On the theory and applications of flexible anti-cycling index selection rules for linear optimization problems

Adrienn Nagy

PhD Thesis

Supervisor: Tibor Illés
Associate Professor, PhD

Supervisor: Margit Kovács
Candidate of Math. Science

Eötvös Loránd University of Sciences, Institute of Mathematics,
Doctoral School

Director of doctoral school:
Professor Miklós Laczkovich
member of the Hungarian Academy of Sciences

Applied mathematics doctoral program

Director of program:
Professor György Michaletzky
Doctor of Sciences

This thesis was written at the Department of Operations Research at the Eötvös Loránd University of Sciences.

Budapest
May, 2014

Contents

Acknowledgements

I would like to thank my supervisors, dr. Tibor Illés and dr. Margit Kovács for their continuous support and guidance through the last many years. I'm grateful for all my teachers for their advice and suggestions through my years at the University. I would like to give special thanks to dr. Zoltán Csörnyei who has never lost the faith that I will complete my doctoral studies, and not allowed me to lose faith either. I owe thanks to my bosses, Mark Isaacs and Chris Oakley for making it possible to fit my doctoral studies and tasks with my everyday work through their flexibility and support. The greatest thanks go to my father who always stood by my side and supported me in no matter what my decisions was, even when I decided that I will not continue my doctoral studies. A special thanks is due to dr. Zsolt Csizmadia and dr. Tibor Illés who has almost invisibly guided me back to the path, in such a way that by the time I have noticed it was too late, and I become more dedicated the my chosen field of science than ever before.

Köszönetnyilvánítás

Ezúton szeretném a köszönetemet kinyilvánítani témavezetőimnek, dr. Illés Tibornak és dr. Kovács Margitnak az elmúlt években való támogatásért és útmutatásért. Köszönettel tartozom még az összes tanáromnak, akik tanácsaikkal, javaslataikkal segítettek az egyetemi éveim alatt. Külön kiemelném dr. Csörnyei Zoltánt, aki soha nem adta fel a reményt, hogy egyszer eljutok idáig, és nem hagyta hogy én magam feladjam. Köszönet illeti a főnökeimet, Mark Isaacs-t és Chris Oakley-t, hogy rugalmas és támogató hozzáállásukkal lehetővé tették, hogy a doktori tanulmányokat és elkötelezettségeket a mindenapi munkával összeegyeztethessem. Legnagyobb köszönettel azonban édesapámnak tartozom, aki az évek során mindig mellettem áll, támogatott minden döntésemben. Még azokban a pillanatokban is, mikor doktori tanulmányaim megszakítása mellett döntöttem. Külön köszönet illeti dr. Csizmadia Zsoltot és dr. Illés Tibort, akik olyan észrevétlenül tereltek vissza az útra, hogy mire feleszméltem mi történt, már késő volt és csak még jobban eme tudományterület elkötelezett hívévé váltam.

Chapter 1

Introduction

This chapter provides a short summary to the topics covered by this thesis, includes a short historical overview, and establishes the contributions of the thesis.

1.1 Overview and historical review

Linear programming is possibly the most often cited methodology of operations research. The roots of linear programming go back to the 19th century: for example the Fourier-Motzkin elimination, Farkas' lemma, Minkowski's theorem. For the development and applicability of linear programming, the simplex algorithm invented by Dantzig had been a fundamental result, which he published in 1947. For their models based on linear programming and their application in economics, Kantorovich and Koopmans received a Nobel price in 1975.

Today, it is the golden age for applying linear programming. Each year, their are a large number of new results and applications using linear programming are published. There are several factors for the success of the application of linear programming, like the invention of numerically efficient methods, the wide and cheap availability of powerful hardware, the easily accessible low cost data, and the increasing number of operations research educated professionals and analysts.

The simplex method is still the basis of many optimization methods (either directly, or embedded say in a mixed integer programming problem (MIP) search). The primal simplex method moves along the edges of the corresponding polyhedron from vertex to vertex (feasible basis solution). Since Dantzig's invention several alternative pivot methods have been invented, like the dual variant, the monotonic build-up simplex methods of Anstreicher and Terlaky from 1994, or the criss–cross method by Terlaky from 1985; these methods have fundamentally different theoretical and practical behaviour.

The simplex method is known to be able to cycle on some problems, i.e. may be able to return to the same basis it has already visited before after a finite number of pivots. For the simplex type methods where an objective is monotone, cycling may happen on so called degenerate problems where the same vertex is described by multiple feasible solutions. For handling cycling, at least in the theoretical variants, many pivot methods rely on secondary index selection rules like the minimal index or most often selected variable rules. For the simplex variants where the objective is monotone, finiteness can also be achieved by the means of perturbation or lexicography (these two are known to be equivalent under specific conditions). In fact, the application of perturbation had been the method of choice for arguing for the finiteness of pivot methods in most early publications.

The application of an index selection rule presents significant restrictions to a numerical implementation of the methods, and most often numerical stability and efficiency considerations are preferred. In most numerical implementations the presence of numerical round off error serves as natural perturbation and helps to break out of any cycle [49]. It is important to note, that adding small, artificial perturbation to a problem may help to ensure that cycling does not occur by enforcing an improvement on the objective for all pivots. However, doing so also makes it necessary to have a final clean up phase of the method, where any such perturbation is removed (otherwise it wouldn't solve the same problem). Although this is not a problem in most cases, it may introduce a secondary level of cycling.

Leonid Khachiyan published his ellipsoid method variant in 1979 (which is a specialization of the method of Naum Z. Shor originally developed for nonlinear optimization problems). The number of iterations needed by the ellipsoid method can be bounded by a polynomial of the bit length of its input data and the number of rows and variables in the problem. The simplex method is not known to have this important theoretical complexity property. While most simplex variants are efficient in practice, with an observed expected number of necessary pivot operations being around $O(n + m)$, the ellipsoid method which is efficient in theory has an impractical numerical efficiency as the theoretical and observed complexity are close to each other, $O(n^8)$. This renders the ellipsoid method unusable as the problem size grows.

Narendra Karmarkar published his projective scaling interior point method in 1984. Karmarkar's method is a polynomial method for solving linear programming problems which is efficient in theory and is also the basis of many interior point method implementations. Following the publications of his results, in the next 2 decades there were several thousand publications for both linear and convex optimization problems, inspired by his method, revolutionizing the theory and practice of both linear and convex optimization. Typical professional implementations of the interior point method for linear programming do not

require more than 50 iterations to solve a problem. An iteration count above 500 is rare.

Despite the theoretical complexity superiority and its efficiency in practice, interior point methods are still do not dominate the solution methods of linear optimization problems, but rather share it with simplex methods. This is due to the favourable warm starting properties of simplex methods, which makes a difference when solving mixed integer problems (the dual method) or column generation problems (the primal method). As interior point methods return an approximate solution near the analytic centre, a basis solution returned by a simplex method tends to have a simpler structure which is important in certain applications.

The thesis considers pivot algorithms for the standard linear programming, the linear complementarity problem and the quadratic linear programming problem.

Starting from the linearly constrained quadratic optimization problem, the Karush-Kuhn-Tucker conditions give a special case for the associated *linear complementarity problem*. The general linear complementarity problem has received significant research attention, and has been a popular field of research, having a diverse field of applications. To solve linear complementarity problems, the first algorithms were based on pivot methods. Among the most well known pivot methods for the linear complementarity problems are *Lemke–* [48] and the criss–cross algorithm [41]. An algorithm from Terlaky [61] works without the need to grow the pivot tableau (i.e. works on the original quadratic problem rather than the associated Karush-Kuhn-Tucker system).

Solving linear complementarity problems is closely related to the properties of the matrix of the problem. An interesting direction for research has been to identify what are those matrices and matrix properties, for which the criss–cross method solves the linear complementarity problem in a finite numbers of pivots. The first results in this field were related to the so called *bisymmetric matrices* – the class of linear complementarity problems that we get when we formulate the Karush-Kuhn-Tucker conditions for a quadratic optimization problem – for which the criss–cross algorithm is finite if anti-cycling index selection rules are applied [4, 41, 64].

One of the most important theoretical questions has been to identify the matrix properties necessary such that the corresponding linear complementarity problem is a convex optimization problem and that the pivot algorithms could solve it in a finite number of steps (pivots). It has been shown that the class of *sufficient* matrices introduced by [16] is sufficient for the solution space to be convex, and it was later proven that sufficient matrices are exactly those for which the criss–cross method can solve (or prove infeasibility) the linear complementarity problem with an arbitrary right-hand-side vector in a finite number of iterations. It is natural to investigate what happens in the case when we have no a-priori

information about the properties of the matrix. Fukuda and his co-authors were the first to develop [26] such a variant of the criss–cross method that behaves exactly the same as the original algorithm [23] for problems with a sufficient matrix, though for problems with a matrix for which sufficiency does not hold it either still solves the problem or it provides in a finite number of steps an easy to verify proof that the matrix provided is not sufficient. Csizmadia and Illés [18] have shown that the criss–cross method can be extended to several alternative index selection rules to work – in the same sense as for the Fukuda et al. version – for solving general linear complementarity problems. Recently, Csizmadia et al. has introduced a class of index selection rules – the so called *s-monotone index selection rules* – that incorporate these index selection rules and provide a family of criss–cross variants with the above extension [19]. A more detailed survey on the related matrix classes and the s-monotone index selection rule can be found in the PhD thesis of Csizmadia [17].

For the solution of linear complementarity problems, primal-dual interior point approaches are widespread. Primal-dual methods are used to solve linearly constrained quadratic problems by the means of iterating following the central path. The central path problems solve for a smaller and smaller centrality parameter for each iteration. The termination criteria for the primal dual interior point methods is the duality gap to shrink beneath an a priory given $\varepsilon > 0$ value. In such cases, we say that the interior point method has produced a ε-*optimal* solution.

The existence and uniqueness of the central path [43] is central to the workings of the primal-dual interior point methods. Accurate solutions can be created by the method of Illés et al. [37].

When comparing pivot methods to interior point algorithms, our expectations are that for the most important matrix classes (positive definite, bisymmetric or sufficient), when working with a linear complementarity problem given by rational numbers and vectors, we expect the interior point algorithms to be efficient (i.e. to have a polynomial bound on their iteration count) in theory as well.

For linear complementarity problems arising from linearly constrained quadratic programming problems, Kojima et al. has generalized an existing, small-step primal-dual interior point algorithm [44] for which a polynomial iteration count could be proved [45]. For linear complementarity problems given by bisymmetric matrices, Kojima et al. proved the polynomial iteration complexity by means of an inequality derived from the positive semidefinite part of the objective in the underlying quadratic optimization problem. The question naturally arises, if there are matrix classes with similar properties that would confirm to the same proof approach based on this inequality and yield a polynomial complexity. Kojima and

his co-authors answered this question [43] by the introduction of the P*(κ) matrices, where $\kappa \geq 0$ is a real, pre-determined parameter. The class of P*(κ) matrices can be considered as a possible generalization of positive semi-definite matrices that also have the important property that the central path corresponding to the linear complementarity problem defined by the matrix has a unique solution for any $\mu > 0$ centrality parameter.

The definition of P* matrices, which is the union of all P*(κ) matrices – where κ is taken to be all non-negative numbers – creates the connection between the P*(κ) matrices and sufficient matrices, as Väliaho has shown that "*P*(κ) matrices are just sufficient*" [65], and the opposite inclusion has been shown by Cottle and Guu [14, 15] and Kojima et al. [43], showing that the two matrix classes are equal.

For linear complementarity problems with sufficient matrices, there is active research even today, and new algorithms or proofs related to existing methods are still being published. As an example, we mention two results [38, 36] that play an important role in the extension to the solvability of linear complementarity problems. One of the interesting questions had been if an interior point algorithm could be created that works in a similar way to Fukuda et al.'s in the sense that for problems with a sufficient matrix they work exactly like the former interior point methods [38, 36], while when the matrix is not sufficient, they either solve the problem *in polynomial time* or provide a proof that the matrix is not sufficient. The first detailed study related to this has been presented in the PhD thesis of Nagy [56].

For the latest publications related to pivot methods for linear complementarity problems, it can be said that typically they consider linear complementarity problems given by sufficient matrices.

1.2 The structure and contributions of the thesis

The first chapter has presented a introduction to linear optimization and has provided a historical overview.

The thesis is organized around flexible index selection rules for the linear feasibility, the linear programming, the linear complementarity, and the linearly constrained convex quadratic programming problems. The analyses of flexible index selection rules are brought into a common framework by applying the concept of **s**-monotone index selection rules [17, 19].

The thesis contains contributions to both the theory, and the computational aspects of **s**-monotone index selection rules; for algorithms for which finiteness proof already existed,

the thesis presents numerical studies, while for some pivot algorithms it presents novel proofs of finiteness.

The second chapter fixes the notations used throughout the thesis, and defines the various problem classes including the linear programming problem, the linear complementarity problem and the convex quadratic programming problem with linear constraints.

The third chapter presents the basic versions of the pivot algorithms considered by the thesis: the primal simplex method, the primal monotonic build-up simplex method, the criss–cross method for linear programming and the criss–cross method for the linear complementarity problem, and finally the primal simplex method for the quadratic programming problem. An example is provided for all methods. The chapter is closed by presenting the **s**-monotone index selection rules, providing several examples including the minimal index, last-in-first-out and most-often-selected rules and some generalizations.

The forth chapter introduces a testing framework and the major data sets used in the thesis. Comparing pivot algorithms and flexible index selection rules is not an easy task as implementation details can easily bias the result dramatically. To address the issue, the thesis proposes a framework that aims to minimize the effect of any algorithmic specific feature when comparing the effectiveness of pivot algorithms and flexible index selection rules. The framework is published in [34]. Two implementations are used for the numerical study: one in Matlab used to solve smaller instances, and one using the linear algebra of FICO Xpress that is used to solve a large set of problems from public benchmark sets. Some implementation details and a summary to the format of the public benchmark is given.

The fifth chapter focuses on the pivot methods for linear programming. For the sake of completeness, the finite proofs with **s**-monotone index selection rules of the primal simplex method and the monotonic build-up simplex method are summarized. The traditional criss–cross method for the linear programming problem is shown to be finite when **s**-monotone index selection rules are used [54]. The computational effectiveness of the flexible index selection rules is demonstrated using a Matlab implementation of the algorithm testing smaller test instances, and an implementation based on the linear algebra of FICO Xpress (through the public API) solving several larger instances from various well known public benchmarks [34].

The sixth chapter looks at the criss–cross method for the linear complementarity problem. The proofs presented for the finiteness when **s**-monotone rules are applied for problems with sufficient matrices, in the thesis a new, more general proof is presented: the proofs do not require the minimality of the cycling example. Although a small generalization (and in theory equivalent to the original [20]). A Matlab implementation of the EP-theorem

like generalized LCP criss–cross algorithm is used to solve linear complementarity problems arising in market equilibrium and bimatrix game problems; these problems yield matrices for the linear complementarity problem formulation for which the matrix is not sufficient; the numerical results are part of [20].

The final, seventh chapter of the thesis considers the primal simplex method for linearly constrained convex quadratic programming problems. Finiteness of the algorithm is proven when **s**-monotone index selection rules are applied. The proof is rather general: it shows that any index selection rule that only relies on the sign structure of the reduced costs / transformed right hand side vector and for which the traditional primal simplex method is finite is necessarily finite as well for the primal simplex method for linearly constrained convex quadratic programming problems. The proof is based on the careful analysis of a cycling situation and is based on showing that a cycling example necessarily includes a subproblem that behaves like a traditional linear programming problem. To the best knowledge of the author, finiteness of the primal simplex method for linearly constrained convex quadratic programming problems has only been published before using perturbation arguments. The result has been published in [33, 35].

The thesis is closed by a short summary.

Chapter 2

Notations and problem definitions

In this chapter, we summarize the notations used in this thesis, state the basic form of the linear optimization problems and their extensions that are considered, and for the sake of completeness also state some well known, fundamental results.

Throughout the thesis, matrices are denoted by italic capital letters, vectors by bold, scalars by normal letters and index sets by capital calligraphic letters. Columns of a matrix are indexed as a subscript while rows are indexed by superscripts. $A \in \mathbb{R}^{m \times n}$ and $M \in \mathbb{R}^{n \times n}$ denotes the original problem matrix for various problem types, while $\mathbf{b}$ and $\mathbf{q}$ the right hand sides and $\mathbf{c}$ the objective vector.

As all problems considered in the thesis are linearly constrained, we can assume without loss of generality that the linear constraint matrices are of full rank, i.e. $rank(A) = m$. Should this assumption not hold, we can eliminate (or show infeasibility) any redundant constraint by the application of the Gauss-Jordan elimination. A regular $m \times m$ rectangular submatrix of the constraint matrix is called a *basis*.

2.1 The linear programming problem

Let $A \in \mathbb{R}^{m \times n}$, $\mathbf{c} \in \mathbb{R}^n$, $\mathbf{b} \in \mathbb{R}^m$ be a matrix and vectors of appropriate dimensions, then

$$\min \mathbf{c}^T \mathbf{x} \tag{2.1}$$

$$A\mathbf{x} = \mathbf{b} \tag{2.2}$$

$$\mathbf{x} \geq \mathbf{0} \tag{2.3}$$

is a *primal linear programming (P-LP) problem*, while the *dual linear programming (D-LP) problem* can be defined as follows

$$\max \mathbf{b}^T \mathbf{y} \tag{2.4}$$

$$A^T \mathbf{y} \leq \mathbf{c} \tag{2.5}$$

where $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$ are primal and dual decision vectors, respectively.

For a given basis B, we denote the nonbasic part of A by N; the corresponding set of indices for the basis and nonbasic part are denoted by $\mathcal{I}_B$ and $\mathcal{I}_N$ respectively. The corresponding short pivot tableau for B is denoted by $T := B^{-1}N$, while the transformed right hand side and objective is denoted by $\overline{\mathbf{b}} := B^{-1}\mathbf{b}$ and $\overline{\mathbf{c}} := \mathbf{c}^T B^{-1}$. The algorithms in this thesis will refer to the rows and columns of the short pivot tableau corresponding to a given basis B as $\mathbf{t}_j := B^{-1}\mathbf{a}_j$ and $\mathbf{t}^{(i)} := (B^{-1}N)^{(i)}$ [40]. We will denote the individual coefficients of the pivot tableau as t_{ij}.

Let us associate to the problem the (primal) pivot tableau

A	$\mathbf{b}$
$\mathbf{c}^T$	$*$

and let us assume that A_B is an $m \times m$ regular submatrix of the matrix A, thus forming a basis of the linear system $A\,\mathbf{x} = \mathbf{b}$. In this case the (primal) basic pivot tableau associated with the (P-LP) problem and basis A_B is

$A_B^{-1}A$	$A_B^{-1}\mathbf{b}$
$\mathbf{c}^T - \mathbf{c}_B^T A_B^{-1} A$	$-\mathbf{c}_B^T A_B^{-1}\mathbf{b}$

The variables corresponding to the column vectors of the basis A_B are called *basic variables*. Now we are ready to define (column) vectors $\mathbf{t}^{(i)}$ and $\mathbf{t}_j$ with dimension $(n+2)$, corresponding to the (primal) basic tableau of the (P-LP) problem, where $i \in \mathcal{I}_B$ and $j \in \mathcal{I}_N$, respectively, in the following way:

$$(\mathbf{t}^{(i)})_k = t_{ik} = \begin{cases} t_{ik} & \text{if} \quad k \in \mathcal{I}_B \cup \mathcal{I}_N \\ \bar{b}_i & \text{if} \quad k = b \\ 0 & \text{if} \quad k = c \end{cases} \tag{2.6}$$

and

$$(\mathbf{t}_j)_k = t_{kj} = \begin{cases} t_{kj} & \text{if} \quad k \in \mathcal{I}_B \\ -1 & \text{if} \quad k = j \\ 0 & \text{if} \quad k \in (\mathcal{I}_N \setminus \{j\}) \cup \{b\} \\ \bar{c}_j & \text{if} \quad k = c \end{cases} \tag{2.7}$$

where b and c denotes indices associated with vectors $\mathbf{b}$ and $\mathbf{c}$, respectively. Furthermore, we define $\mathbf{t}^{(c)}$ and $\mathbf{t}_b$ vectors in the following way

$$(\mathbf{t}^{(c)})_k = t_{ck} = \begin{cases} \bar{c}_k & \text{if} \quad k \in \mathcal{I}_B \cup \mathcal{I}_N \\ 1 & \text{if} \quad k = c \\ -\mathbf{c}_B^T A_B^{-1}\mathbf{b} & \text{if} \quad k = b \end{cases} \tag{2.8}$$

and

$$(\mathbf{t}_b)_k = t_{kb} = \begin{cases} \bar{b}_k & \text{if} \quad k \in \mathcal{I}_B \\ -1 & \text{if} \quad k = b \\ 0 & \text{if} \quad k \in \mathcal{I}_N \\ -\mathbf{c}_B^T A_B^{-1}\mathbf{b} & \text{if} \quad k = c \end{cases} \tag{2.9}$$

and from now on we assume that c is always a basic index, while b is always a nonbasic index of the (P-LP) problem. The following result is widely used in the proofs of finiteness of simplex type methods.

Result 2.1.1 *[Orthogonality theorem, Klafszky and Terlaky, 1991] Let a (P-LP) problem be given, with rank$(A) = m$ and assume that $\mathcal{I}_{B'}$ and $\mathcal{I}_{B''}$ are two arbitrary bases of the problem. Then*

$$(\mathbf{t}''^{(i)})^T \mathbf{t}'_j = 0 \tag{2.10}$$

for all $i \in \mathcal{I}_{B''}$ and for all $j \notin \mathcal{I}_{B'}$.

The solvability of a linear complementarity problem is well described by the famous Farkas Lemma [24]. We use a form that is slightly different from the original form of the primal problem (which is the one favoured by the description of the simplex method) as this form is a little more intuitive for the Farkas Lemma. All forms are equivalent, and can be deduced from one another by simple manipulation of the stated forms.

Theorem 2.1.1 (Farkas-lemma) *Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ be given. Then exactly one of the following two systems has a solution:*

1. $\exists \mathbf{x} \in \mathbb{R}^n$ *such that* $A\mathbf{x} \leq \mathbf{b}$.
2. $\exists \mathbf{y} \in \mathbb{R}^m$ *such that* $A^T\mathbf{y} = \mathbf{0}$ *and* $\mathbf{y}^T\mathbf{b} < 0$.

In words, a linear system either has a solution, or there is an algebraic proof that a contradiction can be derived from it's defining constraints. A similarly famous result is known for characterizing the optimal solutions of a linear programming problem.

Theorem 2.1.2 (Duality theorem) *Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ be given. Assume that the*

$$\max \mathbf{c}^T\mathbf{x} \tag{2.11}$$

$$A\mathbf{x} \leq \mathbf{b} \tag{2.12}$$

system has an optimal solution. Then its dual pair

$$\min \mathbf{b}^T\mathbf{y} \tag{2.13}$$

$$A^T\mathbf{y} = \mathbf{c} \tag{2.14}$$

$$\mathbf{y} \geq \mathbf{0} \tag{2.15}$$

also has an optimal solution, and the two optimum are equal.

An intuitive interpretation of the duality theorem is that the maximum of the primal problem equals to the strictest constraint that can be deduced from its constraints that is parallel to the objective. The exact definition of the strong duality theorems are the following:

Theorem 2.1.3 (Weak duality theorem) *If $\mathbf{x}$ is a feasible solution of the primal problem and $\mathbf{y}$ is a feasible solution of the dual problem, then $\mathbf{c}^T\mathbf{x} \leq \mathbf{b}^T\mathbf{y}$.*

Theorem 2.1.4 (Strong duality theorem) *If there is a feasible solution to the primal problem and there is a feasible solution to the dual problem, then both problems have an optimal solution and the optimal values of the objective functions are equal.*

The following table shows the relationship of the primal and dual problems.

Primal/Dual	$\nexists$ feasible solution	$\exists$ optimal solution	unbounded problem
$\nexists$ feasible solution	$\surd$	x	$\surd$
$\exists$ optimal solution	x	$\surd$ (Strong duality)	x
unbounded problem	$\surd$	x	x (Weak duality)

In the table above, $\surd$ marks the combinations that can, and x marks the combinations that can't exist together.

Because of the weak duality theorem we know that there isn't such a problem where both the primal and the dual are unbounded. In this case if any of them has a solution, then it's objective function would be a constraint (bound) for the other objective function, so thus should be feasible.

The following problem is an example for that case when neither the primal nor the dual problem has a solution:

$$\begin{aligned} \max(2x_1 + x_2) \\ x_1 + x_2 = 2 \\ x_1 + x_2 = 1 \end{aligned}$$

Converting the previous problem to the following form

$$\begin{aligned} \max(2x_1^+ + 2x_1^- + x_2^+ + x_2^-) \\ x_1^+ - x_1^- + x_2^+ - x_2^- \leq \ \ 2 \\ -x_1^+ + x_1^- - x_2^+ + x_2^- \leq -2 \\ x_1^+ - x_1^- + x_2^+ - x_2^- \leq \ \ 1 \\ -x_1^+ + x_1^- - x_2^+ + x_2^- \leq -1 \\ x_1^+, x_1^- \geq \ \ 0 \\ x_2^+, x_2^- \geq \ \ 0 \end{aligned}$$

it is easy to see that the dual form of this problem is itself. Thus neither of the problems have a feasible solution.

Linear programming problems are possibly the most widely applied mathematical programming technique, either by itself or as part of a more complex algorithm (like in an integer program). Linear programming problems are considered to be relatively easily solvable; their theoretical complexity is polynomial (ellipsoid and interior point methods) and even the simplex algorithms that are not efficient in theory (they may have an exponential complexity, like for the primal simplex on the Klee-Minty cube [42]) are still very efficient in practice.

2.2 The linear complementarity problem

Linear complementarity problems are a generalization of linear programming problems, where the variables are organized into pairs, and the product of these pairs are required to be zero.

Let us consider the linear complementarity problem (P-LCP) in the standard form: find vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, such that

$$-M\mathbf{u} + \mathbf{v} = \mathbf{q}, \tag{2.16}$$

$$\mathbf{u}\ \mathbf{v} = \mathbf{0}, \tag{2.17}$$

$$\mathbf{u}, \mathbf{v} \geq \mathbf{0} \tag{2.18}$$

where $M \in \mathbb{R}^{n\times n}$, $\mathbf{q} \in \mathbb{R}^n$ and $\mathbf{u}\,\mathbf{v} = (u_1 v_1, \ldots, u_n v_n) \in \mathbb{R}^n$.

The (ordered) set of all indices is $\mathcal{I} := \{1, 2, ..., n, \bar{1}, \bar{2}, ..., \bar{n}\} \cup \{q\}$ which includes the index of the right hand side, and so $|\mathcal{I}| = 2n + 1$. The complementary pair of an index α is denoted by $\bar{\alpha}$. To simplify notation, $\overline{\bar{\alpha}} = \alpha$ for any index $\alpha \in \mathcal{I} \setminus \{q\}$, in other words the complementary index pair of $\bar{\alpha}$ is α.

Similarly to the case of the linear programming problem, for a given basis B, we denote the nonbasic part of M by N; the corresponding set of indices for the basis and nonbasic part are denoted by $\mathcal{I}_B$ and $\mathcal{I}_N$, respectively. For any given basis B we denote the corresponding short pivot tableau as $T := B^{-1}N$ where $N \in \mathbb{R}^{n\times n}$ is the non basic part of the $[-M, I] \in \mathbb{R}^{n\times 2n}$ matrix, while the transformed right hand side is denoted by $\overline{\mathbf{q}} := B^{-1}\mathbf{q}$. Individual coefficients of the short pivot tableau will be denoted by t_{ij}. The complementary variable pairs (u_l, v_l) will usually be referred to as variable pairs. When it is advantageous, the element wise Hadamard product will be emphasized by "$\cdot$".

A basis of the linear complementarity problem (2.16)-(2.18) is called *complementarity* if the so called complementarity conditions hold, i.e. $\mathbf{xz} = \mathbf{0}$ and $\mathbf{sy} = \mathbf{0}$. Using the new variables, we can also express complementarity as $\mathbf{uv} = \mathbf{0}$.

In general, the linear complementarity problems are NP-complete, as the complementarity conditions can be used to represent that a variable must be binary: let $u = 1 - v$, $0 \leq u \leq 1$, then $uv = 0$ is equivalent with $u \in \{0, 1\}$ making it easy to model most know NP-complete problem as general linear complementarity problems.

However, there are specific matrix classes, for which the linear complementarity problem is computationally tractable, and efficient algorithms to solve them exist. One of the most general such matrix classes is the class of sufficient matrices, that can be considered to be the generalizations of positive semi-definite matrices.

Definition 2.2.1 (Column sufficient matrix) *[13] A matrix $M \in \mathbb{R}^{n \times n}$ is* column sufficient *if no vector $\mathbf{x} \in \mathbb{R}^n$ exists, for which*

$$\begin{cases} x_i \left(M\mathbf{x}\right)_i \leq 0 & \text{holds } \forall i \in \{1, \ldots, n\} \\ x_j \left(M\mathbf{x}\right)_j < 0 & \text{for at least one } j \in \{1, \ldots, n\} \end{cases} \tag{2.19}$$

and it is called row sufficient *if its transposed version is column sufficient. Finally, the matrix M is* sufficient *if it is both column and row sufficient.*

If the matrix of a linear complementarity problem is sufficient, then it's solution set is convex [13]. Notice, that this means that the combinatorial structure of the solution set is limited compared to the original problem statement, as at least one variable of each complementarity pair will be zero in any solution. Sufficient matrices are also fundamental for the criss–cross methods, as those are the matrices for which the criss–cross method can always be applied and will be finite (if appropriate index selection rules are applied) [23].

For linear complementarity problems with sufficient matrices, an alternative theorem similar to the linear programming case exists. The dual of a linear complementarity problem is defined as

$$(\mathbf{x}, \mathbf{y}) \in V(M, \mathbf{q})^{\perp} := \left\{ (\mathbf{x}, \mathbf{y}) \mid \begin{array}{l} \mathbf{x} + M^T\mathbf{y} = \mathbf{0}, \;\; \mathbf{q}^T\mathbf{y} = -1 \\ \mathbf{xy} = \mathbf{0}, \;\; \mathbf{x}, \, \mathbf{y} \geq \mathbf{0} \end{array} \right\} \qquad (D - LCP)$$

We are ready to state the alternative theorem of linear complementary problems.

Theorem 2.2.1 *[27] For any sufficient matrix $M \in \mathbb{R}^{n\times n}$ and right hand side vector $\mathbf{q} \in \mathbb{R}^n$, exactly one of the following alternatives holds:*

(1) the primal LCP problem (P-LCP) has a feasible complementary solution $(\mathbf{u}, \mathbf{v})$,

(2) the dual LCP problem (D-LCP) has a feasible complementary solution $(\mathbf{x}, \mathbf{y})$.

The most frequent application of linear complementarity problems arise from convex quadratic optimization problems.

2.3 The quadratic linear programming problem

Since the beginning of the 1950s, the following *linearly constrained quadratic optimization problem (QP)* has received significant attention. For a quadratic optimization problem, the linear part of the objective is denoted by $\mathbf{c} \in \mathbb{R}^n$, while the quadratic matrix by $Q \in \mathbb{R}^{n\times n}$.

$$\begin{aligned} \min \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} \\ A\mathbf{x} \leq \mathbf{b} \\ \mathbf{x} \geq \mathbf{0} \end{aligned}$$

where $Q \in \mathbb{R}^{n\times n}$ and $A \in \mathbb{R}^{m\times n}$ are matrices and $\mathbf{c} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$ are vectors and $\mathbf{x} \in \mathbb{R}^n$ is the vector of unknowns. The *feasible region*

$$\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\} \subset \mathbb{R}^n \tag{2.20}$$

is a convex polyhedron and the objective $f : \mathcal{P} \to \mathbb{R}$ a quadratic function given in the

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T\mathbf{x} \tag{2.21}$$

form. A feasible solution $\mathbf{x}^* \in \mathcal{P}$ is called *optimal*, if

$$f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \text{holds for all } \mathbf{x} \in \mathcal{P}. \tag{2.22}$$

The *set of optimal solutions* is defined as

$$\mathcal{P}^* = \{\mathbf{x}^* \in \mathcal{P} : f(\mathbf{x}^*) \leq f(\mathbf{x}) \quad \text{holds for all } \mathbf{x} \in \mathcal{P}\}. \tag{2.23}$$

From as early as the 1960s, most research has concentrated on the versions of the convex optimization problem for which efficient algorithms and matching important practical application areas have been found. The class of efficiently tractable subclass has quickly been identified, for when Q is positive semi-definite matrix, the problem becomes a convex optimization problem.

For the solution of the linearly constrained, convex quadratic programming problem, the traditional simplex methods (first the primal then followed by the dual version) has been extended in the late 1950s, early 1960s, and several publications [69, 68, 67, 70, 72] made them widely recognised at the time. A common trait of these have been that finiteness was ensured by the means of *perturbation*.

When Q is positive semi-definite, the quadratic optimization problem (QP) is convex [22]. The Lagrange function associated with the problem [22] is as follows:

$$\begin{aligned} L &: \mathbb{R}^{m+n}_{\oplus} \to \mathbb{R} \\ L(\mathbf{x}, \mathbf{y}, \mathbf{z}) &= f(\mathbf{x}) + \mathbf{y}^T(A\mathbf{x} - \mathbf{b}) - \mathbf{z}^T\mathbf{x} \end{aligned} \tag{2.24}$$

Applying the convex Karush-Kuhn-Tucker theorem, we get that $\mathbf{x}^*$ is an optimal solution if and only if there exists $\mathbf{y}^* \in \mathbb{R}^m_{\oplus}, \mathbf{z}^* \in \mathbb{R}^n_{\oplus}$ such that $(\mathbf{y}^*, \mathbf{z}^*) \neq \mathbf{0}$ and $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ satisfies the

$$Q\mathbf{x} + c + A^T\mathbf{y} - \mathbf{z} = \mathbf{0} \tag{2.25}$$

$$A\mathbf{x} - \mathbf{b} \leq \mathbf{0} \tag{2.26}$$

$$\mathbf{y}^T(A\mathbf{x} - \mathbf{b}) = \mathbf{0} \tag{2.27}$$

$$\mathbf{z}^T\mathbf{x} = 0 \tag{2.28}$$

$$\mathbf{x}, \mathbf{y}, \mathbf{z} \geq \mathbf{0} \tag{2.29}$$

system. Introducing the $\mathbf{s} \in \mathbb{R}^m_{\oplus}$ variable, the above system yields the following criteria for our convex optimization problem:

$$-Q\mathbf{x} - A^T\mathbf{y} + \mathbf{z} = \mathbf{c} \tag{2.30}$$
$$A\mathbf{x} + \mathbf{s} = \mathbf{b} \tag{2.31}$$
$$\mathbf{y}^T\mathbf{s} = 0 \tag{2.32}$$
$$\mathbf{z}^T\mathbf{x} = 0 \tag{2.33}$$
$$\mathbf{x}, \mathbf{s}, \mathbf{y}, \mathbf{z} \geq \mathbf{0} \tag{2.34}$$

The same expressed in a matrix form, we get the bisymmetric linear complementarity problem associated with the linearly constrained convex quadratic optimization problem:

$$\begin{pmatrix} \mathbf{s} \\ \mathbf{z} \end{pmatrix} - \begin{pmatrix} -A & 0 \\ Q & A^T \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{c} \end{pmatrix} \qquad (B - LCP)$$

Linear complementarity problems in the above form are often referred to as (B-LCP) to refer to that M is bisymmetric. The problem still conforms to the standard LCP conditions:

$$\mathbf{y}^T\mathbf{s} = 0 \tag{2.35}$$
$$\mathbf{z}^T\mathbf{x} = 0 \tag{2.36}$$
$$\mathbf{x}, \mathbf{s}, \mathbf{y}, \mathbf{z} \geq 0 \tag{2.37}$$

The matrix of the linear complementarity problem associated with the linearly constrained quadratic optimization problem is denoted by $M \in \mathbb{R}^{K \times K}$ where $K = n + m$. The full right hand side vector of the linear complementarity problem consisting of both the original right hand side $\mathbf{b}$ and linear objective $\mathbf{c}$ is denoted by $\mathbf{q} \in \mathbb{R}^{n+m}$ (just as the right hand side of a general LCP problem). To summarize, the linearly constrained convex quadratic optimization problem is equivalent to the following linear complementary problem: we need to find vectors $\mathbf{u}$ and $\mathbf{v}$ such that

$$-M\mathbf{u} + \mathbf{v} = \mathbf{q} \tag{2.38}$$
$$\mathbf{u}^T\mathbf{v} = 0 \tag{2.39}$$
$$\mathbf{u}, \mathbf{v} \geq 0 \tag{2.40}$$

holds, where

$$M = \begin{pmatrix} Q & A^T \\ -A & 0 \end{pmatrix}, \quad \mathbf{q} = \begin{pmatrix} \mathbf{c} \\ \mathbf{b} \end{pmatrix} \quad \text{where} \quad \mathbf{v} = \begin{pmatrix} \mathbf{z} \\ \mathbf{s} \end{pmatrix} \quad \text{and} \quad \mathbf{u} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}.$$

Due to (2.40) and (2.39) we know that $v_j u_j = 0$, (j = 1,. . . ,m+n). Note that in the definition of the M matrix, for the sake of notation we have swapped the order of rows.

The matrix of a linear complementarity problem arising from the Karush-Kuhn-Tucker conditions of a convex quadratic optimization problem is called a bisymmetric matrix, and has several known properties that are useful both in practice and theory [63].

A summary of the weak and strong duality theorems, the optimality conditions and solving methods / algorithms for the convex quadratic optimization problem can be found in the book by de Klerk et al. [22].

Chapter 3

Algorithms and index selection rules

In this chapter we present the algorithms considered in this thesis, followed by the index selection rules.

3.1 Pivot algorithms for linear programming

Most theoretical forms of the pivot algorithms work on the standard form of the problem - all constraints are equations and all variables have a uniform lower bound of zero and no upper bounds. Any LP problem can easily be converted to this form, although the practical effect of such a conversion can be significant. This standard form and its dual is

$$\begin{aligned} \min \mathbf{c}^T\mathbf{x} \\ A\mathbf{x} = \mathbf{b} \\ \mathbf{x} \geq 0 \end{aligned} \qquad\qquad \begin{aligned} \max \mathbf{y}^T\mathbf{b} \\ \mathbf{y}^T A \leq \mathbf{c} \end{aligned}$$

3.1.1 The primal simplex method

The following pseudo code summarizes the primal simplex method [21]. After the pseudo code, we describe the key steps of the primal simplex method, provide a connection between the pseudo code and the description of the algorithm.

The primal simplex method

input data:
$A \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^n$, $\mathcal{I} = \{1, \ldots, n\}$;
The problem is given in the canonical form: $\min \mathbf{c}^T\mathbf{x}, A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0$;
B feasible basis, $\mathcal{I}_B$ its index set, $\overline{\mathbf{b}} := B^{-1}\mathbf{b}$, the current basis solution;
begin
calculate $\overline{\mathbf{c}} := \mathbf{c} - (\mathbf{c}_B^T B^{-1})A$, the reduced costs;
$\mathcal{I}_- := \{i \in \mathcal{I}_N \,|\, \overline{c}_i < 0\}$, indices of the non optimal columns; (1)
while $(\mathcal{I}_- \neq \emptyset)$ **do**
choose $q \in \mathcal{I}_-$ according to the used index selection rule; (3)
calculate $\mathbf{t}_q := B^{-1}\mathbf{a}_q, the transformed incoming columns$;
if $(\mathbf{t}_q \leq 0)$ **then**
STOP: problem dual infeasible; (5)
else
let $\vartheta := min\left\{\frac{\overline{b_i}}{t_{iq}} | i \in \mathcal{I}_B,\ t_{iq} > 0\right\}$; (primal ratio test) (4)
choose $p \in \mathcal{I}_B$ where $\frac{\overline{b_p}}{t_{pq}} = \vartheta$ according to the used index selection rule;
endif
pivoting;
update B^{-1}, $\overline{\mathbf{b}}$, $\overline{\mathbf{c}}$ and $\mathcal{I}_-$; (6)
check need for re-inversion; (7)
endwhile
STOP: An optimal solution is found; (2)
end

The key steps of the algorithm are

1. Determine the possible incoming column index set I_- which contains those variables for which the reduced costs are negative.

2. If the I_- set is empty, there are no more improving columns and the algorithm terminates.

3. Otherwise use the *index selection rule* to select an improving column.

4. Perform a primal ratio test to determine the step size, and select the outgoing column using the *index selection rule*.

5. If there are no suitable pivot elements then the problem is dual infeasible.

6. Carry out the pivot, update the basis and the factorization.

7. Check if re-inversion is necessary, and start over again.

The following flow chart of the primal simplex algorithm contains the basic pivot tableaus, not the exact description from the pseudo code of the algorithm.

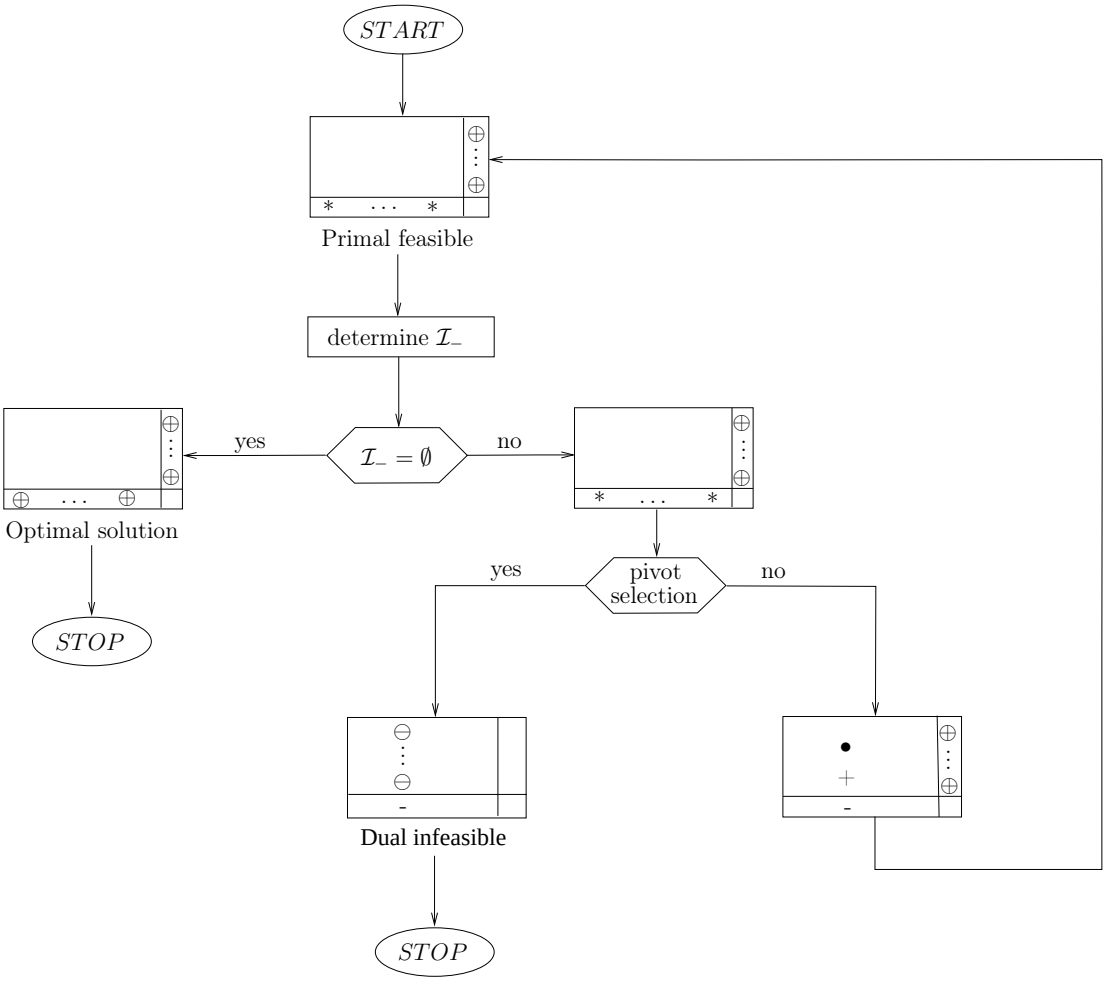


Figure 3.1: Flow chart of the primal simplex algorithm.

A significant advantage of the primal simplex method is its simplicity. In the case of a non-degenerate pivot, the value of the objective function increases monotonically, ensuring finiteness.

The following example shows how the primal simplex algorithm works on a small problem.

Example 3.1.1 *Let's solve the following linear programming problem with the primal simplex algorithm.*

$$\begin{aligned} \min(-x_1 - 2x_2 - 4x_3 - x_4) \\ x_1 + x_3 + x_4 \leq 5 \\ x_1 + x_2 + x_4 \leq 6 \\ x_3 \leq 7 \\ x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

We have to convert this problem to a canonic linear programming problem:

$$\begin{aligned} \min(-x_1 - 2x_2 - 4x_3 - x_4) \\ x_1 + x_3 + x_4 + u_1 = 5 \\ x_1 + x_2 + x_4 + u_2 = 6 \\ x_3 + u_3 = 7 \\ x_1, x_2, x_3, x_4, u_1, u_2, u_3 \geq 0 \end{aligned}$$

As this is the first example in the thesis for which we present full simplex tableaus, it is appropriate to note here that the pivot positions on the tableaus will be marked by both being written in bold, and also by being boxed. Traditionally, the row of the objective is marked with a 'z'. This 'z' is sometimes thought of as the slack variable of a constraint made of the objective. The solution with the primal simplex algorithm is the following:

1.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
u_1	***1***	*0*	*1*	*1*	*1*	*0*	*0*	*5*
u_2	*1*	*1*	*0*	*1*	*0*	*1*	*0*	*6*
u_3	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*7*
z	*-1*	*-2*	*-4*	*-1*	*0*	*0*	*0*	*0*

2.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
x_1	*1*	*0*	*1*	*1*	*1*	*0*	*0*	*5*
u_2	*0*	***1***	*-1*	*0*	*-1*	*1*	*0*	*1*
u_3	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*7*
z	*0*	*-2*	*-3*	*0*	*1*	*0*	*0*	*5*

3.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
x_1	*1*	*0*	***1***	*1*	*1*	*0*	*0*	*5*
x_2	*0*	*1*	*-1*	*0*	*-1*	*1*	*0*	*1*
u_3	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*7*
z	*0*	*0*	*-5*	*0*	*-1*	*2*	*0*	*7*

4.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
x_3	*1*	*0*	*1*	*1*	*1*	*0*	*0*	*5*
x_2	*1*	*1*	*0*	*1*	*0*	*1*	*0*	*6*
u_3	*-1*	*0*	*0*	*-1*	*-1*	*0*	*1*	*2*
z	*5*	*0*	*0*	*5*	*4*	*2*	*0*	*32*

So the optimal solution is $x_1 = 0$, $x_2 = 6$, $x_3 = 5$, and $x_4 = 0$. The optimal objective function value is 32.

3.1.2 The (primal) monotonic build-up simplex method

The following pseudo code summarizes the monotonic build-up simplex method.

The monotonic build-up (MBU) simplex algorithm

input data:
$A \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^n, \mathcal{I} = \{1, \ldots, n\}$;
The problem is given in the canonical form: $\min \mathbf{c}^T\mathbf{x}, A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0$;
B feasible basis, $\mathcal{I}_B$ its index set, $\overline{\mathbf{b}} := B^{-1}\mathbf{b}$, the current basic solution;
begin
calculate $\overline{\mathbf{c}} := \mathbf{c} - (\mathbf{c}_B^T B^{-1})A$, the reduced costs;
$\mathcal{I}_- := \{i \in \mathcal{I}_N \mid \overline{c}_i < 0\}$, indices of non optimal columns; (1)
while $(\mathcal{I}_- \neq \emptyset)$ **do**
choose $s \in \mathcal{I}_-$, the driving variable according to the used index selection rule; (3)
while $(s \in \mathcal{I}_-)$ **do**
calculate $t_s := B^{-1}\mathbf{a}_s$ and let $\mathcal{K}_s = \{i \in \mathcal{I}_B \mid t_{is} > 0\}$;
if $(\mathcal{K}_s = \emptyset)$ **then** STOP: problem dual infeasible; (5)
else
let $\vartheta := min\left\{\frac{\overline{\mathbf{b}_i}}{t_{is}} \mid i \in \mathcal{K}_s\right\}$; (the primal ratio test) (4)
choose $r \in \mathcal{K}_s$ according to the used index selection rule, where $\frac{\overline{\mathbf{b}_r}}{t_{rs}} = \vartheta$;
let $\theta_1 := \frac{|\overline{\mathbf{c}_s}|}{t_{rs}}$ and $\mathcal{J} = \{i \in \mathcal{I} \mid \overline{\mathbf{c}_i} \geq 0 \text{ and } t_{ri} < 0\}$, where $\mathbf{t}^{(r)} := \mathbf{e}_i B^{-1}A$;
if $(\mathcal{J} = \emptyset)$ **then** $\theta_2 := \infty$;
else
$\theta_2 := min\left\{\frac{\overline{\mathbf{c}_i}}{|t_{ri}|} | i \in \mathcal{J}\right\}$; (the dual ratio test) (6)
choose $q \in \mathcal{J} : \theta_2 = \frac{\overline{\mathbf{c}_q}}{|t_{rq}|}$ according to the used index selection rule;
endif
if$(\theta_1 \leq \theta_2)$ **then** pivoting on the t_{rs} element; (7)
else pivoting on the t_{rq} element;
endif
update B^{-1}, $\overline{\mathbf{b}}$, $\mathbf{c}$ and $\mathcal{I}_-$ index set; (8)
check need for re-inversion; (9)
endif
endwhile
endwhile
STOP: An optimal solution is found; (2)
end

The key steps of the monotonic build-up simplex method are

1. If there is no active driving column, then determine the possible incoming column index set I_- which contains the variables with negative reduced cost.

2. If the I_- set is empty, there are no more improving columns and the algorithm terminates.

3. Otherwise use the *index selection rule* to select s as the index of an improving column. This column will serve as the driving column, and it's value is monotonically increased until it becomes dual feasible.

4. Perform a primal ratio test on the positive elements of the transformed pivot column $\mathbf{t}_s$ of the driving variable and select the outgoing variable r using the *index selection rule*.

5. If there are no suitable pivot elements then the problem is dual infeasible.

6. Using the row $\mathbf{t}^r$ selected by the primal ratio test, carry out a dual ratio test over $\mathcal{J}$ the dual feasible columns with negative pivot elements in row r, breaking ties using the *index selection rule*.

7. Compare the values of the two ratio test θ_1 and θ_2. If it's ratio is not larger, then choose the pivot t_{rs} selected by the primal ratio test, otherwise choose the pivot t_{rq} selected by the dual one.

8. Carry out the pivot, update the basis and the factorization.

9. Check if re-inversion is necessary, and iterate.

For the sake of completeness, let's see the flow chart of the monotonic build-up simplex algorithm. In the flow chart, the decision highlighted as green is an auxiliary pivot that keeps the same driving dual variable, while red is the pivot that makes the selected dual driving variable feasible. The following figure contains the basic pivot tableaus, not the exact description, which can be found in the pseudo code of the algorithm.

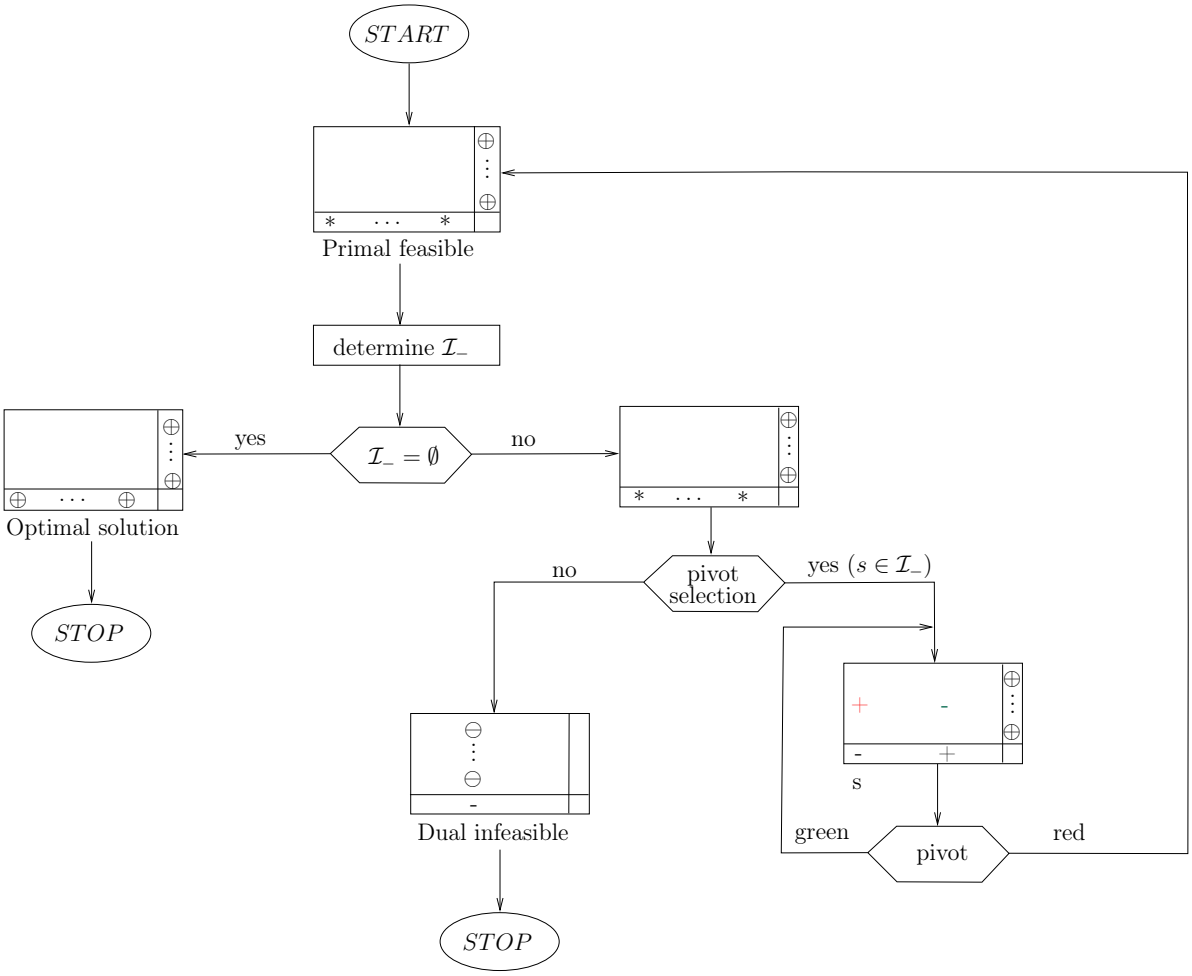


Figure 3.2: Flow chart of the primal MBU-simplex algorithm.

It is not straightforward why the MBU algorithm is correct as it is presented, so it is important to restate the following result:

Theorem 3.1.1 *[6] Consider any pivot sequence of the MBU algorithm. Following a pivot selected by the dual side ratio test, the next basis produced by the algorithm has the following properties:*

1. $\bar{c}_s < 0$ *i.e. the driving variable is still dual infeasible,*
2. *if* $\bar{b}_i < 0$ *then* $t_{is} < 0$, *and*
3. $\max\left\{\frac{\bar{b}_i}{t_{is}} \mid \bar{b}_i < 0\right\} \leq \min\left\{\frac{\bar{b}_i}{t_{is}} \mid t_{is} > 0\right\}$.

It is important to note that the proof of this result also shows that property *(3)* of the theorem holds for every basis produced by a pivot of the algorithm where the dual ratio

test was smaller. Property *(2)* shows that whenever the primal ratio test is ill defined, the problem is indeed unbounded, even if the current basis is primal infeasible.

Theorem 3.1.2 *[6] When the MBU algorithm performs a pivot selected by the primal side ratio test on t_{rs}, then the next basis is primal feasible.*

Note that if the selected pivot is primal non-degenerate then the objective function strictly increases, providing progress and finiteness.

The following example shows how the primal simplex algorithm works in a small problem.

Example 3.1.2 *Let's solve the following example with the monotonic build-up algorithm.*

$$\begin{aligned} \min(-x_1 - 2x_2 - 4x_3 - x_4) \\ x_1 + x_3 + x_4 \leq 5 \\ x_1 + x_2 + x_4 \leq 6 \\ x_3 \leq 7 \\ x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

As we see related to the simplex method, the problem should be converted to the canonical form:

$$\begin{aligned} \min(-x_1 - 2x_2 - 4x_3 - x_4) \\ x_1 + x_3 + x_4 + u_1 = 5 \\ x_1 + x_2 + x_4 + u_2 = 6 \\ x_3 + u_3 = 7 \\ x_1, x_2, x_3, x_4, u_1, u_2, u_3 \geq 0 \end{aligned}$$

The solution with the monotonic build-up simplex algorithm is the following. The driving dual variable (reduced cost) is marked in bold.

1.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
u_1	***1***	*0*	*1*	*1*	*1*	*0*	*0*	*5*
u_2	*1*	*1*	*0*	*1*	*0*	*1*	*0*	*6*
u_3	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*7*
z	***-1***	*-2*	*-4*	*-1*	*0*	*0*	*0*	*0*

2.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
x_1	*1*	*0*	*1*	*1*	*1*	*0*	*0*	*5*
u_2	*0*	*1*	*-1*	*0*	***-1***	*1*	*0*	*1*
u_3	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*7*
z	*0*	***-2***	*-3*	*0*	*1*	*0*	*0*	*5*

3.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
x_1	*1*	***1***	*0*	*1*	*0*	*1*	*0*	*6*
u_1	*0*	*-1*	*1*	*0*	*1*	*-1*	*0*	*-1*
u_3	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*7*
z	*0*	***-1***	*-4*	*0*	*0*	*1*	*0*	*6*

4.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
x_2	*1*	*1*	*0*	*1*	*0*	*1*	*0*	*6*
u_1	*1*	*0*	***1***	*1*	*1*	*0*	*0*	*5*
u_3	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*7*
z	*1*	*0*	***-4***	*1*	*0*	*2*	*0*	*12*

5.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
x_2	*1*	*1*	*0*	*1*	*0*	*1*	*0*	*6*
u_1	*1*	*0*	*1*	*1*	*1*	*0*	*0*	*5*
u_3	*-1*	*0*	*0*	*-1*	*-1*	*0*	*1*	*2*
z	*5*	*0*	*0*	*5*	*4*	*2*	*0*	*32*

So the optimal solution is $x_1 = 0$, $x_2 = 6$, $x_3 = 5$, and $x_4 = 0$. The optimal value of the original problem is 32, just like we have seen with the primal simplex algorithm.

3.1.3 First phase for the primal simplex and the monotonic build-up simplex method

For both methods, a standard first phase using artificial slacks is used. Assume a problem in the form

$$A\mathbf{x} = \mathbf{b} \tag{3.1}$$

$$\mathbf{x} \geq \mathbf{0} \tag{3.2}$$

Here, we can assume that $\mathbf{b} \leq \mathbf{0}$ as otherwise we can multiply the appropriate rows by -1. By introducing an artificial identity matrix and matching non-negative variables, we get a system with an obvious initial feasible basis. By minimizing the sum of these artificial variables, we can use the same primal simplex method to find a feasible basis to the original problem. If the minimum is nonzero, than the original problem has no feasible solution.

$$\min \mathbf{1}^T \mathbf{s}^* \tag{3.3}$$

$$I\mathbf{s}^* + A\mathbf{x} = \mathbf{b} \tag{3.4}$$

$$\mathbf{s}^*, \mathbf{x} \geq \mathbf{0} \tag{3.5}$$

$$\tag{3.6}$$

If the optimum is found and zero, but there are still artificial slack variables in the basis, we can always perform degenerate pivots to remove any such artificial variable.

For practical purposes, if we transform the first phase objective $\mathbf{1}^T\mathbf{s}$ to the basis of the artificial slacks, as we only need to carry out simple elimination steps, it is easy to see that the first phase objective becomes $\min -\mathbf{1}^T A$.

To demonstrate the usage of the first phase, we will solve the same example with the primal simplex and the monotonic build-up simplex method as well. Let's see the first phase of the primal simplex problem. Following the tradition of denoting the objective with variable 'z', the first phase objective will be denoted by z^*.

Example 3.1.3 *Let's solve the following problem:*

$$\begin{aligned} \min(-x_1 - 2x_2 - 3x_3 - 2x_4 - 3x_5) & \\ x_1 + 2x_2 + x_4 + x_5 &\leq 110 \\ x_1 + 2x_3 + x_4 + x_5 &= 80 \\ 2x_2 + x_5 &\geq 40 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$

Convert to the canonical from the problem above and introduce the secondary objective function (z^*) as well:

$$\begin{aligned} z = \min(-x_1 - 2x_2 - 3x_3 - 2x_4 - 3x_5) & \\ z^* = \min(-x_1 - 2x_2 - 2x_3 - x_4 - 2x_5 + u_3) & \\ x_1 + 2x_2 + x_4 + x_5 + u_1 &= 110 \\ x_1 + 2x_3 + x_4 + x_5 + v_2^* &= 80 \\ 2x_2 + x_5 - u_3 + v_3^* &= 40 \\ x_1, x_2, x_3, x_4, x_5, u_1, u_3, v_1^*, v_2^* &\geq 0 \end{aligned}$$

The solution with the primal simplex algorithm is the following:

1.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	v_1^*	v_2^*	**b**
u_1	*1*	*2*	*0*	*1*	*1*	*1*	*0*	*0*	*0*	*110*
v_1^*	***1***	*0*	*2*	*1*	*1*	*0*	*0*	*1*	*0*	*80*
v_2^*	*0*	*2*	*0*	*0*	*1*	*0*	*-1*	*0*	*1*	*40*
z	*-1*	*-2*	*-3*	*-2*	*-3*	*0*	*0*	*0*	*0*	*0*
z^*	*-1*	*-2*	*-2*	*-1*	*-2*	*0*	*1*	*0*	*0*	*0*

2.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	v_1^*	v_2^*	**b**
u_1	0	2	-2	0	0	1	0	-1	0	30
x_1	1	0	2	1	1	0	0	1	0	80
v_2^*	0	2	0	0	**1**	0	-1	0	1	40
z	0	-2	-1	-1	-2	0	0	1	0	80
z^*	0	-2	0	0	-1	0	1	1	0	80

3.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	v_1^*	v_2^*	**b**
u_1	0	2	-2	0	0	1	0	-1	0	30
x_1	1	-2	2	1	0	0	1	1	1	40
x_5	0	2	0	0	1	0	-1	0	1	40
z	0	2	-1	-1	0	0	-2	1	2	160
z^*	0	0	0	0	0	0	0	1	1	120

Now, we can drop the column of the v_1^* and the v_2^* vectors and the secondary objective function's row. We get the following table:

4.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	**b**
u_1	0	2	-2	0	0	1	0	30
x_1	1	-2	**2**	1	0	0	1	40
x_5	0	2	0	0	1	0	-1	40
z	0	2	-1	-1	0	0	-2	160

Now a feasible solution is the $x_1 = 40$, $x_2 = 0$, $x_3 = 0$, $x_4 = 0$ and $x_5 = 40$, but we would like to find the optimal value of the objective function thus we have to continue the pivoting on 3^{rd} element in the 2^{nd} row.

5.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	**b**
u_1	1	0	0	1	0	1	1	70
x_3	$\frac{1}{2}$	-1	1	$\frac{1}{2}$	0	0	$\frac{1}{2}$	20
x_5	0	2	0	0	1	0	-1	40
z	$\frac{1}{2}$	1	0	$-\frac{1}{2}$	0	0	$-\frac{3}{2}$	180

6.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	**b**
u_1	0	**2**	-2	0	0	1	0	30
x_4	1	-2	2	1	0	0	**1**	40
x_5	0	2	0	0	1	0	-1	40
z	1	0	1	0	0	0	-1	200

7.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	**b**
u_1	0	**2**	-2	0	0	1	0	30
u_3	1	-2	2	1	0	0	1	40
x_5	1	0	2	1	1	0	0	80
z	2	-2	3	1	0	0	0	240

8.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	**b**
x_2	0	1	-1	0	0	$\frac{1}{2}$	0	15
u_3	1	0	0	1	0	1	1	70
x_5	1	0	2	1	1	0	0	80
z	2	0	1	1	0	1	0	270

We get the optimal solution which is the following: $x_1 = 0$, $x_2 = 15$, $x_3 = x_4 = 0$ *and* $x_5 = 80$. *The optimal value of the objective function is 270.*

Let's solve the same problem with the monotonic build-up simplex algorithm.

Example 3.1.4 *The original problem is the following:*

$$\begin{aligned} \min(-x_1 - 2x_2 - 3x_3 - 2x_4 - 3x_5) \\ x_1 + 2x_2 + x_4 + x_5 \leq 110 \\ x_1 + 2x_3 + x_4 + x_5 = 80 \\ 2x_2 + x_5 \geq 40 \\ x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

Just like for the simplex algorithm, we have to convert the problem to the canonic form and introduce the secondary objective function (z^*) *as well:*

$$\begin{aligned} z = \min(-x_1 - 2x_2 - 3x_3 - 2x_4 - 3x_5) \\ z^* = \min(x_1 - 2x_2 - 2x_3 - x_4 - 2x_5 + u_3) \\ x_1 + 2x_2 + x_4 + x_5 + u_1 = 110 \\ x_1 + 2x_3 + x_4 + x_5 + v_2^* = 80 \\ 2x_2 + x_5 - u_3 + v_3^* = 40 \\ x_1, x_2, x_3, x_4, x_5, u_1, u_3, v_1^*, v_2^* \geq 0 \end{aligned}$$

The solution with the monotonic build-up simplex algorithm is the following:

1.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	v_1^*	v_2^*	**b**
u_1	*1*	*2*	*0*	*1*	*1*	*1*	*0*	*0*	*0*	*110*
v_1^*	***1***	*0*	*2*	*1*	*1*	*0*	*0*	*1*	*0*	*80*
v_2^*	*0*	*2*	*0*	*0*	*1*	*0*	*-1*	*0*	*1*	*40*
z	*-1*	*-2*	*-3*	*-2*	*-3*	*0*	*0*	*0*	*0*	*0*
z^*	***-1***	*-2*	*-2*	*-1*	*-2*	*0*	*1*	*0*	*0*	*0*

2.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	v_1^*	v_2^*	**b**
u_1	0	**2**	-2	0	0	1	0	-1	0	30
x_1	1	0	2	1	1	0	0	1	0	80
v_2^*	0	2	0	0	1	0	-1	0	1	40
z	0	-2	-1	-1	-2	0	0	1	0	80
z^*	0	**-2**	0	0	-1	0	1	1	0	80

3.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	v_1^*	v_2^*	**b**
x_2	0	1	-1	0	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	15
x_1	1	0	2	1	1	0	0	1	0	80
v_2^*	0	0	**2**	0	1	-1	-1	1	1	10
z	0	0	-3	-1	-2	1	0	0	0	110
z^*	0	0	**-2**	0	-1	1	1	0	0	110

4.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	v_1^*	v_2^*	**b**
x_2	0	1	0	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	$\frac{1}{2}$	20
x_1	1	0	0	1	0	1	1	0	-1	70
x_3	0	0	1	0	$\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	5
z	0	0	0	-1	$-\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{3}{2}$	$\frac{3}{2}$	$\frac{3}{2}$	125
z^*	0	0	0	0	0	0	0	1	1	120

Now, we can drop the column of the v_1^* and the v_2^* vectors and the secondary objective function's row. We get the following table:

5.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	**b**
x_2	0	1	0	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	20
x_1	1	0	0	**1**	0	1	1	70
x_3	0	0	1	0	$\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{1}{2}$	5
z	0	0	0	**-1**	$-\frac{1}{2}$	$-\frac{1}{2}$	$-\frac{3}{2}$	125

Now a feasible solution is the $x_1 = 70$, $x_2 = 20$, $x_3 = 5$, $x_4 = 0$ and $x_5 = 0$, but we would like to find the minimum value of the objective function thus we have to continue the pivoting on 4^{th} element in the 2^{nd} row.

6.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	**b**
x_2	0	1	0	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	20
x_4	1	0	0	1	0	1	1	70
x_3	0	0	1	0	**$\frac{1}{2}$**	$-\frac{1}{2}$	$-\frac{1}{2}$	5
z	1	0	0	0	**$-\frac{1}{2}$**	$\frac{1}{2}$	$-\frac{1}{2}$	195

7.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	**b**
x_2	0	1	-1	0	0	$\frac{1}{2}$	0	15
x_4	1	0	0	1	0	1	**1**	70
x_5	0	0	2	0	1	-1	-1	10
z	1	0	1	0	0	0	**-1**	200

8.	x_1	x_2	x_3	x_4	x_5	u_1	u_3	**b**
x_2	*0*	*1*	*-1*	*0*	*0*	$\frac{1}{2}$	*0*	*15*
u_3	*1*	*0*	*0*	*1*	*0*	*1*	*1*	*70*
x_5	*1*	*0*	*2*	*1*	*1*	*0*	*0*	*80*
z	*2*	*0*	*1*	*1*	*0*	*1*	*0*	*270*

Now we find that the optimal objective is 270 for the optimal solution, which is $x_2 = 5$ and $x_5 = 80$.

As the example demonstrates, when the ratio test condition $\Theta_1 \leq \Theta_2$ always hold, the MBU simplex performs the same pivots as the traditional primal simplex would.

3.1.4 The criss–cross method for linear programming

For completeness, we state the criss–cross method for linear programming problems.

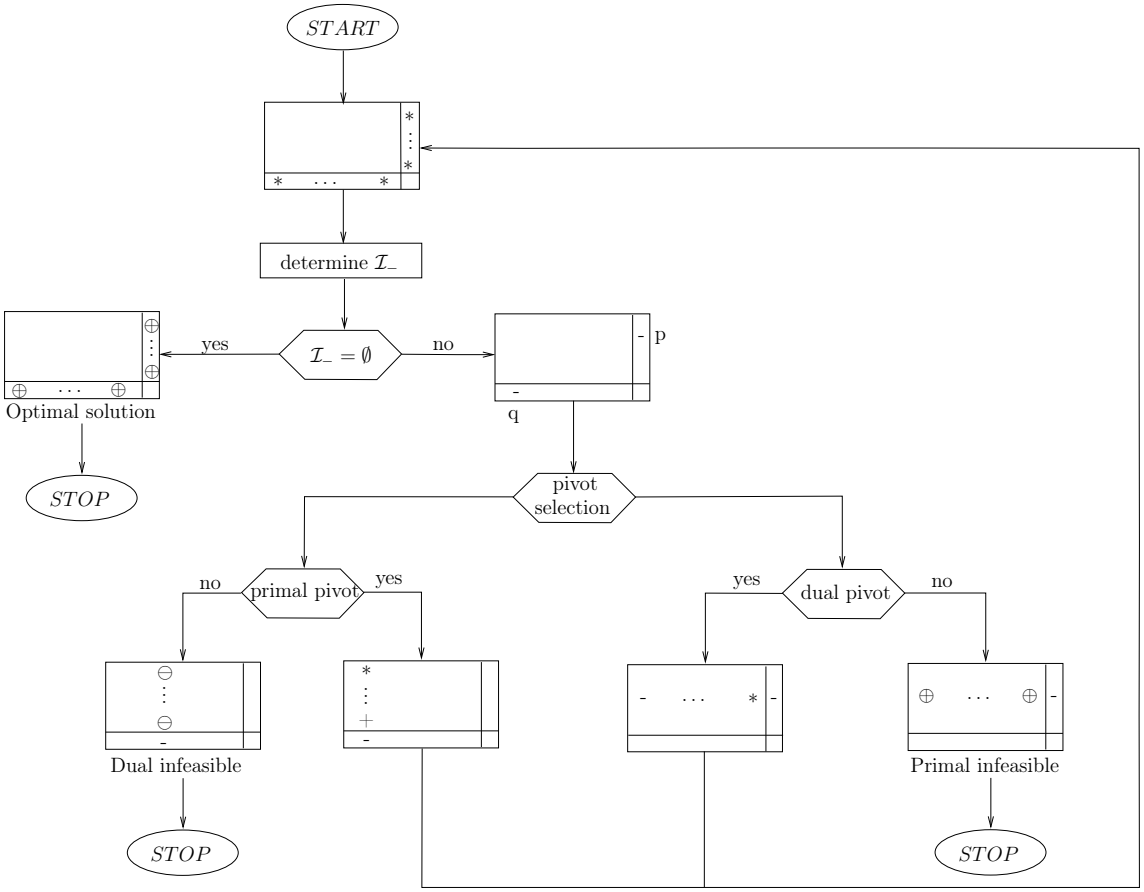


Figure 3.3: Flow chart of the criss–cross algorithm.

The criss–cross method

input data:
$A \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^n$, $\mathcal{I} = \{1, \ldots, n\}$;
The problem is given in the canonical form: $\min \mathbf{c}^T\mathbf{x}, A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq 0$;
B basis, $\mathcal{I}_B$ its index set, $\overline{\mathbf{b}} := B^{-1}\mathbf{b}$, the current basic solution;
begin
calculate $\overline{\mathbf{c}} := \mathbf{c} - (\mathbf{c}_B^T B^{-1})A$, the reduced costs;
calculate $\overline{\mathbf{b}} = B^{-1}\mathbf{b}$, the transformed right hand side;
$\mathcal{I}_- := \{i \in \mathcal{I}_N \mid \overline{c}_i < 0\} \cup \{j \in \mathcal{I}_B \mid \overline{b_j} < 0\}$; (1)
while $(\mathcal{I}_- \neq \emptyset)$ **do**
let $p := min\{i \in I_B : \overline{\mathbf{b}_i} < 0\}$;
let $q := min\{j \in I_N : \overline{\mathbf{c}_j} < 0\}$;
let $r := min\{p, q\}$;
if(r = p) **then**
if $(\mathbf{t}^{(p)} \geq 0)$ **then**
STOP: problem primal infeasible;
else
let $q := min\{j \in I_N : t_{pj} < 0\}$; (2)
endif
else
if$(\mathbf{t}_r \leq 0)$ **then**
STOP: problem dual infeasible;
else
let $p := min\{i \in I_B : t_{iq} > 0\}$; (3)
endif
endif
pivoting on the t_{pq} element;
update $\mathcal{I}_-$ and $I_B := I_B \cup \{q\} \setminus \{p\}$; (4)
endwhile
STOP: an optimal solution is found;
end

The main differences compared to the primal simplex algorithm:

1. Simultaneous primal and dual side pricing.
2. Dual type pivot on an infeasible row.
3. Primal type pivot on a non-optimal column.
4. Neither primal, nor dual side feasibility is maintained.

The following example shows how the criss–cross algorithm works on a small problem.

Example 3.1.5 *Let's solve the following problem with the criss–cross algorithm.*

$$\begin{aligned} \min(-x_1 - 2x_2 - 4x_3 - x_4) \\ x_1 + x_3 + x_4 \leq 5 \\ x_1 + x_2 + x_4 \leq 6 \\ x_3 \leq 7 \\ x_1, x_2, x_3, x_4 \geq 0 \end{aligned}$$

Convert the example to the canonical form:

$$\begin{aligned} \min(-x_1 - 2x_2 - 4x_3 - x_4) \\ x_1 + x_3 + x_4 + u_1 = 5 \\ x_1 + x_2 + x_4 + u_2 = 6 \\ x_3 + u_3 = 7 \\ x_1, x_2, x_3, x_4, u_1, u_2, u_3 \geq 0 \end{aligned}$$

The solution with the criss–cross algorithm is the following:

1.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
u_1	***1***	*0*	*1*	*1*	*1*	*0*	*0*	*5*
u_2	*1*	*1*	*0*	*1*	*0*	*1*	*0*	*6*
u_3	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*7*
z	*-1*	*-2*	*-4*	*-1*	*0*	*0*	*0*	*0*

2.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
x_1	*1*	*0*	*1*	*1*	*1*	*0*	*0*	*5*
u_2	*0*	***1***	*-1*	*0*	*-1*	*1*	*0*	*1*
u_3	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*7*
z	*0*	*-2*	*-3*	*0*	*1*	*0*	*0*	*5*

3.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
x_1	*1*	*0*	***1***	*1*	*1*	*0*	*0*	*5*
x_2	*0*	*1*	*-1*	*0*	*-1*	*1*	*0*	*1*
u_3	*0*	*0*	*1*	*0*	*0*	*0*	*1*	*7*
z	*0*	*0*	*-5*	*0*	*-1*	*2*	*0*	*7*

4.	x_1	x_2	x_3	x_4	u_1	u_2	u_3	**b**
x_3	*1*	*0*	*1*	*1*	*1*	*0*	*0*	*5*
x_2	*1*	*1*	*0*	*1*	*0*	*1*	*0*	*6*
u_3	*-1*	*0*	*0*	*-1*	*-1*	*0*	*-1*	*2*
z	*5*	*0*	*0*	*5*	*4*	*2*	*0*	*32*

The solution of the problem is $x_1 = 0$, $x_2 = 6$, $x_3 = 5$ and $x_4 = 0$. The optimal value of the objective function is 32.

Let's see a little more complex example.

Example 3.1.6 *With this problem, the primal simplex and the MBU-simplex algorithms are using a first phase algorithm. With the criss–cross we don't have to use any first phase method.*

Let's solve the following problem:

$$\begin{aligned} \min(-x_1 - 2x_2 - 3x_3 - 2x_4 - 3x_5) \\ x_1 + 2x_2 + x_4 + x_5 \leq 110 \\ x_1 + 2x_3 + x_4 + x_5 = 80 \\ 2x_2 + x_5 \geq 40 \\ x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

Convert to the canonical from the problem above:

$$\begin{aligned} \min(-x_1 - 2x_2 - 4x_3 - x_4) \\ x_1 + x_3 + x_4 + u_1 = 5 \\ x_1 + x_2 + x_4 + u_2 = 6 \\ x_3 + u_3 = 7 \\ x_1, x_2, x_3, x_4, u_1, u_2, u_3 \geq 0 \end{aligned}$$

The solution with the criss–cross algorithm is the following:

1.	x_1	x_2	x_3	x_4	x_5	u_1	u_2	v_1^*	v_2^*	**b**
u_1	1	2	0	1	1	1	0	0	0	110
v_1^*	1	0	2	1	1	0	0	1	0	80
v_2^*	0	2	0	0	1	0	**-1**	0	1	40
	-1	-2	-3	-2	-3	0	0	0	0	0

In those problems, where we have to introduce artificial slack variables because of the greater than equal and the equality constraint when using the primal simplex or MBU simplex methods; when solving the problem with the criss–cross algorithm, using the Gauss-Jordan elimination algorithm, we have to exchange the slack variable with another item from the basis. So now we have to pivot on the 7^{th} element of the 3^{rd} row.

2.	x_1	x_2	x_3	x_4	x_5	u_1	u_2	v_1^*	v_2^*	**b**
u_1	1	2	0	1	1	1	0	0	0	110
v_1^*	1	0	**2**	1	1	0	0	1	0	80
u_2	0	-2	0	0	-1	0	1	0	-1	-40
c	-1	-2	-3	-2	-3	0	0	0	0	0

3.	x_1	x_2	x_3	x_4	x_5	u_1	u_2	v_1^*	v_2^*	**b**
u_1	1	**2**	0	1	1	1	0	0	0	110
x_3	$\frac{1}{2}$	0	1	$\frac{1}{2}$	$\frac{1}{2}$	0	0	$\frac{1}{2}$	0	40
u_2	0	-2	0	0	-1	0	1	0	-1	-40
c	$\frac{1}{2}$	-2	0	$-\frac{1}{2}$	$-\frac{3}{2}$	0	0	$\frac{3}{2}$	0	120

After we entered into the bases two elements in place of the artificial slack variables, we get the following tableau. On this tableau we can start to solve the problem with the criss–cross algorithm.

4.	x_1	x_2	x_3	x_4	x_5	u_1	u_2	**b**
u_1	1	**2**	0	1	1	1	0	110
x_3	$\frac{1}{2}$	0	1	$\frac{1}{2}$	$\frac{1}{2}$	0	0	40
u_2	0	-2	0	0	-1	0	1	-40
c	$\frac{1}{2}$	-2	0	$-\frac{1}{2}$	$-\frac{3}{2}$	0	0	120

5.	x_1	x_2	x_3	x_4	x_5	u_1	u_2	**b**
x_2	$\frac{1}{2}$	1	0	$\frac{1}{2}$	**$\frac{1}{2}$**	$\frac{1}{2}$	0	55
x_3	$\frac{1}{2}$	0	1	$\frac{1}{2}$	$\frac{1}{2}$	0	0	40
u_2	1	0	0	1	0	1	1	70
c	$\frac{3}{2}$	0	0	$\frac{1}{2}$	$-\frac{1}{2}$	1	0	230

6.	x_1	x_2	x_3	x_4	x_5	u_1	u_2	**b**
x_5	1	2	0	1	1	1	0	110
x_3	0	-1	1	0	0	$-\frac{1}{2}$	0	-15
u_2	1	0	0	1	0	1	1	70
c	2	1	0	1	0	$\frac{3}{2}$	0	285

7.	x_1	x_2	x_3	x_4	x_5	u_1	u_2	**b**
x_5	1	0	2	1	1	0	0	80
x_2	0	1	-1	0	0	$\frac{1}{2}$	0	15
u_2	1	0	0	1	0	1	1	70
c	2	0	1	1	0	1	0	270

The solution is $x_1 = 0$, $x_2 = 15$, $x_3 = x_4 = 0$ *and* $x_5 = 80$. *The optimal objective value is 270.*

If the problem gets solved in the form which only contains less than equal constraints, then the starting bases contains that new variables which we introduced when the problem was converted to the canonical form. Otherwise when there is any equality or greater than equal linear constraints in the problem, we have to introduce the so called slack variables. In this case we can see that the first phrase is the same algorithm if we are using the simplex and the MBU-simplex algorithms, but if we are using the criss–cross algorithm, we simply have to use the Gauss-Jordan elimination algorithm to change the slack variables to any other element to set the initial basis.

3.2 The criss–cross method for linear complementarity problems

The criss–cross algorithms for LCP problems need a starting complementary solution, for which $\mathbf{u} = \mathbf{0}$ and $\mathbf{v} = \mathbf{q}$ is a possible selection. Starting from the initial complementarity solution, the criss–cross method performs a sequence of so called diagonal and exchange pivots.

If in any complementary basis during the algorithm, the value of the basic variable v_j is infeasible and $t_{jj} < 0$, then the algorithm performs a *diagonal pivot* during which variable u_j enters the basis and its complementary variable pair v_j leaves it.

If $t_{jj} = 0$ and the algorithm select v_j to leave the basis, then it pivots on such an index k for which $t_{jk} < 0$. The next basis tableau is no longer complementary. To restore the complementarity of the basis tableau and the solution, the algorithm carries out a pivot on the transpose position at index pair (k, j) as well. According to Lemma 6.1.2, in the case a special problem class of sufficient matrices $t_{kj} > 0$ will hold in this situation. These two pivots together are called an *exchange pivot*.

During an exchange pivot, variables u_j and u_k enter the basis and their complementary pairs v_j and v_k leave it. To emphasize which index has been selected first, we say that u_j and v_j have been selected *actively*, and to emphasize that index k has been selected afterwards, we say that u_k and v_k have been selected *passively*: the terms passively and actively describe the order in which the indices are selected, irrespective of the variable types (i.e. the same rule applies to an exchange pivot involving different combinations like (v_j, u_k) exchanged with (u_j, v_k)). The pivot tableau structures of a diagonal and an exchange pivot is presented in Figure 3.4.

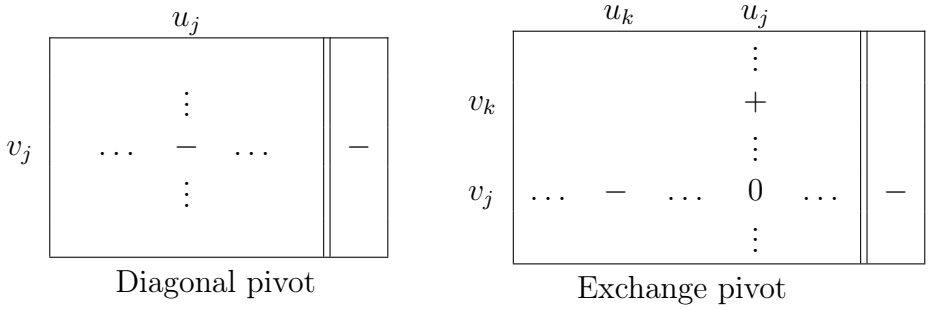


Figure 3.4: Diagonal and exchange pivots.

The linear complementarity criss–cross method

---

input data:
$T = -M$, $\bar{\mathbf{q}} = \mathbf{q}$, $r = 1$, initialize Q and $\mathbf{s}$;
begin
  determine $\mathcal{J} := \{i \in \mathcal{I} \mid \bar{\mathbf{q}}_i < 0\}$;
  while $(\mathcal{J} \neq \emptyset)$ **do**
    Select $k \in \mathcal{J}$ according to the index selection rule;
    if $(t_{kk} < 0)$ **then**
      diagonal pivot on t_{kk}, update $\mathbf{s}$; (1)
    else /* $t_{kk} = 0$ */
      $K := \{\alpha \in I \mid \bar{t}_{k\alpha} < 0\}$;
      if $(K = \emptyset)$ **then**
        STOP: The LCP problem has no feasible solution;
      else
        let $l \in \mathcal{K}$ be selected according to the index selection rule;
        exchange pivot on t_{kl} and t_{lk}; (2)
      endif
    endif
  endwhile
  STOP: we have a complementary feasible solution;
end

The criss–cross method for linear complementarity problems is deceptively simple:

1. If a diagonal pivot is made, the basis remains complementary.

2. If it is not possible to do a diagonal pivot, then to maintain complementarity two pivots are made called an exchange pivot.

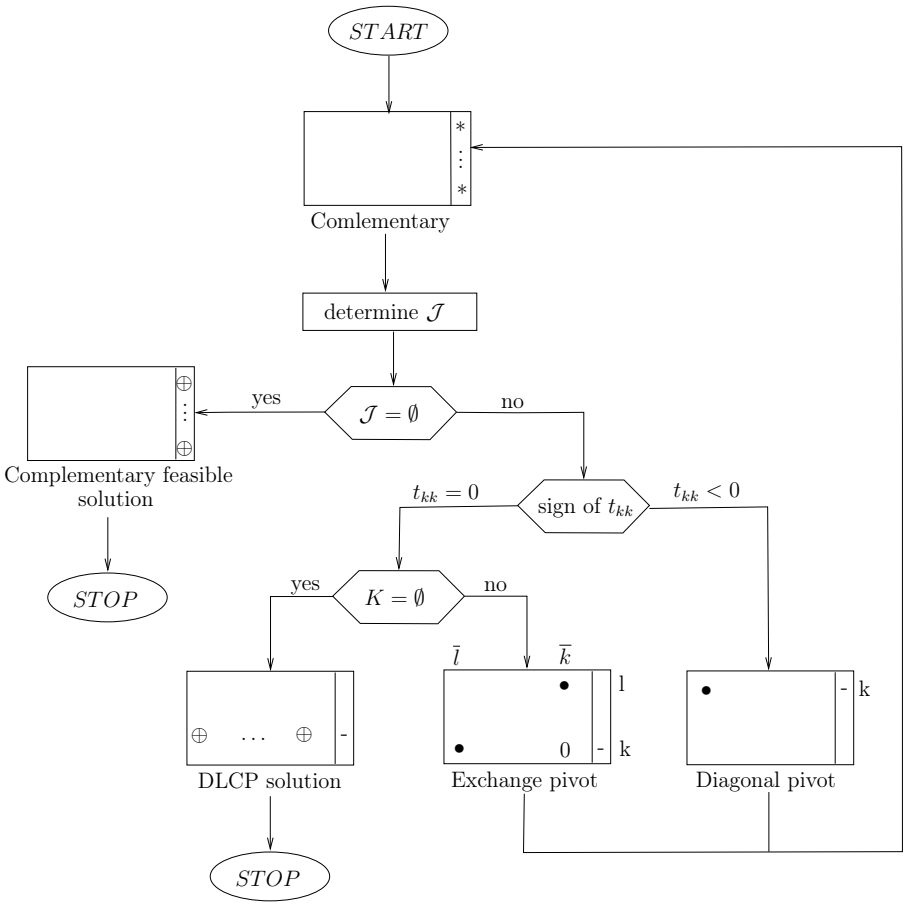


Figure 3.5: Flow chart of the LCP criss–cross algorithm.

Example 3.2.1 *The basic version of the criss–cross method relies on the properties of the matrix M of the linear complementarity problem, and so it is better suited to present an example once the relevant matrix classes have been discussed; an example is presented in Chapter 6 on page 118, Examples 6.2.1.*

The matrix classes under which the criss–cross algorithm for linear complementary problems are capable to operate as presented in in this section will be discussed in Chapter 6.

3.3 The quadratic primal simplex method

We analyze the quadratic simplex method of Wolfe [72], which is well summarized and explained in the paper of Panne et al. [67] as well.

The quadratic primal simplex algorithm starts from a complementary, primal feasible matrix. Such a basis can easily be created from a feasible basis of the original primal feasibility problem extended with the slack variables of the added new dual constraints, i.e. with the addition of variables $\mathbf{z}$ to the basis (see $(B-LCP)$). A primal feasible basis of the original problem can also be created using variants of the MBU-simplex or the criss–cross method [8, 17, 41].

Definition 3.3.1 *Assume a problem $(B-LCP)$ is given (as defined on page 22). A basis of the linear complementarity problem is called* almost complementary*, if it is complementary with the exception of two index pairs, i.e. there exists $\{i,j\} \in \{1 \dots 2n\}$ such that $u_k v_k = 0$ holds for all $k \in \{1 \dots 2n\} - \{i,j\}$*

In between generating two complementary bases, the quadratic primal simplex method can generate an arbitrary number of almost complementary basis.

Definition 3.3.2 *Let a problem $(B-LCP)$ be given. A sequence of pivots carried out by the quadratic primal simplex algorithm between two consecutive complementary basis is called a* loop*.*

A cycle of the quadratic primal simplex algorithm starts with an arbitrary primal feasible, complementary basis. In the case of such a basis, if all dual variables are also feasible, then according to the Karush-Kuhn-Tucker conditions the algorithm has found an optimal solution to the original problem. If an infeasible dual variable exists, then the quadratic primal simplex algorithm selects an arbitrary (index selection rule) infeasible dual variable from the basis. The role of an index selection rule here is to ensure the finiteness of the algorithm.

Definition 3.3.3 *The infeasible dual variable selected by the quadratic primal simplex method on a complementary basis is called the* dual driving variable *or simply the* driving variable*.*

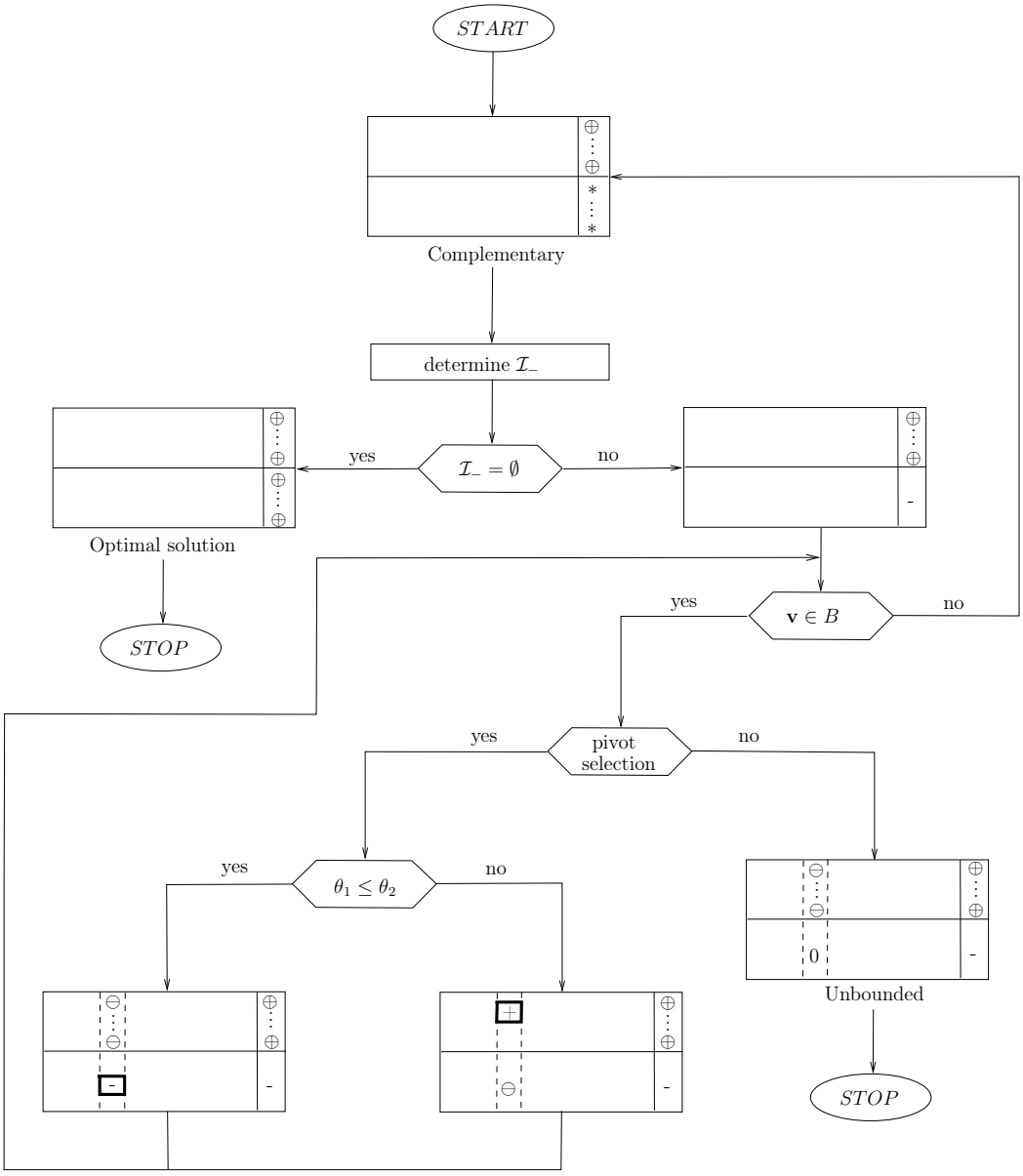


Figure 3.6: Flow chart of the quadratic simplex algorithm.

After selecting a dual driving variable, the algorithm carries out a series of pivots making up the loop, while the selected dual variable keeps remaining in the bases, or the algorithm identifies an infinite improving direction proving that the problem is unbounded.

After the selection of the driving dual variable – so starting from a complementary basis – the entering variable is the complementary pair of the selected driving dual variable. This variable (and its transformed column) defines an improving direction [68]. To maintain primal feasibility, the algorithm performs a primal side ratio test on the primal variables in

the basis.

$$\theta_2 = min\left\{\frac{\bar{q}_s}{t_{sj}} | s \in \mathcal{I}_B,\ s \text{ primal variable for which } t_{sj} > 0\right\}. \tag{3.7}$$

This θ_2 defines how much the value of the selected primal variable can be increased before the step would be limited by hitting a constraint.

The algorithm also calculates a ratio θ_1 as well, which is the value of the ratio test in the row of the driving variable, i.e. $\theta_1 := \frac{\bar{q}_v}{t_{vj}}$; or is defined as $\theta_1 = \infty$ if the coefficient in the pivot tableau is the intersection of the driving dual variable and its complementary primal is zero (if the problem is convex then the diagonal pivot elements are always non-positive). θ_1 represents the step length, when the change in the quadratic objective flips signs when the value of the entering primal variable is increased.

In the case when $\theta_1 = \theta_2 = +\infty$ the problem is unbounded [68].

If $\theta_1 \leq \theta_2$ holds, then the algorithm carries out a pivot in the row of the driving (dual) variable. In such cases, the number of primal variables in the basis is increased by one, the next basis is primal feasible and complementarity remains holding, the algorithm closes the loop in a single pivot, and the objective improves (as the row of the driving variable is always non-degenerate due to the algorithms selection rules).

If $\theta_2 < \theta_1$ then the algorithm carries out a pivot on the position identified as minimal by the primal ratio test. If the ratio test does not uniquely define that pivot row, then we apply the selected anti-cycling index selection rule to break the tie. The resulting tableau after the pivot will become almost complementary.

In the case of an almost complementarity basis, the algorithm selects the *dual* variable of the variable pair outside the basis, and selects the pivot row the same way as for the complementary case: it carries out a primal ratio test to maintain primal feasibility, and extends the ratio test with the extra ratio calculated from the row of the selected driving variable. If the ratio test allows a pivot in the row of the driving variable, the algorithm selects that for the pivot row and closes the loop [68].

The flow chart of the quadratic primal simplex method for the $(B - LCP)$ problem is presented on Figure 3.6, while it's pseudo code is the following:

The quadratic primal simplex method

input data:

A primal feasible T complementary tableau corresponding to basis B;

begin

$\mathcal{I}_- := \{k \in \mathcal{I}_B^d | \bar{q}_k < 0\}$ the indices of negative dual variables in the basis;

while $(\mathcal{I}_- \neq \emptyset)$ **do**

let $v \in \mathcal{I}_-$ be (the index of) an arbitrary driving dual variable; (1)

let $j \in \mathcal{I}_N^p$ be the complementary pair of v (primal, outside the basis); (2)

while (v is in the basis) **do**

if $(t_{vj} = 0)$ **then**

$\theta_1 := \infty$;

else

$\theta_1 := \frac{\bar{q}_v}{t_{vj}}$; (3)

endif

$\theta_2 := min\{\frac{\bar{q}_s}{t_{sj}} | s \in \mathcal{I}_B^p,\ s$ is a primal variable for which $t_{sj} > 0\}$ (4)

if $(\min(\theta_1, \theta_2) = \infty)$ **then**

STOP: problem is unbounded;

endif

if $(\theta_1 \leq \theta_2)$ **then**

pivot on tableau coefficient t_{vj};

else

let $z \in \mathcal{I}_B^p$ be such that $\frac{\bar{q}_z}{t_{zj}} = \theta_2$, using the index selection rule to break ties;

pivot on tableau coefficient t_{zj};

there is exactly one complementary pair outside the basis;

let j be the index of the dual of this pair, the new incoming variable; (5)

endif

endwhile

$\mathcal{I}_- := \{k \in \mathcal{I}_B^d | \bar{q}_k < 0\}$;

endwhile

STOP: the tableau is optimal;

end

The quadratic primal simplex incorporates both primal and dual side information into the same tableau:

1. The driving variable represents the reduced cost of the incoming column.

2. The incoming column first is the complementary pair of the selected driving dual variable.

3. The ratio test in the row of the driving variable represent when the sign of reduced cost (gradient) the quadratic objective becomes zero.

4. The primal part ratio test represents when a constraint would become violated.

5. The incoming dual in the case of a loop is also an improving direction.

Example 3.3.1 *The complexity of the algorithm justifies an example. With the use of this example, we would like to draw attention to another property as well: that is, at least as long as we use a full basis tableau to carry out the calculations, the first phase of the algorithm is technically non-trivial, as if we would like to avoid having to calculate the full complementary problem (the Karush-Kuhn-Tucker extended) system by the means of a re-inversion, it is much easier to perform the first phase already on the full pivot tableau of the linear complementarity problem.*

Consider the following simple problem:

$$\begin{aligned} \min x_1^2 + x_3^2 - 2x_1x_3 \\ x_1 - x_2 + x_3 = 1 \\ x_1 + x_2 = 2 \\ x_1, x_2, x_3 \geq 0 \end{aligned}$$

For the application of the Karush-Kuhn-Tucker problem, let us introduce the following variable pairs: The pair of a primal variable x_i is the variable z_i representing the reduced cost - which is the slack of the dual rows at the same time - and similarly the pair of the primal slack variable s_j are the dual variables denoted by y_j.

The initial (short) pivot tableau for the problem when starting from an initial bases made of the slack variables:

1.	x_1	x_2	x_3	y_1	y_2	
s_1^*	1	-1	**1**	0	0	1
s_2^*	1	1	0	0	0	2
z_1	-1	0	1	-1	-1	0
z_2	0	0	0	1	-1	0
z_3	1	0	-1	-1	0	0

The first pivot follows the objective of the first phase of the primal simplex method selecting the column of x_3. We only consider the first two rows when doing the ratio test, but we do carry out the corresponding complementary pivot as well, in order to maintain complementarity. Notice, that this secondary pivot (in the complementary position) does not affect primal feasibility, as as long as we do not make a pivot in the intersection of the original primal columns and the new dual rows, the part of the matrix corresponding to the new dual variables and original primal rows will remain the all zero matrix. The first two pivots are carried out at positions x_3 and s_1^* and next a y_1 and z_3.

2.	x_1	x_2	s_1^*	y_1	y_2	
x_3	1	-1	1	0	0	1
s_2^*	1	1	0	0	0	2
z_1	-2	1	-1	-1	-1	-1
z_2	0	0	0	1	-1	0
z_3	2	-1	1	**-1**	0	1

3.	x_1	x_2	s_1^*	z_3	y_2	
x_3	1	-1	1	0	0	1
s_2^*	1	**1**	0	0	0	2
z_1	-4	2	-2	-1	-1	-2
z_2	2	-1	1	1	-1	1
y_1	-2	1	-1	-1	0	-1

Still performing the first phase of the primal simplex method, the next pivots are x_2 exchanging with s_2^* and y_2 exchanging with z_2.

4.	x_1	s_2^*	s_1^*	y_1	y_2	
x_3	2	1	1	0	0	3
x_2	1	1	0	0	0	2
z_1	-6	-2	-2	-1	-1	-6
z_2	3	0	1	1	**-1**	3
z_3	-3	-1	-1	-1	0	-3

5.	x_1	s_2^*	s_1^*	y_1	z_2	
x_3	2	1	1	0	0	3
x_2	1	1	0	0	0	2
z_1	**-9**	-3	-3	-2	-1	**-9**
y_2	-3	-1	-1	-1	-1	-3
y_1	-3	-1	-1	-1	0	-3

The 5th tableau is now primal feasible. All three dual variables in the basis are dual infeasible, so all three primal variables outside the basis define an improving direction. However, variables s_1^* and s_2^* are artificial slack variables, so they cannot return to the basis (we could remove these from the tableau). So the improving primal variable is x_1. According to the extended primal ratio test, we can carry out a diagonal ratio test in the row of z_1.

6.	z_1	s_2^*	s_1^*	y_1	z_2	
x_3	$\frac{2}{9}$	$\frac{1}{3}$	$\frac{1}{3}$	$-\frac{4}{9}$	$-\frac{2}{9}$	*1*
x_2	$\frac{1}{9}$	$\frac{2}{3}$	$-\frac{1}{3}$	$-\frac{2}{9}$	$-\frac{1}{9}$	*1*
x_1	$-\frac{1}{9}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{2}{9}$	$\frac{1}{9}$	*1*
y_2	$-\frac{1}{3}$	*0*	*0*	$-\frac{1}{3}$	$-\frac{2}{3}$	*0*
y_1	$-\frac{1}{3}$	*0*	*0*	$-\frac{1}{3}$	$\frac{1}{3}$	*0*

The tableau is now feasible and optimal. The optimal solution is the all 1*, i.e.* $x_1 = x_2 = x_3 = 1$.

3.4 Flexible index selection rules

Most commercial solvers employ some kind of a greedy approach in selecting an incoming variable instead of an index selection rule; like steepest edge. Also, as long as the objective improves, no index selection is necessary. In the case of degeneracy, they typically perturb the problem. One of the sources of cycling could be coming from removing this perturbation, which cause the right hand side to change, needing some clean up iterations. This is a situation when cycling might occur, and when index selection rules might have a very practical role.

To ensure finiteness of the algorithm in the presence of degenerate pivots, we use flexible index selection rules. As a common framework of these rules, we use the concept of **s**-monotone index selection rules introduced in [19]. This framework defines a preference vector **s** for each index selection rule, called the primary rule. In a tie situation when the algorithm has to select an index among a set of candidates, it picks the one with the maximum value in **s**. If this value is not uniquely defined, the algorithm can select any index with maximal **s** value (this freedom is the flexibility of the rule), while still preserving finiteness. The choice of index in these situations will be based on a secondary rule, aiming the benefit from the provided flexibility.

The following finite [19] **s**-monotone primary rules are used in the thesis, which we present with the associated update of the **s** vector.

- *Minimal index/Bland:* The variables with the smallest index is selected.
  Initialize **s** consisting of constant entries

$$s_i = n - i, i \in \mathcal{I} \tag{3.8}$$

  where n is the number of the columns in the problem.

- *Last-In-First-Out (LIFO):* The most recently moved variable is selected.
  Initialize $\mathbf{s}$ equal to $\mathbf{0}$. For a pivot at (k, l) in the r^{th} iteration update $\mathbf{s}$ as

$$s_i^{'} = \begin{cases} r & \text{if } i \in \{k, l\} \\ s_i & \text{otherwise} \end{cases} \tag{3.9}$$

- *Most-Often-Selected-Variable (MOSV):* Select the variable that has been selected the largest number of times before.
  Initialize $\mathbf{s}$ equal to $\mathbf{0}$. For a pivot at (k, l) in the r^{th} iteration update $\mathbf{s}$ as

$$s_i^{'} = \begin{cases} s_i + 1 & \text{if } i \in \{k, l\} \\ s_i & \text{otherwise} \end{cases} \tag{3.10}$$

These primary pivot rules are used in combination with the following two secondary rules:

- *Bland's or the minimal index rule:* select the variable with the smallest index.
- *Dantzig's or the most negative rule:* select the variable with the smallest reduced cost.

The flexibility of the LIFO and MOSV rules will be used by the secondary rule. Naturally, benefits can be expected from the use of Dantzig's rule, and we will refer to these as

- *Hybrid-LIFO:* when the LIFO is not unique, select the one according to the most negative rule.
- *Hybrid-MOSV:* when the MOSV is not unique, select the one according to the most negative rule.

3.4.1 The s-monotone Index Selection Rules

Pivot based methods (like the simplex algorithm [21], MBU-simplex algorithm [6] or the criss–cross algorithm [74, 59, 60]) often feature the following similar principles:

1. The main flow of the algorithm is defined by a **pivot selection rule** which defines the basic characteristics of the algorithm, though the pivot position defined by it is not necessary unique (see for instance [21, 51, 12]), a series of "wrong" choices may even lead to cycling [51, 12].
2. To avoid the possibility of cycling, an **index selection rule** is used as an anti-cycling strategy (see for instance [10, 12, 62]), which may be flexible [17, 31] but usually at several bases during the algorithm, it defines the pivot position uniquely.

For several pivot algorithms, – like simplex, MBU-simplex or criss–cross algorithms – proofs of finiteness are often based on the orthogonality theorem (see Theorem 2.1.1) [40, 9], considering a minimal cycling example [9]. In minimal cycling examples, all variables should move during a cycle – if such exists – and following the movements of the least preferred variable of the index selection rule [10, 58, 31, 39], using orthogonality theorem we obtain contradiction. Examples of such pivot and index selection rules include

1. Pivot selection rules for (P-LP):

   (a) **simplex** [21] (Pivot column selection: negative reduced cost. Pivot element selection: using ratio test. Preserving non negativity of the right hand side.)

   (b) **MBU simplex** [6] (Pivot column selection: negative reduced cost, choosing driving variable. Pivot element selection: defining driving and auxiliary pivots using primal and after that dual ratio tests. Monotone in the reduced cost of the driving variable.)

   (c) **criss–cross** [60] (Pivot column/row selection is based on infeasibility – negative right hand side or negative reduced cost –. Pivot element selection: admissible pivot positions.)

2. Index selection rules:

   (a) Bland's or the minimal index rule

   (b) Last-In-First-Out (LIFO)

   (c) Most-Often-Selected-Variable (MOSV)

LIFO and MOSV index selection rules for linear programming problems were first used by S. Zhang [58] to prove the finiteness of the criss–cross algorithm with these anti-cycling index selection rules. Bilen, Csizmadia and Illés [9] proved that variants of MBU simplex algorithm are finite with both LIFO and MOSV index selection rules, while Csizmadia in his PhD Thesis [17] showed that the simplex algorithm is finite when the LIFO and MOSV are applied. These results led to the joint generalization of the above mentioned anti-cycling index selection rules.

The following general framework for proving the finiteness of several pivot algorithms and index selection rule combinations is introduced, as in [17].

Definition 3.4.1 (Possible pivot sequence [17]) *A sequence of index pairs*

$$\mathcal{S} = \{\mathcal{S}_k = (i_k, o_k) : i_k, o_k \in \mathbb{N} \text{ for some consecutive } k \in \mathbb{N}\},$$

is called a possible pivot sequence*, if*

(i) $n = \max\{\max_{k\in\mathbb{N}} i_k, \max_{k\in\mathbb{N}} o_k\}$ *is finite,*

(ii) there exists a (P-LP) with n *variables and the* $rank(A) = m$*, and*

(iii) the (possibly infinite) pivot sequence, is such that the moving variable pairs of (P-LP) correspond to the index pairs of $\mathcal{S}$*.*

The index pairs of a possible pivot sequence are thus only required to comply with the basic and nonbasic status. It is now easy to show that

Proposition 3.4.1 *If a possible pivot sequence is not finite then there exists a (sub)set of indices,* $\mathcal{I}^*$*, that occur infinitely many times in* $\mathcal{S}$*.* ■

Let us introduce the concept of *pivot index preference*.

Definition 3.4.2 (Pivot index preference [17]) *A sequence of vectors* $\mathbf{s}_k \in \mathbb{N}^n$ *is called a* pivot index preference *of an index selection rule, if in iteration* j*, in the case of ambiguity according to a pivot selection rule, the index selection rule selects an index with highest value in* $\mathbf{s}_j$ *among the candidates.*

The concept of **s**-monotone index selection rule aims to formalize a common monotonicity property of several index selection rules.

Definition 3.4.3 (s-monotone index selection rules [17]) *Let* $n \in \mathbb{N}$ *be given. An index selection rule is called* **s**-monotone*, if*

1. *there exists a pivot index preference* $\mathbf{s}_k \in \mathbb{N}^n$*, for which*
   - *(a) the values in the vector* $\mathbf{s}_{j-1}$ *after iteration* j *may only change for* i_j *and* o_j*, where* i_j *and* o_j *are the indices involved in the pivot operation,*
   - *(b) the values may not decrease.*

2. *For any infinite possible pivot sequence $\mathcal{S}$ and for any iteration j there exists iteration $r \geq j$ such that*

   (a) *the index with minimal value in $\mathbf{s}_r$ among $\mathcal{I}^* \cap \mathcal{I}_{B_r}$ is unique (let it be l), where $\mathcal{I}_{B_r}$ is the set of basic indices in iteration r, and $\mathcal{I}^*$ is the set of all indices that appear infinitely many times in $\mathcal{S}$,*

   (b) *in iteration $t > r$ when index $l \in \mathcal{I}^*$ occurs again in $\mathcal{S}$ for the first time, the indices of $\mathcal{I}^*$ that occurred in $\mathcal{S}$ strictly between S_r and S_t have a value in $\mathbf{s}_t$ higher than the index l.*

We now provide examples for **s**-monotone index selection rules.

Theorem 3.4.1 *The following index selection rules:*

1. *the minimal index rule,*
2. *the Most-Often-Selected-Variable rule and*
3. *the Last-In-First-Out index selection rule*

are **s***-monotone index selection rules.*

The proof of this theorem [17] follows from the following observation and two lemmas.

For the minimal index rule let us set each vector $\mathbf{s}_k$ to be equal to the vector $(n, n-1, \ldots, 1)^T$. Then it is easy to show that the minimal index rule is **s**-monotone.

Lemma 3.4.1 *The LIFO index selection rule is* **s***-monotone index selection rule [19, 20].*

Proof. Let us initiate the vector **s** to be the all zero vector. In a pivot when x_{i_k} leaves and x_{o_k} enters the basis in the k^{th} iteration, the values of **s** are modified to favor these variables:

$$s'_i = \begin{cases} k & \text{if } i \in \{i_k, o_k\}, \\ s_i & \text{otherwise,} \end{cases} \tag{3.11}$$

and assume that a possible pivot sequence $\mathcal{S}$ is generated using the pivot index preference.

It is clear that the series of **s** vectors defined in such a way, form a pivot index preference for the LIFO rule. Furthermore, it is obvious that the properties 1 (a) and 1 (b) of the **s**-monotone index selection rule are satisfied.

In the case of an infinite possible pivot sequence and an arbitrary iteration j either all the s_i, $i \in \mathcal{I}^* \cap \mathcal{I}_{B_j}$ values are already different or if some have the same (initial) value, meaning they have not moved yet, then there should be an iteration later when these variables move

for the first time. Let us denote that iteration by r when the last variable having index from $\mathcal{I}^*$ moves for the first time. Then for the vector $\mathbf{s}_r$ 2 (a) holds. Property 2 (b) follows from the definition of update for vector $\mathbf{s}$. ■

We prove that the most-often-selected variable rule is an $\mathbf{s}$-monotone index selection rule, too.

Lemma 3.4.2 *The MOSV index selection rule is an* $\mathbf{s}$*-monotone index selection rule [19, 20].*

Proof. Let the vector $\mathbf{s}$ be initialized as the zero vector. In a pivot when x_{i_k} leaves and x_{o_k} enters the basis in the k^{th} iteration, the values of $\mathbf{s}$ are modified to make the selection of these variables more favourable for the algorithm:

$$s'_i = \begin{cases} s_i + 1 & \text{if } i \in \{i_k, o_k\}, \\ s_i & \text{otherwise,} \end{cases} \tag{3.12}$$

and assume that a possible pivot sequence $\mathcal{S}$ is generated using the pivot index preference that defines the pivot index preference for MOSV. Due to the definition of the MOSV update, the properties 1 (a) and 1 (b) of the $\mathbf{s}$-monotone index selection rule is satisfied.

For any infinite possible pivot sequence, define $\mathcal{I}^*$ as the set of indices appearing infinitely many times in the sequence. Let us denote by $\mathcal{I}_{N_j}$ the set of nonbasic indices for the j^{th} iteration and let $\mathcal{M}_N = \mathcal{I}_{N_j} \cap \mathcal{I}^*$ and $\mathcal{M}_B = \mathcal{I}^* \setminus \mathcal{M}_N$. We define the numbers γ_i as follows:

$$\gamma_i = \begin{cases} s_i, & \text{if } i \in \mathcal{M}_N \\ s_i + 1, & \text{if } i \in \mathcal{M}_B \end{cases} \tag{3.13}$$

Let

$$\mathcal{P} = \{i \in \mathcal{I}^* \mid i \in \arg\min_{k \in \mathcal{I}^*} \gamma_k\} \qquad \text{and} \qquad \min_{k \in \mathcal{I}^*} \gamma_k = \rho. \tag{3.14}$$

We continue to update $\mathbf{s}$ according to the possible pivot sequence. Since $\mathcal{P} \subset \mathcal{I}^*$, thus for any $i \in \mathcal{P}$ there exists an iteration, such that variable x_i enters the basis for the first time after iteration j. When this happens, we delete its index from $\mathcal{P}$, thus $\mathcal{P} := \mathcal{P} \setminus \{i\}$. After finitely many iterations, such a set $\mathcal{P}$ is obtained, for which $\mid \mathcal{P} \mid = \mid \{l\} \mid = 1$. Once the size of $\mathcal{P}$ is one, let the first iteration when variable x_l enters be r. We show that in iteration r the choice of x_l is unique. In this case $s_l = \rho$, regardless whether x_l was moving in iteration j or not. Because of the pivot rule, $\rho < s_i$ if $i \in \mathcal{I}^* \setminus \mathcal{P}$ and since every variable with index $i \in \mathcal{P} \setminus \{l\}$ has at least once entered the basis after iteration j and is currently not in the

basis, their $\mathbf{s}$-values must be at least $\rho + 2$. On the other hand, if it was a basic variable in iteration j then its $\mathbf{s}$ value is at least $\rho + 1$. Thus 2 (a) also holds.

Since the variable x_l enters the basis in iteration r, and every other variable with index in $\mathcal{I}^*$ entering the basis after x_l already has a higher $\mathbf{s}$ value than x_l in basis $\mathcal{I}_{B_r}$, according to the MOSV rule, thus 2 (b) also holds. ■

Analyzing the proofs of the previous two lemmas we can conclude that the 1 (a) and 1 (b) requirements of the definition of $\mathbf{s}$-monotone index selection rule are satisfied with the proper update strategy used to define the pivot index preference. Proving property 2 (a) there are three important ingredients: (i) the assumption that the pivot sequence is infinite, (ii) the finiteness of the index set, and (iii) the monotone increasing property of the pivot index preference, namely that for the vectors $\mathbf{s}_{k+1}, \mathbf{s}_k \in \mathbb{R}^n : \ \mathbf{s}_{k+1} \geq \mathbf{s}_k$ and $\mathbf{s}_{k+1} \neq \mathbf{s}_k$ hold for any iteration $k \in \mathbb{N}$. Property 2 (b) explains the changes in the $\mathbf{s}$-values of those variables that belong to the index set $\mathcal{I}^*$ and have moved between two consecutive moves of the least preferred variable. This property depends strongly on the monotonicity of the pivot index preference and on the property 2 (a) too.

Now, we are ready to introduce generalizations of the MOSV and LIFO rules. Let us define these rules using their pivot index preferences [19].

Let the vector $\mathbf{s}$ be initialized as the zero vector. In a pivot when x_{i_k} leaves and x_{o_k} enters the basis in the k^{th} iteration, the values of $\mathbf{s}$ are modified to increase the favour of these variables.

Generalized-Last-In-First-Out rule (GLIFO): Let us consider a *strictly monotone increasing* sequence of positive rational numbers $\{p_k : k \in \mathbb{N}\}$, namely for all $k \in \mathbb{N}$ indices $p_{k+1} > p_k > 0$ hold.

$$s'_i = \begin{cases} p_k & \text{if } i \in \{i_k, o_k\}, \\ s_i & \text{otherwise,} \end{cases} \tag{3.15}$$

It is quite easy to show that this slight modification of the pivot index preference of LIFO rule will lead to an $\mathbf{s}$-monotone index selection rule, too. Namely, all the steps of the proof of Lemma 3.4.1, remain true; in fact after each variable that moves at all has moved at least once the sequences defined by GLIFO are the same as those by LIFO.

However, the generalization of MOSV define a significantly more general class of pivot sequences. We can generalize the MOSV rule as well by modifying its pivot index preference.

Generalized-Most-Often-Selected-Variable rule (GMOSV): Let us consider a *monotone increasing* sequence of positive rational numbers $\{p_k : k \in \mathbb{N}\}$, namely for all $k \in \mathbb{N}$ indices

$p_{k+1} \geq p_k$ hold.

$$s'_i = \begin{cases} s_i + p_k & \text{if } i \in \{i_k, o_k\}, \\ s_i & \text{otherwise,} \end{cases} \tag{3.16}$$

However, to show that GMOSV is an **s**-monotone index selection rule we need a slightly more careful analysis of the proof of Lemma 3.4.2.

The requirements 1 (a) and 1 (b) are simply satisfied because of the definition of the corresponding pivot index preference. Justifying 2 (a), we need to modify the definition of γ_i introduced in the proof of Lemma 3.4.2, slightly. Let

$$\gamma_i = \begin{cases} s_i, & \text{if } i \in \mathcal{M}_N \\ s_i + p_j, & \text{if } i \in \mathcal{M}_B \end{cases} \tag{3.17}$$

since we would like to analyze the situation in the iteration j. Taking into consideration the monotone increasing nature of the p_k sequence we are able to identify – after finitely many iterations – the least preferred variable x_l in some iteration r.

Showing the uniqueness of the choice of x_l in the iteration r we need to do only a slightly more careful analysis of the situation. It remains true that $s_l = \rho$, regardless whether x_l was moving in iteration j or not. Furthermore, because of the monotone increasing nature of the p_k sequence, $\rho < s_i$ if $i \in \mathcal{I}^* \setminus \mathcal{P}$ and since every variable with index $i \in \mathcal{P} \setminus \{l\}$ has at least once entered the basis after iteration j and now is not in the basis, their values in **s** must be at least $\rho + p_u + p_v$, where $r > u, v > j$. On the other hand, if it was a basic variable in iteration j than its **s** value is at least $\rho + p_u$, where $r > u > j$. Since $p_u \geq p_j$ and $p_v \geq p_j$ hold we have verified that 2 (a) also holds.

Our last task is to show that 2 (b) holds as well. For this, let us collect all available information, namely we know that at the iteration r the following inequalities

$$s_l^r < s_i^r, \qquad \forall i \in \mathcal{I}^* \setminus \{l\} \tag{3.18}$$

hold and that $s_l^t = s_l^r + p_r$. Let $i \in \mathcal{I}^* \setminus \{l\}$ be the index of such variable x_i that has moved between the iteration r and t, at least once, for instance in iteration k, where $t > k > r$ holds. Then

$$s_i^t \geq s_i^{k+1} = s_i^k + p_k \geq s_i^k + p_r \geq s_i^r + p_r > s_l^r + p_r = s_l^t, \tag{3.19}$$

thus 2 (b) is really satisfied.

Now we are ready to state the following result [19].

Lemma 3.4.3 *GLIFO and GMOSV index selection rules are* **s***-monotone index selection rules.*

It is easy to check that when the sequence is constant $p_k = 1$ for all k, then GMOSV becomes MOSV, and that GMOSV is indeed a generalization of MOSV (e.g. it allows for switching from MOSV to LIFO).

3.4.2 s-monotone rules for the linear complementarity problem

To apply the **s**-monotone index selection rules for the criss–cross method for linear complementarity problems, a small extension is necessary compared to that one for linear programming [19].

Definition 3.4.4 *We say that an* **s***-monotone pivot rule used while solving an LCP problem is symmetric, if for any index $i \in \mathcal{I}$ its and its complementary pair's* **s***-value is the same, i.e. $s(i) = s(\bar{i})$ for all $i \in \mathcal{I}$.*

The above extension is not strictly necessary for all pivot algorithms using **s**-monotone index selection rules solving linear complementarity problems, as we show for the case of convex quadratic primal simplex in Chapter 7.

Chapter 4

A test framework and test sets

To create a comprehensive, readily accessible and medium difficult set of test problems, we have used a selection of problems from the following 3 databases:

1. NETLIB [2]
2. Miplib 3 [1]
3. Miplib 2010 [1]

From among the Miplib collections, we have only considered the linear part of the problem. The detailed description of the results will be in the next chapter.

Benchmark sets are typically compiled from problems that are either computationally hard and/or otherwise complex and interesting; and while they provide suitable grounds for testing they set a very high requirement standard. Some of the NETLIB problems are known to be numerically challenging [57] - especially in the absence of presolve - and while the Miplib sets were created to be difficult or large integer problems, their relaxations are typically easier to solve. Many of the Miplib problems are known to be highly degenerate, making them ideal for testing flexible index selection rules. In section 5.9, numerical results are presented for a separate selection of industrial problems taken from the literature [46].

The purpose of these benchmarks is the comparison of solvers. Currently, the most comprehensive collection of up-to-date tests are published by Hans Mittelmann on [50] .

4.1 Testing framework

Different practical implementations of the revised primal and dual simplex methods still form one of the bases of most mathematical programming applications. Several different simplex

algorithm variants and different pivot selection rules have been invented [62], but with the exception of problem specific alternatives like the network simplex method, the alternative methods have not gained similar importance, and are generally not widely available.

To ensure finiteness for degenerate problems, a large number of well know index selection rules are available. While many of these rules are mainly of theoretical interest, there are a significant number [19] of flexible index selection rules [5], claiming practical applicability by offering various degrees of flexibility while still providing theoretical finiteness [17].

Traditional index selection rules are typically not applied directly in implementations of the simplex methods, as in the case of stalling progress is most often achieved by some means of perturbation [49]. Also, it is difficult to provide solid numerical evidence to the practical benefits of index selection rules. The reasons are many fold:

- implementations have to either compete against well established commercial and academic solvers with several years of work hours invested in them,
- or alternatively are based on own implementations where the implementation efforts are often dominated by addressing numerical challenges,
- efficiency of the implementations tend to be a function of method specific algorithmic features (like shifting for the primal method, or bound flipping for the dual method, quality of the basis factorization and updates) making direct comparisons of the algorithms measure the depth of implementations instead of the algorithms themselves.
- cycling is a relatively rare phenomenon as often claimed, see [12](chapter 3, Pitfalls and How to Avoid Them) and [51](chapter 9.4, Do computer codes use the lexico minimum ratio rule?).

In consequence, numerical results using different implementations are hard to judge and to be used as a reference (for a detailed description on implementing the simplex methods see Maros's book [49]).

The purpose in Chapter 5 will be to provide numerical evidence that the monotonic build-up (MBU) simplex method [6] is a viable alternative to the tradition primal simplex method, and to provide a measure of the advantages of the use of flexible index selection rules.

Chapter 3 of the thesis stated the form of the linear programming problem used and summarized the algorithmic results the paper is built on: the traditional revised simplex method and the MBU-simplex method, the s-monotone index selection rules including the

Last-In-First-Out (LIFO) and the Most-Often-Selected-Variable (MOSV) rules and some of their generalizations that make use of the flexibility provided.

We propose to address the difficulties in comparing different pivot methods and index selection rules. The main highlights are

- the standard form of the linear problems is used,
- the tests are carried out on public benchmark sets which are transformed to the standard form,
- only such algorithmic features are used that are generally applicable to all of the investigated algorithms,
- the numerical linear algebra used is based on the API of a well established linear programming solver,
- the implementations are made available for reference and reproducibility [52].

Our goal is to compare the properties and efficiency of the selected pivot methods with the flexible index selection rules in a suitable numerical environment. For this purpose, we work on the standard form of the problems and without applying a presolver - which would have undone the conversion to the standard form anyway.

We aim to provide a framework for the comparisons, where the comparison is based on the main algorithmic features and the index selection rules, not between the depth of the implementation. In order to provide the uniform test environment for different algorithms and their variants depending on the anti-cycling pivot rules in our implementations, the basic version of the algorithms are implemented without further computational techniques that usually accelerate the computations, but often are specific to the algorithm in question, and would thus make a fair comparison difficult. To implement such improvements for both algorithms in an appropriate and equally efficient way is a nontrivial, challenging and interesting task.

The algorithms are initiated from a slack basis, and the traditional first phase method is used to produce an initial feasible bases for both algorithms.

One of the crucial numerical properties of most simplex methods is to maintain monotonicity in the selected (ether primal or dual) form of feasibility. In the presence of numerical error, one of the most widely applied methods to address infeasibility occurring due to numerical error is shifting. Shifting removes numerical infeasibility by rounding small negative

numbers to zero in the transformed right hand side. In turn, to remove the side effects when re-inverting (re-introducing infeasibility and change in the objective due to changes in the solution can result in breaking the monotonicity of the objective that in turn can cause cycling if not addressed in other ways even in the presence of the index selection rules), the original right hand side is also shifted by adding the original column of the shifted variable to the non-transformed right hand side by the same amount. This perturbation has to be removed at the end, and a clean up phase applied. Our implementation does not apply this technique.

On the other hand, maintaining the concept of dual monotonicity in the monotonic build-up simplex method has proven to be crucial for its correctness (not only to its performance). However, the required monotonicity can be achieved by simply not allowing it to chose a column twice as a driving variable, even if the round of errors make a feasible dual value slightly infeasible again (relative to the selected optimality tolerance).

To achieve a fair comparison, problems where these rules were causing significant deviance from feasibility or optimality have been removed from the test sets.

To address scaling, most solvers automatically re-scale the problems. We have tested the algorithms on the transformed (to standard from) original and on a re-scaled version of the test sets. We used the exposed scaling functionality of the Xpress solver, and have observed an increase of 2% in success rate on the selected test set. As the improvement was not significant, we opted not to scale the problems.

The selected problems were converted to the standard form. The average increase in the number of rows, columns and non-zeros is shown in Table 4.1:

Number of extra rows	+2139%
Number of extra columns	+143%
Number of extra nonzeros	+27%

Table 4.1: The average increase in problem size as a result of the conversion to standard form.

The very large increase in terms of row numbers is down to a relatively few problems that are flat (have a much larger number of columns than rows and the columns are both upper and lower bounded which is typical for several binary problems. On problems with both lower and upper bounds, the conversion to the standard form introduces $+n$ new rows, which can be a significant increase if $n \gg m$ where n is the number of original columns and m is the number of original rows).

As a note, Xpress itself takes 39% longer on average to solve the standard forms with presolve off (the choice of the primal algorithm is important in this respect, as converting the variables with both lower and upper bounds affects dual more through the missed opportunities of bound flips than primal, while the primal ratio test would need to consider both bounds anyway).

The criteria for a problem to be included was the following:

1. all algorithm and index selection rule combinations solved the problem successfully,
2. the optimal objective matched the value reported by Xpress,
3. Xpress was able to solve the problem within 5 minutes,
4. a time limit of 1 hour was used for Netlib, and 5 minutes for the Miplib datasets.

Using this criteria, 108 problems were selected, with the average problem statistics presented in Table 4.2. The 11 assembly line balancing and workforce skill balancing problems (Balancing) from [28] and [46] were treated as a separate set.

	Total size	Selected	Average rows	Average columns	Average density
Netlib	98	43	505	1082	2.07%
Miplib3	64	28	660	1153	1.20%
Miplib2010	253	37	1493	2500	0.63%
Balancing	11	11	279	470	3.03%

Table 4.2: The average size statistics of the selected test problems. The average values are calculated on the selected subset of problems, using the standard form.

4.2 Implementation details

The first version of implementations for all algorithms were implemented in Matlab (as a continuation of [53]), using Matlab's QR decomposition routines. The QR decomposition was chosen over the LU decomposition simply because Matlab has a built-in rank 1 update for the QR decomposition, but no such for the LU decomposition.

The Matlab implementation was used successfully for testing the criss–cross method for both the linear programming problem, and the linear complementary related problems, where the bottleneck was usually the lack of sufficiency of the matrix for the LCP case, and the combinatorial nature for "convergence" for the criss–cross method for both. As a result, the size of the problems solved by the criss–cross method remained small compared to what is considered average size normally in linear programming. Some relevant results have been published in [32].

In contrast to the criss–cross implementation, in the case of the primal simplex and monotonic build-up simplex implementation, it quickly becomes obvious that the bottleneck is in the numerical behavior of the algorithms. It must be noted though, that it is also true that in the case of the criss–cross, the lack of indicators for numerical errors also means that the numerical experiments could simply be oblivious to the presence of numerical issues. In practice, this meant convergence issues for the criss–cross. For the simplex variants, numerical issues are often very simple to notice, with these cases being the most prominent: when the basis becomes infeasible even though the ratio test suggests it should remain feasible, or when the objective function takes a backward step even though an improving direction (column) has been selected.

As explained earlier in the chapter, to be able to provide an independent testing framework, we have aimed to avoid algorithm specific considerations to handle numerical issues, and so the choice has been to replace the numerical linear algebra used with one that is targeted at solving linear programming problems, in this case Xpress's C API (application programming interface).

Xpress has been selected due to two major considerations. The optimization market has changed considerably in recent years, and these considerations are no longer as straightforward as they used to be, however they do not invalidate or outdate the numerical results: they simply mean that the choice of Xpress would not be as obvious as it was when the research and work on the thesis have started.

- From among the 2 major vendors of the time, Ilog CPLEX and Dash Xpress, the university had an up to date Xpress license with support agreement, and only had a

CPLEX version that was several years behind.

Since, then neither Ilog nor Dash exists: the first was bought up by IBM, the later by FICO. Also, there is a third major vendor, GUROBI, consisting of developers who have left CPLEX. Due to increased competition, since then all these vendors provide free licenses for academic work. According to public benchmarks (like Hans Mittelman's) the major vendors performance is very comparable, so Xpress could have been the choice if the others had been an alternative before.

- There were two reasons for not using an arbitrary precision linear algebra package. One was performance / speed considerations, as arbitrary precision packages can be very computationally extensive, while the other was availability.

  These considerations are still valid, with the amendment that SCIP, the academic solver developed by the ZIB Institute Berlin has developed an iterative refinement technique, that can generate basis solutions to arbitrary precisions by iteratively refining the digits up to the desired accuracy. This technology could become a framework to consider in the future, but at the moment it does not efficiently tie onto the basis updates to the best knowledge of the author.

The support agreement for Xpress proved to be very beneficial. During testing, two issues were found and reported to Xpress, both fixed very quickly, for which the author wishes to express her gratitude to the Xpress development team. Both issues were related to how the API was used: the relevant exposed API functions were designed and tested with tools for generating cuts during a MIP search in minds, and have not been used in the way as in these tests before.

It must be mentioned that the author of this thesis has since been employed by FICO as a software quality assurance engineer for the Xpress optimization team. All numerical results presented have been prepared and submitted to publication before the date the author joined FICO. The results after this date are more theoretical / modelling focused [33, 35] and [55].

4.2.1 Functions used during the standard form creation

This section provides the minimal documentation of the Xpress functions [25] that are necessary to read the code related to the thesis.

In the conversion to the standard form and before starting to process the pivot algorithm, the code should declare the problem with the *XPRSprob xprob;* line. The following functions were used during the creation of the standard form of the linear programming problem. After all of the functions a small example shows the usage of it, taken from the code itself.

XPRSreadprob reads in a problem from MPS or LP format

XPRSreadprob(xprob, argv[1], "") reads the MPS or LP format problem into the *xprob* object variable from the 1^{st} argument of the *main* function.

XPRSgetintattrib retrieves the value of an attribute of the problem

XPRSgetintattrib(xprob, XPRS_ROWS, &nrow) retrieves the number of the rows from the problem and puts the value into the *nrow* variable. The second parameter - $XPRS\_ROWS$ - sets which attribute of the problem is being retrieved.

XPRSgetintattrib(xprob, XPRS_COLS, &ncol) retrieves the number of the rows from the problem and puts the value into the *ncol* variable. The second parameter - $XPRS\_COLS$ - sets which attribute of the problem is being retrieved.

XPRSgetrowtype retrieves back the type of the row, which can be (N - neutral, L - less or equal, E - equal, R - ranged, both lower and upper bounds, G - greater or equal)

XPRSgetrowtype(xprob, &rowtype, i, i); retrieves the type of the i^{th} row and puts it into the *rowtype* variable. The function could get back the type of more than one row at the same time. The last two parameters sets the index range of the rows.

XPRSgetrhs retrieves the right hand side of the problem

XPRSgetrhs(xprob, rhs, 0, nrow-1) retrieves the right hand side vector of the problem and puts it into the *rhs* variable. The last two parameters set the range of the right hand side indices to get.

XPRSgetrhsrange retrieves the range of the right hand side, which is the range of the row

XPRSgetrhsrange(xprob, &range, i, i); retrieves the row range of the i^{th} element of the right hand side and puts it into the *range* variable. The function may retrieve the range of more than one element at the same time. The last two parameters sets the range for which the function retrieves the ranges.

XPRSgetobj retrieves the objective coefficients of the problem

XPRSgetobj(xprob, &dobj, j, j); retrieves one value from the objective function coefficients of the problem and puts them into the *dobj* variable. The last two

parameters set the index range of the objective function for which to retrieve the values.

XPRSchgrowtype changes the type of a row (N, L, E, R, G) represents the same types as for the get function above.

XPRSchgrowtype(xprob, 1, &i, &newtype); changes the type of the row to a new type which is in the *newtype* variable. The function could change the type of more than one row at the same time, indices described by the 3rd argument. The second parameter sets the number of the row types to change.

XPRSgetcols retrieves a column or more columns as a sparse vector

XPRSgetcols(xprob, colstart, rowindices, colvalues, nrow, &ncoeff, j, j); retrieves the non-zero elements of a column.

XPRSgetlb retrieves the lower bound of a column

XPRSgetlb(xprob, &dlb, j, j); retrieves the lower bound of the j^{th} column in the problem and puts into the *dlb* variable. The function could get back the lower bound of more than one column at the same time. The last two parameters sets the index range of the columns.

XPRSgetub retrieves the upper bound of a column

XPRSgetub(xprob, &dub, j, j); retrieves the upper bound of the j^{th} column in the problem the *dub* variable. The function could get back the upper bound of more than one column at the same time. The last two parameters sets the index range of the columns.

XPRSchgbounds changes the bounds of a column

XPRSchgbounds(xprob, 1, &j, &qubtype, &dpinf); changes the bounds of the j^{th} column. The function can change the bounds of more than one column at the same time. The second parameter sets the number of the columns to change. In this example, as used in the conversion code, the function changes the upper bound of column j to infinity.

XPRSchgcoef changes some coefficients in the matrix

XPRSchgcoef(xprob, rowindices[k], j, -colvalues[k]); changes the coefficient of the j^{th} column to the values are in the last parameter. The second parameter sets

the row index of the coefficient. The function changes only one coefficient in the matrix at one time, there are other functions to change multiple ones efficiently.

XPRSchgrhs changes the right hand side in the matrix

XPRSchgrhs(xprob, 1, &i, &rhs[i]); This function changes the i^{th} value of the right hand side to the value in the *rhs* array. The function could change more than one right hand side item at the same time. The second parameter sets the number of the items, and the third parameter sets the indices for the rows for which to change the values.

The following other functions were used for the implementation of the primal simplex algorithm and the MBU-simplex algorithm.

XPRSloadbasis loads the (initial) basis

XPRSloadbasis(xprob, rstatus, cstatus) loads the basis and sets which variables are in the basis.

XPRSgetpivotorder retrieves the pivot order: which variable is in the basis in which row

XPRSgetpivotorder(xprob, mpiv) returns the pivot order of the basis variables and puts it into the *mpiv* variable.

XPRSpivot carries out a pivot

XPRSpivot(xprob, pcol+nrowspare, mpiv[prow]) does a pivot. The second parameter sets the index of the variable who enter the basis; the third parameter sets the index of the variable who leave in the basis. It is important that column indices start from the sum of these two attributes: XPRS_ROWS and XPRS_SPAREROWS.

XPRSftran multiplies with the inverse of the basis from left side

XPRSftran(xprob, rhs) calculates the transformed right hand side ($B^{-1}\mathbf{b}$), and puts the result into the same vector (*rhs*). The *rhs* array in this case needs to be initiated to be equal the original right hand side vector.

XPRSbtran multiplies with the inverse of the basis from right side

XPRSbtran(xprob,cb) calculates the $\mathbf{c}_b B^{-1}$, and puts the result into the same vector (*cb*). In this case the *cb* array needs to be initialed to be the basic part of the objective function.

4.2.2 The MPS file format

Although not directly relevant to the topic of the thesis, as all the public benchmark models the thesis works with are available in the so called MPS (Mathematical Programming System) format, it is possibly useful to present a short description of the format.

The format has originally been introduced by IBM to describe linear and mixed integer linear programming problems [29]. At the time of the invention of the format, punch cards were still common and the format reflects this by prescribing fixed column positions for all the various data; this version is called the fixed format MPS file. Since then, the column position restrictions have been removed, called the free format, and several extensions exits for example to store quadratic data either for the objective or the constraints.

Several of the public test problems are in the fixed format MPS, especially the ones in NETLIB. Not all fixed format MPS files are compatible with the free format, as spaces were originally allowed as part of the column and row names.

The format is columns oriented, in contrast to the traditional algebraic representations. This also makes it necessary to name all rows. The fields in the fixed format version start at positions 2., 5., 15., 25., 40., and 50. The one exception is the section headers, that start at position 1. In the free format, they start anywhere, and are separated by spaces or tabulators.

Typically, these files contain all capital letters, but with the exception of the section headers small letters are also allowed. Obviously, the names of the rows and columns do not directly affect the solves of the problems the MPS files describe. However, the order of the columns and rows does.

In the original format, the sense of the optimization, i.e. if it is minimization or maximization was not included. Interestingly, different solvers assume different default optimization senses; other ones require the user to explicitly define the objective sense. Converting between minimization or maximization inside a model itself can be done by multiplying the objective by -1, but we need to remember to multiply the optimal solution value with -1 as well ($\max \mathbf{c}^T\mathbf{x} = -\max(-\mathbf{c})^T\mathbf{x}$).

A line can be made a comment by putting a * in the first character. It is good practice not to have spaces in the names.

An MPS file is made up of sections. Section `NAME` is followed by the name of the model. The `ROWS` section describes the constraints, defining their names and type. A type `E` means equal constraint, `L` a less or equal ($\leq$) constraint, `G` a greater or equal one, ($\geq$), and finally

`N` for a constraint with no sense which is called a neutral, or free row. The first one of these is taken as the objective. There may be multiple objectives described this way.

The largest part of most MPS files is the `COLUMNS` section describing the constraint matrix A. The columns must follow one another continuously, but the order according to the rows is flexible.

The `RHS` section defines the right hand side values. There can be several different right hand side alternatives defined. The `BOUNDS` section, which is optional includes the bounds for the variables, `UP` defining an upper or `LO` a lower bound. It is possible to define integrality constraints in the bounds section. For example `LI` is the same as `LO` but the variable is also required to be integer. For variables not listed in the bounds section, a default restriction of $\mathbf{x} \geq 0$ applies.

Also optional is the `RANGES` section. This defines constraints that are both lower and upper bounded, i.e. has both less or equal and greater or equal sides.

An MPS files is closed by the `ENDATA` section.

The free MPS format defines a few extensions. Most newer public benchmark sets are in free format MPS, like the MIPLIB sets. The free MPS format is similar to the fixed format, but it introduces several convenience features. As mentioned before, the fields are no longer fixed, but can be arbitrarily separated by spaces or tabulators. The new `OBJSENSE` section allows to define the objective sense, to be defined in between sections `NAME` and `ROWS`, with the possible values of `MAXIMIZE, MAX, MINIMIZE, MIN`.

Normally, the first neutral row is used as the objective. The `OBJNAME` section allows to select any row as the objective.

As an example, consider the following small problems:

$$\begin{aligned} \max(\mathbf{x}_1 + 2\mathbf{x}_2 + 4\mathbf{x}_3 + \mathbf{x}_4) \\ \mathbf{x}_1 + \mathbf{x}_3 + \mathbf{x}_4 \leq 5 \\ \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_4 \leq 6 \\ \mathbf{x}_3 \leq 7 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \geq 0 \end{aligned}$$

$$\begin{aligned} \max(\mathbf{x}_1 + 2\mathbf{x}_2 + 3\mathbf{x}_3 + 2\mathbf{x}_4 + 3\mathbf{x}_5) & \\ \mathbf{x}_1 + 2\mathbf{x}_2 + \mathbf{x}_4 + \mathbf{x}_5 &\leq 110 \\ \mathbf{x}_1 + 2\mathbf{x}_3 + \mathbf{x}_4 + \mathbf{x}_5 &= 80 \\ 2\mathbf{x}_2 + \mathbf{x}_5 &\geq 40 \\ \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5 &\geq 0 \end{aligned}$$

and convert it to the free (it uses the OBJSENSE extension) MPS format:

```
NAME EXAMPLE1
OBJSENSE
 MAX
ROWS
 N  OBJECTIVE
 L  CTR1
 L  CTR2
 L  CTR3
COLUMNS
    XONE      OBJECTIVE  1
    XONE      CTR1       1
    XONE      CTR2       1
    XTWO      OBJECTIVE  2
    XTWO      CTR2       1
    XTHREE    OBJECTIVE  4
    XTHREE    CTR1       1
    XTHREE    CTR3       1
    XFOUR     OBJECTIVE  1
    XFOUR     CTR1       1
    XFOUR     CTR2       1
RHS
    RHS1      CTR1       5
    RHS1      CTR2       6
    RHS1      CTR3       7
ENDATA
```

```
NAME EXAMPLE2
OBJSENSE
 MAX
ROWS
 N  OBJECTIVE
 L  CTR1
 E  CTR2
 G  CTR3
COLUMNS
    XONE      OBJECTIVE    1
    XONE      CTR1         1
    XONE      CTR2         1
    XTWO      OBJECTIVE    2
    XTWO      CTR1         2
    XTWO      CTR3         2
    XTHREE    OBJECTIVE    3
    XTHREE    CTR2         2
    XFOUR     OBJECTIVE    2
    XFOUR     CTR1         1
    XFOUR     CTR2         1
    XFIVE     OBJECTIVE    3
    XFIVE     CTR1         1
    XFIVE     CTR2         1
    XFIVE     CTR3         1
RHS
    RHS1      CTR1       110
    RHS1      CTR2        80
    RHS1      CTR3        40
ENDATA
```

Chapter 5

Flexible index selection rules for linear programming

This chapter investigates the well-known pivot algorithms like the primal simplex algorithm [21] and the primal monotonic build-up simplex algorithm [6] with **s**-monotone index selection rules which are finite algorithms for solving linear programming problems. We present proofs that these algorithms are finite when **s**-monotone index selection rules are used, and present a numerical study to demonstrate their practical efficiency.

This chapter is mainly based on [19]. The finiteness results using **s**-monotone index selection rules are based on the results of [17, 19, 20]. The numerical study and results are the author's own results and have been published in [34] and partially in [19].

For the sake of completeness, we also show that the criss–cross algorithm [60] is also finite with **s**-monotone index selection rules; the result has been published in [54]

5.1 The primal simplex algorithm with s-monotone index selection rules

We show the important part of the proof (i.e. defining the almost terminal tableaus and deriving the contradiction using the orthogonality theorem) for the primal simplex algorithm (SA) first. The pseudo-code of the simplex algorithm with **s**-monotone index selection rule is the following:

The primal simplex algorithm with s-monotone index selection rules

input data:

$A \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^n$ a feasible basis B, initialized $\mathbf{s}$ vector;

output:

An optimal solution, or a certificate that the problem is unbounded;

begin

$\mathcal{I}_N^- := \{i \in \mathcal{I}_N | \bar{c}_i < 0\}$;

while $(\mathcal{I}_N^- \neq \emptyset)$ **do**

let $q \in \{i \in \mathcal{I}_N^- | \bar{c}_i < 0\}$ be arbitrary with a maximal $\mathbf{s}$-value;

if $(\mathbf{t}_q \leq 0)$ **then**

STOP: problem is unbounded, certificate is $\mathbf{t}_q$;

else

let $\vartheta := \min\left\{\frac{\bar{b}_i}{t_{iq}} \mid i \in \mathcal{I}_B,\ t_{iq} > 0\right\}$ denote be result of the primal ratio test;

let $p \in \mathcal{I}_B$ arbitrary for which $\frac{\bar{b}_p}{t_{pq}} = \vartheta$ and has a maximal $\mathbf{s}$-value;

endif

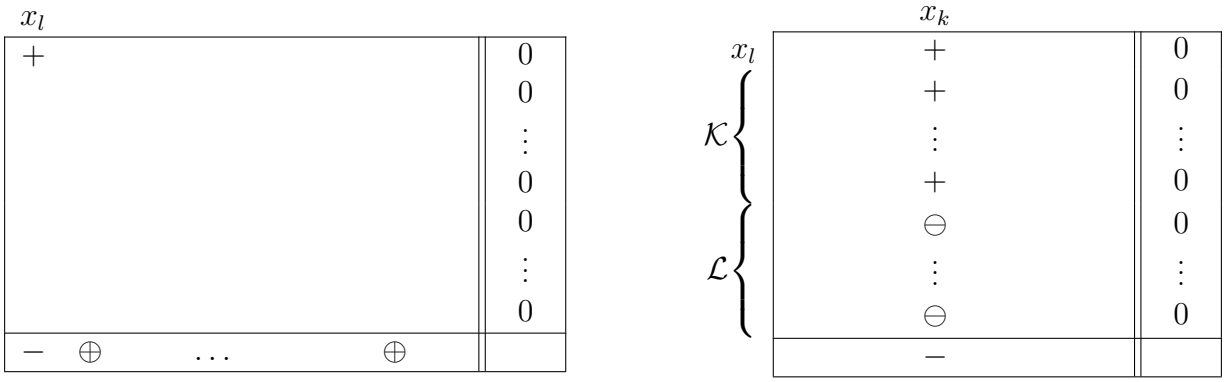
pivot on (p, q);

endwhile

STOP: The solution is optimal;

end

It is easy to verify that in a minimal cycling example all the variables are moving during a cycle and that the right hand side values are zeros (any nondegenerate pivot would improve the objective). Thus the minimal cycling example should be completely primal degenerate, otherwise the objective would improve.



x_l enters the basis in basis B'

x_l leaves the basis B''

Figure 5.1: Key tableaus of a minimal cycling example for simplex algorithm with s-monotone index selection rule.

Consider basis B' of the minimal cycling example, when the least preferred variable x_l enters the basis (according to the **s**-monotone rule there must be such a situation). According to the column selection rule of the simplex algorithm, the objective function row of the pivot tableau for basis B' must have a negative coefficient for the nonbasic variable x_l and nonnegative coefficients for all other nonbasic variables. (This is the structure of the first almost terminal pivot tableau.)

In a minimal cycling example, since every variable moves infinitely many times, variable x_l must leave the basis. Consider basis B'' when this happens the first time, i.e. x_l leaves the basis for the first time after B'. According to the **s**-monotone index selection rule, the choice of the leaving variable is selected from those basic variables that are least preferred, in our case x_l is such a variable. (This defines the second almost terminal pivot tableau.)

Consider the vector $\mathbf{t}'^{(c)}$ corresponding to the objective function row for basis B' and the vector $\mathbf{t}''_k$ corresponding to the entering variable x_k for basis B''. Let

$$\mathcal{K} = \{i \in \mathcal{I}_{B''} \mid t''_{ik} > 0\} \setminus \{l\}, \quad \text{and} \quad \mathcal{L} = \{j \in \mathcal{I}_{B''} \mid t''_{jk} \leq 0\}. \tag{5.1}$$

Then

$$\mathbf{t}^{(c)'T}\mathbf{t}''_k = \sum_{i\in\mathcal{K}} t'_{ci}t''_{ik} + \sum_{j\in\mathcal{L}} t'_{cj}t''_{jk} + t'_{cl}t''_{lk} \leq t'_{cl}t''_{lk}, \tag{5.2}$$

since $t'_{cj} \geq 0$ and $t''_{jk} \leq 0$ holds for all $j \in \mathcal{L}$, and $t'_{ci} = 0$ for all $i \in \mathcal{K}$ because of the 1 (b) criterion (of Definition 3.4.3 on page 54), the values of $\mathbf{s}$ may not decrease, and those variables that have been pivoted on after basis B' must have a larger **s**-value than variable

x_l. By the 2 (b) criterion of **s**-monotone index selection rule, the variables corresponding to the index set $\mathcal{K}$, have not moved since basis B' and so have a corresponding zero value in $\mathbf{t}'^{(c)}$. Since $t'_{cl} < 0$ and $t''_{lk} > 0$, we have $\mathbf{t}^{(c)'T}\mathbf{t}''_k < 0$, contradicting the orthogonality theorem. This proves that the primal simplex algorithm with **s**-monotone index selection rules are finite.

Theorem 5.1.1 *The primal (dual) simplex algorithm with* **s***-monotone index selection rules are finite for linear programming problems.*

5.2 The primal MBU-simplex algorithm with s-monotone index selection rules

The monotonic build-up simplex algorithm (MBU-SA) was introduced in [6]. Starting from a primal feasible basis, the algorithm achieves the feasibility of the dual variables one by one while also keeping those dual variables that have become feasible before feasible throughout the algorithm. During such a major iteration of the algorithm, the primal solution may become infeasible, however the solution is always primal feasible again once the dual feasibility of the selected column is achieved.

We repeat the pseudo-code of the algorithm, this time specialized to using the **s**-monotone index selection rule.

The primal MBU-simplex algorithm with s-monotone index selection rules

input data:

$A \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^n$ a feasible basis B.

output:

An optimal solution or a certificate that the problem is unbounded;

begin

initialize vector $\mathbf{s}$;

$\mathcal{I}_N^- := \{i \in \mathcal{I}_N | \bar{c}_i < 0\}$;

while $(\mathcal{I}_N^- \neq \emptyset)$ **do**

let the driving variable $s \in \{i \in \mathcal{I}_N | \bar{c}_i < 0\}$ be arbitrary;

while $(\bar{c}_s < 0)$ **do**

let $\mathcal{K}_s = \{i \in \mathcal{I}_B | t_{is} > 0\}$;

if $(\mathcal{K}_s = \emptyset)$ **then**

STOP: problem is unbounded, certificate is $\mathbf{t}_s$;

else

let $\vartheta := \min\left\{\frac{\bar{b}_i}{t_{is}} | i \in \mathcal{K}_s\right\}$ the value of the primal ratio test;

let $r \in \mathcal{K}_s$ arbitrary such that $\frac{\bar{b}_r}{t_{rs}} = \vartheta$ and has a maximal $\mathbf{s}$-value;

let $\theta_1 := \frac{|\bar{c}_s|}{t_{rs}}$, and let $\mathcal{J} = \{i \in \mathcal{I}_N | \bar{c}_i \geq 0, t_{ri} < 0\}$;

if $(\mathcal{J} = \emptyset)$ **then** $\theta_2 := \infty$;

else

$\theta_2 := \min\left\{\frac{\bar{c}_i}{|t_{ri}|} | i \in \mathcal{J}\right\}$ the value of the dual ratio test;

let $q \in \mathcal{J}$ arbitrary such that $\theta_2 = \frac{\bar{c}_q}{|t_{rs}|}$ and with maximal $\mathbf{s}$-value;

endif

if $(\theta_1 \leq \theta_2)$ **then**

pivot on (r, s) (*driving pivot*);

else

pivot on (r, q) (*auxiliary pivot*);

endif

endif

endwhile

endwhile

STOP: the solution is optimal;

end

A pivot in the column of the driving variable is called a *driving pivot*, while any other pivot an *auxiliary pivot*. Theorems 3.1.1 and 3.1.2 stating that the MBU-simplex algorithm is well-defined [6].

If the problem is nondegenerate (both from the primal and from the dual side) then as the objective function strictly increases in each iteration [6] the algorithm must be finite, just like in the case of the traditional simplex method.

In the original paper [6], lexicography was used to ensure finiteness for degenerate problems. In [19] if was shown that the algorithm is finite whenever an **s**-monotone pivot rule is applied.

Lemma 5.2.1 *[19] Both driving- and auxiliary pivots may only increase the reduced cost of the driving variable.*

Proof. After a pivot on the driving variable - which was the selected dual infeasible variable - it becomes feasible, while an auxiliary pivot increases the reduced cost of the driving variable without making it nonnegative, or leaves it unchanged. (Follows from the Theorem 3.1.1.) ■

There are other monotone properties that can be proved for the primal MBU-algorithm.

Lemma 5.2.2 *In any internal loop, i.e. a sequence of auxiliary pivots made by the MBU-algorithm while working on making the same driving variable x_r feasible, the value* $\max\left\{\frac{\bar{b}_i}{t_{is}} \mid \bar{b}_i < 0\right\}$ *never decreases [19].*

Lemma 5.2.3 *For any $a, b, \Theta \in \mathbb{R}$ for which $b \neq 0$ and $\frac{a}{b} = \Theta$ and for $c, d, \lambda \in \mathbb{R}$ for which $d \cdot \lambda \neq 0$ and $b + \lambda d \neq 0$ holds, then if $\frac{a+\lambda c}{b+\lambda d} = \Theta$, then $\frac{c}{d} = \Theta$ [19].*

We are ready to prove that the MBU-simplex algorithm with **s**-monotone pivot rules is finite. The proof is based on a contradiction. Consider a minimal example on which the algorithm is not finite. As the number of possible bases is finite, this is only possible if the algorithm cycles, and the same basis are generated infinitely many times. Due to minimality, in such an example each variable must move infinitely many times except an infeasible variable that must remain the selected driving variable for all bases of the cycle.

Lemma 5.2.4 *Assume that we would like to solve a minimal cycling example using the primal MBU-simplex algorithm. The following properties hold:*

1. *All pivot tableaus generated by the algorithm are dual degenerate for all variables except a single variable. This variable remains the same throughout the algorithm and never enters the basis.*

2. *All variable moves infinitely many times, except one, which never enters the basis.*

3. *No driving pivot is made.*

4. *The primal ratio test always yields the same value. Furthermore, in any basis generated by the algorithm, for the column r of the driving variable, the ratio $\frac{\bar{b}_i}{t_{ir}}$ is the same for all i, where $t_{ir} \neq 0$.*

Proof. By Lemma 5.2.1, a pivot in any non dual-degenerate column strictly increases the value of the driving variable. While a pivot in a nondegenerate column leaves the column of the short pivot tableau nondegenerate, a degenerate pivot doesn't change the row of the objective function in the tableau.

As had been argued before, for a minimal cycling example there is an infeasible variable x_r that remains the driving variable and stays infeasible. As a consequence, only auxiliary pivots are made by the algorithm. Due to the minimality of the cycling example, all other variables must move infinitely many times. As $\bar{c}_i = 0$ for all $i \neq r$ it follows that 1 holds.

Statements 2 and 3 follow immediately from 1.

Since the driving variable never changes according to Theorem 3.1.1 and Lemma 5.2.2, the value of the ratio test becomes a constant after finitely many iterations. Using Lemma 5.2.3, it can be stated that the value of the ratio test remains constant. ■

According to Lemma 5.2.4 a minimal cycling example includes a single infeasible dual variable, which then must be also the driving variable. Both the primal and dual ratio tests are trivial and as such the selection of indices are solely based on the index selection rule. Let x_r be the driving variable. Let variable x_l in basis B' correspond to the situation of the second criterion of **s**-monotone index selection rules, and let B'' be the first basis when the selected variable x_l leaves the basis after B'. (Observe, that since the driving variable never enters the basis, $l \neq r$.) Using the observations stated in Lemma 5.2.4, the almost terminal pivot tableaus for bases B' and B'' feature a sign structure as presented in Figure 5.2.

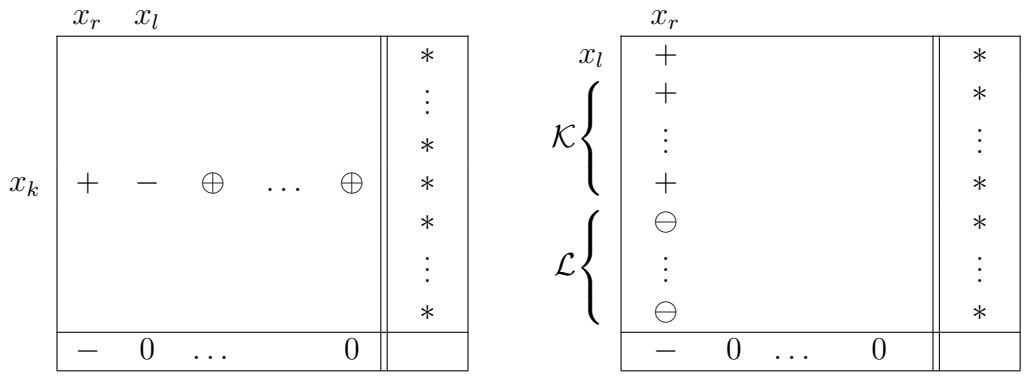


Figure 5.2: Almost terminal pivot tableaus for the MBU-simplex algorithm.

Theorem 5.2.1 *The MBU-simplex algorithm with* **s***-monotone index selection rule is finite. [19]*

Proof. Assume to the contrary that the algorithm is not finite, and consider a minimal cycling example with entering variable x_l and leaving variable x_k in basis B' described in the second criterion of **s**-monotone index selection rules, and basis B'' when variable x_l becomes non-basic for the first time after B'.

Consider vector $\mathbf{t}'^{(k)}$ for basis B' and vector $\mathbf{t}''_r$ for basis B''. Let

$$\mathcal{K} = \{i \in \mathcal{I}_{B''} \mid t''_{ir} > 0\} \setminus \{l\}, \text{ and } \mathcal{L} = \{j \in \mathcal{I}_{B''} \mid t''_{jr} \leq 0\}. \tag{5.3}$$

Then

$$\mathbf{t}^{(k)'T}\mathbf{t}''_r = \sum_{i\in\mathcal{K}} t'_{ki}t''_{ir} + \sum_{j\in\mathcal{L}} t'_{kj}t''_{jr} + t'_{kr}t''_{rr} + t'_{kl}t''_{lr} \leq t'_{kr}t''_{rr} + t'_{kl}t''_{lr}, \tag{5.4}$$

using that $t'_{kj} \geq 0$ and $t''_{jr} \leq 0$ for all $j \in \mathcal{L}$, and $t'_{ki} = 0$ for all $i \in \mathcal{K}$ because by the first criterion, the values of **s** may only increase, and those variables that have moved in our out of the basis since B' have a greater value in **s** than variable x_l. By the third criterion of **s**-monotonicity, these variables have not moved since basis B', thus must have a corresponding zero value in $\mathbf{t}'^{(c)}$. As $t'_{kl} < 0$, $t''_{lr} > 0$, $t'_{kr} > 0$ and $t''_{rr} = -1$, it follows that $\mathbf{t}^{(k)'T}\mathbf{t}''_r < 0$ which contradicts the orthogonality theorem. ■

Similar arguments lead to the finiteness proofs of the dual simplex and dual MBU-simplex algorithms with **s**-monotone index selection rule.

Theorem 5.2.2 *The MBU-simplex algorithm with* **s***-monotone index selection rules are finite for linear programming problems.*

Note that minimality is a technical assumption that simplifies the proofs, but the proofs easily generalize to any infinite example as the variables that do not move have a zero contribution to the **t** vectors. In these proofs there would not be direct value explicitly writing these cases out, in contrast to the LCP case (chapter 6) where the constructs in the proof become part of the algorithm.

5.3 The criss–cross algorithm with s-monotone index selection rules

The finiteness proof of the criss–cross algorithm with **s**-monotone index selection rule is reported in [54]. The skeleton of the proof follows that in [30].

Theorem 5.3.1 *Let the standard linear programming problem be given. The criss–cross algorithm is finite when an* **s***-monotone index selection rule is applied.*

Proof. The criss–cross algorithm terminates with one of it's terminal tableaus: when the problem is optimal, primal infeasible or dual infeasible.

Let us assume to the contrary, that the criss–cross method in not finite with an **s**-monotone index selection rule. As the number of possible different bases is finite, the algorithm must cycle in the sense that it must visit the same basis multiple times. Let us consider a minimal cycling example, in which all variable moves; as the selection according to the **s** vector always only relies on the value of the variables in **s**, which are only updated for those variables that move in the bases, this assumption is not restrictive: the part of each cycling example in which the variables move an infinite time is such an example.

According to the definition of **s**-monotone rules, there exists an iteration where the index with the minimal **s** value is unique. Let this index be l. We can assume that the variable x_l is outside the basis (if not, we can consider the dual which is a symmetric case in the case of the criss–cross method). Let the basis in which x_l enters the basis be B', while when it leaves again be B''.

We call a criss–cross iteration primal, if the negative valued index is selected from the objective row, and a dual iteration if the index with a negative value is selected from the right hand side vector.

The following 4 cases are possible.

(a) primal-primal: x_l enters the basis in a primal iteration, and then also leaves in a primal iteration.

(b) primal-dual: x_l enters the basis in a primal iteration, and leaves in a dual iteration

(c) dual-primal: x_l enters the basis in a dual iteration, and leaves in a primal iteration

(d) dual-dual: x_l enters the basis in a dual iteration, and also leaves in a dual iteration

According to the definition of **s** monotone index selection rules, the **s** value for those variables that move in between B' and B'' is larger than of x_l. Let the index set of those variables that have not moved in between B' and B'' be denoted by $\mathcal{K}$ partitioned to $\mathcal{K}_B$ and $\mathcal{K}_N$ based on if $k \in \mathcal{K}$ is in the basis or not, while the index set of those variables that have moved at least once be $\mathcal{L}$, with the index l not being included in either. The corresponding cases are shown on Figure 5.3, describing the following cases:

(1.) x_l enters the basis in a primal iteration

(2.) x_l enters the basis in a dual iteration

(3.) x_l leaves the basis in a primal iteration

(4.) x_l leaves the basis in a dual iteration

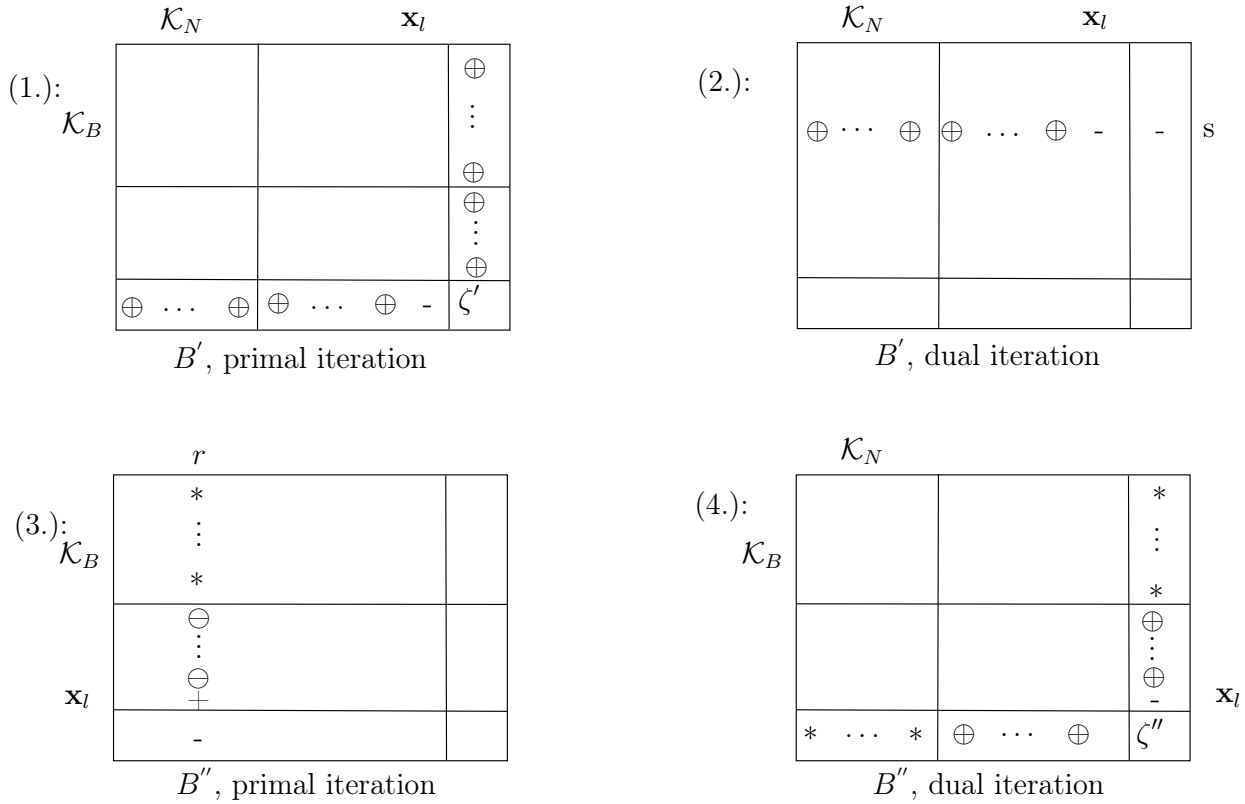


Figure 5.3: The possible tableaus during the proof.

We will show that none of the tableau combinations explained before are possible, by showing that it is possible to select a combination of $\mathbf{t}$ vectors for each case that would violate the orthogonality theorem, and therefore prove that the algorithm can not cycle.

The presentation of the relevant $\mathbf{t}$ vectors will follow the same structured decomposition according to the index sets of interest:

$\mathbf{t} =$

$\mathcal{K}_B$	$\mathcal{K}_N$	$\mathcal{L}_B \setminus \{l\}$	$\mathcal{L}_N \setminus \{l\}$	l	b	c

From table (1.), B':

$\mathbf{t}^{'(c)} =$

$0 \ \cdots \ 0$	$\oplus \ \ldots \ \oplus$	$0 \quad \ldots \quad 0$	$\oplus \quad \ldots \quad \oplus$	-	ζ'	1

$\mathbf{t}'_b =$

$\oplus \ \ldots \ \oplus$	$0 \ \cdots \ 0$	$\oplus \quad \ldots \quad \oplus$	$0 \quad \ldots \quad 0$	0	-1	ζ'

From table (2.), B':

$\mathbf{t}'^{(s)} =$

$0 \ \cdots \ 0$	$\oplus \ \cdots \ \oplus$	$0 \ \cdots \ 0 \ 1 \ 0 \ \cdots \ 0$	$\oplus \quad \cdots \quad \oplus$	-	-	0
		s				

From table (3.), B'':

$\mathbf{t}''_r =$

$* \ \cdots \ *$	$0 \ \cdots \ 0$	$\ominus \quad \cdots \quad \ominus$	$0 \ \cdots \ 0 \ \text{-}1 \ 0 \ \cdots \ 0$	+	0	-
			r			

From table (4.), B'':

$\mathbf{t}''_b =$

$* \ \cdots \ *$	$0 \ \cdots \ 0$	$\oplus \quad \cdots \quad \oplus$	$0 \quad \cdots \quad 0$	-	-1	ζ''

$\mathbf{t}''^{(c)} =$

$0 \ \cdots \ 0$	$* \ \cdots \ *$	$0 \quad \cdots \quad 0$	$\oplus \quad \cdots \quad \oplus$	0	ζ''	1

We are ready to consider the combinations of the ways x_l may enter at basis B' and leave the basis at basis B''.

Case (a), primal-primal: x_l enters the basis in a primal iteration, and then also leaves in a primal iteration. It is easy to see, that in all cases that follow, the contribution from indices from $\mathcal{K}$ is zero.

Consider vectors $\mathbf{t}'^{(c)}$ and $\mathbf{t}''_r$. As $t'_{ci} \geq 0$ and $t''_{ir} \leq 0$, for all $i \in \mathcal{I} \setminus \{n\}$, and $t''_{br} = 0$, and $t'_{cc} = 1$, $t''_{cr} < 0$, $t'_{cn} < 0$, $t''_{nr} > 0$ follows from the corresponding sign structures, we get that

$$0 = (\mathbf{t}''_r)^T \mathbf{t}'^{(c)} = \sum_{i \in \mathcal{I} \setminus \{n\}} t'_{ci}\, t''_{ir} + t'_{cn}\, t''_{nr} + t'_{cc}\, t''_{cr} + t'_{cb}\, t''_{br} \leq t'_{cn}\, t''_{nr} + t'_{cc}\, t''_{cr} < 0. \tag{5.5}$$

which contradicts the orthogonality theorem.

Case (b), primal-dual: x_l enters the basis in a primal iteration, and leaves in a dual iteration.

Consider vectors $\mathbf{t}'^{(c)}$ and $\mathbf{t}''_b$, and also $\mathbf{t}''^{(c)}$ and $\mathbf{t}'_b$.

As $t'_{ci} \geq 0$, $t''_{ci} \geq 0$, $t'_{ib} \geq 0$ and $t''_{ib} \geq 0$ for all $i \in \mathcal{I} \setminus \{n\}$, and $t'_{cc} = t''_{cc} = 1$, $t'_{bb} = t''_{bb} = -1$, and also $t'_{cb} = \zeta'$ and $t''_{cb} = \zeta''$, we get the following

$$(\mathbf{t}''_b)^T \mathbf{t}'^{(c)} = \sum_{i \in \mathcal{I} \setminus \{n\}} t'_{ci}\, t''_{ib} + t'_{cn}\, t''_{nb} + t'_{cc}\, t''_{cb} + t'_{cb}\, t''_{bb} \geq t'_{cn}\, t''_{nb} + \zeta'' - \zeta' \tag{5.6}$$

and

$$(\mathbf{t}'_b)^T \mathbf{t}''^{(c)} = \sum_{i \in \mathcal{I} \setminus \{n\}} t''_{ci}\, t'_{ib} + t''_{cn}\, t'_{nb} + t''_{cc}\, t'_{cb} + t''_{cb}\, t'_{bb} \geq \zeta' - \zeta'', \tag{5.7}$$

using that $t'_{nb} = t''_{cn} = 0$. As the two scalar products must be zero according to the orthogonality theorem, their sum should also equal to zero.

$$0 = (\mathbf{t}''_b)^T \mathbf{t}'^{(c)} + (\mathbf{t}'_b)^T \mathbf{t}''^{(c)} \geq t'_{cn}\, t''_{nb} + \zeta'' - \zeta' + \zeta' - \zeta'' = t'_{cn}\, t''_{nb} > 0. \tag{5.8}$$

as $t''_{nb} < 0$, $t'_{cn} < 0$, contradicting the orthogonality theorem.

Case (c), dual-primal: x_l enters the basis in a dual iteration, and leaves in a primal iteration

Consider vectors $\mathbf{t}'^{(s)}$ and $\mathbf{t}''_r$. Using $t'_{si} \geq 0$ and $t''_{ir} \leq 0$ for all $i \in \mathcal{I} \setminus \{n\}$, and that $t'_{sc} = 0$ and $t''_{br} = 0$, and also $t'_{sn} < 0$ and $t''_{nr} > 0$, we get that

$$0 = (\mathbf{t}''_r)^T \mathbf{t}'^{(s)} = \sum_{i \in \mathcal{I} \setminus \{n\}} t'_{si}\, t''_{ir} + t'_{sn}\, t''_{nr} + t'_{sc}\, t''_{cr} + t'_{sb}\, t''_{br} \leq t'_{sn}\, t''_{nr} < 0. \tag{5.9}$$

contradicting the orthogonality theorem.

Case (d), dual-dual: x_l enters the basis in a dual iteration, and also leaves in a dual iteration

Consider vectors $\mathbf{t}'^{(s)}$ and $\mathbf{t}''_b$. As $t'_{si} \geq 0$ and $t''_{ib} \geq 0$ for all $i \in \mathcal{I} \setminus \{n\}$ and that $t'_{sc} = 0$, and also that $t'_{sn} < 0$, $t''_{nb} < 0$, $t'_{sb} < 0$ and $t''_{bb} = -1$ we get that

$$0 = (\mathbf{t}''_b)^T \mathbf{t}'^{(s)} = \sum_{i \in \mathcal{I} \setminus \{n\}} t'_{si}\, t''_{ib} + t'_{sn}\, t''_{nb} + t'_{sc}\, t''_{cb} + t'_{sb}\, t''_{bb} \geq t'_{sn}\, t''_{nb} + t'_{sb}\, t''_{bb} > 0. \tag{5.10}$$

contradicting the orthogonality theorem.

As we have shown that all cases lead to a contradiction, we have proved that the criss–cross algorithm is finite with **s** monotone index selection rules. ■

Theorem 5.3.2 *The criss–cross algorithm with* **s***-monotone index selection rules are finite for linear programming problems.*

5.4 Numerical experiments

For sake of completeness, we present both the results obtained from the Matlab implementation, as well as the C implementation, which dominates the result for the linear programming test cases, as explained in Chapter 4.

We chose our test problems from the NETLIB data set [2]. This is a public, well-known test set. Although these are considered to be easy problems for state of the art solvers nowadays, they are numerically non-trivial. This LP problem collection contains 98 problems. The size and complexity of these problems are diverse.

5.5 Numerical tests on selected problems with Matlab

In this section, we summarize our numerical results of our simplex and MBU-simplex implementations. We implemented the two algorithms using three s-monotone index selection rules, the minimal-index, the Last-In-First-Out (LIFO) and the Most-Often-Selected-Variable (MOSV) index selection rules. In the following charts, SA denotes the simplex algorithm, and MBU-SA the MBU-simplex algorithm.

Our purpose is to demonstrate some properties and efficiency of the selected simplex methods and index selection rules. Our implementation works on the standard form of the problems; we implemented the simple, textbook versions of the algorithms without any significant numerical tricks and considerations, and without applying a presolver. The purpose was not to obscure the behaviour of the base algorithm and index selection rules, as explained in Chapter 4.

Tableau 1 presents the selected small problems.

Problems	Rows	Columns	Non zeros	Density	Opt. value
ADLITTLE	57	97	465	8,410%	2.255E+05
AFIRO	28	32	88	9,821%	-4.648E+02
AGG	489	163	2541	3,188%	-3.599E+07
AGG2	517	302	4515	2,892%	-2.024E+07
BLEND	75	83	521	8,369%	-3.081E+01
KB2	44	41	291	16,131%	-1.750E+03
RECIPE	92	180	752	4,541%	-2.666E+02
SC105	106	103	281	2,574%	-5.220E+01
SC205	206	203	552	1,320%	-5.220E+01
SC50A	51	48	131	5,351%	-6.458E+01
SC50B	51	48	119	4,861%	-7.000E+01
SCAGR25	472	500	2029	0,860%	-1.475E+07
SCAGR7	130	140	553	3,038%	-2.331E+06
SCFXM1	337	457	2612	1,696%	1.842E+04
STOCFOR1	118	111	474	3,619%	-4.113E+04

Table 5.1: The selected problems.

Table 5.2 shows the iteration counts; the minimum is highlighted in blue, the maximums are in red.

Problems	SA			MBU-SA		
	MIN	LIFO	MOSV	MIN	LIFO	MOSV
AFIRO	**29**	**81**	76	**39**	**71**	64
KB2	**139**	232	**242**	**121**	**209**	167
SC50A	**57**	**73**	70	**76**	74	**72**
SC50B	**64**	**58**	60	**58**	**59**	**59**
ADLITTLE	**186**	**404**	384	**244**	347	**376**
BLEND	**175**	466	**531**	**216**	223	**227**
RECIPE	**233**	385	**546**	**242**	270	**274**
SC105	**133**	286	**387**	**179**	**280**	267
STOCFOR1	**226**	334	**348**	**279**	307	**470**
SCAGR7	**314**	544	**602**	**270**	339	**378**
SC205	**283**	958	**1622**	**381**	661	**741**
SCFXM1	**949**	2074	**2685**	**1389**	**1372**	1375
SCAGR25	**2614**	**3236**	2928	1464	**1684**	**627**
AGG	**656**	1275	**1295**	678	**858**	**161**
AGG2	**1113**	1138	**1238**	**795**	**1780**	1590

Table 5.2: Iteration counts.

For both algorithms, the smallest iteration counts were produced by the minimal index index selection rule. The worst iteration counts occurred with the MOSV index selection rule, in the case of simplex method. For the MBU-simplex, the worst iteration counts were reported for about 50% with the MOSV and about 50% with the LIFO index selection rule.

Observe, the MBU-simplex tended to need fewer iterations than the simplex method. There are some problems, where it made less than half of the simplex iteration counts (e.g. *BLEND, RECIPE* and *SCAGR25*). An iteration graph is presented on Figure 5.4.

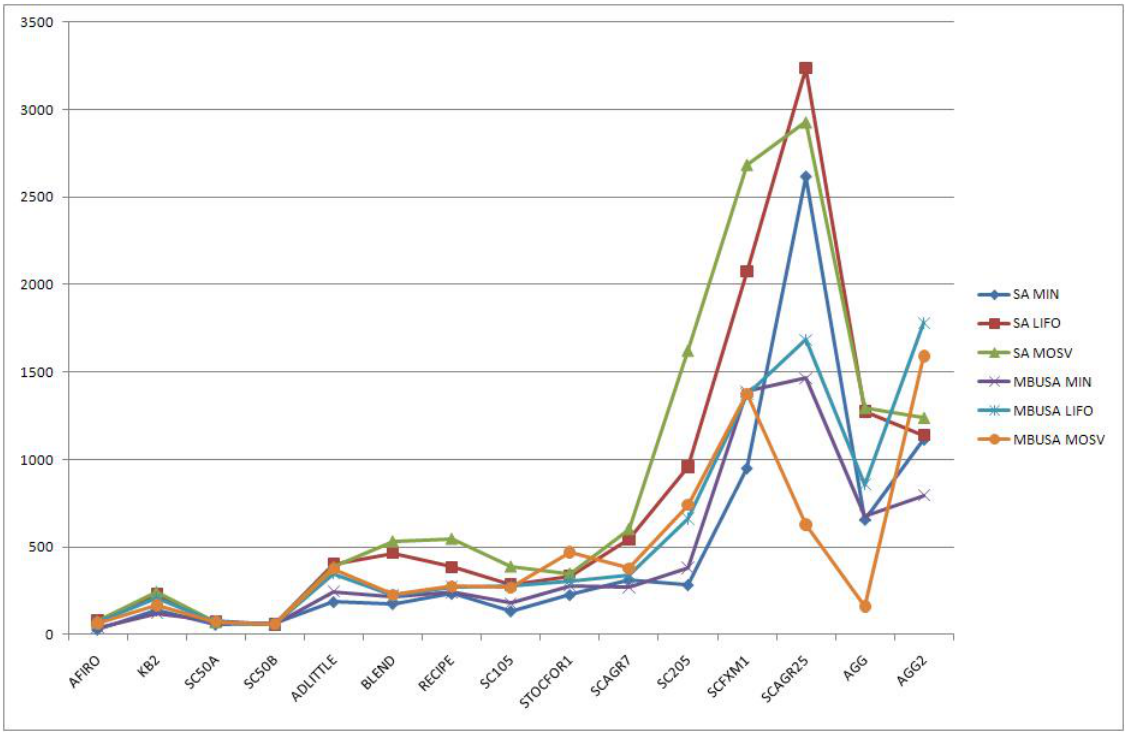


Figure 5.4: Iteration counts.

The next tables summarize the average iteration counts of the 15 problems, where the minimum average iteration counts are highlighted in red.

Problem	SA			MBU-SA		
	MIN	LIFO	MOSV	MIN	LIFO	MOSV
Sum Its	7171	11544	13014	6431	8534	6848
Average	478,067	769,6	867,6	**428,734**	568,934	456,534

Table 5.3: The sum and average iteration counts all the 15 problems.

One can see that the most efficient was the MBU-simplex algorithm with the minimal-index index selection rule. The second fewest iterations on average was also by the MBU-simplex algorithm, with the Most Often Selected Variable index selection rule.

In the next tableau, we summarize the sum of iteration counts and the average iteration counts for the first 10 problems. These problems are easier and smaller than the last 5 problems, making such a separation interesting.

Problem	SA			MBU-SA		
	MIN	LIFO	MOSV	MIN	LIFO	MOSV
Sum Its	1556	2863	3246	1724	2179	2354
Average	**155,6**	286,3	324,6	172,4	217,9	235,4

Table 5.4: The sum and average iteration counts for the first 10 problems.

Looking at the first 10 problems only, there is a change in the ordering. The fastest is now the simplex method with the minimal-index index selection rule, followed by the MBU-simplex algorithm with the minimal-index index selection rule, and then finally the MBU-simplex algorithm with the Last-in-First-Out-index selection rule.

The next tableau summarizes the iteration counts for the last 5 larger problems.

Problem	SA			MBU-SA		
	MIN	LIFO	MOSV	MIN	LIFO	MOSV
Sum Its	5615	8681	9768	4707	6355	4494
Average	1123	1736,2	1953,6	941,4	1271	**898,8**

Table 5.5: The sum and average iteration counts for the last 5 problems.

Looking at the large problems only, the quickest is the MBU-simplex algorithm with the Most Often Selected Variable index selection rule, followed by the MBU-simplex algo-

rithm with the minimal-index index selection rule, and finally the simplex method with the minimal-index index selection rule.

This would suggest, that as the size of the problems get bigger, the MBU-simplex algorithm is more efficient than the simplex method, and the secondary index selection rules are also more useful than the minimal-index index selection rule. Although such a small test set is naturally too small to draw any strong conclusions, it does demonstrate that the MBU method can be more efficient than the simplex method, making it a worthy addition as an alternative solution method.

If we are considering the results on a problem by problem basis, we get the best results with the simplex algorithm and the minimal index rule, while the Most-Often-Selected-Variable, and the Last-In-First-Out perform notably. It would be worth making use of the extra flexibility of these methods in the implementations.

This set of problems is too small to draw any strong conclusions, and so the next section present results achieved on a much larger problem set.

5.6 Numerical results using external linear algebra functions

This section presents the numerical results obtained by the implementation using an the linear algebra of Xpress.

We use the primal version of the simplex algorithm, as it is more comparable to the MBU method than the dual. The MBU-simplex method is not a dual method in the sense that it does not maintain dual feasibility, but instead it is complete when dual feasibility is achieved, just like in the case of the primal simplex. However, it is not strictly a primal method either, as primal feasibility is not maintained in every iteration, but rather it is restored every time the feasibility of a new dual variable is achieved. Still, these properties make the MBU arguably a primal like method, supporting the choice of selecting the primal simplex method for comparison.

In comparison, the monotonic build-up simplex method is relatively complex, as the design to maintain monotonicity in the feasibility of the dual variables (reduced cost) requires an extra dual ratio test in each iteration. The monotonic build-up simplex algorithm selects an infeasible dual variable (called the driving column), and works to achieve its feasibility while keeping all the already dual feasible variables dual feasible.

The basic form of the criss–cross method as presented in Section 3.1.4 is inferior in practice to both the simplex and the monotonic build-up simplex method. This is due to the combinatorial nature of it's convergence properties, and the lack of a monotone merit function.

There is a total of 108 problems in the selected test set (see Chapter 4 for the selection criteria). However, it is interesting to see how large the selected test set would be if only either the simplex or the MBU algorithm were used in the selection process.

	All versions solved	**Number of extra problems**
Simplex	131	131-108=23 (+21%)
MBU	118	118-108=10 (+9.2%)
Either MBU or Simplex	141	141-108=33 (+30.5%)

Table 5.6: Number of problems solved across all index selection rules using either the simplex or the MBU algorithm.

The fact that our simple implementation of the MBU managed to solve fewer problems is not surprising, as it is a more complex algorithm than the traditional primal simplex and as such with more opportunities for numerical issues. It is notable how large the number of problems discarded from the selection due to only one of the algorithms is, and clearly indicates the very different solution path the algorithms take. It is also notable that even though the traditional simplex method is more stable, it is not dominating the MBU variant.

The complete solution statistics is presented in Table 5.7:

	Most negative	**Minimal**	**LIFO**	**MOSV**	**Hybrid LIFO**	**Hybrid MOSV**
Simplex	209	150	**168**	150	161	**175**
MBU	150	145	149	149	146	146

Table 5.7: Total number of problems solved by algorithm and index selection rule combinations on all candidate problems.

The highest success rate is achieved by the simplex method with the Dantzig index selection rule. The result using this method is provided as a reference only, as it is not a

theoretically finite method. As this test primarily measures numerical stability, the most robust method proved to be the relatively simpler one, doing the smaller number of iterations. Although this result is expected, its lead is larger than anticipated. It is also interesting that the Dantzig rule is less dominant in the case of the MBU; intuitively this is because the dual ratio test overwrites the original column selection. The question naturally arises: what selection rule (possibly greedy without the need of theoretical finiteness) would yield the best fit with the MBU method?

Definition 5.6.1 *The multiplicity of a simplex type method on a given test problem set and index selection rule is defined as the total sum of the number of all alternative pivot positions in which the index selection rule had left the choice of the pivot position open (i.e. offered flexibility).*

A selection of detailed results are presented in Table 5.8 for the NETLIB, and in Table 5.9 for the Miplib sets. In these tables, for each model and algorithm combinations, 3 numbers are presented: iteration, multiplicity and run time.

	MBU Dantzig	MBU Minimal	MBU LIFO	MBU MOSV	MBU H-LIFO	MBU H-MOSV	Simplex Dantzig	Simplex Minimal	Simplex LIFO	Simplex MOSV	Simplex H-LIFO	Simplex H-MOSV
	247	298	397	454	236	257	156	419	393	457	234	163
ADLITTLE	1392	0	1830	1623	1002	1365	8734	0	6111	6308	5036	6300
	0s	0s	0s	0s	0s	0s	0s	0s	0s	0s	0s	0s
	3135	2824	2821	2821	2801	3626	3589	10407	7100	8022	9028	5507
FIT1P	163797	0	885953	885953	171835	169080	2862086	0	1653008	1659394	1381221	1928092
	8s	7s	7s	7s	7s	9s	7s	21s	14s	16s	18s	11s
	17261	20857	12656	17183	21183	18031	4287	11434	14230	12320	22595	6300
MAROS-R7	3355631	0	5697238	7431172	2560415	3514255	14601018	0	20624812	20893306	13668626	16756305
	251s	312s	175s	238s	308s	257s	43s	111s	137s	120s	246s	65s
	1044	1261	985	981	1061	1030	630	5306	2760	4494	1986	1064
SCFXM1	35470	0	40217	42890	25459	29254	132223	0	141913	144835	80158	109563
	1s	1s	1s	1s	1s	1s	0s	3s	1s	3s	2s	0s
	651	663	660	645	653	565	523	714	655	696	591	489
SCORPION	31098	0	32285	31495	27225	29122	84824	0	60634	61300	63462	68268
	0s	1s	0s	0s	0s	1s	0s	0s	1s	1s	0s	0s
	767	1718	286	1953	1008	917	338	185742	4480	42395	671	460
SCSD1	2057	0	43	1315	1031	2100	55034	0	79952	82081	44795	52397
	0s	1s	0s	2s	1s	1s	0s	94s	3s	22s	0s	0s
	2904	1776	1902	1852	3042	2808	2283	2239	2299	2333	2130	2462
SEBA	385775	0	429015	427110	271092	323841	1141102	0	751784	755741	639317	948853
	6s	4s	4s	4s	6s	6s	4s	4s	4s	4s	3s	4s
	2380	2300	2172	2423	2303	2160	1805	2414	2388	2257	2326	2025
SHELL	119134	0	138724	159986	57587	87425	991889	0	855983	883871	598344	813375
	6s	6s	5s	7s	6s	6s	4s	4s	4s	4s	4s	4s
	6751	7478	6743	6476	6879	6613	5667	7216	7448	6887	6796	5626
SIERRA	4417685	0	5226096	5234851	4350699	4482015	9503307	0	8289979	8326468	7989135	8218504
	43s	48s	43s	42s	44s	42s	27s	34s	35s	33s	32s	26s
	573	486	499	518	589	561	494	489	570	526	616	552
VTP-BASE	23435	0	33600	33747	20888	23157	89166	0	68299	69958	72766	73293
	0s	0s	0s	0s	1s	0s	0s	1s	0s	1s	1s	0s

Table 5.8: Numerical results on a set of NETLIB problems.

	MBU Dantzig	MBU Minimal	MBU LIFO	MBU MOSV	MBU H-LIFO	MBU H-MOSV	Simplex Dantzig	Simplex Minimal	Simplex LIFO	Simplex MOSV	Simplex H-LIFO	Simplex H-MOSV
	416	505	538	502	474	442	266	1078	827	835	380	321
bell3a	10070	0	20636	20546	8940	10140	36278	0	47999	49317	34298	37170
	0s	1s	0s	0s	1s	0s	0s	1s	1s	0s	0s	0s
	11984	8515	8211	8680	15945	11915	11276	49770	38905	68400	66300	6374
eilB101	4138209	0	4452538	4458614	4020427	4058276	22943568	0	4809496	4838383	11656561	12109399
	85s	61s	59s	64s	113s	85s	60s	260s	204s	360s	360s	34s
	2962	2771	2857	2916	3208	2916	4730	4155	4478	4507	5101	9989
neos-1616732	86609	0	87594	87368	90364	87368	1695071	0	894758	894390	799322	1034908
	11s	10s	10s	11s	12s	11s	13s	12s	12s	13s	15s	27s
	17198	18872	28100	26097	16156	16733	16221	27614	28167	24722	17103	14467
neos-555424	13083516	0	10522883	10444208	12902065	12917539	58089052	0	33859418	33747224	32763158	39078410
	221s	245s	360s	339s	215s	217s	161s	271s	279s	246s	172s	144s
	45	48	48	48	47	45	49	50	51	50	50	49
flugpl	418	0	441	441	376	382	861	0	769	770	632	730
	0s	0s	0s	0s	0s	0s	0s	0s	0s	0s	0s	0s
	4977	8271	6845	7121	5361	5321	4531	8281	7488	8725	5610	4158
gesa2	1598041	0	1795470	1944212	1409791	1483927	6582595	0	5705763	5651121	3339443	4002298
	21s	37s	29s	31s	23s	24s	15s	26s	24s	28s	19s	13s
	13083	11751	11884	11983	12125	13084	5675	44897	115709	208458	72643	29707
go19	400003	0	381439	380045	407048	400386	3402593	0	973954	973515	714247	951911
	19s	18s	18s	18s	18s	20s	8s	59s	153s	275s	102s	40s
	3197	7370	5379	5256	2686	2448	3877	10241	7664	9452	4277	4385
mik	41538	0	63430	65280	29556	33851	1137428	0	193253	194068	183961	231635
	3s	8s	5s	5s	2s	2s	3s	7s	6s	7s	3s	3s
	998	864	1004	1113	902	762	866	2106	1516	1343	1270	1211
p0201	41505	0	43162	45395	36531	38800	156780	0	104780	105236	102030	110295
	0s	1s	0s	1s	1s	0s	1s	2s	1s	0s	1s	0s
	583	505	514	520	549	569	690	635	553	572	772	644
rgn	16538	0	19613	19849	15601	16357	96388	0	54088	54428	51905	63232
	1s	0s	1s	1s	0s	1s	1s	0s	0s	0s	1s	0s

Table 5.9: Numerical results on a set of Miplib problems.

5.7 Iteration and time

In this section, all results refer to the selected 108 test problems.

Table 5.10 presents the fastest solution times among the simplex and the MBU-simplex algorithms, all timings rounded up to seconds (number of times a given algorithm was fastest with ties included in all).

	Most negative	Minimal	LIFO	MOSV	Hybrid LIFO	Hybrid MOSV
Simplex	84	36	32	40	48	**69**
MBU	31	25	31	28	30	31

Table 5.10: Fastest solution time achieved in the selected test set. Rounded to seconds, so the numbers include some multiplicity.

Table 5.11 presents the total sums of iteration counts.

Iteration	Most negative	Minimal index	LIFO	MOSV	Hybrid LIFO	Hybrid MOSV
Simplex	264 672	988 803	757 289	1 015 929	625 707	**341 381**
MBU	588 423	442 149	503 547	580 842	437 821	**428 639**

Table 5.11: The total sums of iteration counts.

As expected, for the primal simplex, the most negative variable rule is the most efficient. From among the theoretically finite index selection rules, the Hybrid-MOSV proved to be best. Although the MBU makes significantly less iterations, it does use more information in all iterations: the MBU spends more time per iteration, as it needs to calculate the transformed row as well (which is typically computationally significantly more expensive than the transformed column). However, even though each iteration is more expensive, it seems to pay off in terms of the average total time by doing fewer iterations. Table 5.12 presents the total solution times.

Time	Most negative	Minimal index	LIFO	MOSV	Hybrid LIFO	Hybrid MOSV
Simplex	1 017	2 626	2 286	2 744	2 275	**1 100**
MBU	2 269	2 086	2 312	2 498	2 083	**2 036**

Table 5.12: Total solution times.

The MBU seems to be the fastest in average time as well, so the extra investment per iteration pays off, although possibly not surprisingly, the faster strategy proved to be the most negative rule.

Our results indicate that:

- The most-negative rule with simplex is the fastest combination: expected.
- Note for the MBU: spends many iterations in the dual ratio loops.
- Hybrid-MOSV is the most efficient among the theoretically finite index selection rules.
- The MBU takes less iteration on average but the more time consuming iterations often pay off.

5.8 Iteration and multiplicity of choice

The question arises, what was the level of freedom in choosing the incoming variable in the case of the flexible index selection rules and if it exhibits the expected correlation with efficiency. The sum of this freedom will be called multiplicity in the next tableaus.

In Table 5.13 and 5.14, 'I' stands for iteration, 'MP' the multiplicity, 'S' and 'M' the simplex and MBU-simplex algorithms respectively. The multiplicity is the sum of the possible index selection choices added together for all iterations and for all problems.

I	Most negative	Minimal index	LIFO	MOSV	Hybrid LIFO	Hybrid MOSV	SUM
S	265k	989k	757k	1 016k	626k	**341k**	2 981k
M	588k	442k	504k	581k	438k	**429k**	3 994k

Table 5.13: Iteration counts in thousand iterations.

MP	Most negative	Minimal index	LIFO	MOSV	Hybrid LIFO	Hybrid MOSV	SUM
S	369M		239M	240M	223M	**251M**	1 322M
M	98M		98M	98M	89M	92M	475M

Table 5.14: Multiplicities expressed in millions of choices, extended version of table 5.10.

The expected correlation between speed and level of multiplicity is apparent, supporting the benefits of flexible rules. The MBU appears to reduce flexibility faster, due to the larger number of **s** updates through the dual ratio test, which makes the flexible index selection rules more rigid much quicker than for the simplex; especially in the case of the LIFO rule.

Figure 5.5 plots the connection between multiplicity and total solution time (as presented in Table 5.15). The horizontal axis shows the time, while the vertical one the multiplicity.

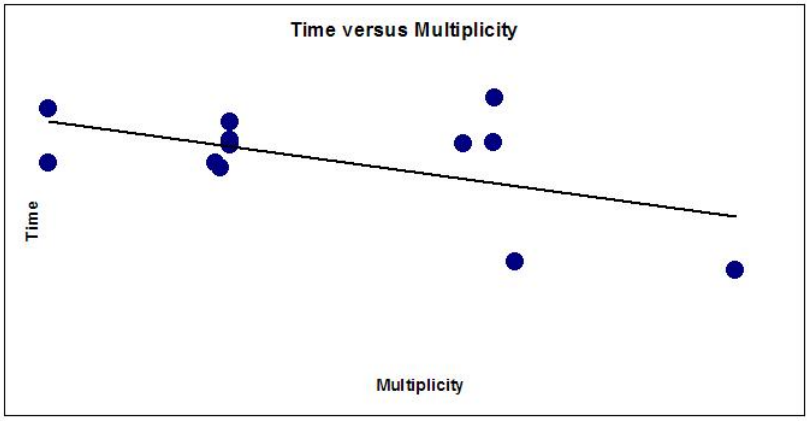


Figure 5.5: Connection between multiplicity and total solution time.

Time	1 017	1 100	2 036	2 083	...
Multiplicity	369	250	92	89	...

Table 5.15: Ordered set of total time and multiplicities.

Turning the previous observation around, we conclude that the longer it took to solve the problems, the more rigid index selection rule was applied.

5.9 Numerical results on selection of industrial problems

Detailed example run statistics are provided on a set of assembly line balancing and workforce skill balancing problems, see [28] and [46] respectively. As with the Miplib sets, only the linear part of the models was considered. These problems provide an insight to the solution of simpler, but realistic problems. This set contains 11 problems. All algorithm and index selection rule combinations solved all problems, except for SALBP-1-ESC-3LEV problem on which all combinations of the MBU method failed due to numerical issues, incorrectly declaring the problem infeasible.

On these problems, the increase in sizes due to the transformation to the standard form were 1015%, 130% and 59% in the number of rows, columns and elements of the matrix, respectively.

Table 5.16 collects the relevant run statistics. For each model and algorithm combinations, 3 numbers are presented: iteration, multiplicity and run time; with the exception of the first 9 problems, all running times are below 1 second and were omitted from the table.

5.10 Summary

We have demonstrated that the flexible index selection rules could be a viable alternative in practice when cycling occurs possibly from the (removal of) auto-perturbation itself. Another interesting field of application would be to use these flexible rules in exact arithmetic implementations. The greedy approach in column selection is fastest and the flexible index selection rules can successfully exploit such strategies and provide a significant performance improvement over the more rigid rules - while maintaining theoretical finiteness as well.

We have also demonstrated that the MBU algorithm can be a practically competitive alternative to the traditional (primal) simplex method, as it is demonstrated by Table 5.6 and Table 5.7; their theoretically finite versions are comparable both in terms of iteration counts and in terms of solution times, as summarized in Table 5.11 and Table 5.12.

Some algorithmic concepts like the direct handling of both lower and upper bounded variables, the handling of range and equality constraints or free variables could undoubtedly be implemented in both algorithms in such a way that their presence does not deteriorate the running times in favour of either methods. While we would expect that such extensions would not change the analysis of this thesis, it would make a larger set of problems addressable by the implementations.

It would be interesting to test the flexible index selection rules on special problem classes like network problems, that are highly degenerate and numerically stable.

	MBU Dantzig	MBU Minimal	MBU LIFO	MBU MOSV	MBU H-LIFO	MBU H-MOSV	Simplex Dantzig	Simplex Minimal	Simplex LIFO	Simplex MOSV	Simplex H-LIFO	Simplex H-MOSV
SALBM-1	321	392	332	315	316	380	104	369	389	423	187	127
	2246	0	2205	2011	1747	1760	5175	0	5654	5637	4550	4624
SALBM-1	319	406	352	353	369	388	175	575	492	479	323	247
-ESC	4112	0	3996	3918	3938	3959	11364	0	8996	8920	8185	8609
SALBM-1	465	490	478	463	372	367	160	635	724	674	240	305
-HSC	2977	0	3034	3319	2645	2807	9822	0	8784	9079	7586	8113
SALBM-1	536	578	521	541	554	684	187	907	969	1080	448	519
-LSC	3566	0	3718	3803	3053	2861	12089	0	8970	9305	8830	10175
SALBM-2	126	138	143	143	116	116	62	167	166	156	120	110
	546	0	868	836	508	549	1622	0	1614	1629	1336	1390
SALBM-2	198	184	173	181	197	195	84	283	226	278	135	149
-ESC	1134	0	1208	1132	978	1078	3195	0	2999	3015	2422	2459
SALBM-2	154	205	222	219	150	145	96	277	320	322	180	152
-HSC	867	0	1084	1055	842	895	3179	0	2835	2863	2218	2523
SALBM-2	250	245	226	214	228	201	74	368	480	425	132	125
-LSC	1034	0	1136	1145	890	959	2248	0	3125	3241	2285	2108
SALBP-1	11535	22303	7515	12438	15086	13630	1214	11530	7750	10076	7646	14577
	475151	0	373007	365498	263814	115885	940908	0	1147003	1159227	976951	969663
	25s	54s	17s	28s	32s	29s	2s	18s	13s	16s	12s	25s
SALBP-1	N/A						4374	100108	35066	161847	72664	68718
-ESC-3LEV							4375057	0	1847066	1849618	1830788	2012826
							10s	211s	74s	351s	158s	171s
SALBP-2	3595	4382	3876	4048	4448	4245	535	15405	5667	14312	4924	4785
	52566	0	51302	51592	35683	36562	170252	0	146070	150643	132050	169991
	3s	4s	3s	3s	3s	3s	0s	9s	3s	9s	3s	3

Table 5.16: Numerical results on a set of industrial problems.

It could also be argued, that as the monotone simplex method applies a dual ratio test, it is not a primal algorithm, and it could be reasonable to include a dual simplex algorithm in the comparisons.

The numerical experiments of this chapter have been published in [20].

Chapter 6

Flexible index selection rule for criss–cross algorithms of linear complementarity problems

This chapter analyses the numerical behaviour of the criss–cross algorithm for the linear complementarity problem. For the sake of completeness, we summarize the theory of **s**-monotone index selection rules for the well-known criss–cross method and the relevant Existentially Polynomial (EP)-type extensions [20]. This extension to the criss–cross method, in contrast to most LCP solution methods that require a priori information about the properties of the input matrix can attempt to solve any LCP problems.

One of the most general matrix properties often required for finiteness of the pivot algorithms (or polynomial complexity of interior point algorithms) is sufficiency. However, there is no known polynomial time method for checking the sufficiency of a matrix (classification of column sufficiency of a matrix is co-NP-complete).

A simple extension of the criss–cross algorithm is presented for LCPs using existentially Polynomial (EP)-type theorems with general matrices, following [17]. Computational results obtained using the extended version of the criss–cross algorithm for bi-matrix games and for the Arrow-Debreu market equilibrium problem with different market size is presented.

The extension to EP theorems of the criss–cross method is first presented in [26]. The application of the **s**-monotone rule to this the EP extended version is first presented in [17]. The numerical experiments presented in [20] are the author's addition.

6.1 Sufficient matrices and the criss–cross method

The algorithms presented in this chapter will use the concept of *strictly sign reversing* and *strictly sign preserving* vectors:

Definition 6.1.1 *[26] A vector* $\mathbf{x} \in \mathbb{R}^{2n}$ *is called* strictly sign reversing *if*

$$\begin{aligned} &x_i x_{\bar{i}} \leq 0 \quad \textit{holds for all indices } i = 1, \ldots, n \textit{ and} \\ &x_i x_{\bar{i}} < 0 \quad \textit{holds for at least one index } i \in \{1, \ldots, n\}. \end{aligned} \tag{6.1}$$

A vector $\mathbf{x} \in \mathbb{R}^{2n}$ *is called* strictly sign preserving *if*

$$\begin{aligned} &x_i x_{\bar{i}} \geq 0 \quad \textit{holds for all indices } i = 1, \ldots, n \textit{ and} \\ &x_i x_{\bar{i}} > 0 \quad \textit{holds for at least one index } i \in \{1, \ldots, n\}. \end{aligned} \tag{6.2}$$

To simplify the definition of the next lemma, let us introduce the subspaces

$$V := \left\{(\mathbf{u}, \mathbf{v}) \in \mathbb{R}^{2n} \mid -M\mathbf{u} + \mathbf{v} = \mathbf{0}\right\} \tag{6.3}$$

and

$$V^{\perp} := \left\{(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{2n} \mid \mathbf{x} + M^T\mathbf{y} = \mathbf{0}\right\}, \tag{6.4}$$

where $\mathbf{u}, \mathbf{v}, \mathbf{x}$ and $\mathbf{y}$ are all vectors of length n. V and $V^{\perp}$ are orthogonal complementary subspaces [26] of $\mathbb{R}^{2n}$.

Lemma 6.1.1 *[26] A matrix* $M \in \mathbb{R}^{n \times n}$ *is sufficient if and only if there exists no strictly sign reversing vector* V *and no strictly sign preserving vector in* $V^{\perp}$.

The following lemma summarizing the sign structure of sufficient matrices is central to the working of the criss–cross algorithm for linear complementarity problems.

Lemma 6.1.2 *[13] Let* M *be a sufficient matrix,* B *a complementary basis and* $\bar{M} = B^{-1}M$, $T = [t_{ij} \mid i \in J_B, j \in J_N]$ *the corresponding short pivot tableau. Then*

(a) $t_{ii} := \bar{m}_{i\bar{i}} =\geq 0$ *for all* $i \in J_B$, *i.e. the diagonals of the short pivot tableau are non-negative,*

(b) for all $i \in J_B$, *if* $t_{ii} := \bar{m}_{i\bar{i}} = 0$ *then* $\bar{m}_{i\bar{j}} = \bar{m}_{j\bar{i}} = 0$ *or* $\bar{m}_{i\bar{j}} \cdot \bar{m}_{j\bar{i}} < 0$ *for all* $j \in J_B$, $j \neq i$ *(*$t_{ij} = t_{ji} = 0$ *or* $t_{ij}t_{ji} < 0$ *in the notation of the short pivot tableau), i.e. the transpose positions of the short pivot tableau are of reveres sign.*

The proof of Lemma 6.1.2 is constructive, and provides a certificate constructed from the pivot tableau that the original matrix M is not sufficient if the structure defined by the lemma is violated [26, 18].

Given a perturbation matrix P, the *permutation* of matrix $M \in \mathbb{R}^{n\times n}$ is defined as $P^T MP$.

Lemma 6.1.3 *[23] For an $M \in \mathbb{R}^{n\times n}$ row (column) sufficient matrix, an arbitrary P perturbation matrix and $D \in \mathbb{R}^{n\times n}_{+}$ is a positive diagonal matrix, the following holds:*

1. *the perturbed matrix $P^T MP$ of matrix M is also row (column) sufficient,*
2. *the product DMD (i.e. scaled M matrix) is also row (column) sufficient,*
3. *every principal sub-matrix of M is also row (column) sufficient.*

A sufficient matrix $\bar{M}$ is also sufficient after any number of arbitrary principal pivots. The class of sufficient matrices is closed under principal block pivot operations [66], and as a consequence their properties are preserved during the criss–cross type algorithms, as the exchange pivot operations carried out by the criss–cross algorithms are equivalent to single 2×2 block pivot.

The matrix of Example 6.1.1 will be used in all examples of the chapter.

Example 6.1.1 *As an example of a non-sufficient matrix, consider*

$$M = \begin{pmatrix} 1 & -1 & -1 & 0 & -1 \\ -1 & 2 & 0 & 0 & -1 \\ 1 & -2 & -1 & -2 & 0 \\ -4 & -1 & -1 & 2 & 4 \\ -1 & 0 & 2 & 0 & 1 \end{pmatrix}$$

This matrix is neither column, nor row sufficient. In this example, the operator $\cdot$ denotes the Hadamard, element wise product. Consider the column vector

$$\mathbf{x}^T = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

then

$$\mathbf{x} \cdot (M\mathbf{x}) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \cdot \left[\begin{pmatrix} 1 & -1 & -1 & 0 & -1 \\ -1 & 2 & 0 & 0 & -1 \\ 1 & -2 & -1 & -2 & 0 \\ -4 & -1 & -1 & 2 & 4 \\ -1 & 0 & 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \right] =$$

$$= \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} -1 \\ 0 \\ -1 \\ -1 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \end{pmatrix}$$

a strictly sign reversing vector. Similarly, for row vector

$$\mathbf{y} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

we have

$$(\mathbf{y}M) \cdot \mathbf{y} = \begin{pmatrix} -3 & -3 & -2 & 0 & 4 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & -2 & 0 & 0 \end{pmatrix}$$

making $\mathbf{y}$ *a proof that* M *is not sufficient according to Definition 2.2.1 on Page 19.*

As a consequence, it is well-characterized when the primal linear complementarity problem has no feasible complementary solution in the case when the matrix M is sufficient and rational: a solution to the corresponding dual linear complementarity problem is a polynomial sized certificate to the infeasibility of the primal problem.

When using **s**-monotone index selection rules, the role of the **s**-vector is to maintain a history about the movements of the variables. Recall from Chapter 5 the symmetric version of the update; in the LCP case when solving with the criss–cross, these updates are generalized as follows: when an exchange pivot is made, the values of the **s** vector are updated first for the indices of the passively selected variable pair first, and only then followed by the update for the indices of the actively selected variable pair, thus maintaining the symmetry of vector **s** (i.e. the value of any variable pair (u_i, v_i) in **s** is the same after each pivot). This way, the **s** vector's update in the case of an exchange pivot is the same as if two pivots subsequential diagonal pivots were made.

Note, that Lemma 6.1.3 guarantees that the pivot tableau's matrix remains sufficient, while it is clear that both the diagonal and the exchange pivot operations preserve comple-

mentarity.

The pseudo-code and flow chart of the criss–cross type algorithm using **s**-monotone index selection rules is presented below and its flow chart in Figure 6.1.

Criss–cross type algorithm with symmetric s-monotone pivot rule

---

input data:
problem in (2.16), where M is sufficient, $T := -M$, $\bar{q} := q$, $r := 1$ and initialize $\mathbf{s}$;
begin
determine $\mathcal{J} := \{i \in \mathcal{I} : \bar{q}_i < 0\}$;
while $(\mathcal{J} \neq \emptyset)$ **do**b
$\mathcal{J}_{\max} := \{j \in \mathcal{J} \mid s(j) \geq s(\alpha) \text{ for all } \alpha \in \mathcal{J}\}$, let $k \in \mathcal{J}_{\max}$ arbitrary;
if $(\bar{t}_{kk} < 0)$ **then**
diagonal pivot on $\bar{t}_{kk}$, update vector $\mathbf{s}$ for variable pair (u_k, v_k);
let $r := r + 1$;
else /* $t_{kk} = 0$ */
$\mathcal{K} := \{i \in \mathcal{I} : \bar{t}_{ki} < 0\}$;
if $(\mathcal{K} = \emptyset)$ **then**
STOP: D-LCP solution (The LCP problem has no feasible solution);
else
$K_{\max} = \{\beta \in \mathcal{K} \mid s(\beta) \geq s(\alpha), \text{for all } \alpha \in \mathcal{K}\}$;
let $l \in \mathcal{K}_{\max}$ arbitrary;
exchange pivot on $\bar{t}_{kl}$ and $\bar{t}_{lk}$;
update vector $\mathbf{s}$ for variable pair (u_l, v_l) as in iteration $r + 1$;
then for variable pair (u_k, v_k) as in iteration $r + 2$;
let $r := r + 2$.
endif
endif
enwhile
STOP: A feasible complementary solution has been computed;
end

The sufficiency of matrix M ensures that (Lemma 6.1.2) the sign of the chosen pivot elements will be as desired in the case of exchange pivot. The algorithm terminates only if there is no solution or if it has found the solution, so it is sufficient to prove that it is

finite. As the number of possible different bases is finite, the proof of finiteness can rely on showing that the criss–cross algorithm cannot cycle when **s**-monotone index selection rules are applied.

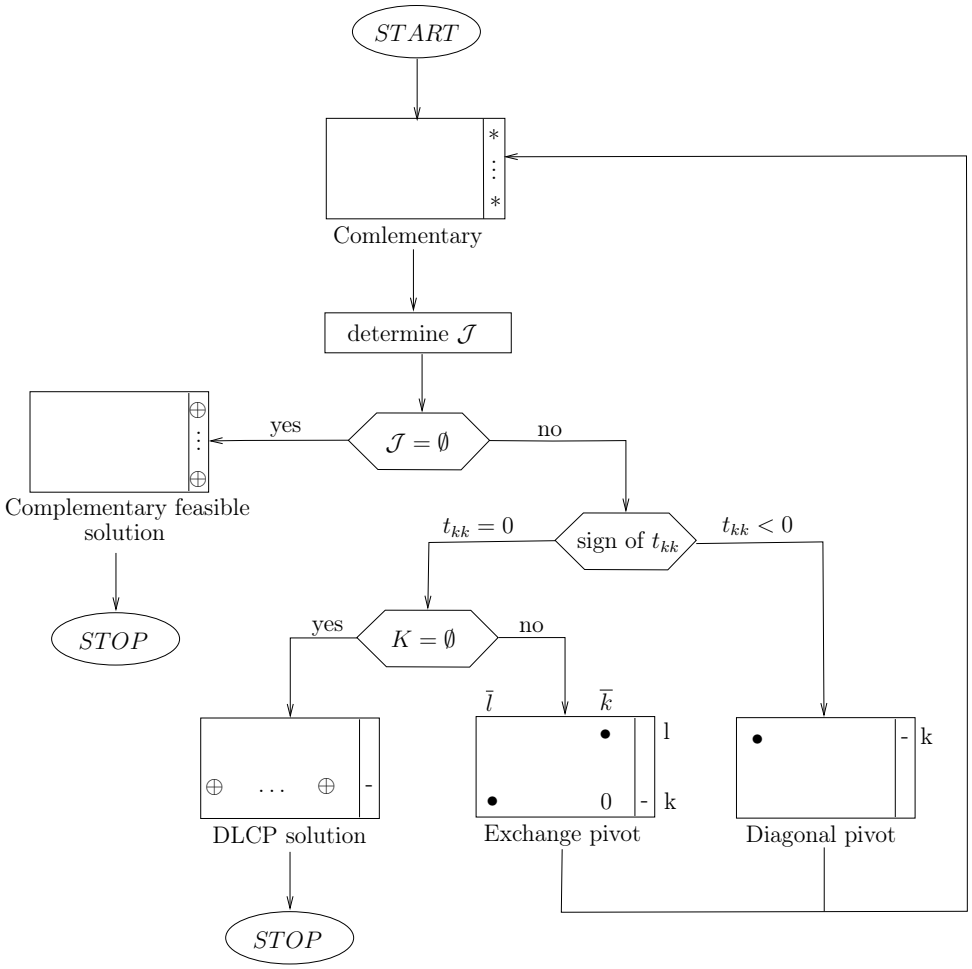


Figure 6.1: Flow chart of the criss–cross algorithm with symmetric pivot rule.

6.1.1 Almost terminal tableaus of the criss–cross method

The finiteness proof of the algorithm is indirect, i.e. we assume that the algorithm cycles and find properties of the algorithm / pivot tableaus that causes contradictions, proving that the assumption that the algorithm cycles is incorrect.

The results of this section are a generalization of the proof presented in [18] to the **s**-monotone case. It is further different from that in [20] and [17] in that in this version of the proofs, while they do follow the same conceptional path, a version of the proof is presented that is more directly connected to the algorithmic implementation by working on the original

problem, and by not requiring the minimality of the problem. As the implementation of this version of the LCP criss–cross will depend on the vectors discussed in the proofs, this is a justified complication.

Assume that an example exists on which the algorithm cycles (as the number of possible bases is at most $\begin{pmatrix} 2n \\ n \end{pmatrix}$, this is necessary if the algorithm is not finite). Also assume, that all variables that only move a finite many times are not being pivoted on any more in the example.

The second criterion of **s**-monotonicity aims at capturing the fundamental idea of most finiteness proofs based on orthogonality, and let us consider the situation it describes: let the variable described in the definition be u_l, outside the bases. The basis where variable u_l enters the basis the first time afterwards the situation described by the second criterion of **s**-monotonicity is denoted by B'.

From among the possible pivot choices, u_l and v_l have the smallest **s** value, and using the symmetry of the rule we get that v_l is the only infeasible variable in basis B' among those variables that will keep changing basis status in the cycling example.

Similarly as in the proof of finiteness for the criss–cross method for linear programming problems, $\mathcal{K}$ denotes the set of indices not moved in between basis B' and the next time variable u_l moves again (which will have to occur, as the definition of **s**-monotonicity considered the variables that move infinity in any infinite pivot sequence).

The structure of the relevant short pivot tableaus are presented in Figures 6.2 and 6.3:

1. The algorithm selects u_l as the incoming variable to the basis.

The diagonal element $t_{ll} < 0$ is negative and a diagonal pivot is made [Figure 6.2, tableau (a)]: u_l enters the basis and v_l leave. The vector **s** is updated symmetrically for the complementary variable pair (u_l, v_l).

2. The algorithm selects u_l to enter the basis, $t_{ll} = 0$ and an exchange pivot is carried out [Figure 6.2, tableau (b)]. Variables u_l and u_j enter the basis and variables v_l and v_j leaves. The vector **s** is modified symmetrically (i.e. the complementary variable pairs always have the same value in **s**), first for the variable pair (u_j, v_j) followed by for variable pair (u_l, v_l) performing the update as if they have moved in a subsequent iteration (the logic of the update mimics two subsequent diagonal pivots.

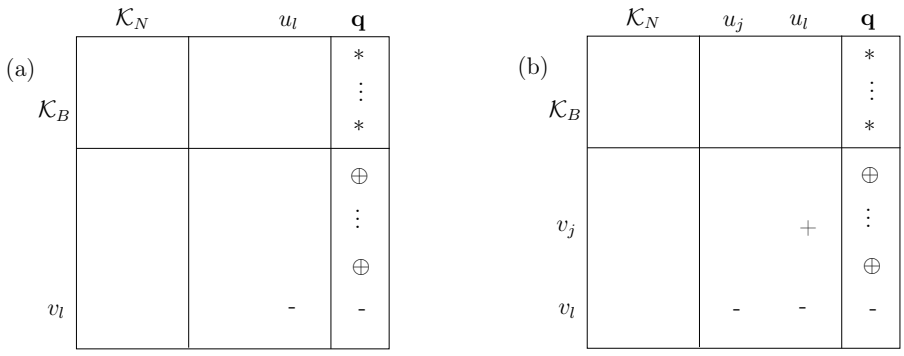


Figure 6.2: Variable u_l is actively selected to enter the bases: first as a diagonal pivot, second as a first pivot of an exchange one.

The column of $\mathbf{q}$ is the same in tableau (b) and in tableau (a), and it is indifferent for the proofs whether u_j or v_j is in the basis. Consider the case when v_j is in the basis; the other case can be proven in a symmetric way.

3. Variable u_j is chosen to enter the basis, $t_{jj} = 0$ and an exchange pivot is carried out, the algorithm chooses the variable u_l as the secondary position (passively), [Figure 6.3, tableau (c)].

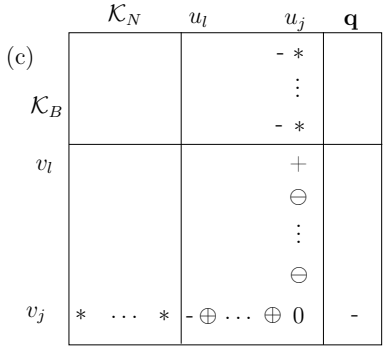


Figure 6.3: Variable u_l is passively selected as the secondary pivot position in an exchange pivot to enter the basis. The expression $-*$ represents a mirrored reverse sign from the transposed position in the matrix.

When u_l enters the basis, in the transformed row of v_j only in the columns corresponding to u_l and the right hand side $\mathbf{q}$ can have a negative element outside of $\mathcal{K}_N$ and as $t_{jj} = 0$ according to Lemma 6.1.2 the sign structure of the column of u_j should mirror the sign structure with the signs flipped. It does not make a difference whether u_j or v_j is in the basis, and so we assume that v_j is in the basis. The vector $\mathbf{s}$ is updated in

the symmetric way first for variable pair (u_l, v_l) followed by variable pair (u_j, v_j) as if moved in the next iteration (again mimicking as if two diagonal pivots were made).

It will be important when extending the algorithm to handle problems where the sufficiency of the matrix may not be known beforehand, that cases 1. and 2. relied only on the conditions described by the **s**-monotone rule for the sign structures, while the 3^{rd} case considered the sufficiency of the matrix when calculating the sign structure for column u_j by mirroring it with a flipped sign structure.

As the algorithm is assumed to cycle and u_l was selected from among the variables that move an infinite many times, then there must be a later basis B'' in which u_l leaves the basis for the first time after basis B'. The possible pivot tableaus for this iteration are presented in Figures 6.4 and 6.5).

A. Using the pivot rule the algorithm chooses variable u_l to leave the basis, $t_{ll} < 0$, and a diagonal pivot is carried out [Figure 6.4, tableau (A)].

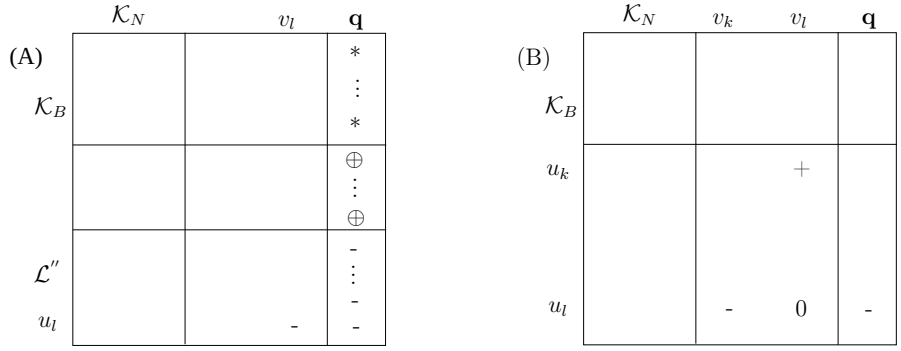


Figure 6.4: Variable u_l is actively selected to leave the bases.

B. Variable u_l is selected to leave the basis, $t_{ll} = 0$ and an exchange pivot is carried out: v_k (or u_k) enters the basis and u_k (or v_k) leaves [Figure 6.4, tableau (B)].

C. Variable u_k (or v_k) is selected, $t_{kk} = 0$ and an exchange pivot takes is carried out, v_l enters the basis and u_k leaves it [Figure 6.5, tableau (C)].

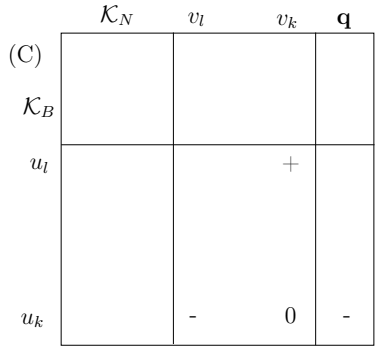


Figure 6.5: Variable u_l is selected passively to leave the basis in the second pivot during an exchange pivot.

The proof is based on showing that none of table $(a) - (c)$ corresponding to basis B' can be followed by any of $(A) - (C)$ corresponding to basis B'', assuming that the matrix M is sufficient. For the extension to the case when the sufficiency of the matrix is not known beforehand, it will be important to distinguish the cases where the proof is based solely on the combinatorial properties of the **s**-monotone rule, or when the sufficiency of the matrix is used; this latter one will mostly be necessary in the case when the sign structure of a column or row is mirrored with a reverse sign stricture from another pivot row or column using Lemma 6.1.2.

Auxiliary lemmas

This section summarizes the lemmas used in the proofs of finiteness. The results in this section follow that in [20] and [17] but modified such that the minimality of the examples are not required, making them directly applicable for the implementations, keeping the working arrays exactly as seen when calculated on actual examples.

The proof that tableau (c) will not be followed by either tableaus (A) or (B) does not depend on the sufficiency of M.

Lemma 6.1.4 *A tableau $T_{B'}$ corresponding to case (c) cannot be followed by tableau $T_{B''}$ corresponding to either case A (or B), as vectors $\mathbf{t}'^{(\bar{j})}$) and $\mathbf{t}''_q$ created from the row of the basic variable v_j in tableau $T_{B'}$ and from the column of $\mathbf{q}$ in $T_{B''}$ has*

$$(\mathbf{t}'^{(\bar{j})})^T \mathbf{t}''_q > 0. \tag{6.5}$$

which would contradict the orthogonality theorem.

Proof. As the variables in $\mathcal{K}$ have not moved in between B' and B'', we have that

$$\sum_{i \in \mathcal{K}} t'_{\bar{j}k} t''_{iq} = \sum_{i \in \mathcal{K}_B} t'_{\bar{j}i} * 0 + \sum_{i \in \mathcal{K}_N} 0 * t''_{iq} = 0. \tag{6.6}$$

as all variables match up being in the basis and not in the basis in the pairs for $\mathbf{t}'$ and $\mathbf{t}''$ as the variables have not moved since.

Let $\mathcal{L}'' := \{\, i \in \mathcal{I}_{B''} \mid \bar{q}''_i < 0 \,\}$ denote the indices in the basis that are not in $\mathcal{K}_B$ and have negative right hand side. Using the third criterion of the **s**-monotone index selection rules, variables in $\mathcal{L}'' - \{\bar{j} \cup l\}$ cannot have moved since basis B', or otherwise their **s**-value would be higher than that of u_l, and so the index selection rule would have selected from among $\mathcal{L}'' - \{\bar{j} \cup l\}$ instead of u_l, so $\mathcal{L}'' \subseteq \mathcal{I}_{B'} \cap \mathcal{I}_{B''}$. Thus it holds that $t'_{\bar{j}i} = 0$ for all indices $i \in \mathcal{L}'' \setminus \{\bar{j}, l\}$, and

$$\sum_{i \in \mathcal{L}'' \setminus \{\bar{j}, l\}} t'_{\bar{j}i} t''_{iq} = 0. \tag{6.7}$$

According to the symmetric update of **s**-monotone rules for LCP problems, in the exchange pivot carried out on B', the **s**-value was first updated for the complementary pair (u_l, v_l) and only after that for (u_j, v_j) as if it was the next pivot. Due to this organization of the update, we know that both t''_{jq} and $t''_{\bar{j}q}$ must be nonnegative, as one is zero as not in the basis, while the other has moved since B' and due to symmetric update has a larger **s**-value than that of u_l, so it cannot be negative otherwise it would have been chosen instead of u_l.

From tableau (c) we know that $t'_{\bar{j}j} = 0$, $t'_{\bar{j}\bar{j}} = 1$, (definition of $\mathbf{t}''$) $t'_{\bar{j}\bar{l}} = 0$, $t'_{\bar{j}l} < 0$ and $t'_{\bar{j}q} < 0$ so

$$t'_{\bar{j}\bar{j}}\, t''_{\bar{j}q} + t'_{\bar{j}j} t''_{jq} + t'_{\bar{j}\bar{l}} t''_{\bar{l}q} + t_{\bar{j}l}\, t''_{lq} + t'_{\bar{j}q}\, t''_{qq} \geq t'_{\bar{j}l} t''_{lq} - t'_{\bar{j}q} > 0, \tag{6.8}$$

as $t''_{qq} = -1$ holds by the definition of the $\mathbf{t}$ vectors, and $t''_{lq} < 0$ also holds due to the pivot selection (tableaus (A) and (B)).

In the case $h \notin \mathcal{K} \cup \mathcal{L}'' \cup \left\{j, \bar{j}, l, \bar{l}, q\right\}$ then $t'_{\bar{j}h} \geq 0$ as seen on the tableaus, and $t''_{hq} < 0$[1] holds for all indices k in $\mathcal{L}''$ by construction, and by the index selection rule $t'_{jh} = 0$ (these variables must not have moved since basis B', otherwise the index selection rule would have not selected l).

$$\sum_{h \notin \mathcal{K} \cup \mathcal{L}'' \cup \{j, \bar{j}, l, \bar{l}, q\}} t'_{\bar{j}h}\; t''_{hq} \geq 0. \tag{6.9}$$

[1] This is incorrectly presented in [17] and [20], but as are paired up by zeros it does not invalidate the proof.

	$\mathcal{K}_B$	$\mathcal{K}_N$	$\mathcal{L}'' \setminus \{\bar{j}, l\}$	j	$\bar{j}$	l	$\bar{l}$	q	rest
$\mathbf{t}'^{(\bar{j})} =$	0 ... 0	* ... *	0 ... 0	0	1	-	0	-	$\oplus$ ... $\oplus$
$\mathbf{t}''_q =$	* ... *	0 ... 0	* ... *	$\oplus$	$\oplus$	-	*	-1	$\oplus$ ... $\oplus$

The result follows as we sum up inequalities (6.7)-(6.9).

■

Similarly, using orthogonality it can be shown that tableaus (a) and (b) will not be followed by tableau (C).

Lemma 6.1.5 *A table $T_{B'}$ corresponding to case (a) (or (b)) cannot be followed by a tableau $T_{B''}$ corresponding to (C), as the vectors $\mathbf{t}'_q$ and $\mathbf{t}''^{(k)}$ from the column of $\mathbf{q}$ in tableau $M_{B'}$ and row u_k of tableau $M_{B''}$ we have*

$$(\mathbf{t}''^{(k)})^T \, \mathbf{t}'_q > 0. \tag{6.10}$$

that would contradict the orthogonality theorem.

Proof. As the variables in $\mathcal{K}$ have not moved since, we have that

$$\sum_{i \in \mathcal{K}} t'_{\bar{j}k} t''_{iq} = \sum_{i \in \mathcal{K}_B} t'_{\bar{j}i} * 0 + \sum_{i \in \mathcal{K}_N} 0 * t''_{iq} = 0. \tag{6.11}$$

as all variables match up being in the basis and not in the basis in the pairs for $\mathbf{t}'$ and $\mathbf{t}''$ as the variables have not moved since.

Let $\mathcal{L}'' := \{i \in \mathcal{I}_{N''} \mid t''_{ki} < 0\}$. Because of the third criterion of **s**-monotonicity, for all index $h \in \mathcal{L}''$ the corresponding variables have not moved since B', otherwise they would have been chosen instead of l, so $t'_{iq} = 0$ for every $i \in \mathcal{L}'' \setminus \{l\}$, thus

$$\sum_{i \in \mathcal{L}'' \setminus \{l\}} t''_{ki} \, t'_{iq} = 0. \tag{6.12}$$

Furthermore, for an index $h \notin \mathcal{K} \cup \mathcal{L}'' \cup \{\bar{k}, k, \bar{l}, l, q\}$, $t''_{kh} \geq 0$ and $t'_{hq} \geq 0$, therefore

$$\sum_{j \notin \mathcal{K} \cup \mathcal{L}'' \cup \{\bar{k}, k, \bar{l}, l, q\}} t''_{kj} t'_{jq} \geq 0. \tag{6.13}$$

According to tableaus $T_{B'}$ and $T_{B''}$ it holds that $t'_{qq} = -1$, $t''_{kk} = 1$, $t''_{k\bar{k}} = t''_{kl} = t'_{lq} = 0$ and

$t''_{k\bar{l}} < 0,\ t''_{kq} < 0,\ t'_{\bar{l}q} < 0,\ t'_{\bar{k}q} \geq 0$ and $t'_{kq} \geq 0$ so

$$t''_{k\bar{k}} t'_{\bar{k}q} + t''_{kk}\, t'_{kq} + t''_{k\bar{l}}\, t'_{\bar{l}q} + t''_{kl} t'_{lq} + t''_{kq}\, t'_{qq} = t'_{kq} + t''_{k\bar{l}}\, t'_{\bar{l}q} - t''_{kq} \geq t''_{k\bar{l}} t'_{\bar{l}q} - t''_{kq} > 0.$$

$\mathbf{t}'_q =$

$\mathcal{K}_B$	$\mathcal{K}_N$	$\mathcal{L}'' \setminus \{l\}$	k	$\overline{k}$	l	$\bar{l}$	q	rest
$*\ \ldots\ *$	$0\ \ldots\ 0$	$0\ \ldots\ 0$	$\oplus$	$\oplus$	0	-	-1	$\oplus\ \ldots\ \oplus$

$\mathbf{t}^{''(k)} =$

$0\ \ldots\ 0$	$*\ \ldots\ *$	$*\ \ldots\ *$	1	0	0	-	-	$\oplus\ \ldots\ \oplus$

The result follows as we sum up inequalities (6.12) – (6.14).

■

Next we show that tableaus (a) (or (b)) will not be followed by either tableaus (A) or (B). These proofs are using the sufficiency of the matrix and so they affect the algorithm if it aims to detect the lack of sufficiency.

Lemma 6.1.6 *Tableau (a) (or (b)) cannot be followed by tableau (A) (or (B)) as for the complementary solutions $(\mathbf{u}', \mathbf{v}')$ and $(\mathbf{u}'', \mathbf{v}'')$ corresponding to tableaus (a) (or (b)) and (A) (or (B)) respectively, the Hadamard product*

$$(\mathbf{u}' - \mathbf{u}'') \cdot\ M\ (\mathbf{u}' - \mathbf{u}'') \lneqq \mathbf{0}, \tag{6.14}$$

i.e. $(\mathbf{u}' - \mathbf{u}'')$ is a strictly sign reversing vector with respect to M which contradicts the sufficiency of the matrix.

Proof. All four cases can be proven at the same time.

First, we reduce the expression in the lemma to a cross product of the solutions only, i.e. remove the matrix M from the expression.

$$\begin{aligned}
(\mathbf{u}' - \mathbf{u}'') \cdot\ M\ (\mathbf{u}' - \mathbf{u}'') &= (\mathbf{u}' - \mathbf{u}'') \cdot\ (\mathbf{q} + M\,\mathbf{u}' - \mathbf{q} - M\,\mathbf{u}'') && (6.15)\\
&= (\mathbf{u}' - \mathbf{u}'') \cdot\ (\mathbf{v}' - \mathbf{v}'') && (6.16)\\
&= \mathbf{u}' \cdot\ \mathbf{v}' - \mathbf{u}' \cdot\ \mathbf{v}'' - \mathbf{u}'' \cdot\ \mathbf{v}' + \mathbf{u}'' \cdot\ \mathbf{v}'' && (6.17)\\
&= -\mathbf{u}' \cdot\ \mathbf{v}'' - \mathbf{u}'' \cdot\ \mathbf{v}', && (6.18)
\end{aligned}$$

where the last equation holds as the solutions are complementary.

Just like in the previous lemmas, as the variables in $\mathcal{K}$ have not moved since, we have that

$$\sum_{i \in \mathcal{K}} u'_i \; v''_i + u''_i \; v'_i = 0. \tag{6.19}$$

as exactly one of the indices will be in the basis or outside the basis as the tableau is complementary, and these variables have not moved since.

Let $\mathcal{L}'' := \{\, i \in \mathcal{I}_{B''} \mid \bar{q}''_i < 0 \,\}$. Using **s**-monotonicity it is clear that the variables indexed by $\mathcal{L}''$ must have not moved since bases B' or otherwise the algorithm would have selected one among them, and so for all $i \in \mathcal{L}'' \backslash \{l\}$ the value of u'_i (or v''_i) and u''_i (or v'_i) must be zero:

$$u'_i \; v''_i + u''_i \; v'_i = 0. \tag{6.20}$$

By the structure of tableau (a) (or (b)), and tableau (A) (or (B)) it is clear that $u'_l = 0, v'_l < 0$ and $u''_l < 0$, $v''_l = 0$ so,

$$u'_l \; v''_l + u''_l \; v'_l > 0. \tag{6.21}$$

For any $h \notin \mathcal{L}''$ it holds that $u'_h, v'_h, u''_h, v''_h \geq 0$ so

$$u'_h \; v''_h + u''_h \; v'_h \geq 0. \tag{6.22}$$

Summing up, the vector $(\mathbf{u}' - \mathbf{u}'')$ is such that $(\mathbf{u}' - \mathbf{u}'') \cdot \; M \; (\mathbf{u}' - \mathbf{u}'') \lneqq \mathbf{0}$ which contradicts the sufficiency of the matrix.

■

It is important to note that vector $\mathbf{u}' - \mathbf{u}''$ can be used as a proof for the lack of sufficiency of M, and can easily be created from the tableaus of bases B' and B'' (as they are the basis solutions).

Finally, we show that tableau (c) cannot be followed by tableau (C). This case also depends on the sufficiency of matrix M.

Lemma 6.1.7 *Tableau $T_{B'}$ corresponding to case (c) cannot be followed by tableau $T_{B''}$ corresponding to case (C), as for vectors $\mathbf{t}'_j$ and $\mathbf{t}''^{(k)}$ corresponding to the column of u_j in tableau $T_{B'}$ and to the row of u_k in tableau $T_{B''}$ we would have*

$$(\mathbf{t}''^{(k)})^T \, \mathbf{t}'_j < 0, \tag{6.23}$$

contradicting the orthogonality theorem.

Proof. As the variables in $\mathcal{K}$ have not moved since, we have that

$$\sum_{i\in\mathcal{K}} t'_{\bar{j}k}t''_{iq} = \sum_{i\in\mathcal{K}_B} t'_{\bar{j}i} * 0 + \sum_{i\in\mathcal{K}_N} 0 * t''_{iq} = 0. \tag{6.24}$$

as all variables match up being in the basis and not in the basis in the pairs for $\mathbf{t}'$ and $\mathbf{t}''$ as the variables have not moved since.

Let $\mathcal{L}'' = \{i \in \mathcal{I}_{N''} : t''_{ki} < 0\} \setminus \{j\}$. Using the second criterion of **s**-monotone rules again, the variables of the indices $\mathcal{L}''$ have not moved since B', so $(\mathcal{I}_{N''} \setminus \mathcal{L}'') \subset \mathcal{I}_{B'}$ and $\mathcal{L}'' \subset \mathcal{I}_{N'}$, thus $t'_{ij} = 0$ if $i \in \mathcal{K}''_k$. Summing up,

$$\sum_{i\in\mathcal{L}''\cup\{q\}} t''_{ki}\, t'_{ij} = 0. \tag{6.25}$$

Also, if $h \notin \mathcal{K} \cup \mathcal{L}'' \cup \left\{q, l, \bar{l}, j, \bar{j}, k, \bar{k}\right\}$ then $t'_{hj} \leq 0$ according to tableau (c). According to the definition of $\mathcal{L}''$, $t''_{kh} \geq 0$, so

$$\sum_{h\notin\mathcal{K}\cup\mathcal{L}''\cup\left\{q,l,\bar{l},j,\bar{j},k,\bar{k}\right\}} t''_{kh}\, t'_{hj} \leq 0, \tag{6.26}$$

From tableaus $T_{B'}$ and $T_{B''}$ using the definition of vector $\mathbf{t}$ it follows that

$$t'_{lj} = t''_{k\bar{k}} = t'_{\bar{j}j} = t'_{qj} = t''_{kl} = 0,\ t''_{kk} = 1,\ t'_{jj} = -1 \ \text{ and } t'_{kj} \leq 0,\ t''_{k\bar{l}} < 0,\ t'_{\bar{l}j} > 0 \tag{6.27}$$

so

$$t''_{kq}\, t'_{qj} + t''_{kl}\, t'_{lj} + t''_{k\bar{l}}\, t'_{\bar{l}j} + t''_{kj}\, t'_{jj} + t''_{k\bar{j}}\, t'_{\bar{j}j} + t''_{kk}\, t'_{kj} + t''_{k\bar{k}}\, t'_{\bar{k}j} < -t''_{kj}. \tag{6.28}$$

As the update of the **s** vector has been defined to be symmetric, during the exchange pivot in tableau c the update of **s** is first carried out for (u_l, v_l) followed by (u_j, v_j) as if in a second pivot. This way the variable pair (u_j, v_j) is already considered to be moved since B' and so using the second criterion of **s**-monotone pivot rules the associated **s** value must be larger than for index l. This can only hold if $t''_{kj} \geq 0$ either as it is not in the bases, or because of its corresponding **s** value.

	$\mathcal{K}_B$	$\mathcal{K}_N$	$\mathcal{L}''\cup\{q\}$	k	$\bar{k}$	l	$\bar{l}$	j	$\bar{j}$	q	rest
$\mathbf{t}'_j =$	$*\ \ldots\ *$	$0\ \ldots\ 0$	$0\ \ldots\ 0$	$\ominus$	*	0	+	-1	0	0	$\ominus\ \ldots\ \ominus$

$\mathbf{t}''^{(k)} =$	$0\ \ldots\ 0$	$*\ \ldots\ *$	$*\ \ldots\ *$	1	0	0	-	$\oplus$	*	*	$\oplus\ \ldots\ \oplus$

The result follows as we sum up inequalities (6.25) – (6.28).

■

6.1.2 Finiteness of the criss–cross method

In this section we prove the finiteness of the criss–cross algorithm using **s**-monotone index selection rules.

Theorem 6.1.1 *The criss–cross type algorithm with* **s***-monotone index selection rules is finite for the linear complementarity problem with sufficient matrices.*

Proof. Assume to the contrary that the algorithm is not finite. As the number of possible different basis are finite, it is only possible for the algorithm to generate an infinite number of iterations if it cycles. Consider such a cycling example, and consider the index l described by the definition of **s**-monotone rule, also used in the auxiliary lemmas. Once variable u_l enters the basis after basis B', it cannot leave it again:

If it enters in the case (a) or (b) and leaves the basis in the case (A) or (B), Lemma 6.1.6. contradicts the sufficiency of matrix M.

If it enters in the case (c) and leaves the basis in the case (A) or (B), Lemma 6.1.4. contradicts the orthogonality theorem.

If it enters in the case (c) and leaves the basis in the case (C), Lemma 6.1.7. contradicts the orthogonality theorem.

If it enters in the case (a) or (b) and leaves the basis in the case (C), Lemma 6.1.5. contradicts the orthogonality theorem.

This contradicts the definition of index l from the definition of the **s** monotone index selection rule, proving the theorem. ■

Figure 6.6 shows the cases in which the sufficiency of matrix T has been used in the proof of finiteness of the criss–cross type algorithm.

	(a)	(b)	(c)
(A)	$*$	$*$	
(B)	$*$	$*$	
(C)			$*$

Figure 6.6: The cases when sufficiency of the pivot matrix is used.

6.2 EP theorems and the linear complementarity problem

This section presents as generalization if the algorithm in the sense of EP theorems as in [17]. As motivation, Example 6.2.1 demonstrates that the criss–cross algorithm may solve a LCP problem even if the matrix is not sufficient.

Example 6.2.1 *To demonstrate the criss–cross method, consider the linear complementarity problem with the matrix presented in Example 6.1.1 with the corresponding short pivot tableau where the identity matrix corresponding to* $\mathbf{v}$ *is used as an initial complementary basis.*

Solving the problem with the cross-cross method, pivoting first on diagonal elements (u_1, v_1) *and* (u_2, v_2)*:*

1.	u_1	u_2	u_3	u_4	u_5	$\mathbf{q}$
v_1	***-1***	*1*	*1*	*0*	*1*	*-1*
v_2	*1*	*-2*	*0*	*0*	*1*	*-2*
v_3	*-1*	*2*	*1*	*2*	*0*	*16*
v_4	*4*	*1*	*1*	*-2*	*-4*	*6*
v_5	*1*	*0*	*-2*	*0*	*-1*	*4*

2.	v_1	u_2	u_3	u_4	u_5	$\mathbf{q}$
u_1	*-1*	*-1*	*-1*	*0*	*-1*	*1*
v_2	*1*	***-1***	*1*	*0*	*2*	*-3*
v_3	*-1*	*1*	*0*	*2*	*-1*	*17*
v_4	*4*	*5*	*5*	*-2*	*0*	*2*
v_5	*1*	*1*	*-1*	*0*	*0*	*3*

then (u_4, v_4)*:*

3.	v_1	v_2	u_3	u_4	u_5	$\mathbf{q}$
u_1	*-2*	*-1*	*-2*	*0*	*-3*	*4*
u_2	*-1*	*-1*	*-1*	*0*	*-2*	*3*
v_3	*0*	*1*	*1*	*2*	*1*	*14*
v_4	*9*	*5*	*10*	***-2***	*10*	*-13*
v_5	*2*	*1*	*0*	*0*	*2*	*0*

4.	v_1	u_2	u_3	v_4	u_5	$\mathbf{q}$
u_1	*-2*	*-1*	*-2*	*0*	*-3*	*4*
u_2	*-1*	*-1*	*-1*	*0*	*-2*	*3*
v_3	*9*	*6*	*11*	*1*	*11*	*1*
u_4	$-\frac{9}{2}$	$-\frac{5}{2}$	*-5*	$-\frac{1}{2}$	*-5*	$\frac{13}{2}$
v_5	*2*	*1*	*0*	*0*	*2*	*0*

arriving at the solution $u_1 = 4, u_2 = 3, v_3 = 1, u_4 = 6.5$ *and all other variables zero. The method has found a feasible complementary solution for a problem with non-sufficient matrix.*

Example 6.2.1 compensated for an obvious matrix coefficient making the matrix non-sufficient (diagonal entry for (u_3, v_3)) with a large right hand side value. As example 6.2.2 shows, this is not necessary.

Example 6.2.2 *Consider the same M matrix as in Example 6.2.1, but with a different right hand side.*

1.	u_1	u_2	u_3	u_4	u_5	$\mathbf{q}$
v_1	***-1***	*1*	*1*	*0*	*1*	*-1*
v_2	*1*	*-2*	*0*	*0*	*1*	*2*
v_3	*-1*	*2*	*1*	*2*	*0*	*-1*
v_4	*4*	*1*	*1*	*-2*	*-4*	*5*
v_5	*1*	*0*	*-2*	*0*	*-1*	*1*

2.	v_1	u_2	u_3	u_4	u_5	$\mathbf{q}$
u_1	*-1*	*-1*	*-1*	*0*	*-1*	*1*
v_2	*1*	*-1*	*1*	*0*	*2*	*1*
v_3	*-1*	*1*	*0*	*2*	*-1*	*0*
v_4	*4*	*5*	*5*	*-2*	*0*	*1*
v_5	*1*	*1*	*-1*	*0*	*0*	*0*

Pivoting on (u_1, v_1) arrives at a feasible complementary solution, even though the row with the diagonal with incorrect sign in respect to sufficiency started out to be infeasible.

The purpose of the generalization is to be able to solve (some of) the instances that lack sufficiency.

The main idea behind an EP theorem is to provide a set of alternatives, one of which must always hold, and ones that can be validated by a constructive proof that has a bit length size that can be bounded by a polynomial of the bit length size of the input data.

The general form of an EP (Existentially Polynomial time) theorem is as follows [11]:

$$[\forall \mathbf{x} : F_1(\mathbf{x}) \text{ or } F_2(\mathbf{x}) \text{ or } \ldots \text{ or } F_k(\mathbf{x})] \tag{6.29}$$

where $F_i(\mathbf{x})$ is a statement of the form

$$F_i(\mathbf{x}) = [\exists \mathbf{y}_i \text{ for which } \|\mathbf{y}_i\| \leq \|\mathbf{x}\|^{n_i} \text{ and } f_i(\mathbf{x}, \mathbf{y}_i)]\,. \tag{6.30}$$

The extended algorithm makes use of the following two theorems.

Theorem 6.2.1 *[26] Let the matrix $M \in \mathbb{R}^{n \times n}$ be not sufficient. In this case, a certificate exists that M is not sufficient, the coding size of which is polynomially bounded by the input length of matrix M .*

Theorem 6.2.2 *[26]. For any rational matrix $M \in \mathbb{Q}^{n \times n}$ and rational vector $\mathbf{q} \in \mathbb{Q}^n$, at least one of the following must hold:*

(1) the primal LCP problem (P-LCP) has a complementary and feasible solution, that has a bit length size that is polynomially bounded by the bit length size of input matrix M and right hand side vector $\mathbf{q}$.

(2) the dual LCP problem (D-LCP) has a complementary and feasible solution, that has a bit length size that is polynomially bounded by the bit length size of the input matrix M and right hand side vector $\mathbf{q}$.

(3) in the case that matrix M is not sufficient, there exists a certificate for the M not being sufficient, that has a bit length that is polynomially bounded by the bit length size of the input matrix M.

Cases (1) and (2) are exclusive regardless of any properties of the matrix M, while case (3) may hold separately or at the same time as either case (1) or (2).

The criss–cross algorithm can be modified to conform to the 3 cases of the Lemma, i.e. either solve the primal LCP problem, or show that the primal problem is infeasible by finding a solution to the dual problem, or finding a certificate that the input matrix is not sufficient.

According to Lemma 6.1.2, the pivots can always be made if the matrix is sufficient, and if it is not its proof gives a recipe for creating the certificate that matrix M is not sufficient.

The extended version of the criss–cross type method for the general linear complementarity problem is presented below. The definition and role of Q as marked at (1)-(4) in the pseudo code will be discussed afterwards.

The criss–cross type algorithm with s-monotone index selection rules in the form of EP-theorems

input data:

$T = -M$, $\bar{\mathbf{q}} = \mathbf{q}$, $r = 1$, initialize Q and $\mathbf{s}$; (1)

begin

$\mathcal{J} := \{i \in \mathcal{I} \mid \bar{\mathbf{q}}_i < 0\}$;

while $(\mathcal{J} \neq \emptyset)$ **do**

$\mathcal{J}_{\max} := \{\beta \in \mathcal{J} \mid \mathbf{s}(\beta) \geq \mathbf{s}(\alpha)$, for all $\alpha \in \mathcal{J}\}$;

let $k \in \mathcal{J}_{\max}$ be arbitrary;

check $-\mathbf{u}' \cdot \mathbf{v}'' - \mathbf{u}'' \cdot \mathbf{v}''$ with the help of $Q(k)$; (2)

if $(-\mathbf{u}' \cdot \mathbf{v}'' - \mathbf{u}'' \cdot \mathbf{v}'' \lneqq \mathbf{0})$ **then**

STOP: M is not sufficient, certificate: $\mathbf{u}' - \mathbf{u}''$;

endif

if $(t_{kk} < 0)$ **then**

diagonal pivot on t_{kk}, update $\mathbf{s}$;

$Q(k) = [\mathcal{J}_B, \bar{\mathbf{t}}_q]$, $r := r + 1$; (3)

else if $(t_{kk} > 0)$ **then**

STOP: M is not sufficient, create certificate;

else /* $t_{kk} = 0$ */

$K := \{\alpha \in I \mid \bar{t}_{k\alpha} < 0\}$;

if $(K = \emptyset)$ **then** STOP: DCLP solution;

else

$\mathcal{K}_{\max} = \{\beta \in K \mid \mathbf{s}(\beta) \geq \mathbf{s}(\alpha)$, for all $\alpha \in K\}$;

let $l \in \mathcal{K}_{\max}$ be arbitrary;

if (($\mathbf{t}_k$, $\mathbf{t}^k$) or ($\mathbf{t}_l$,$\mathbf{t}^l$) sign structure is violated) **then**

STOP: M is not sufficient, create certificate;

endif

exchange pivot on t_{kl} and t_{lk}, update $\mathbf{s}$ first for (u_k, v_k);

then for (u_l, v_l) as in a next iteration;

$Q(k) = [J_B, \bar{\mathbf{t}}_q]$, $Q(l) = [\emptyset, \mathbf{0}]$, $r := r + 2$; (4)

endif

endif

endwhile

STOP: we have a complementary feasible solution;

end

As Example 6.2.3 shows, in the case of lack of sufficiency the criss–cross method may cycle.

Example 6.2.3 *Consider the same M matrix as in Example 6.2.1 and Example 6.2.2 but with another, different right hand side.*

The criss–cross algorithm first pivots on (u_1, v_1). In the second tableau, an exchange pivot is necessary, pivoting on (u_3, v_5) and then (u_5, v_3) leading to Tableau 4.

1.	u_1	u_2	u_3	u_4	u_5	**q**
v_1	***-1***	1	1	0	1	-1
v_2	1	-2	0	0	1	2
v_3	-1	2	1	2	0	0
v_4	4	1	1	-2	-4	10
v_5	1	0	-2	0	-1	0

2.	v_1	u_2	u_3	u_4	u_5	**q**
u_1	-1	-1	-1	0	-1	1
v_2	1	-1	1	0	2	1
v_3	-1	1	0	2	-1	1
v_4	4	5	5	-2	0	6
v_5	1	1	***-1***	0	0	-1

On Tableau 4, another exchange pivot takes place, (v_3, u_5) and then (v_5, u_3) resulting in Tableau 6.

3.	v_1	u_2	v_5	u_4	u_5	**q**
u_1	-2	-2	-1	0	-1	2
v_2	2	0	1	0	2	0
v_3	-1	1	0	2	***-1***	1
v_4	9	10	5	-2	0	1
u_3	-1	-1	-1	0	0	1

4.	v_1	u_2	v_5	u_4	v_3	**q**
u_1	-1	-3	-1	-2	-1	1
v_2	0	2	1	4	2	2
u_5	1	-1	0	-2	***-1***	-1
v_4	9	10	5	-2	0	1
u_3	-1	-1	-1	0	0	1

5.	v_1	u_2	v_5	u_4	u_5	**q**
u_1	-2	-2	-1	0	-1	2
v_2	2	0	1	0	2	0
v_3	-1	1	0	2	-1	1
v_4	9	10	5	-2	0	1
u_3	-1	-1	***-1***	0	0	1

6.	v_1	u_2	u_3	u_4	u_5	**q**
u_1	-1	-1	-1	0	-1	1
v_2	1	-1	1	0	2	1
v_3	-1	1	0	2	-1	1
v_4	4	5	5	-2	0	6
v_5	1	1	-1	0	0	-1

Tableau 6 coincides with Tableau 2: the pivot choices have been unique as the negative right hand side values were unique: the algorithm cycles.

The extended finiteness considerations to the algorithm in the case of matrices where sufficiency is not known beforehand requires the algorithm to detect the lack of sufficiency. This is required for two reasons. One is when the structure of the matrix would mean that the selected pivot operations cannot be carried out because the sign structure of the matrix

is not as expected. This is straightforward algorithmically, as the pivot columns / row needs to be calculated anyway.

The harder to handle side effect of the potential lack of sufficiency is the possibility of cycling. It is not viable to record the basis that the algorithm has already visited and catch cycling indirectly, due to the exponential number of potential (even if complementary) basis. Instead, the algorithm makes sure that the situations analysed in the proofs are valid, which in practice means that in each pivot the full sign structure of the selected pivot row and column is checked.

We have presented a slightly modified version of the finiteness proofs compared to the original presented in [20] and [17], in that we did not assume minimality of the cycling example. This simplifies the arguments that the checking of the sign strictures of the relevant rows and columns of the pivot tableau is sufficient to enforce complementarity in the implementation.

In all tableau combinations of the proofs above with the exception of tableaus (a)-(b) and (A)-(B), enforcing the match of the sign structure (i.e. that it matches the results of Lemma 6.1.2) is sufficient to enforce that due to the orthogonality theorem, cycling cannot occur.

In the case of (a)-(b) and (A)-(B), the proofs rely on the product

$$-\mathbf{u}' \cdot \mathbf{v}'' - \mathbf{u}'' \cdot \mathbf{v}'. \tag{6.31}$$

To enforce that these cases are covered, in all iterations for the variable chosen actively (in other words not as a result of a second index selection during an exchange pivot) the relevant vectors are saved, that the product of 6.31 is checked. If the product is not strictly negative, the vector $\mathbf{u}' - \mathbf{u}''$ proofs the lack of sufficiency.

To save the relevant vectors, the algorithm introduces $Q(p)$ $(p = 1, \ldots, n)$, initialized to

$$Q(p) := \begin{bmatrix} [1, \ldots, n] \\ [0, \ldots, 0] \end{bmatrix} \quad p = 1, \ldots, n. \tag{6.32}$$

For a variable u_l or v_l that left the basis actively, the algorithm either by a diagonal pivot or as the first selected one of an exchange pivot, we update the corresponding vector $Q(l)$ such that it can be used as a storage to check that the conditions described by the auxiliary lemmas hold. To do this, first the *indices* of variables in basis before the pivot operation is carried out is saved, while the *values* of the basic variables are saved to the second vector

[19].

$$Q(l) := \left[\begin{array}{l} [\text{ indices of the basis variables}] \\ [\text{ values of the basic variables}]\,. \end{array} \right] \tag{6.33}$$

If variable u_l or v_l enters the basis passively (its index is the second selection during an exchange pivot), then the values of $Q(l)$ are cleared:

$$Q(l) := \left[\begin{array}{l} [1, \ldots, n] \\ [0, \ldots, 0] \end{array} \right] \tag{6.34}$$

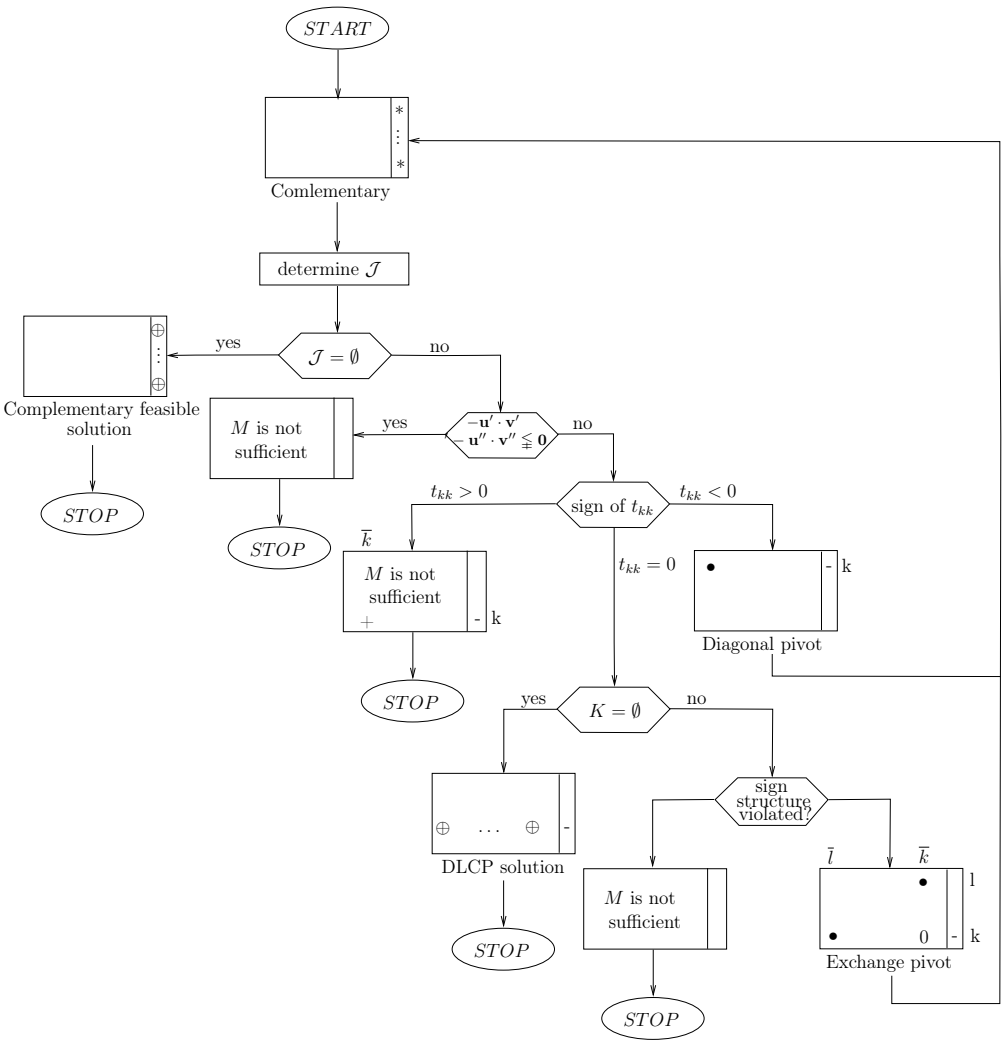


Figure 6.7: Flow chart of the criss–cross algorithm in the form of EP-theorems.

An operation $Q(j) = [\{I\}, \{\mathbf{h}\}]$ means that to the entry of j in the list Q, we write list I to the place of basic indices, while the values of the vector $\mathbf{h}$ in the place of $\mathbf{q}$.

When the algorithm has to perform a pivot on an actively selected variable, it checks if the variable in question has also been selected actively the last time it was pivoted on. If so, it checks the product of (6.31) using the current solution vector and that which has been saved in Q for the variable (clearing it to zeros when a variable is selected passively makes this check automatic by resulting in a zero in the cases when the variable was last selected passively). Once the check is carried out, the algorithm updates Q. Note, that since the complementary variable pairs move together in and out of the basis, it is not necessary to reserve space for both in Q, as (6.31) assumed that the product is checked when a variable enters the basis (so for a complementary pair, Q is filled out for the one leaving the basis during the pivot).

It is not necessary to save all the data to Q at each pivot operation. However, by always storing the data the algorithm becomes simpler, and the worst case storage space requirement is equivalent to storing n^2 integer and n^2 rational numbers, which is comparable to the storage space necessary to store the original matrix (although in large scale implementations this is not true as the M matrix is typically sparse).

In the case when there is no solution to $(P - LCP)$ ($K = \emptyset$ in the flow chart), the corresponding pivot tableau is presented in Figure 6.8. In this case, vector

$$(\mathbf{x}', \mathbf{y}') = \mathbf{t}^{(k)} \mid_{J_N \cup J_B} . \tag{6.35}$$

is orthogonal to every row of $[-M^T \mid -I]$ according to the orthogonality theorem, thus $M^T\mathbf{x}' + \mathbf{y}' = \mathbf{0}$. Applying this to the column of the right-hand side vector $\mathbf{q}$ we have

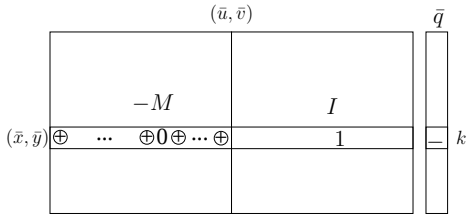
$$(\mathbf{x}', \mathbf{y}')^T \mathbf{t}_q \mid_{J_N \cup J_B} = (\mathbf{x}', \mathbf{y}')^T (\mathbf{q}, \mathbf{0}) = \mathbf{x}'^T \mathbf{q} = q_k. \tag{6.36}$$

which means that $(\mathbf{x}, \mathbf{y}) = (\mathbf{x}', \mathbf{y}')/(-q_k)$ is a solution to the dual LCP problem $(D - LCP)$, as the nonnegativity and complementary requirements follow from the structure of the pivot tableau.

To summarize, the extended criss–cross algorithm can be started from any complementarity basic solution of an arbitrary linear complementarity problem, without the need for a priori information on the properties of the M matrix, and using **s**-monotone index rule the algorithm will terminate in a finite number of steps, either solving the problem, or proving that it is infeasible, or giving a certificate that matrix M is not sufficient (as in Theorem

6.2.2).

The next example demonstrates that the modified criss–cross algorithm (Figure 6.7) identifies the lack of sufficiency for matrix M.



$$(x,y)_i = \begin{cases} -t_{ki}/\bar{q}_k & \text{if } i \in I_N \\ -1/\bar{q}_k & \text{if } i = k \\ 0 & \text{otherwise.} \end{cases}$$

Figure 6.8: Assembling a dual solution in case no primal solution exists.

Example 6.2.4 *As an example, consider the problem presented in Example 6.2.3 again. The extended algorithm would stop on Tableau 3: the sign of the second pivot position of the exchange pivot violates the expected sign structure.*

6.3 Computational experiences

In this section we provide some numerical experience using the proposed algorithm for solving general LCP problems arising from Arrow-Debreu exchange matrix problems and bimatrix games. The experiments have been carried out in Matlab, using the built in QR decomposition and update for the basis. To maximize the chance of success, in positions when the selected **s**-monotone rule (MOSV) offered flexibility of pivot selection, a random position has been selected from among the eligible choices.

We implemented our algorithms in Matlab [3], with the inverse of the basis being calculated using dense matrix factorization. During these experiments, the use of dense linear algebra did not prove to be a bottleneck (more details can be found in Section 4.2).

The results presented here emphasize the applicability of the new variant of the criss–cross method together with the flexibility of the **s**-monotone index selection rules. For a comprehensive numerical study concentrating on the **s**-monotone index selection rules see [34] and Chapter 5.

6.3.1 A market equilibrium problem

Consider the exchange market equilibrium problem as described by Walras [71]. There are m traders (players) and n goods on the market, where each good type j has a price $p_j \geq 0$.

Each trader i is assumed to have an initial endowment of commodities $\mathbf{w}_i = (w_{i1}, \ldots, w_{in}) \in \mathbb{R}^n_\oplus$. The traders will sell their product on the market and use their income to buy a bundle of goods $\mathbf{x}_i = (x_{i1}, \ldots, x_{in}) \in \mathbb{R}^n_\oplus$. A trader i has a utility function u_i, which describes his preferences for the different bundle of commodities and a budget constraint $\mathbf{p}^T\mathbf{x}_i \leq \mathbf{p}^T\mathbf{w}_i$. Finally, each trader i maximizes his individual utility function subject to his budget constraint.

Each trader optimizes his own utility function u_i with these side constraints:

$$\begin{array}{rcl} \max & u_i(\mathbf{x}_i) & \\ \mathbf{p}^T\mathbf{x}_i & \leq & \mathbf{p}^T\mathbf{w}_i \\ \mathbf{x}_i & \geq & \mathbf{0}, \end{array} \tag{6.37}$$

where the vector of prices $\mathbf{p}$ is an equilibrium for the exchange economy; if there is a bundle of goods $\mathbf{x}_i(\mathbf{p})$ (so a maximizer of the utility function u_i subject to the budget constraint) for all traders i, such that

$$\sum_{i=1}^{m} x_{ij}(\mathbf{p}) \leq \sum_{i=1}^{m} w_{ij} \quad \text{for all goods } j. \tag{6.38}$$

The exchange market equilibrium problem pursues prices where the demand $\sum_i x_{ij}(\mathbf{p})$ does not exceed the supply $\sum_i w_{ij}$ for any good j.

Arrow and Debreau [7] proved that under mild conditions, for concave utility functions, the exchange markets equilibrium exists. Using the Leontief utility function

$$u_i(\mathbf{x}_i) = \min_j \left\{ \frac{x_{ij}}{a_{ij}} : \ a_{ij} > 0 \right\}, \tag{6.39}$$

where $A = (a_{ij}) \in \mathbb{R}^{n\times n}_\oplus$ is the Leontief coefficient matrix, Ye [73] has shown that the solution of the Arrow-Debreu competitive market equilibrium problem with Leontief's utility function is equivalent to the following linear complementarity problem:

$$A^T\,\mathbf{u} + \mathbf{v} = \mathbf{e}, \quad \mathbf{u} \geq \mathbf{0},\, \mathbf{v} \geq \mathbf{0}, \quad \mathbf{u}\,\mathbf{v} = \mathbf{0} \ \text{ and } \ \mathbf{u} \neq \mathbf{0} \tag{6.40}$$

where the matrix A has non-negative entries.

It is easy to see that this problem will almost always have a non-sufficient matrix, and as such is a demanding problem class for the generalized criss–cross algorithm.

The computational experiences have been carried out using a 100 problems, ranging $n \in \{10, 20, 40, 60, 80, 100, 200, 300, 400, 500\}$ taking 10 random instances for each value of n. For this problem the trivial basis corresponding to the columns of $\mathbf{v}$ is not valid, as it is a feasible solution to the problem, but $\mathbf{u} \neq \mathbf{0}$ does not hold.

To address this problem, a structural crash heuristic has been used, based on the following heuristics. Start from the empty selection $B = \emptyset$. For any set of vectors, define it's support as $supp(B) = \{j : \exists i : \mathbf{a}_i \in B, a_{ji} \neq 0\}$. In each iteration of the crash heuristics for all $i \in 1 \ldots n$, if there is such an index j for which $a_{ji} \neq 0$ and $i \notin supp(B)$, then add $\mathbf{a}_i$ to B, else, add the corresponding identity vector e_i to the B. It is easy to see, that this procedure will yield a complementary basis, and unless $A = 0$ it will contain at least one column corresponding to $\mathbf{u}$. When the selection of the column from A was not unique, the algorithm has selected randomly. Each experiment was repeated 10 times, yielding a total of 1000 test runs.

n	**Successful solves**	**Mean iterations**	**Maximum iterations**
10	762	0.619	12
20	318	1.432	12
40	26	1.445	9
60	13	1.479	10
80	2	1.396	9
100	0	1.295	10
200	0	1.213	9

Table 6.1: Numerical experiments on random market equilibrium problems.

The number of successful solves diminished very quickly with size, also the algorithm terminates after a very small iteration count, with declaring the matrix not sufficient in most cases. There has been no successful solves for $n \geq 100$ suggesting that most of the successful solves for smaller sizes strongly depend on luck associated with the crash heuristics applied.

6.3.2 A bimatrix game

In this problem, two companies are considering entering n markets. Entering a market has both fixed, and variable costs, and there are fixed transport charges. The problem of finding optimal profit strategies can be formulated as a bimatrix game [47]. Consider a bimatrix game defined by A and B.

Theorem 6.3.1 *The Karush-Khun-Tucker conditions of the bimatrix game can equivalently be formulated as*

$$\begin{aligned} \max \mathbf{x}^T(A &+ B)\mathbf{y} - \alpha - \beta && (6.41)\\ \mathbf{x}, \mathbf{y} &\geq \mathbf{0} && (6.42)\\ \mathbf{1}_m^T\mathbf{x} &= \mathbf{1} && (6.43)\\ \mathbf{1}_n^T\mathbf{y} &= \mathbf{1} && (6.44)\\ A\mathbf{y} &\leq \alpha\mathbf{1}_m && (6.45)\\ B^T\mathbf{x} &\leq \beta\mathbf{1}_n && (6.46) \end{aligned}$$

where the zero valued solutions are the Nash-equilibria.

This Karush-Khun-Tucker conditions for this quadratic programming problem can be stated as

$$\begin{pmatrix} Q & P^T \\ -P & O \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{t} \end{pmatrix} + \begin{pmatrix} \mathbf{s} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{c} \\ \mathbf{b} \end{pmatrix} \tag{6.47}$$

$$\begin{aligned} \mathbf{u}\,\mathbf{t} &= \mathbf{0} && (6.48)\\ \mathbf{s}\,\mathbf{v} &= \mathbf{0} && (6.49)\\ \mathbf{u}, \mathbf{t}, \mathbf{s}, \mathbf{v} &\geq \mathbf{0} && (6.50) \end{aligned}$$

where

$$Q = \begin{pmatrix} O & -A - B \\ (-A-B)^T & O \end{pmatrix} \qquad \mathbf{u} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \in \mathbb{R}^{m+n}$$

$$P = \begin{pmatrix} B^T & O & 0 & 0 & -1 & 1 \\ O & A & -1 & 1 & 0 & 0 \\ \mathbf{1}^T & \mathbf{0} & 0 & 0 & 0 & 0 \\ -\mathbf{1}^T & \mathbf{0} & 0 & 0 & 0 & 0 \\ \mathbf{0} & \mathbf{1}^T & 0 & 0 & 0 & 0 \\ \mathbf{0} & -\mathbf{1}^T & 0 & 0 & 0 & 0 \end{pmatrix} \qquad \mathbf{t} = \begin{pmatrix} \alpha^+ \\ \alpha^- \\ \beta^+ \\ \beta^- \end{pmatrix} \in \mathbb{R}^4.$$

As A and B can be chosen arbitrarily, it is easy to see that the resulting LCP will not necessarily be sufficient.

Experiments have been carried out for payoff matrices with $n = m = 2, 3, \ldots, 7$, with each experiment using random values for A and B and repeated 1000 times. As a trivial

initial complementary bases, the identity columns corresponding to variables $(\mathbf{s}, \mathbf{v})$ have been selected.

Payoff matrix size	**Local solution (obj > 0.001)**	**Close to zero (obj < 0.001)**	**Local solution (obj > 0.01)**	**Close to zero (obj < 0.01)**
2x2	563	437	516	484
3x3	768	232	684	316
4x4	908	92	801	199
5x5	931	69	827	173
6x6	960	40	855	145
7x7	973	27	871	129

Table 6.2: Numerical experiments on random bimatrix games.

Although the number of successful solves where the objective of the original quadratic problem is zero diminishes, the algorithm managed to find solution in a reasonable portion of the problems. Computational results are summarized on Table 6.2, where the optimal objective function value, due to numerical computational errors, claimed to be optimal in the interval $[0, \varepsilon)$. In the case of column two of the Table 6.2, $\varepsilon = 0.001$, while in column three $\varepsilon = 0.01$. This shows that the computational precision may influence which problem will be declared as solved.

6.4 Summary

After summarizing the theoretical results of [20] which presented a criss–cross method using $\mathbf{s}$-monotone index selection rules and has been extended in the sense of EP theorems, we have presented numerical experiments that make use of the flexibility of the rules by increasing the chance of finding feasible complementary basis even in the lack of sufficiency by selecting the pivots randomly when the rank in $\mathbf{s}$ does not uniquely define the pivot position, when solving general linear complementarity problems arising from bimatrix games and the Arrow-Debreu market equilibrium problem, where the algorithm has proved to be applicable in small dimensions.

The numerical experiments of this chapter have been published as part of [20].

Chapter 7

The primal quadratic simplex method using index selection rules

In this chapter, we prove the finiteness of the quadratic simplex method when applied to the linearly constrained convex quadratic optimization problem, and when ties are resolved using anti-cycling index selection rules. The original quadratic simplex method was developed by Wolfe and Panne and Whinston, and was published in several papers in the 1960s. In the original presentation, finiteness was ensured by the means of perturbation.

We show that for the quadratic simplex method to cycle, it is necessary that it is degenerate (i.e. there are basis in which all variables in the basis taking part in the ratio test has a primal value of zero), and that in the Karush-Kuhn-Tucker system associated with the problem, all components in the transformed column that correspond to the quadratic objective are zero. It follows from our proof that the quadratic simplex method is finite with the application of all those index selection rules, that only rely on the sign structure of the transformed right-hand-side and the reduced costs, and for which the tradition linear programming primal simplex method is finite.

The results of this chapter are novel contributions that have been published in [33] and reported in English in [35].

7.1 Finiteness of the quadratic primal simplex method

There were several publications related to the quadratic primal simplex method presented in Figures 3.6 at the beginning of the 1960s [48, 69, 68, 67, 70, 72]. Originally, the finiteness of the algorithm was ensured by the means of perturbation. We prove the finiteness of the algorithm with the help of anti-cycling index selection rules.

The **s**-monotone index selection rules have been used to show the finiteness of the criss–cross algorithm for convex quadratic optimization problems (applied to their associated linear complementarity problem) [4, 17, 19].

A key observation in terms of our results, is that the **s**-monotone index selection rules, in the case of a tie (not unique selection of driving variable or ratio test) the rule selects according to a well defined preference vector, and does not rely on the actual values of the pivot tableau.

Our proof depends on the analysis of the properties of the algorithm, and the tableau of the linear complementarity problem associated with the quadratic programming problem. In general, unfortunately the bi-symmetry of the pivot tableau is not preserved directly, though with a small extension a similar property can be proven.

Lemma 7.1.1 *[63] Starting from a bisymmetric matrix corresponding to a quadratic programming problem, any complementary pivot tableau that has been transformed from the original tableau remains bi-symmetric, with the exception that in the intersection of the original primal variables and the dual variables, instead of a zero matrix we have a positive semi-definite matrix.*

The exception part of the above theorem can be avoided, if we use the symmetric formulation of the quadratic problem [41].

We prove finiteness by the means of deducing it to a known method. We first show, that for a cycling example, all pivots are fully primal degenerate, so the primal solution no longer changes. Intuitively, this means that the linearized problem corresponding to the current solution no longer changes. We show that in such cases, the pivots carried out by the algorithm can be matched on a one to one basis to the pivots of an appropriately selected linear programming problem, on which the primal simplex method would carry out the exact same pivots when the same index selection rule that only relies on the sign structure of the transformed right hand side and reduced costs [19] is applied: as a consequence, a cycling example for the quadratic primal simplex would indicate cycling of the primal simplex method on the appropriately selected linear programming problem yielding a contradiction. Our proof does not generalize in an immediate way to lexicography (to our best knowledge, the finiteness of the quadratic simplex method using lexicography is not published).

Let us assume that the algorithm is not finite, and consider a cycling (counter) example. Since the proof is based on deduction to a known case, it is not strictly necessary to assume the minimality of the example, and it would not significantly simplify the reasoning.

We first show, that in the case of a cycling counter example, the algorithm (after a while) only carries out a single type of loop, which are loops of length two.

Lemma 7.1.2 *Let us consider the convex quadratic programming problem given with the associated linear complementarity problem. In the case of any cycling example, the quadratic primal simplex method can only carry out a finite number of loops of length one.*

Proof. From the statement of the quadratic primal simplex algorithm, we know that loops of length one correspond to a single pivot carried out in the row of the driving variable, and that in this case $0 < \theta_1 \leq \theta_2$ holds. Since we have selected the driving variable such that it transformed right hand side value is negative, we know that this row and the pivot operation is non-degenerate. In such cases, the transformed column of the incoming variable defines a strictly improving direction, and the value of the original quadratic objective function improves [68]. As the objective strictly decreases in the case of one long loops, and the objective of the problem is monotone during the quadratic primal simplex method [68] and the number of possible base is finite, we have proven that there can only be a limited number of loops of length one. ■

Now we consider the case of loops longer than two.

Lemma 7.1.3 *Consider the convex quadratic programming problem given by its associated linear complementarity problem. Then the quadratic primal simplex method, for any cycling example, can only carry out a finite number of loops which have a length different from two.*

Proof. According to Lemma 7.1.2 there can only be a finite number of loops of length one. Notice that the number of primal variables in the basis can only ever increase in the case of loops of length one. For loops of lengths other then one, after the first pivot, up until when we do a loop closing pivot in the row of the driving dual variable, each pivot in between a dual variable enters the basis, while a primal leaves. In other words if a loop is of length 3 or longer, then the number of dual variables in the basis increases monotonically. As their number could only decrease with loops of length one, it is necessary that the number of loops of length 3 or longer is finite. ■

To summarize, we can state that for any cycling counter example, after a finite number of pivot operations, the algorithm carries out an infinite sequence of loops of length two.

The question naturally arises if it would not be logical to deduce the proof back to the finiteness of the criss–cross algorithm instead, as the loops of length two correspond to the case of exchange pivots [41, 4, 17]. A potential difficulty is that the quadratic primal

simplex, once a driving dual variable (i.e. a row of the pivot tableau) is selected, the selection of the incoming column is pre-determined (the corresponding primal pair of the driving dual variable), so the second index selection step of the criss–cross algorithm is omitted. However, we do have the two index selections: the driving variable (dual variable) and then the ratio-test (primal variable), but the deduction would be more complicated.

Lemma 7.1.4 *Consider the convex quadratic programming problem in the form of* $(B - LCP)$*. In the case of a cycling example, after a finite number of pivots, the primal part of the basis-solution no longer changes.*

Proof. Using lemmas 7.1.2 and 7.1.3 we can assume that the algorithm only carries out loops of length two on the cycling example.

In the case of a loop of length two, the first pivot of the loop would improve the value of the original objective if the pivot was non-degenerate [68].

For the second pivot, more properties hold; using the results of [63], we know that for the second pivots of a loop of length two, at the intersection of the column of the dual variable entering the basis, and row from which the primal variable has left the basis in the previous iteration, the value of the pivot tableau coefficient t_{ij} is non-positive, and if it is negative then the pivot improves the value of the original quadratic objective – meaning that this case can only occur for a finite number of iterations – or that if it equals zero then in the transformed pivot tableau column of the incoming dual variable all coefficient that correspond to primal variables in the basis are zero. In effect, this means that those parts of the pivot tableau no longer change. ■

Using the results of the proof above [63], we can prove the following stronger results:

Lemma 7.1.5 *Consider the convex quadratic programming problem in the corresponding* $(B - LCP)$ *form. On a cycling example, after a finite number of pivot iterations, the quadratic primal simplex method only carries out loops of length two, for which the following holds:*

- *The first pivot of the loop is a degenerate pivot, during which a primal variable enters the basis while another primal variable leaves it.*
- *In the second, final pivot of the loop, the dual variable pair of the primal variable that entered in the previous iteration leaves the basis, while the dual pair of the primal variable that has left the basis in the previous iteration enters the basis.*
- *In the second pivot of the loop, in the transformed column of the entering dual variable, all coefficients in the rows corresponding to primal variables in the basis are zero.*

Proof. It follows from Lemmas 7.1.1. – 7.1.5, and similarly to the proof of Lemma 7.1.5 from the results of [63]. ■

Similar properties can be proven for the dual part of the first pivot of the loop as well.

Lemma 7.1.6 *Consider the convex quadratic programming problem in its corresponding* $(B - LCP)$ *form. On a cyclic example, after a finite number of pivots, in the quadratic primal simplex algorithm, when stating a loop from a complementary tableau, in the transformed tableau of the entering primal variable, the coefficients in the rows that correspond to dual variables in basis are zero.*

Proof. Using lemma 7.1.1, as the intersection of the row of the selected driving dual variable and its incoming primal pair is a zero valued diagonal element of a positive semi-definite matrix, it follows that the row and column of this positive semi-definite matrix has all zero coefficients as otherwise it would not be positive semi-definite, as the 2×2 principal submatrix defined by these positions would have a negative determinant. ■

We have already proven with the previous lemmas, that in the case of a cycling example, the transformed columns of the variables moving are all zeros: in the rows for which there is a dual variable pivoted into that row if it is the first pivot of the 2 long loop, and in the rows where there are primal variables pivoted into the basis if it is the second pivot the 2 long loop. The structure of these pivot tableaus are shown on Figure 7.1.

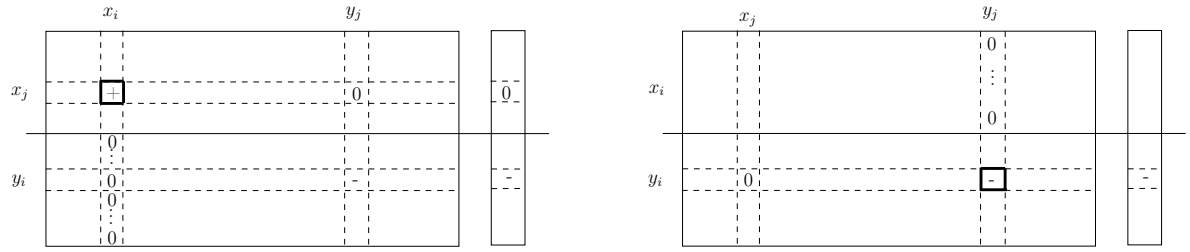


Figure 7.1: The structure of the pivot tableau for the first and second pivots for a loop of lengths two of a cycling example after cycling has already started.

Consider a cycling example, and according to Lemmas 7.1.2. and 7.1.3 assume that the quadratic primal simplex algorithm now only performs degenerate loops of length 2. For any complementary basis, let us denote by $\mathcal{I}_B^p$ and $\mathcal{I}_B^d$ the primal and dual variables in the basis respectively. Similarly, let $\mathcal{I}_N^p$ and $\mathcal{I}_N^d$ denote the corresponding primal and dual variables outside the basis.

Let $G = \bar{M}_{\mathcal{I}_B^p \mathcal{I}_N^p}$ the matrix defined by index sets $\mathcal{I}_B^p$ and $\mathcal{I}_N^p$, while $\mathbf{d} = \bar{\mathbf{q}}_{\mathcal{I}_B^p}$ is the right hand side corresponding to index set $\mathcal{I}_B^p$, and further $\mathbf{f} = \bar{\mathbf{q}}_{\mathcal{I}_B^d}$ is the right hand side corresponding to index set $\mathcal{I}_B^d$.

According to Lemma 7.1.1, $G = \bar{M}_{\mathcal{I}_B^p \mathcal{I}_N^p} = -\bar{M}_{\mathcal{I}_N^d \mathcal{I}_B^d}$. It can be seen that the $\bar{M}$ transformed basis tableau is the same as the Karush-Kuhn-Tucker conditions for problem (LP_{sub}):

$$\begin{aligned} &\min \mathbf{f}^T \mathbf{x} \\ &G\mathbf{x} \leq \mathbf{d} \end{aligned} \tag{LP_{sub}}$$

when we disregard the part of the tableau that does not play any role in the cycling.

Using Lemmas 7.1.6., 7.1.5. and 7.1.1, we see that the pivot tableau transforms the same way as the linear complementarity problem's tableau for the loops of length two. Moreover, the selection of the driving dual variable refers to the column selection according to the reduced costs, while the ratio test over the primal variables correspond to the primal ratio test of the smaller linear complementarity problem.

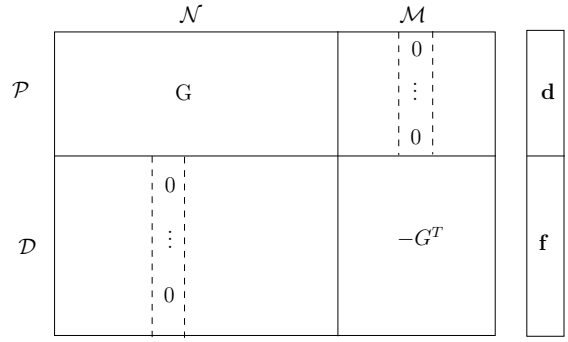


Figure 7.2: The structure of the pivot tableau for the quadratic programming problem in the case of the variables taking part of a cycling example.

So the quadratic primal simplex method working on the $(B - LCP)$ problem can cycle if and only if the corresponding primal simplex method cycles on the (LP_s) linear programming problem.

We have already seen that the primal simplex method can not cycle if such an index selection rule is applied for ensuring finiteness that is **s**-monotone (e.g. minimal index, LIFO or the MOSV rule) [19]. Thus we have proven the following:

Theorem 7.1.1 *The convex quadratic simplex method working on the corresponding linear complementarity problem is finite when s-monotone index selection rules are applied.*

7.2 Summary

We have shown that the quadratic primal simplex algorithm is finite when s-monotone index selection rules are applied. This result can straightforwardly be applied to the dual version of the quadratic simplex method. The proof is general, and is applicable to other index selection rules as well, for ones that only rely on the sign structure of the transformed right hand side and reduced costs when applied to a linear programming problem.

The results of this chapter are novel, and have been published in [33] and reported in English in [35].

Summary

The thesis is organized around flexible index selection rules for the linear feasibility, the linear programming, the linear complementarity, and the linearly constrained convex quadratic programming problems. The analyses of flexible index selection rules are brought into a common framework by applying the concept of **s**-monotone index selection rules [17, 19].

The thesis contains contributions to both the theory, and the computational aspects of **s**-monotone index selection rules; for algorithms for which finiteness proof already existed, the thesis presents numerical studies, while for some pivot algorithms it presents novel proofs of finiteness.

The traditional criss–cross method for the linear programming problem is shown to be finite when **s**-monotone index selection rules are used [54]. For the linear complementarity version of the criss–cross method, the thesis presents a slightly more general version of the proof for the finiteness when **s**-monotone rules are applied: the proof does not require the minimality of the cycling example; this version fits the numerical experiment [20] better.

For the primal simplex method for linearly constrained convex quadratic programming problems finiteness is proven when **s**-monotone index selection rules are used [33, 35]. The proof is rather general: it shows that any index selection rule that only relies on the sign structure of the reduced costs / transformed right hand side vector and for which the traditional primal simplex method is finite is necessary finite as well for the primal simplex method for linearly constrained convex quadratic programming problems. To the best knowledge of the author, finiteness of the primal simplex method for linearly constrained convex quadratic programming problems has only been published before using perturbation arguments.

The thesis presents a computational study for the effectiveness of flexible index selection rules. For the linear complementarity problems, a Matlab implementation of the EP-theorem like generalized LCP criss–cross algorithm is used to solve linear complementarity problems arising in market equilibrium and bimatrix game problems [20]; these problems yield matrices for the linear complementarity problem formulation for which the matrix is not sufficient.

Comparing pivot algorithms and flexible index selection rules is a non-trivial task as implementation details can easily bias the result dramatically. To address the issue, the thesis proposes a framework [34] that aims to minimize the effect of any algorithmic specific feature when comparing the effectiveness of pivot algorithms and flexible index selection rules. Two implementations are used for the numerical study: one in Matlab used to solve smaller instances, and one using the linear algebra of Xpress that is used to solve a large set of problems from public benchmark sets; the computational effectiveness of the flexible index selection rules is demonstrated by both experiment sets [34].

Összefoglalás

A tézis a lineáris megengedettségi feladatok, a lineáris programozási feladatok, a lineáris komplementaritási feladatok, illetve a lineáris feltételes konvex kvadratikus programozási feladatok rugalmas indexválasztási szabályainak az elméletét és gyakorlatát vizsgálja, az analízist a [17, 19]-ben bevezetett **s**-monoton indexválasztási szabályok köré építve.

A tézis egyaránt tartalmaz numerikus, illetve új elméleti eredményt. Azon algoritmusok esetén melyre már volt ismert végességi eredmény az **s**-monoton indexválasztási szabályok alkalmazása mellett a tézis numerikus eredményekkel támasztja alá azok gyakorlati fontosságát, míg bizonyítja a végességet némely pivot módszerre melyre korábban nem volt ismert a végesség az **s**-monoton szabályok alkalmazása esetén.

A hagyományos lineáris programozásbeli criss–cross módszer végessége közvetlen módon az első alkalommal kerül bizonyításra **s**-monoton szabályok alkalmazása mellett. A lineáris komplementaritási változat esetén a végesség a korábbiaknál egy árnyalattal általánosabb módon kerül bizonyításra, melyben a vizsgált feladatok minimalitása nincs feltéve. Az így nyert bizonyításokból származtatott módszer közvetlenebb módon alkalmazható a [20]-ban bemutatott numerikus eredmények során.

A lineáris feltételes konvex kvadratikus programozási feladatra a tézis a végességet **s**-monoton indexválasztási szabályok alkalmazása mellett bizonyítja [33]. A visszavezetésen alapuló bizonyítás egy általános eredmény mely azt mutatja meg, hogy tetszőleges olyan indexválasztási szabály, mely véges a hagyományos lineáris programozásbeli primál szimplex algoritmusra nézve és kizárólag a transzformált jobboldal és redukált költségek előjelszerkezetén alapul, az szükségképpen véges a lineáris feltételes konvex kvadratikus feladat esetén is. A szerző legjobb tudomása szerint ez az első publikált eredmény, mely ezen algoritmus végességét nem perturbációs megfontolásokkal bizonyítja.

A tézis számos, a flexibilis indexválasztási szabályokra vonatkozó numerikus hatékonysági elemzést mutat be. A lineáris komplementaritási feladatokra az EP tételek szellemében általánosított criss–cross módszert vizsgálja piaci egyensúlyi és bimátrix játékok megoldása során felmerülő LCP feladatok megoldásának segítségével [20]; ezen feladatok mátrixa nem elégséges, így alkalmasak az általánosított criss–cross módszer vizsgálatára.

Indexválasztási szabályok numerikus összehasonlítása nem nyilvánvaló probléma: az implementálási részletek drasztikus mértékben befolyásolhatják az eredményeket. A tézis javaslatot tesz összehasonlítási elvekre [34], melyek célja a megvalósítási részletekből adódó hiba mértékének a minimalizálása. A konkrét összehasonlítást két implementáció segítségével mutatja be: kisebb feladatokra egy Matlab implementáció, nagyobb feladatokra pedig az Xpress megoldó lineáris algebrájára épülő implementációval demonstrálja a flexibilis indexválasztási szabályok numerikus hatékonyságát [34].

Bibliography

[1] Miplib database. Available at http://miplib.zib.de/.

[2] Netlib database. Available at http://www.netlib.org/lp/data/.

[3] Matlab the language of technical computing, 2008. Version 7.6.0.324 (R2008a).

[4] A. A. Akkeleş, L. Balogh, and T. Illés. A véges criss-cross módszer új variánsai biszimmetrikus lineáris komplementaritási feladatra. *Alkalmazott Matematikai Lapok*, 21:1–25, 2003.

[5] A. A. Akkeleş, L. Balogh, and T. Illés. New variants of the criss-cross method for linearly constrained convex quadratic programming. *European Journal of Operational Research*, 157(1):74–86, 2004.

[6] K. M. Anstreicher and T. Terlaky. A monotonic build-up simplex algorithm for linear programming. *Operations Research*, 42(3):556–561, 1994.

[7] K. J. Arrow and G. Debreu. Existence of an equilibrium for competitive economy. *Econometrica*, 22:265–290, 1954.

[8] F. Bilen, Zs. Csizmadia, and T. Illés. Anstreicher-terlaky típusú monoton szimplex algoritmusok megengedettségi feladatokra. *Alkalmazott Matematikai Lapok*, 24(1-2):163–185, 2007.

[9] F. Bilen, Zs. Csizmadia, and T. Illés. Anstreicher-terlaky type monotonic simplex algorithms for linear feasibility problems. *Optimization Methods and Software*, 22(4):679–695, 2007.

[10] R. G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2:103–107, 1977.

[11] K. Cameron and J. Edmonds. Existentially polytime theorems. In *Polyhedral combinatorics (Morristown, NJ, 1989)*, volume 1 of *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 83–100. Amer. Math. Soc., Providence, RI, 1990.

[12] V. Chvátal. *Linear programming*. A Series of Books in the Mathematical Sciences. W. H. Freeman and Company, New York, 1983.

[13] R. W. Cottle, J. S. Pang, and V. Venkateswaran. Sufficient matrices and the linear complementarity problem. *Linear Algebra and its Applications*, 114/115:231–249, 1989.

[14] R.W. Cottle and S.-M. Guu. Two characterizations of sufficient matrices. *Linear Algebra and its Applications*, 170:65–74, 1992.

[15] R.W. Cottle and S.-M. Guu. On a subclass of $\mathbf{p}_*$. *Linear Algebra and its Applications*, 223/224:325–335, 1995.

[16] R.W. Cottle, J.-S. Pang, and V. Venkateswaran. Sufficient matrices and the linear complementarity problem. *Linear Algebra and its Applications*, 114-115:231–249, 1989.

[17] Zs. Csizmadia. *New pivot based methods in linear optimization, and an application in petroleum industry*. PhD thesis, Eötvös Loránd University of Sciences, 2007. Available at www.cs.elte.hu/∼csisza.

[18] Zs. Csizmadia and T. Illés. New criss-cross type algorithms for linear complementarity problems with sufficient matrices. *Optimization Methods & Software*, 21(2):247–266, 2006.

[19] Zs. Csizmadia, T. Illés, and A. Nagy. The s-monotone index selection rules for pivot algorithms of linear programming. *European Journal of Operation Research*, 221(3):491–500, 2012.

[20] Zs. Csizmadia, T. Illés, and A. Nagy. The s-monotone index selection rule for criss-cross algorithms of linear complementarity problems. *Acta Universitatis Sapientiae Informatica*, 5(1):103–139, 2013.

[21] G. B. Dantzig. *Linear programming and extensions*. Princeton University Press, Princeton, N.J., 1963.

[22] E. de Klerk, C. Roos, and T. Terlaky. *Nemlineáris Optimalizálás*. Operaciokutats No. 5. Budapesti Kozgazdasagtudomanyi es Allamigazgatasi Egyetem, Operaciokutats Tanszek, Budapest, 2004.

[23] D. den Hertog, C. Roos, and T. Terlaky. The linear complementarity problem, sufficient matrices, and the criss-cross method. *Linear Algebra and its Applications*, 187:1–14, 1993.

[24] Gy. Farkas. Theorie der einfachen ungleichungen. *Journal für die Reine und Angewandte Mathematik*, 124:1–27, 1901.

[25] FICO. *Xpress-Optimizer Reference Manual*, 2009. Available at http://www.fico.com/en/products/fico-xpress-optimization-suite/.

[26] K. Fukuda, M. Namiki, and A. Tamura. EP theorems and linear complementarity problems. *Discrete Applied Mathematics. Combinatorial Algorithms, Optimization and Computer Science*, 84(1-3):107–119, 1998.

[27] K. Fukuda and T. Terlaky. Linear complementarity and oriented matroids. *Journal of the Operations Research Society of Japan*, 35(1):45–61, 1992.

[28] Bowman E. H. Assembly line balancing by linear programming. *Operation Research*, 8:385–389, 1960.

[29] IBM. *IBM Optimization Subroutine Library, Guide and reference*, third editon edition, 1991. Release 2, Third edition.

[30] T. Illés. Lineáris optimalizálás elmélete és pivot algoritmusai. Technical report, Operációkutatási Tanszék, Eötvös Loránd Tudományegyetem, 2013. Operations Research Report, 2013-03.

[31] T. Illés and K. Mészáros. A new and constructive proof of two basic results of linear programming. *Yugoslav Journal of Operations Research*, 11(1):15–30, 2001.

[32] T. Illés and A. Nagy. Lineáris programozási szoftverek tesztelése. In *Informatika a felsőoktatásban 2008: IF2008 Konferencia kötete*, 2007. Available at http://www.agr.unideb.hu/if2008/kiadvany/papers/F14.pdf.

[33] T. Illés and A. Nagy. A kvadratikus szimplex algoritmus végessége indexválasztási szabályok alkalmazása esetén. *Alkalmazott Matematikai Lapok*, 30:1–21, 2013.

[34] T. Illés and A. Nagy. Computational aspects of simplex and mbu-simplex algorithms using different anti-cycling pivot rules. *Optimization*, 63(1):49–66, 2014. Published online: 18 Jul 2013.

[35] T. Illés and A. Nagy. Finiteness of the quadratic primal simplex method when **s**-monotone index selection rules are applied. 2014. Operations Research Report, 2014-01.

[36] T. Illés and M. Nagy. Mizuno–todd–ye típusú prediktor–korrektor algoritmus elégséges mátrixú lineáris komplementaritási feladatokra. *Alkalmazott Matematikai Lapok*, 22:41–46, 2005.

[37] T. Illés, JM. Peng, C. Roos, and T. Terlaky. A strongly polynomial rounding procedure yielding a maximally complementary solution for p*(κ) linear complementarity problems. *SIAM Journal on Optimization*, 11:320–340, 2000.

[38] T. Illés, C. Roos, and T. Terlaky. Polynomial affine-scaling algorithms for p*(κ) linear complementary problems. *Lecture Notes in Economics and Mathematical Systems*, 452:119–137, 1997.

[39] T. Illés and T. Terlaky. Pivot versus interior point methods: pros and cons. *European Journal of Operational Research*, 140(2):170–190, 2002. O.R. for a united Europe (Budapest, 2000).

[40] E. Klafszky and T. Terlaky. The role of pivoting in proving some fundamental theorems of linear algebra. *Linear Algebra and its Applications*, 151:97–118, 1991.

[41] E. Klafszky and T. Terlaky. Some generalizations of the criss-cross method for quadratic programming. *Optimization*, 24(1-2):127–139, 1992.

[42] V. Klee and G. J. Minty. How good is the simplex algorithm? In *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin)*, pages 159–175. Academic Press, New York, 1972.

[43] M. Kojima, N. Megiddo, T. Noma, and A. Yoshise. *A unified approach to interior point algorithms for linear complementarity problems*, volume 538 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1991.

[44] M. Kojima, S. Mizuno, and A. Yoshise. *A primal-dual interior point algorithm for linear programming*. Progress in Mathematical Programming, Interior-Point and Related Methods. Springer, New York, 1988. N. Megiddo, ed.

[45] M. Kojima, S. Mizuno, and A. Yoshise. A polynomial-time algorithm for a class of linear complementarity problems. *Mathematical Programming*, 44:1–26, 1989.

[46] T. Koltai and V. Tatay. Formulation of simple workforce skill constraints in assembly line balancing models. *Periodica Polytechnica, Social and Management Sciences*, 19/1:43–50, 2011.

[47] C. E. Lemke and J. T. Howson, Jr. Equilibrium points of bimatrix games. *J. Soc. Indust. Appl. Math.*, 12:413–423, 1964.

[48] C. E. Lemke and J. T. Howson, Jr. On complementary pivot theory. In *Mathematics of decision sciences*, volume Part 1, pages 95–114. AMS, Providence, Rhode Island, 1968.

[49] I. Maros. *Computational techniques of the simplex method*. International Series in Operations Research & Management Science, 61. Kluwer Academic Publishers, Boston, MA, 2003. With a foreword by András Prékopa.

[50] H. Mittelmann. Mittelmann test set. Available at http://plato.asu.edu/bench.html.

[51] K. G. Murty. *Linear and combinatorial programming*. Robert E. Krieger Publishing Co. Inc., Melbourne, FL, 1985. Reprint of the 1976 original.

[52] A. Nagy. Source code of the implementations. Available at http://bolyai.cs.elte.hu/opres/orr/SourceCodes.htm.

[53] A. Nagy. új típusú pivot módszerek numerikus összehasonlítása a lineáris programozásban. Master's thesis, Eötvös Loránd University of Sciences, 2007. Available at http://parad.web.elte.hu/Magamrol/NagyADiplomamunka.pdf.

[54] A. Nagy. Finiteness of the criss-cross method for the linear programming problem when **s**-monotone index selection rules are applied. 2014. Operations Research Report, 2014-02.

[55] A. Nagy and Sz. Takács. A létszámkeret-optimalizálás egy modellje. *Munkaügyi Szemle Online*, 2:98–104, 2014.

[56] M. Nagy. *Interior point algorithms for general linear complementarity problems*. PhD thesis, Eötvös Loránd University of Sciences, 2009.

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

[58] Zhang. S. A new variant of criss-cross pivot algorithm for linear programming. *European Journal of Operational Research*, 116(3):607–614, 1997.

[59] T. Terlaky. Egy új, véges criss-cross módszer lineáris programozási feladatok megoldására. *Alkalmazott Matematikai Lapok*, 10(3-4):289–296, 1983.

[60] T. Terlaky. A convergent criss-cross method. *Optimization*, 16(5):683–690, 1985.

[61] T. Terlaky. A new algorithm for quadratic programming. *European Journal of Operational Research*, 32(2):294–301, 1987.

[62] T. Terlaky and S. Z. Zhang. Pivot rules for linear programming: a survey on recent theoretical developments. *Annals of Operations Research*, 46/47(1-4):203–233, 1993. Degeneracy in optimization problems.

[63] A. W. Tucker. Principal pivotal transformations of square matrices. *SIAM Review*, 305, 1963.

[64] H. Väliaho. A new proof for the criss-cross method for quadratic programming. *Optimization*, 25(4):391–400, 1992.

[65] H. Väliaho. $\mathbf{P}_*$-matrices are just sufficient. *Linear Algebra and its Applications*, 239:103–108, 1996.

[66] H. Väliaho. Criteria for sufficient matrices. *Linear Algebra and its Applications*, 233:109–129, 1996.

[67] C. van de Panne and A. Whinston. A parametric simplicial formulation of houthakker's capacity method. *Operational Research Quarterly*, 15:355–388, 1964.

[68] C. van de Panne and A. Whinston. Simplicial methods for quadratic programming. *Naval Research Logistics*, 11:273–302, 1964.

[69] C. van de Panne and A. Whinston. A parametric simplicial formulation of houthakker's capacity method. *Econometrica*, 34(2):354–380, 1966.

[70] C. van de Panne and A. Whinston. The symmetric formulation of the simplex method for quadratic programming. *Econometrica*, 37(3):507–527, 1969.

[71] L. Walras. Elements of pure economics, or the theory of social wealth, 1874. 1899, 4th ed.; 1926, rev. ed., 1954, Engl. Transl.

[72] P. Wolfe. The simplex method for quadratic programming. *Econometrica*, 27(3):382–398, 1959.

[73] Y. Ye. A path to the arrow-debreu competitive market equilibrium. *Math Programming*, 111(1/2):315–348, 2008.

[74] S. Zionts. The criss-cross method for solving linear programming problems. *Management Science*, 15(7):426–445, 1969.