to bases with strictly increasing objective function values, so no basis can be repeated. On some problems, sequences of degenerate iterations can get very long even if no cycling occurs and slow down the solution process considerably. Cunningham [17] introduced the term *stalling* to describe this behavior.

As we have learned in the previous section finite precision arithmetic almost always leads to tiny numerical inaccuracies in the results of practically all arithmetic operations. In the simplex method these inaccuracies act as small perturbations which resolve ties in the ratio test. This is often said to be the reason why *pure cycling* as defined above is very unlikely to occur in practise. Wunderling [75, p. 60] points out though, that nowadays simplex algorithms are often deployed to solve the LP relaxations of large, highly degenerated combinatorial optimization problems. These problems typically feature perfectly conditioned 0/1 constraint matrices such that virtually no numerical perturbations occur and degeneracy persists in its pure form. On these problems, the effects of stalling aggravate and pure cycling poses a real thread, which has to be handled properly.

Especially on badly conditioned problems and also as a possible consequence of Harris' ratio test (cf. section 6.2.2.1), small backward steps in the objective function can occur, sometimes even accompanied by a loss of dual feasibility. When the objective function value and the basic solution oscillate by very small amounts in subsequent iterations *numerical cycling* can occur, although there is no degeneracy present in the strict sense. This situation is well described in [25]. It constitutes one of the major challenges to stability in implementations of the simplex method.

Bland [12] was the first, who developed a pivot selection strategy (*smallest subscript* or *Bland's rule*) which guaranties termination of the simplex method in the presence of degeneracy. Unfortunately, it turned out to be inefficient in practise since it leads to a very large number of iterations. Since then, several anti-cycling and anti-degeneracy techniques with better practical behavior have been proposed. The most established are Wolfe's *'ad hoc' method* [74, 59], *perturbation* techniques like the one described in [9], the *EXPAND* procedure of [30] and *shifting* strategies like in [75, p. 60f.]. A good overview and description of these techniques can be found in [46, p. 230ff.]. They also seem to be well suited to prevent numerical cycling. While these techniques mainly effect the dual ratio test it is well known that dual steepest edge pricing (as described in section 3.3) can greatly reduce the number of degenerate iterations (especially in conjunction with perturbation). In our tests also a simple way of resolving ties in pricing by randomization turned out to be very useful to reduce stalling and diminish the risk of cycling.

In section 6.3 we will present the strategies which we incorporated into our code. They are based on the above ideas and came to our knowledge through the COIN LP implementation of the dual simplex method [44].

6.2 Techniques to ensure numerical stability

The overall strategy of our code to prevent numerical failure consists of three pillars:

1. A numerically stable LU-factorization and -update procedure. Computation of the initial primal and dual basic solutions (after refactorization) with highest possible precision.

2. A stabilizing ratio test, which avoids the selection of small pivot elements. Prohibition of very small pivot elements.

3. A "multi-layer safety net" consisting of techniques like e.g. numerical accuracy checks, optional abortion and restart of an iteration and double checking before reporting final results.

Regarding the LU-factorization we confer to section 5.2 and [68]. The main focus of this section lies on the points 2 (section 6.2.2) and 3 (section 6.2.3). We start by introducing the concept of numerical tolerances.

6.2.1 Numerical tolerances

The most important underlying computational technique to realize these strategies is the use of numerical tolerances. Table 6.1 gives an overview including notation and default values in our implementation. By the first two *feasibility tolerances* the

primal feasibility tolerance	ϵ^P	10^{-7}
relative primal feasibility tolerance	ϵ^r	10^{-9}
dual feasibility tolerance	ϵ^D	10^{-7}
pivot tolerance	ϵ^α	$[10^{-7}, 10^{-5}]$
zero tolerance	ϵ^z	10^{-12}
drop tolerance	ϵ^0	10^{-14}

Table 6.1: Tolerances: notation and default values.

feasibility definitions given in section 2.4 are slightly relaxed. Here, we say that a basis $\mathcal{B}$ is primal feasible if

$$l_j - l_j\epsilon^r - \epsilon^P \leq x_j \leq u_j + u_j\epsilon^r + \epsilon^P \qquad \text{for all } j \in \mathcal{B} \tag{6.8}$$

and dual feasible if

$$d_j \geq -\epsilon^D \qquad \text{for all } j \in \tilde{\mathcal{N}} \text{ with } x_j = l_j, \tag{6.9}$$

$$d_j \leq \epsilon^D \qquad \text{for all } j \in \tilde{\mathcal{N}} \text{ with } x_j = u_j \text{ and} \tag{6.10}$$

$$-\epsilon^D \leq d_j \leq \epsilon^D \qquad \text{for all } j \in \tilde{\mathcal{N}} \text{ with } x_j \text{ free,} \tag{6.11}$$

where $\tilde{\mathcal{N}}$ is the set of nonfixed nonbasic variables. Tolerances are indispensable to account for the limited arithmetic precision. The reason why especially the feasibility tolerances are much higher than machine precision (usually 10^{-16}) is that from the user's point of view a final result satisfying machine precision is usually not needed. Since primal feasibility is the stopping criterion of the dual simplex method (if an optimal solution exists) unnecessary iterations can be spared. A similar purpose is pursued by the use of a *zero tolerance* and a *drop tolerance*: arithmetic operations on insignificant numerical values are skipped if they are below the zero tolerance and physically set to zero if they are below the drop tolerance. Drop tolerances are particularly important in LU-factorization and LU-update. Besides the increase of efficiency tolerances are needed to improve numerical stability. Small pivot elements which might lead to an ill-conditioned basis are prohibited by a *pivot tolerance.* If a pivot element is below this threshold the iteration is aborted and no basis change is performed. We will describe this mechanism in detail in section 6.2.3. In the next section we will discuss how the dual feasibility tolerance can be used to allow for the selection of better sized pivot elements.

6.2.2 Stabilizing ratio tests

The first who proposed to utilize feasibility tolerances to find better pivot elements in the ratio test was P. Harris [33]. In the next two sections we will discuss several variants of her idea for the simple and the bound flipping dual ratio test.

Algorithm 17: Modified standard ratio test

```
if $x_p < l_p$ then set $\tilde{\alpha}^r \leftarrow -\alpha^r$, if $x_p > u_p$ then set $\tilde{\alpha}^r \leftarrow \alpha^r$.
Let $\mathcal{F} = \{j \,:\, j \in \tilde{\mathcal{N}}, x_j$ free or $(x_j = l_j$ and $\tilde{\alpha}^r_j > \epsilon^\alpha)$ or
                                        $(x_j = u_j$ and $\tilde{\alpha}^r_j < -\epsilon^\alpha)\}$.
if $\mathcal{F} = \emptyset$ then abort iteration: the LP is tentatively dual unbounded.
else
    $\theta \leftarrow \infty$
    foreach $j \in \mathcal{F}$ do
        if $\frac{d_j}{\tilde{\alpha}^r_j} < \theta$ then
            $\theta^D \leftarrow \frac{d_j}{\tilde{\alpha}^r_j}$
            $q \leftarrow j$
        else if $\frac{d_j}{\tilde{\alpha}^r_j} < \theta + \epsilon^z$ then
            if $|\tilde{\alpha}^r_j| > |\tilde{\alpha}^r_q|$ then
                $\theta \leftarrow \frac{d_j}{\tilde{\alpha}^r_j}$
                $q \leftarrow j$
            end
        end
    end
    $\theta^D \leftarrow \frac{d_q}{\alpha^r_q}$
end
```

6.2.2.1 Modified standard ratio test

The main disadvantage of the "textbook" ratio test as we described it in section 3.1.4 is that there is practically no freedom to choose between different entering variables and their associated pivot elements. In fact, the minimum in equation (3.13) is unique if neither the current nor the next iteration is dual degenerate. From a numerical point of view even in these cases the minimum will be unique since most ties are resolved by small numerical inaccuracies on the level of the machine precision.

A variant of the "textbook" ratio test is shown in algorithm 17. It encourages the selection of greater pivot elements in the case of near degeneracy. This is done at the cost of creating small dual infeasibilities. Since the zero tolerance ϵ^z is several magnitudes smaller than the dual feasibility tolerance ϵ^D, these newly created dual infeasibilities usually do not entail a switch to dual phase I. However, the existence even of small dual infeasibilities always brings about new numerical dangers. We will come back to this issue at the end of this section. Algorithm 17 also prevents pivot elements below the pivot tolerance. Since the pivot tolerance is several magnitudes greater than machine precision we cannot declare dual unboundedness right away if the candidate set $\mathcal{F}$ is empty. Therefore, we change into a *tentative solution state*, which we try to verify in the further course of the method (cf. section 6.2.3). Since algorithm 17 requires only one single loop over the transformed pivot row α^r it is called a *one-pass ratio test.*

6.2.2.2 Harris' ratio test

To increase the flexibility of the ratio test even further, Harris [33] proposed to accept the creation of infeasibilities within the full range of the feasibility tolerance. We will describe her approach in the context of the dual ratio test for the case $t \geq 0$. In a first step (or *pass*) a bound Θ_{max} on the dual step length θ^D is computed which ensures that the respective strict dual feasibility conditions of the nonbasic nonfixed variables are not violated by more than the dual feasibility tolerance. For this purpose we split the candidate set $\mathcal{F}^+$ as define in equation (3.12) into two subsets

$$\mathcal{F}_l^+ = \{j \in \mathcal{N} : (x_j\ free \text{ or } x_j = l_j) \text{ and } \alpha_j^r > 0\} \quad \text{and} \tag{6.12}$$

$$\mathcal{F}_u^+ = \{j \in \mathcal{N} : (x_j\ free \text{ or } x_j = u_j) \text{ and } \alpha_j^r < 0\}. \tag{6.13}$$

Then it is not difficult to see that Θ_{max} can be computed as follows:

$$\Theta_{max} = \min\left\{ \min_{j \in \mathcal{F}_l^+} \left\{ \frac{d_j + \epsilon^D}{\alpha_j^r} \right\}, \min_{j \in \mathcal{F}_u^+} \left\{ \frac{d_j - \epsilon^D}{\alpha_j^r} \right\} \right\}. \tag{6.14}$$

In a second pass, among all breakpoints which do not exceed the bound Θ_{max} the one with the pivot element of greatest absolute value $|\alpha_j^r|$ is chosen to determine the entering variable and the dual step length:

$$q \in \arg\max_{j \in \mathcal{F}^+} \left\{ |\alpha_j^r| \ : \ \frac{d_j}{\alpha_j^r} \leq \Theta_{max} \right\} \quad \text{and} \quad \theta^D = \frac{d_q}{\alpha_q^r}. \tag{6.15}$$

Undoubtedly, Harris' ratio test yields better sized pivot elements and therefore limits the deterioration of the basis condition in the LU-update. However, it also comes with two severe disadvantages:

1. The method requires in principle two passes of the transformed pivot row α^r. Although the second pass can be limited to the elements in the candidate set $\mathcal{F}^+$, the computational effort is considerably higher compared to a one pass method.

2. The creation of small dual infeasibilities is an inherent property of this ratio test. If the methods selects such a slightly dual infeasible variable as leaving variable, the corresponding breakpoint and therefore the dual step length will be of wrong sign. This can have two perilous consequences. Firstly, the dual objective function will move into the wrong direction which brings up the danger of numerical cycling. Secondly, during the update of the reduced cost vector other nonbasic variables belonging to the candidate set $\mathcal{F}^-$ can be pushed beyond the dual feasibility tolerance. This will provoke a call to the dual phase I, which again results in a decreasing objective function value and the thread of numerical cycling. To understand why this case can actually happen, imagine a nonbasic variable $j \in \mathcal{F}^-$ with $x_j = l_j$, which is already at the

edge of dual infeasibility with $d_j = -\epsilon^D$. Even the smallest move with a dual step length $\theta^D < 0$ would push this variable beyond the tolerance. Although in the case of backward steps the dual step length is bounded by $-\left|\frac{\epsilon^D}{\alpha_q^r}\right|$ from below, variables in $\mathcal{F}^-$ move by an amount of $\left|\epsilon^D \frac{\alpha_j^r}{\alpha_q^r}\right|$ towards their infeasibility bound (or pass it by this amount in the worst case), which can be a high multiple of the feasibility tolerance.

In implementation we skip candidates with pivot elements that fall below the pivot tolerance (as in algorithm 17). We also made some good experiences with a variant, where we use a special tolerance ϵ^H instead of ϵ^D in equation 6.14. Depending on the current numerical status ϵ^H varies between ϵ^z and ϵ^D (a typical value is $0.01 * \epsilon^D$). The idea is to find a good compromise between the need for better pivot elements in numerically difficult situations and the unnecessary creation of significant dual infeasibilities. Note, that with $\epsilon^H < \epsilon^D$ it may happen that $\Theta_{max} < 0$ if a reduced cost value d_j is slightly dual infeasible with $\epsilon^H < |d_j| \leq \epsilon^D$. This case can actually occur since we always have to anticipate dual infeasibilites up to the dual feasibility tolerance, which is also the stopping criterion for dual phase I. Theoretically it is no problem but it has to be considered properly in implementation.

6.2.2.3 Shifting

As mentioned before we always have to be able to deal with dual infeasibilities within the tolerance in the dual ratio test, no matter wether we use a Harris variant or not. This is true due to two main reasons: 1. Usually, the dual phase I is terminated when no dual infeasibility exceeds a certain phase I feasibility tolerance ϵ^{D1} (e.g. $\epsilon^{D1} = 0.1 * \epsilon^D$). So there may still exist dual infeasibilities below this tolerance. 2. For numerically very difficult problems, numerical errors can reach the level of the feasibility tolerances. One consequence can be that new dual infeasibilities (above or below the tolerance) occur after the recomputation of the reduced cost in the context of a refactorization.

While the above variant of Harris ratio test reduces the danger of "really" loosing dual feasibility (resulting in a call to dual phase I), we still have to avoid backward steps to rule out the risk of numerical cycling. The main approach to do this is called *shifting* and was worked out in detail for the primal simplex method in [30] in the context of the anti-cycling procedure EXPAND. The idea is to locally manipulate the problem data to avoid the negative step. For the dual simplex method this means manipulations of cost coefficients. We discussed the same idea in section 4.4 as a way to make an arbitrary basis dual feasible. Here, we use it to achieve a zero or even slightly positive dual step length and still fulfill the dual basic constraint for the entering variable, such that $\bar{d}_q = 0$ after the basis change.

Suppose, that we have determined a dual step length $\theta^D = \frac{d_q}{\tilde{\alpha}_q^r} < 0$ at the end of the dual ratio test. Then the following is done:

1. Set $\theta^D \leftarrow 0$ (or alternatively: $\theta^D \leftarrow 10^{-12}$).

2. Compute shift $\Delta_q \leftarrow \theta^D \tilde{\alpha}_q^r - d_q$.

3. Set $c_q \leftarrow c_q + \Delta_q$.
   Set $d_q \leftarrow d_q + \Delta_q$.

To avoid a negative step it suffices to set θ^D to zero in step 1. In the EXPAND procedure it is recommended to set θ^D to a small positive value in the order of the zero tolerance to achieve a guaranteed positive step and prevent cycling. This can obviously make other nonbasic variables violate their feasibility bound after the update of the reduced costs. Therefore, in the EXPAND concept the positive "mini-step" is accompanied by a small increase of the feasibility tolerance in every iteration. The disadvantage of expanding the tolerance is though, that the new tolerance is valid for all of the nonbasic variables and thus allows for greater infeasibilities in the further course of the method. For this reason we choose an alternative way which is inspired by the COIN implementation of the dual simplex method [44]. We perform further shifts only for those reduced cost values which would violate the feasibility tolerance. These shifts are computed as follows:

$$\Delta_j = \begin{cases} \max\left\{\theta^D \tilde{\alpha}_j^r - d_j - \epsilon^D, 0\right\} & \text{if } j \in \mathcal{F}_l, \\ \min\left\{\theta^D \tilde{\alpha}_j^r - d_j + \epsilon^D, 0\right\} & \text{if } j \in \mathcal{F}_u. \end{cases} \tag{6.16}$$

Since $\theta^D = 10^{-12}$ is very small, only few of these additional shifts are different from zero and among these most are in the range of 10^{-14} to 10^{-10}. That way the changes to the cost coefficients are kept as small as possible. We also experimented with random shifts of greater magnitude to achieve a more significant positive step, a greater distance to infeasibility and resolve degeneracy at the same time (a similar shifting variant was proposed by [75, p. 61] to break stalling and prevent cycling). However, our impression was that the COIN procedure is superior, if it is combined with the perturbation procedure described in section 6.3.1. There we will also discussed how the cost modifications are removed.

6.2.2.4 Stabilizing bound flipping ratio test

The bound flipping ratio test as described in section 3.2 and algorithm 4 already offers a much greater flexibility to avoid small pivots that the standard ratio test. Let

$$k \in \{1, \ldots, |\mathcal{F}|\} \; : \quad \delta_k \geq 0 \quad \text{and} \quad \delta_{k+1} < 0 \tag{6.17}$$

be the last mini-iteration before the slope δ becomes negative. If $|\tilde{\alpha}_{q_k}^r|$ is too small and much smaller the largest available pivot element we can always go back to a preceding mini-iteration. The goal is the find a good compromise between a well sized pivot element and a good progress in the objective function. However, especially for problems with few boxed variables, the additional flexibility does not suffice. Also, it might sometimes be more efficient to accept small infeasibilities and achieve a lower iteration count by going the farthest possible step in the bound flipping ratio test. Therefore, we will now describe, how Harris' idea can be incorporated into the BFRT.

We basically apply Harris' ratio test in every mini-iteration. Let $\mathcal{Q}^i$ be the set of remaining breakpoints in mini-iteration i and

$$\mathcal{Q}_l^i = \left\{ j \in \mathcal{Q}^i : \tilde{\alpha}_j^r > 0 \right\} \quad \text{and} \tag{6.18}$$
$$\mathcal{Q}_u^i = \left\{ j \in \mathcal{Q}^i : \tilde{\alpha}_j^r < 0 \right\}. \tag{6.19}$$

Then an upper bound Θ_{max}^i on the maximal step length in mini-iteration i can be determined by

$$\Theta_{max}^i = \min \left\{ \min_{j \in \mathcal{Q}_l^i} \left\{ \frac{d_j + \epsilon^D}{\tilde{\alpha}_j^r} \right\}, \min_{j \in \mathcal{Q}_u^i} \left\{ \frac{d_j - \epsilon^D}{\tilde{\alpha}_j^r} \right\} \right\}. \tag{6.20}$$

Instead of looking at only one breakpoint per mini-iteration we then consider all remaining breakpoints at once, which do not exceed the maximal step length Θ_{max}^i. We denote the set of these breakpoints by $\mathcal{K}_i$ with

$$\mathcal{K}_i = \left\{ j \in \mathcal{Q}^i : \frac{d_j}{\tilde{\alpha}_j^r} \leq \Theta_{max}^i \right\}. \tag{6.21}$$

Finally, we can determine the possible entering index q_i and the corresponding dual step length by

$$q_i \in \arg\max_{j \in \mathcal{K}_i} \left\{ |\tilde{\alpha}_j^r| \right\} \quad \text{and} \quad \theta_i^D = \frac{d_{q_i}}{\tilde{\alpha}_{q_i}^r} \tag{6.22}$$

and perform the update operations

$$\mathcal{Q}_{i+1} = \mathcal{Q}_i \setminus \mathcal{K}_i \quad \text{and} \tag{6.23}$$
$$\delta_{i+1} = \delta_i - \sum_{j \in \mathcal{K}_i} (u_j - l_j) |\tilde{\alpha}_j^r|. \tag{6.24}$$

Again we can go back to preceding mini-iterations if the pivot element corresponding to the greatest possible step is not good enough. An important implementation issue is choice of a adequate data structure for the set of breakpoints. The incorporation of Harris' ideas has significant consequences for this choice. [27] and [47] recommend to organize the set of breakpoints as a heap data-structure to allow for a very fast access to the next smallest breakpoint. This implicates a considerable computational effort to build up the heap structure, which arises independently of the number of actually performed mini-iterations. We conducted some experiments, which approved Maros' observation, that the number of flipped variables may vary dramatically even in subsequent iterations. Furthermore, on instances with few or no boxed variables virtually no bound flips are possible at all. Therefore, we opted for an implementation based on simple linear search combined with a forceful reduction technique for the set of eligible candidate breakpoints.

Algorithm 18 describes our algorithmic realization of the BFRT with Harris' tolerance, which is inspired by the dual simplex implementation of the COIN LP code [44]. It proceeds in three phases. In phase 1, which needs one pass of the transformed pivot row, a complete set of breakpoints $\mathcal{Q}$ is determined. At the same time a Harris bound Θ_{max} is computed with respect to the first possible breakpoint. This bound is used as a starting point in phase 2, which tries to reduce the set of candidate breakpoints. This is done by doubling the dual step length and collecting passed breakpoints until the slope changes sign. Only for these breakpoints the actual BRFT+Harris procedure with its subsequent linear searches is conducted. Typically the number of candidates can be greatly reduced by very few passes in phase 2 and therefore, phase 3 can be carried out very efficiently.

6.2.3 Refactorization, accuracy checks and stability control

As described in chapter 5 the solution of the linear systems required by the dual simplex method (up to three FTran and one BTran operation per iteration) is performed via an LU-factorization of the basis, which is updated in every iteration. In each execution of the LU-update procedure the LU-representation of the basis (cf. equation (5.50)) is extended by one additional row-eta-vector. Therefore, the computational effort to solve the linear systems constantly increases. On the other hand the update procedures described in section 5.3 do not inherently consider numerical stability. So even if a good pivot element in found in the ratio test, the numerical stability of the LU-representation may deteriorate significantly due to small diagonal elements used during the elimination process in the LU-update. Therefore, an important question of a simplex implementation is, how often a refactorization should be performed to achieve both: speed *and* numerical stability. Modern simplex codes all apply basically the same strategy: determine a *refactorization frequency*[3] which optimizes speed and perform additional refactorizations, when numerical problems occur.

6.2.3.1 Refactorization for speed

The basic reasoning to find a refactorization frequency with respect to speed is the following (cf. [15, p. 111.]). Let T_0 be the time needed for refactorization and T_i be the time spent on FTran and BTran operations in iteration i, then

$$T_r^* = \frac{1}{r}\sum_{i=0}^{r} T_i \tag{6.25}$$

is the average time required to solve the linear systems in the r-th iteration since the last refactorization. Obviously, the $T_1^*, T_2^*, T_3^*, \ldots$ first decrease, since the overhead T_0 gets distributed over more and more iterations, and then from a certain point start to increase, since the growing number of L-eta-vectors begins to dominate the solution process. Based on this simple observation, there are two relevant approaches to determine a good refactorization frequency ϕ:

1. Constant frequency: ϕ is set to a constant value at the start of the algorithm and is not changed during a solution run. The justification is, that the T_i can be assumed to be proportional to the number of rows (size of the basis) with some constant factor. Then the minimization of T_r^* leads to a constant value

[3]Number of iterations between two factorizations.

Algorithm 18: Bound flipping ratio test with Harris' tolerance.

```
Phase 1: Determine candidate set.
If $x_p < l_p$ set $\tilde{\alpha}^r \leftarrow -\alpha^r$ and $\delta_0 \leftarrow l_p - x_p$. If $x_p > u_p$ set $\tilde{\alpha}^r \leftarrow \alpha^r$ and $\delta_0 \leftarrow x_p - u_p$.
Compute $\mathcal{Q} \leftarrow \{j : j \in \mathcal{N}, x_j$ free or $(x_j = l_j$ and $\tilde{\alpha}^r_j > 0)$ or
                                   $(x_j = u_j$ and $\tilde{\alpha}^r_j < 0)\}$.
Compute $\Theta_{max} \leftarrow \min\left\{\min_{j\in\mathcal{Q}_l}\left\{\frac{d_j+\epsilon^D}{\tilde{\alpha}^r_j}\right\}, \min_{j\in\mathcal{Q}_u}\left\{\frac{d_j-\epsilon^D}{\tilde{\alpha}^r_j}\right\}\right\}$.

Phase 2: Reduce candidate set. Find interesting region for $\theta^D$.
Set $\theta^D \leftarrow 10.0 * \Theta_{max}$.
Set $\delta \leftarrow \delta_0$.
Set $\hat{\delta} \leftarrow 0$.
while $\delta - \hat{\delta} \geq 0$ do
    Set $\delta \leftarrow \delta - \hat{\delta}$.
    Compute $\tilde{\mathcal{Q}} \leftarrow \left\{j \in \mathcal{Q}_l : d_j - \theta^D\tilde{\alpha}^r_j < -\epsilon^D\right\} \cup \left\{j \in \mathcal{Q}_u : d_j - \theta^D\tilde{\alpha}^r_j > \epsilon^D\right\}$.
    Compute $\Theta_{max} \leftarrow \min\left\{\min_{j\in\mathcal{Q}_l}\left\{\frac{d_j+\epsilon^D}{\tilde{\alpha}^r_j}\right\}, \min_{j\in\mathcal{Q}_u}\left\{\frac{d_j-\epsilon^D}{\tilde{\alpha}^r_j}\right\}\right\}$.
    Set $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \tilde{\mathcal{Q}}$.
    Set $\hat{\delta} \leftarrow \sum_{j\in\tilde{\mathcal{Q}}}(u_j - l_j)|\tilde{\alpha}^r_j|$.
    Set $\theta^D \leftarrow 2.0 * \Theta_{max}$.
end

Phase 3: Perform BFRT with Harris' tolerance on interesting region.
while $\tilde{\mathcal{Q}} \neq \emptyset$ and $\delta \geq 0$ do
    Compute $\Theta_{max} \leftarrow \min\left\{\min_{j\in\tilde{\mathcal{Q}}_l}\left\{\frac{d_j+\epsilon^D}{\tilde{\alpha}^r_j}\right\}, \min_{j\in\tilde{\mathcal{Q}}_u}\left\{\frac{d_j-\epsilon^D}{\tilde{\alpha}^r_j}\right\}\right\}$.
    Let $\mathcal{K} = \left\{j \in \tilde{\mathcal{Q}} : \frac{d_j}{\tilde{\alpha}^r_j} \leq \Theta_{max}\right\}$.
    Select $q \in \arg\max_{j\in\mathcal{K}}\left\{|\tilde{\alpha}^r_j|\right\}$.
    Set $\tilde{\mathcal{Q}} \leftarrow \tilde{\mathcal{Q}} \setminus \mathcal{K}$.
    Set $\delta \leftarrow \delta - \sum_{j\in\mathcal{K}}(u_j - l_j)|\tilde{\alpha}^r_j|$.
end
If $\tilde{\mathcal{Q}} = \emptyset$ then terminate: the LP is dual unbounded.
Set $\theta^D \leftarrow \frac{d_q}{\alpha^r_q}$.
```

for ϕ. A semi-rigorous analysis (for PFI) for this approach can be found in [15, p. 112f.]). For LU-factorization and update $\phi = 100$ has turned out to be a reasonable value. Fine-tuning for special problem classes is left to the user.

2. Dynamic frequency: ϕ is dynamically changed during the solution run. In an analysis conducted by [75, p. 85f.] he assumes that the times T_0 and T_i mainly depend on the number of nonzero elements in the current basis matrix and the updated LU-representation, respectively. By simple bookkeeping he keeps track of the quantities T_r^* and triggers a refactorization, when they start increasing.

In our implementation we made good experiences with the simpler constant refactorization frequency. We use a slightly modified variant, where we incorporate the number of rows m in the determination of ϕ in the following way:

$$\phi = \min\left\{100 + \frac{m}{200}, 2000\right\}. \tag{6.26}$$

The constants in (6.26) are dependent on the system architecture and individual characteristics of the code and have been determined by extensive benchmarking. The reason to choose a higher refactorization frequency for models with many rows is, that the time for refactorization shows a greater dependency on the number of rows than the solution times for FTran and BTran, especially for hypersparse problems.

6.2.3.2 Refactorization for stability

In every LU-update the numerical accuracy of the LU-factors decreases due to the accumulation of small errors caused by the limited precision arithmetic. The degree of accumulation and amplification depends on the condition number of the basis matrix, which may worsen in basis changes with small pivot elements ($< 10^{-4}$ usually causes problems). For numerically easy problems (e.g. those with 0/1 problem matrices and well modeled problems) both of these threads are irrelevant and no additional refactorizations are necessary. For some problems, especially those which are badly modeled, significant errors can occur, which in most cases can be fixed or even prevented by a precocious refactorization and recomputation of the primal and dual basic solution. In the following we give a list of computationally cheap accuracy checks and "numerical events", which can be used to monitor the numerical status of a solution run:

- **Pivot element test.** After the FTran operation in step 7 of elaborated dual simplex method (algorithm 7), two versions of the pivot element α_q^r are available. The first version, which we will denote by α^{BT}, is the q-th entry of the transformed pivot row, which was computed in step 7:

$$\alpha^{BT} = (A_{\mathcal{N}}^T \rho_r)_q. \tag{6.27}$$

The second version denoted by α^{FT} is the r-th entry of the transformed pivot column computed in step 7:

$$\alpha^{FT} = (B^{-1}a_q)_r \tag{6.28}$$

If the deviation $|\alpha^{BT} - \alpha^{FT}|$ of these two quantities exceeds a certain threshold ϵ^s, a refactorization is triggered. In our code we set

$$\epsilon^s = 10^{-9} * (1 + |\alpha^{FT}|) \tag{6.29}$$

In general, α^{FT} can be assumed to be numerically more precise than α^{BT}, since besides the FTran operation no additional computations are involved. Therefore, one option in implementation is to always proceed with α^{FT} after FTran and to (re-)compute the primal and dual step length. While the update of the primal and dual basic solution vectors can be performed with greater accuracy in this case, the new dual step length might slightly differ from the value assumed in the dual ratio test. This can lead to the (uncontrolled) loss of dual feasibility, since it clearly counteracts the shifting policy in the dual ratio test. Therefore, we keep using α^{BT} in spite of its potential numerical weaknesses[4].

- **LU-update test.** We already described this accuracy check in section 5.3.1 as a feature of the Forrest/Tomlin update procedure. If

$$|\alpha_q^r - \frac{\bar{u}_m^m}{u_t^t}| > \epsilon^u \tag{6.30}$$

the basis change is aborted and a refactorization is triggered. To prevent a repetition the selected leaving variable is excluded from pricing for a certain number of iterations. In our code we set

$$\epsilon^u = 10^{-6} \cdot |\alpha_q^r|. \tag{6.31}$$

- **DSE weight test.** As mentioned in section 3.3 the dual steepest weight β_r corresponding to the leaving variable can be recomputed from its definition at low costs since the r-th row of the basis matrix is readily available from a previous step of the algorithm (computation of the transformed pivot row). Let

$$\beta_r^U = \beta_r \quad \text{and} \tag{6.32a}$$
$$\beta_r^R = \rho_r^T \rho_r \tag{6.32b}$$

be the updated and the recomputed version of β_r, respectively. Due to the higher accuracy it is advisable to always replace β_r by β_r^R. Furthermore, the deviation between these two version of β_r can be used to get an impression of the current accuracy of the DSE weights, which can have great impact on the effectiveness of DSE pricing (bad accuracy can lead to a significant increase

[4]In the remainder of this text we always assume that $\alpha_q^r = \alpha^{BT}$.

of the total number of iterations). We do not use this test as a trigger for refactorization or even recomputation of all DSE weights from their definition (usually is very expensive), but as a tool in development and debugging to improve the general numerical behavior of our code.

- **Reduced cost test.** For the reduced cost value d_q associated with the entering variable q the following is true:

$$d_q = c_q - y^T a_q = c_q - c_{\mathcal{B}}^T B^{-1} a_q = c_q - c_{\mathcal{B}}^T \alpha_q. \tag{6.33}$$

Therefore, d_q can be recomputed using the transformed pivot column α_q after the FTran operation in step 7 of algorithm 7. The recomputed version of d_q is in general more accurate than the updated version. The deviation between the two versions can be used to measure the accuracy of the reduced cost vector d. If it exceeds a certain threshold (e.g. 10^{-9}), the recomputation of the dual basic solution from its definition or a refactorization can be triggered. As in the pivot element test one could also replace d_q by the recomputed, more accurate value. However, the induced change in the dual step length can lead to dual infeasibilities and counteract the shifting strategy in the dual ratio test. In the latest version we use this test merely as a tool in development and debugging.

- **Primal and dual error.** Let $\hat{x}_{\mathcal{B}}$ and $\hat{y}$ be the primal and dual basic solution vectors resulting from the solution of the corresponding systems 2.13 and 2.15a. Then the (maximal) primal error err^P and the (maximal) dual error err^D are defined as

$$err^P = \max_{i \in \{1,\dots,m\}} \{|(\tilde{b} - B\hat{x}_{\mathcal{B}})_i|\} \quad \text{and} \tag{6.34}$$

$$err^D = \max_{i \in \{1,\dots,m\}} \{|(c_{\mathcal{B}} - B^T \hat{y})_i|\}. \tag{6.35}$$

Computing err^P and err^D in this manner is rather expensive. Therefore we use these measures only for development and debugging.

- **Loss of feasibility.** In spite of all precautions entries of the dual basic solution can become dual infeasible beyond the dual infeasibility tolerance due to numerical errors. It is important to incorporate adequate test to detect these entries. In our code this is done in the dual ratio test. When a reduced cost value is found to be dual infeasible the iteration is aborted, a refactorization and a switch to the dual phase I is triggered.

6.3 Techniques to reduce degeneracy and prevent cycling

6.3.1 Perturbation

As one can conclude from the definitions in section 2.4, degeneracy is in the first instance a property of a given LP problem. If most of the bases of an LP problem are degenerate by a high degree (dual, primal or both), the problem is said to be *highly degenerate*. Consequently, also the susceptibility to stalling is a property of the problem data. It is well known, that particularly the LP relaxations of combinatorial optimization problems are often highly degenerate. The basic idea of the perturbation technique is to lower the amount of problem-inherent degeneracy by randomly changing the problem data by small margins. Primal degeneracy can be resolved by perturbing the right-hand-side and/or the bounds of a problem. Dual degeneracy can be resolved by perturbing the cost vector. In the following we will consider only the latter.

Many variants of the perturbation method have been discussed in the literature. They differ with respect to the following points:

- **When to perturb?** Is the perturbation applied at the start of the method or only when stalling occurs?
- **What to perturb?** Do we perturb only the cost coefficients of the degenerate positions or all of them?
- **How to perturb?** Do we perturb by random quantities within a fixed interval or do we consider the size of the cost coefficients or other measures?
- **How to remove the perturbation?** Do we remove the perturbations during or only at the end of the solution run? Which method is used to remove them?

Our perturbation procedure, which is similar to the one implemented in the COIN LP code [44], is applied prior to the dual simplex method, if many cost coefficients have the same value. In fact, we sort the structural part of the cost vector and count the number of different values. If this number is less then one fourth of the total number of structural variables, we assume the problem to be significantly dual degenerate and perturb all structural non-fixed, non-free variables. If the problem was not perturbed at the start and the objective function value does not improve for at least `maxcycle` iterations (we use $\texttt{maxcycle} = 3 \cdot \phi$, where ϕ is the refactorization frequency), then we perturb only the degenerate positions of the above subset of the cost vector.

The determination of the cost changes is the main difference compared to other methods from the literature. While these methods aim solely at resolving degeneracy, our perturbation procedure also tries to exploit degeneracy to keep the number of nonzero elements in the basis matrix low. In general, the presence of degeneracy increases the flexibility in the dual ratio test. If the problem is numerically stable, this flexibility can be used to select entering variables with as few nonzero elements in their corresponding columns of the problem matrix as possible. The effect is, that the spike in LU-update stays sparser and the number of nonzeros added to the LU-factorization lower. But if the problem is perturbed at the start, most of this flexibility is lost[5]. Therefore, the number of nonzero elements is considered in the determination of the cost perturbations. The determination of a cost perturbation ξ_j is performed in four steps:

1. Determine the right magnitude of ξ_j. Set ξ_j to a fixed value, which consists of a constant fraction (typically $100 \cdot \epsilon^D$) and a variable fraction dependent on the size of the cost coefficient (typically $\psi \cdot c_j$, with $\psi = 10^{-5}$).

2. Randomize ξ_j and set the right sign w.r.t. dual feasibility. This is done by setting
$$\xi_j \leftarrow \begin{cases} -0.5\xi_j(1+\mu) & \text{if } u_j < \infty, \\ 0.5\xi_j(1+\mu) & \text{o.w.,} \end{cases} \tag{6.36}$$
where μ is a random number within the interval $[0, 1]$. Note, that by using equation (6.36) ξ_j basically keeps the size determined in step 1.

3. Incorporate nonzero counts. Dependent on the number ν_j of nonzero elements in column a_j, ξ_j is multiplied by a weighting factor w_k:
$$\xi_j \leftarrow w_{\nu_j} \cdot \xi_j. \tag{6.37}$$
The weight-vector w displays the tradeoff between the two goals "resolve degeneracy" and "keep nonzero count low". We use the following scheme:
$$w^T = (10^{-2}, 10^{-1}, 1.0, 2.0, 5.0, 10.0, 20.0, 30.0, 40.0, 100.0). \tag{6.38}$$
If $\nu_j > 10$ we use $w_{10} = 100.0$ in equation (6.37). As mentioned before most practical LP problems have less than fifteen nonzero entries per column independent of the number of rows.

4. Ensure that ξ_j stays within an interval $[\xi_{min}, \xi_{max}]$ with
$$\xi_{min} = \min\{10^{-2}\epsilon^D, \psi\} \quad \text{and} \tag{6.39}$$
$$\xi_{max} = \max\{10^3\epsilon^D, \psi \cdot 10 \cdot \tfrac{1}{n} \sum_{j=1}^{n} c_j\}. \tag{6.40}$$
If ξ_j exceeds or falls below these thresholds, it is multiplied with 0.1 or 10, respectively, until it falls into the above interval.

At the end of the dual simplex method we restore the original cost vector, which can lead to a loss of dual feasibility of the current basic solution. If the method terminated due to primal feasibility, we switch to the primal simplex method (to be more precise: primal phase II) to restore dual feasibility (cf. [9]). Usually, only few iterations are necessary to find a new optimal basis. If the method terminated

[5]In this sense the perturbation also works against Harris' ratio test.

due to dual unboundedness, we just restore the cost vector and start over with the dual phase I if necessary to confirm unboundedness. The same procedure is used to remove the cost shiftings performed in the dual ratio test.

6.3.2 Randomized pricing

It is well known that dual steepest edge pricing as described in section 3.3 is by itself an effective technique to reduce the number of degenerate iterations. The leaving variable is determined by equation 3.53. While for most problems ties are rather unlikely, they occur with increased frequency on well conditioned LP problems, which are highly degenerate. One possible criterion to resolve a tie in dual pricing is the number of nonzero elements in the corresponding column of the U-part of the LU-factorization. However, we made better experiences with resolving the ties totally randomly. On some problems this also lead to a dramatic reduction of (dual) degenerate iteration. Implementation details are given in sections 8.2.2.

Chapter 7

Further computational aspects

7.1 LP preprocessing, scaling and crash procedures

LP preprocessing, scaling and crash procedures are performed prior to the (dual) simplex algorithm. While preprocessing and scaling methods aim at improving the model formulation given by the user, the crash procedure is supposed to quickly obtain a starting basis which is superior to the all logical basis.

LP preprocessing

It is widely recognized that LP preprocessing is very important for solving large-scale linear and integer optimization problems efficiently (cf. [14] and [49]). This is true for both interior point and simplex algorithms. Although LP software and computers have become much faster, LP models have increased in size. Furthermore, LP optimizers are used in interactive applications and in integer programming where many LP problems have to be solved. More efficient algorithms and improved implementation techniques are therefore still very important. Furthermore all practical LP/IP models are generated by computer programs either directly or within a modeling system. The model generator derives the computer model from the mathematical model structure and the model data. Most model generators have very limited capabilities for data analysis. As a consequence, there is usually a significant part of the model that is redundant. The main goals of LP preprocessing are:

- eliminate as many redundant constraints as possible
- fix as many variables as possible
- transform bounds of single structural variables (either tightening / relaxing them during LP preprocessing or tightening bounds during IP preprocessing)
- reduce the number of variables and constraints by eliminations

We refer here to the techniques described in [49]. The standard LP-preprocessing for LPs to be solved with an interior point or primal simplex algorithm uses bound tightening in an early phase of the LP-preprocessing. At the end of LP-preprocessing there is a reverse procedure where bounds are relaxed, i.e. redundant bounds are removed from the model.

As mentioned previously, boxed variables play a key role in the dual simplex algorithm. Therefore tightened bounds are not relaxed if the dual simplex algorithm is used in our code. In chapter 9 we demonstrate the surprising impact of this strategy in our computational results.

Scaling

Scaling procedures try to decrease the numerical difficulty of a user given (or already preprocessed) model by reducing the spread of magnitudes of nonzeros in the problem data. This is done by multiplying the rows and columns of the constraint matrix with real valued *scaling factors*. There are several methods in the literature to determine these factors, surveys can be found in [73] and [46, p. 110ff.]. The method used in MOPS was first presented in [9] and is implemented with minor modifications described in [65].

Crash

The purpose of a crash procedure is to find a good starting basis for the (dual) simplex method with the following properties:

1. Easy to factorize. The all logical basis is clearly the best choice with respect to this requirement. However, it leaves no flexibility to take other considerations into account. Therefore, most crash procedures try to generate a triangular or near triangular basis as a good compromise.

2. As feasible, optimal and sparse as possible. Ties in the triangulation procedure are resolved by a weight function preferring variables with a high primal feasibility range and a low nonzero count in the corresponding column of the constraint matrix. Free variables are tried to be pivoted into and fixed variables out of the basis. The probability of a variable to be part of an optimal basis is estimated by the objective function coefficient and simple geometric arguments (cf. [54]).

3. Numerically stable. As in the factorization procedure the threshold pivoting criterion (cf. [23]) can be used to avoid the generation of an ill-conditioned basis.

An overview of common crash procedures can be found in [46, p. 244ff.]. The method included in the MOPS system is described in [65]. In the context of this dissertation we did not change this procedure. However, we conducted some computational tests with the result that we could not achieve a significant performance improvement compared to the all logical basis. One important reason is the increased density of a crash basis, which leads to slower iterations in the beginning of a solution run and has to be overcompensated by a reduction of the total iteration count. In the context of the dual simplex method this is problematic due to the other important reason, which is dual steepest edge pricing. If we start the dual simplex method with an all logical basis, then the dual steepest edge weights β_i can be initialized with 1.0, which is the correct value with respect to their definition. For any other starting basis, we can either keep this cheap initialization as a heuristic or compute the exact weights according to formula 3.45. This is computationally very expensive since it requires the solution of up to m linear systems for ρ_i plus up to m inner products. Using crash and the simple initialization lead to disastrous results: both the total number of iteration and the total solution time increased dramatically for almost all of the test models. Using the time consuming exact initialization was much more promising. The total number of iterations could be reduced significantly in many cases. However, this saving did not compensate the additional effort for most of the test models. Therefore, our default is not to use a crash basis for the dual simplex in MOPS.

7.2 Computation of the pivot row

The computation of the pivot row α^r (step 7 of algorithm 7) is one of the most time consuming operations of the dual simplex method. For some models it can take more than 50% of the total solution time (cf. [11]) even for sophisticated implementations. Therefore, this step has to be implemented with great care.

To begin with we recall that the problem data complies with the internal model representation described in (2.3). From the definition of α^r we can directly conclude, that

$$\alpha_j^r = \rho_r^{j-\bar{n}} \quad \text{if } j > \bar{n}. \tag{7.1}$$

Thus no computation is necessary at all for columns corresponding to logical variables.

In principle, the structural part of α^r can be computed in two different ways. Using *columnwise* computation the entries α_j^r are computed independently by the inner product

$$\alpha_j^r = {\rho_r}^T a_j = \sum_{i \in \mathcal{I}(a_j)} \rho_r^i a_j^i, \tag{7.2}$$

where $\mathcal{I}(a_j)$ denotes the set of row indices of nonzero elements in column a_j. In equation (7.2) we can exploit sparsity of $A_\mathcal{N}$ by using the columnwise compact storage of $\bar{A}$, which is available anyway. Basic and fixed variables can easily be skipped. It is also easy to build up an index stack for those nonzero elements of α^r, which exceed the zero tolerance ϵ^z. The disadvantage of the columnwise computation is, that the sparsity of ρ_r cannot be exploited.

In the *rowwise* computation α^r is calculated as follows:

$$\alpha^r = A_\mathcal{N}^T \rho_r = \sum_{i \in \mathcal{I}(\rho_r)} \rho_r^i a_\mathcal{N}^i, \tag{7.3}$$

where $a_\mathcal{N}^i$ denotes the i-th row of $A_\mathcal{N}$. Sparsity of $A_\mathcal{N}$ can be exploited by using a rowwise compact storage of $\bar{A}$ to calculate the product $\rho_r^i a_\mathcal{N}^i$. This needs to be done only for nonzero positions of ρ_r, which can drastically reduce the computational effort, especially for hypersparse models.

However, the rowwise computation also has some disadvantages. The need for an additional rowwise datastructure for $\bar{A}$ significantly increases the memory requirements of the method. Furthermore, positions of α^r corresponding to basic or fixed variables cannot easily be skipped. In [11] Bixby recommends to maintain a rowwise data structure only for nonbasic nonfixed positions and update it in every basis change. Inserting a column into the data structure is usually very cheap, since the time required per nonzero element is constant. But for deleting a column this time is linear in the number of nonzeros of the affected row, so this operation can be quite expensive. In fact, in our tests we made bad experiences for models with many more columns than rows. On the other hand we could not see a significant speedup compared to the version without updating, so we decided to use a complete rowwise storage of $\bar{A}$ in our implementation. When the dual simplex method is used as a part of a branch-and-cut procedure for mixed-integer programming this data structure has to be derived anyway for other purposes. A further disadvantage of the rowwise computation is, that the build-up of an index stack requires an additional pass over the nonzero elements, since α^r has to be hold in a dense array to calculate the sum in equation (7.3). For these reasons, we use both the columnwise *and* the rowwise computation dependent on the density of ρ^r. If it exceeds 30% then the columnwise computation is used. For models with a many more columns than rows this threshold is lowered to 10%.

Finally we want to mention an interesting idea brought to our mind by P.Q. Pan in personal conversation, which could be named *partial ratio test*. In a first pass only those nonbasic nonfixed positions of the transformed pivot row are computed for which the corresponding reduced cost entry is zero. As soon as one of these positions defines a breakpoint (α^r_j nonzero and of right sign), it is taken as the entering variable. In this case the iteration must be degenerate and the further computation of the transformed pivot row as well as the dual ratio test and the update of the reduced cost vector can be skipped. Pan reports a significant improvement of his code by this idea. Up to now, we have not tested this technique, since it obviously collides with several other concepts in our code, as the bound flipping ratio test, the cost shifting scheme and the perturbation.

Part III

Implementation and results

Chapter 8

Implementation

8.1 The Mathematical OPtimization System MOPS

8.1.1 MOPS and its history

MOPS[1] is a high performance Mathematical Programming Software System for solving large-scale LP and mixed integer optimization problems (MIPs), which has chiefly been developed by U. Suhl since the year 1987. The system is written in FORTRAN77 and has been ported to various popular system platforms ranging from Windows PCs and servers to mainframes. It has been deployed in many practical applications for over two decades (see e.g. [69], [67], [62] and [37]) and has continuously been improved in algorithms, software design and implementation (see table 8.1). The system started as a pure LP-solver based on a high speed primal simplex algorithm. In its present form MOPS also features powerful dual simplex and interior point engines, as well as an effective branch-and-cut procedure. It belongs to the few competitive systems in the world to solve large-scale linear and mixed integer programming problems. The algorithms and computational techniques used in MOPS have been documented in several scientific publications

- on the primal simplex and system architecture [65, 66],
- on the LU-factorization [68] and LU-Update [63],
- LP-preprocessing [70, 49],
- super node processing [71],
- and on the dual simplex algorithm [38] and dual phase I [39].

The algorithmic improvements in the LP as well as in the IP part of MOPS can be seen by considering the solution times of the benchmark model oil, which is small by today's standard. The model has 5563 constraints, 6181 structural variables, 74 are 0-1 variables and 39597 nonzeros in the coefficient matrix. Tables 8.2 and 8.3 show the improvements for LP- and IP-optimization on oil, respectively.

Year	Version	Description
1987	1.0	primal simplex, LU-factorization and PFI-update
1988	1.1	LU-update of the basis factorization
1989	1.2	LP-preprocessing, update
1991	1.3	new pivot row selection minimizing the sum of infeasibilities
1992	1.4	new scaling, ftran, devex
1994	2.0	mixed 0-1-programming with super node processing
1995	2.5	mixed integer programming with general node selection
1997	3.0	first version of dual simplex algorithm for branch-and-bound phase
1998	3.5	improved super node processing
1999	4.0	additional interior point algorithm to solve initial LP
2001	5.0	new memory management, improved numerical kernels
2003	6.0	improved super node processing with cover lifting
2003	7.0	fixed charge and general bound reduction by solving LPs
2004	7.6	new dual simplex algorithm for initial LP and branch & bound
2005	7.9	improved super node processing with Gomory cuts

Table 8.1: History of MOPS development.

Year	Version	Hardware platform	Solution time (secs)
1991	1.4	I486 (25 MHz)	612.4
1995	2.5	P133 Win 3.11	20.7
1999	4.0	PIII (400 MHz), Win 98	5.1
2001	5.0	PIII (500 MHz), Win 98	3.9
2002	6.0	PIV (2,2 GHz), Win 2000	0.9
2005	7.9	PIV (3,0 GHz), Win 2000, primal	1.1
2005	7.9	PIV (3,0 GHz), Win 2000, dual	1.6
2005	7.9	PIV (3,0 GHz), Win 2000, IPM	0.6

Table 8.2: Improvement of MOPS LP optimization on model oil.

8.1.2 External system architecture

The external system architecture of MOPS is depicted in figure 8.1. All MOPS routines for model management and solution algorithms are part of a dynamic (mops.dll) and a static (mops.lib) link library, which can be integrated into user's applications. Using the static library, the data structures of the internal model representation (*IMR*, cf. model (2.3)) and the solution subroutines can directly be accessed via the IMR- and the C/FORTRAN interface. For the DLL a special set of interface functions is obtained.

For each system platform a MOPS executable (mops.exe) exists, which is controlled via text files. The problem data can be imported either in standard mps-format (cf. [50]) or a MOPS-specific triplet file format. Parameter settings are passed

[1]**M**athematical **OP**timization **S**ystem

Year	Version	Hardware platform	Solution time (secs)
1994	2.0	PII (500 MHz) LIFO-MIP	1794.3
1995	2.5	PII (500 MHz), general node selection	450.1
1999	4.0	PIV (2,2 GHz), IPM for initial LP	75.2
2003	6.3	PIV (2,2 GHz) various improvements	39.6
2005	7.9	PIV (3,0 GHz) Gomory cuts, dual in bb	12.9

Table 8.3: Improvement of MOPS IP optimization on model oil.

to the solver via a text based profile (mops.pro). Further files are used for solution data, statistics and the import and export of LP bases and B&B solution trees. Furthermore, a DLL-based MS Excel Add-In (ClipMOPS) can be used as a user friendly tool to edit and solve smaller LP and IP models.

8.1.3 LP / MIP solution framework

The internal solution architecture of MOPS for LP and MIP problems is displayed in figure 8.2. For LP problems it comprises two and for MIP problems three major phases. In the first phase, referred to as *data management phase*, memory is allocated and the problem data is read or generated and converted into IMR format. MOPS does its own memory management in the sense that no dynamic memory allocation is used. Instead a large block of memory (default 512MB, adjustable by user) is allocated prior to the model generation step. Then the arrays, which are needed for the problem data and the requested solution method(s) are accommodated within this block. Since the arrangement of the arrays is critical for a good cashing behavior, those arrays are grouped together, which are accessed simultaneously in the inner numerical kernels of the respective solution algorithm.

The *LP solution phase* starts with LP-preprocessing, which is run with slightly different default parameter settings dependent on the solution context. If the solution method is primal simplex or interior point (IPM) and the model to solve is a pure LP, then maximal preprocessing is used and reduced bounds are relaxed after preprocessing. As mentioned in section 7.1 it turned out to be crucial for the dual simplex method, that reduced bounds are kept after preprocessing. If the root LP relaxation of a MIP model is solved, a slightly restricted preprocessing is applied, since full preprocessing can lead to a less effective super node processing in the MIP solution phase. If primal or dual simplex are selected as solution algorithm, the problem matrix is scaled and a starting basis is constructed (default for the dual simplex is all-logical-basis). If the problem was perturbed the respective dual algorithm is used to remove the perturbation (DSX removes primal pertrubation, PSX removes dual perturbation). If the IPM solver is used and produces an optimal solution, it is usually not basic. In such a case, an equivalent optimal basic solution has to be identified by a simplex type *crossover* algorithm. The LP solution phase is completed by the LP postprocessing step, which maps the solution of the preprocessed model to the original model. If the model does not contain any integer variables, the solution process stops at this point.

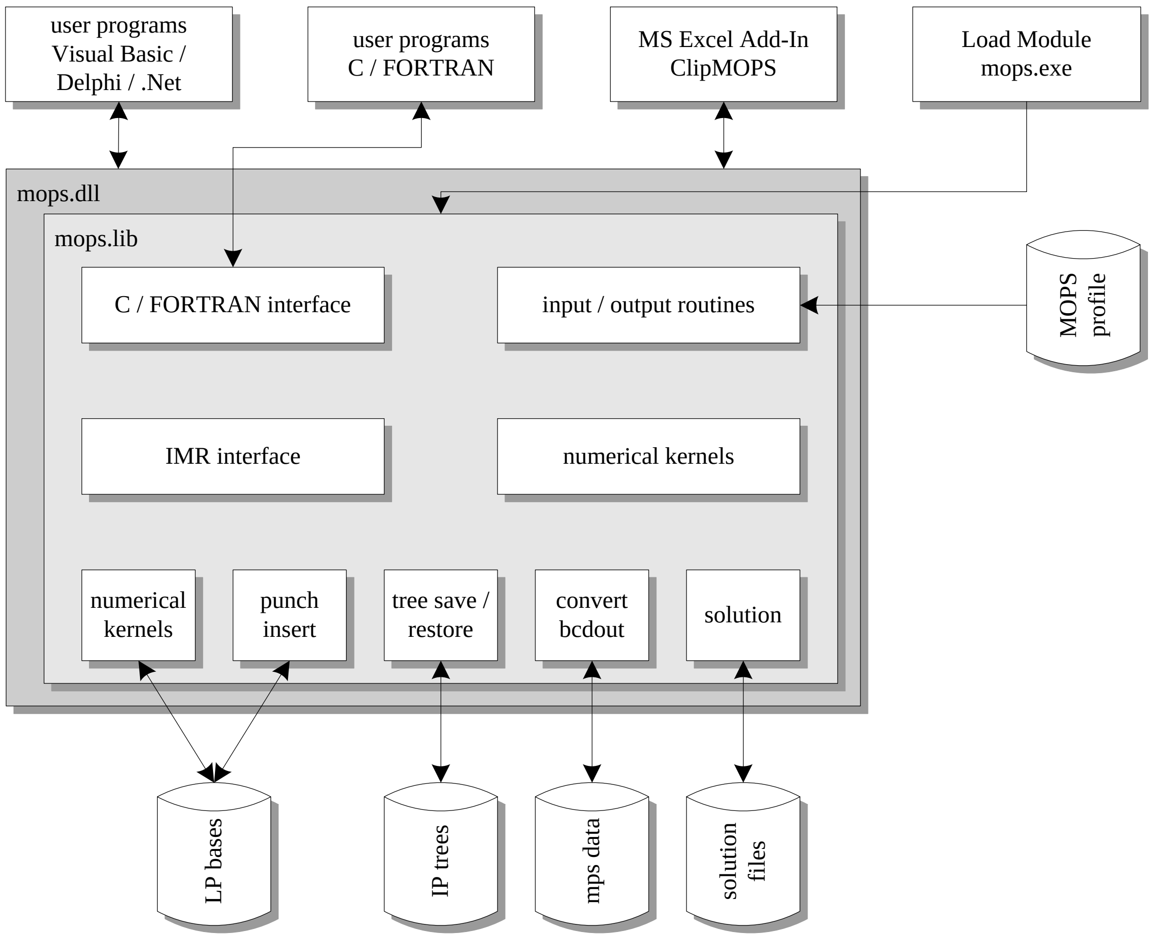


Figure 8.1: The external architecture of MOPS

If the model does contain integer variables, the *MIP solution phase* is entered. The first step is to apply *super node processing* techniques to the initial (MIP-)problem to tighten its LP relaxation. An important role plays the derivation of cutting planes, which are added to the model as additional constraints and as such increase the size of the basis. Usually a great set of possible cuts can be derived. The difficulty is to add only those cuts, which result in a considerable improvement of the LP relaxation and do not cause too much fill-in in the LU-factorization. After the root node has been processed, a primal MIP heuristic is applied to find a feasible MIP solution and an upper bound (for minimization) on the MIP objective function value. Then a branch-and-bound procedure is started, which by default uses the dual simplex method to reoptimize after branching. Dependent on its progress further nodes are chosen as super nodes and further cuts are generated.

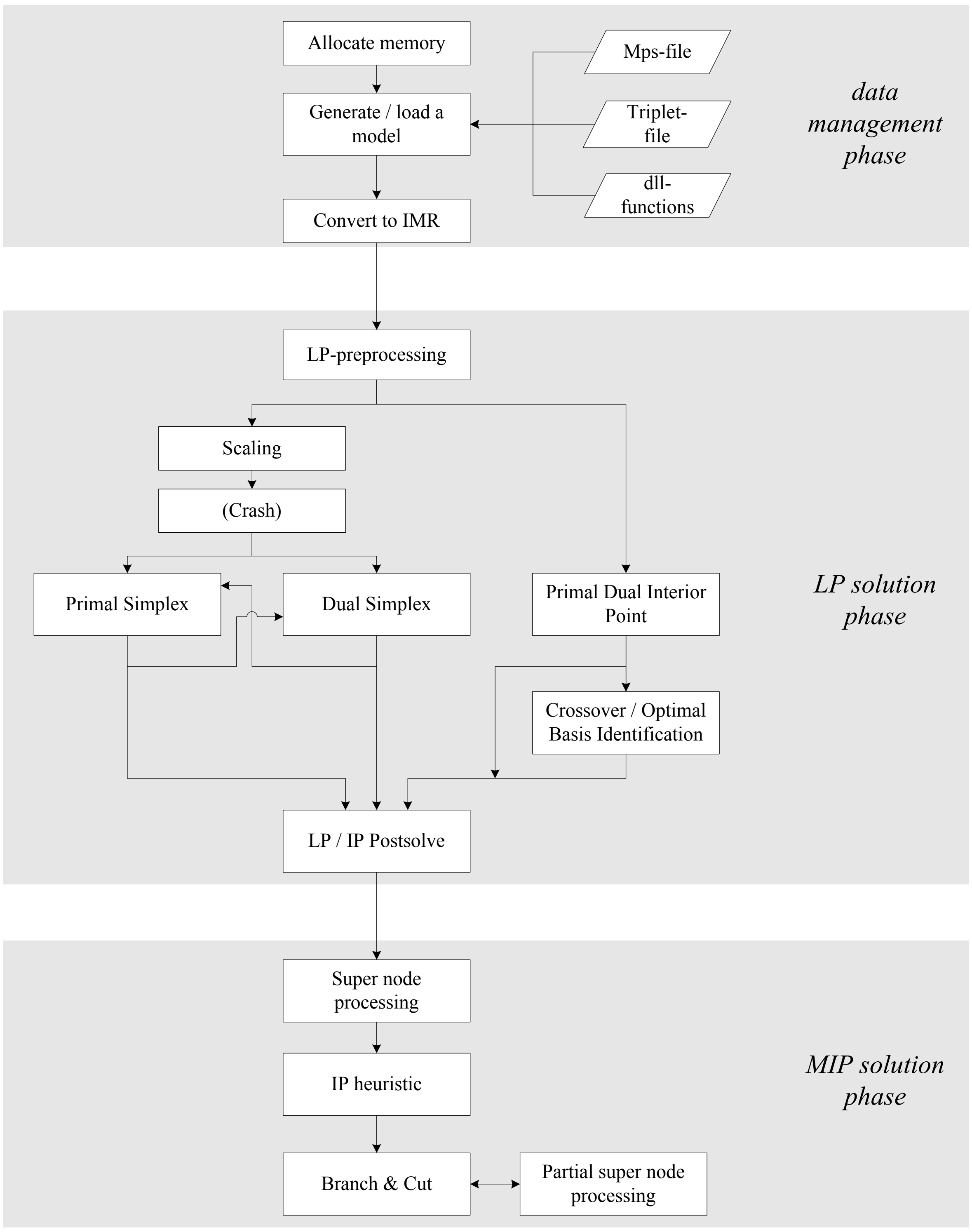


Figure 8.2: LP/IP solution framework in MOPS

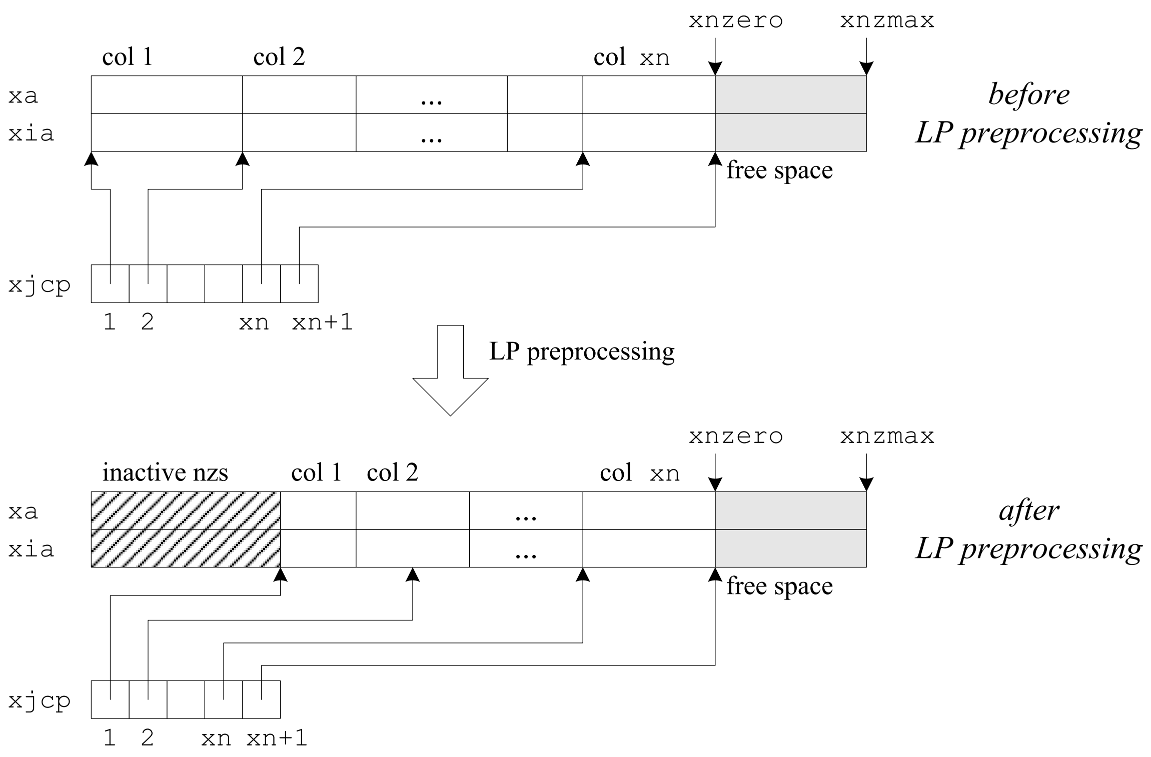


Figure 8.3: Columnwise compact storage for $\bar{A}$ before and after LP preprocessing.

8.2 The dual simplex code

8.2.1 Basic data structures

As mentioned before all data structures in MOPS are based on arrays, which are arranged within a large memory block b previous to the execution of a major solution routine. The array b (default size: 512MB) is allocated at the start of the system. In the following we will describe the data structures for problem data and, more importantly, the intermediate and final solution vectors, which are constantly accessed during the solution process.

Figure 8.3 shows the standard data structure, which is used to store the constraint matrix $\bar{A}$ of the IMR (2.3) in columnwise compact form. Only nonzero entries in the structural part are stored. Two arrays xa and xia are used to store the values and the row indices, respectively. The column pointer arrays xjcp contains the start indices of every column in xa and xia. In LP preprocessing many rows, column and nonzeros of the constraint matrix may be deleted. To allow for restoring the original model these entries are copied to the beginning of xa and xia and are excluded from further consideration by setting xjcp[1] to the first active entry in these arrays. Furthermore, the numbers of active rows xm, columns xn and variables xj = xm + xn are adjusted to their new, smaller values and the original values are saved.

In the initialization phase of the dual simplex routine an additional rowwise copy of $\bar{A}$ is generated following the same principles as above. Here only active parts of $\bar{A}$ are considered. The rowwise compact storage of $\bar{A}$ is needed for an efficient rowwise

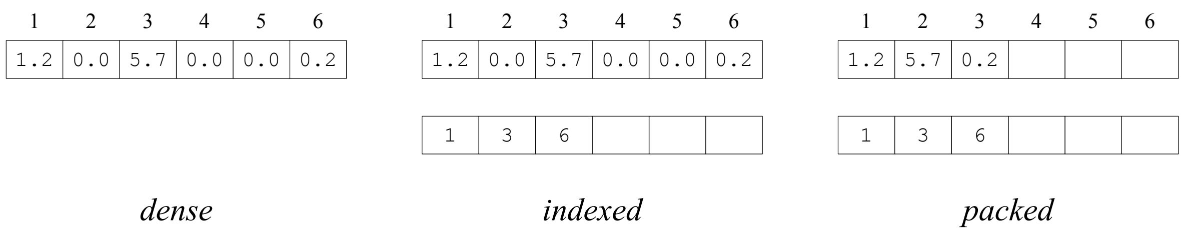


Figure 8.4: Dense, indexed and packed storage for mathematical vectors.

computation of the transformed pivot row α^r (cf. section 7.2).

Mathematical vectors can in principle be stored in three different ways (see figure 8.4):

- *Dense storage* means, that a simple array is used to store an exact image of the mathematical vector, including zero entries.
- *Indexed storage* uses a second array for the indices of the nonzero elements in addition to a dense array for zero and nonzero values.
- In *packed storage* only nonzero entries and their indices are stored, both in compact form.

The remaining data of the IMR (c, l and u) is stored in simple dense arrays xcost, xlb and xub. Although at the beginning cost coefficients of logical variables are zero, it is of size xj to handle cost shiftings and perturbations. At the start a copy of the cost vector is created, which is used at the end of the method to restore the original cost vector after potential cost modifications. The current primal and dual basic solution is stored in dense arrays xbxx (for $x_{\mathcal{B}}$), xx (for $x_{\mathcal{N}}$), xpi (for y) and xdjsc (for $d_{\mathcal{N}}$). In our version of the dual simplex method we do not update xpi. It is only used to recomputed the reduced cost vector after refactorization. The arrays xx and xdjsc are of size xj. Entries corresponding to basic positions are either zero or bear some undefined value. The nonbasic primal variables are updated mainly due to historic reasons, because the first dual phase I method based on artificial bounds (cf. section 4.3). For our version of the dual simplex method we actually need only the status of the nonbasic primal variables (wether at upper or at lower bound). In MOPS a large constant xinf $= 10^{20}$ is used for infinite bounds. The set of indices of basic variables (*basis heading*) is stored in the array xh. The order of the entries in xh also reflects the columnwise permutation of the basis matrix B in the context of the LU-factorization. The status of the primal variables is retained in a bitwise coded integer array xkey of size xj. It allows for efficient tests on wether a variable is basic, nonbasic at upper or at lower bound, wether it is a fixed or a free variable or if it was rejected as a potential leaving variable in a previous iteration.

The data structures for the transformed pivot row α^r, the transformed pivot column α_q and the r-th row of the basis inverse ρ_r are particularly important for the overall performance of the code. We use packed storage for the structural part of α^r and ρ_r (which is equivalent to the logical part of α^r). This allows for very fast loops during the rowwise computation of α^r, the dual ratio test and the update of xdjsc if α^r and ρ_r are sparse. This is especially the case for large hypersparse problems. While ρ_r is usually also sparse on typical medium sized and small problems, the structural part of α^r can get fairly dense. For these problems, where the density of α^r often exceeds 40% dense storage would pay off. To avoid further complication of the code, we decided to pass on this option for now.

The input and output vectors of our FTran and BTran routines are all in indexed storage. Consequently, we need a loop over the nonzero elements of ρ_r after the BTran operation in step 7 to create the packed format. However, we also keep the dense representation of ρ_r. This is needed as an input for the FTran operation in the context of the update of the DSE weights and might be needed for the columnwise computation of α^r. For the transformed pivot column α_q we use indexed storage, which is readily obtained by the FTran operation in step 7. We decide upon the density of α_q, wether we use the corresponding index array during the update of xbxx and the DSE weights, which are stored in a dense array xbeta. If it exceeds 30%, we use dense processing[2].

While the packed arrays are overwritten in every iteration, the dense arrays for ρ_r and α_q have to be zero at the beginning of every iteration. We use the index arrays to zero out the dense arrays after each iteration, which is crucial especially for hypersparse problems (the careful organization of this clearing operation alone saved about 15% of runtime on the large hypersparse NetLib model ken-18).

8.2.2 Pricing

As described in section 3.3, we use dual steepest edge pricing (DSE) based on the method called "Dual Algorithm I" by Forrest and Goldfarb [26] to select a variable to leave the basis. Three tasks have to be covered to implement this strategy:

1. (Re-)Initialization of the vector of DSE weights.

2. Update of the DSE weights.

3. Efficient selection of a leaving variable by passing through the weighted primal infeasibilities.

During the development process of our code we learned that the way each of these three tasks is implemented has a great impact on the performance of the code. In the following we will discuss the issues, that made a difference in our implementation.

8.2.2.1 Initialization and update of DSE weights

As mentioned before the DSE pricing strategy "Dual Algorithm I" can be viewed as a heuristic itself, since it does not guarantee that actually the leaving variable corresponding to the steepest edge is selected if boxed variables are present. Nevertheless, we learned from the work with our code that the DSE weights have to be

[2]This entails that the code for the respective update routines practically doubles: one version loops over the index array, the other version loops over xm.

initialized and updated with greatest possible accuracy. Even small errors can lead to a substantial increase of the iteration count.

The update is conducted according to equations (3.47a) and (3.50). At first we recompute the value of the weight $\beta_r = \rho_r^T \rho_r$ by its definition using the indexed storage of ρ_r. Since β_r is used in the update of all other weights, spending this extra effort is worthwhile to avoid accumulation of errors. In debugging mode we also test the numerical accuracy of β_r at this point by comparing the recomputed value with the updated version. To obtain the updated value $\bar{\beta}_r$ according to equation (3.47a) β_r has to be divided by the square of the pivot element α_q^r. In the context of the DSE update we use the FTran version of it instead of the BTran version, since it is generally of higher numerical precision (cf. section 6.2.3.2). Next we perform the FTran operation on ρ_r to obtain the vector τ according to equation (3.49) and start the update loop for the remaining weights. If α_q is sparse we only loop over its nonzero positions using the associated index vector.

In our very first DSE implementation we computed the update formula for the $\bar{\beta}_i$ exactly as it is stated in equation (3.50). We then tested the following reorganized form (it is easy to see, that it is equivalent to (3.50)):

$$\bar{\beta}_i = \beta_i + \alpha_q^i \left(\alpha_q^i \bar{\beta}_r + \kappa \tau_i \right) \quad \text{with } \kappa = \frac{-2}{\alpha_q^r}. \tag{8.1}$$

In both cases we avoid negative weights by setting $\bar{\beta}_i = \max\{\bar{\beta}_i, 10^{-4}\}$ as recommended in [26]. On most models (8.1) worked substantially better than the original version. The reason is clear: it avoids to square α_q^i right away but performs the division first. Therefore, smaller values are involved in the computations, which reduces the numerical error. We were surprised that the numerical accuracy of the DSE weights had such great impact on their efficacy.

This was confirmed by testing several variants of initializing the weights. Computing the weights from their definition for an arbitrary basis is very expensive (up to m BTran operations). For an all-logical basis we know that 1.0 is the correct initial value. But even if we start with an all logical basis, it may change substantially during the dual phase I, where we do not necessarily apply DSE pricing at all (e.g. in Pan's dual phase I, cf. section 4.5). Furthermore, due to the permutation of the basis after each refactorization, the vector of DSE weights becomes invalid. Due to these considerations we just reset the weights to 1.0 after every refactorization in an early version of our code. Later, we changed this simple strategy as follows:

- If no crash is used we initialize the weights with 1.0, otherwise (with crash) we do the expensive initialization from their definition (BTran is only necessary for rows, which are not unit vectors).
- Prior to each refactorization the weights are copied to a dense vector of length xj in such a way that weight β_i is assigned to the position corresponding to the index $\mathcal{B}(i)$ of the associated basic variable. After the factorization the weights are reassigned taking into account the new permutation of the basis.
- To have the correct values in dual phase II, the weights are always updated in

dual phase I, even if they are not used (e.g. in Pan's phase I algorithm).

- During the update of the weights we save the old values of the weights that are changed in a packed array. If an iteration is aborted after the weights have already been updated, we use this array to restore the original values. This situation can occur if the stability test in LU-update fails.

With these changes we achieved a great improvement compared to the simpler implementation. A crashed starting bases lead to a reduction of the total iteration count only if the DSE weights were correctly initialized. However, even then runtime did not improve in most of the cases, since the average time per iteration increased.

If the dual simplex method is deployed to reoptimize during branch-and-bound, there is no time to recompute the weights in every node of the B&B tree. Here, the default is to reuse the weights of the last LP-iteration.

8.2.2.2 Vector of primal infeasibilities

In the actual pricing step 7 we have to loop through the weighted primal infeasibilities to select the leaving variable. In the first version of our code we just implemented a simple loop over the primal basic variables following equation 3.53. Primal feasibility was tested within this loop.

Typically, 20–40% of the primal basic variables are infeasible at the start. For some problems this fraction is even lower. During the solution process the number of infeasibilities usually decreases continuously, until the basis is finally primal feasible. To exploit this observation, we changed our code to maintain an explicit indexed vector of the primal infeasibilities[3]. In the following xpifs denotes the dense array of the squared infeasibilities and xpifsi denotes the corresponding index array. This data structure is renewed after each recomputation (after each refactorization) and adjusted during the update of the primal basic solution $x_{\mathcal{B}}$. Four cases can occur in the update of a primal basic variable $x_{\mathcal{B}(i)}$:

1. $x_{\mathcal{B}(i)}$ stays primal feasible. The vector of primal infeasibilities does not change.

2. $x_{\mathcal{B}(i)}$ becomes primal infeasible. The squared infeasibility is recorded in xpifs and the index i is added to xpifsi, which length increases by one.

3. $x_{\mathcal{B}(i)}$ becomes primal feasible. The i-th entry of xpifs is replaced by a small constant 10^{-50} to signal feasibility. xpifsi is not changed.

4. $x_{\mathcal{B}(i)}$ stays primal infeasible. xpifs[i] is adjusted. xpifsi is not changed.

Typically, the number of indices stored in xpifsi increases moderately between two refactorizations and drops down to the actual number of infeasibilities after the next recomputation of $x_{\mathcal{B}}$. Using this vector of infeasibilities lead to great improvements of runtime especially on large models. But also on many smaller problems considerable savings could be achieved.

[3]Actually, the squared infeasibilities (distances to the respective bounds) are stored to further speed up the DSE pricing loop.

8.2.2.3 Partial randomized pricing

On large easy problems the computational effort spent in the pricing loop can be further reduced by considering only a small fraction ϕ of the infeasible positions. Only if all of these positions are actually feasible, further entries are examined to guarantee the selection of a leaving variable unless the basis is optimal. In each iteration ϕ is dynamically adjusted within an interval of $\frac{1}{20}$ to 1 dependent on the current relative iteration speed. As an estimate of the computational effort per iteration we take a ratio of the number of nonzero elements in the LU-factors and the number of rows xm. The default is $\phi = \frac{1}{8}$, which is decreased if the ratio drops below a value of 2.0 and increased if it exceeds a value of 10.0.

To give all primal infeasibilities the same probability to be considered we determine a random starting point for the pricing loop. On some problems this randomization of pricing also lead to fewer degenerate iteration. Furthermore, it reduces the risk of numerical cycling (cf. section 6.3.2).

8.2.3 Ratio test

Our implementation of the bound flipping ratio test (BFRT) is similar to the routine ClpSimplexDual::dualColumn(...) of the COIN LP Dual Simplex code [44]. We already gave an algorithmic description of this very sophisticated implementation in section 6.2.2.4 and algorithm 18. In this section we will discuss data structures and implementation details.

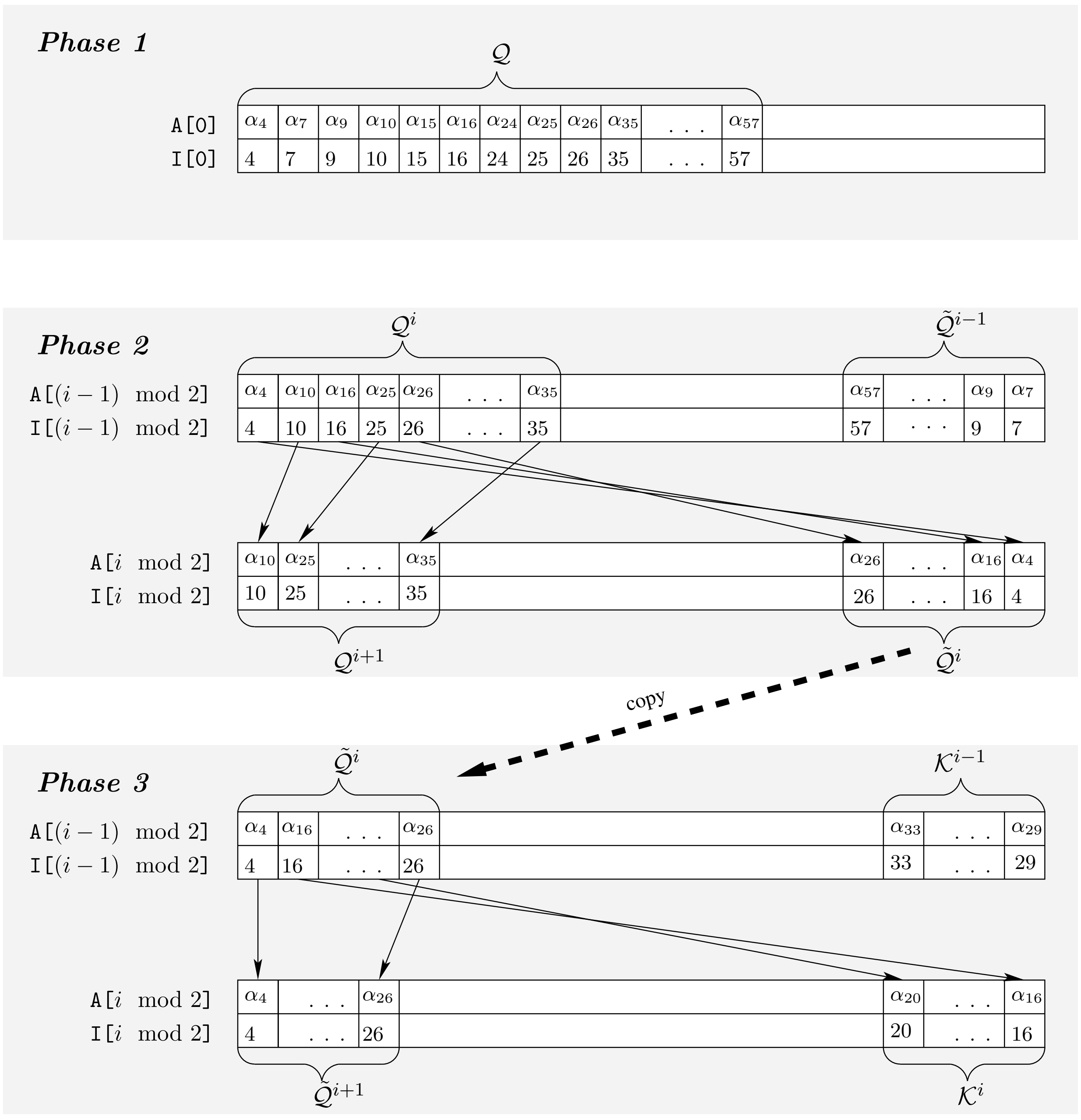
The crucial issue of the implementation of the BFRT is, how the set $\mathcal{Q}$ of candidate breakpoints is organized. Maros [47] and Fourer [27] recommend a heap data structure for partially sorting the breakpoints, since it provides the best theoretical complexity. The drawback of this approach is that a relatively expensive[4] build-heap operation has to be performed independently of the number of breakpoints, that can actually be passed.

Therefore, the COIN code follows a different path. After the set of breakpoints $\mathcal{Q}$ has been determined in a first pass over the transformed pivot row (*phase 1* in algorithm 18), this set is successively reduced to find a set of interesting breakpoints $\tilde{\mathcal{Q}}$ (*phase 2*). Only for these breakpoints the BFRT with Harris' tolerance is performed in *phase 3* as described in section 6.2.2.4.

The processing of the breakpoints in the three phases is depicted in figure 8.5. It is organized by means of two packed vector data structures of length xj with value arrays A[0] and A[1] and index arrays I[0] and I[1]. In phase 1, for every breakpoint in $\mathcal{Q}$ the corresponding pivot row entry[5] α_j^r and the column index j is stored in A[0] and I[0], respectively. These breakpoints constitute the set $\mathcal{Q}^1$ in phase 2. In the first round of phase 2, temporary reduced cost values are computed for positions in $\mathcal{Q}^1$ based on a first reasonable guess for the dual step length θ^D. If a position stays dual feasible for this θ^D, it joins the set $\mathcal{Q}^2$ (the set of remaining breakpoints), which is stored at the beginning of the second packed vector (A[1] and I[1]). If a position becomes dual infeasible for this θ^D, it joins the set $\tilde{\mathcal{Q}}^1$ (the set of possibly

[4]though the theoritical complexity bound is $\mathcal{O}(n)$

[5]actually $\tilde{\alpha}_j^r$



Phase 1
$\mathcal{Q}$
A[0]
I[0]
α_4 α_7 α_9 α_{10} α_{15} α_{16} α_{24} α_{25} α_{26} α_{35} . . . α_{57}
4 7 9 10 15 16 24 25 26 35 . . . 57
Phase 2
$\mathcal{Q}^i$
$\tilde{\mathcal{Q}}^{i-1}$
A[$(i-1)$ mod 2]
I[$(i-1)$ mod 2]
α_4 α_{10} α_{16} α_{25} α_{26} . . . α_{35} α_{57} . . . α_9 α_7
4 10 16 25 26 . . . 35 57 · · · 9 7
A[i mod 2]
I[i mod 2]
α_{10} α_{25} . . . α_{35} α_{26} . . . α_{16} α_4
10 25 . . . 35 26 . . . 16 4
$\mathcal{Q}^{i+1}$
$\tilde{\mathcal{Q}}^i$
copy
Phase 3
$\tilde{\mathcal{Q}}^i$
$\mathcal{K}^{i-1}$
A[$(i-1)$ mod 2]
I[$(i-1)$ mod 2]
α_4 α_{16} . . . α_{26} α_{33} . . . α_{29}
4 16 . . . 26 33 . . . 29
A[i mod 2]
I[i mod 2]
α_4 . . . α_{26} α_{20} . . . α_{16}
4 . . . 26 20 . . . 16
$\tilde{\mathcal{Q}}^{i+1}$
$\mathcal{K}^i$

Figure 8.5: Implementation of the bound flipping ratio test with Harris' tolerance.

passed breakpoints), which is stored at the end of the second packed vector. In the latter case also the sum of the slope changes $\tilde{\delta}$ is adjusted. If at the end of round 1 the sum of the slope changes $\tilde{\delta}$ lets the remaining slope δ drop below zero, phase 2 is terminated and $\tilde{\mathcal{Q}}^1$ is passed to phase 3. Otherwise, further bound flips are possible and the next round of phase 2 is started with an increased θ^D on the remaining breakpoints in $\mathcal{Q}^2$. This is done by simply flipping pointers of the two packed vectors, such that vector 1 takes the role of vector 0 and vice versa.

Figure 8.5 shows the situation in round i. If i is odd, the candidate breakpoints $\mathcal{Q}^i$ are situated at the head of the source vector 0 (A[0] and I[0]) and the sets of remaining and possibly flipped breakpoints are written to the head and the tail of the target vector 1, respectively. If i is even, the vectors are switched. Note, that at the end of round i we always have two sets of possibly flipped breakpoints stored, namely $\tilde{\mathcal{Q}}^i$ and $\tilde{\mathcal{Q}}^{i-1}$. If we proceed with the next round of phase 2, one of these sets is discarded (i.e., overwritten). All of the entries of both sets represent potential entering variables. Normally, we would discard $\tilde{\mathcal{Q}}^{i-1}$, since its entries are associated with a smaller θ^D and therefore lead to a lower progress in the dual objective function. Only if all of the α_j's in $\tilde{\mathcal{Q}}^i$ are two small to guarantee a save LU-update and considerably smaller than the greatest α_j in $\tilde{\mathcal{Q}}^{i-1}$, we will discard $\tilde{\mathcal{Q}}^i$ and replace it by $\tilde{\mathcal{Q}}^{i-1}$. Thereby a small pivot with great step length is avoided if a better sized pivot with smaller step length is available.

At the start of phase 3 the set of possibly flipped breakpoints $\tilde{\mathcal{Q}}^i$, which was determined in the last iteration i of phase 2, is copied from the end of the current target vector to the beginning of the current source vector. It must contain at least one breakpoint, which is associated with a positive slope and a well sized pivot element (if one exists). In every round i of phase 3 a Harris' threshold Θ_{max} is determined for the breakpoints in $\tilde{\mathcal{Q}}^i$. Those breakpoints, which meet or fall below Θ_{max} form the current candidate set $\mathcal{K}^i$. One of them is chosen to determine the tentative entering variable. The other, remaining breakpoints are saved in the set $\tilde{\mathcal{Q}}^{i+1}$. If the sum of the slope changes induced by the elements in $\tilde{\mathcal{Q}}^i$ does not result in a negative slope δ, the next round is started. Finally, at the end of phase 3, the shifting procedure described in section 6.2.2.3 is applied to the elements of $\mathcal{K}^i$ to assure dual feasibility.

8.2.4 FTran, BTran, LU-Update and factorization

In chapter 5 we presented mathematical techniques and algorithms to solve the up to four systems of linear equations in each iteration of the dual simplex method. In this section we will show the data structures, which we use to implement these methods. Furthermore, we will give a detailed description of the our FTran procedure with an emphasis on hypersparsity issues. Finally, we will discuss an alternative implementation based on the classical Forrest/Tomlin update, which has advantages on extremely hypersparse models.

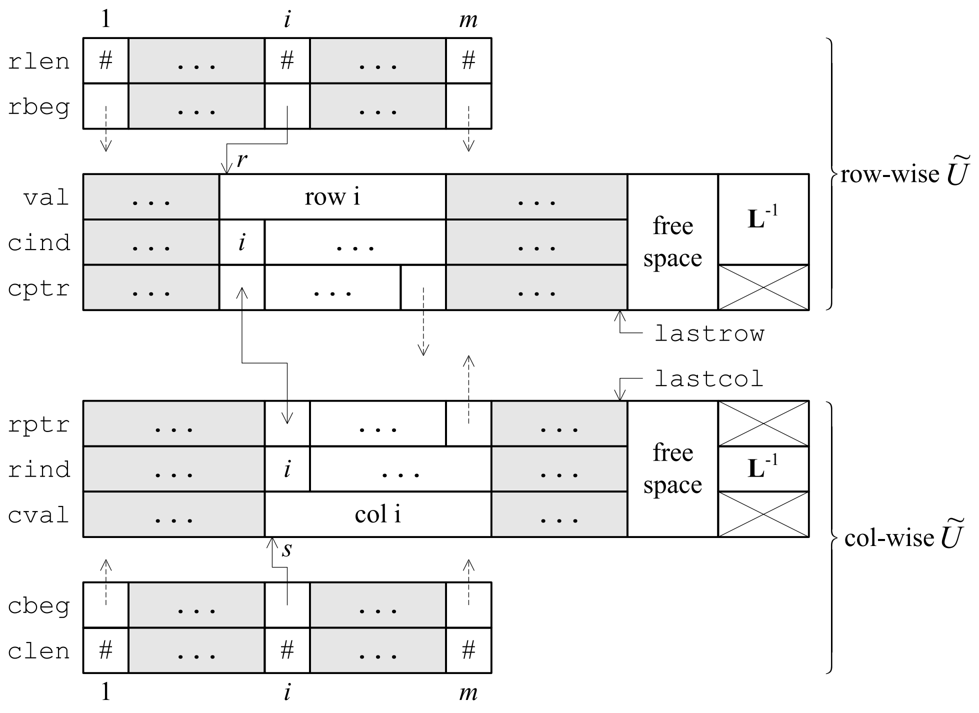


Figure 8.6: Data structure for $\tilde{U}$.

8.2.4.1 Data structures for the LU-factors

Since the basis for our solution routines is the LU-factorization of the basis matrix B given in equation 5.50, we need data structures for $\tilde{U}$, the eta-matrices $\tilde{L}^j$ and the permutation matrix P and its inverse. We store $\tilde{U}$ instead of U to be able to insert the spike $\bar{\alpha} = \tilde{L}^{-1}a_q$ without permutation.

The permuted upper triangular matrix $\tilde{\mathbf{U}}$ Similar to the data structure for $\bar{A}$ (cf. section 8.2.1) both a columnwise and rowwise representation of $\tilde{U}$ is stored in compact form. Figure 8.6 illustrates the involved arrays and index variables. The nonzero elements of a row and column are stored in the arrays val and cval, respectively[6]. The corresponding column- and row-indices are placed in the arrays cind and rind. Note, that these are the indices of $\tilde{U} = P^{-1}UP$ and not those of the upper triangular matrix U. Starting position and number of nonzeros of a row/column are saved in rbeg/cbeg and rlen/clen, respectively. Thus, row i of $\tilde{U}$ starts at position rbeg[i] and end at position rbeg[i] + rlen[i] − 1. Columns are accessed in the same manner. The index variables lastrow and lastcol point to the last used positions in the rowwise and columnwise representation, respectively.

The diagonal nonzero elements of $\tilde{U}$ are always stored at the starting position of the respective row and column. Consequently, the diagonal element of row i can simply be accessed by val[rbeg[i]], same for columns. Note, that due to the symmetric permutation the diagonal elements of U coincide with the diagonal elements of $\tilde{U}$

[6]We choose the identifier val instead of rval to make clear that this array is also used to store the eta-vectors (see next paragraph).

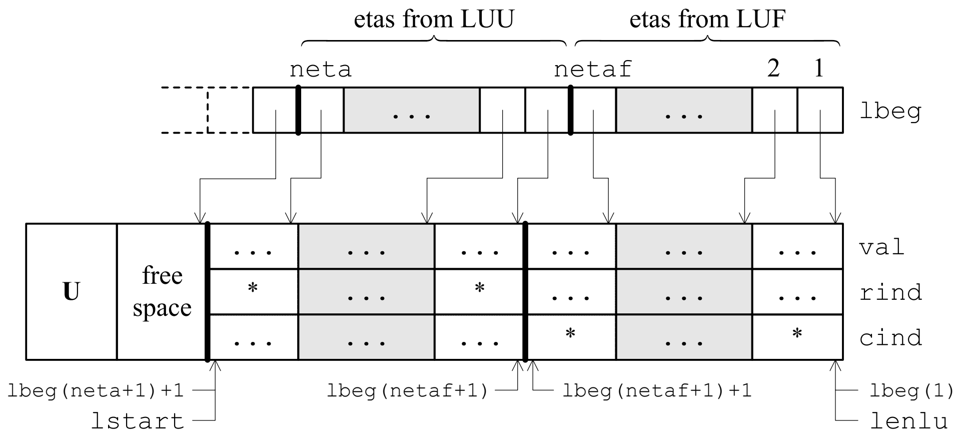


Figure 8.7: Data structure for the eta vectors of $\tilde{L}^{-1}$ (1).

(with changed positions). The order of the remaining elements is arbitrary.

Unlike the problem matrix $\bar{A}$ the matrix $\tilde{U}$ has to be updated in every iteration in the LU-update procedure. To allow for an efficient update of its column- and rowwise representation we deploy two further arrays cptr and rptr. For every nonzero element of the rowwise representation cptr contains the corresponding index in the columnwise representation, vice versa for rptr. Given these arrays it would actually suffice to store the nonzeros of $\tilde{U}$ only once – say rowwise – columnwise access would still be possible. However, this would result in a suboptimal usage of the processor's cache memory (cf. [68]). Therefore we keep holding two copies of the numerical values.

Columns corresponding to logical variables are always permuted to the beginning of $\tilde{U}$. They are not stored explicitly, only their number is kept in the variable nlogic.

The eta-vectors of $\tilde{\mathbf{L}}^{-1}$ For the eta-matrices $\tilde{L}^j$ we partly use the same arrays as for $\tilde{U}$. Only the nonzero elements of the nontrivial column/row of an eta-matrix are stored in the array val starting from its tail (see figure 8.7). The starting position of each eta-vector is noted in the array lbeg. The first netaf entries correspond to column-eta-vectors from the LU-factorization, followed by the row-eta-vectors generated in LU-update. The total number of eta-vectors is neta. The arrays rind and cind contain the associated row and column indices. Consequently, the areas in figure 8.7, which are marked with a *, contain the same entry for all elements of an eta-vector. In the variables lstart and lenlu we keep the first and the last position of an element of $\tilde{L}^{-1}$. nentl corresponds to the total number of nonzero elements of all eta-vectors.

For each eta-matrix $\tilde{L}^j$ ($1 \leq j \leq \texttt{netaf}$) from LU-factorization we also store the index of the nontrivial column in etacol[j]. The array coleta constitutes the inverse of etacol (see figure 8.8). If for a column p no eta vector exists, we set coleta[p] $= 0$.

The permutation As mentioned before we store $\tilde{U} = P^T U P$ and not U. To be able to reconstruct U (e.g. in FTranU and BTranU) the permutation has to be stored and updated explicitly. Since the rows and columns of U are permuted symmetrically,

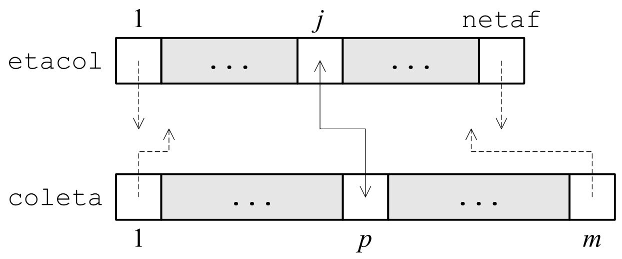


Figure 8.8: Data structure for the eta vectors of $\tilde{L}^{-1}$ (2).

only one single array perm is necessary. If perm$[i] = j$, then the i-th row/column of U corresponds to the j-th row/column of $\tilde{U}$. At some points of the code we also need the inverse of perm (e.g., to determine the last nonzero position in the spike $\bar{\alpha}$), which is stored in the array permback. Extending the example from above, we have permback$[j] = i$ meaning that the j-th row/column of $\tilde{U}$ corresponds to the i-th row/column of U.

8.2.4.2 Exploiting hypersparsity

In section 5.4.1 we described the two-phase method of Gilbert and Peierls [29] to solve hypersparse linear systems. Here, we will briefly discuss the data structures, which we use to implement the depth-first-search in the symbolic phase of the method, and how we switch between sparse and hypersparse solution methods.

Depth-first-search The depth-first-search (DFS) on the compact storage of $\tilde{U}$ (which corresponds to the graph $\mathcal{G}$ in section 5.4.1) is implemented iteratively rather than recursively. In algorithm 24 we give detailed pseudo code for the hypersparse FTranU operation. Figure 8.9 shows the involved data structures. The array work serves both as input and as output array for the numerical values of the right-hand side and the result vector. nzlist contains the nonzero positions of the input vector. Hence, both input and output vector are stored in indexed fashion. Four arrays of size xm are used to organize the DFS:

dfsstack acts as a stack for indices of the visited but not yet scanned nodes.

dfsnext contains for every node in dfsstack the index of the next not yet visited node among its adjacent nodes.

dfsmark is used to mark the visited nodes.

dfslist When a node is scanned and its index is deleted from dfsstack it is added to dfslist, which eventually contains a topological ordering of the nonzero positions of the result vector.

To achieve an optimal cashing behavior these four arrays are arranged in straight succession in the global memory block b. The variables nnzlist, ndfstack and ndfslist contain the number of entries in the corresponding arrays.

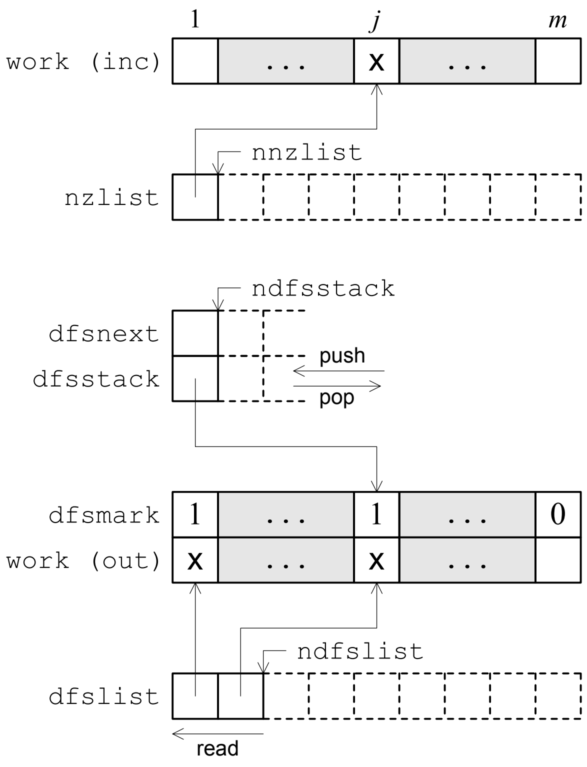


Figure 8.9: Data structure for depth-first-search.

Switching criteria According to our experience it is advisable to use the hypersparse version rather than the sparse version of a solution method if the density of the result vector of a linear system falls below 5%. For we do not know the number of nonzero entries in a result vector in advance, we use a simple estimate, which is based on the average number of resulting nonzeros in the same system over all previous iterations since the last refactorization. Algorithm 19 shows the principle implementation of this idea. In sum we sum up the ratios of the number of nonzeros of the result vectors by the number of input elements after numcalls number of calls of the solution method of a linear subsystem (say FTranU). Divided by numcalls we get an average value on how many output nonzeros were caused by one input nonzero. This is used to compute the predicted number of output nonzeros predic in the current linear subsystem. sum and numcalls are set to 0 after every refactorization.

Algorithm 19: HypersparsityTest

```
numcalls = numcalls + 1
numin = nnzlist
predic = numin * max(sum / numcalls, 1.0)
if (predic / xm) ≥ 0.05 then return false else return true
    ...
sum = sum + (nnzlistout / numin)
```

Note, that the statistic variables sum and numcalls have to be maintained inde-

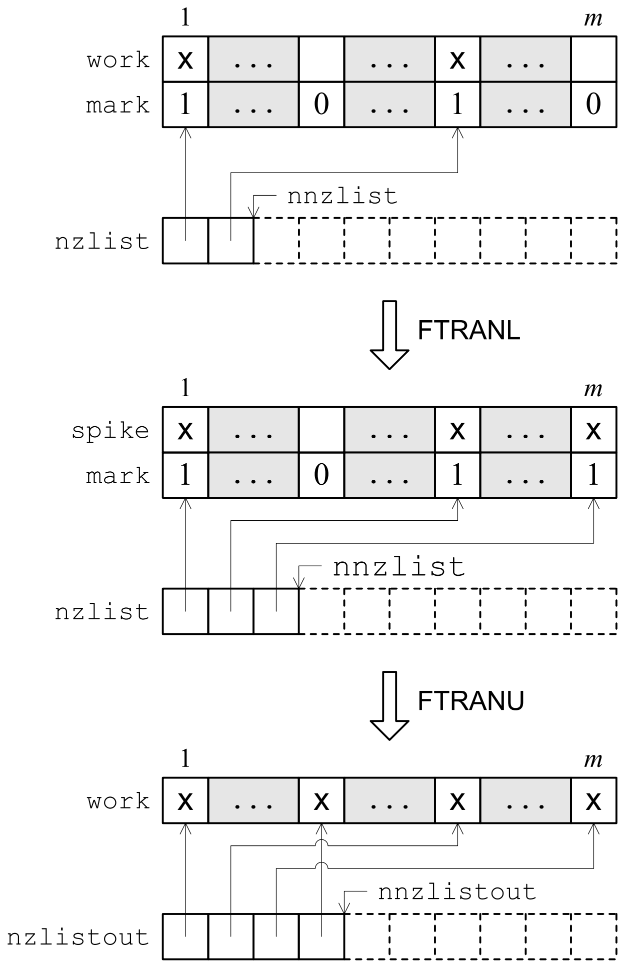


Figure 8.10: Data-flow of FTran operation.

pendently for every hypersparse solution procedure. In our code specially tailored versions are deployed in FTranL, FTranU, BTranU and LU-update.

8.2.4.3 Forward Transformation (FTran)

Three of the four linear systems, which have to be solved in each iteration of the elaborated dual simplex algorithm, require an FTran operation. Due to this great importance we will present detailed pseudo code of our FTran implementation in this section, which is based on the mathematical description of section 5.4.2.

Figure 8.10 shows the datastructures used to manage the input vector, the output vector and the spike $\bar{\alpha}$ in the course of the FTran operation. All of the three vectors are stored as indexed arrays, enhanced by an 0/1 vector mark, which is used during the computations to identify the indices of potentially nonzero positions and collect them in the respective index arrays (nzlist and nolistout). The floating point array work is initialized with the input vector and contains the result vector at the end of the operation.

Algorithm 20 serves as the driver routine of the FTran operation. Dependent on the density of the intermediate results it branches to the sparse or hypersparse version to solve the subsystems FTranL-F and FTranU (cf. equations 5.51). Furthermore it copies the spike $\bar{\alpha}$ after FTranL-U, which is only necessary in the FTran operation in step 7 of algorithm 7. For both of the other FTran calls (update of DSE weights and bound flipping ratio test), this part is skipped.

Algorithm 20: FTran (Pseudocode)

```
input  : work, mark, nzlist, nnzlist
output: spike, mark, nzlist, nnzlist, work, nzlistout, nnzlistout, last
1  if HypersparsityTest(FTranL-F) = false then
2      FTranL-F
3  else
4      FTranL-F_hyper
5  end
6  FTranL-U
   // copy work into spike
7  for k = 1 to nnzlist do spike(nzlist(k)) = work(nzlist(k))
8  if HypersparsityTest(FTranU) = false then
9      FTranU_sparse
10 else
11     FTranU_hyper
12 end
```

The FTranL-part of the FTran operation comprises the subroutines `FTranL-F_sparse` (algorithm 21), `FTranL-F_hyper` and `FTranL-U` (algorithm 22). We leave out the pseudo-code for `FTranL-F_hyper`, instead we give pseudo-code for `FTranU_hyper` (algorithm 24) as an example for the implementation of a hypersparse solution routine. In `FTranL-U` it is not possible to exploit (hyper-)sparsity, since the eta-vectors generated in LU-update are in rowwise format.

An important detail is the efficient construction of the index vector of nonzero elements nzlist. In our implementation of `FTranL-F` we we have to collect the nonzero indices in the innermost loop, which is quite expensive. The reason is, that the outer loop considers only positions, for which an eta-vector actually exists. Alternatively, we could run over xm in the outer loop, perform the **if**-block, if a corresponding eta-vector exists, and add nonzero indices at the end of that block (similar to `FTranU`). However, we opt for the first variant, since netaf is often much smaller that xm. In any case, the derivation of the index vectors causes a noticeable computational overhead, which on some small and dense problems is not necessarily compensated in other parts of the algorithm. However, they are indispensable for larger and sparser problems and for dynamic switching between sparse and hypersparse solution routines.

`FTranU_sparse` corresponds to the columnwise solution method described in algorithm 15. To exploit upper triangularity we run over the columns of U and find

Algorithm 21: FTranL-F_sparse

```
for j = 1 to netaf do
    p = etacol(j)
    if abs(work(p)) > zerotol then
        ap = work(p)
        ks = lbeg(j + 1) + 1; ke = lbeg(j)
        for k = ks to ke do
            i = rind(k)
            work(i) = work(i) + val(k) * ap
            if mark(i) = 0 then
                // put in result-list
                mark(i) = 1; nnzlist = nnzlist + 1; nzlist(nnzlist) = i
                if permback(i) ≥ last then last = permback(i)
            end
        end
    end
end
```

Algorithm 22: FTranL-U

```
for k = lbeg(netaf + 1) to lbeg(neta + 1) + 1 step -1 do
    j = cind(k)
    if abs(work(j)) > zerotol then
        p = rind(k)
        work(p) = work(p) + val(k) * work(j)
        // put in result-list
        if mark(p) = 0 then
            // put in result-list
            mark(p) = 1; nnzlist = nnzlist + 1; nzlist(nnzlist) = i
            if permback(p) ≥ last then last = permback(p)
        end
    end
end
```

Algorithm 23: FTranU_sparse()

```
nnzlistout = 0
for k = last to nlogic + 1 step -1 do
    j = perm(k)
    if abs(work(j)) > zerotol then
        ks = cbeg(j); ke = ks + clen(j) − 1
        aj = work(j)/cval(ks); work(j) = aj
        for kk = ks + 1 to ke do
            i = rind(kk)
            work(i) = work(i) − cval(kk) ∗ aj
        end
        // put in result-list
        nnzlistout = nnzlistout + 1; nzlistout(nnzlistout) = j
    end
end
for k = nlogic to 1 step -1 do
    j = perm(k)
    if abs(work(j)) > zerotol then
        // put in result-list
        nnzlistout = nnzlistout + 1; nzlistout(nnzlistout) = j
    end
end
```

Algorithm 24: FTranU_hyper

```
FTranU_hyper_DFS
for k = ndfslist to 1 step -1 do
    j = dfslist(k)
    dfsmark(j) = 0
    see lines 4 - 12 from algorithm FTranU_sparse
end
```

Algorithm 25: FTranU_hyper_DFS

```
// depth-first search:  create list of nz positions in result
ndfsstack = 0; ndfslist = 0
for k = 1 to nnzlist do
    j = nzlist(k)
    if dfsmark(j) = 0 then
        if clen(j) > 1 then
            // put on stack
            ndfsstack = 1; dfsstack(1) = j; dfsnext(1) = cbeg(j) + 1
        else
            // no more nz in this column, mark and put in list
            ndfslist = ndfslist + 1; dfslist(ndfslist) = j; dfsmark(j) = 1
        end
    end
    while ndfsstack > 0 do
        finish = true
        j = dfsstack(ndfsstack)
        ks = dfsnext(ndfsstack); ke = cbeg(j) + clen(j) − 1
        for kk = ks to ke do
            i = rind(kk)
            if dfsmark(i) = 0 then
                if clen(i) > 1 then
                    // put on stack
                    dfsnext(ndfsstack) = kk + 1; ndfsstack = ndfsstack + 1
                    dfsstack(ndfsstack) = i; dfsnext(ndfsstack) = cbeg(i) + 1
                    finish = false
                    exit for
                else
                    // no more nz in this column, mark and put in list
                    ndfslist = ndfslist + 1; dfslist(ndfslist) = i; dfsmark(i) = 1
                end
            end
        end
        if finish = true then
            // node scanned, take from stack, mark and put in list
            ndfsstack = ndfsstack − 1
            ndfslist = ndfslist + 1; dfslist(ndfslist) = j; dfsmark(j) = 1
        end
    end
end
```

the corresponding column indices in $\tilde{U}$ via the permutation array perm. Two details further improve the performance in `FTranU_sparse`:

- We start iterating at the column index last instead of xm in U, which is the last nonzero element in the input vector. last has to be determined anyway in FTranL, since it is needed in LU-update as the last nonzero position l in the non-permuted spike $\bar{\alpha}'$ (cf. section 5.3.2).
- As we mentioned before columns corresponding to logical variables are arranged at the beginning of U. Since logical columns do not change the result, we stop iterating at position nlogic+1. However, we have to keep going through the logical columns in a second loop to complete the vector nzlistout of nonzero indices.

The hypersparse version `FTranU_hyper` basically coincides with `FTranU_sparse` in the numerical phase. However, it iterates only over the nonzero positions of the result vector, which have been determined previously in `FTranU_hyper_DFS`. Since the nonzero positions are always appended at the end of dfslist during the DFS, it has to be traversed in reverse order to form a valid topological order (cf. section 5.4.1).

We already presented the data structures used in `FTranU_hyper_DFS` in section 8.2.4.2. In the outer loop (line 2) a depth-first-search is started for every nonzero element of the input vector work. If the current index j is not yet scanned (line 4) and the respective column of $\tilde{U}$ contains further off-diagonal elements (which correspond to adjacent edges of the node j in the graph $\mathcal{G}$), we add j to the stack dfsstack (line 6) as the starting point of the DFS. Otherwise we mark j as scanned and add it to the list of nonzero positions dfslist (line 8). The DFS is started in line 11. Every time a node is scanned the corresponding index is added to the nonzero vector (lines 28 to 31). Finally, dfslist contains the nonzero positions of the result vector in reversed topological order.

8.2.4.4 LU-update and factorization

Since our implementation of the dual simplex method is based on the existing code of the MOPS system, the routines for LU-factorization and update were already available. Their implementation is described in [68] and [63]. While we left the factorization routine unchanged, we extended the LU-update procedure to exploit hypersparsity. As indicated in section 5.3.2 the computation of the entries of a row-eta-matrix $\tilde{L}^j$ in lines 9 to 22 of algorithm 13 is equivalent to a reduced BTranU operation. Therefore, we apply the same techniques to exploit hypersparsity as discussed in the previous sections.

In our tests we learned, that the update of the permutation in the Suhl/Suhl update (lines 23 to 27 of algorithm 22) can become a relatively time consuming operation on very hypersparse models (like ken-18 from the NetLib [5] test set). In our code we split this operation in two separate loops, one for the update operation for perm (corresponding to P^{-1}) in line 26 and one for permback (corresponding to P) in line 25. Due to better data locality the loop for perm can be executed much faster than that for permback. Alternatively, one could use a quite sophisticated implementation of the classical Forrest/Tomlin update for this type of problems, which we found in the COIN LP code [44]. Its description is beyond the scope of this thesis. There, the update of the permutation vectors can be realized as a constant time operation. However, it is well known that the Forrest/Tomlin is generally inferior to the Suhl/Suhl update since it produces more fill-in. Therefore, we to stick to the latter. It is an open question to us, wether a constant time update of the perturbation can also be achieved for the Suhl/Suhl update.

8.2.5 Overview

The figures 8.11 and 8.12 give an overview of the main steps of our implementation of the dual simplex method. The first of them comprises initialization, refactorization and dual phase I while the latter shows the flow of the main loop. Note that dependent on which type of method is used (cf.chapter 4) the dual phase I can be a whole algorithm by itself (e.g. Pan's method).

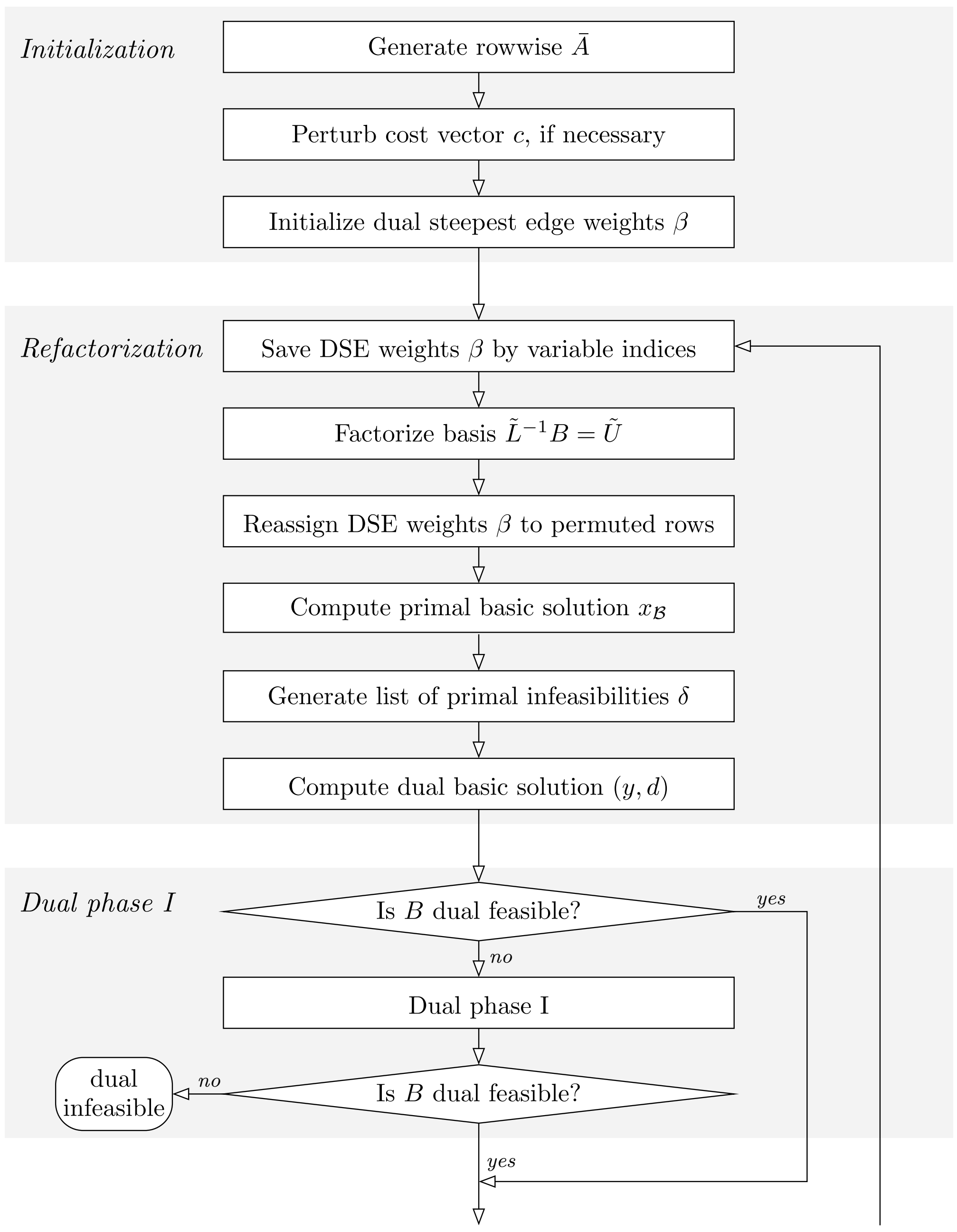

Initialization
Generate rowwise $\bar{A}$
Perturb cost vector c, if necessary
Initialize dual steepest edge weights β
Refactorization
Save DSE weights β by variable indices
Factorize basis $\tilde{L}^{-1}B = \tilde{U}$
Reassign DSE weights β to permuted rows
Compute primal basic solution $x_{\mathcal{B}}$
Generate list of primal infeasibilities δ
Compute dual basic solution (y, d)
Dual phase I
Is B dual feasible?
yes
no
Dual phase I
dual infeasible
no
Is B dual feasible?
yes

Figure 8.11: Implementation overview: initialization, refactorization and dual phase I.

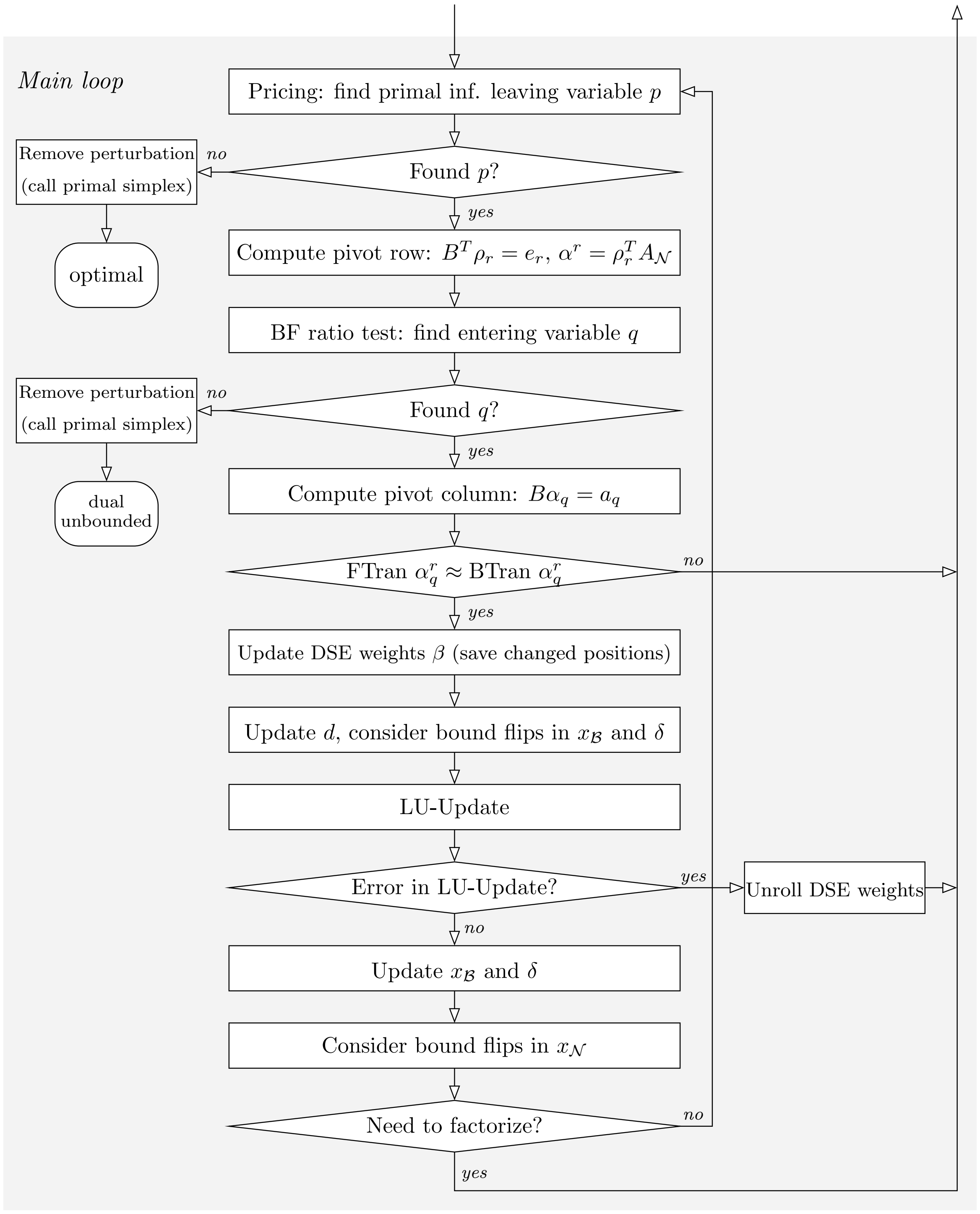

Main loop
Pricing: find primal inf. leaving variable p
Found p?
no
Remove perturbation (call primal simplex)
optimal
yes
Compute pivot row: $B^T \rho_r = e_r$, $\alpha^r = \rho_r^T A_{\mathcal{N}}$
BF ratio test: find entering variable q
Found q?
no
Remove perturbation (call primal simplex)
dual unbounded
yes
Compute pivot column: $B\alpha_q = a_q$
FTran $\alpha_q^r \approx$ BTran α_q^r
no
yes
Update DSE weights β (save changed positions)
Update d, consider bound flips in $x_{\mathcal{B}}$ and δ
LU-Update
Error in LU-Update?
yes
Unroll DSE weights
no
Update $x_{\mathcal{B}}$ and δ
Consider bound flips in $x_{\mathcal{N}}$
Need to factorize?
no
yes

Figure 8.12: Implementation overview: main loop.

Chapter 9

Numerical results

In this chapter we present computational results produced by our dual simplex code. In sections 9.1 and 9.2 we describe the composition of our set of test problems and the performance measures used in our analysis. We then present a study on the dual phase 1 in section 9.3, where we evaluate the performance of the different dual phase 1 methods described in chapter 4. In section 9.4 we document the continuous progress during the development of our code and point out again the crucial implementation techniques. Finally, we benchmark the performance of our dual simplex code compared to other renowned LP-systems in section 9.5.

9.1 Test problems

Table A.1 specifies names and dimensions of a set of one hundred test problems, which constitute the basis for the computational results of this chapter. The problems are taken from five different sources (column *Source*):

n NetLib and Kennington test set [5], 17 problems.

m MipLib2003 test set [3], 17 problems[1].

t Mittelmann test set [4], 21 problems.

c BPMPD test set [1], 20 problems.

o Other. 25 problems from our private test problem collection [2]. The problems HAL_M_D, MUN1_M_D, MUN18_M_D and PTV15 are multi-depot bus scheduling models described in [37].

In addition, we generated a second, preprocessed version of each problem instance by means of the COIN LP code[2][44]. These pretreated instances will be used in section 9.5 to benchmark the dual simplex implementations of the different LP-systems independently of their respective LP-preprocessing. For both versions we give the model dimensions in terms of the number of structural variables (column *Structs*), the number of constraints (column *Constrs*) and the number of nonzero entries in the constraint matrix (column *Nzs*).

[1]We only solve the LP-relaxation of these problems.

[2]At the beginning of the method ClpSimplexDual::dual(...) we write out the preprocessed model in MPS format (see [50] for a description) using the method writeMps(...).

9.2 Performance measures

The interpretation and analysis of data generated in a benchmarking process is not a trivial task. The often used performance measures arithmetic mean (average) or cumulative total (sum) have the significant drawback that a small number of the most difficult problems tend to dominate the results. The geometric mean seems to be more appropriate since it tends to represent the behavior on the majority of test instances. The disadvantages of this measure are however, that it does not have an intuitive meaning and that large deviations on few models are not represented adequately. All of these "numerical" measures have the problem that they cannot deal with failures in a sound way.

For these reasons we use the visualization technique of *performance profiles* (see [22]) in addition to sum and geometric mean. Let $\mathcal{S}$ be the set of algorithms/solvers, which are to be compared, and $\mathcal{P}$ the set of test instances. Since our main criterion for comparison is CPU solution time, we define

$$t_{p,s} = \text{CPU time required to solve problem } p \in \mathcal{P} \text{ by solver } s \in \mathcal{S}.$$

The *performance ratio*

$$r_{p,s} = \frac{t_{p,a}}{\min\{t_{p,s'} : s' \in \mathcal{S}\}} \tag{9.1}$$

gives the performance of solver s on problem p compared to the best performance by any solver in $\mathcal{S}$ on this problem. A parameter $r_M \geq r_{p,s}$ for all p, s is chosen and $r_{p,s}$ is set to r_M if and only if solver s fails to solve problem p.

Then, the *performance profile* of solver s is given by the function

$$\tau \mapsto \frac{1}{n_{\mathcal{P}}}|\{p \in \mathcal{P} : r_{p,s} \leq \tau\}|, \tag{9.2}$$

where $n_{\mathcal{P}} = |\mathcal{P}|$. It represents the fraction of problems, for which solver s has a performance ratio of at most $\tau \in \mathbb{R}$. The right hand side of equation 9.2 can be written as

$$P(r_{p,s} \leq \tau : 1 \leq s \leq n_{\mathcal{S}}), \tag{9.3}$$

since it is equivalent to the probability for solver s, that a performance ratio $r_{p,s}$ is within a factor τ of the best possible ratio. In our diagrams we often use a logarithmic scale for τ.

9.3 Study on dual phase 1

From the dual phase 1 methods presented in chapter 4 we implemented the two approaches of minimizing the sum of the dual infeasibilities (subproblem (SP) and algorithmic approach, algorithms 8 and 9, respectively), the method by cost modification (CM) (section 4.4) and Pan's method (algorithm 11).

After we implemented the subproblem approach and saw its mathematical equivalence and convincing performance we did not proceed to advance the code of the algorithmic approach. Since the latter did not benefit from the improvements in our

	Pan + SP approach	SP approach	Cost Mod.
Sum CPU Time (secs)	8521.1	9254.9	11625.8
Geom. Mean CPU Time	30.1	31.3	32.7
Sum Total Iters	2093129	2160389	2183591
Geom. Mean Total Iters	18602.6	18910.7	19729.6
Sum Degen. Iters	260277	308879	649566
Sum Dual Phase 1 Iters	432107	461127	–
Geom. Mean Dual Phase 1 Iters	164.9	195.1	–
Sum Dual Simplex Iters	2079179	2146392	1676332
Sum Primal Simplex Iters	13950	13997	507259

Table 9.1: Benchmark of dual phase 1 methods.

dual phase 2 code, it is not competitive anymore and we will not present numerical results for it.

As mentioned in section 4.5 Pan's method does not provide a theoretical convergence guarantee, which turned out to pose a problem on a handful of mostly numerically difficult problems. Therefore we provided our Pan code with a simple checking rule, which switches to the subproblem phase 1, if the number of dual infeasibilities does not decrease adequately. This variant will be denoted by *Pan+SP*.

We also tried two versions of the *CM* method: in the first version we restored the original cost coefficients after each refactorization, if the corresponding reduced cost values stay dual feasible, in the second version we do not touch the modified cost vector until the end of the dual phase 2. Here, we will present results only for the first variant, since it worked clearly better (as expected) than the second one.

Tables A.7 to A.10 show the detailed benchmarking results for the four different methods on those 46 problems of our test set, for which the starting (all-logical) bases is dual infeasible and cannot be made dual feasible by pure feasibility correction (cf. section 4.1.2). These test runs were conducted under Windows XP Professional on a standard Intel Pentium IV PC with 3,2 GHz and 1GB of main memory. Our code was compiled with Compaq Visual Fortran Compiler V6.6. For each method we give the total CPU solution time in seconds (including time for LP-preprocessing), total number of iterations, number of degenerate iterations[3], number of dual phase 1 iterations (except for the CM method) and the number of iterations spent in the dual and the primal simplex[4], respectively. Table 9.1 summarizes the results by listing sums and geometric means (where appropriate) for runtime and iteration counts. Pan's method is not included since it failed on three problems to achieve a dual feasible basis. Figure 9.1 shows a performance profile over runtime.

Each of the four method performs quite well on about two thirds of the test instances compared to the respective best method, with a slight edge for *Pan+SP*. On roughly 20% of the problems the original *Pan* method and the *CM* method show significant difficulties. For the *CM* method these are mainly those problems, which

[3]We consider an iteration as degenerate, if $\theta^D < \epsilon^D$, where θ^D denotes the dual step length and ϵ^D denotes the dual feasibility tolerance.

[4]The primal simplex is generally used at the end of the dual phase 2 to remove dual infeasibilities caused by restoring the original cost vector after cost perturbation and shiftings.

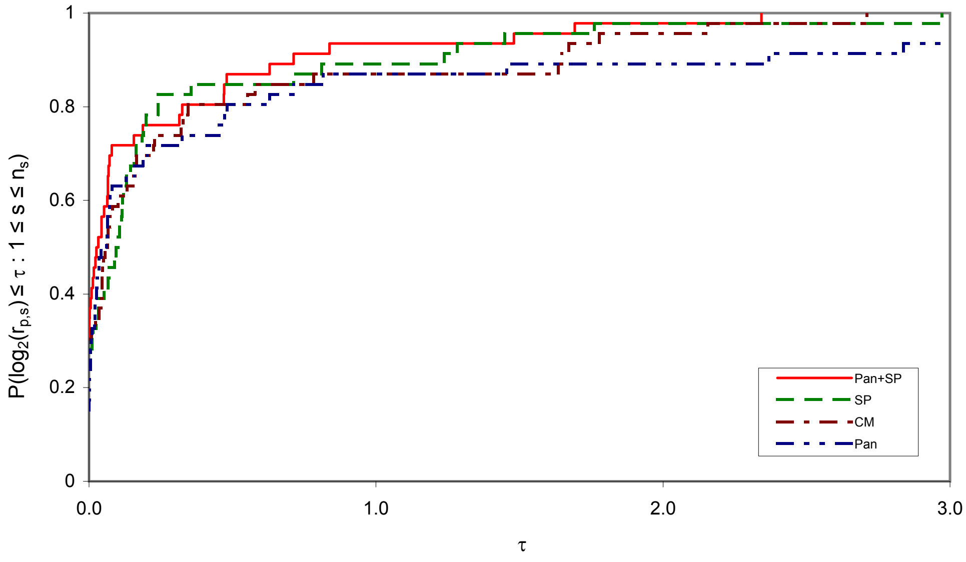


Figure 9.1: Performance profile over phase 1 test set: solution time using four different dual phase 1 methods.

need a large proportion of primal simplex[5] iterations (WATSON_1, WATSON_2, DEGEN4, ULEVIMIN). An exception is the instance DBIC1, on which the primal simplex seems to be superior to the dual simplex. It can be seen as a fundamental drawback of this method, that it is basically unpredictable how many iterations are performed by the dual simplex and how many are performed by the primal simplex method for a given problem. For our test set, almost 25% of the iterations were in primal simplex (cf. table 9.1). This makes it almost impossible for the user to choose the right solution algorithm for his LP model. Furthermore, this method has an inherent tendency to increase the number of degenerate iterations. *Pan* has problems on numerically difficult instances like P13, P14, P19 and P20. On these problems, the *SP* method works significantly better. To summarize we can say, that the combined method *Pan+SP* has the best overall performance, with a slight edge compared to *SP*.

In a second experiment we investigate the impact of the treatment of the primal bounds after LP preprocessing (cf. section 7.1). Two variants are compared on the complete problem test set: the first one keeps the reduced bounds, the second expands the bounds after LP preprocessing. As above the test runs were conducted under Windows XP Professional on a standard Intel Pentium IV PC with 3,2 GHz and 1GB of main memory, but here we use an executable generated by the Intel Visual Fortran Compiler V9.0, which turned out to be superior to the Compaq version. The detailed benchmarking data is given in tables A.5 (*Original Models*) and A.6 and summarized in table 9.2 and the performance profile in figure 9.2.

[5]The primal simplex code in MOPS is inferior to the dual simplex code in particular on large problems, since it still lacks some important implementation techniques (e.g. hypersparsity). But even with an improved primal code the dual simplex is generally seen as superior to the primal simplex (cf. [10]).

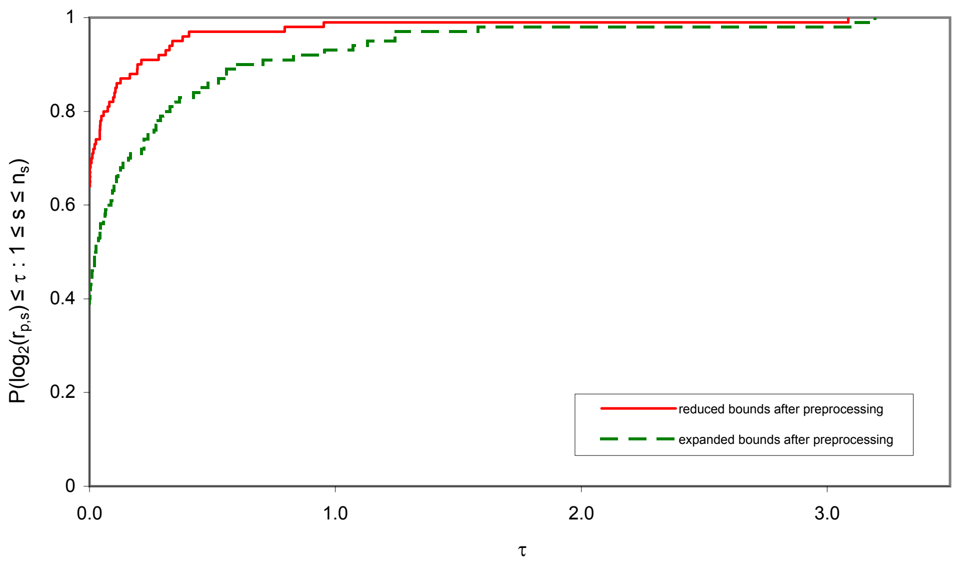


Figure 9.2: Performance profile over all test models: solution time with reduced and expanded bounds after LP preprocessing.

	Reduced Bounds	Expanded Bounds
Sum CPU Time (secs)	38816.5	39643.8
Geom. Mean CPU Time	23.0	25.6
Sum Total Iters	3816302	3971111
Sum Degen. Iters	446812	460097
Sum Dual Phase 1 Iters	433351	497465
Number of Problems With Dual Phase 1	46	52

Table 9.2: Benchmark with reduced and expanded bounds after LP-preprocessing on original test set.

The performance profile shows, that keeping the reduced bounds after LP preprocessing clearly improves the overall performance of the dual simplex method. The number of dual phase 1 iterations is significantly reduced for many models, for some of the models the execution of a dual phase 1 method even becomes superfluous. This is not surprising, since additional finite bounds tend to increase the number of boxed variables, which are made dual feasible by feasibility correction. Furthermore, the number of degenerate iterations decreases compared to the variant with expanded bounds. Tighter bounds probably increase the impact of the bound flipping ratio test, which can be seen as a anti-degeneracy technique.

9.4 Chronological progress study

In this section we present computational results produced by twelve different versions of our code (see table 9.3), which reflect the progress of a two-years development pro-

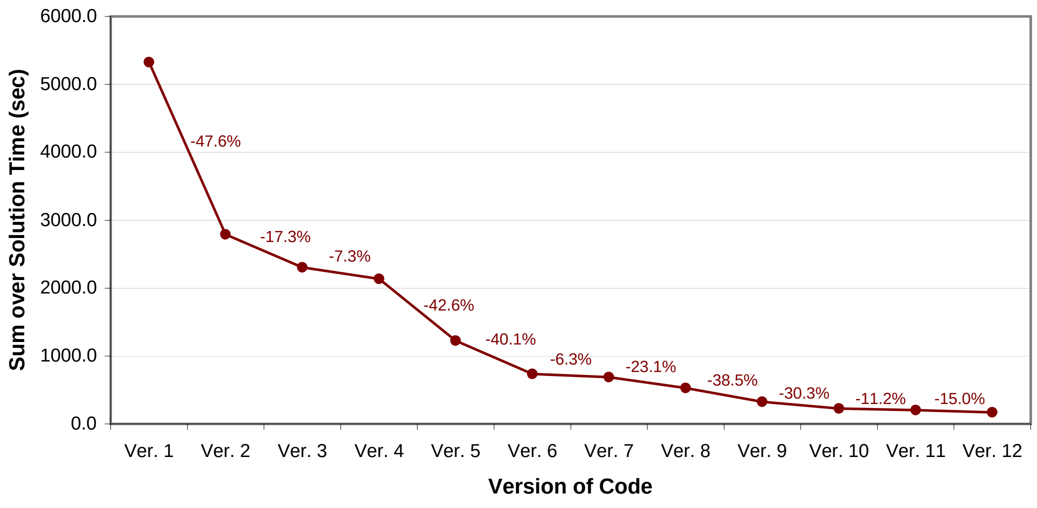
Version	Added Implementation Technique / Modification
Ver. 1	First Version
Ver. 2	Dual Steepest Edge
Ver. 3	Packed Storage of α^r
Ver. 4	Packed Storage of ρ
Ver. 5	Vector of Primal Infeasibilities
Ver. 6	Bound Flipping Ratio Test, Cost Perturbation
Ver. 7	Numerical Stability
Ver. 8	Hypersparse FTran,BTran
Ver. 9	Tight bounds after LP Preprocessing
Ver. 10	Revised Dual Steepest Edge
Ver. 11	Hypersparse LU-Update
Ver. 12	Randomized Partial Pricing

Table 9.3: Progress in our dual simplex code: implementation techniques in chronological order.

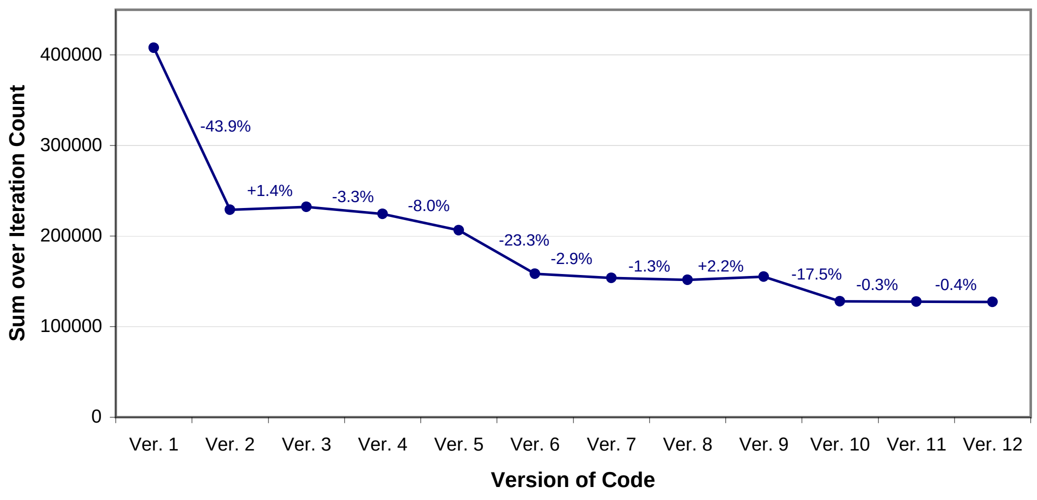
cess. The corresponding executables were generated with the Compaq Visual Fortran Compiler V6.6 and the test runs were conducted under Windows XP Professional on a standard Intel Pentium IV PC with 2,0 GHz and 512MB of main memory. With the first versions of our code we could not solve some of the NetLib test problems due to numerical problems. Therefore, this study is limited to a selection of thirteen large NetLib problems, which could already be solved with these early versions. Tables A.12 and A.13 show the detailed solution times and iteration counts, respectively. Visualizations of the sums over time and iterations for each version are depicted in figure 9.3.

The first version (*Ver. 1*) basically corresponded to algorithm 2 with Devex pricing. The two required systems of linear equations per iteration were solved based on the LU-factorization and -update described in chapter 5. Sparsity was exploited in FTran and BTran and by a rowwise computation of the transformed pivot row, if appropriate. Intermediate solution vectors like ρ, α_q and α^r were stored in dense arrays. In this phase of development we tested our code mainly on the NetLib problems. It already worked quite well on well conditioned, not to sparse problems (like MAROS-R7, D2Q06C, FIT2P). Some of the NetLib problems could not be solved due to numerical problems (like GROW15, GROW22, GREENBEA, GREENBEB, the PILOT-problems) and others due to missing techniques, mainly cost perturbation and dual steepest edge pricing (like the PDS-models and our test models from bus scheduling [37], the smallest of which is the model PTV15). The model KEN-18 could be solved correctly, but our code was about 200 times slower in terms of solution time than other state-of-the-art solvers (like Cplex 8.0).

The techniques in table 9.3 can be related to the three goals: 1. decrease number of iterations, 2. decrease time per iteration, and 3. improve numerical stability. Three techniques contributed to the first goal: dual steepest edge pricing, the bound flipping ratio test and cost perturbation. In our first attempt to implement DSE pricing (*Ver. 2*) we reset the DSE weights to 1.0 after every refactorization (as it



(a) Sum over solution time of 13 large NetLib models.



(b) Sum over iterations count of 13 large NetLib models.

Figure 9.3: Chronological progress in the development process of the MOPS Dual Simplex code.

was done with the Devex weights before). Later, we revised our implementation (*Ver. 10*) according to our description in section 8.2.2.1 and were surprised when we saw the effects. Also BFRT and cost perturbation (*Ver. 6*) reduced the number of iterations significantly. On the test set used in this section, this is mainly due to the ratio test. Cost perturbation turned out to be crucial to solve the PDS-models.

Time per iteration could be reduced significantly by incorporating the implementation techniques to exploit hypersparsity: packed data structures for intermediate result vectors and reorganization of zeroing out operations (*Ver. 3* and *Ver. 4*), hypersparse FTran, BTran and LU-updated operations (*Ver. 8* and *Ver. 11*) and reorganization of the pricing loop (*Ver. 5* and *Ver. 12*). These techniques lead to a particular breakthrough on the KEN-problems, which now could be solved in competitive time.

The breakthrough in terms of numerical stability was achieved in *Ver. 7* with the incorporation of Harris' ratio test and the cost shifting technique described in sections 6.2.2.3 and 6.2.2.4. This was the first version of our code, which solved all of the NetLib test problems correctly.

9.5 Overall benchmarks

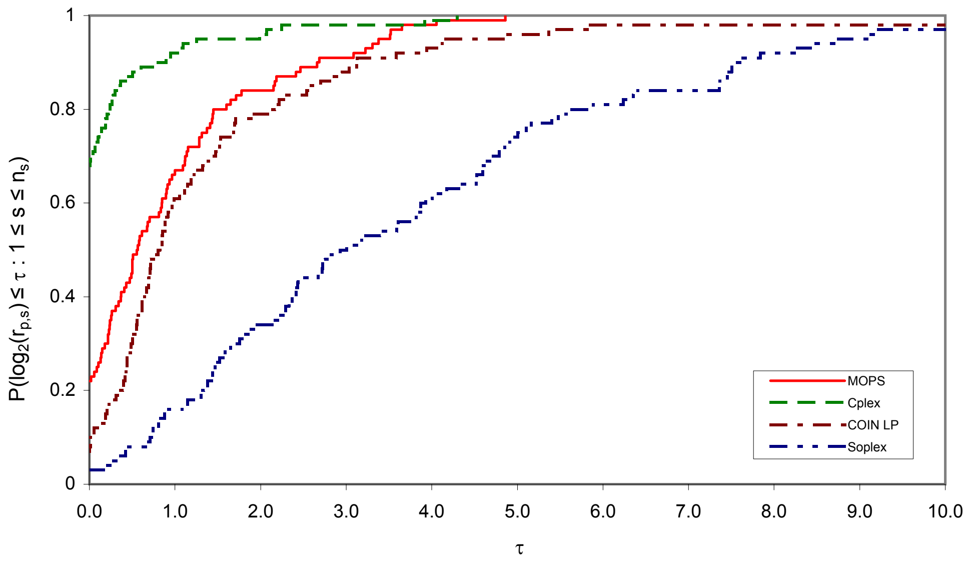
To evaluate the performance of the MOPS dual simplex code compared to dual simplex implementations of the LP-systems Soplex 1.2.1, COIN LP (CLP) 1.02.02 and Cplex 9.1 we conducted two types of benchmarks. In the first experiment we solved the original version of our test set problems using the dual simplex method[7] of the four solvers with default settings and the respective system-specific LP preprocessing. In the following we refer to this experiment as *system benchmark*. To be able to analyze the performance the of dual simplex codes independently of the quite different LP-preprocessors we wrote out unscaled versions of the test problems after the COIN LP preprocessing in MPS-format and solved these modified problems. But this time we turned off the LP preprocessors of the four solvers. We will refer to this second experiment as *dual benchmark*.

The test runs were conducted under Windows XP Professional on a standard Intel Pentium IV PC with 3,2 GHz and 1GB of main memory. The MOPS executable was built by the Intel Visual Fortran Compiler V9.0, the Soplex and CLP code was compiled using Microsoft Visual C++ V6.0. The detailed results are given in the appendix in tables A.2 to A.5. Figure 9.4 shows two performance profiles over solution time, one for the system and one for the dual benchmark. Tables 9.4 and 9.5 give an overview of the solution times for the most difficult among our test set problems for the respective benchmark[8].

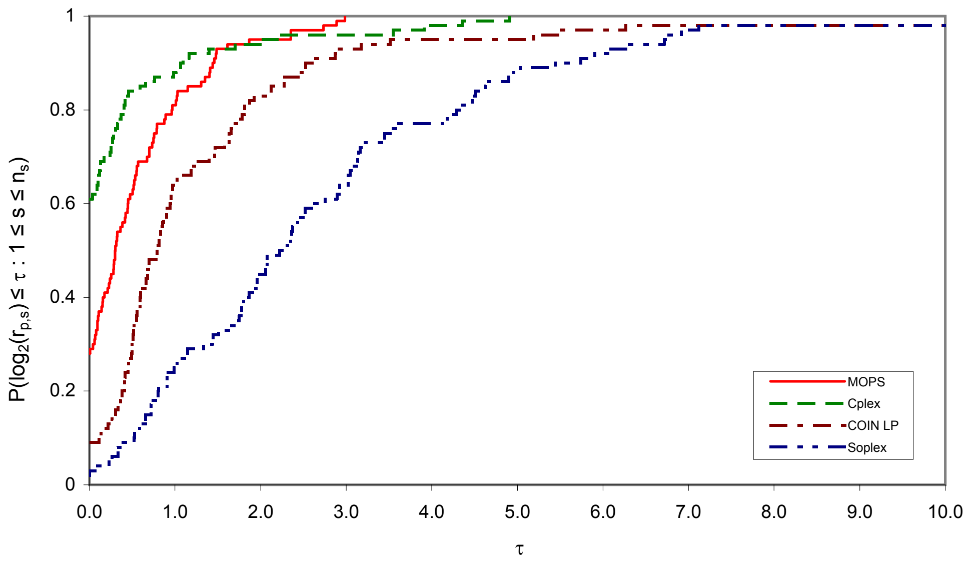
The system performance profile shows, that MOPS clearly outperforms CLP and

[7]For Soplex, we used the entering algorithm, which conceptually can be seen as the primal simplex method using the dual simplex method to achieve primal feasibility at the start. In practise however, three fourths of the iterations (average, may vary tremendously on different problems) are performed by the dual simplex part of the code.

[8]We sorted by the best (shortest) solution time required by any of the four solvers and extracted the twenty-five problems with the longest solution time.



(a) System benchmark: performance profile over solution time on original models including LP-preprocessing.



(b) Benchmark of dual simplex codes (no LP-preprocessing): performance profile over solution time on models produced by COIN LP-preprocessing.

Figure 9.4: Overall benchmarks.

	MOPS 7.9	Soplex 1.2.1	CLP 1.02.02	Cplex 9.1
MUN1_M_D	15720.8	>72000.0	f	14295.8
RAIL4284	5944.9	10058.1	7417.0	4019.8
MUN18_M_D	3142.3	>72000.0	f	5469.3
STP3D	2769.1	6726.7	1253.0	1021.2
STORMG2_1000	3464.4	15550.5	1747.4	537.9
NUG12	369.5	2763.4	430.4	189.4
QAP12	281.5	2271.5	339.4	188.5
WATSON_2	1301.7	6822.5	153.3	160.8
PDS-100	636.9	43859.1	193.8	143.2
DBIC1	871.3	26905.9	450.6	137.8
SELF	183.3	104.4	122.2	105.4
WORLD	97.5	1399.2	123.7	91.9
MOMENTUM3	980.6	894.5	1025.9	85.5[6]
P13	195.7	f	396.0	80.3
MOD2	70.4	630.9	93.5	69.6
FOME11	57.8	173.4	74.3	59.6
NEMSWRLD	65.9	151.7	82.9	56.0
LP22	48.5	69.6	59.2	41.7
ULEVIMIN	41.5	22565.2	61.0	47.8
FA	44.3	197.8	54.8	40.4
DANO3MIP	105.6	98.2	73.3	39.8
HAL_M_D	55.2	286.3	67.6	37.6
ATLANTA-IP	37.3	68.5	38.8	56.6
LPL1	80.8	1748.7	51.0	36.7
PDS-40	86.4	2895.5	35.8	35.5

Table 9.4: System benchmark: solution time (including time for LP preprocessing) on 25 most difficult original models.

	MOPS 7.9	Soplex 1.2.1	CLP 1.02.02	Cplex 9.1
MUN1_M_D	14143.4	>72000.0	f	15382.9
RAIL4284	5865.1	10325.0	7132.2	5495.1
MUN18_M_D	3181.6	>72000.0	f	12716.2
STP3D	1993.3	5275.1	1249.6	1519.4
STORMG2_1000	3894.7	12171.7	1727.6	492.3
NUG12	478.7	1791.2	430.4	463.9
QAP12	462.1	2961.2	339.4	455.7
MOMENTUM3	561.6	1045.3	1024.9	211.0
DBIC1	250.6	3678.1	448.9	144.9
WATSON_2	174.2	2943.4	143.1	225.4
PDS-100	319.1	47991.6	184.3	129.0
SELF	125.2	101.8	122.2	122.4
P16	93.6	506.4	101.0	246.4
WORLD	111.4	500.7	123.1	81.6
P15	67.9	390.1	380.6	152.1
MOD2	83.1	447.1	92.9	66.2
NEMSWRLD	69.6	171.3	82.6	60.1
P13	52.3	571.0	394.7	57.0
FOME11	52.2	141.2	73.6	78.5
LP22	48.4	74.1	59.1	47.0
FA	41.0	212.8	54.4	44.0
LPL1	81.7	704.1	49.8	40.3
HAL_M_D	38.6	286.9	67.2	42.2
ATLANTA-IP	40.0	63.0	38.3	52.6
P05	56.6	119.8	52.6	34.9

Table 9.5: Benchmark of dual simplex codes (no LP preprocessing): solution time on 25 most difficult models pretreated by COIN LP preprocessing.

Soplex, which both fail on two of the largest problems (MUN1_M_D and MUN18_M_D)[9]. On about 20% of the test models, MOPS achieves the best solution time. However, on this benchmark, Cplex 9.1 clearly yields the best overall performance. In the dual benchmark however, MOPS achieves a better total solution time and is more stable with respect to difficult problems. For about 30% of the problems it achieves the best solution time and for about 80% of the problems it is at most two times slower than the best solver.

It is interesting to see, that MOPS and Soplex show significantly better results on the problems pretreated by the COIN LP preprocessing, while the results of Cplex 9.1 slightly worsen. Therefore we conclude, that great improvement of the performance of the MOPS dual simplex code can be achieved by working on a better fine tuning of the MOPS LP preprocessing (which is however already quite sophisticated).

[9]CLP fails to remove the cost perturbation, Soplex exceeds the 12 hour time limit.

Chapter 10

Summary and Conclusion

In this thesis we presented the mathematical algorithms, computational techniques and implementation details of our dual simplex code, which outperforms the best existing open-source and research codes and is competitive to the leading commercial LP-systems.

In part I we laid out the algorithmic basis for our implementation of the dual simplex method, which is a classical two-phases approach. After introducing basic concepts in chapter 2 we developed a new elaborate algorithmic description of the dual phase II in chapter 3, which incorporates the two crucial mathematical techniques "dual steepest edge pricing" and "bound flipping ratio test" into the revised dual simplex algorithm. In chapter 4 we described several dual phase I methods. We showed, that the algorithmic approach to minimize the sum of dual infeasibilities can be explicitly modeled as a subproblem, which is mathematically equivalent and can directly be solved by the dual phase II. Therefore the subproblem approach is much easier to implement. Furthermore, we gave the first description of Pan's dual phase I method for general LPs with explicit lower and upper bounds. We overcame the main drawback of Pan's method, which showed bad convergence on very few numerically rather difficult test problems, by combining it with the subproblem approach. The resulting algorithm outperformed the other methods in our computational tests.

In part II we gave detailed descriptions of techniques to achieve computational efficiency and numerical stability. In chapter 5 we addressed the efficient solution of the required systems of linear equations based on the LU-factorization of the basis. To our knowledge we gave the first complete mathematical description of this technique, which allows to use only one instead of two permutation matrices in the FTran and BTran operations. Additionally, we discussed in detail, how the technique of Gilbert and Peierls [29] to exploit hypersparsity can be efficiently incorporated into this solution framework. In chapter 6 we described techniques to solve numerically difficult LP problems and reduce the number of degenerate iterations. Our main contribution is the integration of Harris' ratio test with bound flipping and cost shifting techniques. We also gave a detailed description of the cost perturbation technique, which works well in our code.

In part III we discussed important implementation issues and presented computational results. Chapter 8 focussed on the implementation of the dual pricing step, the dual ratio test and the exploitation of hypersparsity. The computational progress achieved by the presented techniques was analyzed in chapter 9. Furthermore, this chapter contained a study on the dual phase I and an overall benchmark study.

We could show, that our new combined dual phase I approach outperforms other methods from literature. We also studied the impact of the bound handling in LP preprocessing on dual phase I and the overall performance. Finally, we compared the performance of our code to that of the open-source codes COIN LP 1.02.02 and Soplex 1.2.1 and the commercial code Cplex 9.1. While it turned out to be significantly superior to Soplex and well better that COIN LP, it was slightly inferior to Cplex 9.1. We want to emphasize again, that neither the techniques used in the COIN LP code nor the internals of the Cplex code are documented in research literature.

Future research in this field can pursue several possible directions. One direction is to try to achieve further improvements in the given mathematical setting. Often inconspicuous implementation details have a great impact on the performance of the code (like the bound handling after LP preprocessing). To discover and document such details it is necessary to investigate the behavior of our dual simplex code on exceptionally difficult test problems. One important aspect is for instance to find better criteria, which allow for a problem specific switching between the different computational techniques (dense, sparse, hypersparse linear systems; simple and bound flipping ratio test etc.). Another possible research direction is to improve the dual simplex algorithm on the mathematical side. For instance, Pan proposed a new promising formulation of the dual simplex algorithm in [56]. It is however not clear yet, whether his algorithm is computationally competitive.

Bibliography

[1] Bpmpd test problems. http://www.sztaki.hu/∼meszaros/bpmpd/.

[2] Dsor test problems. http://dsor.upb.de/koberstein/lptestset/.

[3] Miplib 2003 test problems. http://miplib.zib.de/.

[4] Mittelmann test problems. ftp://plato.asu.edu/pub/lpfree.html.

[5] Netlib test problems. http://www.netlib.org/lp/data/.

[6] Sequential objectoriented simplex. http://www.zib.de/Optimization/Software/Soplex/, 1996.

[7] O. Axelsson. A survey of preconditioned iterative methods for linear systems of algebraic equations. *BIT*, 25:166–187, 1985.

[8] R. H. Bartels and G. H. Golub. The simplex method of linear programming using lu decomposition. *Commun. ACM*, 12(5):266–268, 1969.

[9] M. Benichou, J. Gautier, G. Hentges, and G. Ribiere. The efficient solution of large-scale linear programming problems. *Mathematical Programming*, 13:280–322, 1977.

[10] R. Bixby. Solving real-world linear programs: a decade and more of progress. 2002.

[11] R. E. Bixby and A. Martin. Parallelizing the dual simplex method. *INFORMS Journal on Computing*, 12(1):45–56, 2000.

[12] R. Bland. New finite pivot rule for the simplex method. *Mathematics of Operations Research*, 2:103–107, 1977.

[13] R. K. Brayton, F. G. Gustavson, and R. A. Willoughby. Some results on sparse matrices. *Mathematics of Computation*, 24(122):937–954, 1970.

[14] A. L. Brearley, G. Mitra, and H. P. Williams. Analysis of mathematical programming problems prior to applying the simplex method. *Mathematical Programming*, 8:54–83, 1975.

[15] V. Chvatal. *Linear Programming.* W. H. Freeman and Company, New York, 16th edition, 2002.

[16] L. Collatz and W. Wetterling. *Optimierungsaufgaben.* Springer-Verlag, Berlin, 1966.

[17] W. H. Cunningham. Theoretical properties of the network simplex method. *Mathematics of Operations Research*, 4:196–208, 1979.

[18] G. B. Dantzig. Programming in a linear structure. *U. S. Air Force Comptroller, USAF, Washington, D.C.*, 1948.

[19] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In [40], pages 339–347.

[20] G. B. Dantzig and W. Orchard-Hays. The product form for the inverse in the simplex method. *Mathematical Tables and Other Aids toComputation*, 8:64–67, 1954.

[21] G. B. Dantzig, A. Orden, and P. Wolfe. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5:183–195, 1955.

[22] E. D. Dolan and J. J. More. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[23] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Oxford University Press, Oxford, 1986.

[24] J. J. H. Forrest and J. A. Tomlin. Updating triangular factors of the basis to maintain sparsity in the product form simplex method. *Mathematical Programming*, 2:263–278, 1972.

[25] J. J. H. Forrest and J. A. Tomlin. Implementing the simplex method for the optimization subroutine library. *IBM Systems Journal*, 31(1):11–25, 1992.

[26] John J. Forrest and Donald Goldfarb. Steepest-edge simplex algorithms for linear programming. *Math. Program.*, 57(3):341–374, 1992.

[27] Robert Fourer. Notes on the dual simplex method. Draft report, 1994.

[28] S. I. Gass. The first linear-programming shoppe. *Operations Research*, 50(1):61–68, 2002.

[29] J. R. Gilbert and T. Peierls. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sci. Statist. Comput.*, 9:862–874, 1988.

[30] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, 45:437–474, 1989.

[31] P. E. Gill, W. Murray, and M. H. Wright. *Numerical Linear Algebra and Optimization*, volume 1. Addison-Wesley Publishing Company, 1991.

[32] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The John Hopkins University Press, Baltimore, London, 3 edition, 1996.

[33] P. M. J. Harris. Pivot selection methods of the DEVEX LP code. *Math. Programming*, 5:1–28, 1973.

[34] E. Hellermann and D. Rarick. Reinversion with the preassigned pivot procedure. *Mathematical Programming*, 1(2):195–215, 1971.

[35] A. Hoffman, M. Mannos, D. Sokolosky, and D. Wiegmann. Computational experience in solving linear programs. *SIAM Journal*, 1:1–33, 1953.

[36] A. J. Hoffman. Linear programming at the national bureau of standards. In [43].

[37] N. Kliewer, T. Mellouli, and L. Suhl. A time-space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operations Research*, to appear 2005.

[38] A. Koberstein. Progress in the dual simplex algorithm for solving large scale LP problems: Techniques for a fast and stable implementation. *forthcoming.*

[39] A. Koberstein and U.H. Suhl. Progress in the dual simplex algorithm for solving large scale lp problems: Practical dual phase 1 algorithms. *to appear in: Computational Optimization and Applications*, 2005.

[40] T. C. Koopmans, editor. *Activity Analysis of Production and Allocation*, New York, 1951. John Wiley and Sons.

[41] E. Kostina. The long step rule in the bounded-variable dual simplex method: Numerical experiments. *Mathematical Methods of Operations Research*, 55:413–429, 2002.

[42] C. E. Lemke. The dual method of solving the linear programming problem. *Naval Research Logistics Quarterly*, 1:36–47, 1954.

[43] J. K. Lenstra, A. H. G. Rinnooy Kan, and A. Schrijver, editors. *History of Mathematical Programming.* North-Holland, Amsterdam, 1991.

[44] R. Lougee-Heimer, F. Barahona, B. Dietrich, J. P. Fasano, J. J. Forrest, R. Harder, L. Ladanyi, T. Pfender, T. Ralphs, M. Saltzman, and K. Schienberger. The coin-or initiative: Open-source software accelerates operations research progress. *ORMS Today*, 28(5):20–22, October 2001.

[45] H. M. Markowitz. The elimination form of the inverse and its applications to linear programming. *Management Science*, 3:255–269, 1957.

[46] I. Maros. *Computational techniques of the simplex method.* Kluwer's International Series, 2003.

[47] I. Maros. A generalized dual phase-2 simplex algorithm. *European Journal of Operational Research*, 149(1):1–16, 2003.

[48] I. Maros. A piecewise linear dual phase-1 algorithm for the simplex method with all types of variables. *Computational Optimization and Applications*, 26:63–81, 2003.

[49] C. Mészáros and U. H. Suhl. Advanced preprocessing techniques for linear and quadratic programming. *OR Spectrum*, 25(4):575–595, 2003.

[50] B. A. Murtagh. *Advanced Linear Programming: Computation and Practice.* McGraw-Hill, INC., 1981.

[51] W. Orchard-Hays. History of the development of lp solvers. *Interfaces*, 20(4):61–73, July-August 1990.

[52] A. Orden. Solution of systems of linear inequalities on a digital computer. *Proceedings of the meeting of the ACM, May 2, 1952, Pittsburgh, PA*, 1952. Directorate of Management Analysis, Headquarters, U.S. Air Force, Washington, D.C.

[53] M. W. Padberg. *Linear optimization and extensions.* Springer, Berlin, 1995.

[54] P. Q. Pan. Practical finite pivoting rules for the simplex method. *OR Spektrum*, 12:219–225, 1990.

[55] P. Q. Pan. The most-obtuse-angle row pivot rule for achieving dual feasibility: a computational study. *European Journal of Operations Research*, 101(1):167–176, 1997.

[56] P. Q. Pan. A revised dual projective pivot algorithm for linear programming. 2004.

[57] F.M. Kirillova R. Gabasov and O.I. Kostyukova. A method of solving general linear programming problems. *Doklady AN BSSR*, 23(3):197–200, 1979. (in Russian).

[58] J. K. Reid. A sparsity-exploiting variant of the bartels-golub decomposition for linear programming bases. *Mathematical Programming*, 24:55–69, 1982.

[59] D. Ryan and M. Osborne. On the solution of highly degenerate linear programmes. *Mathematical Programming*, 41:385–392, 1988.

[60] M. A. Saunders. *The complexity of computational problem solving*, chapter The complexity of LU updating in the simplex method, pages 214–230. University of Queensland Press, St. Lucia, Queensland, 1976.

[61] M. A. Saunders. *Sparse Matrix Computations*, chapter A fast, stable implementation of the simplex method using Bartels-Golub updating, pages 213–226. Academic Press, New York et. al., 1976.

[62] I. Steinzen, A. Koberstein, and U.H. Suhl. Ein entscheidungsunterstützungssystem zur zuschnittoptimierung von rollenstahl. In [64], pages 126–143.

[63] L. M. Suhl and U. H. Suhl. A fast lu-update for linear programming. *Annals of Operations Research*, 43:33–47, 1993.

[64] L. M. Suhl and S. Voss, editors. *Quantitative Methoden in ERP und SCM, DSOR Beiträge zur Wirtschaftsinformatik.* 2004.

[65] U. H. Suhl. Mops - mathematical optimization system. *European Journal of Operational Research*, 72:312–322, 1994.

[66] U. H. Suhl. Mops - mathematical optimization system. *OR News*, 8:11–16, 2000.

[67] U. H. Suhl. It-gestützte, operative sortimentsplanung. In B. Jahnke and F. Wall, editors, *IT-gestützte betriebliche Entscheidungsprozesse*, pages 175–194. Gabler, 2001.

[68] U. H. Suhl and L. M. Suhl. Computing sparse lu factorizations for large-scale linear programming bases. *ORSA Journal on Computing*, 2(4):325–335, 1990.

[69] U. H. Suhl and L. M. Suhl. *Operational Research in Industry*, chapter Solving Airline Fleet Scheduling Problems with Mixed-Integer Programming, pages 135–156. MacMillan Press Ltd., 1999.

[70] U. H. Suhl and R. Szymanski. Supernode processing of mixed-integer models. *Computational Optimization and Applications*, 3:317–331, 1994.

[71] U. H. Suhl and V. Waue. Fortschritte bei der lösung gemischt-ganzzahliger optimierungsmodelle. In [64], pages 35–53.

[72] R. P. Tewarson. The product form of inverses of sparse matrices and graph theory. *SIAM Review*, 9(1):91–99, 1967.

[73] J. A. Tomlin. On scaling linear programming problems. In M. L. Balinski and E. Hellerman, editors, *Computational Practise in Mathematical Programming*, chapter Mathematical Programming Study 4, pages 146–166. North-Holland, Amsterdam, 1975.

[74] P. Wolfe. The composite simplex algorithm. *SIAM Review*, 7(1):42–54, 1965.

[75] R. Wunderling. Paralleler und objektorientierter simplex. Technical report, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1996.

Appendix A

Tables

		Original models			COIN LPP models		
Name	Source	Structs	Constrs	Nzs	Structs	Constrs	Nzs
AA3	c	8627	825	70806	8560	757	67484
AIR04	m	8904	823	72965	8873	777	69933
ATLANTA-IP	m	48738	21732	257532	17487	20004	183240
BAS1LP	c	4461	5411	582411	4443	5409	582390
BAXTER	t	15128	27441	95971	10738	18863	78420
CO9	c	14851	10965	166766	11045	7071	83093
CQ9	c	13778	9451	149212	11234	6645	77268
CRE-B	n	72447	9648	256095	31668	5170	106881
CRE-D	n	69980	8926	242646	25087	3985	82771
D2Q06C	n	5167	2171	32417	4665	1787	30478
DANO3MIP	m	13873	3202	79655	13825	3139	80277
DBIC1	t	183235	43200	1038761	140359	33688	781948
DBIR1	c	27355	18804	1058605	25015	7157	998555
DEGEN4	n	6711	4420	101377	6649	4353	101078
DFL001	n	12230	6071	35632	9915	3935	31754
DS	m	67732	656	1024059	67732	656	1024059
EX3STA1	c	8156	17443	59419	7335	16622	58057
FA	o	25324	12251	72520	21858	8824	63420
FAST0507	m	63009	507	409349	62173	484	401766
FIT2P	n	13525	3000	50284	13525	3000	50284
FOME11	t	24460	12142	71264	19830	7870	63508
FOME20	t	105728	33874	230200	77464	8394	199524
FXM4_6	t	30732	22400	248989	27976	18020	219298
GEN4	t	4297	1537	107094	4173	1475	104228
GREENBEA	n	5405	2392	30877	3361	1313	20932
GREENBEB	n	5405	2392	30877	3354	1314	20824
HAL_M_D	o	56334	15000	119242	55509	14169	117610
JENDREC1	c	4228	2109	89608	3535	2109	88915
KEN-13	n	42659	28632	97246	24815	10959	57064
KEN-18	n	154699	105127	358171	89445	39873	210544
LP22	t	13434	2958	65560	8693	2872	60181
LPL1	c	125000	39951	381259	82637	32460	252764
LPL3	c	33538	10828	100377	22575	7797	67493
MAROS-R7	n	9408	3136	144848	6605	2152	80194
MOD2	t	31728	35664	198250	28685	28651	124850
MODEL10	c	15447	4400	149000	10427	2894	94559
MOMENTUM2	m	3732	24237	227974	3408	20871	188584
MOMENTUM3	m	13532	56822	542009	13401	56490	541035
MSC98-IP	m	21143	15850	92918	12824	15294	81668
MUN1_M_D	o	1479833	163142	3031285	1472987	156074	3020407
MUN18_M_D	o	675333	62148	1379406	674732	61528	1379589
MZZV11	m	10240	9499	134603	9514	8926	133951
MZZV42Z	m	11717	10460	151261	11526	10379	150592
NEMSPMM2	c	8413	3089	78887	7363	1808	59246
NEMSWRLD	c	27174	7138	190907	21878	4922	153345
NET12	m	14115	14021	80384	14115	14021	80384
NSCT2	t	14981	23003	675156	11304	7797	612116
NUG08	c	1632	912	7296	1632	912	7296
NUG12	c	8856	3192	38304	8856	3192	38304
OSA-30	n	100024	4350	600138	96128	4279	262890
OSA-60	n	232966	10280	1397793	224137	10209	584276
P01	o	3329	6448	25451	2916	3998	17208
P02	o	24316	22323	222175	7087	6358	104480
P03	o	34557	33482	105392	14474	13461	44941
P04	o	22707	19993	129622	6165	4417	61903
P05	o	121147	41695	351592	86745	28179	170523
P06	o	194445	76842	336264	97153	21174	172790
P07	o	139709	54057	330444	95092	41409	238280
P08	o	146840	58248	571495	52749	19056	269684
P09	o	21311	2501	255210	16191	1979	192070
P10	o	724096	508946	1431278	198189	6411	204429
P11	o	136265	66050	407668	66560	17036	166829
P12	o	23479	19368	208530	4583	3214	68332
P13	o	65111	57600	321992	12816	9803	108373
P14	o	102076	92048	589583	9398	5104	74260
P15	o	158347	57335	352509	68060	12514	151390
P16	o	178281	69163	673493	76772	12906	156212

– continued on next page –

– continued from previous page –

		Original models			COIN LPP models		
Name	Source	Structs	Constrs	Nzs	Structs	Constrs	Nzs
P17	o	28015	21484	91194	19781	14176	59184
P18	o	28709	21828	92944	20354	14458	60683
P19	o	32155	25400	265312	8483	5813	85256
P20	o	27256	26070	137367	5128	3982	51191
PDS-10	n	48763	16558	106436	31443	3845	83182
PDS-100	n	505360	156243	1086785	417201	78328	996496
PDS-20	t	105728	33874	230200	77464	8394	199524
PDS-40	t	212859	66844	462128	167622	22088	419586
PILOT	n	3652	1441	43167	3161	1173	38791
PILOT87	n	4883	2030	73152	4416	1787	69910
PTV15	o	124383	14704	254418	121362	11857	248374
QAP12	t	8856	3192	38304	8856	3192	38304
RAIL4284	t	1092610	4284	11279748	1090563	4176	11174834
RAIL507	t	63009	507	409349	62172	484	401763
RAT5	c	9408	3136	137413	5408	1774	53548
RD-RPLUSC-21	m	622	125899	852384	543	54182	352312
SCFXM1-2R-256	c	57714	37980	213159	47966	25918	163902
SELF	t	7364	960	1148845	7364	960	1148845
SEYMOUR	m	1372	4944	33549	1153	4808	33047
SGPF5Y6	t	308634	246077	828070	42188	19499	118168
SLPTSK	c	3347	2861	72465	3347	2861	72465
SOUTH31	c	35421	18738	112897	35223	17660	93077
SP97AR	m	14101	1761	290968	14101	1706	283446
STORMG2_1000	t	1259121	528185	3341696	1110112	378036	2938345
STORMG2-125	t	157496	66185	418321	138869	47286	367623
STP3D	m	204880	159488	662128	178873	139206	583370
T0331-4L	c	46915	664	430982	21626	664	253203
T1717	m	73885	551	325689	16428	551	85108
ULEVIMIN	c	44605	6590	162206	41227	4535	176452
VAN	m	12481	27331	487296	7552	22400	482176
WATSON_1	t	383927	201155	1052028	188053	77474	660469
WATSON_2	t	671861	352013	1841028	331914	135714	1158510
WORLD	t	32734	35510	198793	30273	28348	124841

Table A.1: Model dimensions.

Cplex 9.1	Original models			COIN LPP models		
results	Time	Iters	Phase 1	Time	Iters	Phase1
AA3	1.2	1952	0	1.9	3024	0
AIR04	2.8	3725	0	4.6	5536	0
ATLANTA-IP	56.6	29337	601	52.6	28522	765
BAS1LP	9.6	3683	0	9.7	3840	0
BAXTER	2.3	6019	0	2.3	5921	0
CO9	18.4	15882	5	25.6	17793	21
CQ9	11.1	13601	0	10.7	15508	0
CRE-B	4.0	10472	0	2.6	10287	0
CRE-D	1.6	7293	0	1.3	6900	0
D2Q06C	2.4	4238	235	2.2	4745	161
DANO3MIP	39.8	31579	23	32.1	26027	14
DBIC1	137.8	47719	0	144.9	50566	0
DBIR1	1.3	1604	0	0.4	1518	0
DEGEN4	7.6	4822	8	7.3	6670	0
DFL001	21.7	19152	0	25.8	23327	0
DS	26.7	7133	0	26.1	7133	0
EX3STA1	41.9	8531	1846	55.0	9461	3517
FA	40.4	38325	0	44.0	39771	0
FAST0507	18.1	5832	0	19.0	5909	0
FIT2P	4.0	5170	0	4.0	5170	0
FOME11	59.6	37745	0	78.5	47914	0
FOME20	8.0	16702	0	5.1	14681	0
FXM4_6	4.8	32677	14006	3.2	24985	9370
GEN4	1.6	542	0	2.5	650	9
GREENBEA	0.9	3379	1907	0.4	2393	231
GREENBEB	1.4	4170	168	0.7	3210	72
HAL_M_D	37.6	20204	0	42.2	21301	0
JENDREC1	2.2	3372	1236	2.1	3372	1236
KEN-13	1.3	13513	0	1.2	13797	0
KEN-18	9.8	49224	0	9.9	49625	0
LP22	41.7	22470	8	47.0	25914	33
LPL1	36.7	35442	0	40.3	35147	2
LPL3	2.4	6295	0	2.1	6339	0
MAROS-R7	3.0	2800	0	3.0	2751	0
MOD2	69.6	34799	0	66.2	43948	0
MODEL10	17.0	21203	0	18.3	23612	0
MOMENTUM2	3.4	16788	464	4.6	3982	74
MOMENTUM3	85.5	76718	1030	211.0	33636	8
MSC98-IP	75.7	40947	2255	73.6	38909	2448
MUN1_M_D	14295.8	324017	0	15382.9	348118	0
MUN18_M_D	5469.3	141395	0	12716.2	245630	0
MZZV11	141.3	55872	0	135.6	59700	1832
MZZV42Z	71.5	28882	0	97.7	46003	707
NEMSPMM2	4.9	6314	138	3.6	7056	169
NEMSWRLD	56.0	33274	0	60.1	39619	4
NET12	7.8	7927	0	7.7	7927	0
NSCT2	1.3	6313	0	0.9	6513	0
NUG08	1.8	6345	0	2.1	6751	0
NUG12	189.4	86232	0	463.9	150851	0
OSA-30	1.5	2285	0	1.0	2395	0
OSA-60	4.1	4790	0	2.4	4751	0
P01	1.7	3522	3	1.5	3428	0
P02	23.2	12728	208	28.0	13366	8
P03	4.3	2349	0	7.2	2380	0
P04	16.9	12523	163	14.8	10101	0
P05	33.8	34595	28	34.9	34733	1
P06	7.0	28519	0	5.6	28505	6
P07	15.0	30154	0	13.2	30274	0
P08	15.7	25085	0	18.8	33672	0
P09	6.0	7593	0	8.3	8847	0
P10	3.8	168	0	0.4	367	0
P11	1.3	3679	0	0.4	2495	0
P12	5.0	4719	0	4.8	4675	0
P13	80.3	29522	13	57.0	22199	0
P14	5.6	5671	0	4.1	4714	0
P15	42.2	7183	0	152.1	13136	0
P16	57.6	7602	0	246.4	14586	109

– continued on next page –

– continued from previous page –

Cplex 9.1	Original models			COIN LPP models		
results	Time	Iters	Phase 1	Time	Iters	Phase1
P17	5.6	3014	0	6.2	2634	0
P18	2.6	2520	0	7.5	3314	0
P19	22.0	11613	4	23.4	10488	3
P20	17.6	13539	0	37.9	28925	1669
PDS-10	1.3	4978	0	0.7	4805	0
PDS-100	143.2	110512	0	129.0	106160	0
PDS-20	8.4	16702	0	4.9	14681	0
PDS-40	35.5	48835	0	23.3	41258	0
PILOT	3.2	3767	523	2.7	3400	17
PILOT87	30.6	14667	1370	13.2	7012	66
PTV15	15.9	15240	0	8.2	12793	0
QAP12	188.5	79902	0	455.7	151392	0
RAIL4284	4019.8	44114	1	5495.1	56886	0
RAIL507	11.6	3053	0	10.6	3219	0
RAT5	1.7	2336	0	1.7	2336	0
RD-RPLUSC-21	1.4	374	0	0.9	134	0
SCFXM1-2R-256	8.6	22344	0	13.3	24601	103
SELF	105.4	11694	0	122.4	13968	0
SEYMOUR	2.5	3291	0	2.2	3075	0
SGPF5Y6	3.2	16878	3042	1.2	18261	2952
SLPTSK	1.8	3207	2576	1.2	2214	1747
SOUTH31	56.1	24253	0	57.3	25099	0
SP97AR	1.5	1429	0	1.6	1565	0
STORMG2_1000	537.9	754914	0	492.3	693337	0
STORMG2-125	14.4	71442	0	13.7	78480	0
STP3D	1021.2	103469	0	1519.4	125533	0
T0331-4L	19.9	5915	0	25.2	7721	0
T1717	7.1	3925	0	6.9	4088	0
ULEVIMIN	47.8	42536	0	36.3	38506	0
VAN	18.3	10452	0	17.9	10841	0
WATSON_1	22.9	63944	179	21.2	70953	1851
WATSON_2	160.8	361486	296768	225.4	408280	330200
WORLD	91.9	41466	84	81.6	46492	84
Sum	28105.1	3542128	328892	39415.3	3849037	359489
Geo. Mean	14.3	–	–	14.0	–	–

Table A.2: Results with Cplex 9.1 Dual Simplex.

CLP results	Time	PP Time	Time in Dual	Iters
AA3	2.0	0.1	1.9	2423
AIR04	4.3	0.1	4.2	4107
ATLANTA-IP	38.8	0.5	38.3	15190
BAS1LP	2.8	0.9	1.9	1094
BAXTER	2.1	0.2	1.9	3952
CO9	27.0	0.4	26.6	13996
CQ9	15.6	0.3	15.3	12502
CRE-B	7.1	0.4	6.7	8928
CRE-D	4.5	0.6	3.9	6513
D2Q06C	4.5	0.2	4.3	5842
DANO3MIP	73.3	0.2	73.1	30850
DBIC1	450.6	1.7	448.9	51128
DBIR1	5.8	1.3	4.5	2424
DEGEN4	9.9	0.2	9.7	7557
DFL001	33.6	0.3	33.3	22270
DS	32.3	0.0	32.3	5724
EX3STA1	73.8	0.1	73.6	12455
FA	54.8	0.4	54.4	38178
FAST0507	15.3	0.6	14.7	3504
FIT2P	6.9	0.0	6.9	4830
FOME11	74.3	0.7	73.6	39414
FOME20	9.0	2.0	7.0	13759
FXM4_6	4.9	0.8	4.2	19150
GEN4	92.5	0.2	92.3	3855
GREENBEA	0.7	0.1	0.6	2398
GREENBEB	1.2	0.1	1.1	3637
HAL_M_D	67.6	0.4	67.2	25863
JENDREC1	4.1	0.1	4.0	4142
KEN-13	2.7	0.5	2.2	13031
KEN-18	15.0	1.8	13.3	47761
LP22	59.2	0.1	59.1	22327
LPL1	51.0	1.2	49.8	28832
LPL3	1.4	0.3	1.2	4249
MAROS-R7	4.4	0.2	4.2	2467
MOD2	93.5	0.6	92.9	33269
MODEL10	37.7	0.4	37.3	24112
MOMENTUM2	26.7	0.3	26.3	9933
MOMENTUM3	1025.9	1.0	1024.9	75259
MSC98-IP	277.5	0.3	277.2	100222
MUN1_M_D	f	f	f	f
MUN18_M_D	f	f	f	f
MZZV11	9.3	0.3	9.0	10495
MZZV42Z	3.6	0.4	3.3	7044
NEMSPMM2	6.6	0.2	6.4	8974
NEMSWRLD	82.9	0.4	82.6	36940
NET12	1.9	0.0	1.9	2071
NSCT2	3.0	0.8	2.2	5482
NUG08	2.0	0.0	2.0	4126
NUG12	430.4	0.0	430.4	96853
OSA-30	4.0	0.9	3.1	2258
OSA-60	13.3	1.8	11.5	5013
P01	2.7	0.1	2.6	4059
P02	103.7	0.8	102.9	30581
P03	65.3	0.3	65.0	16924
P04	48.7	0.5	48.2	19126
P05	54.6	2.1	52.6	33935
P06	13.2	3.7	9.5	29179
P07	31.1	5.7	25.4	39590
P08	39.3	2.8	36.5	36509
P09	32.0	0.3	31.7	15379
P10	107.7	91.3	16.4	7251
P11	54.0	51.4	2.6	9914
P12	13.8	0.6	13.2	8311
P13	396.0	1.3	394.7	61403
P14	15.9	1.9	14.0	11955
P15	384.3	3.8	380.6	59969
P16	106.1	5.0	101.0	13775

– continued on next page –

– continued from previous page –

CLP results	Time	PP Time	Time in Dual	Iters
P17	34.5	0.4	34.2	14830
P18	21.5	0.4	21.2	7971
P19	143.3	0.9	142.5	34881
P20	49.3	0.5	48.9	22301
PDS-10	2.1	0.9	1.2	5183
PDS-100	193.8	9.4	184.3	149094
PDS-20	8.9	1.9	7.1	13759
PDS-40	35.8	4.0	31.8	41360
PILOT	5.0	0.1	4.9	3651
PILOT87	24.9	0.2	24.6	7876
PTV15	13.3	0.9	12.4	13370
QAP12	339.4	0.0	339.4	80891
RAIL4284	7417.0	284.8	7132.2	62140
RAIL507	16.1	0.6	15.6	2975
RAT5	2.8	0.1	2.6	2102
RD-RPLUSC-21	21.8	21.1	0.7	201
SCFXM1-2R-256	27.0	0.9	26.0	37318
SELF	122.2	0.0	122.2	4594
SEYMOUR	3.2	0.1	3.1	3131
SGPF5Y6	19.3	15.9	3.4	24111
SLPTSK	4.2	0.0	4.2	4381
SOUTH31	54.6	0.3	54.3	18315
SP97AR	2.1	0.4	1.8	1209
STORMG2_1000	1747.4	19.8	1727.6	582344
STORMG2-125	28.6	1.8	26.8	68532
STP3D	1253.0	3.3	1249.6	104809
T0331-4L	32.7	0.7	31.9	7749
T1717	11.3	1.0	10.4	4381
ULEVIMIN	61.0	1.4	59.6	30400
VAN	22.1	0.7	21.4	9239
WATSON_1	97.0	4.8	92.3	139347
WATSON_2	153.3	10.2	143.1	208336
WORLD	123.7	0.7	123.1	38306

Table A.3: Results with COIN LP 1.02.02 Dual Simplex.

Soplex 1.2.1	Original models				COIN LPP models			
results	Time	Iters	Dual Its	Prim Its	Time	Iters	Dual Its	Prim Its
AA3	2.0	2171	2171	0	2.1	2236	2236	0
AIR04	4.4	3677	3677	0	4.5	3726	3726	0
ATLANTA-IP	68.5	16622	16358	264	63.0	15730	15578	152
BAS1LP	6.8	2995	2995	0	6.8	2994	2994	0
BAXTER	11.4	7845	7845	0	7.2	5924	5923	1
CO9	33.5	14853	14417	436	23.1	12658	12655	3
CQ9	28.8	15211	14886	325	17.8	12589	12587	2
CRE-B	64.6	22556	22556	0	14.6	12697	12697	0
CRE-D	58.0	20636	20636	0	6.9	9644	9644	0
D2Q06C	6.1	6596	5265	1331	4.5	5535	5251	284
DANO3MIP	98.2	42068	39076	2992	116.5	50451	41652	8799
DBIC1	26905.9	935458	745161	190297	3678.1	300344	300344	0
DBIR1	30.8	19478	5783	13695	12.1	4685	4685	0
DEGEN4	21.1	10459	6394	4065	26.5	12467	12467	0
DFL001	60.5	22013	22013	0	52.5	25126	25126	0
DS	22.0	4581	4581	0	22.0	4581	4581	0
EX3STA1	67.3	11694	0	11694	59.0	10606	0	10606
FA	197.8	58267	58267	0	212.8	63891	63891	0
FAST0507	21.0	4614	4614	0	17.6	4037	4037	0
FIT2P	6.5	5817	5817	0	6.9	7104	7104	0
FOME11	173.4	45672	45672	0	141.2	50097	50097	0
FOME20	250.3	52980	52980	0	112.0	36597	36597	0
FXM4_6	69.2	34346	24297	10049	63.5	41807	32880	8927
GEN4	51.9	1050	1050	0	7.9	638	638	0
GREENBEA	4.6	6050	3063	2987	1.5	3876	2819	1057
GREENBEB	6.4	7664	7372	292	2.7	5502	5453	49
HAL_M_D	286.3	44408	44408	0	286.9	44629	44629	0
JENDREC1	8.0	4490	2276	2214	8.0	4490	2276	2214
KEN-13	227.0	61007	61007	0	92.3	32801	32801	0
KEN-18	4243.6	237364	237364	0	1968.9	139824	139824	0
LP22	69.6	19191	19190	1	74.1	21979	21979	0
LPL1	1748.7	132277	132277	0	704.1	78228	78225	3
LPL3	3.2	4873	4873	0	1.9	4081	4081	0
MAROS-R7	23.9	6011	6011	0	4.6	2458	2458	0
MOD2	630.9	64736	64736	0	447.1	58386	58386	0
MODEL10	53.5	21585	13239	8346	62.6	30189	30175	14
MOMENTUM2	18.7	6205	6065	140	15.3	6333	6207	126
MOMENTUM3	894.5	47830	47726	104	1045.3	60610	60476	134
MSC98-IP	215.4	57066	54281	2785	117.8	33022	30540	2482
MUN1_M_D	>72000.0	–	–	–	>72000.0	–	–	–
MUN18_M_D	>72000.0	–	–	–	>72000.0	–	–	–
MZZV11	552.6	148557	148557	0	164.0	48543	48541	2
MZZV42Z	34.0	17589	17589	0	28.8	14784	14712	72
NEMSPMM2	14.2	11708	8474	3234	9.7	9820	7228	2592
NEMSWRLD	151.7	33812	32506	1306	171.3	42511	42510	1
NET12	2.4	1512	1512	0	2.4	1512	1512	0
NSCT2	8.8	8873	853	8020	6.9	7492	7482	10
NUG08	3.3	5331	5331	0	3.4	5331	5331	0
NUG12	2763.4	280440	280440	0	1791.2	196852	196852	0
OSA-30	18.7	3182	3182	0	58.6	8689	8689	0
OSA-60	107.8	6246	6246	0	331.3	16148	16148	0
P01	7.4	8568	4559	4009	2.4	4336	4335	1
P02	325.9	41620	23788	17832	251.0	65377	65376	1
P03	103.7	10415	117	10298	14.3	4852	4834	18
P04	387.5	55710	40176	15534	76.2	34307	34307	0
P05	174.5	37904	37898	6	119.8	33333	33325	8
P06	526.5	74077	74077	0	51.9	31082	31076	6
P07	2650.9	176179	176179	0	886.7	111165	111165	0
P08	2749.1	147692	146753	939	1007.8	125372	124464	908
P09	9.6	5962	5881	81	14.6	8344	8281	63
P10	2242.7	390429	390424	5	39.0	116884	116884	0
P11	216.0	55796	55784	12	18.3	29753	29753	0
P12	173.3	29597	14041	15556	24.3	14667	14667	0
P13	f	f	f	f	571.0	107667	107666	1
P14	1900.5	95907	53994	41913	44.1	24030	24027	3
P15	3590.8	95562	21657	73905	390.1	32071	31652	419
P16	5126.7	115646	25861	89785	506.4	39365	39361	4

– continued on next page –

– continued from previous page –

Soplex 1.2.1	Original models				COIN LPP models			
results	Time	Iters	Dual Its	Prim Its	Time	Iters	Dual Its	Prim Its
P17	95.2	15451	454	14997	26.1	8850	8850	0
P18	111.1	17398	1214	16184	33.0	10306	10279	27
P19	f	f	f	f	161.9	41118	41113	5
P20	f	f	f	f	30.2	14541	13391	1150
PDS-10	25.4	18841	18841	0	8.0	11224	11224	0
PDS-100	43859.1	922628	922628	0	47991.6	1058917	1058917	0
PDS-20	250.8	52980	52980	0	112.8	36597	36597	0
PDS-40	2895.5	186524	186524	0	2443.7	198742	198742	0
PILOT	15.6	6578	566	6012	5.4	4447	4429	18
PILOT87	55.1	14062	7420	6642	44.4	12381	11644	737
PTV15	31.4	17682	17682	0	72.2	17497	17497	0
QAP12	2271.5	239642	239642	0	2961.2	360323	360323	0
RAIL4284	10058.1	62350	62340	10	10325.0	63067	63067	0
RAIL507	13.0	2791	2791	0	11.1	2676	2676	0
RAT5	40.6	9489	9489	0	3.1	2118	2118	0
RD-RPLUSC-21	5.4	339	338	1	12.5	304	304	0
SCFXM1-2R-256	155.8	45722	35114	10608	108.2	42314	42195	119
SELF	104.4	4392	4392	0	101.8	4392	4392	0
SEYMOUR	2.9	2731	2731	0	2.8	2565	2565	0
SGPF5Y6	733.6	124654	104834	19820	10.0	19406	19143	263
SLPTSK	8.1	3218	722	2496	9.1	3526	583	2943
SOUTH31	53.7	18689	18623	66	64.1	18536	18536	0
SP97AR	1.8	1223	1223	0	1.9	1291	1291	0
STORMG2_1000	15550.5	679823	679823	0	12171.7	630928	630928	0
STORMG2-125	210.2	84930	84930	0	160.1	79153	79153	0
STP3D	6726.7	153415	153409	6	5275.1	142454	142442	12
T0331-4L	69.1	10837	10837	0	42.0	9546	9546	0
T1717	33.2	4623	4623	0	9.9	4169	4169	0
ULEVIMIN	22565.2	4748307	3381593	1366714	76.1	24342	24310	32
VAN	15.2	8120	8085	35	16.6	8347	8300	47
WATSON_1	1943.5	184968	9087	175881	2552.7	386132	383204	2928
WATSON_2	6822.5	290751	406	290345	2943.4	238539	537	238002
WORLD	1399.2	123302	123302	0	500.7	61831	61768	63

Table A.4: Results with Soplex 1.2.1 Dual Simplex.

MOPS 7.9	Original models			COIN LPP models		
results	Time	Iters	Phase 1	Time	Iters	Phase1
AA3	1.5	2249	0	1.7	2367	0
AIR04	3.1	3784	0	3.5	4265	0
ATLANTA-IP	37.3	15802	0	40.0	16545	0
BAS1LP	2.0	1399	0	0.8	724	0
BAXTER	3.3	4736	0	2.5	4851	0
CO9	15.8	12096	77	15.1	12483	0
CQ9	14.3	13624	4	12.7	13336	0
CRE-B	6.4	7706	27	4.3	7755	0
CRE-D	3.5	5663	25	1.9	5574	0
D2Q06C	3.2	5463	77	3.0	5722	29
DANO3MIP	105.6	56184	0	89.6	50964	0
DBIC1	871.3	67597	1220	250.6	45106	0
DBIR1	1.8	1921	0	2.9	2538	0
DEGEN4	14.2	8767	1813	8.1	7483	0
DFL001	20.6	19185	30	20.8	20232	0
DS	29.7	4404	0	40.4	6402	0
EX3STA1	10.6	4230	1093	16.9	7203	5
FA	44.3	36125	0	41.0	35069	0
FAST0507	18.1	5012	0	18.1	5423	0
FIT2P	4.3	4880	0	4.5	5274	0
FOME11	57.8	38983	24	52.2	37849	0
FOME20	14.9	22682	0	7.4	13734	0
FXM4_6	6.6	23370	1944	6.6	22937	1684
GEN4	2.6	619	0	2.6	596	0
GREENBEA	0.7	3071	1453	0.4	2461	30
GREENBEB	1.2	3723	81	1.0	4435	2
HAL_M_D	55.2	26647	0	38.6	23526	0
JENDREC1	3.0	4377	2590	2.1	3132	993
KEN-13	3.4	15659	0	1.8	13107	0
KEN-18	29.8	53256	0	16.7	47642	0
LP22	48.5	22700	0	48.4	22994	0
LPL1	80.8	33883	0	81.7	34918	0
LPL3	1.1	4054	0	1.0	4177	0
MAROS-R7	3.8	2605	0	5.9	3587	0
MOD2	70.4	28801	1	83.1	33794	0
MODEL10	32.5	29331	1	21.7	23207	0
MOMENTUM2	32.2	12081	0	23.3	11033	0
MOMENTUM3	980.6	83498	0	561.6	53537	0
MSC98-IP	31.8	18676	0	3.6	5726	0
MUN1_M_D	15720.8	422393	0	14143.4	394227	0
MUN18_M_D	3142.3	139359	0	3181.6	141465	0
MZZV11	92.5	40079	4	11.0	11385	0
MZZV42Z	6.8	9221	0	4.5	7561	0
NEMSPMM2	5.6	8003	482	4.7	8498	74
NEMSWRLD	65.9	33202	0	69.6	39666	0
NET12	1.9	1732	25	1.7	2026	0
NSCT2	1.5	5660	0	1.8	5988	0
NUG08	1.8	4974	0	1.6	4135	0
NUG12	369.5	107631	0	478.7	123744	0
OSA-30	2.8	2139	0	1.9	2327	0
OSA-60	9.0	4331	0	7.3	5002	0
P01	1.4	2972	0	1.8	4387	0
P02	51.5	19947	1575	34.3	17379	0
P03	3.8	1860	0	7.7	4906	0
P04	41.9	22550	6137	21.0	14263	0
P05	67.5	38081	1	56.6	38246	8
P06	12.6	28527	21	10.3	31164	6
P07	51.3	52730	1757	36.9	42338	0
P08	455.9	81204	0	31.3	35040	0
P09	5.5	7919	1	22.6	19943	0
P10	8.2	7979	264	1.8	7406	0
P11	6.0	17905	430	1.5	9823	0
P12	22.6	14183	423	8.2	7631	0
P13	195.7	49475	10218	52.3	22583	0
P14	92.3	44092	7485	4.1	7074	0
P15	21.8	3816	0	67.9	12565	0
P16	28.1	4370	0	93.6	14210	0

– continued on next page –

– continued from previous page –

MOPS 7.9	Original models			COIN LPP models		
results	Time	Iters	Phase 1	Time	Iters	Phase1
P17	3.9	2265	0	6.5	5033	0
P18	4.0	2284	0	7.1	5271	0
P19	251.4	65871	12706	19.5	10918	0
P20	45.6	26185	5855	30.2	22062	0
PDS-10	1.8	6833	0	1.0	5118	0
PDS-100	636.9	237628	0	319.1	151212	0
PDS-20	14.9	22682	0	7.1	13734	0
PDS-40	86.4	70158	0	37.8	42116	0
PILOT	3.8	3493	116	3.8	3796	0
PILOT87	31.5	11265	102	16.5	7035	0
PTV15	11.7	13564	0	8.8	13160	0
QAP12	281.5	80121	0	462.1	125538	0
RAIL4284	5944.9	56076	0	5865.1	56714	0
RAIL507	16.4	3405	0	12.4	2836	0
RAT5	1.9	2142	0	1.8	2137	0
RD-RPLUSC-21	17.8	210	1	0.4	179	0
SCFXM1-2R-256	28.3	35062	0	36.5	37667	0
SELF	183.3	8519	0	125.2	6024	0
SEYMOUR	3.5	4097	15	2.8	3664	0
SGPF5Y6	17.7	44023	12132	2.7	25767	0
SLPTSK	9.6	8300	7818	3.1	3868	3438
SOUTH31	30.2	18402	19	29.0	18264	0
SP97AR	1.3	1170	0	1.2	1159	0
STORMG2_1000	3464.4	506109	1000	3894.7	556389	0
STORMG2-125	38.9	62391	125	36.2	63794	0
STP3D	2769.1	149935	0	1993.3	122929	0
T0331-4L	23.8	5754	0	24.0	6488	0
T1717	11.5	3585	0	8.5	4401	0
ULEVIMIN	41.5	22282	1	18.0	15925	0
VAN	47.8	14198	0	14.1	8827	0
WATSON_1	238.2	145297	104811	140.9	137312	0
WATSON_2	1301.7	264235	249365	174.2	201317	10954
WORLD	97.5	33584	2	111.4	38704	97
Sum	38816.5	3816302	433351	33312.7	3376048	17320
Geo. Mean	23.0	–	–	15.7	–	–

Table A.5: Results with MOPS 7.9 Dual Simplex.

MOPS 7.9 (expanded bounds) results	Total Time	Total Iters	Degen. Iters	Phase 1 Iters
AA3	1.6	1976	0	0
AIR04	3.2	3784	0	0
ATLANTA-IP	28.7	13069	4034	0
BAS1LP	2.1	1399	608	0
BAXTER	3.1	4621	438	0
CO9	18.6	13678	2708	776
CQ9	12.5	12329	454	65
CRE-B	6.1	7571	518	99
CRE-D	2.7	5524	361	82
D2Q06C	3.1	5012	652	356
DANO3MIP	101.5	54574	34062	0
DBIC1	880.9	64485	57299	2447
DBIR1	1.6	1881	244	0
DEGEN4	14.3	8767	2	1813
DFL001	24	17444	37	54
DS	41.6	6402	10	0
EX3STA1	10.5	4230	3129	1093
FA	43.6	35776	1	150
FAST0507	17.9	5012	84	0
FIT2P	4.4	4880	0	0
FOME11	67	35146	134	58
FOME20	14.6	20758	222	1549
FXM4_6	7.9	26597	3446	9165
GEN4	2.6	619	612	0
GREENBEA	0.9	2837	19	1361
GREENBEB	1.9	5032	245	227
HAL_M_D	56.8	26554	109	0
JENDREC1	3.2	4377	0	2590
KEN-13	3.3	15745	0	0
KEN-18	31.8	54582	3	0
LP22	43.3	20049	183	0
LPL1	97.3	32842	2	0
LPL3	1.1	3977	0	0
MAROS-R7	3.8	2605	9	0
MOD2	148	39037	4853	85
MODEL10	97.2	53714	1322	352
MOMENTUM2	18.6	9176	3177	0
MOMENTUM3	914.2	75332	24536	0
MSC98-IP	31.8	18676	15692	0
MUN1_M_D	15702.6	419305	12796	0
MUN18_M_D	3430.3	143738	5220	0
MZZV11	10.9	11174	844	0
MZZV42Z	6.6	9221	961	0
NEMSPMM2	7.2	8145	1503	728
NEMSWRLD	70.3	26783	736	133
NET12	1.5	1708	32	0
NSCT2	1.5	5740	30	0
NUG08	1.4	4016	54	0
NUG12	278.9	82180	7718	0
OSA-30	23.9	2834	30	0
OSA-60	82.4	5845	94	0
P01	2.1	3754	1330	0
P02	48.2	19394	77	5463
P03	3.7	1860	1748	0
P04	99.1	48999	49	10490
P05	67.7	38081	23	1
P06	16	30825	2145	71
P07	52.2	53345	22	1854
P08	500.7	84265	35598	0
P09	5.7	7919	1122	1
P10	8.4	7987	1	264
P11	6.3	18204	240	430
P12	33.3	19771	37	5254
P13	379.7	91668	208	12715
P14	47.7	23834	103	9198
P15	26.5	3901	3729	0
P16	30.5	4211	3968	0

– continued on next page –

– continued from previous page –

MOPS 7.9 (no bound red) results	Total Time	Total Iters	Degen. Iters	Phase 1 Iters
P17	3.8	2258	2165	0
P18	4.2	2275	2165	0
P19	282.1	68515	173	14100
P20	81.1	45522	6933	7788
PDS-10	2	6460	42	735
PDS-100	870.4	239575	5039	5760
PDS-20	14.4	20758	222	1549
PDS-40	126.2	66830	916	2830
PILOT	4.6	3888	47	232
PILOT87	33	11703	2633	175
PTV15	10.1	13432	0	0
QAP12	668.2	177774	16898	0
RAIL4284	5961.7	56076	1116	0
RAIL507	16.7	3405	119	0
RAT5	2.1	2142	0	0
RD-RPLUSC-21	18	210	5	1
SCFXM1-2R-256	38	33725	2560	1549
SELF	168	8591	3953	0
SEYMOUR	3.3	4097	41	15
SGPF5Y6	18	44023	27605	12132
SLPTSK	9.6	8300	0	7818
SOUTH31	30.1	18456	0	22
SP97AR	1.4	1170	108	0
STORMG2_1000	2766	506483	5237	2000
STORMG2-125	34	62497	648	250
STP3D	2773.4	149935	3092	0
T0331-4L	29.8	6687	5	0
T1717	13.4	4194	0	0
ULEVIMIN	41.5	22282	18	1
VAN	47.7	14198	3899	0
WATSON_1	342.6	153036	36051	122181
WATSON_2	1302.1	264235	92892	249365
WORLD	213.4	49603	5892	38
Sum	39643.8	3971111	460097	497465
Geom. Mean	25.6	–	–	–

Table A.6: Results with MOPS 7.9 Dual Simplex with expanded bounds after LP preprocessing.

Pan+SP results	CPU Time (secs)	Total Iters	Degen. Iters	Phase 1 Iters	Dual Iters	Primal Iters
CO9	18.4	12394	2032	77	12392	2
CQ9	16.2	13578	371	4	13574	4
CRE-B	7.8	7884	556	27	7884	0
CRE-D	4.2	6086	411	25	6086	0
D2Q06C	3.4	5502	905	77	5502	0
DBIC1	1099.5	70493	61692	1220	70493	0
DEGEN4	15.3	8452	0	1813	8452	0
DFL001	22.1	17734	48	30	17632	102
EX3STA1	18.4	5359	4229	1093	5359	0
FOME11	57.2	35660	112	24	35444	216
FOME12	163.5	76982	174	58	76747	235
FOME13	427.2	147110	567	151	146004	1106
FXM4_6	6.1	22644	911	1944	22644	0
GREENBEB	1.5	4035	113	81	4035	0
JENDREC1	3.2	4377	0	2590	4377	0
MOD2	78.0	29430	4602	1	29430	0
MODEL10	37.6	28388	321	1	28345	43
MZZV11	88.9	35361	9116	4	35361	0
NEMSPMM2	6.4	8033	1308	482	8033	0
NET12	1.7	1644	33	25	1644	0
P02	68.7	24223	140	1575	24223	0
P04	51.5	24479	47	6137	24479	0
P05	69.7	38081	23	1	38081	0
P06	12.5	28379	1199	21	28379	0
P07	50.2	52376	23	1757	52375	1
P09	8.1	8204	1485	1	8179	25
P10	8.6	7979	1	264	7979	0
P11	5.8	17861	90	430	17861	0
P12	16.3	9579	86	423	9579	0
P13	490.6	111176	291	10218	111175	1
P14	70.2	31423	58	7485	31422	1
P19	266.8	58964	95	12706	58954	10
P20	40.6	23495	3367	5855	23264	231
PILOT	5.0	3977	71	116	3870	107
PILOT87	29.0	9426	2553	102	9383	43
RD-RPLUSC-21	17.7	212	5	1	212	0
SEYMOUR	3.3	3771	61	15	3771	0
SGPF5Y6	16.9	44023	27605	12132	44007	16
SLPTSK	3.4	3745	123	7818	3741	4
SOUTH31	32.8	18465	0	19	18465	0
STORMG2_1000	3456.8	505824	5136	1000	505824	0
STORMG2-125	37.5	63210	659	125	63210	0
ULEVIMIN	58.3	23557	25	1	23557	0
WATSON_1	253.2	148074	32780	104811	144549	3525
WATSON_2	1272.4	258304	91946	249365	250026	8278
WORLD	98.8	33176	4907	2	33176	0
Sum	8521.1	2093129	260277	432107	2079179	13950
Geom. Mean	30.1	18602.6	–	164.9	18551.1	–

Table A.7: Results on Dual Phase 1: Combined Method ”Pan + Subproblem-Approach”.

Pan results	CPU Time (secs)	Total Iters	Degen. Iters	Phase 1 Iters	Dual Iters	Primal Iters
CO9	18.7	12394	2032	77	12392	2
CQ9	16.2	13578	371	4	13568	10
CRE-B	7.8	7884	556	27	7884	0
CRE-D	4.3	6086	411	25	6086	0
D2Q06C	3.4	5502	905	76	5502	0
DBIC1	1119.3	70493	61692	1220	70493	0
DEGEN4	15.3	8452	0	1615	8452	0
DFL001	22.2	17734	48	30	17651	83
EX3STA1	18.4	5359	4229	757	5359	0
FOME11	57.3	35660	112	24	35490	170
FOME12	163.6	76982	174	58	76718	264
FOME13	427.9	147367	795	151	146334	1033
FXM4_6	6.1	22644	911	1944	22644	0
GREENBEB	1.5	4035	113	81	4035	0
JENDREC1	3.2	4377	0	2590	4377	0
MOD2	78.0	29430	4602	1	29430	0
MODEL10	37.6	28388	321	1	28339	49
MZZV11	89.0	35361	9116	4	35361	0
NEMSPMM2	6.4	8033	1308	724	8033	0
NET12	1.7	1644	33	25	1644	0
P02	68.7	24223	140	1528	24223	0
P04	f	f	f	f	f	f
P05	70.2	38081	23	1	38081	0
P06	12.8	28379	1199	21	28379	0
P07	50.2	52376	23	1755	52375	1
P09	8.1	8204	1485	1	8171	33
P10	8.6	7979	1	264	7979	0
P11	5.8	17861	90	430	17860	1
P12	16.3	9579	86	457	9579	0
P13	f	f	f	f	f	f
P14	69.0	32285	125	1843	32284	1
P19	f	f	f	f	f	f
P20	233.2	227028	4326	207453	226400	628
PILOT	5.0	3977	71	116	3880	97
PILOT87	29.0	9426	2553	102	9405	21
RD-RPLUSC-21	17.8	212	5	1	212	0
SEYMOUR	3.3	3771	61	15	3771	0
SGPF5Y6	17.3	44023	27605	12132	44007	16
SLPTSK	3.4	3745	123	2813	3741	4
SOUTH31	32.9	18465	0	19	18465	0
STORMG2_1000	3456.4	505824	5136	1000	505824	0
STORMG2-125	37.5	63210	659	125	63210	0
ULEVIMIN	58.5	23557	25	1	23557	0
WATSON_1	282.6	133435	6548	93649	130124	3311
WATSON_2	2241.0	233271	11875	215752	223392	9879
WORLD	98.8	33176	4907	2	33176	0

Table A.8: Results on Dual Phase 1: Pan's method.

SP results	CPU Time (secs)	Total Iters	Degen. Iters	Phase 1 Iters	Dual Iters	Primal Iters
CO9	21.0	12935	2455	82	12935	0
CQ9	18.2	13728	359	4	13720	8
CRE-B	8.2	7885	517	23	7885	0
CRE-D	4.0	5594	352	21	5594	0
D2Q06C	3.8	5641	931	90	5641	0
DBIC1	1700.4	107282	84034	12679	107282	0
DEGEN4	13.7	7552	1	851	7552	0
DFL001	23.7	18922	81	17	18804	118
EX3STA1	31.2	6979	6913	4000	6979	0
FOME11	64.1	38427	60	22	38315	112
FOME12	154.8	73181	182	46	72911	270
FOME13	430.3	147132	530	111	146351	781
FXM4_6	6.5	23234	1245	2363	23234	0
GREENBEB	1.3	3596	93	80	3596	0
JENDREC1	3.1	4193	0	2182	4193	0
MOD2	75.9	29029	4536	10	29029	0
MODEL10	37.5	28006	234	1	27955	51
MZZV11	69.0	29296	7203	11	29296	0
NEMSPMM2	6.7	7689	1497	278	7689	0
NET12	1.6	1454	37	27	1454	0
P02	65.6	22846	82	4790	22846	0
P04	125.3	57391	267	7365	57391	0
P05	79.2	38252	19	1	38252	0
P06	13.1	28389	1223	21	28389	0
P07	54.4	53199	22	1734	53198	1
P09	9.2	9752	1690	1	9678	74
P10	9.3	8082	1	264	8082	0
P11	6.1	17868	99	430	17867	1
P12	15.9	9515	66	753	9515	0
P13	514.1	119070	1009	14984	119070	0
P14	68.7	29510	60	7559	29509	1
P19	166.3	39116	96	7681	39106	10
P20	32.6	17940	3276	3306	17554	386
PILOT	4.8	3769	58	114	3703	66
PILOT87	28.3	9517	2671	275	9489	28
RD-RPLUSC-21	18.6	212	5	1	212	0
SEYMOUR	3.5	4114	45	15	4114	0
SGPF5Y6	19.4	53484	37781	22137	53426	58
SLPTSK	3.4	3974	340	3215	3970	4
SOUTH31	33.6	18363	0	14	18363	0
STORMG2_1000	3491.2	507286	5255	1000	507286	0
STORMG2-125	36.8	62252	642	125	62252	0
ULEVIMIN	57.7	22672	21	1	22672	0
WATSON_1	245.7	147753	34608	107433	144288	3465
WATSON_2	1376.3	270799	103393	254999	262236	8563
WORLD	101.1	33509	4890	11	33509	0
Sum	9254.9	2160389	308879	461127	2146392	13997
Geo Mean	31.3	18910.7	–	195.1	18857.7	–

Table A.9: Results on Dual Phase 1: Minimization of the sum of dual infeasibilities, subproblem approach.

CM results	CPU Time (secs)	Total Iters	Degen. Iters	Dual Iters	Primal Iters
CO9	20.5	12879	2138	12867	12
CQ9	15.4	13022	352	13011	11
CRE-B	7.7	7713	571	7684	29
CRE-D	4.2	5682	505	5671	11
D2Q06C	3.7	6225	1044	5507	718
DBIC1	216.7	34056	22143	5824	28232
DEGEN4	61.0	36062	28679	10084	25978
DFL001	24.8	19459	32	19400	59
EX3STA1	13.3	3791	3672	0	3791
FOME11	63.9	38181	207	37904	277
FOME12	159.8	75132	146	74893	239
FOME13	458.4	152376	703	151426	950
FXM4_6	5.9	21584	2971	21584	0
GREENBEB	1.5	4471	399	3608	863
JENDREC1	2.6	3881	0	3148	733
MOD2	74.6	28718	4572	28718	0
MODEL10	38.6	28342	277	28289	53
MZZV11	63.8	30204	6370	29626	578
NEMSPMM2	6.8	7524	1677	7063	461
NET12	1.2	1943	1191	553	1390
P02	67.8	25188	11633	19284	5904
P04	65.4	29306	7490	29306	0
P05	72.9	38055	24	38055	0
P06	13.0	28336	1235	28328	8
P07	62.7	55016	1062	53473	1543
P09	5.2	12112	6229	477	11635
P10	10.1	8306	217	7940	366
P11	7.2	18483	334	17814	669
P12	23.3	15122	3706	11830	3292
P13	151.8	39728	17152	29486	10242
P14	25.2	13590	5423	7854	5736
P19	149.3	31506	8306	26833	4673
P20	101.3	100286	40490	1846	98440
PILOT	8.2	7081	822	3694	3387
PILOT87	42.2	14404	5270	13936	468
RD-RPLUSC-21	18.3	210	6	209	1
SEYMOUR	3.3	3846	47	3831	15
SGPF5Y6	57.9	51516	47191	44598	6918
SLPTSK	2.1	3272	639	3140	132
SOUTH31	33.6	18461	0	18458	3
STORMG2_1000	3477.8	505322	5219	505322	0
STORMG2-125	38.9	62747	663	62747	0
ULEVIMIN	180.6	49992	27992	34507	15485
WATSON_1	1606.9	175711	138869	65534	110177
WATSON_2	4054.3	311752	236991	147972	163780
WORLD	102.3	32998	4907	32998	0
Sum	11625.8	2183591	649566	1676332	507259
Geo Mean	32.7	19729.6	–	–	–

Table A.10: Results on Dual Phase 1: Cost modification + Primal Simplex.

Version	Added Implementation Technique / Modification
Ver. 1	First Version
Ver. 2	Dual Steepest Edge
Ver. 3	Sparse Pivot Row α^r
Ver. 4	Sparse ρ
Ver. 5	Vector of Primal Infeasibilities
Ver. 6	Bound Flipping Ratio Test
Ver. 7	Numerical Stability
Ver. 8	Hypersparse FTran,BTran
Ver. 9	Tight bounds after LP Preprocessing
Ver. 10	Revised Dual Steepest Edge
Ver. 11	Hypersparse LU-Update
Ver. 12	Randomized Partial Pricing

Table A.11: Progress in our dual simplex code: implementation techniques in chronological order.

	Ver. 1	Ver. 2	Ver. 3	Ver. 4	Ver. 5	Ver. 6	Ver. 7	Ver. 8	Ver. 9	Ver. 10	Ver. 11	Ver. 12
CRE-B.STA	26.56	35.78	16.20	19.17	18.47	21.59	15.76	18.56	13.80	14.86	12.48	11.78
CRE-D.STA	18.94	31.44	12.00	9.11	9.08	9.50	7.68	6.78	6.55	5.62	5.70	4.91
D2Q06C.STA	11.03	12.50	12.81	12.98	12.92	13.20	12.78	12.97	8.38	5.80	5.53	5.77
DFL001.STA	72.69	72.27	73.69	70.58	61.53	67.53	63.11	60.20	64.58	32.47	33.61	35.26
FIT2D.STA	10.31	10.80	12.31	11.02	10.97	1.28	1.27	1.25	1.25	0.97	0.97	1.00
FIT2P.STA	17.25	22.11	15.39	16.33	15.11	12.48	11.53	12.47	12.58	9.52	9.41	9.38
KEN-11.STA	22.75	14.86	7.53	7.53	4.66	3.59	3.32	2.93	2.10	1.67	1.41	1.25
KEN-13.STA	194.53	112.41	75.33	87.94	46.14	22.74	19.91	11.79	13.36	9.42	7.39	6.27
KEN-18.STA	4856.25	2268.62	1889.39	1720.76	860.84	371.91	306.48	159.87	158.46	112.05	87.16	59.44
MAROS-R7.STA	5.28	6.48	6.09	6.11	6.70	7.78	7.11	7.03	7.22	7.11	7.92	6.81
OSA-14.STA	4.41	7.25	9.80	8.41	9.23	10.58	14.63	13.17	3.56	2.09	2.47	2.17
OSA-30.STA	15.56	29.56	43.86	39.89	43.81	49.95	55.11	54.65	11.34	5.80	5.95	6.42
OSA-60.STA	71.72	165.84	132.69	123.48	124.38	141.03	168.39	166.78	21.93	19.31	21.33	20.67
Sum	5327.28	2789.92	2307.09	2133.31	1223.84	733.16	687.08	528.45	325.11	226.69	201.33	171.13

Table A.12: Chronological progress: solution time.

	Ver. 1	Ver. 2	Ver. 3	Ver. 4	Ver. 5	Ver. 6	Ver. 7	Ver. 8	Ver. 9	Ver. 10	Ver. 11	Ver. 12
CRE-B	8624	8605	7561	8200	8200	8132	7277	7453	8992	7929	7518	7412
CRE-D	7561	8800	6551	5850	5958	5646	5020	4959	5727	5289	5215	5061
D2Q06C	10746	10455	10840	10727	10427	10168	9743	9751	7785	5829	5504	5473
DFL001	41542	28590	31635	30086	26999	26129	25566	24510	29228	16670	16930	17855
FIT2D	6324	6207	6306	5738	5606	251	256	256	256	142	142	170
FIT2P	14202	14761	14925	15334	12593	6939	6934	6699	6699	4862	4824	4761
KEN-11	14663	9470	9427	9596	9772	8046	8100	8127	8042	7608	7609	7609
KEN-13	51746	24644	24331	25879	23244	16950	16878	16648	16965	15878	15893	15630
KEN-18	243349	105354	108610	101476	91279	63629	61378	60708	60537	53800	53652	53069
MAROS-R7	2441	2462	2462	2462	2603	2738	2656	2656	2656	2502	2770	2591
OSA-14	966	1363	1344	1194	1382	1383	1529	1462	1211	1068	1064	1041
OSA-30	1941	2563	2709	2636	2836	2821	2728	2804	2529	2134	2158	2157
OSA-60	3825	5686	5387	5177	5593	5532	5777	5777	4526	4347	4427	4425
Sum	407930	228960	232088	224355	206492	158364	153842	151810	155153	128058	127706	127254

Table A.13: Chronological progress: total iteration count.