Advances in Interior Point Methods for Large-Scale Linear Programming

Marco Colombo

Doctor of Philosophy
University of Edinburgh
2007

Se'n foi me de chesta brögna ché?

If nothing else it true, well this is
If nothing else meant anything
The silence ended and I forgot...

I wish that I could find it, I wish I'd let it go
I wish my arms could hold it, will I ever know?
It may or may not happen...

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Marco Colombo*)

Abstract

This research studies two computational techniques that improve the practical performance of existing implementations of interior point methods for linear programming. Both are based on the concept of symmetric neighbourhood as the driving tool for the analysis of the good performance of some practical algorithms.

The symmetric neighbourhood adds explicit upper bounds on the complementarity pairs, besides the lower bound already present in the common $\mathcal{N}_{-\infty}$ neighbourhood. This allows the algorithm to keep under control the spread among complementarity pairs and reduce it with the barrier parameter μ. We show that a long-step feasible algorithm based on this neighbourhood is globally convergent and converges in $\mathcal{O}(nL)$ iterations.

The use of the symmetric neighbourhood and the recent theoretical understanding of the behaviour of Mehrotra's corrector direction motivate the introduction of a weighting mechanism that can be applied to any corrector direction, whether originating from Mehrotra's predictor–corrector algorithm or as part of the multiple centrality correctors technique. Such modification in the way a correction is applied aims to ensure that any computed search direction can positively contribute to a successful iteration by increasing the overall stepsize, thus avoiding the case that a corrector is rejected. The usefulness of the weighting strategy is documented through complete numerical experiments on various sets of publicly available test problems. The implementation within the HOPDM interior point code shows remarkable time savings for large-scale linear programming problems.

The second technique develops an efficient way of constructing a starting point for structured large-scale stochastic linear programs. We generate a computationally viable warm-start point by solving to low accuracy a stochastic problem of much smaller dimension. The reduced problem is the deterministic equivalent program corresponding to an event tree composed of a restricted number of scenarios. The solution to the reduced problem is then expanded to the size of the problem instance, and used to initialise the interior point algorithm. We present theoretical conditions that the warm-start iterate has to satisfy in order to be successful. We implemented this technique in both the HOPDM and the OOPS frameworks, and its performance is verified through a series of tests on problem instances coming from various stochastic programming sources.

Acknowledgements

I would like to thank Prof. Jacek Gondzio for proposing such an interesting and challenging project. He shared his enthusiasm for interior point methods, and instilled in me a very keen interest in working in this field.

Andreas Grothey's contribution to the success of this research cannot be underestimated. He has had the patience of introducing me to OOPS, and helped me out in the frequent occasions of panic.

I am particularly indebted to Andrea, who has been close to me through most of this time, and has experienced the ups and downs of my PhD life. She is one of my best friends, and she taught me so much!

Many friends deserve to be mentioned here as well, for the time spent together, the conversations, and the help that I received in many forms from them. My thoughts go especially to Aghlab, Gaurav, Iria, Paulo, Eleni and Giorgos.

It's been fun sharing room 6320 with Cathy, David, Elingborg, Rosemary and Tomas. They have been great office mates and friends. Cathy has also proofread this thesis, and she has done a fantastic job. It was great to see her corrections marked in green, they had such a positive feeling!

I got lots of support from my family, in particular in the last few months. It's been really helpful feeling cheered and spurred by them. This work is dedicated to all of them. T'et capì!

Table of Contents

Chapter 1

Background and introduction.

In this chapter we present the background and scope of this research. In particular we introduce the field of linear programming, discuss its relevance, and compare some solution methods, specifically with regard to their computational complexity. Further, we motivate our research and provide an outline of the chapters of this thesis.

1.1 Linear programming

Linear programming is a relatively new discipline in the mathematical spectrum. It was developed as mathematical models were being introduced for economic and military planning in the years immediately following the end of World War II. The realisation of its usefulness came simultaneously with the development of a solution method, the simplex method. The introduction of the first computer calculators was crucial to the blossoming and increase of this newly born area of study. Historical accounts of the birth and development of linear programming can be drawn from many sources, such as [21, Chapter 2] and [82]. Dantzig's personal recollections are also in [22].

A broad definition of linear programming has been given by Dantzig [22]:

> "*Linear programming can be viewed as part of the great revolutionary development which has given mankind the ability to state general goals and to lay out a path of detailed decisions to take in order to "best" achieve its goals when faced with practical situations of great complexity.*"

Further, Dantzig [22] mentions the essential components of linear programming:

> "*Our tools for doing this are ways to formulate real-world problems in detailed mathematical terms (models), techniques for solving the models (algorithms), and engines for executing the steps of algorithms (computers and software).*"

An *optimization problem* can be described in terms of decision variables, a set of constraints, and an objective function. The investigation of optimization problems stems from the natural desire to solve a problem in the "best possible way". It is interesting to note that while the need for an objective function is obvious now, it was not clear when the first problems were modelled: the set of feasible solution used to be investigated with some ad-hoc criteria, instead of being guided by the optimization of some quantity [22].

A *linear programming problem* is an optimization problem in which the objective function and constraints are linear. Linear programming problems arise directly from real-life applications (for example in economics, finance, logistics, and other areas), or as approximations to more complicated formulations, as most real-life relationships are nonlinear. Another important source of linear programs is the continuous relaxation of integer programming problems [82].

Among the class of convex optimization problems, linear programming has a peculiar feature which is described by the following Theorem.

Theorem 1.1 (Fundamental theorem of linear programming)**.** *For a linear programming problem with a feasible domain $\mathcal{P}$ containing at least one extreme point, the optimal objective value is either unbounded or is achievable at one extreme point of $\mathcal{P}$.*

The set of linear constraints defines a *polyhedron* that constitutes the *feasible region*. According to Theorem 1.1, in looking for a solution we can restrict our attention to the vertices of this polyhedron. The polyhedron corresponding to a linear system of m constraints in n variables ($m < n$) has a number of vertices equal to

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} \geq \left(\frac{n}{m}\right)^m. \tag{1.1}$$

This number is an overestimate, as not all of these choices correspond to feasible points.

The fact that the number of vertices is finite guarantees termination of any algorithm that explores all vertices. However this number is exponential, as can be clearly seen by further manipulating (1.1):

$$\left(\frac{n}{m}\right)^m \geq 2^m \quad \text{for } n \geq 2m.$$

This observation gives rise to the need of defining an algorithm that uses an intelligent way to discover an optimal vertex among the multitude of non-optimal ones.

An important feature of any algorithm is its efficiency, that is how much effort is needed for the algorithm to provide an answer for some given input. The

concept of *computational complexity* was introduced in the 70s, as the greater availability of computing machines required a deeper insight on the computational performance of different algorithms. The computational complexity of an algorithm can be used as a measure of the growth in the computational effort as a function of the size of the problem. Therefore, it provides a worst-case measure. The formal notion of *efficiency* is that a problem has an algorithm running in time proportional to a polynomial function of its input size. That is, we consider an algorithm efficient if it runs in time $\mathcal{O}(n^k)$ on any input of size n, for some constant $k > 0$. An exhaustive presentation of this topic exceeds the aims of this thesis. We refer the reader to available introductions in the area, [82, Chapter 2] among others.

Complexity proofs rely on two assumptions that are necessary simplifications:

1. Computations are performed in exact arithmetic;

2. The numerical data of a problem instance is rational.

Computational complexity is measured by the number of elementary operations required to perform the algorithmic steps until termination. It often depends on the size of the binary representation of the input, usually denoted by L.

Many algorithms have been proposed for solving a variety of optimization problems. However, despite their diversity, they are based on the same general framework which is summarised in the steps of Algorithm 1.1.

Algorithm 1.1 Generic optimization algorithm

Given: An initial iterate w;

Repeat:

Determine a search direction Δw.

Compute the distance α of how far to move along the search direction.

Move to the next point $w + \alpha\Delta w$.

Until Some termination criteria are met.

Each element of this generic framework (starting point, search direction, step-size, termination criteria) has to be accurately specified in order to define a particular algorithm. In what follows, we will introduce the main ideas behind three different solution methods for linear programming: the simplex method, the ellipsoid method, and the class of interior point methods.

1.1.1 The simplex method

The *simplex method* was introduced in 1947 by George Dantzig [21]. The introduction of the simplex method happened simultaneously with the realisation of

linear programming as an efficient modelling tool for practical decision making.

The simplex method exploits the insight provided by the fundamental theorem of linear programming (1.1), which states that the optimal solution, if it exists, is at one of the vertices of the feasible polytope. Thus it reaches a solution by visiting a sequence of vertices of the polyhedron, moving from each subsequent vertex to an adjacent one characterised by a better objective function value (in the non-degenerate case). Since the number of vertices is finite, termination is guaranteed.

Moreover, given the monotonic method of choosing the next vertex, the set of possible vertices decreases after each iteration, in the non-degenerate case. Degeneracy occurs when a vertex in $\mathbb{R}^m$ is defined by $p > m$ constraints, and a step of length zero may be produced. In such a case, the simplex method does not actually move away from the current vertex, and thus no improvement in the objective function value can be achieved.

In terms of practical efficiency, the simplex algorithm has long been considered the undisputed method for solving linear programming problems. However, the simplex method has exponential complexity. It is possible that all the vertices of the feasible polyhedron have to be visited before an optimal solution is reached. Klee and Minty [53] were the first to provide an example of pathological behaviour of the simplex method. In their example, a linear program with n variables and $2n$ inequalities, the simplex method visits each of the 2^n vertices.

However, no cases of exponential number of iterations have been encountered in real-life problems, and usually only a fraction of the vertices are actually traversed before the optimal one is found. Moreover, in most cases the simplex algorithm shows polynomial behaviour, being linear in m and sublinear in n [28, p.94]. A survey on the efficiency of the simplex method is done by Shamir [83], where a probabilistic analysis (as opposed to worst-case analysis) is also presented.

The gap between the observed and theoretical worst-case performances of the simplex method is still unexplained. Given this theoretical drawback, a great deal of effort has been put into finding an algorithm for linear programming which is characterised by a polynomial-time bound.

1.1.2 The ellipsoid method

In 1979 a breakthrough occurred, as Khachiyan showed how to adapt the *ellipsoid method* for convex programming to the linear programming case, and determined the computational complexity of linear programming.

In Khachiyan's ellipsoid method, the feasible polyhedron is inscribed in a sequence of ellipsoids of decreasing size. The first ellipsoid has to be large enough

to include a feasible solution to the constraints; the volume of the successive ellipsoids shrinks geometrically. Therefore it generates improving iterates in the sense that the region in which the solution lies is reduced at each iteration in a monotonic fashion. The algorithm either finds a solution, as the centres of the ellipsoids converge to the optimal point, or states that no solution exists. More details on the ellipsoid method can be found in [82, Chapter 13] and [74, Chapter I.6], for example.

The exciting property of the ellipsoid method is that it finds a solution in $\mathcal{O}(n^2L)$ iterations, and thus has polynomial complexity. However, since the ellipsoid algorithm generally attains this worst-case bound [30], its practical performance is not competitive with other solution methods. Besides, it displays other drawbacks related to large round-off errors and a need for dense matrix computation. Nevertheless, the ellipsoid method is often used in the context of combinatorial optimization as an analytic tool to prove complexity results for algorithms [74].

1.1.3 Interior point methods

Interior point methods were being developed in the 60s and the beginning of the 70s as methods to solve nonlinear programming problems with inequality constraints. However, they fell from favour and received less and less attention because of their inefficiency and the presence of strong competitors such as sequential quadratic programming [89].

Since their reintroduction, this time to solve linear programs, following Karmarkar's groundbreaking paper [52], interior point methods have attracted the interest of a growing number of researchers. This algorithm was also proved to have polynomial complexity: indeed, it converges in $\mathcal{O}(nL)$ iterations. As opposed to Khachiyan's ellipsoid method, in practice Karmarkar's algorithm actually performs much better than its worst-case bound states.

The main idea behind interior point methods is fundamentally different to the one that inspires the simplex algorithm. Here, the optimal vertex is approached by moving through the interior of the feasible region. This is done by creating a family of parametrised approximate solutions that asymptotically converge to the exact solution. Therefore, by embedding the linear problem in a nonlinear context, an interior point method escapes the "curse of dimensionality" characteristic of dealing with the combinatorial features of the linear programming problem. For details on Karmarkar's algorithm, we refer to [28, Chapter 6].

Karmarkar [52] explained the advantage of an interior point approach as follows:

> "*In the simplex method, the current solution is modified by introducing a nonzero coefficient for one of the columns in the constraint matrix. Our method allows the current solution to be modified by introducing several columns at once.*"

Karmarkar announced that his method was extremely successful in practice, claiming to beat the simplex method by a large margin (50 times, as reported in [89]). A variant of Karmarkar's original algorithm was then proposed and implemented by Adler, Resende, Veiga and Karmarkar [1]. Since then, the theoretical understanding has considerably improved, many algorithmic variants have been proposed and several of them have shown to be computationally viable alternatives to the simplex method.

Over the last 20 years, an impressive wealth of theoretical research has been published, and computational developments have brought life to the field of linear programming, which had not seemed to attract much attention anymore. Among the positive consequences of the renewed interest in linear programming are the improvements to the implementations of simplex-based solvers [11, 12].

There are classes of problems that are best solved with the simplex method, and others for which an interior point method is preferred. Size, structure and sparsity play a major role in the choice of algorithm for computations. As a rule of thumb, with the increase of problem dimension, interior point methods become more effective. However, this does not hold in the hyper-sparse case, where the simplex method is virtually unbeatable [12, 43], and for network problems, where the specialised network simplex method can exploit the structure in an extremely efficient manner [74].

Interior point methods are well-suited to solving very large scale optimization problems. Their theory is well understood [80, 90, 91] and the techniques used in their implementation are documented in extensive literature (see, for example, [3, 38, 61] and the references therein). They can be applied to a wide range of situations with no need of major changes. In particular, they have been successfully applied to complementarity problems, quadratic programming, convex nonlinear programming, second-order cone programming and semidefinite programming.

1.2 Motivation and scope of this research

Optimization algorithms are extremely important in real-life applications. Theoretical advances are necessary for a deeper understanding of the the available techniques and the opening of new avenues of research. However, theory per se rarely has a direct impact on the lives of those who rely on optimization as a tool

to solve their problems. It is therefore necessary that the insight gathered from theoretical studies is then transformed into effective practical tools. This usually requires the implementation of computer programs with the aims of accuracy, speed, and reliability.

The process of creating computationally efficient methods from theoretical studies is not as direct as it might sound, but is somewhat an art in itself. It often involves relaxing many of the theoretical assumptions, while ensuring other properties and conditions. Therefore we will put great effort in accompanying theoretical results with the corresponding computational considerations. While in a few cases these can be treated simultaneously, generally that will not be possible. There are a few reasons for this:

- Theoretical assumptions may not be realistic. This is the case when a condition stated in a theorem is not realistically satisfiable in practice (for example, bounds on some quantities).
- Theoretical requirements may be computationally expensive. This happens when checking the satisfaction of a certain condition is not viable in practice.
- Theoretical assumptions may be too restrictive. This occurs when the theory predicts the conditions under which a certain behaviour happens, but in practice the same behaviour happens under more relaxed conditions.
- Theoretical analysis provides a worst-case result which may be very far from the average one.

On the other hand, the practical implementation of an optimization algorithm happens in a context that is not amenable to theoretical analysis for the following reasons:

- Finite precision of floating-point arithmetic;
- Dependence on the numerical inputs;
- Heuristic algorithmic choices.

For these reasons, we will take a very pragmatic view of the field of optimization. The approach that permeates this research and the presentation of its results is markedly computationally oriented. We recognise the importance and validity of theoretical developments, and we appreciate the insights that such results provide. Hence, we will discuss the theoretical foundations of interior point methods for linear programming and contribute to it. However, as we are keen on improving

existing computer implementations of interior point methods, our main drive will be the development of implementable techniques for interior-point-based linear programming solvers.

The contributions of this research are threefold.

First, we introduce and formalise the concept of *symmetric neighbourhood* as the driving tool for the analysis and understanding of the good behaviour of some practical algorithms. The practical success of an interior point code is negatively affected by the presence of unbalanced complementary products. In this sense, the quality of centrality (understood in a simplified way as complementarity) in a practical algorithm is not conveniently characterised by either of the two neighbourhoods $\mathcal{N}_2$ or $\mathcal{N}_{-\infty}$ (see Section 2.1.2 for their definition) which are usually employed in theoretical developments of interior point methods. The symmetric neighbourhood $\mathcal{N}_s$ is a variation on the more common $\mathcal{N}_{-\infty}$ neighbourhood in which the complementarity pairs are bounded above and below by a quantity that depends on the barrier parameter μ but does not depend directly on n. Within the $\mathcal{N}_s$ neighbourhood, the spread among complementarity pairs is kept under control and is reduced with the barrier parameter μ. This has beneficial consequences on the quality of the search directions generated by Newton's method. We show that a long-step feasible algorithm based on the symmetric neighbourhood is globally linearly convergent and has the same computational complexity as an algorithm based on the wide $\mathcal{N}_{-\infty}$ neighbourhood, as it converges in $\mathcal{O}(nL)$ iterations. Also, the symmetric neighbourhood plays a central role in the development of the two computational techniques that we introduce and study in this work.

The most significant use of the notion of symmetric neighbourhood is in the *weighted correctors technique*, which is our second contribution. The weighting mechanism, which is an adaptive technique to judge the most effective way of including a corrector in the search direction, works particularly well in conjunction with multiple centrality correctors. The main objective of this research is to analyse the efficiency of corrector directions in the light of the theoretical studies of Cartis [14, 15]. It concentrates on ensuring that a corrector direction computed at the current iterate is not rejected because it produces a short stepsize. Such behaviour is usually manifested when the point is badly centered or highly infeasible. For this reason, ensuring that an appropriate (centrality) corrector is accepted and produces an increase in the stepsize is crucial for moving the iterate to a more favourable position. From the computational point of view, we have to consider that a failed iteration comes with the high cost of having factorised the matrix for no real improvement. We argue that following a quadratic trajectory in generating a corrector direction is not always desirable. In particular, as the

stepsizes in the primal and dual spaces are often much smaller than 1, trying to correct for the whole linearisation error, as done in Mehrotra's second-order corrector, is too demanding and not always necessary. In generating a corrector direction, we propose to use the difference between the achievable complementarity products and a carefully chosen target vector of complementary products. Moreover, it is unrealistic to expect that a corrector direction $\Delta_c w$ can absorb the error made by a predictor direction $\Delta_p w$; however, this direction can be used to find a new composite direction $\Delta^\omega w = \Delta_p w + \omega \Delta_c w$, where $\omega \in [0, 1]$. We propose to measure the quality of the composite direction by the length of the step that can be made along it, and choose the ω that maximizes such step. The introduction of the weighting mechanism, which determines the contribution of each corrector direction to the overall search direction, allows a better implementation of the targeting procedure of the multiple centrality correctors technique with improved performance. We have implemented the weighted correctors strategy in the HOPDM interior point code, and have studied its performance with thorough computational experiments. From this testing we see that such a strategy produces considerable improvements, particularly for large-scale problems.

Our third contribution is in the context of the generation of *warm-start iterates for stochastic linear programs*, and the practical value of the symmetric neighbourhood is also appreciable here. In this setting we develop an efficient way of constructing an advanced starting point which greatly reduces the computational effort of solving a problem instance. Large-scale stochastic linear programs are very structured problems that are generated from event trees which can be very big, particularly in the multistage case. We show that it is possible to exploit the inherent structure of the stochastic programming problem and obtain a computationally viable warm-start point by solving a problem of much smaller dimension that corresponds to a reduced event tree. We produce a reduced tree by trying to capture the important information in the stochastic scenario space while keeping the dimension of the corresponding (reduced) deterministic equivalent small. It is essential for the effectiveness of an interior point algorithm that the iterates do not approach the boundary of the feasible region too early. This is particularly important in reoptimization, as the warm-start iterate should be able to quickly absorb perturbations in the problem data. This suggests solving the reduced problem with low accuracy. As the size of the reduced problem is generally very small, this phase of the algorithm is relatively inexpensive. Once an approximate reduced-tree solution is found, we then expand it to the full size of the original problem, thus generating the warm-start point. We present theoretical conditions that the warm-start iterate should satisfy to guarantee a

successful warm-start of the complete problem. We also show that from an iterate within the symmetric neighbourhood of the central path, we can produce a corrector direction that absorbs the perturbation and keeps the point inside a larger symmetric neighbourhood, and move along such direction with full step. The implementation within the HOPDM and OOPS interior point solvers shows remarkable advantages on a series of problem instances.

The original results presented in this thesis have been the basis for two papers that have been submitted for publication, jointly with Jacek Gondzio [18], and with Jacek Gondzio and Andreas Grothey [19].

Outline of the thesis

In Chapter 2 we introduce and formalise some basic notions on linear programming and present the main theoretical results that are at the heart of many interior-point methods. We concentrate on the path-following class of primal–dual interior point methods, as they provide the framework for most practical implementations. Alongside reviewing the theoretical achievements known in the literature, we introduce and analyse the concept of symmetric neighbourhood.

Chapter 3 is dedicated to presenting the main techniques adopted in computational implementations of interior point methods. We consider when theoretical analysis and practical implementation diverge, paying particular attention to two important strategies used in generating effective search directions: Mehrotra's predictor–corrector algorithm and the multiple centrality correctors technique. We also review the main results in the area of warm-start approaches for interior point methods.

In Chapter 4 we continue the discussion on Mehrotra's algorithm, presenting some recent insight on the cases of failure of the corrector direction, and studying the use of weighted correctors in the generation of search directions. This will be compared to the subspace searches approach, which uses an auxiliary linear subproblem to combine a given set of directions. The advantages and drawbacks of both strategies is presented together with a rich computational study on the implementation of the weighted correctors technique.

In Chapter 5 we turn our attention to the field of stochastic programming, explain its relevance and present some basic concepts. We introduce a warm-start technique that exploits the inherent structure of stochastic linear programs. Alongside some theoretical results, we show our computational experience on some standard test problems and larger instances coming from the telecommunication industry.

Finally, in Chapter 6 we present our conclusions and discuss directions for future work.

Chapter 2

Interior point methods for linear programming.

The theory at the heart of interior point methods is well understood, and a number of survey papers, monographs and academic books are available [38, 41, 79, 80, 86, 89, 90]. This chapter is devoted to the derivation and analysis of primal–dual path-following interior point methods. We present the elements that are at the basis of this successful class of algorithms, concentrating on their theoretical properties and attractive features. We also introduce and analyse the concept of a symmetric neighbourhood, and its consequences for practical algorithms.

2.1 Derivation of primal–dual methods

Consider the following primal–dual pair of linear programming problems in standard form

$$\begin{array}{llll} & \min\limits_{x} & c^T x & \\ (P) & \text{s.t.} & Ax = b, & \\ & & x \geq 0; & \end{array} \qquad\qquad \begin{array}{llll} & \max\limits_{(y,s)} & b^T y & \\ (D) & \text{s.t.} & A^T y + s = c, & \\ & & s \geq 0, & \end{array} \tag{2.1}$$

where $A \in \mathbb{R}^{m\times n}$, $x, s, c \in \mathbb{R}^n$ and $y, b \in \mathbb{R}^m$, $m < n$. We assume, without loss of generality, that A has full row rank, as linearly dependent rows can be removed without changing the solution set. This implies that a feasible $s \geq 0$ determines in a unique way the value of y. In fact, the y variables can be eliminated thus producing the symmetric combined primal–dual form studied by Todd and Ye [87].

We define the sets of primal feasible points and of dual feasible points as

$$\mathcal{P} = \{x : Ax = b,\ x \geq 0\}, \qquad \mathcal{D} = \{(y, s) : A^T y + s = c,\ s \geq 0\},$$

and, using this notation, we can rewrite the primal–dual pair (2.1) as

$$\min_{x}\ c^T x \quad \text{s.t.}\ \ x \in \mathcal{P}; \qquad \max_{(y,s)}\ b^T y \quad \text{s.t.}\ \ (y, s) \in \mathcal{D}.$$

In the derivation of interior point methods we will concentrate on two closely related sets: the set of primal interior points and the set of dual interior points

$$\mathcal{P}^0 = \{x \in \mathcal{P} : x > 0\}, \qquad \mathcal{D}^0 = \{(y, s) \in \mathcal{D} : s > 0\}.$$

We introduce the notation

$$w = (x, y, s)$$

to denote a primal–dual point, and we consider the set of feasible primal–dual points and the set of primal–dual interior points

$$\mathcal{F} = \mathcal{P} \times \mathcal{D}, \qquad \mathcal{F}^0 = \{w \in \mathcal{F} : (x, s) > 0\}.$$

A primal–dual point $w \in \mathcal{F}^0$ is said to be *strictly feasible* for the primal–dual pair (2.1).

We recall here some well-known results on the relationship between problems (P) and (D). These can be found in plenty of sources, for example [17, 82]. Our presentation is inspired by [42, 63, 90].

Lemma 2.1 (Weak duality)**.** *Let $w \in \mathcal{F}$. Then $c^T x \geq b^T y$.*

Proof. Since $x \in \mathcal{P}$ and $(y, s) \in \mathcal{D}$, the following holds:

$$c^T x - b^T y = c^T x - x^T A^T y = x^T (c - A^T y) = x^T s \geq 0. \qquad \square$$

The weak duality property states that the primal and dual objective values bound each other. The difference $c^T x - b^T y$ is called the *duality gap*. When the objectives in both problems achieve their respective bounds, that is, when the duality gap is zero, the primal–dual solution has to be optimal. This can be formalised in the following lemma.

Lemma 2.2 (Strong duality)**.** *A point $x \in \mathcal{P}$ is an optimal solution if and only if there exists a pair $(y, s) \in \mathcal{D}$ such that $c^T x = b^T y$.*

Problem (P) has a feasible solution if and only if $\mathcal{P} \neq \emptyset$. If $\mathcal{P} \neq \emptyset$ and $\mathcal{D} \neq \emptyset$, then both (P) and (D) admit an optimal solution $w^* = (x^*, y^*, s^*)$, and by Lemma 2.2 the objective function values of both problems coincide at that point. Otherwise, if one of the sets $\mathcal{P}$ or $\mathcal{D}$ is empty, then the other is either unbounded or empty as well. In such cases, an optimal solution for problem (2.1) does not exist.

Optimality conditions let us recognise that a solution has been found. They also provide insight on the development of algorithms for finding a solution. The

Karush-Kuhn-Tucker (KKT) conditions express first-order optimality conditions for the primal–dual pair (2.1). They can be written as

$$\begin{array}{rcl} Ax & = & b \\ A^T y + s & = & c \\ XSe & = & 0 \\ (x, s) & \geq & 0, \end{array} \tag{2.2}$$

where $X, S \in \mathbb{R}^{n \times n}$ are diagonal matrices with elements x_i and s_i respectively, and $e \in \mathbb{R}^n$ is a vector of ones. In other words, an optimal solution is characterised by primal feasibility, dual feasibility and complementarity.

Complementarity can be seen as a certificate for optimality in linear programming [47, 82]. For non-optimal feasible iterates, complementarity measures the distance of the iterate to optimality:

$$c^T x - b^T y = x^T s. \tag{2.3}$$

The quantity $x^T s$ is called the *complementarity gap*. When it is driven to zero, then a feasible solution is also optimal. We remark that the equality between the duality gap and the complementarity gap of equation (2.3) holds only for a feasible point.

In what follows, we make the standard assumption for the development of interior point methods that $\mathcal{P}^0 \neq \emptyset$ and $\mathcal{D}^0 \neq \emptyset$. This is also referred to as the *interior point assumption*. The interior point assumption corresponds to assuming that the primal–dual optimal face is bounded [42, Lemma 2.2]. Cases when this assumption does not hold can be considered by allowing the algorithm to accept infeasible iterates (see Section 2.2.2) or by introducing some controlled perturbations that enlarge the primal–dual feasible set [16].

2.1.1 The barrier problem

Many algorithms used in mathematical programming can be interpreted as path-following. Here we restrict our attention to the path described by the use of a logarithmic barrier function in linear programming. Given the linear program in standard form (P), it is possible to write the corresponding *barrier problem*:

$$(P_\mu) \quad \min_x \; c^T x - \mu \sum_{i=1}^{n} \ln x_i \quad \text{s.t} \;\; x \in \mathcal{P}^0.$$

Problem (P_μ) denotes a family of problems parameterised by the scalar quantity $\mu > 0$ (typically small), which is called the *barrier parameter* in the interior point literature.

The presence of the logarithmic barrier in the objective function of (P_μ) forces the iterates to stay in the interior of the feasible region, as this term heavily penalises points that are too close to the boundary. However, the influence exerted by the logarithmic barrier can be controlled through the penalty parameter μ. The weight on the barrier regulates the distance from the iterates to the boundary: as μ tends to zero, problem (P_μ) resembles problem (P) more and more closely. It is worth noting that such an approach is viable only if it is actually possible to find a point that strictly satisfies the constraints, that is, if $\mathcal{P}^0 \neq \emptyset$. If the feasible domain $\mathcal{P}$ is bounded, then both (P) and (P_μ) admit optimal solutions.

The objective function of problem (P_μ) is a strictly convex function. Therefore, for a fixed μ, the problem is either unbounded or has exactly one minimum. The minimizer, if it exists, is completely characterised by the associated KKT conditions:

$$\begin{aligned} Ax &= b \\ \mu X^{-1}e + A^T y &= c \\ x &> 0. \end{aligned}$$

By substituting $s = \mu X^{-1}e$, we obtain the standard (primal–dual) formulation of the so called *perturbed* KKT *conditions*:

$$\begin{aligned} Ax &= b \\ A^T y + s &= c \\ XSe &= \mu e \\ (x, s) &> 0. \end{aligned} \tag{2.4}$$

If the perturbed KKT conditions (2.4) have a solution for a particular $\hat{\mu} > 0$, then it has solution for every $\mu > 0$. System (2.4) determines a unique continuous smooth curve $w(\mu) = (x(\mu), y(\mu), s(\mu))$ toward the optimal set as $\mu \to 0$. In interior-point terminology, this curve is called the *central path*. The study of the primal–dual properties of the central path was pioneered by Megiddo [63] and Bayer and Lagarias [5].

Under the assumptions that for a particular $\mu > 0$ the point $w(\mu)$ is primal and dual feasible, we can state a similar result to the one expressed by (2.3), and define the duality gap $g(\mu)$ as a function of the barrier parameter:

$$g(\mu) = c^T x(\mu) - b^T y(\mu) = x(\mu)^T s(\mu). \tag{2.5}$$

That is, for any value of μ the duality gap corresponds to the complementarity gap. Hence reducing either of them is identical.

Moreover, as $XSe - \mu e = 0$ corresponds to $x_i s_i = \mu$, $i = 1, \ldots, n$, we have

$$x(\mu)^T s(\mu) = \sum_{i=1}^{n} x_i(\mu) s_i(\mu) = n\mu, \tag{2.6}$$

and for $\mu \to 0$, also $g(\mu) \to 0$. Equations (2.5) and (2.6), together with the fact that $c^T x(\mu) \geq c^T x^* = b^T y^* \geq b^T y(\mu)$, imply that as $\mu \to 0$

$$c^T x(\mu) \to c^T x^* \quad \text{and} \quad b^T y(\mu) \to b^T y^*,$$

so the objective function values for the perturbed problem converge to those achieved by an optimal solution (x^*, y^*, s^*) of the original problem. Furthermore, the following, stronger result holds [63].

Theorem 2.3. *Under the assumptions of primal feasibility, dual feasibility, and full row rank of matrix A, then as* $\mu \to 0$:

$$x(\mu) \to x^*, \quad (y(\mu), s(\mu)) \to (y^*, s^*).$$

Theorem 2.3 states that, under standard well-definedness conditions, the central path converges to a optimal solution of problem (2.1). Therefore, the central path can be used as a guideline to reach the optimal set. Algorithms that rely on the central path for finding the solution belong to the *path-following* class of interior point methods. We will describe them in more detail in Section 2.2.

The solution reached by following the central path is characterised by *strict complementarity*. This is described in the following result [90, 91].

Theorem 2.4 (Strict complementarity)**.** *If* (P) *and* (D) *are feasible, then there exist a point* $x^* \in \mathcal{P}$ *and a pair* $(y^*, s^*) \in \mathcal{D}$ *such that*

$$(x^*)^T s^* = 0 \quad \text{and} \quad x_i^* + s_i^* > 0, \quad i = 1, \ldots, n.$$

A solution (x^*, s^*) that satisfies the above theorem is said to be *strictly complementary*. On the grounds of a strictly complementary solution we can define the concept of *optimal partition*. Following Jansen [47], we define the support set of a vector $v \in \mathbb{R}^n$ as

$$\sigma(v) = \{i : v_i > 0,\ i = 1, \ldots, n\},$$

and partition the set of indices $\{1, \ldots, n\}$ as

$$\mathcal{B} = \sigma(x^*), \quad \mathcal{M} = \sigma(s^*).$$

From Theorem 2.4 it follows that this partition is well-defined, in the sense that a strictly complementary solution satisfies both $\mathcal{B} \cap \mathcal{M} = \emptyset$ and $\mathcal{B} \cup \mathcal{M} = \{1, \ldots, n\}$. The notions of strict complementarity and optimal partition are recurrent motifs in the analysis of interior point methods.

In the common case of multiple solutions, an interior point method algorithm terminates at the analytic center of the optimal face rather than at a vertex; in

some respects, through the concept of optimal partition, we can interpret this situation as having determined the whole set of optimal solutions. In contrast, the choice of the solution vertex provided by the simplex method is arbitrary, and depends on factors like the pivoting rule.

Often having a basic solution that identifies a vertex is considered to be an exact solution. However, we should discuss what we mean by "exact" solution. In most cases we do not need the additional precision of being on a vertex solution rather than at the analytic center of the optimal face. In this sense, integer programming represents a notable exception, as the integer solutions are at the vertices of the convex hull of feasible integer points. The difference between having an optimal basis or an optimal partition has important consequences on the use of the solution for sensitivity analysis [47, 93].

Vavasis and Ye [88] studied the properties of the curvature of the central path, discovering that the central path is characterised by $\mathcal{O}(n^2)$ curves of high degree and segments where it is relatively straight. Such curves appear in correspondence with changes in the optimal partition. Close to the end, when the optimal partition has been identified, the central path becomes a straight line [63]. In this region, the algorithm displays the quadratic convergence property typical of Newton's method.

We now consider the limit of (P_μ) for $\mu \to \infty$, and therefore find the point from which the central path departs. This corresponds to finding the point $\hat{x}$ that minimizes the barrier function, that is

$$\hat{x} = \arg\min_{x \in \mathcal{P}^0} \big(-\sum_{i=1}^{n} \ln x_i \big).$$

The point $\hat{x}$ is the *analytic center* of the feasible polytope, and was first studied by Sonnevend [84]. Given the strict convexity of the barrier function, the concept of analytic center is well defined. As the analytic center minimizes the barrier, it is the point farthest away from the boundary.

However, there is a problem with defining the central path in terms of analytic center: the central path is affected by the presence of redundant constraints. This happens because it is an exclusively analytic concept, which does not exploit geometric considerations. To overcome this disadvantage, other types of centers (center of gravity, center of the ellipsoid of maximum volume that can be inscribed in $\mathcal{P}$, volumetric center) can be defined, but they usually are too demanding to compute [41].

Such a drawback of the central path has been shown to have the potential for extreme consequences by Deza *et al.* [23], who managed to replicate the behaviour of the simplex method on the Klee-Minty cube within an interior point context.

This was obtained by adding an exponential number of redundant constraints parallel to the faces of the cube, so that the central path gets heavily distorted and goes through an arbitrarily small neighbourhood of all vertices of the cube.

Here we mention the fact that presolve techniques are usually implemented to find and remove as many as possible of these constraints, but while they are based on successful heuristics, they are not optimal. In particular, they may find implied constraints hard to detect [16].

2.1.2 Neighbourhoods of the central path

As we have seen, following the central path is the recommended way of traversing the interior of the feasible region towards the optimal solution. Nevertheless, it should be clear that keeping the iterates *exactly on* the central path is an unachievable aim. Finding a point that solves the perturbed complementarity conditions (2.7) for a specific μ is as difficult as solving the optimization problem itself. Therefore, we never insist on this extremely restrictive requirement, but we rather allow the iterates to be somewhere around the central path. This leads to the introduction of the concept of *neighbourhood* of the central path. We can define several neighbourhoods, characterised by different properties. Two neighbourhoods are often used in theoretical developments.

The first is based on the Euclidean norm, and it is often referred to as the *tight neighbourhood*:

$$\mathcal{N}_2(\theta) = \{w \in \mathcal{F}^0 : \|XSe - \mu e\|_2 \leq \theta\mu\},$$

where $\theta \in (0, 1)$. This neighbourhood defines points which lie very close to the central path. Search directions generated from points in this neighbourhood can be followed with a full step, and the barrier parameter can be decreased by a small amount at each iteration (giving rise to the name of *short-step algorithms* to the algorithms that are based on this neighbourhood). The closeness to the central path that the tight neighbourhood imposes and maintains produces the best convergence result for linear programming: short-step algorithms converge in $\mathcal{O}(\sqrt{n}L)$ iterations [55, 72]. However, since the reduction in the barrier parameter at each iteration is very small, the practical value of short-step algorithms is limited.

The other commonly used neighbourhood is based on the one-sided infinity norm, and it is often called the *wide neighbourhood*:

$$\mathcal{N}_{-\infty}(\gamma) = \{w \in \mathcal{F}^0 : x_i s_i \geq \gamma\mu,\ i = 1, \ldots, n\},$$

where $\gamma \in (0, 1)$. Algorithms based on such a neighbourhood are allowed to generate iterates that follow the central path more loosely. The iterates have more

freedom of movement as they can get closer to the boundary of the feasible set. However, the Newton direction computed from points in the wide neighbourhood has weaker properties, and a linesearch procedure is needed to ensure that the positivity of the (x, s) iterates is preserved. Algorithms based on the wide neighbourhood (usually called *long-step algorithms*) are less conservative than their short-step counterparts, and can decrease the barrier parameter more rapidly. Efficient implementations of interior point methods are based on some variation of a long-step algorithm.

In Section 2.3 we will study a variation of the $\mathcal{N}_{-\infty}$ neighbourhood which better describes the centrality requirements needed for a practical algorithm.

2.2 Path-following algorithms

We now bring together the elements we presented above and describe a complete path-following algorithm. We then discuss some theoretical results for algorithms in this class.

Primal–dual path-following methods solve the perturbed KKT conditions (2.2) by asking the complementarity pairs to align to a specific barrier parameter $\mu > 0$,

$$XSe = \mu e, \tag{2.7}$$

while enforcing $(x, s) > 0$. However, up to now, we have not defined how to choose the barrier parameter μ and how to update it at each iteration.

Given a starting iterate w^0 that belongs to some neighbourhood $\mathcal{N}$ such that $(x^0, s^0) > 0$, the value μ^0 of the initial barrier parameter is given by

$$\mu^0 = \frac{(x^0)^T s^0}{n}.$$

With the progress of iterations we would like the perturbed KKT conditions (2.4) to better and better approximate the system (2.2) of optimality conditions for the original problem. Hence, at each iteration, μ is monotonically decreased by the factor $\sigma \in (0, 1)$, called the *centering parameter* for reasons that will become clear later on. The choice of the centering parameter σ is algorithm-dependent. We provide theoretical insights on some possible choices in Section 2.2.1.

Path-following interior point methods seek a solution to the system of equations (2.4)

$$F(w) = \begin{bmatrix} Ax - b \\ A^T y + s - c \\ XSe - \sigma\mu e \end{bmatrix} = 0, \tag{2.8}$$

which is nonlinear in the perturbed complementarity constraints. We use Newton's method to linearise the system around the current point according to

$$\nabla F(w)\Delta w = -F(w),$$

where $\nabla F(w)$ is the Jacobian of the function (2.8) evaluated at the current primal–dual iterate w. The linearisation produces the Newton system

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} b - Ax \\ c - A^T y - s \\ -XSe + \sigma\mu e \end{bmatrix} = \begin{bmatrix} \xi_b \\ \xi_c \\ \xi_\mu \end{bmatrix}, \tag{2.9}$$

which needs to be solved for a search direction $\Delta w = (\Delta x, \Delta y, \Delta s)$, with $\mu = x^T s/n$, $\sigma \in (0, 1)$. We will discuss some ways of solving system (2.9) in Section 3.1.4. The search direction Δw thus computed is used to generate a new iterate

$$w^{k+1} = w^k + \alpha \Delta w,$$

where $\alpha \in (0, 1]$ is the largest feasible stepsize computed in such a way that $w^{k+1} \in \mathcal{N}$.

In Algorithm 2.1 we present the general framework of a primal–dual path-following algorithm. We will provide more details on each of the steps of this algorithm in Chapter 3.

Algorithm 2.1 Primal–dual path-following algorithm

Given: An initial iterate $w^0 \in \mathcal{N}$ such that $(x^0, s^0) > 0$;

Repeat:

Solve system (2.9) with a specified σ for a search direction Δw.

Evaluate the maximum feasible stepsize α_k in direction Δw.

Update the iterate $w^{k+1} = w^k + \alpha_k \Delta w$.

Until Some termination criteria are met.

2.2.1 Feasible methods

A feasible algorithm is characterised by the requirement that all primal and dual iterates always lie within the interior of the feasible region. For this reason, these algorithms need to start from a strictly feasible point $w^0 \in \mathcal{F}^0$.

In the feasible case, $\xi_b = \xi_c = 0$ in the right-hand side of system (2.9). Since the search direction computed from (2.9) guarantees

$$A\Delta x = 0 \quad \text{and} \quad A^T \Delta y + \Delta s = 0, \tag{2.10}$$

we can easily verify that feasibility is maintained throughout the algorithm:

$$A(x + \Delta x) = Ax + A\Delta x = b,$$
$$A^T(y + \Delta y) + (s + \Delta s) = (A^T y + s) + (A^T \Delta y + \Delta s) = c.$$

Moreover, using (2.10) we obtain

$$\Delta x^T \Delta s = -\Delta x^T (A^T \Delta y) = -(A\Delta x)^T \Delta y = 0. \tag{2.11}$$

As far as the progress in optimization is concerned, we can evaluate the complementarity gap that we would obtain when taking a step of length α in the direction Δw:

$$x(\alpha)^T s(\alpha) = x^T s + \alpha(s^T \Delta x + x^T \Delta s) + \alpha^2 \Delta x^T \Delta s = (1 - \alpha(1 - \sigma))x^T s,$$

where we used (2.11) and the fact that $s^T \Delta x + x^T \Delta s = -x^T s + \sigma\mu$. Now, dividing through by n, we obtain

$$\mu(\alpha) = x(\alpha)^T s(\alpha)/n = (1 - \alpha(1 - \sigma))\mu. \tag{2.12}$$

From (2.12) we observe that the progress in optimization depends on both α and σ. For a fixed σ, the length of the step α taken in the search direction Δw computed from (2.9) measures the reduction in complementarity gap: the longer the step, the bigger the reduction. This motivates the attempts to enlarge the stepsize by the use of corrector techniques (see Sections 3.2 and 3.3 and Chapter 4).

The centering parameter σ plays an important role as well. We can see that the biggest reduction is obtained for $\sigma = 0$. This does not come as a surprise, as the choice of $\sigma = 0$ corresponds to solving the KKT conditions (2.2) which describe the optimality conditions for the linear program (2.1). The choice of $\sigma = 1$, instead, leaves the complementarity gap unchanged. While this does not produce progress towards optimality, it tends to move the iterate closer to the central path. A step taken in a direction computed with $\sigma = 1$ is often called a *pure centering step*. It is therefore essential to choose σ appropriately, trying to balance the often conflicting aims of optimality and centrality.

Kojima, Mizuno and Yoshise [56] proposed a polynomial-time long-step algorithm that makes use of the wide $\mathcal{N}_{-\infty}$ neighbourhood to measure the distance of the iterates from the central path. The update of the barrier parameter for this family of algorithms happens for a constant

$$\sigma \in [\sigma_{\min}, \sigma_{\max}]$$

independent of n. This is an aggressive update, and a full Newton step is usually not feasible, so the stepsize needs to be damped. Also, more than one iteration may have to be performed before reducing the barrier parameter again. This algorithm, as well as many similar variants, converges in $\mathcal{O}(nL)$ iterations.

This result was then refined by the same group [55] and by Monteiro and Adler [72], who both presented a primal–dual algorithm for linear programming based on the tight $\mathcal{N}_2$ neighbourhood with the property of convergence in $\mathcal{O}(\sqrt{n}L)$ iterations. This is still the best complexity result for interior point methods for linear programming. In a short-step feasible method based on the $\mathcal{N}_2$ neighbourhood the barrier parameter is reduced by

$$\sigma = 1 - \delta/\sqrt{n} \tag{2.13}$$

at each iteration, for some positive constant δ, usually very small (0.05 according to Gonzaga [40]). As σ is always very close to 1, a lot of emphasis is put on maintaining centrality rather than advancing towards optimality. The slight reduction of the barrier parameter at each step guarantees that one iteration of Newton's method can keep the point in the tight neighbourhood of the central path. The choice of (2.13) allows convergence in $\mathcal{O}(\sqrt{n}L)$ iterations to be proved. However, this is a worst-case analysis, and in practice the same would happen even with a bigger update of the barrier parameter. This suggests that studying ways of allowing a more substantial reduction of the barrier parameter, at least in some iterations, would be worthwhile.

One important result in this direction was obtained by Mizuno, Todd and Ye [71], who introduced a short-step predictor–corrector method. Their strategy uses two nested neighbourhoods $\mathcal{N}_2(\theta^2)$ and $\mathcal{N}_2(\theta)$, $\theta \in (0, 1)$, and exploits the quadratic convergence property of Newton's method in such a tight neighbourhood of the central path. Their algorithm alternates between two search directions characterised by different properties. Starting from a point in the $\mathcal{N}_2(\theta^2)$ neighbourhood, by choosing $\sigma = 0$ in (2.9), the predictor direction gains optimality, possibly at the expense of worsening centrality, keeping the iterate in a larger neighbourhood $\mathcal{N}_2(\theta)$ of the central path. Then, a pure re-centering step is performed by setting $\sigma = 1$, leaving the duality gap unchanged but moving the iterate back into a tighter $\mathcal{N}_2(\theta^2)$ neighbourhood. Hence, on every second step the algorithm produces a point in $\mathcal{N}_2(\theta^2)$.

The Mizuno-Todd-Ye predictor–corrector algorithm [71], achieves the $\mathcal{O}(\sqrt{n}L)$ convergence property thanks to the optimizing predictor direction which guarantees the same progress achievable by a short-step feasible method, with the only difference that the value of the barrier parameter is reduced over two iterations.

An important contribution that this technique makes, is the idea of targeting optimality and centrality independently. The use of the very restrictive $\mathcal{N}_2$ neighbourhood makes the Mizuno-Todd-Ye algorithm unattractive for practical applications, but it provides a scheme upon which more computationally attractive methods can be constructed, as we will discuss in Chapter 3.

2.2.2 Infeasible methods

The results presented up to here concentrated on feasible methods. For these methods we assume that a strictly feasible starting point is readily available. However, finding a strictly feasible starting point is, in general, a nontrivial task, as solving the feasibility problem is an optimization problem in its own right. Moreover, the feasible region may have an empty interior, in which case the theory developed above does not apply. Allowing an infeasible starting point is particularly important for the algorithms implemented in practical interior point solvers. For these reasons, a need exists for developing techniques that do not require feasibility of the starting iterate.

A way to find a strictly feasible starting point involves solving an artificial Phase I subproblem by using the big-M method. However, the performance is dependent on the choice of the values given to the weights, and the use of very large values, while theoretically satisfying, causes numerical instabilities [58]. This is worsened by the presence of dense columns that compromise the computational efficiency.

A very different approach is based on the homogeneous self-dual formulation introduced by Ye, Todd and Mizuno [92]. This and a simplified variant are presented in [90, Chapter 9]. The self-dual formulation wraps the optimization problem into one of slightly larger dimension, but for which a strictly feasible solution is known from the start. Therefore, once embedded in the homogeneous self-dual form, the problem can be solved with a feasible algorithm. This formulation also has the very appealing property of being able to detect infeasibility with accuracy. The use of a self-dual formulation, however, comes with a price from a computational viewpoint, particularly because of the need for two extra backsolves at each iteration.

It is possible to develop an algorithm which only requires the x and s components to be strictly positive. This was initiated by Lustig [58], who proposed some new feasibility restoration directions. These were obtained as limiting directions as the big-M weight tends to infinity, and were shown to be equivalent to those obtained by an infeasible algorithm. However, it was the work of Kojima, Megiddo and Mizuno [54] that provided full theoretical analysis of convergence

of an infeasible interior point method as well as a stepsize rule that guarantees global convergence of the algorithm.

In such an algorithm, all iterates are usually infeasible, but the limit points are feasible and optimal. This is obtained by using a neighbourhood that admits infeasible points:

$$\mathcal{N}_{-\infty}(\gamma,\beta) = \{w : \|(\xi_b,\xi_c)\| \leq \beta\mu\frac{\|(\xi_b^0,\xi_c^0)\|}{\mu_0},\ (x,s) > 0,\ x_i s_i \geq \gamma\mu,\ i = 1,\ldots,n\},$$

where $\gamma \in (0,1)$ and $\beta \geq 1$ are parameters, and ξ_b^0, ξ_c^0 are the primal and dual residuals, respectively, at the initial iterate w^0.

In the $\mathcal{N}_{-\infty}(\gamma,\beta)$ neighbourhood there is no strict feasibility requirement for the iterates; however, the residuals at each iteration must be bounded above by a multiple of the complementarity measure μ. By reducing μ we can force the primal and dual residuals ξ_b and ξ_c to zero, thus approaching complementarity and feasibility at the same speed.

Letting $w(\alpha) = (x(\alpha), y(\alpha), s(\alpha)) = w + \alpha\Delta w$, then we can show that

$$\xi_b(\alpha) = (1-\alpha)\xi_b \quad \text{and} \quad \xi_c(\alpha) = (1-\alpha)\xi_c,$$

so infeasibilities reduce linearly with α, while for the complementarity gap

$$x(\alpha)^T s(\alpha) = (1-\alpha(1-\sigma))x^T s + \alpha^2 \Delta x^T \Delta s,$$

a reduction happens for a sufficiently small α. When feasibility is restored, that is when $\xi_b = \xi_c = 0$, an infeasible algorithm becomes identical to a feasible algorithm.

A globally convergent infeasible interior point algorithm has been implemented and tested by Lustig, Marsten and Shanno [60], who report positive results on a set of feasible test problems. The order of convergence for an infeasible algorithm was established by Zhang [95] to be $\mathcal{O}(n^2 L)$.

2.3 Symmetric neighbourhood

In Section 2.1.2 we discussed two neighbourhoods employed in the theory of interior point methods, and illustrated their main features and drawbacks. The $\mathcal{N}_2$ neighbourhood follows the central path very tightly, and the short-step methods based on it are extremely conservative and, in practice, very slow. The $\mathcal{N}_{-\infty}$ provides a much better framework for practical algorithms, as it allows the barrier parameter to reduce quickly. However, as it does not actively enforce an upper bound on the complementary products, it may allow the iterates to produce very unbalanced products.

The issue of unbalanced complementary products is very important for the practical success of an interior point code. It's important to stress how the complementary pairs play a role in the Newton system, and how their bad scaling causes a bad behaviour of Newton's method, which produces unreliable directions.

Jansen [47] considered the ratio

$$\varrho(XSe) = \frac{\min(XSe)}{\max(XSe)} \tag{2.14}$$

between the smallest and the largest complementarity pair as an indication of the quality of a point. The ratio (2.14) is a measure between 0 and 1, and is 1 for a perfectly centered iterate. Atkinson and Vaidya [4] noticed that the region in which Newton's method converges becomes smaller as the ratio $\varrho(XSe)$ decreases.

We argue that the quality of centrality (understood in a simplified way as complementarity) for a practical implementation of an interior point algorithm is *not* well characterised by either of two neighbourhoods $\mathcal{N}_2$ or $\mathcal{N}_{-\infty}$ commonly used in theoretical developments of interior point methods.

Practical experience with the primal–dual algorithm in HOPDM [31] suggests that one of the features responsible for its efficiency is the way in which the quality of centrality is assessed. By "centrality" we understand here the spread of complementarity products $x_i s_i$, $i = 1, \ldots, n$. Large discrepancies within the complementarity pairs, and therefore bad centering, create problems for the search directions: an unsuccessful iteration is caused not only by small complementarity products, but also by very large ones. This can be explained by the fact that Newton's direction tries to compensate for very large products, as they provide the largest gain in complementarity gap when a full step is taken. However, the direction thus generated may not properly consider the presence of very small products, which then become blocking components when the stepsizes are computed.

The notion of spread in complementarity products is not adequately represented in a computational setting by either of the two neighbourhoods $\mathcal{N}_2$ or $\mathcal{N}_{-\infty}$. To overcome this disadvantage, here we formalise a variation on the usual $\mathcal{N}_{-\infty}(\gamma)$ neighbourhood, in which we introduce an upper bound on the complementarity pairs. We propose using a symmetric neighbourhood $\mathcal{N}_s(\gamma)$, in which complementarity pairs have to satisfy $\gamma\mu \leq x_i s_i \leq \gamma^{-1}\mu$, where $\gamma \in (0,1)$, for a strictly feasible iterate $w \in \mathcal{F}^0$. This neighbourhood was implicitly used in [32] to define an achievable target for multiple centrality correctors (we refer the reader to Section 3.3).

We define the symmetric neighbourhood to be the set

$$\mathcal{N}_s(\gamma) = \{w \in \mathcal{F}^0 : \gamma\mu \leq x_i s_i \leq \frac{1}{\gamma}\mu,\ i = 1, \ldots, n\}, \tag{2.15}$$

where $\mu = x^T s/n$, and $\gamma \in (0, 1)$.

While the $\mathcal{N}_{-\infty}$ neighbourhood ensures that some products do not approach zero too early, it does not prevent products from becoming too large with respect to the average. In other words, it does not provide a complete picture of the centrality of the iterate. The symmetric neighbourhood $\mathcal{N}_s$, on the other hand, promotes the decrease of complementarity pairs which are too large, thus taking better care of centrality.

An upper bound on the size of complementary products is implicit in the $\mathcal{N}_{-\infty}$ neighbourhood. To find it, suppose that all but one of the complementarity products are at the lower bound $\gamma\mu$:

$$n\mu = x_1 s_1 + \sum_{i=2}^{n} x_i s_i = x_1 s_1 + (n-1)\gamma\mu,$$

from which it follows that in general

$$x_i s_i \leq (n(1-\gamma) + \gamma)\mu, \quad i = 1, \ldots, n. \tag{2.16}$$

We note the dependence on the problem dimension in the definition of the upper bound, hence its ineffectiveness for large-scale problems.

We now determine the value of n for which the upper bound of the symmetric neighbourhood is tighter than the implicit upper bound (2.16), that is

$$(n(1-\gamma) + \gamma)\mu > \frac{1}{\gamma}\mu.$$

After some trivial manipulations we find that the symmetric neighbourhood imposes a tighter upper bound for

$$n > \frac{1+\gamma}{\gamma}.$$

For a value of $\gamma = 0.1$, the bound is tighter whenever $n > 11$.

2.3.1 Theoretical analysis

Many theoretical developments aim at lowering the upper bound on the number of steps needed for convergence. The results provided by such worst-case complexity analysis are informative but exceedingly pessimistic.

Theoretical proofs of complexity generally follow a common scheme. First they rely on a computable measure of the closeness to the central path, accomplished by the concept of neighbourhood. Second, they show that the direction computed by solving the Newton system (2.9) can be followed with a strictly positive step (and therefore some progress is made at every iteration) and that it generates an

iterate that retains the property of being in some neighbourhood of the central path (possibly larger than the one before). Finally, they require a decrease in the barrier parameter that allows derivation of a polynomial upper bound on the number of iterations needed to reach the desired level of accuracy.

We analyse a long-step feasible path-following algorithm based on the symmetric neighbourhood $\mathcal{N}_s(\gamma)$, where the search direction Δw is found by solving system (2.9) with $r = (0,\ 0, -XSe+\sigma\mu e)^T$, $\sigma \in (0,1)$, $\mu = x^T s/n$. The exposition closely follows the presentation of Wright [90, Chapter 5].

Our main results are presented in Theorem 2.6 and Theorem 2.7. However, we first need a technical result that corresponds to Lemma 5.10 in [90], the proof of which is unchanged by the use of $\mathcal{N}_s$ rather than $\mathcal{N}_{-\infty}$.

Lemma 2.5. *If $w \in \mathcal{N}_s(\gamma)$, then* $\|\Delta X \Delta S e\| \leq 2^{-3/2}\Big(1 + \frac{1}{\gamma}\Big)n\mu$.

We now prove that it is possible to find a strictly positive stepsize α such that the new iterate $w(\alpha) = w + \alpha\Delta w$ does not leave the symmetric neighbourhood, and thus this neighbourhood is well defined. This result extends Theorem 5.11 in [90].

Theorem 2.6. *If $w \in \mathcal{N}_s(\gamma)$, then $w(\alpha) \in \mathcal{N}_s(\gamma)$ for all*

$$\alpha \in \left[0,\ 2^{3/2}\gamma\frac{1-\gamma}{1+\gamma}\frac{\sigma}{n}\right].$$

Proof. Let us express the complementarity product in terms of the stepsize α along the direction Δw:

$$\begin{aligned} x_i(\alpha)s_i(\alpha) &= (x_i + \alpha\Delta x_i)(s_i + \alpha\Delta s_i) \\ &= x_i s_i + \alpha(x_i\Delta s_i + s_i\Delta x_i) + \alpha^2\Delta x_i\Delta s_i \qquad (2.17) \\ &= (1-\alpha)x_i s_i + \alpha\sigma\mu + \alpha^2\Delta x_i\Delta s_i. \end{aligned}$$

We need to study what happens to this complementarity product with respect to both bounds of the symmetric neighbourhood. Let us first consider the bound $x_i s_i \leq \frac{1}{\gamma}\mu$. By Lemma 2.5, equation (2.17) implies

$$x_i(\alpha)s_i(\alpha) \leq (1-\alpha)\frac{1}{\gamma}\mu + \alpha\sigma\mu + \alpha^2 2^{-3/2}\left(1 + \frac{1}{\gamma}\right)n\mu.$$

At the new point $w(\alpha)$, the duality gap is $x(\alpha)^T s(\alpha) = n\mu(\alpha)$. The relation $x_i(\alpha)s_i(\alpha) \leq \frac{1}{\gamma}\mu(\alpha)$ holds provided that

$$(1-\alpha)\frac{1}{\gamma}\mu + \alpha\sigma\mu + \alpha^2 2^{-3/2}\left(1 + \frac{1}{\gamma}\right)n\mu \leq \frac{1}{\gamma}(1 - \alpha + \alpha\sigma)\mu,$$

from which we derive a first bound on the stepsize:

$$\alpha \leq 2^{3/2} \frac{1-\gamma}{1+\gamma} \frac{\sigma}{n} = \bar{\alpha}_1.$$

Considering now the bound $x_i s_i \geq \gamma\mu$ and using again Lemma 2.5, equation (2.17) implies

$$x_i(\alpha)s_i(\alpha) \geq (1-\alpha)\gamma\mu + \alpha\sigma\mu - \alpha^2 2^{-3/2}\left(1+\frac{1}{\gamma}\right)n\mu.$$

Hence, $x_i(\alpha)s_i(\alpha) \geq \gamma\mu(\alpha)$ provided that

$$(1-\alpha)\gamma\mu + \alpha\sigma\mu - \alpha^2 2^{-3/2}\left(1+\frac{1}{\gamma}\right)n\mu \geq \gamma(1-\alpha+\alpha\sigma)\mu,$$

from which we derive a second bound on the stepsize:

$$\alpha \leq 2^{3/2}\gamma \frac{1-\gamma}{1+\gamma} \frac{\sigma}{n} = \bar{\alpha}_2.$$

Therefore, for

$$\alpha \in [0,\, \min(\bar{\alpha}_1, \bar{\alpha}_2)] = [0,\, \bar{\alpha}_2],$$

we satisfy both bounds of the symmetric neighbourhood.

To conclude our proof we need to show that the $w(\alpha)$ iterate is still strictly feasible. Feasibility is trivially maintained, as it can be shown by the same argument used in Section 2.2.1. For positivity, we have that

$$x_i(\alpha)s_i(\alpha) \geq \gamma\mu(\alpha) = \gamma(1-\alpha(1-\sigma))\mu > 0,$$

as $\gamma \in (0,1)$, $\sigma \in (0,1)$, and $\mu > 0$. Hence $x(\alpha) > 0$ and $s(\alpha) > 0$, and $w(\alpha) \in \mathcal{N}_s(\gamma)$. □

In the next theorem, we prove the global linear convergence of the algorithm based on the symmetric neighbourhood.

Theorem 2.7. *Given γ and $0 < \sigma_{\min} < \sigma_{\max} < 1$, there is a constant δ independent of n such that*

$$\mu^{k+1} \leq \left(1 - \frac{\delta}{n}\right)\mu^k, \qquad \forall k \geq 0. \tag{2.18}$$

Proof. Using (2.12) and Theorem 2.6, we have

$$\mu^{k+1} = [1-\alpha(1-\sigma)]\,\mu^k \leq \left[1 - \frac{2^{3/2}}{n}\gamma\frac{1-\gamma}{1+\gamma}\sigma(1-\sigma)\right]\mu^k.$$

The term $\sigma(1-\sigma)$ is concave and assumes strictly positive values in the interval $(0,\, 1)$, with its minimum attained at one of the endpoints of the interval. Therefore, we prove our claim by setting

$$\delta = 2^{3/2}\gamma\frac{1-\gamma}{1+\gamma}\min\Big\{\sigma_{\min}(1-\sigma_{\min}),\, \sigma_{\max}(1-\sigma_{\max})\Big\}.$$ □

It is interesting to note that the introduction of the upper bound on the complementarity pairs does not change the polynomial complexity result proved for the long-step variant in the $\mathcal{N}_{-\infty}(\gamma)$ neighbourhood [90, Theorem 5.12]. In fact, the following theorem holds.

Theorem 2.8. *For a starting point $w^0 \in \mathcal{N}_s(\gamma)$, $\gamma \in (0, 1)$, such that $\mu^0 \leq 1/\epsilon^\tau$, where $\epsilon > 0$ is the convergence tolerance and $\tau > 0$, there is an index $K = \mathcal{O}(n \ln \frac{1}{\epsilon})$ for which*

$$\mu^k \leq \epsilon$$

for all iterations $k \geq K$.

Proof. This proof reworks Theorem 3.2 from [90] in our case. Iterating from (2.18) we obtain

$$\mu^k \leq \left(1 - \frac{\delta}{n}\right)^k \mu^0.$$

We now take logarithms of both sides

$$\ln \mu^k \leq k \ln \left(1 - \frac{\delta}{n}\right) + \ln \mu^0 \leq k \ln \left(1 - \frac{\delta}{n}\right) + \tau \ln \frac{1}{\epsilon} \leq -k\frac{\delta}{n} + \tau \ln \frac{1}{\epsilon},$$

where the last inequality is derived from the fact that $\ln(1 + \beta) \leq \beta$ for $\beta > -1$. The convergence tolerance is met when

$$-k\frac{\delta}{n} + \tau \ln \frac{1}{\epsilon} \leq \ln \epsilon = -\ln \frac{1}{\epsilon},$$

which holds for

$$k \geq \frac{1 + \tau}{\delta} n \ln \frac{1}{\epsilon}.$$ □

Therefore, the additional upper bounds on the complementarity pairs introduced with the symmetric neighbourhood do not produce any losses in the theoretical results proved for the $\mathcal{N}_{-\infty}$, but contribute to providing a better practical description of the centrality of a point. This understanding provides some additional insight into the desired characteristics of a well-behaved iterate.

The use of the symmetric neighbourhood will be one of the theoretical motivations of this work. Through it, in Section 3.3 we will put the work of [32] inside a more sound framework. Then we will use it again in the presentation of the weighted corrector directions strategy of Chapter 4, and in the analysis of an original warm-start strategy for stochastic programming of Chapter 5.

Chapter 3

Practical implementations of interior point methods.

In this chapter we turn our attention to the computational side of interior point methods. We concentrate on the main strategies which are at the basis of effective implementations of interior point methods for linear programming, and present other issues that are specific to practical algorithms. These have been documented extensively; for example, see [3, 38, 90] and the references therein. We also review some of the warm-start techniques for interior point methods that have been proposed in the literature.

3.1 Considerations for practical algorithms

In Sections 2.2.1 and 2.2.2 we presented theoretical results on the order of convergence of some interior point algorithms. In practice, convergence is much faster than stated by those results, as optimality is usually reached in a number of iterations proportional to the logarithm of the problem dimension. This was also shown by Ye [91, Chapter 6] through an average-case and probabilistic analysis. As in the analysis of the simplex method, we see here a large gap between the predicted and observed performances that is still to be fully understood.

Interior point methods are well-suited to solving very large scale optimization problems. Practical algorithms are very different from the ones used for theoretical purposes, and they usually implement some variation of infeasible interior point algorithms. In particular they show differences in the computation of the search directions, the evaluation of the stepsize, the use of the neighbourhood concept, and the update of the barrier parameter. This is further complicated by issues of computational efficiency and numerical stability, which often suggest the use of amended techniques or heuristic approaches, which make the analysis of the algorithms implemented in practice extremely difficult.

In the rest of this section we illustrate some of the algorithmic differences that are relevant to any practical implementation of interior point methods. They concern the choice of the starting point, the techniques used for solving the Newton system (2.9), the computation of the stepsize, the termination criteria and the use of correctors in the search directions. This selection has been made according to the relevance for this thesis. We will therefore not discuss other topics of extreme importance in practical algorithms such as presolve techniques, detection of infeasibilities, linear algebra implementations, use of iterative methods in the solution of the Newton system, and many others. Valuable references for these additional topics can be found in [3, 38].

3.1.1 Mehrotra's starting point heuristic

The initialisation of an interior point method consists of two logically independent steps: the presolve phase (in which some heuristic strategies are employed in finding duplicate rows or columns, discovering fixed variables, removing redundant constraints and tightening the bounds) and the process of finding an initial iterate. Here we discuss only the latter, and refer the reader to [2] for a treatment of the important topic of presolve techniques.

The choice of an initial iterate for interior point methods is a critical one. It challenges both the feasible and infeasible algorithms, and the solutions proposed in the two contexts are completely different. For infeasible algorithms the major hurdle of finding a feasible starting point is removed. However, the practical performance is very sensitive to the initial iterate, so the use of arbitrary points is not recommended. In particular, two requirements become important: the centrality of the point and the magnitude of the corresponding infeasibilities.

Mehrotra [65] introduced a tool to find a starting point that attempts to fulfil the above requirements. In this heuristic, we solve two least squares problems which aim to satisfy the primal and dual constraints:

$$\begin{aligned} &\min_{x} \; x^T x \quad &&\text{s.t.} \;\; Ax = b, \\ &\min_{(y,s)} \; s^T s \quad &&\text{s.t.} \;\; A^T y + s = c. \end{aligned}$$

These problems have solution

$$\tilde{x} = A^T(AA^T)^{-1}b, \quad \tilde{y} = (AA^T)^{-1}Ac, \quad \tilde{s} = c - A^T\tilde{y}.$$

The solution $\tilde{w}$ is further shifted inside the positive orthant, and the starting point is

$$w^0 = (\tilde{x} + \delta_x e,\ \tilde{y},\ \tilde{s} + \delta_s e),$$

where δ_x and δ_s are positive quantities. Their values depend on the distance of $\tilde{x}$ and $\tilde{s}$ to non-negativity and an additional correction term to ensure strict positivity.

The following variation is described in [38]:

$$\begin{aligned} \min_{x} \ & c^T x + \rho x^T x \quad \text{s.t.} \ \ Ax = b, \\ \min_{(y,s)} \ & b^T y + \rho s^T s \quad \text{s.t.} \ \ A^T y + s = c, \end{aligned}$$

where the parameter ρ is fixed to a predetermined value, in order to compensate for the contribution of the primal and dual objectives.

Mehrotra's starting point strategy has some drawbacks. It is scale dependent, it is affected by the presence of redundant constraints, and it does not guarantee producing a well-centered iterate. However it is commonly employed in interior point codes, and it is considered to be an effective heuristic for determining a starting point.

Considerations on what constitutes an appropriate starting point for interior point methods will be discussed again in Section 3.4, where we survey some warm-start strategies.

3.1.2 Computation of the stepsize

The computation of different stepsizes in the primal and dual spaces is done almost universally in implementations of interior point methods for linear programming. This has the advantage of speeding up the restoration of feasibility. According to [38], the use of different stepsizes contributes to a reduction of 10% in the number of iterations required when solving the Netlib set of tests.

In order to ensure that the (x, s) components of the iterate remain positive after moving along the Δw direction, we need to employ a linesearch procedure and find the maximum feasible stepsizes α_P and α_D such that

$$x + \alpha_P \Delta x > 0, \qquad s + \alpha_D \Delta s > 0.$$

The achievable stepsizes for a given search direction Δw, in the primal and dual space respectively, are computed as:

$$\alpha_P = \min \left\{ -\frac{x_i}{\Delta x_i} : \Delta x_i < 0 \right\}, \quad \alpha_D = \min \left\{ -\frac{s_i}{\Delta s_i} : \Delta s_i < 0 \right\}. \tag{3.1}$$

These stepsizes are then shortened with a factor $\alpha_0 = 0.99995$ to ensure strict positivity [38, 59].

We remark that in the computation of the stepsizes, we maintain the strict positivity of the iterates, without restricting the point inside a neighbourhood.

While this is done on the grounds of computational efficiency, we may lose the global convergence property ensured by keeping the iterates in a neighbourhood of the central path.

3.1.3 Termination criteria

In contrast to active set algorithms, interior point methods reach a solution only asymptotically. Because of the presence of the barrier term that keeps the iterates away from the boundary, they can never produce an exact solution. When working in the context of limited precision of the floating point representation typical of computers, feasibility and complementarity can be attained only within a certain level of accuracy.

For these reasons, criteria have to be established according to which the termination of the algorithm can be decided. Some common termination criteria used in practice are the following [38]:

$$\frac{\|Ax-b\|}{1+\|x\|_\infty} \leq 10^{-p}, \qquad \frac{\|A^Ty+s-c\|}{1+\|s\|_\infty} \leq 10^{-p}, \qquad \frac{|c^Tx-b^Ty|}{1+|b^Ty|} \leq 10^{-q}. \quad (3.2)$$

The values of p and q required depend on the specific application. In the literature, it is common to use the value $p = q = 8$.

Another set of criteria, implemented in the interior point code PCx [20] is given by:

$$\frac{\|Ax-b\|}{1+\|b\|} \leq 10^{-p}, \qquad \frac{\|A^Ty+s-c\|}{1+\|c\|} \leq 10^{-p}, \qquad \frac{\mu}{1+|c^Tx|} \leq 10^{-q}. \quad (3.3)$$

The last condition in (3.3) is weaker than the corresponding one in (3.2) as, through μ, it depends on n. Therefore, in this case it is common to require $q = 10$.

The third condition in each set is usually the most important, as it is commonly attained only after the feasibility requirements are satisfied. It is worth noting that the criteria (3.2) and (3.3) are dependent on the scaling of the data.

Once a solution has been found within a prescribed optimality tolerance, such a point can be projected onto a face of the polyhedron in an efficient way. This can be done using a strongly polynomial algorithm due to Megiddo [64]. This procedure goes under the name of *basis crossover*, as, given a complementary primal–dual solution, it generates a basis that is both primal and dual feasible.

3.1.4 Solving the Newton system

The solution of system (2.9) is the computationally dominant step in each iteration of an interior point algorithm. Throughout this thesis, we will restrict our

attention to using a direct approach in solving these equations. We remark, however, that a wealth of research has explored the use of iterative methods in the computation of the search direction [7, 75].

System (2.9) is usually reduced to two other formulations by exploiting the block structure of its matrix. The *augmented system* formulation is obtained by using the last row of (2.9) to eliminate $\Delta s = X^{-1}(\xi_\mu - S\Delta x)$. This produces

$$\begin{bmatrix} -X^{-1}S & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \xi_c - X^{-1}\xi_\mu \\ \xi_b \end{bmatrix}, \tag{3.4}$$

which is a symmetric but indefinite system. By further eliminating Δx, we reduce system (3.4) to the set of *normal equations*

$$AD^2A^T\Delta y = AD^2(\xi_c - X^{-1}\xi_\mu) + \xi_b, \tag{3.5}$$

where we introduced the notation $D^2 = S^{-1}X$. Under the standard assumption of full row rank for A, matrix AD^2A^T is positive definite, since $D_i^2 = x_i/s_i > 0$ for all $i = 1, \ldots, n$.

Besides the issue of definiteness, the two formulations differ in terms of sparsity and conditioning, the normal equations usually being denser and worse conditioned. The choice between the augmented system and normal equations techniques depends also on the relative density of AD^2A^T with respect to A.

Normal equations are to be avoided when there are dense columns in A, as they generate dense blocks in AD^2A^T. In the broad context of weighted least squares computations, the choice between the augmented system and normal equations techniques was studied long before the development of interior point methods, see for example [25].

The augmented system formulation requires particular attention to the development of linear algebra routines because it involves an indefinite matrix. This raises problems of numerical stability, and an accurate choice of pivoting strategies is fundamental. Maros and Mészáros [62] presented an in-depth study of the properties of the augmented system formulation.

A direct approach computes the Cholesky factorisation $M = LDL^T$ of the constraint matrix M with a lower triangular L and a diagonal matrix D. In the computation of the factors, inevitably new nonzero entries are generated (the so called *fill-in*). The amount of fill-in is strongly influenced by the ordering in which the operations are performed. Various ordering techniques have been developed to reduce the fill-in in the Cholesky factors, thus maintaining sparsity and reducing the required work [81]. Once the factors have been computed, the solution of system $Mx = r$ happens in two steps: first by solving $Lz = r$ for z, then $L^Tx = D^{-1}z$ for x. The solution phase is also called the *backsolve phase*. A

careful implementation of the linear algebra routines that perform the ordering, factorisation and solution of the Newton system is one of the key points for an efficient interior point solver.

The computation of the Cholesky factors dominates the cost of each iteration. As this is usually a major computational task, the efforts in the theory and practice of interior point methods concentrate on reducing the number of times the Newton system matrix (2.9) has to be factorised. In particular, it is worth adding more (cheap) backsolves if this reduces the number of (expensive) factorisations.

Two techniques have proved particularly successful in reducing the number of iterations within practical algorithms: Mehrotra's predictor–corrector algorithm [65] and multiple centrality correctors [32]. Mehrotra's predictor–corrector technique [65] uses two backsolves per factorisation; the multiple centrality correctors technique [32] allows recursive corrections. A larger number of backsolves per iteration is allowed, leading to a further reduction in the overall number of factorisations. These techniques have been implemented in most commercial and academic interior point solvers for linear and quadratic programming such as BPMPD, Cplex, HOPDM, Mosek, Ipopt, OOPS, OOQP, PCx and Xpress (see Appendix B in [90]).

Since these two methods were developed, there have been a number of attempts to investigate their behaviour rigorously and thus provide further insight on their success. Such objectives are difficult to achieve because correctors use heuristics that are effective in practice but hard to analyse theoretically. Besides, both correcting techniques are applied to long-step and infeasible algorithms which have very little in common with the short-step and feasible algorithms that display the best known theoretical complexity. These strategies will be the focus of the next two sections.

3.2 Mehrotra's predictor–corrector algorithm

In Mehrotra's predictor–corrector algorithm [65], first a predictor direction is generated to make progress towards optimality, and then a corrector is computed to remedy for some of the error made by the predictor and move the iterate closer to the central path.

A number of advantages can be obtained by exploiting the linearity in the Newton system (2.9) and considering the right-hand as

$$\begin{bmatrix} b - Ax \\ c - A^T y - s \\ -XSe + \sigma\mu e \end{bmatrix} = \begin{bmatrix} b - Ax \\ c - A^T y - s \\ -XSe \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \sigma\mu e \end{bmatrix},$$

thus splitting the computation of the Newton direction into two steps, corresponding to solving the linear system (2.9) independently for the two right-hand sides above.

First, we can postpone the choice of the centering parameter σ and base it on the assessment of the quality of the pure Newton direction computed with the first part of the right-hand side; second, the error made by this direction may be taken into account and corrected. Mehrotra's predictor–corrector algorithm [65] translates these observations into a powerful computational method. This technique is extremely efficient in practice [59, 65]. Since its introduction, it has been considered the method of choice for practical implementations because it is usually very fast and reliable. Moreover, it has a convincing interpretation in terms of second-order approximations to the central path.

3.2.1 Affine-scaling predictor direction

The *predictor direction* $\Delta_a w = (\Delta_a x, \Delta_a y, \Delta_a s)$ is obtained by solving system (2.9) with right-hand side

$$r_a = \begin{bmatrix} b - Ax \\ c - A^T y - s \\ -XSe \end{bmatrix}. \tag{3.6}$$

This corresponds to computing the pure Newton direction for the original KKT system (2.2), and this direction is often called the *affine-scaling* direction. This direction optimizes strongly, as it targets a point for which all complementarity products are zero. The achievable stepsizes for the predictor direction, in the primal and dual space respectively, are computed according to (3.1).

As it targets a point for which $XSe = 0$, the affine-scaling direction may be distracted by points that have small complementarity products but are not optimal. In particular, it may well point towards the boundary of the positive orthant or approach an infeasible vertex, generating a very small stepsize. This effect is usually worsened if the current iterate is badly centered, and therefore only a very small step is acceptable in order to maintain positivity or to keep the iterate in some neighbourhood of the central path.

Since it completely ignores the central path, the affine-scaling direction is not enough in a practical implementation of interior point methods, but has to be complemented by other techniques. The role of the centering term is to remedy this situation by causing the search direction to rotate towards the interior of the feasible region, therefore allowing a longer stepsize. Tapia *et al.* [85] noted that the choice of the centering parameter can be crucial both in theory and in

practice, and suggest that it is a function of the Newton step. This calls for two separate backsolves at each iteration.

3.2.2 Second-order corrector direction

Mehrotra's algorithm exploits a centrality corrector in order to remedy badly centered points. The purpose of this direction is to move closer to the central path, and therefore reduce the spread in complementarity products, without aiming for more optimality. This is a somewhat conservative direction, which it is hoped will provide more room for movement at the next iteration.

One tool introduced by Mehrotra [65] is a dynamic evaluation of the centering parameter σ. It is based on a simple heuristic that evaluates the quality of the predictor direction in order to judge the amount of centering term needed. The length of the feasible steps α_P and α_D for the affine-scaling direction are used to predict the complementarity gap after such a step:

$$g_a = (x + \alpha_P \Delta_a x)^T (s + \alpha_D \Delta_a s). \tag{3.7}$$

The ratio $g_a/x^T s \in (0, 1)$ measures the quality of the predictor direction. A small ratio indicates a successful reduction of the complementarity gap. On the other hand, if the ratio is close to one, then very little progress is achievable along the direction $\Delta_a w$, and a bigger centering term is recommended.

In [65] the following choice of the new barrier parameter is suggested

$$\left(\frac{g_a}{x^T s}\right)^2 \frac{g_a}{n} = \left(\frac{g_a}{x^T s}\right)^3 \frac{x^T s}{n}, \tag{3.8}$$

corresponding to the choice of $\sigma = (g_a/x^T s)^3$ for the centering parameter. If the predictor provides a good improvement, a small σ is chosen, and very little centering will be used. When, on the other hand, the affine-scaling direction produces very small stepsizes and very little improvement can be achieved, σ will be close to one, and a stronger recentering will occur.

The centering parameter, could be chosen more generally as

$$\sigma = \left(\frac{g_a}{x^T s}\right)^p,$$

for various choices of the exponent p. Mehrotra [65] studied the effect of different values $p = 1, 2, 3, 4$ on a subset of Netlib problems, and concluded that for p between 2 and 4 there was not much difference. Also Lustig *et al.* [59] commented on the weak dependence of the computational performance on the choice of the exponent, and settled for the value $p = 2$.

A further important contribution by Mehrotra is the introduction of a second-order direction. As said above, the affine-scaling direction corresponds to a linear approximation to the the trajectory from the current point to the optimal set, where no information about higher-order terms is taken into account. This linearisation, however, produces an error which can be determined analytically. Assuming that a full step in the affine-scaling direction is made, the new complementarity products are equal to

$$(X + \Delta_a X)(S + \Delta_a S)e \;=\; XSe + (S\Delta_a x + X\Delta_a s) + \Delta_a X \Delta_a S e \;=\; \Delta_a X \Delta_a S e,$$

as the third equation in the Newton system (2.9) satisfies $S\Delta_a x + X\Delta_a s = -XSe$. The term $\Delta_a X \Delta_a S e$ corresponds to the error introduced by Newton's method in linearising the complementarity conditions of the KKT system (2.2).

Ideally, we would like the next iterate to be perfectly centered:

$$(X + \Delta X)(S + \Delta S)e = \sigma \mu e,$$

which is equivalent to solving the nonlinear system

$$S\Delta x + X\Delta s = -XSe + \sigma\mu e - \Delta X \Delta S e. \tag{3.9}$$

By comparing (3.9) and (2.9), we see that the linearisation error made by the affine-scaling direction is exactly the $\Delta X \Delta S e$ term that is missing from the right-hand side of the last equation of (2.9).

Mehrotra's corrector term takes into account the linearisation error. Hence, a second-order correction is computed by solving the Newton system (2.9) with right-hand side

$$r_c = \begin{bmatrix} 0 \\ 0 \\ -\Delta_a X \Delta_a S e + \sigma \mu e \end{bmatrix}, \tag{3.10}$$

for the direction $\Delta_c w = (\Delta_c x, \Delta_c y, \Delta_c s)$. Such a corrector direction combines the centrality term $\sigma\mu e$ and the second-order term $\Delta_a X \Delta_a S e$.

Once the predictor and corrector terms are computed, they are added together to produce the composite predictor–corrector direction

$$\Delta_M w = \Delta_a w + \Delta_c w. \tag{3.11}$$

Note that the affine-scaling predictor and Mehrotra's corrector direction are equally weighted in their contribution to the final search direction. This argument will be considered again in Chapter 4, where we study the use of a weighting strategy for the corrector directions.

The next iterate is given by

$$w^{k+1} = w^k + (\alpha_P \Delta_M x^k, \alpha_D \Delta_M y^k, \alpha_D \Delta_M s^k)$$

where α_P and α_D are chosen according to (3.1) with respect to the direction $\Delta_M w$ of (3.11).

Mehrotra's algorithm exploits the Jacobian matrix employed in finding the affine-scaling direction when calculating the corrector direction. Hence it reuses the same Cholesky factors, which improves the computational efficiency. The cost of a single iteration in the predictor–corrector method is only slightly larger than that of the standard method because two backsolves per iteration have to be executed, one for the predictor and one for the corrector.

We summarise Mehrotra's predictor–corrector scheme in Algorithm 3.1.

Algorithm 3.1 Mehrotra's predictor–corrector algorithm

Given: An initial iterate w^0 such that $(x^0, s^0) > 0$;

Repeat:

Solve system (2.9) with right-hand side r_a (3.6) for a predictor direction $\Delta_a w$.

Set μ according to (3.8) and find Mehrotra's corrector direction $\Delta_c w$ by solving system (2.9) with right-hand side r_c (3.10).

Evaluate the maximum feasible stepsize α_k in $\Delta_M w = \Delta_a w + \Delta_c w$.

Update the iterate $w^{k+1} = w^k + \alpha_k \Delta w$.

Until The termination criteria (3.2) are met for some predetermined value of p and q.

The practical advantage of Mehrotra's predictor–corrector technique is that it often produces longer stepsizes before violating the non-negativity constraints. This usually translates into significant savings in the number of iterations: Mehrotra [65] reports on savings of the order of 35%-50% compared to other strategies. For problems for which the factorisation cost is relevant, this leads into significant savings in CPU time [59, 65]. Indeed, Mehrotra's predictor–corrector technique is advantageous in all interior point implementations for linear programming which use direct methods to compute the Newton direction.

An insightful explanation of how Mehrotra's second-order direction helps in lengthening the step has been provided by Jarre and Wechs [48]. Consider one of the equations that define the affine-scaling predictor direction:

$$s_i \Delta_a x_i + x_i \Delta_a s_i = -x_i s_i. \tag{3.12}$$

As $(x_i, s_i) > 0$, we can rewrite (3.12) as

$$\frac{\Delta_a x_i}{x_i} + \frac{\Delta_a s_i}{s_i} = -1. \tag{3.13}$$

We consider two possibilities: either $\Delta_a x_i \Delta_a s_i > 0$ or $\Delta_a x_i \Delta_a s_i < 0$. In the first case, as the right-hand side of (3.13) is negative, it must be true that $\Delta_a x_i < 0$ and $\Delta_a s_i < 0$. From (3.13) we get the bounds

$$\frac{\Delta_a x_i}{x_i} > -1 \quad \text{and} \quad \frac{\Delta_a s_i}{s_i} > -1,$$

which imply

$$x_i + \Delta_a x_i > 0 \quad \text{and} \quad s_i + \Delta_a s_i > 0,$$

that is, this is not a blocking component, as a step of full length can be taken.

In the second case, $\Delta_a x_i \Delta_a s_i < 0$, let us assume that $\Delta_a x_i > 0$ and $\Delta_a s_i < 0$. Considering again (3.13), we obtain the bound

$$\frac{\Delta_a s_i}{s_i} < -1,$$

so $s_i + \Delta_a s_i < 0$, and the stepsize must be damped in order to keep the iterate positive. Therefore, this component may be a blocking component for the stepsize. Consider now the equation for Mehrotra's predictor–corrector direction

$$s_i \Delta_c x_i + x_i \Delta_c s_i = \mu - x_i s_i - \Delta_a x_i \Delta_a s_i > -x_i s_i,$$

where the inequality follows from the fact that $\mu - \Delta_a x_i \Delta_a s_i > 0$, as $\Delta_a x_i \Delta_a s_i < 0$. Using (3.12) we obtain

$$s_i \Delta_c x_i + x_i \Delta_c s_i > s_i \Delta_a x_i + x_i \Delta_a s_i,$$

which we can interpret as an increase in the weighted sum of the search directions. Hence, it is possible that $\Delta_c s_i > \Delta_a s_i$, which would lead to an increase in the stepsize if this was a blocking component in the affine-scaling direction.

As mentioned above, Mehrotra's way of assessing the value of σ and the computation of the second-order term is a heuristic procedure, thus there are no global convergence results or polynomial complexity results. Tapia *et al.* [85] interpreted the Newton step produced by Mehrotra's predictor–corrector algorithm as a perturbed composite Newton method and gave results on the order of convergence. They proved that a level-1 composite Newton method, when applied to the perturbed Karush-Kuhn-Tucker system, produces the same sequence of iterates as Mehrotra's predictor–corrector algorithm. While, in general, a level-m composite Newton method has a Q-convergence rate of $m+2$ [76], the same result does not hold if the stepsize has to be damped to keep non-negativity of the iterates, as is necessary in an interior-point setting. However, under the additional assumptions of strict complementarity and nondegeneracy of the solution and feasibility of the starting point, Mehrotra's predictor–corrector method can be shown to have Q-cubic convergence [85].

3.3 Multiple centrality correctors

Mehrotra's predictor–corrector, as it is implemented in optimization solvers [59, 65], is a very aggressive technique. It is based on the assumption, rarely satisfied, that a *full* step in the corrected direction will be achievable. Moreover, an attempt to correct all complementarity products to the same value μ is also very demanding and occasionally counterproductive. Besides, practitioners noticed that this technique may sometimes behave erratically, especially when used for a predictor direction applied from highly infeasible and badly centered points. Finally, Mehrotra's corrector does not provide CPU time savings when used iteratively [13].

Trying to provide a remedy to the above considerations, Gondzio [32] introduced the multiple centrality correctors technique as an additional tool to complement those presented by Mehrotra. The idea behind this technique is to "force" an increase in the length of the steps by correcting the centrality of Mehrotra's iterate. Instead of attempting to correct for the whole second-order error, multiple centrality correctors concentrate on improving the complementarity pairs which really seem to hinder the progress of the algorithm, that is, the complementarity products that are far from the average.

In introducing the multiple centrality correctors technique, we assume that a long-step path-following algorithm is used, and work with the symmetric neighbourhood $\mathcal{N}_s(\gamma)$ of the central path as defined in Section 2.3. In the framework of multiple centrality correctors, we look for a centrality corrector $\Delta_m w$ for which larger steps are allowed in the composite direction $\Delta w = \Delta_p w + \Delta_m w$. In this context, Mehrotra's predictor–corrector direction (3.11) is considered to be a new predictor direction $\Delta_p w$ to which one or more centrality correctors can be applied. In this sense, multiple centrality correctors can be interpreted as higher-order corrector terms. Different choices for the first predictor direction are also possible and, in certain special circumstances, such as warm-starting, may be justified.

Assume that a predictor direction $\Delta_p w$ is given and the corresponding feasible stepsizes α_P and α_D in the primal and dual spaces are determined. We want to enlarge the stepsizes to

$$\tilde{\alpha}_P = \min(\alpha_P + \delta,\, 1) \quad \text{and} \quad \tilde{\alpha}_D = \min(\alpha_D + \delta,\, 1),$$

for some fixed aspiration level $\delta \in (0, 1)$. We compute a trial point

$$\tilde{x} = x + \tilde{\alpha}_P \Delta_p x, \quad \tilde{s} = s + \tilde{\alpha}_D \Delta_p s, \tag{3.14}$$

and the corresponding complementarity products $\tilde{v} = \tilde{X}\tilde{S}e \in \mathbb{R}^n$. It is worth noting that since we increased the stepsizes from α_P, α_D to $\tilde{\alpha}_P, \tilde{\alpha}_D$, the trial point

(3.14) is infeasible. However, as the trial point is used exclusively in determining a target for the centrality corrector, we will not worry about it.

The products in $\tilde{v}$ are very unlikely all to be equal to μ. Some of them are significantly smaller than μ, including cases of negative components in $\tilde{v}$, and some exceed μ. Instead of trying to correct all of them to the same value μ, we correct only those that lie outwith the symmetric neighbourhood of the central path. As the symmetric neighbourhood (2.15) provides a lower and an upper bound on each complementarity product, this means checking where each $\tilde{v}_i$ lies with respect to both of them.

Namely, we try to move small products ($\tilde{v}_i \leq \gamma\mu$) to the lower bound $\gamma\mu$, and move large products ($\tilde{v}_i \geq \gamma^{-1}\mu$) to the upper bound $\gamma^{-1}\mu$, where $\gamma \in (0,1)$ is the neighbourhood parameter. Complementarity products which satisfy $\gamma\mu \leq \tilde{v}_i \leq \gamma^{-1}\mu$ are already within the bounds of the symmetric neighbourhood and thus reasonably close to their target values: therefore they do not need to be changed. In other words, we attempt to move the iterate inside the symmetric neighbourhood $\mathcal{N}_s(\gamma)$ of the central path.

The corrector term $\Delta_m w$ is computed by solving the usual system of equations (2.9) for a special right-hand side

$$r_m = \begin{bmatrix} 0 \\ 0 \\ t \end{bmatrix}, \tag{3.15}$$

where the target t is defined as follows:

$$t_i = \begin{cases} \gamma\mu - \tilde{v}_i & \text{if } \tilde{v}_i \leq \gamma\mu \\ \frac{1}{\gamma}\mu - \tilde{v}_i & \text{if } \tilde{v}_i \geq \frac{1}{\gamma}\mu \\ 0 & \text{otherwise.} \end{cases} \tag{3.16}$$

We stress that the target point is not on the central path, but in the symmetric neighbourhood and that not all complementary products are corrected. This makes the target (3.16) an achievable one.

One important feature of the multiple centrality correctors technique is that it can be applied recursively to the direction $\Delta_p w := \Delta_p w + \Delta_m w$. The maximum number of centrality correctors allowed is problem dependent. Such a number is determined heuristically, trying to balance the cost of additional backsolves to the savings in iteration count, as detailed in [32]. Each corrector computed is accepted as long as the stepsizes increase at least by a fraction of the aspiration level δ.

In Algorithm 3.2 we summarise the steps of the multiple centrality correctors technique.

Algorithm 3.2 Multiple centrality correctors

Given: An iterate w^k, a search direction $\Delta_p w$, and the maximum number l of correctors allowed;

Repeat:

Solve system (2.9) with right-hand side r_m (3.15), for a centrality corrector direction $\Delta_m w$.

Set $\Delta_p w := \Delta_p w + \Delta_m w$.

Evaluate the maximum stepsize α_k in direction $\Delta_p w$.

Until The stepsize does not increase by at least a fraction of the aspiration level δ or the number of correctors reaches l.

Update the iterate $w^{k+1} = w^k + \alpha_k \Delta_p w$.

The computational experience presented in [32] showed that this strategy is effective, as the stepsizes in the primal and dual spaces computed for the composite direction are larger than those corresponding to the predictor direction. This leads to reductions in the number of iterations, which in turn translate into CPU time savings which increase with the factorisation cost.

Virtually all existing interior point codes implement the multiple centrality correctors technique [90, Appendix B]. Their use depends on the quality of the linear algebra implementation, particularly with respect to the Cholesky factorisation routines, as this affects the ratio between factorisation and backsolve costs.

We will present further improvements in the way corrector directions can be computed in Chapter 4.

3.4 Warm-start with interior point methods

As discussed in Section 3.1.1, the problem of finding a starting point is usually solved by using Mehrotra's starting point heuristic [65], which is considered to be computationally effective for generic problems.

However, many practical applications rely on solving a sequence of closely related problems, where the problem instances differ from each other by some perturbation. This happens within algorithms that are sequential in their nature, such as sequential linear programming or sequential quadratic programming. It is also very common in (mixed) integer programming, when the problems are solved by relaxing the integrality constraints and introducing some branching strategy, such as in branch-and-bound, branch-and-cut, branch-and-price. Similarly, sequential problems appear in the context of solution methods based on cutting planes.

In these situations, we expect the solution of one instance of a problem to

be close to the solution of the next one, in the sense that some of the information gathered in the solution process may still be valid. Therefore, starting the optimization of a problem from the solution of the previous one should reduce the computational effort of solving the perturbed instance. The strategies that exploit an advanced starting point are called *warm-start strategies*.

Warm-start techniques are very successful when implemented with a simplex solver (see, for example, [12]). The main reason for this is that, as the solution of a problem is a vertex, it is an ideal starting point for a perturbed problem instance. If the perturbation is not too big, then optimality can be recovered very quickly. In the context of interior point methods, it is much more difficult to implement a warm-start strategy in a successful way, for the reasons we outline below.

The optimal solution of a linear programming problem found with a path-following interior point method is very close to a vertex of the feasible polytope or, in the common case of multiple solutions, corresponds to the analytic center of the optimal set of solutions. When the polytope changes (due, for example, to the addition of cutting planes or other changes in the problem data), the optimal solution changes as well. In such a case the previously optimal solution may now be very far from the central path of the perturbed problem (see Figure 3.1).

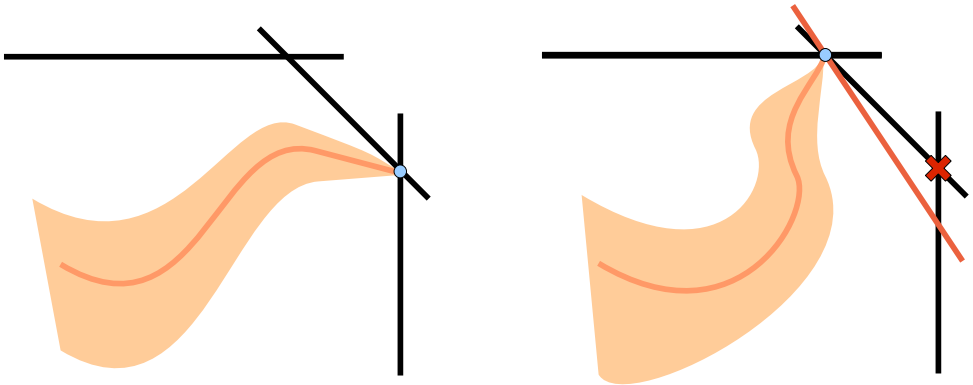

Figure 3.1: Effect of a perturbation on the central path.

As presented in Chapter 2, interior point methods approach the solution to the KKT system of optimality conditions by relaxing the complementarity requirements and obtaining the perturbed system (2.4). As moving towards a vertex can be interpreted as making a decision on the optimal partition, considering the system (2.4) is equivalent to postponing the choice of the optimal partition. If the vertex is not an optimal one, then the central path will not approach it.

The effectiveness of an interior point algorithm degrades when an iterate gets too close to a boundary before optimality is reached, or, equivalently, when the iterate gets far from the central path. In these situations the algorithm may spend many iterations in recovering centrality, during which small steps are usually

generated and thus very slow progress is achieved (see Figure 3.2).

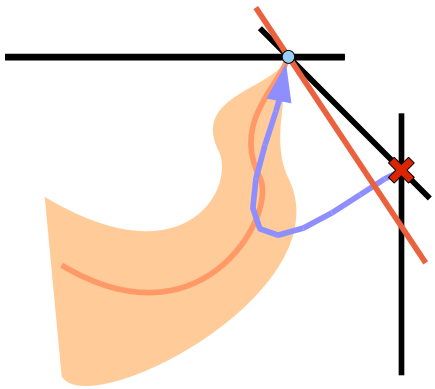

Figure 3.2: Typical behaviour of an interior point method if warm-started from the previously optimal solution.

Hipolito [44] considered the issue of robustness of search directions in interior point methods in these situations. In his analysis, Hipolito showed that if the iterate is close to a boundary, the affine-scaling direction may be parallel to the nearby constraints. In such cases, the corrector direction may also display the same behaviour, and short stepsizes are obtained for the resulting combined search direction. The analysis in [44] concerns the dual affine-scaling algorithm, but it also seems to provide an interesting insight into the misbehaviour of search directions used in primal–dual algorithms when computed from non-central points.

The required features of a good warm-start candidate for an interior point algorithm are somewhat contradictory. The point should not be too close to the boundary of the feasible region in order to be able to absorb larger perturbations in the problem data. Also, it should be sufficiently advanced to provide computational savings over a cold-start iterate. These considerations lead to the idea of storing an approximate μ-center well before reaching optimality [33, 34, 39, 94].

The theory and practice of warm-start techniques for interior point methods is a relatively new and still open field of study. In the remainder of this section we present a review of some of the warm-start approaches proposed in the interior point literature.

3.4.1 Literature review

Mitchell [67] and Mitchell and Todd [70] analyse the potential reduction interior point method within a cutting plane algorithm. They exploit the fact that a primal feasible point can be constructed after a set of new columns is added to the problem. They use this strategy with success in a column generation scheme and more generally in the solution of combinatorial optimization problems.

The role of interior point methods in integer and combinatorial optimization has been comprehensively surveyed in [68] and [69]. This is relevant as these classes of optimization problems are generally solved by formulating a sequence of linear relaxations, where the problem instances differ by the addition of cutting planes and facet-inducing constraints.

Hipolito [44] studies an alternative centering direction in the context of the dual affine-scaling algorithm. Such a direction is designed to move the iterate away from the boundary, overcoming the risk of moving parallel to it that was mentioned earlier in this section. By considering a weighted least squares formulation, Hipolito develops the dual affine-scaling and the corresponding Newton centering directions. This study has an immediate interest for warm-starting approaches, as the resulting direction points towards the interior of the feasible region, thus providing the algorithm with the necessary space to make fast progress. Unfortunately, it is developed only for the dual affine-scaling algorithm, and to the best of our knowledge there have been no studies on how to obtain a similar search direction in the primal–dual context.

Gondzio [33] presents a warm-start procedure for primal–dual interior point methods in the context of a cutting plane method. The interior point method is used to solve a sequence of restricted master problems, which differ by one or more cutting planes. By construction, in such a setting the solution to one problem deeply violates some of the newly added constraints. The optimal solution to a problem is necessarily very close to the boundary, thus is an unattractive starting point for a perturbed problem, and an alternative warm-start iterate needs to be defined. The idea proposed in [33] is to store a nearly optimal point (3–4 digits of accuracy) to be employed as a warm-start point. Because of the cutting plane setting, the problem is then solved to optimality (7–8 digits of accuracy) in order to generate appropriate cuts. As one requirement for a good iterate is centrality, it is of interest to perform a few centering steps on the stored iterate, the cost of which is marginal, as a factorisation is already available. The recentering steps proposed are based on centrality correctors [32]. An auxiliary feasibility recovery procedure may be needed as, due to the addition of cuts, large infeasibilities are often produced.

The warm-start approach proposed in [33] is extended in [39] to the case of solving a sequence of problems with the same dimensions but changing problem data (the objective function or the right-hand side). Such situations arise in the context of decomposition approaches for large structured linear programs. In the case of Dantzig–Wolfe decomposition, successive subproblems differ only in the objective function, while in the case of Benders decomposition they differ only in

the right-hand sides. Following [33], nearly optimal points are saved and used to warm-start the solution of subsequent subproblems [39].

Yıldırım and Wright [94] consider again the case of solving a sequence of problems in fixed dimensions, and analyse the number of iterations required to converge to a solution of the perturbed problem instance from the warm-start point. They obtain worst-case estimates and show that these estimates depend on the size of the perturbation as well as on the conditioning of the problem instances. Thus they obtain conditions under which the complexity of the warm-start approach is better than in the cold-start case.

The strategy proposed in [94] aims to absorb the primal and dual infeasibilities introduced by the perturbation in just one step. This strategy requires backtracking to an iterate for which μ is large enough to allow a full step for the correction direction they produce. The amount of necessary backtracking depends on the magnitude of the perturbation (as measured by the change in the problem data). This is intuitively justified by considering that a large perturbation will produce a large adjustment. It is also essential to guarantee that a full step will not compromise the positivity requirements of the iterate. To ensure the availability of an approximate μ-center from which the perturbation can be absorbed in one step, a subset of iterates for different values of μ is stored. When the size of the perturbation becomes known, the smallest μ that allows to absorb the perturbation is retrieved, and the corresponding iterate is used as a warm-start point for the next problem in the sequence. In [94], two different corrections for the perturbation are studied; one is based on least squares, the other on a Newton step correction. A detailed computational comparison of these strategies has been carried out by John and Yıldırım [49].

Gondzio and Grothey [34] propose a different reoptimization technique for interior point methods. As in [94], they aim at obtaining conditions for perturbations that can be absorbed in one Newton step. However, they measure perturbations by a relative measure of implied primal and dual infeasibilities, and analyse recovery steps in the primal and the dual spaces independently.

This reoptimization procedure is based on two phases. First, an attempt is made to absorb the infeasibilities caused by the perturbation with a full Newton step; second, the centrality of the iterate is improved. A key feature of Gondzio and Grothey's approach [34] is that the primal search direction is governed only by the primal perturbation, and the dual search direction only by the dual perturbation. This corresponds to splitting the search direction into two terms, $\Delta_1 w$ and $\Delta_2 w$,

and solving two independent Newton systems

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta_1 x \\ \Delta_1 y \\ \Delta_1 s \end{bmatrix} = \begin{bmatrix} \xi_b \\ 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta_2 x \\ \Delta_2 y \\ \Delta_2 s \end{bmatrix} = \begin{bmatrix} 0 \\ \xi_c \\ 0 \end{bmatrix}.$$

They produce bounds on the magnitude of primal residual ξ_b and dual residual ξ_c that can be absorbed in a single Newton step. Unlike to the results of [94], the bounds of [34] are easy to compute and thus can be used in practice. If the residuals at the warm-start point do not satisfy these bounds, a different iterate further away from optimality may be more likely to allow for a full Newton step.

A practical implementation is developed within an infeasible interior point method. In this context, a stronger emphasis may be put on reducing the infeasibilities. This is accomplished with additional centering steps. An approximate μ-center is stored for a tolerance level that depends on the magnitude of the expected perturbation. This does not exclude the possibility of storing a sequence of iterates, as proposed in [94], thus postponing the choice of the one to use. As the problem size does not change, an approximate μ-center can immediately be used as an iterate for the next problem. If only one iterate is stored, then the absorption of infeasibilities may be spread across a few iterations whenever the stepsizes fall below a predefined level. As this strategy does not make assumptions upon the centrality of the warm-start iterate, it can be initialised with any iterate. Gondzio and Grothey [34] apply this warm-start strategy successfully to structured problems for crash-start points that come from a cross-decomposition scheme, and thus may lack centrality.

A different approach has been studied by Benson and Shanno [6]. They investigate how to improve the efficiency of interior point methods in a reoptimization context by the use of a primal–dual penalty approach. While standard penalty techniques are applied only in one space, the introduction of penalty parameters in both the primal and the dual problems allows the handling of perturbations in both spaces. For example, an l_1 penalty can deal with changes in the right-hand side of the constraints (primal changes), but has virtually no effect on perturbations of the objective coefficients (dual changes).

Benson and Shanno's strategy relaxes the non-negativity constraints for the decision variables, penalising the violation in the objective, for both the primal and dual problems. The penalised problem allows the variables to become negative, which provides more freedom of movement for the variables, with the immediate advantage of allowing larger stepsizes along the computed search direction to be accepted. This favours faster progress especially in the first few iterations, when the perturbation needs to be absorbed. Benson and Shanno [6] also provide

some computational evidence of the effectiveness of their strategy.

We will introduce a novel warm-start technique tailored to stochastic linear programming in Chapter 5.

Chapter 4

Weighted corrector directions.

In this chapter we first discuss subspace searches as a way to determine search directions for interior point methods for linear programming. Then we illustrate the theoretical understanding of the cases of failure of Mehrotra's corrector direction, drawing particularly from [14, 15], and we introduce a computationally competitive way to improve the performance of multiple centrality correctors. We present the implementation of such a strategy in the HOPDM solver, along with extensive computational experience. The original content of this chapter has already appeared in [18], co-authored with Jacek Gondzio.

4.1 Subspace searches

Subspace searches are a strategy for generating search directions. They are usually derived differently than the Newton system, and are built according to some criteria that are believed to summarise the essential characteristics of a good search direction. In this section we discuss two subspace-search approaches. The first, by Jarre and Wechs [48], generates directions by a recursion on a modification of Mehrotra's corrector. The second, the approach of Mehrotra and Li [66], considers search directions produced through Krylov subspace basis.

4.1.1 Jarre and Wechs's directions

Jarre and Wechs [48] present an implementable technique for generating efficient higher-order search directions in a primal–dual interior-point framework. They start by considering the Newton system (2.9). While it is clear what to consider as right-hand side for primal and dual feasibility constraints (the residual at the current point), the complementarity component leaves more freedom in choosing a target t. They argue that there exists an optimal choice for which the corresponding Newton system would immediately produce the optimizer. Given the

system

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} b - Ax \\ c - A^T y - s \\ t \end{bmatrix}, \tag{4.1}$$

they ask what choice of t in the right-hand side would produce a good search direction.

If w is the current iterate and w^* is an optimal primal–dual solution to problem (2.1), then $\Delta w^* = w^* - w$ satisfies the system (4.1) for the choice

$$t^* = S(x^* - x) + X(s^* - s) = S\Delta x^* + X\Delta s^*. \tag{4.2}$$

It is possible to write t^* as a different expression. Given that $X^*S^*e = 0$, we have:

$$0 = (X + \Delta X^*)(S + \Delta S^*)e = XSe + X\Delta s^* + S\Delta x^* + \Delta X^*\Delta S^*e,$$

from which, using (4.2), we obtain $t^* = -XSe - \Delta X^*\Delta S^*e$. Therefore, as t^* would lead to the optimizer in one single iteration, it can be considered the optimal choice for the target t in (4.1).

Clearly, as we do not know what the solution will be, when system (4.1) is set up, we have no knowledge of t^*. However, some bounds can be determined. Knowing that $x + \Delta x^* \geq 0$ and $s + \Delta s^* \geq 0$, we have $\Delta x^* \geq -x$ and $\Delta s^* \geq -s$, from which we obtain the lower bound

$$t^* = S\Delta x^* + X\Delta s^* \geq -2XSe.$$

An upper bound is derived from a result by Vavasis and Ye [88, Lemma 16], which states that for two strictly feasible points, w and $\hat{w}$, exactly on the central path and $\mu > \hat{\mu} > 0$ we have

$$s_i\hat{x}_i + x_i\hat{s}_i \leq nx_is_i, \qquad i = 1, \ldots, n,$$

which can be written in vector terms as

$$S\hat{X}e + X\hat{S}e \leq nXSe. \tag{4.3}$$

Therefore, thanks to (4.3) we can rewrite (4.2) and derive the following upper bound:

$$t^* = S(x^* - x) + X(s^* - s) = SX^*e + XS^*e - 2XSe \leq (n-2)XSe.$$

In general, it is not obvious how to find a good t. Jarre and Wechs [48] propose searching a subspace spanned by k different directions, $\Delta w_1, \Delta w_2, \ldots, \Delta w_k$, generated from some affinely independent targets $t_1, t_2, \ldots, t_k$. As the quality of a

search direction can be measured by the length of the stepsize and the reduction in the complementarity gap, they aim to find the combination

$$\Delta w(\rho) = \rho_1 \Delta w_1 + \rho_2 \Delta w_2 + \ldots + \rho_k \Delta w_k \tag{4.4}$$

that maximizes these measures. To generate directions they consider how to use Mehrotra's corrector in an iterative fashion, and notice that a straightforward recursion of the type

$$\begin{aligned} A\Delta x^j &= b - Ax \\ A^T \Delta y^j + \Delta s^j &= c - A^T y - s \\ X\Delta s^j + S\Delta x^j &= \mu e - XSe - \Delta X^{j-1} \Delta S^{j-1} e \end{aligned}$$

is usually divergent for general positive starting points. Therefore, they introduce a rescaling of Mehrotra's corrector and other strategies to guarantee good convergence properties.

In order to find the best combination of weights in the direction (4.4), Jarre and Wechs [48] formulate a small linear subproblem that can be solved approximately. The solution of this subproblem produces a search direction $\Delta w(\rho)$ that is generally better than Mehrotra's predictor–corrector direction, as it takes into account and carefully weighs some additional higher-order directions.

4.1.2 Krylov subspace searches

Mehrotra and Li [66] propose a different scheme in which a collection of linearly independent directions is combined through a small linear subproblem. Following the approach of Jarre and Wechs [48] presented above, they express the requirements for a good search direction as a linear program. In particular, they impose conditions aimed at ensuring global convergence of the algorithm when using generic search directions.

The directions considered in the subspace search can include all sorts of linearly independent directions: the affine-scaling direction, Mehrotra's corrector, multiple centrality correctors, Jarre–Wechs directions. In their approach, Mehrotra and Li [66] consider generating directions using a Krylov subspace mechanism.

This new approach for generating corrector directions uses an exact factorisation from an earlier iteration to generate directions via Krylov subspaces. When generating correctors through Krylov subspaces, the directions have to satisfy the primal–dual feasibility constraints, but not the complementarity constraints. Therefore, any convex combination of these directions will still satisfy the feasibility requirements. This gives the additional freedom of choosing the combination

that satisfies the complementarity conditions in the best possible way. However, Mehrotra and Li [66] do not require the linear combination to be convex.

At the k-th iteration of an interior point method we have to solve the Newton system (2.9). We rewrite it in the following terms

$$J_k \Delta_k w = \xi^k,$$

where $J_k = \nabla F(w_k)$ is the Jacobian matrix for system (2.8) evaluated at the current iterate w^k, and ξ^k is the corresponding right-hand side. The direction $\Delta_k w$ is used to compute a trial point:

$$\tilde{x} = x^k + \alpha_P \Delta_k x, \quad \tilde{y} = y^k + \alpha_D \Delta_k y, \quad \tilde{s} = s^k + \alpha_D \Delta_k s,$$

with stepsizes α_P and α_D computed according to (3.1). At the trial point $\tilde{w}$, a usual interior point method would have to solve the system $\tilde{J}\Delta w = \tilde{\xi}$ in order to find the next search direction. Instead, Mehrotra and Li [66] generate a Krylov subspace for $\tilde{J}\Delta w = \tilde{\xi}$.

The Krylov subspace of dimension j is defined as

$$K_j(J_k, \tilde{J}, \tilde{\xi}) = \text{span}\{\xi_J, G\xi_J, G^2\xi_J, \ldots, G^{j-1}\xi_J\},$$

where $\xi_J = J_k^{-1}\tilde{\xi}$, and $G = I - J_k^{-1}\tilde{J}$. Note that to improve the conditioning in this space the term $\tilde{J}$ is preconditioned with J_k, the factors of which have already been found. The subspace $K_j(J_k, \tilde{J}, \tilde{\xi})$ thus generated contains j linearly independent directions.

At most n linearly independent Krylov directions can be computed. However, for reasons of computational efficiency, not all of them are actually generated. Instead, a small number of search directions are considered, so that the linear subproblem is of small dimensions.

In the algorithm of [66], the affine-scaling direction, $\Delta_a w$, Mehrotra's corrector, $\Delta_c w$, and the first j directions, $\Delta_1 w, \Delta_2 w, \ldots, \Delta_j w$, from $K_j(J_k, \tilde{J}, \tilde{\xi})$ are combined with appropriate weights ρ:

$$\Delta(\rho)w = \rho_a \Delta_a w + \rho_c \Delta_c w + \sum_{i=1}^{j} \rho_i \Delta_i w.$$

In some cases, a pure recentering direction, $\Delta_\mu w$, may also be considered in $\Delta(\rho)w$, with appropriate weight ρ_μ.

The choice of the best set of weights ρ in the combined search direction is obtained by solving an auxiliary linear programming subproblem. The subproblem maximizes the rate of decrease in the duality gap whilst satisfying a series of requirements: non-negativity of the new iterate, upper bounds on the magnitude

of the search direction, upper bounds on infeasibilities, decrease in the average complementarity gap, and closeness to the central path. Both the theoretical formulation and the practical implementation allow for the independent choice of weights ρ for the primal and dual spaces, which is often a desirable property for practical algorithms.

The computation of Krylov subspace directions in Mehrotra and Li's approach involves considerable computational cost, as the computation of each Krylov direction requires a backsolve operation. This can be seen from the definition of the power basis matrix

$$G = I - J_k^{-1}\tilde{J},$$

which involves an inverse matrix. In fact, if we take u to be the starting vector in the Krylov sequence, the computation of the vector $J_k^{-1}\tilde{J}u$ requires us to first compute $v = \tilde{J}u$ (matrix–vector multiplication) and then to determine $t = J_k^{-1}v$ (backsolve on the Cholesky factors).

Mehrotra and Li [66] present extensive computational experience for their strategy, showing that the use of up to four Krylov subspace directions embedded in a small linear program results in an algorithm with very good performance. We will discuss these results in more details in Section 4.4, where we compare them with the weighted correctors strategy of Section 4.3.

4.2 Failures of Mehrotra's corrector

As presented in Section 2.2, Newton's method employed in a primal–dual path-following algorithm provides a first-order approximation of the central path, in which the nonlinear perturbed KKT system (2.4) is linearised around the current primal–dual point w^k. Consistently with the standard analysis of Newton's method [29], this linear approximation is valid locally, in a neighbourhood of the point where it is computed. Depending on the specific characteristics of the point, such an approximation may not be a good direction if used outside this neighbourhood.

Mehrotra's algorithm, which we examined in Section 3.2, adds a second-order correction to the search direction in order to construct a quadratic approximation of the central path. This technique is at the basis of all implementations of interior point methods for linear programming, and the practical superiority of a second-order algorithm over a first-order one is broadly recognised. However, it is important to remember that the central path is a highly nonlinear curve that, according to Vavasis and Ye [88], is composed by $O(n^2)$ turns of a high degree and segments in which it is approximately straight. Given the complexity of this

curve, it is unrealistic to be able to approximate it everywhere with a second-order curve.

Upon discussing the choice for the centering term in his algorithm, Mehrotra [65] makes this comment:

> "*It is not clear if the central path [...] is the best path to follow, particularly since it is affected by the presence of redundant constraints. Furthermore, the points on (or near) the central path are only intermediate to solving the linear programming problem. It is only the limit point on this path that is of interest to us.*"

This statement represents a very pragmatic viewpoint of the interior point algorithm. It stresses the fact that while the central path is the reference trajectory for approaching the optimal set, in practice we should not worry too much if the algorithm strays from it. Also it raises a warning about trusting the concept of analytic center too much, as it can easily be distorted by the presence of redundant constraints, as discussed in Section 2.1.1.

4.2.1 The implementation viewpoint

In practical implementations, it is sometimes observed that Mehrotra's corrector produces a shorter stepsize than the one obtained by the predictor direction. This can be considered to be a failure of the corrector, as it does not improve the quality of the predictor direction, as measured by the length of the achievable step. In such situations, a solver may decide to reject Mehrotra's corrector and only use some multiple centrality correctors. As these generally construct less demanding target points, in which only the complementarity products that lie outwith the symmetric neighbourhood are corrected, the chances of them being rejected are smaller. However, if these do not help either, the search direction may be reduced to the affine-scaling predictor direction alone. Undoubtedly such an occurrence has a dramatic impact on the efficiency of the solver, as a lot of CPU time is fruitlessly consumed in trying to find some correction to the predictor direction. If the step from the current point is too short, the new iterate may still lie in a troublesome area of the solution space, and the same situation may arise again in the following iteration.

In Table 4.1 we provide some evidence gathered on the collection of Netlib and Kennington problems by running HOPDM 2.13 [31]. The table lists the problems in which Mehrotra's corrector is rejected. HOPDM's strategy can be summarised in the following steps. First both the predictor and the corrector direction are computed according to Mehrotra's algorithm (see Algorithm 3.1). If the step in the composite direction $\Delta_M w$ is shorter than what would be achieved in the

predictor $\Delta_a w$, a warning is raised (reported in the first column of Table 4.1). At this point some multiple centrality correctors $\Delta_m w$ are computed: if they do not improve on the stepsize, a second warning is raised (reported in the second column of Table 4.1), and the algorithm moves along the affine-scaling direction $\Delta_a w$ alone.

Problem	Δ_c failed	Δ_m failed
cre-a	2	1
cre-b	1	1
degen3	1	1
dfl001	12	10
ganges	1	1
greenbea	9	3

Problem	Δ_c failed	Δ_m failed
ken-13	1	0
maros	1	0
modszk1	2	1
osa-07	1	0
pds-10	1	0
shell	1	0

Table 4.1: Number of times Mehrotra's corrector Δ_c and multiple centrality correctors Δ_m were rejected in HOPDM.

As we can see from Table 4.1, in the HOPDM implementation Mehrotra's corrector is rejected only in 12 problems. Two problems appear particularly troublesome: `dfl001` and `greenbea`. In 15 iterations, multiple centrality correctors provide a successful correction to the predictor direction; however, in 18 cases (10 of which are in problem `dfl001`) the search direction was the affine-scaling predictor alone.

4.2.2 Theoretical insight

The issue of failures of Mehrotra's corrector has recently been analysed by Cartis [14], who provided an example in which the second-order corrector does not behave well. We report it here and make some observations.

Example 4.1 (Cartis [14])**.** Consider the following primal–dual pair:

$$\begin{array}{lll} \min & x_1 + 8x_2 & \\ \text{s.t.} & x_2 + x_3 = 2, & \\ & x \geq 0; & \end{array} \qquad \begin{array}{lll} \max & 2y & \\ \text{s.t.} & s_1 = 1, \quad y + s_2 = 8, & \quad y + s_3 = 0, \\ & s \geq 0, & \end{array}$$

and the starting point:

$$x^0 = (8,\, 1.95,\, 0.05), \quad y^0 = -0.1, \quad s^0 = (1,\, 8.1,\, 0.1).$$

The solution is:

$$x^* = (0,\, 0,\, 2), \quad y^* = 0, \quad s^* = (1,\, 8,\, 0).$$

The first observation we make about the starting point concerns centrality. The vector of complementary pairs and the average complementarity gap are

$$X^0 S^0 e = \begin{bmatrix} 8 \\ 15.795 \\ 0.005 \end{bmatrix}, \qquad \mu^0 = 7.933.$$

There is a big spread among the complementarity products: while the first is almost centered, the second is about twice the average, and the third is very close to zero. The ratio (2.14) between the smallest and the largest pair is

$$\varrho(X^0 S^0 e) \approx 3 \times 10^{-4}.$$

Studying this starting point in terms of a symmetric neighbourhood, we can see that the pair $x_2^0 s_2^0 \in \mathcal{N}_s(0.5)$, but this neighbourhood does not accommodate $x_3^0 s_3^0$. In fact, this pair belongs to the much larger neighbourhood $\mathcal{N}_s(0.0006)$.

Cartis's analysis is based on a feasible primal–dual corrector algorithm (PDC) which combines a standard primal–dual path-following method with a second-order correction. In this algorithm, the predictor direction is computed according to (2.9) for some centering parameter $\sigma \in (0, 1)$, and the corrector solves (2.9) this time with right-hand side

$$\begin{bmatrix} 0 \\ 0 \\ -\Delta_a X \Delta_a S e \end{bmatrix}.$$

The PDC algorithm is not exactly the same as Mehrotra's predictor–corrector algorithm, but its aims are very similar and it is more amenable to theoretical analysis.

Cartis's example [14] shows that for certain starting points the corrector direction, $\Delta_c w$, is always orders of magnitude larger than the predictor $\Delta_a w$, in both the primal and dual spaces. In these cases, while the predictor points towards the optimum, the second-order corrector points away from it. As the final direction is given by

$$\Delta w = \Delta_a w + \Delta_c w,$$

the combined direction is influenced almost exclusively by the corrector, and therefore it is not accurate. The stepsizes generated by moving along the direction Δw are extremely short.

The solution outlined by Cartis in [14], and then further developed in [15], is to reduce the influence exerted by the corrector by weighting it by the square of the stepsize. Besides ensuring that the corrector is really considered a second-order term as in a rigorous Taylor expansion, this proposal allows for better convergence results.

An implementable solution to these problems will be the major focus of the next section.

4.3 Weighting the corrector directions

The discussion of the previous section clarifies the fact that using a full weight for a corrector direction may not always be desirable. The theoretical findings outlined above give rise to the following generalisation

$$\Delta_M^{\omega} w = \Delta_a w + \omega \Delta_c w,$$

where we weight the corrector by a parameter $\omega \in (0, 1]$ independent of the stepsize α.

We choose the weight independently at each iteration in order to maximize the stepsize in the weighted composite direction $\Delta_M^{\omega} w$. This gives us the freedom to find the optimal weight $\hat{\omega}$ in the interval $(0, 1]$. This generalisation allows for the possibility of using Mehrotra's corrector with a small weight, if that helps in producing a better stepsize; on the other hand, the choice $\hat{\omega} = 1$ yields the full Mehrotra corrector of the standard implementations.

We apply the weighting strategy to multiple centrality correctors as well. As discussed in Section 3.3, a trial point in the multiple centrality correctors technique depends on the aspiration level δ which measures the greediness of the centrality corrector. In the existing implementations, this parameter is fixed at coding time to a value determined after tuning to a series of representative test problems. However, for a specific problem such a value may be too conservative or too demanding; moreover, the same value may not be optimal throughout the solution of the same problem. Hence, it makes sense to provide a mechanism that changes these correctors in an adaptive way in order to increase their effectiveness.

We propose to generate a sequence of multiple centrality correctors, and for each of them we choose the optimal weight $\hat{\omega}$ that maximizes the stepsizes in primal and dual spaces for the composite direction

$$\Delta_m^{\omega} w = \Delta_p w + \omega \Delta_m w.$$

Direction $\Delta_p w + \hat{\omega} \Delta_m w$ becomes a predictor $\Delta_p w$ for the next centrality corrector. The correcting process is iterative, and can be interrupted at any stage.

It is worthwhile noting that eventually, after adding k corrector terms, the directions used in the weighted correctors approach have form

$$\Delta w = \Delta_a w + \omega_1 \Delta_1 w + \ldots + \omega_k \Delta_k w,$$

and the affine-scaling term $\Delta_a w$ contributes to it without any reduction. Hence, the larger the stepsize, the more progress we make towards the optimizer.

We formalise the weighted correctors strategy in Algorithm 4.1.

Algorithm 4.1 Weighted correctors algorithm

Given: An initial iterate w^0 such that $(x^0, s^0) > 0$;

Repeat:

Solve system (2.9) with right-hand side r_a (3.6) for the affine-scaling predictor direction $\Delta_a w$.

Set μ according to (3.8) and find Mehrotra's corrector direction $\Delta_c w$ by solving system (2.9) with right-hand side r_c (3.10).

Do a linesearch to find the optimal $\hat{\omega}$ that maximizes the stepsize α in $\Delta_M^{\omega} w = \Delta_a w + \omega \Delta_c w$

Set $\Delta_p w := \Delta_a w + \hat{\omega} \Delta_c w$.

while the stepsize increases by at least a fraction of the aspiration level δ **do**

Solve system (2.9) with right-hand side r_m (3.15), for a centrality corrector direction $\Delta_m w$.

Perform a linesearch to find the optimal $\hat{\omega}$ that maximizes the stepsize α in $\Delta_m^{\omega} w = \Delta_p w + \omega \Delta_m w$.

Set $\Delta_p w := \Delta_p w + \hat{\omega} \Delta_m w$.

end while

Update the iterate $w^{k+1} = w^k + \alpha_k \Delta_p w$.

Until The termination criteria (3.2) are met for some predetermined value of p and q.

In the choice of ω we restrict our attention to the interval

$$[\omega_{\min}, \omega_{\max}] = [\alpha_P \alpha_D,\ 1].$$

There are two reasons for using $\omega_{\min} = \alpha_P \alpha_D$. First, using the stepsizes α_P and α_D for the predictor direction gives

$$(X + \alpha_P \Delta X)(S + \alpha_D \Delta S)e \ = \ XSe + \alpha_P S \Delta X e + \alpha_D X \Delta S e + \alpha_P \alpha_D \Delta X \Delta S e,$$

and the term $\alpha_P \alpha_D$ appears with the second-order error. Secondly, the study of Cartis [14] suggests squaring the stepsize for the corrector. Our computational experience indicates that the use of $\omega = \omega_{\min} = \alpha_P \alpha_D$ is too conservative. Still, as can be seen from Figure 4.1, such $\omega_{\min}$ is a reliable lower bound for attractive weights ω.

The ultimate objective in choosing ω is to increase the current stepsizes α_P and α_D. These stepsizes depend on ω in a complex way. Examples corresponding to a common behaviour are given in Figure 4.2, where we show how the product $\alpha_P^{\omega} \alpha_D^{\omega}$ varies depending on the choice of ω for Mehrotra's corrector at two different iterations of problem `capri` of the Netlib collection. On the left, $\omega \in [0.4, 1]$ and $\hat{\omega} = 0.475$ gives a product $\alpha_P^{\hat{\omega}} \alpha_D^{\hat{\omega}} = 0.583$, better than a value of 0.477 that would

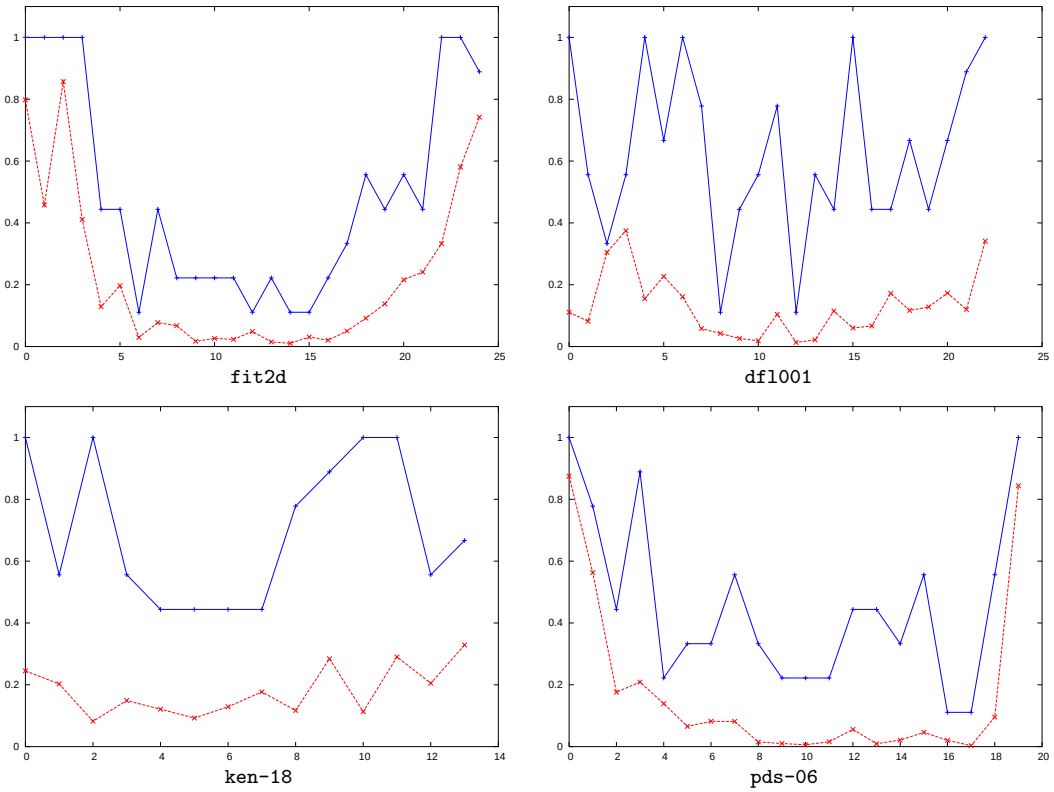

Figure 4.1: Relationship between $\omega_{\min} = \alpha_P \alpha_D$ (dashed) and $\hat{\omega}$ (solid) through the iterations of four different problems.

have been obtained by using a full weight on Mehrotra's corrector. On the right, $\omega \in [0.178, 1]$ and the choice of $\omega \in (0.6, 0.7)$ leads to the best product $\alpha_P^{\hat{\omega}} \alpha_D^{\hat{\omega}}$ of about 0.375.

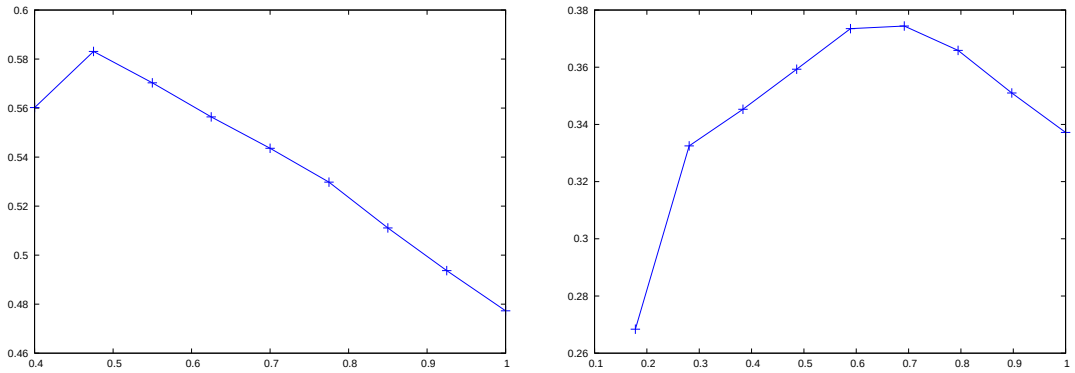

Figure 4.2: Plot of $\alpha_P^{\omega} \alpha_D^{\omega}$ for 9 values of $\omega \in [\alpha_P \alpha_D, 1]$ in two iterations of problem `capri`.

4.4 Numerical results

We have implemented our proposal within the HOPDM interior point solver [31], and tested it in a series of computational experiments, using test problems from

different collections. As a comparison, we present the results obtained by PCx [20], a reference implementation of interior point methods. The two implementations, PCx and HOPDM, have different termination criteria in their default configurations. For the purpose of consistency, in the results shown, we decided to implement in HOPDM the criteria used by PCx. Therefore, for all solvers optimal termination occurs when the conditions (3.3) are met, with feasibility accuracy $p = 8$ and complementarity accuracy $q = 10$.

We use $\gamma = 0.1$ in the definition of the symmetric neighbourhood and define aspiration levels for the stepsizes using the rule

$$\tilde{\alpha}_P = \min(1.5\alpha_P + 0.3,\ 1) \quad \text{and} \quad \tilde{\alpha}_D = \min(1.5\alpha_D + 0.3,\ 1).$$

The values suggested in [32] were more conservative: $\tilde{\alpha} = \min(\alpha + 0.1,\ 1)$. Thanks to the weighting mechanism we can control the contribution of the corrector in an adaptive way, and thus be more demanding in our definition of aspired stepsizes.

Centrality correctors are accepted in the primal and/or dual space if $\alpha_P^{new} \geq 1.01\alpha_P$ and/or $\alpha_D^{new} \geq 1.01\alpha_D$, respectively. Note that this is different from what is described in [32], where a corrector is accepted if $\alpha^{new} \geq \alpha + 0.1\delta$, for an aspiration level $\delta = 0.1$.

We implemented a simple linesearch procedure to locate $\hat{\omega}$. We choose 9 points uniformly spaced in the interval $[\alpha_P\alpha_D, 1]$ and evaluate, for each of these points, the stepsizes α_P^ω and α_D^ω in both spaces. When a larger stepsize α_P^ω or α_D^ω is obtained, the corresponding ω is stored as ω_P or ω_D respectively. Hence, in our implementation we allow different weightings for directions in the primal and dual spaces.

We present our results in terms of number of iterations and number of backsolve operations. The rationale behind this decision is that the multiple centrality correctors technique determines the number of allowed correctors on the basis of the ratio between factorisation cost and backsolve cost. This ratio can be very different across implementations, and is mainly influenced by the linear algebra routines used. HOPDM comes with an in-house linear algebra implementation, while PCx relies on the more sophisticated sparse Cholesky solver of Ng and Peyton. Therefore, the PCx code tends to use less correctors per iteration. In all other respects, the correction scheme closely follows the one of HOPDM and the paper [32].

4.4.1 Comparison with Mehrotra and Li's approach

We first considered the test set used in [66]. It contains 101 problems from the Netlib and Kennington collections. We will refer to is as Mehrotra and Li's test

collection. In Table 4.2 we present the computational comparison outlining the total number of iterations and the total number of backsolves necessary to solve the problems in this test set. Column HO displays the results obtained by the previous implementation of HOPDM, while column dHO reports the results obtained by the current implementation of weighted correctors with a default choice of the number of centrality correctors. The last column presents the relative change between the two versions of HOPDM tested. As a reference, we also report in this table the overall statistics of PCx (release 1.1) on these problems. We found the number of backsolves by counting the number of calls to the functions `IRSOLV()` and `EnhanceSolve()`, for HOPDM and PCx respectively.

	PCx	HO	dHO	Change
Iterations	2114	1871	1445	-22.77%
Backsolves	4849	6043	5717	-5.39%
Backsolves/iter.	2.29	3.23	3.95	+22.29%

Table 4.2: Overall results obtained on Mehrotra and Li's test collection.

From Table 4.2 we first observe the very small number of backsolves per iteration needed by PCx. This is due to the fact that PCx allows the use of Gondzio's multiple centrality correctors only in 4 problems: `dfl001`, `maros-r7`, `pds-10` and `pds-20`. Also we notice that when we allow an adaptive weighting of the correctors there is a tendency to use more correctors per iteration than previously. This happens because the weighting mechanism makes it more likely to accept some correctors that would otherwise have been rejected as too aggressive. While this usually leads to a decrease in iteration count, it also makes each iteration more expensive, as more backsolves are performed.

In Table 4.3 we detail the problems for which we obtained savings in computational time. Given the small dimension of most of the problems in the Netlib collection, we did not expect major savings. However, as the problem sizes increase, we can see that the proposed way of evaluating and weighting the correctors pays off. This led us to investigate further the performance of the proposed implementation, which we will discuss in Section 4.4.2.

Problem	HO	dHO	Problem	HO	dHO
bnl1	0.36	0.25	pilot87	12.62	11.88
dfl001	150.63	114.80	pds-06	24.59	21.31
maros-r7	7.76	7.52	pds-10	96.57	79.29
pilot	5.23	4.35	pds-20	923.71	633.64

Table 4.3: Problems that showed time savings (times are in seconds).

We were particularly interested in comparing the results produced by our weighted correctors approach with those obtained by the subspace search scheme

of Section 4.1.2 and published in [66]. In the tables of results presented in [66], the best performance in terms of iteration count is obtained by PCx-4, which uses 4 Krylov subspace vectors. These directions are combined with an affine-scaling predictor direction and Mehrotra's second-order correction, leading to 6 backsolves per iteration. This number increases when the linear subproblem produces an optimal objective value smaller than a specified threshold or the new iterate fails to satisfy some neighbourhood condition. In these cases the pure centering direction $\Delta_\mu w$ also needs to be computed, and a seventh backsolve is performed.

In Table 4.4 we present a full comparison of the weighted correctors and Mehrotra and Li's strategies. In columns HO-0 and HO-4, we present the results obtained by HOPDM when forcing the use of 0 and 4 multiple centrality correctors. In the column called HO-∞ we report the results obtained when an unlimited number of correctors is allowed (in practice we allow no more than 20 correctors). The last column, labelled dHO, presents the results obtained by the default way of choosing the number of correctors allowed. As we understand the paper [66], PCx-0 uses exactly 2 backsolves per iteration: one to compute the affine-scaling direction, another to compute Mehrotra's corrector; PCx-4 computes four additional Krylov vectors, hence it uses 6 backsolves per iteration. Consequently, up to 2 and 6 backsolves per iteration are allowed in PCx-0 and PCx-4 and in HO-0 and HO-4, respectively.

The number of backsolves reported for HOPDM includes two needed by the initialisation procedure: the number of backsolves should not exceed $2\times$Its$+2$, and $6\times$Its$+2$ respectively for HO-0 and HO-4, where Its is the number of iterations. The observed number of backsolves is often much smaller than these bounds because the correcting mechanism switches off when the stepsizes are equal to 1 or when the corrector does not improve the stepsize. Problem `afiro` solved by HO-4 needs 24 backsolves, 22 of which compute different components of directions, hence the average number of backsolves per iteration is only 22/6 and is much smaller than 6. Occasionally, as a consequence of numerical errors, certain components of direction are rejected on the grounds of insufficient accuracy. In such cases, the number of backsolves may exceed the stated upper bounds. The reader may observe, for example, that `pilot4` is solved by HO-4 in 16 iterations, but the number of backsolves is equal to 100 and exceeds $6 \times 16 + 2 = 98$.

The version HO-∞ requires 1139 iterations to solve the collection of 101 problems, an average of just above 11 iterations per problem. This version has only an academic interest, yet it reveals a spectacular efficiency of interior point methods which can solve difficult linear programs of medium sizes (reaching a couple of

hundred thousand variables) in just a few iterations. In particular, it suggests that if we had a cheap way of generating search directions, then it would be beneficial to have as many as possible.

4.4.2 Larger test problems

We have applied our algorithm to examples from other test collections besides Netlib. We used a collection of medium to large problems taken from different sources. Problems `CH` through `CO9` are MARKAL (market allocation) models; `mod2` through `worldl` are agricultural models used earlier in [32]; problems `route` through `rlfdual` can be retrieved from

`http://www.sztaki.hu/~meszaros/public_ftp/lptestset/misc/`,

and problems `neos1` through `fome13` can be retrieved from

`ftp://plato.asu.edu/pub/lptestset/`.

In Table 4.5 we detail the sizes of these problems and provide a time comparison between our previous implementation (shown in column HO), and the current one (in column dHO). This test collection contains problems large enough to show a consistent improvement in CPU time. We only recorded a deterioration of the performance by more than 1 second in 4 problems (`mod2`, `dbc1`, `watson-1`, `sgpf5y6`). The improvements are significant on problems with a large number of nonzero elements. In these cases, dHO produces savings ranging from about 10% to 30%, with remarkable results in `rail2586` and `rail4284`, for which the relative savings reach 45% and 65%, respectively.

In Figure 4.3, we show the CPU-time based performance profile [24] for the two algorithms. This graph shows the proportion of problems that each algorithm has solved within τ times of the best. Hence, for $\tau = 1$ it indicates that dHO has been the best solver on 72% of the problems, against 28% for HO. For larger values of τ, the performance profile for dHO stays above the one for HO, thus confirming its superiority. In particular, dHO solves all problems within 1.3 times of the best.

More numerical results obtained on some collections of stochastic programming problems and quadratic problems can be found in [18].

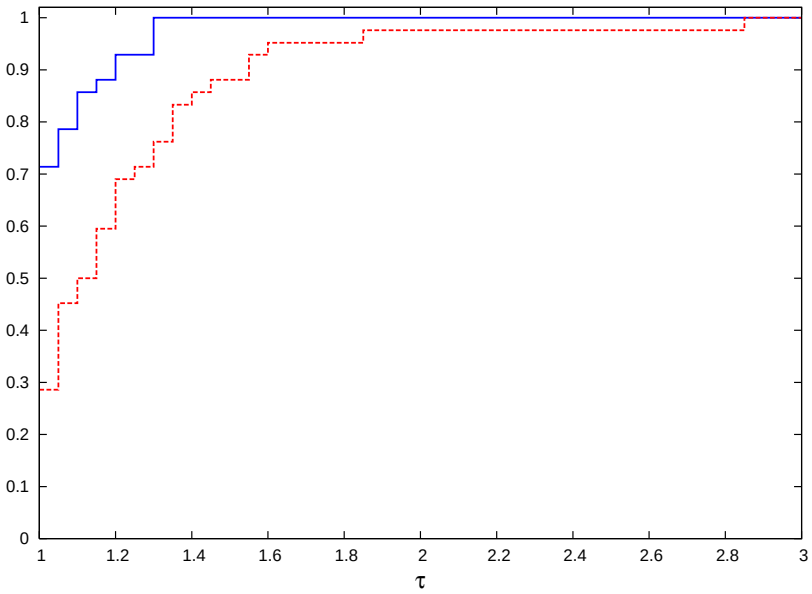

Figure 4.3: Performance profiles for HO (dashed) and dHO (solid) on the set of problems of Table 4.5.

4.5 Tables of results

Mehrotra and Li's test collection

Problem	PCx0	HO-0		PCx4	HO-4		HO-∞		dHO	
	Its	Its	Bks	Its	Its	Bks	Its	Bks	Its	Bks
25fv47	25	27	55	15	15	95	14	184	18	76
80bau3b	36	31	64	22	15	93	13	195	18	78
adlittle	11	12	25	8	9	47	9	91	10	29
afiro	7	8	13	6	6	24	6	35	7	15
agg	18	21	43	10	13	79	12	143	14	57
agg2	22	22	46	15	14	86	12	151	15	61
agg3	21	20	41	14	15	84	14	147	15	62
bandm	17	14	30	11	10	58	9	106	11	34
beaconfd	10	10	19	7	8	40	9	82	8	23
blend	9	10	20	7	8	42	8	61	8	23
bnl1	36	39	80	27	17	107	17	162	25	91
bnl2	32	25	51	19	16	93	13	171	16	93
boeing1	20	20	40	12	13	76	11	142	15	47
boeing2	12	14	28	10	11	61	10	117	12	34
bore3d	15	15	28	11	11	60	10	102	12	33
brandy	19	19	38	13	14	82	12	160	17	50
capri	18	19	38	12	11	63	11	121	14	43
cycle	23	27	55	13	18	104	15	165	19	83
czprob	27	23	46	16	14	82	13	176	19	60
d2q06c	29	31	64	18	17	105	14	201	17	106
d6cube	19	17	35	11	12	64	11	139	12	64
degen2	12	14	29	8	10	62	9	100	11	44
degen3	16	20	42	10	14	84	11	121	14	84

Table 4.4: Comparison with Mehrotra and Li's algorithm.

Problem	PCx0	HO-0		PCx4	HO-4		HO-∞		dHO	
	Its	Its	Bks	Its	Its	Bks	Its	Bks	Its	Bks
dfl001	47	46	97	31	26	152	25	329	22	222
e226	17	19	40	11	14	87	12	132	16	49
etamacro	23	20	42	14	12	73	12	141	13	52
fffff800	27	30	62	16	17	108	17	244	19	84
finnis	23	21	42	14	29	169	17	197	26	81
fit1d	17	16	34	11	12	75	11	148	15	47
fit1p	17	18	38	12	12	72	11	156	15	47
fit2d	22	29	60	12	17	107	12	160	24	83
fit2p	20	24	49	12	15	90	12	150	17	54
forplan	22	19	39	13	13	80	13	145	15	46
ganges	19	12	26	11	8	50	8	93	10	32
gfrd-pnc	18	14	27	11	9	53	8	91	11	32
greenbea	36	37	76	28	23	146	20	263	22	98
greenbeb	32	41	84	25	22	139	19	242	28	99
grow15	18	11	22	13	8	50	7	94	9	26
grow22	21	11	22	15	8	49	7	66	9	28
grow7	15	11	22	11	8	46	7	57	9	26
israel	19	21	44	12	13	81	11	135	16	52
kb2	12	12	24	8	9	47	8	75	10	29
lotfi	14	14	28	9	9	47	9	77	11	31
maros-r7	17	15	30	11	11	62	9	114	10	68
maros	19	20	41	12	13	79	11	136	16	51
nesm	25	27	56	17	14	87	12	153	17	72
perold	32	24	49	21	14	87	13	172	16	69
pilot	36	27	56	22	15	95	14	170	15	95
pilot4	68	30	62	53	16	100	17	228	19	84
pilotja	29	30	62	21	15	95	14	179	16	84
pilotnov	17	15	30	11	10	53	9	87	10	46
pilotwe	48	30	62	27	15	94	15	204	19	83
pilot87	34	31	64	19	16	99	14	220	15	125
recipe	9	8	15	7	7	35	7	49	7	18
scagr25	16	16	32	11	9	52	8	92	11	35
scagr7	14	12	23	9	10	56	9	82	11	32
scfxm1	18	17	35	11	10	63	9	112	14	43
scfxm2	19	19	40	12	11	70	10	140	15	46
scfxm3	21	19	40	12	12	73	10	128	15	47
scrs8	21	17	35	13	11	72	10	112	13	40
scsd1	9	8	15	7	7	30	7	54	7	18
scsd6	12	10	20	9	8	41	8	68	9	24
scsd8	12	11	21	8	7	33	7	60	9	23
sctap1	16	17	33	9	11	58	11	114	13	37
sctap2	13	15	29	9	10	51	8	86	12	33
sctap3	13	15	30	10	11	51	8	87	12	33
seba	14	9	15	9	7	33	7	114	8	19
share1b	19	21	43	12	13	82	12	129	17	54
share2b	17	15	29	12	9	43	8	72	12	35
shell	20	21	42	13	12	73	10	128	14	44
ship04l	12	11	24	8	8	49	8	118	10	31

Table 4.4: Comparison with Mehrotra and Li's algorithm.

Problem	PCx0	HO-0		PCx4	HO-4		HO-∞		dHO	
	Its	Its	Bks	Its	Its	Bks	Its	Bks	Its	Bks
ship04s	13	12	26	8	9	53	8	104	10	31
ship12l	14	13	28	11	8	47	8	103	11	34
ship12s	12	14	29	8	9	51	8	102	11	34
sierra	19	18	38	11	11	66	10	104	14	43
stair	14	18	36	11	16	91	11	145	12	46
standata	13	16	30	8	12	61	10	117	13	35
standmps	22	21	40	16	14	78	13	147	16	46
stocfor1	11	12	22	7	8	43	8	77	9	26
stocfor2	20	19	38	13	12	74	10	151	15	46
truss	20	17	35	12	10	61	9	113	11	46
tuff	19	15	30	12	9	51	9	98	10	39
vtpbase	11	11	20	7	9	40	8	103	9	25
wood1p	19	22	43	14	14	85	13	162	16	65
woodw	30	29	59	19	16	99	14	167	18	77
cre-a	24	25	52	15	15	97	12	143	18	59
cre-b	40	42	86	23	24	148	16	219	22	99
cre-c	25	28	57	15	16	98	14	184	19	63
cre-d	39	43	88	26	22	137	17	233	22	100
ken-07	15	12	25	10	8	48	7	107	10	31
ken-11	21	15	32	13	10	63	9	133	13	40
ken-13	28	19	40	16	11	69	11	138	15	47
ken-18	30	24	49	21	13	78	9	139	15	63
osa-07	18	23	47	12	13	73	11	111	13	41
osa-14	19	19	39	18	12	73	11	106	15	47
osa-30	24	20	40	18	15	93	13	147	17	60
osa-60	22	20	41	15	12	70	11	118	14	46
pds-02	25	19	38	15	14	77	11	124	15	60
pds-06	35	27	56	23	17	100	15	177	17	118
pds-10	41	32	66	27	20	120	18	199	18	155
pds-20	58	48	98	34	23	136	21	253	21	198
Totals	2,194	2,047	4,169	1,438	1,289	7,608	1,139	13,699	1,445	5,717

Table 4.4: Comparison with Mehrotra and Li's algorithm.

Other large LP problems

Problem	Rows	Columns	Nonzeros	HO	dHO	Change
CH	3,852	5,062	42,910	1.03	1.23	19.4%
GE	10,339	11,098	53,763	5.72	5.46	-4.5%
NL	7,195	9,718	102,570	4.37	3.95	-9.6%
BL	6,325	8,018	58,607	2.15	2.14	-0.5%
BL2	6,325	8,018	58,607	2.35	2.31	-1.7%
UK	10,131	14,212	128,341	2.48	3.21	29.4%
CQ5	5,149	7,530	83,564	2.54	2.60	2.4%
CQ9	9,451	13,778	157,598	9.67	8.84	-8.6%
CO5	5,878	7,993	92,788	3.16	3.59	13.6%
CO9	10,965	14,851	176,346	21.10	15.35	-27.3%
mod2	35,664	31,728	220,116	20.59	21.68	5.3%

Table 4.5: Time comparison on larger problems (times are in seconds).

Problem	Rows	Columns	Nonzeros	HO	dHO	Change
world	35,510	32,734	220,748	26.35	23.41	-11.2%
world3	36,266	33,675	224,959	31.13	27.49	-11.7%
world4	52,996	37,557	324,502	73.21	56.14	-23.3%
world6	47,038	32,670	279,024	39.33	32.79	-16.6%
world7	54,807	37,582	333,509	43.14	36.02	-16.5%
worldl	49,108	33,345	291,942	43.95	36.82	-16.2%
route	20,894	23,923	210,025	40.92	33.78	-17.4%
ulevi	6,590	44,605	162,207	9.04	9.55	5.6%
ulevimin	6,590	44,605	162,207	16.52	16.46	-0.4%
dbir1	18,804	27,355	1,067,815	162.18	146.51	-9.7%
dbir2	18,906	27,355	1,148,847	208.93	156.11	-25.3%
dbic1	43,200	183,235	1,217,046	72.96	77.31	5.9%
pcb1000	1,565	2,428	22,499	0.26	0.33	26.9%
pcb3000	3,960	6,810	63,367	1.13	1.16	2.7%
rlfprim	58,866	8,052	265,975	15.63	15.08	-3.5%
rlfdual	8,052	66,918	328,891	71.17	49.79	-30.0%
neos1	131,581	1,892	468,094	169.11	141.89	-16.1%
neos2	132,568	1,560	552,596	113.86	86.13	-24.4%
neos3	132,568	1,560	552,596	132.02	120.59	-8.7%
neos	479,119	36,786	1,084,461	1,785.80	1,386.58	-22.4%
watson-1	201,155	383,927	1,053,564	138.60	166.21	19.9%
sgpf5y6	246,077	308,634	902,275	49.58	64.45	30.0%
stormG2-1000	528,185	1,259,121	4,228,817	1,661.54	1,623.19	-2.3%
rail507	507	63,009	472,358	9.77	10.10	3.4%
rail516	516	47,311	362,207	7.59	5.89	-22.4%
rail582	582	55,515	457,223	9.67	9.60	-0.7%
rail2586	2,586	920,683	8,929,459	1,029.36	566.82	-44.9%
rail4284	4,284	1,092,610	12,372,358	2,779.63	978.48	-64.8%
fome11	12,142	24,460	83,746	407.20	265.21	-34.9%
fome12	24,284	48,920	167,492	766.96	508.61	-33.7%
fome13	48,568	97,840	334,984	1,545.05	990.62	-35.9%

Table 4.5: Time comparison on larger problems (times are in seconds).

Chapter 5

Warm-start strategies for stochastic linear programs.

In this chapter we present the relevant concepts of stochastic programming, stressing the fact that they lead to highly structured problems. We develop an efficient way of constructing an advanced starting point that exploits the inherent structure of a stochastic programming problem. We present our analysis of the warm-start strategy and some very promising computational results. The results presented in this chapter have been the subject of joint work with Jacek Gondzio and Andreas Grothey [19].

5.1 Stochastic programming

Stochastic programming [10, 51] is a technique to help decision-making in many areas of applied mathematics, logistics, engineering, economics and finance where some parameters are unknown.

By stochastic programming, we mean decision and control models in which data evolves over time, and are subject to significant uncertainty. Uncertainty in the data is a commonly observed phenomenon in optimization problems coming from real-life applications. Uncertainty affects problems that aim to plan future actions based on forecasted prices or costs. It can be argued that nearly all practical optimization problems display uncertainty in the data, even if this is not made explicit in the chosen solution method.

When the uncertainty cannot be conveniently forecast, the use of deterministic models is considered inadequate for decision making. In these situations, being able to describe and model the uncertain parameters becomes a requirement for robust decision making. Stochastic programming is the discipline that studies the methods and provides the tools for modelling uncertainty.

Stochastic programming models uncertainty through the analysis of possible

future outcomes (scenarios). As the robustness of the decisions taken increases with the detail of the description, this encourages the generation of very large scenario trees. Stochastic programming aims to take all possible future scenarios into account, weighting them with their respective probabilities. Its strength lies in the adaptability which allows expression of preferences such as restricting exposure to risk. Unlike alternative approaches, it allows modelling of situations in which possible future events are correlated or follow a time-structure, in that realisations of some random process become known in stages and it is possible to react to observed events.

The stochastic programming paradigm is perceived to have some weaknesses. In particular, these relate to the need for reliable forecasts of the probabilities of the future events under consideration (which may not be available), and the fact that taking into account a large number of possible scenarios (especially when applied to multi-stage models) leads to the generation of large-scale structured optimization problems, the solution of which is challenging. With the growing industrial acknowledgement of the benefits of considering uncertainty for planning purposes, it is expected that the need for solving very large problem instances will grow as well.

As the dimensions of the problems increase, the computational advantages of relying on interior point solvers become more and more evident. Very-large-scale problems, however, create more than one difficulty to general purpose solvers. Problems of such sizes can be solved provided that they are both sparse and structured. If that is the case, then the structure present in the matrix should be effectively exploited. Easier access to powerful parallel machines leads to a further advantage coming from assigning the computational work to more than one processing unit through the parallelisation of the linear algebra. This is where structure-exploiting parallel solvers such as OOPS [37] excel. Moreover, structure-exploiting interior point methods can be used not only for linear programming problems, but also for quadratic and nonlinear problems [36].

5.1.1 Stochastic programming concepts

A stochastic programming problem incorporates the uncertain parameters in the model. Following [10], this can be illustrated by the *two-stage stochastic problem*

$$\begin{aligned} \min_{x^1} \quad & (q^1)^T x^1 + I\!E_\xi Q(x^1, \xi) \\ \text{s.t.} \quad & W^1 x^1 = h^1, \\ & x^1 \geq 0, \end{aligned} \tag{5.1}$$

where ξ is a random variable, $\mathbb{E}_\xi$ is the expectation function and

$$\begin{aligned} Q(x^1, \xi) \; = \; \min_{y(\xi)} \quad & q(\xi)^T y(\xi) \\ \text{s.t.} \quad & W(\xi) y(\xi) = h(\xi) - T(\xi) x^1, \\ & y(\xi) \geq 0. \end{aligned} \tag{5.2}$$

We use the convention that first-stage variables and coefficients carry the superscript 1. Problem (5.1)-(5.2) can be interpreted as an optimization problem in which some parameters or coefficients are unknown. While (5.1) models the first stage decisions, (5.2) refers to the second stage decisions, which can be made only after a realisation of the random variable ξ has become known. Note how the first-stage decision variables x^1 appear in (5.2); at the time when the realisation become available, the first-stage decisions have already been made.

In stochastic programming, the uncertain environment is described through a stochastic process which is either assumed to be known, or can be estimated from historical data or conjectured according to some prescribed properties. The continuous process ξ is usually further approximated by a discrete distribution in order to obtain a computationally amenable description. In such a case, the most common techniques [45, 78] generate a finite, but usually very large, number of scenarios that represent an approximate description of the possible outcomes.

The model can be generalised to a *multi-stage model* in which the evolution of uncertainties can be described as an alternating sequence of decisions and random realisations that occur at different stages. A multi-stage stochastic program with recourse is a multi-period mathematical program where parameters are assumed to be uncertain along the time path. The stages do not necessarily refer to time periods, but they correspond to steps in the decision process. In particular, at each stage the realisations of some random parameters become known, and a decision must be taken. The main interest lies in the first-stage decisions which consist of all decisions that have to be made before the information is revealed. Later-stage decisions are allowed to adapt to the information that has become available up to that point.

The discrete stochastic process can be represented as an *event tree* $\mathcal{T}$, where each node denotes a stage when a realisation of the random process becomes known and a subsequent decision is taken. It is common to consider *balanced trees*, in which the number of branches leaving from each node is the same at all stages. An alternative approach, preferred in multi-stage models of very large dimensions, is to consider more branches in the earlier stages than in the later ones, giving rise to the so-called *left-heavy trees*. The framework we introduce in Section 5.2 can deal with any shape of trees. A very simple event tree, which we will use later on for illustration, is shown in Figure 5.1.

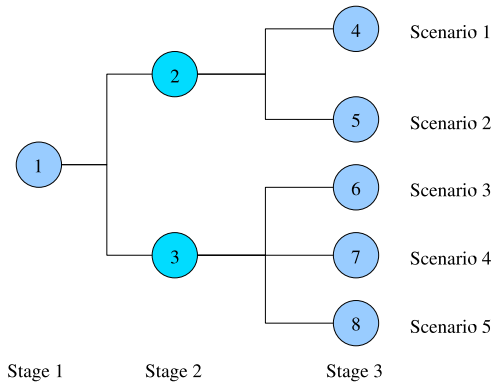

Figure 5.1: An event tree.

To each node of the event tree we associate a set of constraints, an objective function, and the conditional probability of visiting the node from its parent node in the previous stage. A path from the root to a leaf node of the event tree represents a scenario. The probability of each scenario is the product of the conditional probabilities of visiting each node on the path.

As we have already mentioned, a very large number of scenarios is usually needed to adequately capture the characteristics of the underlying continuous distribution, particularly in the multi-stage setting. In this case, the number of scenarios grows exponentially with the number of decisions considered at each stage. The question of how to generate an appropriate scenario tree is not trivial, and has received extensive attention in the literature [26, 45, 46, 78].

5.1.2 The deterministic equivalent formulation

A natural formulation of a stochastic programming problem relies on recursion to describe the dynamics of the modelled process. The term *recourse* means that, at each stage, the decision variables adapt to the different outcomes of the random parameters. For ease of presentation, we consider the linear version of the recourse problem:

$$\begin{aligned} \min_{x^1} \quad & (q^1)^T x^1 + I\!E_\xi\big[\min_{y(\xi)} q(\xi)^T y(\xi)\big] \\ \text{s.t.} \quad & W^1 x^1 = h^1, \\ & T(\xi)x^1 + W(\xi)y(\xi) = h(\xi), \\ & x^1 \geq 0,\ y(\xi) \geq 0, \end{aligned} \tag{5.3}$$

where $y(\xi)$ denotes the recourse action which depends on the outcome of the random process ξ. Note that (5.3) combines (5.1) and (5.2) in a single formulation. After discretising ξ according to $P(y^i = \xi_i) = p^i$, and using the notation $T^i = T(\xi_i)$ (analogously for h^i, W^i, y^i and q^i), $i = 2, \ldots, n$, problem (5.3) can be

written in the *deterministic equivalent formulation*:

$$\begin{array}{lllll} \min\limits_{(x^1,y)} & (q^1)^T x^1 + \sum\limits_{i=2}^{n} p^i (q^i)^T y^i & & & \\ \text{s.t.} & W^1 x^1 & & & = h^1, \\ & T^2 x^1 + W^2 y^1 & & & = h^2, \\ & \vdots & & \ddots & \vdots \\ & T^n x^1 & & + W^n y^n & = h^n, \\ & x^1 \geq 0,\ y^i \geq 0. & & & \end{array} \tag{5.4}$$

The deterministic equivalent formulation is derived by writing each possible realisation of the random parameters explicitly. The formulation (5.4) does not have any stochasticity left, but is completely deterministic, and is therefore a common linear programming problem of (very) large dimensions. Also note that problem (5.4) displays a dual-block angular structure in the constraint matrix.

The same approach can be immediately extended to multi-stage models. To formulate the deterministic equivalent of the multi-stage stochastic programming problem we first need to enumerate all nodes of the event tree. We use a breadth-first search order, i.e. we start from a root node corresponding to the initial stage and end with leaf nodes corresponding to the last stage.

Let $t = 1, 2, \ldots, T$ denote the stages and let $\mathcal{L}_t$ be the set of nodes at stage t. With $a(l)$ we denote the direct ancestor of node $l \in \mathcal{L}_t$ (which is a node that belongs to stage $t-1$), and with $\mathcal{D}_l \subset \mathcal{L}_{t+1}$ the set of children of node l. The decision variables are superscripted with the node number l; similar notation is used for the corresponding matrix and vector blocks. The total number of scenarios is N.

In the case of one-period recourse, the main constraint that describes the dynamics of the system has the form

$$T^l x^{a(l)} + W^l x^l = h^l, \qquad l \in \mathcal{L}_t,\ t = 2, \ldots, T,$$

where T^l is the technology matrix that varies with the node in the event tree, and W^l is the recourse matrix that, in general, depends on realisations within the same stage, but often varies only with time.

The deterministic equivalent formulation of the multi-stage problem has the following general form:

$$\begin{array}{lll} \min\limits_{x} & \sum\limits_{t=1}^{T} \sum\limits_{l \in \mathcal{L}_t} p^l (q^l)^T x^l & \\ \text{s.t.} & W^1 x^1 = h^1, & \\ & T^l x^{a(l)} + W^l x^l = h^l, & l \in \mathcal{L}_t,\ t = 2, \ldots, T, \\ & x^l \geq 0, & l \in \mathcal{L}_t,\ t = 1, \ldots, T. \end{array} \tag{5.5}$$

Note that the probabilities in the objective function of problem (5.5) are the unconditional path probabilities: p^l is the probability that a path goes through node l, which equals the product of the conditional probabilities along the path from the root to node l. Clearly, (5.5) represents a structured linear program. Its structure should be exploited in the solution algorithm.

If the event tree is traversed with depth-first-search ordering of the nodes during the generation of the mathematical program, the corresponding constraint matrix displays a nested dual block-angular structure. Figure 5.2 displays the two possible structures for the event tree of Figure 5.1 according to the chosen ordering of nodes. While the different ordering of blocks within the matrix is not relevant

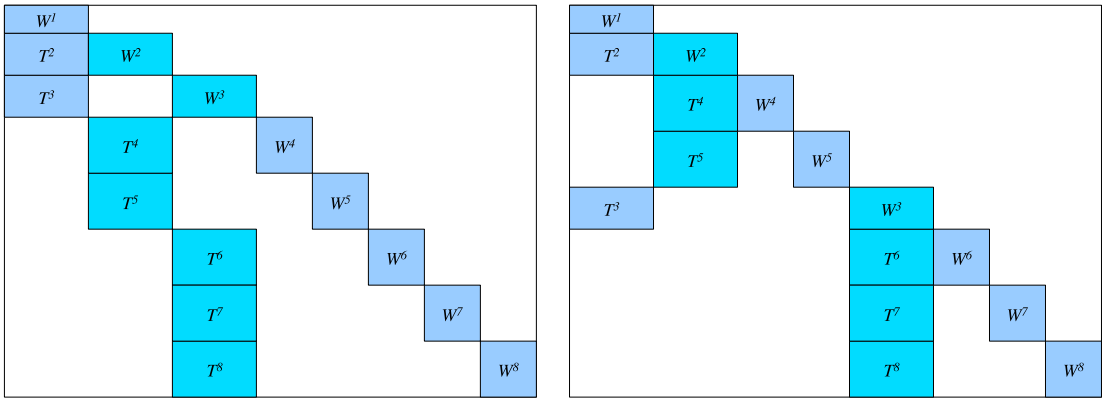

Figure 5.2: Deterministic equivalent corresponding to the event tree of Figure 5.1, with nodes listed in breadth-first order (left) and depth-first order (right).

for general-purpose solvers, the structure-exploiting software OOPS [36, 37] can take full advantage of the nested dual-block angular structure resulting from the depth-first ordering in its internal object-oriented linear algebra representation.

Several solution methods for stochastic linear programs have been presented in the literature. These often rely on some decomposition approach [8, 57, 73]. Kall and Mayer [50] provided a comparison of different solution algorithms for stochastic linear programming problems. In what follows, we consider solving the deterministic equivalent problem directly by using an interior point method.

5.2 A reduced-tree warm-start iterate

We already surveyed warm-start strategies in Section 3.4. While those strategies apply to general linear problems, we introduce an approach tailored to stochastic programming. In particular, we propose to exploit the structure inherent to a stochastic programming problem to generate a good warm-start iterate.

In the event tree corresponding to a large multistage program, the numerous leaf nodes descend from a relatively small number of branches in the first few

stages. Two "neighbouring" scenarios, that is two scenarios that have common nodes, may display large differences concerning later stage decisions, but the decisions taken in the earlier stages are identical (nonanticipativity). Moreover, there may be very little difference among scenarios, and so the large-scale problem can provide a fine-grained solution to a problem that could have been solved more coarsely by using a much smaller tree.

These observations suggest a warm-start technique that can be applied in the context of interior point methods. A warm-start solution is obtained by solving the stochastic optimization problem for a reduced event tree, whose dimension is much smaller than that of the complete one. The solution to the reduced problem is used to construct an advanced iterate for the complete formulation. We provide evidence that this novel way of exploiting the problem structure to generate an initial point provides a better iterate (in terms of centrality, feasibility, and closeness to optimality) than the one produced by a generic strategy.

Techniques for reducing the size of the scenario tree have been studied before from a probabilistic perspective; in some cases considerable savings can be obtained with such methods. Among others, Dupačová *et al.* [27] discuss an optimal scenario reduction technique that couples a large reduction of the scenario tree with a small loss in accuracy. In their example, a reduction by 50% of the scenario tree still maintains about 90% of the original accuracy. Here we are interested in capturing some aspects of the stochasticity of the event tree without assuming further knowledge of the underlying stochastic process that generated it. Given this difference from what is required for example by [27], we will use less sophisticated arguments in finding a reduced tree. We remark that if we have knowledge of the underlying stochastic process, then we could exploit it in the generation of the reduced tree.

We first study how to build a reduced tree by choosing just some of the possible scenarios. We provide some insight on how to make this selection, so that our choice performs better than an arbitrary one. Then we discuss how to obtain a warm-start solution from the reduced tree that corresponds to the chosen scenarios. Our aim is to generate a warm-start iterate that allows the complete problem to be solved to optimality in fewer iterations (and less computing time) than if starting from an iterate chosen with a standard initial point heuristic [65]. With these aims, we propose a way of choosing a subset of scenarios that we believe to be sufficiently representative of the whole tree. Our strategy can be summarised in the steps of Algorithm 5.1.

In the rest of this section we define our method of generating a reduced tree and describe the construction of the complete warm-start iterate.

Algorithm 5.1 Reduced-tree warm-start algorithm

Given: The complete event tree $\mathcal{T}$.

Generate a reduced event tree $\mathcal{T}_{\mathcal{R}} \subset \mathcal{T}$;

Solve the corresponding deterministic equivalent with a loose tolerance;

Use this solution to construct a warm-start iterate for the complete problem;

Solve the complete problem to optimality.

5.2.1 Scenario distance

For the purposes of generating a reduced scenario tree (see Section 5.2.2), we need to define a measure of distance between scenarios. Recalling that a scenario is a path in the event tree from the root to a leaf node, we can encode a scenario s_k, $k = 1, \ldots, N$, as an ordered set of nodes $s_k = \{l_1, \ldots, l_T : l_t = a(l_{t+1}), t = 1, \ldots, T-1\}$. To each node l_t of the tree we associate the 4-tuple $\eta^{l_t} = \{T^{l_t}, W^{l_t}, h^{l_t}, q^{l_t}\}$ of matrices, right-hand side and objective coefficients.

We first define the distance between two nodes i_t and j_t that belong to the same stage t as

$$d(\eta^{i_t}, \eta^{j_t}) = \|T^{i_t} - T^{j_t}\| + \|W^{i_t} - W^{j_t}\| + \|h^{i_t} - h^{j_t}\| + \|q^{i_t} - q^{j_t}\|, \qquad (5.6)$$

for some norm $\|\cdot\|$. Hence, we compute the distance between scenarios i and j as

$$D(s_i, s_j) = \sum_{t=1}^{T} d(\eta^{i_t}, \eta^{j_t}), \qquad i_t \in s_i,\ j_t \in s_j.$$

Scenarios that belong to the same branch of the tree will have smaller distance in general, as they share some of the nodes. Conversely, scenarios are likely to be farther away if they do not share nodes apart from the root.

5.2.2 Reduced tree generation

We generate the reduced tree by taking into account both the structural and the stochastic information available from the problem formulation. By structural information we mean the shape of the event tree, i.e. how the tree branches at the various stages. By stochastic information we mean the probabilities associated to each node in the tree, and consequently to each scenario. Hence we adopt two complementary strategies. First we choose a subset of branches of the event tree; then, in each branch, we choose the most representative leaf nodes.

We try to capture the structure of the complete tree by making sure that a sufficient number of different early stage decisions will appear in the reduced tree. In some sense, we look for a way to span the breadth of the complete tree. For a

defined small value $k < T$, where T is the number of stages in the problem, we choose some of the nodes at the k-th stage, together with all their ancestors up to the root, to appear in the subtree. The choice of nodes to appear in the reduced tree should be guided by probabilities. The rationale for this strategy is to ensure that our warm-start iterate is a good representation of the decisions to be taken in the first few stages, as getting early decisions right is fundamental for easier optimization of the later stages. To illustrate this idea, suppose we deal with a multistage setting where there are $T = 4$ stages, such as in the tree of Figure 5.3: for $k = 2$ we choose nodes 1, 2 and 3 to be in the reduced tree.

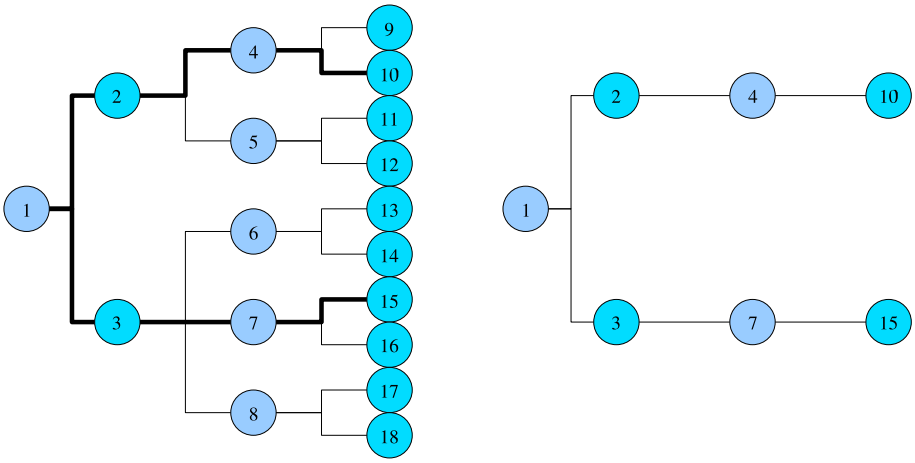

Figure 5.3: Complete tree and the reduced tree corresponding to the chosen scenarios (in bold).

Each of the chosen nodes in the k-th stage now becomes the root of a branch of the tree, which we call a subtree. In each subtree we choose the scenario that minimises the distance to an average scenario in the same subtree. Let S_t be the set of nodes in the subtree S at stage t, and $|S_t|$ its cardinality. For each stage t within subtree S, we determine an artificial node n_t by averaging the data associated to all nodes at this stage:

$$n_t = \frac{1}{|S_t|} \sum_{l_t \in S_t} (T^{l_t}, W^{l_t}, h^{l_t}, q^{l_t}), \qquad k < t \leq T.$$

We define the average scenario for subtree S as the ordered set of nodes $s_k = \{l_k, n_{k+1}, \ldots, n_T\}$. Therefore, the average scenario $\bar{s}$ (in the complete tree) is obtained by listing the nodes from the root of the tree to the root of the subtree S, and then by appending the average nodes. We define it as

$$\bar{s} = \{\, l_1, \ldots, l_k, n_{k+1}, \ldots, n_T \,\},$$

where $l_t = a(l_{t+1})$ for $t = 1, \ldots, k-1$. Scenario $\bar{s}$ is completely artificial, and there is no guarantee that it is feasible. Therefore, we cannot use it directly as our

representative scenario. Instead, we use it as a reference point which we compare all other scenarios to, and thus find the closest scenario among the existing ones. In this way we do not introduce spurious infeasibilities. Hence, in the subtree S we choose the representative scenario s^* as

$$s^* = s_k, \quad k = \arg\min_{i \in S}\{(1 - p^i) D(s_i, \bar{s})\}, \tag{5.7}$$

where, since our ultimate goal is to find the most representative scenario in the subtree, we use the term $(1 - p^i)$ to "bring closer" scenarios that have a higher probability of occurring.

Continuing the example started above, we consider two subsets of scenarios, corresponding to nodes 9–12 (for the subtree rooted at node 2) and to nodes 13–18 (for the subtree rooted at node 3). Within each subset we build the scenario of average nodes and then find the representative scenario. The resulting reduced tree is shown in the right of Figure 5.3.

The reduced tree selection induces a function $r : \mathcal{T} \to \mathcal{T}_\mathcal{R}$ that maps each node of the complete tree to a corresponding node in the reduced tree in the following way: if $i \in \mathcal{T}_\mathcal{R}$, then $r(i) = i$; if $i \notin \mathcal{T}_\mathcal{R}$, then we choose as $r(i)$ the node in the representative scenario corresponding to node i belonging to the same stage t as node i. In other words, to get from node $i \in \mathcal{T}$ to $r(i) \in \mathcal{T}_\mathcal{R}$ we walk up the tree $\mathcal{T}$ until we find a node that is also in $\mathcal{T}_\mathcal{R}$, and from there walk back down the reduced tree until we arrive at the same stage as the original node. This is shown in Figure 5.4.

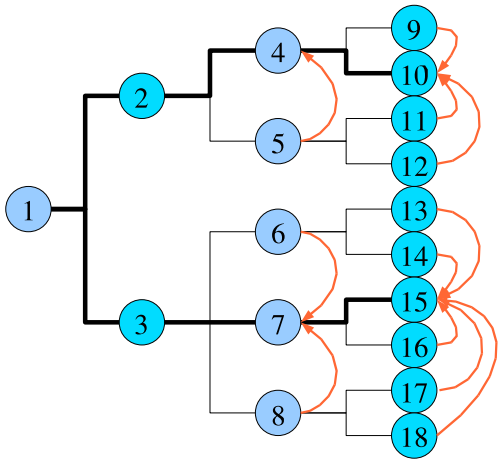

Figure 5.4: Mapping of the nodes in the complete tree to the corresponding reduced tree nodes (nodes on the bold arcs map to themselves).

This mapping is used to decide how to initialise the warm-start iterate for the complete tree, as presented in the next section. We remark that our generation process guarantees that for each $i \in \mathcal{T}$, the following properties hold:

$$a(r(i)) \in \mathcal{T}_\mathcal{R}, \quad \text{and} \quad a(r(i)) = r(a(i)), \tag{5.8}$$

that is, if a node is in the reduced tree then so is its parent, and the mapping $r(\cdot)$ preserves the parent-child relationship.

We introduce the set $\mathcal{I}_l$ of nodes in the complete tree that are initialised by a given node $l \in \mathcal{T}_\mathcal{R}$, that is $\mathcal{I}_l = \{\, i \in \mathcal{T} : r(i) = l \,\}$, $l \in \mathcal{T}_\mathcal{R}$. It should be noted that the reduced-tree generation process can be interpreted as a node-aggregation process, in which all nodes in $\mathcal{I}_l \subset \mathcal{T}$ are aggregated into a single node $l \in \mathcal{T}_\mathcal{R}$. The node aggregation determines the node probabilities p^l_R associated with each node $l \in \mathcal{T}_\mathcal{R}$; we use:

$$p^l_R = \sum_{i \in \mathcal{I}_l} p^i. \tag{5.9}$$

Here and in the following we use the convention that symbols referring to the reduced tree (such as $\mathcal{T}_\mathcal{R}$, p^i_R) carry the subscript $_R$.

It is worth noting the effect of the probability update (5.9) on the conditional probabilities. Denote by $\delta^i = p^i / p^{a(i)}$ the conditional probability of reaching node i, given that its parent $a(i)$ has already been reached. The conditional probabilities in the reduced tree are updated according to

$$\delta^l_R = \sum_{i \in \mathcal{I}_l \cap D_{a(l)}} \delta^i, \quad l \in \mathcal{T}_\mathcal{R}. \tag{5.10}$$

To see that (5.10) holds, we note that for a node l in the k-th stage where the subtree selection takes place we have $\mathcal{I}_l \cap \mathcal{D}_{a(l)} = \mathcal{I}_l$, and (5.10) follows from (5.9) by dividing through by $p^{a(l)}$. For a node l after stage k we have $\delta^l_R = 1$, we need to sum only over the descendants of $a(l)$. With our reduced-tree selection strategy, for such nodes we actually have $\delta^l_R = 1$, since there is only one representative scenario in each subtree.

5.2.3 Construction of the warm-start iterate

Consider the linear programming problem in standard form

$$\min_x \; c^T x \quad \text{s.t.} \;\; Ax = b, \;\; x \geq 0, \tag{5.11}$$

where $A \in \mathbb{R}^{m \times n}$ is full rank, $x, c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. Problem (5.11) corresponds to the deterministic equivalent (5.5) generated from a given event tree $\mathcal{T}$. We will refer to it as the *complete problem*.

From the reduced tree $\mathcal{T}_\mathcal{R}$ we build the reduced deterministic equivalent problem

$$\min_{x_R} \; c^T_R x_R \quad \text{s.t.} \;\; A_R x_R = b_R, \;\; x_R \geq 0, \tag{5.12}$$

with $A_R \in \mathbb{R}^{m_R \times n_R}$, $x_R \in \mathbb{R}^{n_R}$ and $b_R \in \mathbb{R}^{m_R}$. We call problem (5.12) the *reduced problem*. As we expect that $(n_R, m_R) \ll (n, m)$, the reduced problem is much smaller than the complete formulation, and hence much easier to solve.

We solve problem (5.12) with an interior point method. For the reasons presented in Section 3.4, we do not aim for optimality, but instead we aim for a sufficiently advanced primal–dual feasible point. Therefore we stop at an iterate $(x_R^*, y_R^*, s_R^*) \in \mathcal{N}_s(\gamma)$ for a barrier parameter corresponding to merely few digits of accuracy in the solution. This iterate is used to construct the warm-start point $(\hat{x}, \hat{y}, \hat{s})$ for the complete problem.

In our approach, the size of the problem changes in both the number of constraints and the number of variables. Therefore, we are faced with the challenge of using the reduced-tree solution to provide a warm-start iterate for the complete system. This objective can be achieved on a node by node basis by exploiting the inherent structure of the problem. The sections of the vectors $(\hat{x}, \hat{y}, \hat{s})$ corresponding to node $i \in \mathcal{T}$ are initialised from the sections of the vectors (x_R^*, y_R^*, s_R^*) corresponding to node $r(i) \in \mathcal{T}_\mathcal{R}$. This process will be detailed below.

Denote by $(\hat{x}^i, \hat{y}^i, \hat{s}^i)$ the part of the vectors $(\hat{x}, \hat{y}, \hat{s})$ corresponding to node $i \in \mathcal{T}$ and likewise (x_R^i, y_R^i, s_R^i) for components of the solution of the reduced problem. Then we construct the starting point for the complete problem in the following manner:

$$\hat{x}^i = x_R^{r(i)}, \qquad (\hat{y}^i, \hat{s}^i) = \frac{p^i}{p_R^{r(i)}}(y_R^{r(i)}, s_R^{r(i)}), \qquad i \in \mathcal{T}, \tag{5.13}$$

where $p_R^{r(i)}$ is computed according to (5.9). This means that the dual reduced-tree solution is spread among the nodes it initialises, as can be seen here:

$$\sum_{i \in \mathcal{I}_k} (\hat{y}^i, \hat{s}^i) = \sum_{i \in \mathcal{I}_k} \frac{p^i}{p_R^k}(y_R^k, s_R^k) = (y_R^k, s_R^k)\frac{1}{p_R^k}\sum_{i \in \mathcal{I}_k} p^i = (y_R^k, s_R^k), \qquad k \in \mathcal{T}_\mathcal{R}.$$

Considering again the example of Figure 5.3, suppose that in the reduced tree we accepted only the scenarios that end at node 10 and node 15, so that the reduced tree consists of nodes 1, 2, 3, 4, 7, 10 and 15. By solving the corresponding reduced problem, we obtain the parts of the solution vector associated to such nodes. These can be used directly in the complete iterate (Figure 5.5 top). We fill in the missing elements by reproducing the solution from the nodes in the same subtree and the same stage (Figure 5.5 bottom). The proposed way of constructing the complete iterate is easy to implement and its execution time is negligible.

5.3 Analysis of the warm-start iterate

In this section we study how the warm-start iterate generated with the procedures presented above satisfies the conditions expressed by Gondzio and Grothey [34]. Contrary to what is assumed in both [94] and [34], in our approach the dimension

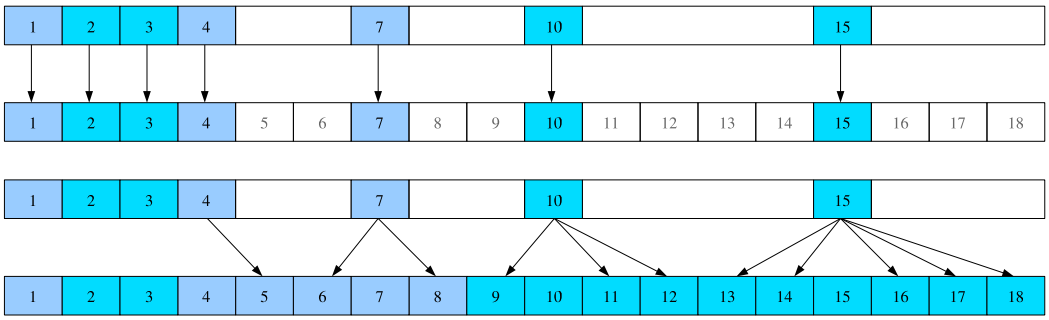

Figure 5.5: Generation of the warm-start iterate.

of the problem changes, as the reduced tree problem is, by construction, much smaller than the complete problem.

However, similarly to what we did with the solution vector, we can expand the reduced problem to one which has the same dimension as the complete problem (5.11). This happens by replicating the blocks in the coefficient matrix and in the objective and right-hand side vectors:

$$\hat{W}^i = W^{r(i)}, \quad \hat{T}^i = T^{r(i)}, \quad \hat{q}^i = q^{r(i)}, \quad \hat{h}^i = h^{r(i)}, \qquad i \in \mathcal{T},$$

as illustrated in Figure 5.6.

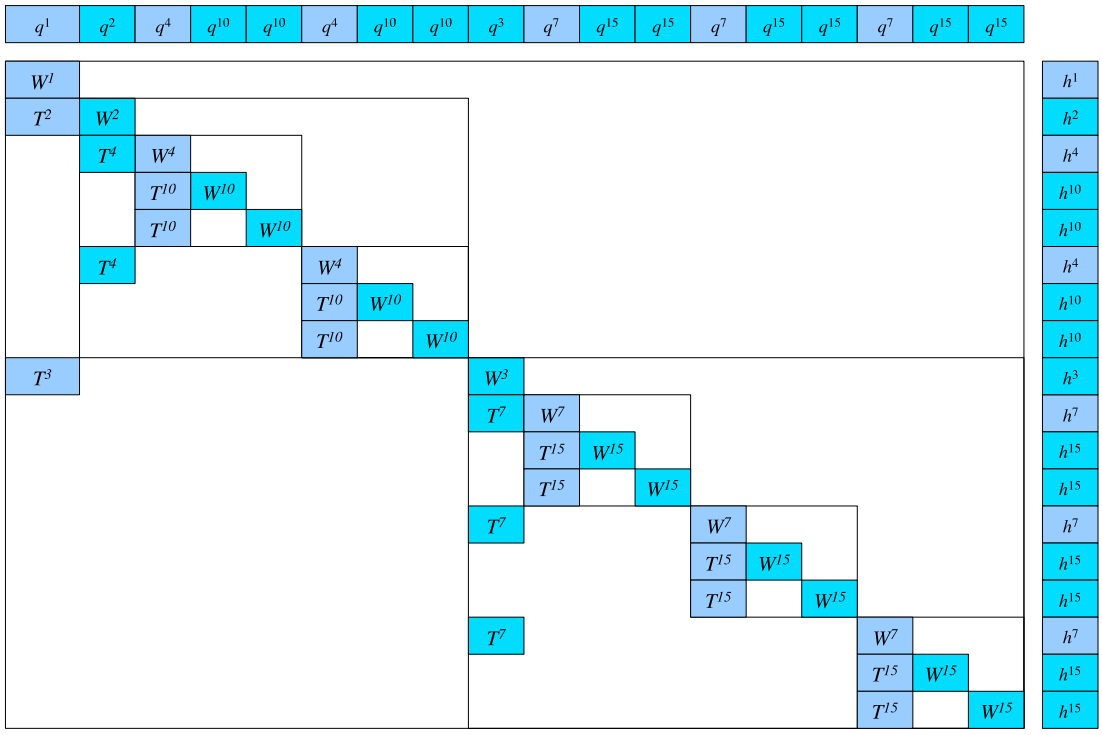

Figure 5.6: The expanded system for the event tree of Figure 5.3.

This corresponds to creating the (artificial) *expanded problem*

$$\min_x \ \hat{c}^T x \quad \text{s.t.} \ \ \hat{A}x = \hat{b}, \ x \geq 0, \tag{5.14}$$

the dimension of which, $\hat{A} \in \mathbb{R}^{m \times n}$, $\hat{c}, x \in \mathbb{R}^n$ and $\hat{b} \in \mathbb{R}^m$, corresponds to the dimension of the complete problem (5.11). Using the notation introduced earlier, we will denote all symbols referring to the expanded problem with a hat $\hat{}$.

To analyse the warm-start iterate we can now follow a two-step procedure. First we note that from an advanced iterate $(x_R, y_R, s_R) \in \mathcal{N}_s^R(\gamma)$ for the reduced problem the procedure in (5.13) constructs a primal–dual feasible point $(\hat{x}, \hat{y}, \hat{s})$ for the expanded problem. Indeed, in Theorem 5.3 we will show that $(\hat{x}, \hat{y}, \hat{s}) \in \widehat{\mathcal{N}}_s(\hat{\gamma})$. In the second step we can use this iterate to warm-start the complete problem. Since in the step from the expanded to the complete problem the problem data changes but not its size, the methods developed in [34, 94] can be used to analyse the warm-start iterate.

We start the analysis with a technical result.

Lemma 5.1. *Let $l \in \mathcal{T}$, then*

$$\sum_{i \in \mathcal{D}_l} T^{r(i)^T} \hat{y}^i = \frac{p^l}{p_R^{r(l)}} \sum_{k \in \mathcal{D}_{r(l)}^R} T^{k^T} y_R^k.$$

Proof. We have this chain of identities:

$$\begin{aligned}
\sum_{i \in \mathcal{D}_l} T^{r(i)^T} \hat{y}^i &= \sum_{i \in \mathcal{D}_l} T^{r(i)^T} y_R^{r(i)} \frac{p^i}{p_R^{r(i)}} \\
&= \frac{p^l}{p_R^{r(l)}} \sum_{i \in \mathcal{D}_l} T^{r(i)^T} y_R^{r(i)} \frac{\delta^i}{\delta_R^{r(i)}} \\
&= \frac{p^l}{p_R^{r(l)}} \sum_{k \in \mathcal{D}_{r(l)}^R} \frac{T^{k^T} y_R^k}{\delta_R^k} \sum_{i \in \mathcal{I}_k \cap \mathcal{D}_l} \delta^i,
\end{aligned}$$

where the second equality follows from $p^i = p^l \delta^i$ and $p_R^{r(i)} = p_R^{r(l)} \delta_R^{r(i)}$ for $i \in \mathcal{D}_l$. The last equality follows from the fact that we can partition $\mathcal{D}_l$ according to which nodes of the reduced tree are used for initialisation: $\mathcal{D}_l = \bigcup_{k \in \mathcal{D}_{r(l)}^R} \mathcal{I}_k \cap \mathcal{D}_l$. The claim then follows from (5.10). □

Theorem 5.2. *If (x_R, y_R, s_R) is primal and dual feasible for the reduced problem (5.12), then the warm-start solution $(\hat{x}, \hat{y}, \hat{s})$ obtained from (5.13) is primal and dual feasible for the expanded problem (5.14).*

Proof. As $\hat{x}^l = x_R^{r(l)}$, primal feasibility is trivially satisfied:

$$T^{r(l)} \hat{x}^{a(l)} + W^{r(l)} \hat{x}^l = h^{r(l)}, \qquad l \in \mathcal{T}. \tag{5.15}$$

Now we consider dual feasibility. By assumption, the reduced problem solution satisfies the dual constraints:

$$W^{r(l)^T} y_R^{r(l)} + \sum_{i \in \mathcal{D}_{r(l)}^R} T^{r(i)^T} y_R^{r(i)} + s_R^{r(l)} = p_R^{r(l)} q^{r(l)}, \qquad r(l) \in \mathcal{T}_{\mathcal{R}}.$$

Multiplying both terms by $p^l/p_R^{r(l)}$ we obtain

$$\frac{p^l}{p_R^{r(l)}}\Big(W^{r(l)^T} y_R^{r(l)} + \sum_{i \in \mathcal{D}_{r(l)}^R} T^{r(i)^T} y_R^{r(i)} + s_R^{r(l)}\Big) = p^l q^{r(l)},$$

which, according to (5.13) and Lemma 5.1, becomes

$$W^{r(l)^T} \hat{y}^l + \sum_{i \in \mathcal{D}_l} T^{r(i)^T} \hat{y}^i + \hat{s}^l = p^l q^{r(l)}, \qquad l \in \mathcal{T}, \tag{5.16}$$

so $(\hat{y}, \hat{s})$ satisfies the dual constraints in the expanded problem. □

Theorem 5.3. *If $(x_R, y_R, s_R) \in \mathcal{N}_s^R(\gamma)$ for some $\gamma \in (0,1)$, then $(\hat{x}, \hat{y}, \hat{s}) \in \widehat{\mathcal{N}}_s(\rho\gamma)$, where*

$$\rho = \min_{i \in \mathcal{T}} \left\{ \frac{p^i}{p_R^{r(i)}} \frac{n}{n_R}, \frac{p_R^{r(i)}}{p^i} \frac{n_R}{n} \right\}. \tag{5.17}$$

Proof. From Theorem 5.2, the warm-start iterate $(\hat{x}, \hat{y}, \hat{s})$ is feasible in the reduced system. Hence, here we only need to prove centrality. We observe that

$$\begin{aligned}
\hat{\mu} &= \frac{\hat{x}^T \hat{s}}{n} = \frac{1}{n} \sum_{i \in \mathcal{T}} (\hat{x}^i)^T \hat{s}^i = \frac{1}{n} \sum_{k \in \mathcal{T}_\mathcal{R}} \sum_{i \in I_k} (\hat{x}^i)^T \hat{s}^i \\
&= \frac{1}{n} \sum_{k \in \mathcal{T}_\mathcal{R}} \sum_{i \in I_k} \frac{p^i}{p_R^k} (x_R^k)^T s_R^k \\
&= \frac{1}{n} \sum_{k \in \mathcal{T}_\mathcal{R}} \frac{1}{p_R^k} (x_R^k)^T s_R^k \sum_{i \in I_k} p^i \\
&= \frac{n_R}{n} \mu_R,
\end{aligned}$$

where we used (5.13) and (5.9), and $\mu_R = x_R^T s_R / n_R$. Hence, since $(x_R, y_R, s_R) \in N_s(\gamma)$ implies $(x_R)_j (s_R)_j \geq \gamma \mu_R$, for $j = 1, \ldots, n_R$, using (5.17) we have

$$\hat{x}_j^l \hat{s}_j^l = (x_R^{r(l)})_j (s_R^{r(l)})_j \frac{p^l}{p_R^{r(l)}} \geq \gamma \mu_R \frac{p^l}{p_R^{r(l)}} = \gamma \hat{\mu} \frac{n}{n_R} \frac{p^l}{p_R^{r(l)}} \geq \rho\gamma\hat{\mu}, \qquad l \in \mathcal{T}.$$

The upper bound $\hat{x}_j^l \hat{s}_j^l \leq \hat{\mu}/(\rho\gamma)$ can be derived similarly. □

5.3.1 Absorbing perturbations

We argue that the difference between the data of the expanded problem (5.14) and that of the original (complete) problem (5.11) can be interpreted as a perturbation between two problem instances of identical dimension. Clearly the expanded system has merely a theoretical interest, as we use it to evaluate the magnitude of the perturbation introduced, and we never generate it in practice.

We assume that a feasible long-step path-following algorithm based on the symmetric neighbourhood $\mathcal{N}_s(\gamma)$ is used to solve the warm-started complete problem. Although the constructed warm-start iterate $(\hat{x}, \hat{y}, \hat{s})$ from (5.13) is feasible in the expanded problem, it is not feasible in the complete problem. As in [94] and [34] we derive conditions that guarantee to absorb these infeasibilities with one *modification step*. For this, consider the following Newton system:

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ \hat{S} & 0 & \hat{X} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} \xi_b \\ \xi_c \\ 0 \end{bmatrix}, \tag{5.18}$$

where $\xi_b = b - A\hat{x}$ and $\xi_c = c - A^T\hat{y} + \hat{s}$ are the infeasibilities incurred by using the expanded iterate $(\hat{x}, \hat{y}, \hat{s})$ to warm-start the complete problem. Such a modification is termed a *Newton step correction* in [94] or an *Additional centering iteration* in [35]. Gondzio and Grothey [34] analyse the same system, but are concerned with absorbing primal and dual infeasibility separately by splitting (5.18) into two separate directions. We will give a more general result and apply it to the situation of warm-start for stochastic programming problems. To avoid overburdening notation we will drop the hat from the warm-start vectors. We will keep it in the neighbourhoods to make a clear distinction between $\widehat{\mathcal{N}}_s(\gamma)$ and $\mathcal{N}_s(\gamma)$ denoting the symmetric neighbourhoods for the expanded problem (5.14) and the complete problem (5.11), respectively.

After some straightforward manipulations following the arguments of [34], the Newton direction (5.18) can be expressed in terms of the primal and dual residuals ξ_b, ξ_c as

$$\begin{aligned} \Delta x &= (XS^{-1}A^T(AXS^{-1}A^T)^{-1}AXS^{-1} - XS^{-1})\xi_c + XS^{-1}A^T(AXS^{-1}A^T)^{-1}\xi_b, \\ \Delta y &= (AXS^{-1}A^T)^{-1}(AXS^{-1}\xi_c + \xi_b), \\ \Delta s &= (I - A^T(AXS^{-1}A^T)^{-1}AXS^{-1})\xi_c - A^T(AXS^{-1}A^T)^{-1}\xi_b. \end{aligned} \tag{5.19}$$

We consider the matrix

$$Q = I - S^{-1}A^T(AXS^{-1}A^T)^{-1}AX,$$

and restate Lemma 3.2 of [34], which provides a bound on the norm of Q, in terms of the symmetric neighbourhood $N_s(\gamma)$.

Lemma 5.4. *If $w \in \mathcal{N}_s(\gamma)$, then $\|Q\|_2 \leq 1/\gamma$.*

Proof. For a point $w \in \mathcal{N}_s(\gamma)$, the following inequalities hold:

$$(x_i s_i)^{-1/2} \leq (\gamma\mu)^{-1/2}, \quad \text{and} \quad (x_i s_i)^{1/2} \leq (\mu/\gamma)^{1/2}.$$

With some manipulations, we can express matrix Q as

$$Q = X^{-1/2}S^{-1/2}\left[I - X^{1/2}S^{-1/2}A^T(AXS^{-1}A^T)^{-1}AX^{1/2}S^{-1/2}\right]X^{1/2}S^{1/2},$$

where the term in square brackets is an orthogonal projection on the null space of $AX^{1/2}S^{-1/2}$, so its Euclidean norm is 1. As desired, we obtain

$$\|Q\|_2 = \|X^{-1/2}S^{-1/2}\|_2\|X^{1/2}S^{1/2}\|_2 \leq 1/\gamma. \qquad \square$$

In the next Lemma we state sufficient conditions for the perturbations to guarantee a full Newton step.

Lemma 5.5. *Let $w \in \widehat{\mathcal{N}}_s(\gamma)$ be the warm-start iterate and define the scaled residuals*

$$\tilde{\xi}_b = X^{-1}A^T(AA^T)^{-1}\xi_b \quad \text{and} \quad \tilde{\xi}_c = S^{-1}\xi_c. \tag{5.20}$$

If for $\beta < 1$ we have

$$\|\tilde{\xi}_b\|_\infty + \|\tilde{\xi}_c\|_\infty \leq \beta\left(1 + \sqrt{n}/\gamma\right)^{-1},$$

then the full Newton step (5.18) from the warm-start iterate can be taken and absorbs the complete infeasibilities.

Proof. Using the definitions of the matrix Q and of the relative residual vectors (5.20), the relations (5.19) simplify to

$$X^{-1}\Delta x = -Q\tilde{\xi}_c + (I - Q)\tilde{\xi}_b = -S^{-1}\Delta s,$$

yielding the bound

$$\begin{aligned}\|X^{-1}\Delta x\|_\infty &\leq \|Q\|_\infty\|\tilde{\xi}_c\|_\infty + (1 + \|Q\|_\infty)\|\tilde{\xi}_b\|_\infty \\ &\leq (1 + \|Q\|_\infty)(\|\tilde{\xi}_b\|_\infty + \|\tilde{\xi}_c\|_\infty).\end{aligned} \tag{5.21}$$

As $(x, y, s) \in \widehat{\mathcal{N}}_s(\gamma)$, using Lemma 5.4 we get that $\|Q\|_\infty \leq \sqrt{n}\|Q\|_2 \leq \sqrt{n}/\gamma$. Substituting it into (5.21), we obtain

$$\|X^{-1}\Delta x\|_\infty \leq \left(1 + \sqrt{n}/\gamma\right)(\|\tilde{\xi}_b\|_\infty + \|\tilde{\xi}_c\|_\infty),$$

which, under the condition of the Lemma, implies

$$\|X^{-1}\Delta x\|_\infty = \|S^{-1}\Delta s\|_\infty \leq \beta, \tag{5.22}$$

that is the full Newton step is feasible, as $\beta < 1$. $\square$

Theorem 5.6. *Let $w \in \widehat{\mathcal{N}}_s(\gamma)$ and $\beta < 1$. Under the conditions of Lemma 5.5, the new point $\tilde{w} = (x + \Delta x, y + \Delta y, s + \Delta s) \in \mathcal{N}_s(\frac{1-\beta^2}{1+\beta^2}\gamma)$.*

Proof. At the new point $\tilde{w}$ the barrier parameter is

$$n\tilde{\mu} = \sum_{i=1}^{n} \tilde{x}_i \tilde{s}_i = \sum_{i=1}^{n} (x_i + \Delta x_i)(s_i + \Delta s_i) = \sum_{i=1}^{n} (x_i s_i + \Delta x_i \Delta s_i), \tag{5.23}$$

as the last equation of (5.18) implies $s_i \Delta x_i + x_i \Delta s_i = 0$, $i = 1, \ldots, n$. Using (5.22) from Lemma 5.5, we have that $\|X^{-1}\Delta x\|_\infty \|S^{-1}\Delta s\|_\infty \leq \beta^2$, and so

$$-\beta^2 x_i s_i \leq \Delta x_i \Delta s_i \leq \beta^2 x_i s_i; \tag{5.24}$$

by summing up all products we obtain

$$-\beta^2 n\mu \leq \sum_{i=1}^{n} \Delta x_i \Delta s_i \leq \beta^2 n\mu,$$

which, by adding $n\mu = \sum_i x_i s_i$ to all terms and using (5.23), leads to

$$(1 - \beta^2) n\mu \leq n\tilde{\mu} \leq (1 + \beta^2) n\mu. \tag{5.25}$$

We now study whether the new iterate is still in (some) symmetric neighbourhood of the central path by checking the pairwise complementary products

$$\tilde{x}_i \tilde{s}_i = x_i s_i + \Delta x_i \Delta s_i = \Big(1 + \frac{\Delta x_i \Delta s_i}{x_i s_i}\Big) x_i s_i.$$

Using (5.24) and (5.25) we obtain

$$\tilde{x}_i \tilde{s}_i \geq (1 - \beta^2)\gamma\mu \geq \frac{1 - \beta^2}{1 + \beta^2} \gamma\tilde{\mu},$$
$$\tilde{x}_i \tilde{s}_i \leq (1 + \beta^2)\frac{\mu}{\gamma} \leq \frac{1 + \beta^2}{1 - \beta^2} \frac{1}{\gamma}\tilde{\mu},$$

which proves the statement of the theorem. □

5.3.2 Conditions on the warm-start iterate

We use Lemma 5.5 to obtain conditions that the reduced tree has to satisfy in order for a warm-start of the complete problem to be successful. In order to prove this result, we need to assume that the primal–dual solution (x_R^*, y_R^*, s_R^*) to the reduced stochastic programming problem is uniformly bounded, say,

$$\max\{\|x_R^*\|_\infty, \|y_R^*\|_\infty, \|s_R^*\|_\infty\} \leq B, \quad \max\{\|(X_R^*)^{-1}e\|_\infty, \|(S_R^*)^{-1}e\|_\infty\} \leq B, \tag{5.26}$$

where $B > 1$. It is worth noting that since we work with the symmetric neighbourhood (2.15), we actually need only the first inequality to hold. Indeed, if $x_j^* \leq B$ then $1/s_j^* \leq x_j^*/(\gamma\mu) \leq B/(\gamma\mu)$ and, similarly, if $s_j^* \leq B$ then $1/x_j^* \leq$

$s_j^*/(\gamma\mu) \leq B/(\gamma\mu)$. In other words, the boundedness of the iterate (x_R^*, y_R^*, s_R^*) implies the boundedness of the component-wise inverses of x_R^* and s_R^*.

The reduced problem solution is in a neighbourhood of the central path for the reduced problem. In particular, this is the case if additional centering steps are computed once the desired tolerance level has been attained [33]. Using the feasibility result of Theorem 5.2, the residuals for the complete problem at the warm-start point $(\hat{x}, \hat{y}, \hat{s})$ are:

$$\begin{aligned} \xi_b &= b - A\hat{x} &&= (b - \hat{b}) - (A - \hat{A})\hat{x}, \\ \xi_c &= c - A^T\hat{y} - \hat{s} &&= (c - \hat{c}) - (A - \hat{A})^T\hat{y}. \end{aligned}$$

It is crucial to ensure that the primal and dual residuals ξ_b and ξ_c are small. By construction, the elements of the vectors $(b - \hat{b})$ and $(c - \hat{c})$ that correspond to nodes in the reduced tree are zero; for the same reason, the corresponding blocks of $(A - \hat{A})$ are zero as well. The elements corresponding to the nodes not considered in the reduced tree will be, in general, non zero. However, as the scenarios in the reduced tree were chosen according to (5.7) in order to minimize the distance from the average case, we expect the perturbations to be small.

We can now state the following result, in which we obtain some bounds on the size of the primal and dual perturbations.

Lemma 5.7. *Let the reduced tree be chosen in such a way that for every node $i \in \mathcal{T}$ the node distance (5.6) is $d(r(i), i) < \varepsilon$, for an $\varepsilon > 0$. If the reduced problem solution is primal and dual feasible and satisfies (5.26), then $\|\xi_b\|_\infty \leq \varepsilon B$ and $\|\xi_c\|_\infty \leq \varepsilon B|\mathcal{T}_\mathcal{R}|$, where $|\mathcal{T}_\mathcal{R}|$ is the number of nodes in the reduced tree.*

Proof. Using the form of the stochastic programming problem (5.5) we can write the primal residual of the complete problem as

$$\|\xi_b\|_\infty = \|b - A\hat{x}\|_\infty = \max\{\|h^l - T^l\hat{x}^{a(l)} - W^l\hat{x}^l\|_\infty : l = 1, \ldots, L_T\}.$$

The contribution of a node $l \in \mathcal{T}$ to ξ_b is

$$\begin{aligned} \|\xi_b^l\|_\infty &= \|h^l - T^l\hat{x}^{a(l)} - W^l\hat{x}^l\| \\ &= \|h^l - h^{r(l)} - (T^l - T^{r(l)})\hat{x}^{a(l)} - (W^l - W^{r(l)})\hat{x}^l\| \\ &\leq \big(\|h^l - h^{r(l)}\| + \|T^l - T^{r(l)}\| + \|W^l - W^{r(l)}\|\big)B \\ &\leq d(l, r(l))B \leq \varepsilon B, \end{aligned}$$

where the step from the first to the second line uses (5.15), and all norms here are infinity norms. This clearly implies that $\|\xi_b\|_\infty \leq \varepsilon B$.

The dual residual for the complete problem at the warm-start point can be written as

$$\|\xi_c\|_\infty = \|c - A^T\hat{y} - \hat{s}\|_\infty = \max\{\|p^l q^l - {W^l}^T\hat{y}^l - \sum_{i \in \mathcal{D}_l} {T^i}^T\hat{y}^i - \hat{s}^l\|_\infty : l = 1, \ldots, L_T\}.$$

The contribution of a node $l \in \mathcal{T}$ to ξ_c is

$$\begin{aligned}\xi_c^l &= p^l q^l - {W^l}^T \hat{y}^l - \sum_{i \in \mathcal{D}_l} {T^i}^T \hat{y}^i - \hat{s}^l \\ &= p^l(q^l - q^{r(l)}) - (W^l - W^{r(l)})^T \hat{y}^l - \sum_{i \in \mathcal{D}_l} (T^i - T^{r(i)})^T \hat{y}^i \\ &= p^l(q^l - q^{r(l)}) - \frac{p^l}{p_R^{r(l)}} \Big[(W^l - W^{r(l)})^T y_R^{r(l)} + \sum_{i \in \mathcal{D}_l} (T^i - T^{r(i)})^T \frac{\delta^i}{\delta_R^{r(i)}} y_R^{r(i)} \Big],\end{aligned}$$

where the step from the first to the second line uses (5.16) and the next step uses (5.13) together with $p^i = p^l \delta^i$ and $p_R^{r(i)} = p_R^{r(l)} \delta_R^{r(i)}$. Taking norms (all norms here are infinity norms) and using the partitioning $\mathcal{D}_l = \bigcup_{k \in D_{r(l)}^R} \mathcal{I}_k \cap \mathcal{D}_l$ we obtain

$$\begin{aligned}\|\xi_c^l\|_\infty &\le \|q^l - q^{r(l)}\| + \|W^l - W^{r(l)}\| \|y_R^{r(l)}\| + \sum_{k \in \mathcal{D}_{r(l)}^R} \|y_R^k\| \sum_{i \in \mathcal{I}_k \cap \mathcal{D}_l \setminus \{k\}} \|T^i - T^k\| \frac{\delta^i}{\delta_R^k} \\ &\le \|q^l - q^{r(l)}\| + \|W^l - W^{r(l)}\| \|y_R^{r(l)}\| + \sum_{k \in \mathcal{D}_{r(l)}^R} \|y_R^k\| \varepsilon \sum_{i \in \mathcal{I}_k \cap \mathcal{D}_l \setminus \{k\}} \frac{\delta^i}{\delta_R^k} \\ &\le \Big(\|q^l - q^{r(l)}\| + \|W^l - W^{r(l)}\| + \sum_{k \in \mathcal{D}_{r(l)}^R} \big(1 - \frac{\delta^k}{\delta_R^k}\big)\varepsilon \Big) B \le \varepsilon B |\mathcal{T}_\mathcal{R}|. \qquad \square\end{aligned}$$

The following result combines the findings of Lemmas 5.5 and 5.7.

Theorem 5.8. *Let the assumptions of Lemma 5.7 be satisfied and*

$$\varepsilon B^2 \max\{\|A\|_\infty \|(AA^T)^{-1}\|_\infty,\ |\mathcal{T}_\mathcal{R}|\} \le \frac{1}{2}\beta \left(1 + \sqrt{n}/\gamma\right)^{-1}.$$

Then the full Newton step (5.18) from the warm-start iterate is feasible and restores primal and dual feasibility.

Proof. Using the definition of $\tilde{\xi}_b$ from (5.20), the bounds (5.26), and Lemma 5.7, we get

$$\|\tilde{\xi}_b\|_\infty = \|X^{-1} A^T (AA^T)^{-1} \xi_b\|_\infty \le \varepsilon B^2 \|A\|_\infty \|(AA^T)^{-1}\|_\infty \le \frac{1}{2}\beta \left(1 + \sqrt{n}/\gamma\right)^{-1},$$

In a similar way, we obtain

$$\|\tilde{\xi}_c\|_\infty = \|S^{-1} \xi_c\|_\infty \le \varepsilon B^2 |\mathcal{T}_\mathcal{R}| \le \frac{1}{2}\beta \left(1 + \sqrt{n}/\gamma\right)^{-1}.$$

Now the result follows from Lemma 5.5. $\square$

It is worth making a few remarks about these results. Theorem 5.8 implies that if we can choose the reduced scenario tree such that $\varepsilon = \max_i\{d(r(i), i)\}$ is small enough to satisfy the bound given in the Theorem, then the warm-start point

constructed from the reduced scenario tree will be successful for the complete problem. Unfortunately we have only limited influence on ε. Indeed ε is the result of the variation of the problem data between the expanded and the complete systems.

There are two approaches that can be adopted if the warm-start for a given reduced tree fails. The first relies on generating a denser reduced tree. This would directly reduce ε, but it would also imply solving a bigger reduced deterministic equivalent, making the solution of the reduced problem more expensive. Since the description of the new reduced tree is more detailed, we guarantee that the infeasibility of the new warm-start point is smaller. The second way, fully explored in the literature (see Section 3.4), suggests reducing the accuracy with which the reduced problem is solved. This is justified by the fact that the amount of perturbation that can be absorbed is directly related to μ for the warm-start point. In our approach we see that, considering (5.26), a change in the accuracy of the reduced-tree solution (x^*, y^*, s^*) affects the value of B that appears in the left-hand side of the bound of Theorem 5.8.

It is important to note, however, that the bounds we derived in Lemma 5.7 and Theorem 5.8 express theoretical requirements. There is a gap between theory and practice. In practice much larger infeasibilities $\|\tilde{\xi}_b\|, \|\tilde{\xi}_c\|$ can be absorbed. This is confirmed by our numerical results where even choosing just 2 scenarios in the reduced tree leads to a significant reduction in the number of interior point iterations required to solve the complete problem.

5.4 Implementation and numerical results

We first implemented the strategy of generating a reduced tree and the corresponding warm-start iterate within the HOPDM [32] solver. We tested a series of publicly available stochastic problems in the SMPS format [9] coming from the POSTS collection available from:

`http://users.iems.northwestern.edu/~jrbirge/html/dholmes/post.html`.

It should be noted that we disabled HOPDM's presolve in order to preserve the dimensions of the problems, and thus obtain sensible warm-start points.

We solved the reduced problem with an optimality tolerance of 5.0×10^{-1}, while the optimality tolerance for the complete problem was set to 5.0×10^{-8}. Computations were performed on a Linux PC with a 3.0GHz Intel Pentium processor and 1GB of RAM. In Table 5.1 we report the dimensions of the problems in terms of the number of stages and scenarios for the complete tree, the number

of iterations and the computing time (in seconds) with cold-start and warm-start. The latter includes the generation and solution of the reduced problem, and the construction of the warm-start iterate.

While the analysis of Section 5.3 is very conservative in its estimates of the absorbable perturbations, in practice we noticed that the reduced-tree warm-start strategy is effective even with a much sparser tree than suggested by the theory. In the warm-start case, the reduced tree was built with only 2 scenarios.

The problems solved show an overall good behaviour of our warm-start strategy, with time savings of up to 59% (for problem `pltexpA5_6`). The generation of the reduced tree and the solution of the corresponding problem (5.12) is generally fast, and, as the problem sizes increase become negligible. However, for the smallest instances of our test set (`fxm2_16`, `fxm3_6` and `fxm4_6`), it is noticeable and consumes the savings produced by using an advanced iterate.

Problem data			Cold start		Warm start	
Name	Stages	Scens	Iters	Time	Iters	Time
fxm2_16	2	16	22	1.2	13	1.0
fxm3_6	3	36	30	1.5	17	1.3
fxm3_16	3	256	40	31.1	20	20.7
fxm4_6	4	216	30	8.2	22	8.3
fxm4_16	4	4,096	41	218.3	27	182.6
pltexpA3_16	3	256	26	153.8	14	87.8
pltexpA4_6	4	216	36	55.8	16	27.5
pltexpA5_6	5	1,296	81	772.0	30	311.5
storm27	2	27	41	95.4	22	53.2
storm125	2	125	73	107.3	36	69.1
storm1000	2	1,000	107	1,498.3	45	831.5

Table 5.1: Results obtained with HOPDM, 2 scenarios in the reduced tree.

5.4.1 Telecommunication problems

Given the favourable results, we implemented the same approach in OOPS [37, 36], where we were able to test larger problem instances. Since OOPS does not have features such as presolve and scaling, the accuracy requested in the solution has to be smaller. We set it to 5.0×10^{-4} which is sufficient for telecommunication applications. On the other hand, OOPS makes an effective use of its structure-exploiting capabilities [37, 36], allowing the solver to tackle large-scale problems and provides access to parallel computing techniques.

We applied our warm-start strategy to the capacity assignment problem with uncertain demand, a model relevant to the telecommunication industry [77]. The objective of this model is to find the optimal choice of capacities to be assigned

to the links in the network in order to minimize unsatisfied customer demands. In our particular application we assume that the topology of the network and the sets of origin–destination pairs are given and are not going to change during the planning horizon.

We model this situation as a two-stage stochastic linear program with recourse. The general model has the following form:

$$\min_{x} \; E_d[f(x,d)] \qquad \text{s.t.} \; \sum_{l\in\mathcal{A}} c_l x_l \leq M, \;\; x \geq 0,$$

where c_l and x_l are the cost and capacity of link $l \in \mathcal{A}$, respectively, and M is a bound on the budget. The objective here is to minimize the expected cost (conditional on the uncertain demand). This general model describes the first stage decision about the link capacities. The function $f(x,d)$ is defined in the following model, which describes the second stage decisions:

$$\begin{aligned} f(x,d) = \min_{z_p} \quad & \sum_{k\in\mathcal{D}} (d_k - \sum_{p\in P_k} z_p) & \\ \text{s.t.} \quad & \sum_{k\in\mathcal{D}} \sum_{p\in\mathcal{P}_k : l\in p} z_p \leq x_l & \forall l \in \mathcal{A} \\ & \sum_{p\in\mathcal{P}_k} z_p \leq d_k & \forall k \in \mathcal{D} \\ & z_p \geq 0, & \end{aligned}$$

where d_k is the demand for the k-th origin–destination pair, $\mathcal{P}_k$ is a given set of paths linking the k-th pair, and z_p is the flow on path p.

To generate the uncertain demands of each scenario we used the approach described in [77]. For each origin–destination pair k we need to have a demand estimate d_k, which can be determined from historic data or from an educated guess. The demand is assumed to be uniformly distributed around this estimate. Hence, the demand d_k^s for the k-th pair in scenario s is given by

$$d_k^s = (1 + \epsilon_k^s) d_k,$$

where ϵ_k^s is a random number generated in the interval $[-\rho, \rho]$. The value of $\rho > 0$ determines the range in which we assume the demand to fluctuate. In our experiments we chose a value of $\rho = 0.5$, thus allowing very large variations in the demand.

The relevant network characteristics of the problems solved are shown in Table 5.2, where we detail the size of the network, the number of demands considered, the overall number of paths and the average number of arcs in each path.

For a problem with N scenarios, the number of constraints and decision variables (including slacks) are

$$m = 1 + N \times (\#\mathcal{A} + \#\mathcal{D}), \qquad \text{and} \qquad n = 1 + \#\mathcal{A} + N \times (\#\mathcal{A} + \#\mathcal{D} + \#\mathcal{P}),$$

Name	Nodes	Arcs	Demands	Paths	Av.Length
mnx	12	50	66	189	2.6
jlg	26	84	264	697	5.6
mgntA	53	158	1,378	3,574	6.7
mgntB	70	210	2,346	6,137	6.4

Table 5.2: Characteristics of the telecommunication networks.

respectively, where $\#\mathcal{A}$ is the number of arcs, $\#\mathcal{D}$ the number of demands, and $\#\mathcal{P}$ the total number of paths. The dimensions of the problems we generated and solved are collected in Table 5.3.

Name	Rows	Cols	Nonzeros	Name	Rows	Cols	Nonzeros
mnx-100	11	30	85	jlg-100	34	104	503
mnx-200	23	61	170	jlg-200	69	209	1,006
mnx-400	46	122	340	jlg-400	139	418	2,013
mnx-800	92	244	680	jlg-800	278	836	4,025
mnx-1600	185	488	1,362	jlg-1600	556	1,672	8,051
mgntA-100	153	511	2,919	mgntB-100	255	869	4,816
mgntA-200	307	1,022	5,838	mgntB-200	511	1,738	9,633
mgntA-400	614	2,044	11,675	mgntB-400	1,022	3,477	19,265

Table 5.3: Approximate size of the problems in thousands.

In the second and third column of Table 5.4 we report the solution statistics for OOPS. Computations were performed on a Linux PC with 3.0GHz Intel Pentium processor and 2GB of RAM. In all cases the reduced tree was built with merely two scenarios. Therefore the computation time corresponding to the solution of the reduced problem (included in time reported in the table) was always negligible. The savings of warm-start over cold-start strategy vary between 40% and 80% in most cases.

We have also solved the smallest instances of these problems with Cplex 9.0 Barrier Solver. The problems `mnx-100`, `jlg-100`, `mgntA-100` and `mgntB-100` were solved in 1.1s, 7.1s, 4379.9s and 9030.4s, respectively. This means that Cplex was about 4 and 2 times faster than OOPS on `mnx-100` and `jlg-100` problems, respectively but it was about 28 times slower than OOPS on more difficult `mgntA-100` and `mgntB-100` problems.

In the fourth and fifth columns of Table 5.4 we report the parallel performance of OOPS on a cluster of four machines with a 3.0GHz Intel Pentium processor and 2GB of RAM each. In this case, we choose the size of the reduced tree to be equal to the number of processors employed for two different reasons. First, it is preferable to assign to OOPS a balanced number of blocks on each processor, so we needed to guarantee that each processor gets at least one block; second, we obtain a more refined starting solution at no additional computational cost.

Problem	Cold start		Warm start		Cold start		Warm start	
	Iters	Time	Iters	Time	Iters	Time	Iters	Time
mnx-100	15	6.7	9	3.9	15	3.9	9	2.6
mnx-200	13	12.9	7	7.3	13	4.6	7	3.5
mnx-400	16	28.9	8	15.5	16	10.5	8	6.3
mnx-800	17	58.8	10	39.5	17	18.8	10	10.7
mnx-1600	19	131.1	10	68.8	19	50.3	10	31.4
jlg-100	21	38.3	6	15.5	21	11.0	6	6.1
jlg-200	45	164.9	17	39.5	45	49.9	17	20.7
jlg-400	44	255.2	18	103.1	43	83.2	19	39.7
jlg-800	27	353.4	10	152.9	29	130.5	10	50.1
jlg-1600	32	855.3	13	360.6	35	286.1	14	129.7
mgntA-100	28	260.0	14	156.2	28	76.9	14	51.6
mgntA-200	50	877.1	35	690.6	50	256.4	34	195.3
mgntA-400	40	1,470.3	14	572.5	40	410.9	14	181.6
mgntB-100	23	511.1	14	318.0	23	137.5	14	103.9
mgntB-200	25	909.4	8	332.4	25	284.2	8	140.5
mgntB-400	29	2,154.5	7	538.1	29	605.5	7	211.6

Table 5.4: Efficiency of the warm-start strategy in OOPS in the serial case (2 scenarios in the reduced tree) and in the parallel case (4 processors and 4 scenarios in the reduced tree).

However the analysis of the parallel results collected in Table 5.4 indicates that the use of a slightly larger reduced tree does not translate into any noticeable improvement in the warm-start runs as measured with the number of warm-start iterations. Obviously, the solution times are reduced but this is the effect of using more processors.

Chapter 6

Conclusions and future research.

In this chapter we summarise the results obtained and present some observations derived from the experience gathered during this research. Our focus here is to draw together the original aspects of this work and present possible directions for future research.

6.1 Research outcomes

We started by considering what characteristics an interior point iterate should have. The main feature is undoubtedly centrality, as following the central path leads to the optimal solution. The centrality requirement is satisfied by defining a neighbourhood of the central path inside which all iterates should lie. This led us to consider and analyse the symmetric neighbourhood of the central path, which expresses both a lower and an upper bound on the complementarity products. A feasible algorithm based on the symmetric neighbourhood $\mathcal{N}_s(\gamma)$ matches the theoretical complexity of one based on the wide $\mathcal{N}_{-\infty}(\gamma)$ neighbourhood, as it converges in $\mathcal{O}(n)$ iterations (Theorem 2.7). Our analysis shows that the presence of an upper bound does not adversely affect the theoretical properties. As this neighbourhood is at the heart of the successful multiple centrality correctors technique [32], we believe that it properly describes the properties of a well-centered iterate for a practical algorithm.

In Chapter 4 we revisited the technique of multiple centrality correctors and added a new degree of freedom to it. Instead of computing the corrected direction as $\Delta w = \Delta_p w + \Delta_c w$, where $\Delta_p w$ and $\Delta_c w$ are generic predictor and corrector terms, we allow the use of a weight $\omega \in (0, 1]$ as a scaling factor for the corrector, and thus compute the search direction as $\Delta^\omega w = \Delta_p w + \omega \Delta_c w$. This led to the weighted correctors scheme of Algorithm 4.1. We combined this modification with the use of a symmetric neighbourhood of the central path as a tool to find appropriate target points. The analysis of Jarre and Wechs [48] discussed in

Section 4.1.1 makes it very clear that the choice of the target t in the right-hand side of the Newton system is the driving tool in finding effective search directions. In our new implementation of multiple centrality correctors, we have pushed the target vector of complementary points further in the infeasible space with the aim to generate a better correction to the current iterate.

We compared our algorithm against the recently introduced Krylov subspace scheme of Mehrotra and Li [66]. The two approaches have similarities, as they look for a set of attractive independent terms from which the final direction is constructed. Mehrotra and Li's approach uses the first few elements from the basis of the Krylov space and solves an auxiliary linear program to find an optimal combination of all available direction terms. The weighted correctors technique generates direction terms using centrality correctors of [32], evaluates the weight for each corrector independently, and it can detect when the use of an additional corrector term would not be beneficial. The extensive computational results presented for different classes of problems demonstrate the potential of the weighted correctors technique, particularly for large-scale problems. The comparison presented in Section 4.4.1 shows some advantage of our scheme over that of [66]. Indeed, with the same number of direction terms allowed, our scheme outperforms Krylov subspace searches by a wide margin. Given that the weighted correctors scheme follows the usual predictor–corrector framework, we expect it to be easier to implement in other interior point software. In particular, such a technique should be used for large-scale problems for which the reduction in number of iterations repays the increased cost of each iteration.

In Chapter 5 we introduced a technique that exploits the near-optimal solution to a stochastic linear program corresponding to a reduced scenario tree to warm-start a much larger problem that encompasses the complete scenario tree. Our way of reducing the dimension of the scenario tree is based on the assumption that we have no knowledge of the underlying stochastic process. Therefore we developed an ad-hoc measure of distance between the scenarios in the data space, and we proposed to choose those that minimize the distance to a selection of representative scenarios. A warm-start solution is obtained by solving the stochastic optimization problem for the reduced event tree, the dimension of which is much smaller than that of the complete one. The solution to the reduced problem is then used to construct an advanced iterate for the complete formulation.

We presented a thorough theoretical analysis of the warm-start iterate generated by our approach. In particular, we derived conditions which should be satisfied by the reduced tree to guarantee a successful warm-start of the complete problem. Most of our analysis concentrated on the primal and dual infeasibilities

at the warm-start point, as these quantities play a major role in the success of a warm-start iterate. We obtained bounds on the scaled residuals at the warm-start point for which a full Newton step aimed at absorbing these residuals is feasible (Lemma 5.5), and proved that the recovery step keeps the point in a symmetric neighbourhood of the central path (Theorem 5.6). In Lemma 5.7 and Theorem 5.8 we specialised the above results to our strategy of generating the warm-start point from a problem of reduced dimension.

We provided computational evidence that this novel way of exploiting the problem structure to generate an initial point provides a better starting iterate than the one produced by a generic starting point strategy. We observed that the iterate generated from the reduced problem provides an advanced starting point for the solution of the complete problem, in general resulting in a decrease in the number of iterations needed. As the computational cost of generating such an iterate is negligible, our warm-start strategy produces consistent savings in computational time.

6.2 Avenues of research

One of the advantages of generating correctors in a recursive way consists in the possibility of stopping the correction phase if the use of the corrector does not offer sufficient improvement. However, we realise that using a small linear programming subproblem to produce the optimal weighting of search directions can only improve our results in terms of stepsizes and number of iterations. It was suggested by Nick Gould and Ken McKinnon to implement a mixed strategy: by the multiple centrality correctors heuristic we can dynamically find the number of correctors needed at each iteration; then we find the weights by solving a subproblem. The subproblem could be very similar to the ones used by Jarre and Wechs [48] or Mehrotra and Li [66].

As we have seen in Chapter 4, many attempts have been made to find new and original search directions. We believe that the direction generated from the Newton system, possibly complemented by Mehrotra's second-order correction, are only one of the possible ways of exploring the solution space. From the study of subspace searches explored in Section 4.1, it is clear that the more directions we consider, the better the final search direction we get. Therefore, if we had a cheap way of generating search directions (rather than from solving a system of linear equations), then these should be employed. In this respect, Mehrotra and Li [66] mention employing previous search directions. The use of these incurs an increased memory requirement in order to store them, but no additional compu-

tational cost. However, it does not seem that they were actually employed in their implementation. This opens some questions on what constitutes a valid previous direction (only affine scaling, the final composite direction or something else).

Considering now the warm-start strategy for stochastic programming, we believe that such approach allows to confirm once again that also interior point methods can be efficiently warm-started. Moreover, it open some attractive avenues of research for warm-start strategies in interior point methods.

If the iterate produced by a reduced tree is not good enough in the sense that the warm-start strategy fails, then another one can be produced by generating a modified reduced tree (more bushy, for example). Since this second tree provides a better approximation to the complete tree, we can expect the infeasibilities of the corresponding warm-start point to be smaller. Hence, the chances for a successful warm-start increase. This leads to the idea of allowing a multi-start procedure, in which a series of reduced trees of increasing size are generated, and the solution to one of them is used to construct an advanced starting point for the next instance.

In our analysis we assumed not to have any information about the underlying stochastic process that governs the uncertainty. However, it is clear that such information would provide additional insight on which the choice of the reduced tree can be based. In such a situation, two trees would be generated, the complete one, and an optimally reduced one. As we have seen, in constructing the warm-start point we need to know the mapping of nodes between the reduced and complete trees. Therefore, the tree-reduction process should provide this information as well.

The relative youth of interior point methods means that there is still a lot to learn and try, particularly in practical implementations. An interesting avenue of research, according to the author, is the development of specialised techniques to exploit the problem structure. This means that the development and diffusion of structure-exploiting codes and of structure-aware modelling languages may become a necessary requirement for a new generation of interior point codes. In this sense, also theoretical developments in these aspects are wanted and necessary.

Bibliography

[1] Ilan Adler, Mauricio G. C. Resende, Geraldo Veiga, and Narendra Karmarkar. An implementation of Karmarkar's algorithm for linear programming. *Mathematical Programming*, 44:297–335, 1989.

[2] Erling D. Andersen and Knud D. Andersen. Presolving in linear programming. *Mathematical Programming*, 71(2):221–245, 1995.

[3] Erling D. Andersen, Jacek Gondzio, Csaba Mészáros, and Xiaojie Xu. Implementation of interior-point methods for large scale linear programs. In T. Terlaky, editor, *Interior Point Methods of Mathematical Programming*, pages 189–252. Kluwer Academic Publishers, 1996.

[4] David S. Atkinson and Pravin M. Vaidya. A scaling technique for finding the weighted analytic center of a polytope. *Mathematical Programming*, 57:163–192, 1992.

[5] Dave A. Bayer and Jeffrey C. Lagarias. The nonlinear geometry of linear programming I. Affine and projective scaling trajectories. *Transactions of the American Mathematical Society*, 314(2):499–526, 1989.

[6] Hande Y. Benson and David F. Shanno. An exact primal-dual penalty method approach to warmstarting interior-point methods for linear programming. Technical report, 2006. Accepted for publication in *Computational Optimization and Applications*.

[7] Luca Bergamaschi, Jacek Gondzio, and Giovanni Zilli. Preconditioning indefinite systems in interior point methods for optimization. *Computational Optimization and Applications*, 28:149–171, 2004.

[8] John R. Birge. Decomposition and partitioning methods for multistage stochastic linear programs. *Operations Research*, 33(5):989–1007, 1985.

[9] John R. Birge, Michael A. H. Dempster, Horand I. Gassmann, Eldon A. Gunn, Alan J. King, and Stein W. Wallace. A standard input format for

multiperiod stochastic linear programs. *Committee on Algorithms Newsletter*, 17:1–19, 1987.

[10] John R. Birge and François Louveaux. *Introduction to stochastic programming*. Springer Series in Operations Research, New York, 1997.

[11] Robert E. Bixby. Progress in linear programming. *ORSA Journal on Computing*, 6(1):15–22, 1994.

[12] Robert E. Bixby. Solving real-world linear programs: a decade and more of progress. *Operations Research*, 50(1):3–15, 2002.

[13] Tamra J. Carpenter, Irvin J. Lustig, John M. Mulvey, and David F. Shanno. Higher-order predictor-corrector interior point methods with application to quadratic objectives. *SIAM Journal on Optimization*, 3(4):696–725, 1993.

[14] Coralia Cartis. Some disadvantages of a Mehrotra-type primal-dual corrector interior-point algorithm for linear programming. Technical Report NA-04/27, Numerical Analysis Group, Computing Laboratory, Oxford University, December 2004.

[15] Coralia Cartis. On the convergence of a primal-dual second-order corrector interior point algorithm for linear programming. Technical Report NA-05/04, Numerical Analysis Group, Computing Laboratory, Oxford University, March 2005.

[16] Coralia Cartis and Nicholas I. M. Gould. Finding a point in the relative interior of a polyhedron. Technical Report NA-07/01, Numerical Analysis Group, Computing Laboratory, Oxford University, February 2007.

[17] Vašek Chvátal. *Linear programming*. W. H. Freeman and Company, New York, 1983.

[18] Marco Colombo and Jacek Gondzio. Further development of multiple centrality correctors. Technical Report MS-05-001, School of Mathematics, The University of Edinburgh, October 2005. Accepted for publication in *Computational Optimization and Applications*.

[19] Marco Colombo, Jacek Gondzio, and Andreas Grothey. A warm-start approach for large-scale stochastic linear programs. Technical Report MS-06-004, School of Mathematics, The University of Edinburgh, August 2006.

[20] Joe Czyzyk, Sanjay Mehrotra, and Steve Wright. PCx user guide. Technical Report OTC 96/01, Optimization Technology Center, May 1996.

[21] George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey, 1963.

[22] George B. Dantzig. Linear programming. *Operations Research*, 50(1):42–47, 2002.

[23] Antoine Deza, Eissa Nematollahi, Reza Peyghami, and Tamás Terlaky. The central path visits all the vertices of the Klee-Minty cube. *Optimization Methods and Software*, 21(5):851–865, 2006.

[24] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

[25] Iain S. Duff, Albert M. Erisman, and John K. Reid. *Direct methods for sparse matrices*. Clarendon Press, Oxford, 1986.

[26] Jitka Dupačová, Giorgio Consigli, and Stein W. Wallace. Scenarios for multistage stochastic programs. *Annals of Operations Research*, 100:25–53, 2000.

[27] Jitka Dupačová, Nicole Gröwe-Kuska, and Werner Römisch. Scenario reduction in stochastic programming. *Mathematical Programming*, 95:493–511, 2003.

[28] Shu-Cherng Fang and Sarat Puthenpura. *Linear optimization and extensions: theory and algorithms*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[29] Roger Fletcher. *Practical methods of optimization*. John Wiley and Sons, Chichester, 2nd edition, 1987.

[30] Donald Goldfarb and Michael J. Todd. Modifications and implementation of the ellipsoid algorithm for linear programming. *Mathematical Programming*, 23:1–19, 1982.

[31] Jacek Gondzio. HOPDM (version 2.12) – A fast LP solver based on a primal-dual interior point method. *European Journal of Operational Research*, 85:221–225, 1995.

[32] Jacek Gondzio. Multiple centrality corrections in a primal-dual method for linear programming. *Computational Optimization and Applications*, 6:137–156, 1996.

[33] Jacek Gondzio. Warm start of the primal-dual method applied in the cutting-plane scheme. *Mathematical Programming*, 83:125–143, 1998.

[34] Jacek Gondzio and Andreas Grothey. Reoptimization with the primal-dual interior point method. *SIAM Journal on Optimization*, 13(3):842–864, 2003.

[35] Jacek Gondzio and Andreas Grothey. A new unblocking technique to warm-start interior point methods based on sensitivity analysis. Technical Report MS-06-005, School of Mathematics, The University of Edinburgh, December 2006.

[36] Jacek Gondzio and Andreas Grothey. Solving non-linear portfolio optimization problems with the primal-dual interior point method. *European Journal of Operational Research*, 181(3):1012–1029, 2007.

[37] Jacek Gondzio and Robert Sarkissian. Parallel interior-point solver for structured linear programs. *Mathematical Programming*, 96:561–584, 2003.

[38] Jacek Gondzio and Tamás Terlaky. A computational view of interior point methods for linear programming. In J. E. Beasley, editor, *Advances in Linear and Integer Programming*, chapter 3, pages 103–144. Oxford University Press, Oxford, England, 1996.

[39] Jacek Gondzio and Jean-Philippe Vial. Warm start and ε-subgradients in a cutting plane scheme for block-angular linear programs. *Computational Optimization and Applications*, 14:17–36, 1999.

[40] Clovis C. Gonzaga. Large step path-following methods for linear programming, part I: barrier function method. *SIAM Journal on Optimization*, 1(2):268–279, 1991.

[41] Clovis C. Gonzaga. Path-following methods for linear programming. *SIAM Review*, 34(2):167–224, 1992.

[42] Osman Güler, Cornelis Roos, Tamás Terlaky, and Jean-Philippe Vial. A survey of the implications of the behavior of the central path for the duality theory of linear programming. *Management Science*, 41(12):1922–1934, 1995.

[43] Julian A. J. Hall and Ken I. M. McKinnon. Hyper-sparsity in the revised simplex method and how to exploit it. *Computational Optimization and Applications*, 32(3):259–283, 2005.

[44] Alexander L. Hipolito. A weighted least squares study of robustness in interior point linear programming. *Computational Optimization and Applications*, 2:29–46, 1993.

[45] Kjetil Høyland, Michal Kaut, and Stein W. Wallace. A heuristic for moment-matching scenario generation. *Computational Optimization and Applications*, 24(2-3):169–185, 2003.

[46] Kjetil Høyland and Stein W. Wallace. Generating scenario trees for multistage decision problems. *Management Science*, 47(2):295–307, February 2001.

[47] Benjamin Jansen. *Interior point techniques in optimization. Complementarity, sensitivity and algorithms.* PhD thesis, Delft University of Technology, 1995.

[48] Florian Jarre and Martin Wechs. Extending Mehrotra's corrector for linear programs. *Advanced Modeling and Optimization*, 1:38–60, 1999.

[49] Elizabeth John and E. Alper Yıldırım. Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimensions. Technical report, Department of Industrial Engineering, Bilkent University, Turkey, May 2006. Accepted for publication in *Computational Optimization and Applications*.

[50] Peter Kall and János Mayer. Some insights into the solution algorithms for SLP problems. *Annals of Operations Research*, 142:147–164, 2006.

[51] Peter Kall and Stein W. Wallace. *Stochastic programming.* John Wiley and Sons, New York, 1994.

[52] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

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

[54] Masakazu Kojima, Nimrod Megiddo, and Shinji Mizuno. A primal–dual infeasible-interior-point algorithm for linear programming. *Mathematical Programming*, 61:263–280, 1993.

[55] Masakazu Kojima, Shinji Mizuno, and Akiko Yoshise. A polynomial-time algorithm for a class of linear complementarity problems. *Mathematical Programming*, 44:1–26, 1989.

[56] Masakazu Kojima, Shinji Mizuno, and Akiko Yoshise. A primal-dual interior point algorithm for linear programming. In N. Megiddo, editor, *Progress in Mathematical Programming: Interior-Point and Related Methods*, chapter 2, pages 29–47. Springer-Verlag, New York, 1989.

[57] Jeff Linderoth and Stephen J. Wright. Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24(2/3):207–250, 2003.

[58] Irvin J. Lustig. Feasibility issues in a primal–dual interior-point method for linear programming. *Mathematical Programming*, 49:145–162, 1991.

[59] Irvin J. Lustig, Roy E. Marsten, and David F. Shanno. On implementing Mehrotra's predictor–corrector interior-point method for linear programming. *SIAM Journal on Optimization*, 2(3):435–449, 1992.

[60] Irvin J. Lustig, Roy E. Marsten, and David F. Shanno. Computational experience with a globally convergent primal–dual predictor–corrector algorithm for linear programming. *Mathematical Programming*, 66:123–135, 1994.

[61] Irvin J. Lustig, Roy E. Marsten, and David F. Shanno. Interior point methods for linear programming: computational state of the art. *ORSA Journal on Computing*, 6(1):1–14, 1994.

[62] István Maros and Csaba Mészáros. The role of the augmented system in interior point methods. *European Journal of Operational Research*, 107:720–736, 1998.

[63] Nimrod Megiddo. Pathways to the optimal set in linear programming. In N. Megiddo, editor, *Progress in Mathematical Programming: Interior-Point and Related Methods*, chapter 8, pages 131–158. Springer-Verlag, New York, 1989.

[64] Nimrod Megiddo. On finding primal- and dual-optimal bases. *ORSA Journal on Computing*, 3(1):63–65, 1991.

[65] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[66] Sanjay Mehrotra and Zhifeng Li. Convergence conditions and Krylov subspace-based corrections for primal-dual interior-point method. *SIAM Journal on Optimization*, 15(3):635–653, 2005.

[67] John E. Mitchell. *Karmakar's algorithm and combinatorial optimization problems*. PhD thesis, Cornell University, 1988.

[68] John E. Mitchell. Interior point algorithms for integer programming. In J. E. Beasley, editor, *Advances in Linear and Integer Programming*, chapter 6, pages 223–248. Oxford University Press, Oxford, England, 1996.

[69] John E. Mitchell, Panos M. Pardalos, and Mauricio G. C. Resende. Interior point for combinatorial optimization. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of combinatorial optimization*, volume 1, pages 189–298. Kluwer Academic Publishers, 1998.

[70] John E. Mitchell and Michael J. Todd. Solving combinatorial optimization problems using Karmarkar's algorithm. *Mathematical Programming*, 56:245–284, 1992.

[71] Shinji Mizuno, Michael J. Todd, and Yinyu Ye. On adaptive step primal-dual interior-point algorithms for linear programming. *Mathematics of Operations Research*, 18:964–981, 1993.

[72] Renato D. C. Monteiro and Ilan Adler. Interior path following primal–dual algorithms. Part I: linear programming. *Mathematical Programming*, 44:27–41, 1989.

[73] John M. Mulvey and Andrzej Ruszczyński. A new scenario decomposition method for large-scale stochastic optimization. *Operations Research*, 43(3):477–490, 1995.

[74] George L. Nemhauser and Laurence A. Wolsey. *Integer and combinatorial optimization*. John Wiley and Sons, New York, 1988.

[75] Aurelio R. L. Oliveira and Danny C. Sorensen. A new class of preconditioners for large-scale linear systems from interior point methods for linear programming. *Linear Algebra and its Applications*, 394:1–24, 2005.

[76] James M. Ortega and Werner C. Rheinboldt. *Iterative solution of nonlinear equations in several variables*. Academic Press, New York, 1970.

[77] Adam Ouorou. Robust capacity assignment in telecommunications. *Computational Management Science*, 3:285–305, 2006.

[78] George Ch. Pflug. Scenario tree generation for multiperiod financial optimization by optimal discretization. *Mathematical Programming*, 89(2):251–271, 2001.

[79] Florian A. Potra and Stephen J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1–2):281–302, 2000.

[80] Cornelis Roos, Tamás Terlaky, and Jean-Philippe Vial. *Interior point methods for linear optimization*. Springer, New York, 2nd edition, 2006.

[81] Edward Rothberg and Bruce Hendrickson. Sparse matrix ordering methods for interior point linear programming. *INFORMS Journal on Computing*, 10(1):107–113, 1998.

[82] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley and Sons, New York, 1986.

[83] Ron Shamir. The efficiency of the simplex method: a survey. *Management Science*, 33(3):301–334, 1987.

[84] György Sonnevend. An "analytical centre" for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming. In *System Modelling and Optimization*, volume 84 of *Lecture Notes in Control and Information Sciences*, pages 866–875. Springer, Berlin, 1986.

[85] Richard Tapia, Yin Zhang, Matthew Saltzman, and Alan Weiser. The Mehrotra predictor-corrector interior-point method as a perturbed composite Newton method. *SIAM Journal on Optimization*, 6(1):47–56, 1996.

[86] Tamás Terlaky, editor. *Interior point methods of mathematical programming*. Kluwer Academic Publishers, 1996.

[87] Michael J. Todd and Yinyu Ye. A centered projective algorithm for linear programming. *Mathematics of Operations Research*, 15(3):508–529, 1990.

[88] Stephen A. Vavasis and Yinyu Ye. A primal–dual interior point method whose running time depends only on the constraint matrix. *Mathematical Programming*, 74(1):79–120, 1996.

[89] Margaret H. Wright. Interior methods for constrained optimization. In *Acta Numerica 1992*, pages 341–407. Cambridge University Press, 1992.

[90] Stephen J. Wright. *Primal-dual interior-point methods*. SIAM, Philadelphia, 1997.

[91] Yinyu Ye. *Interior-point algorithm: theory and analysis*. John Wiley and Sons, New York, 1997.

[92] Yinyu Ye, Michael J. Todd, and Shinji Mizuno. An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19:53–67, 1994.

[93] E. Alper Yildirim and Michael J. Todd. Sensitivity analysis in linear programming and semidefinite programming using interior-point methods. *Mathematical Programming*, 90(2):229–261, 2001.

[94] E. Alper Yıldırım and Stephen J. Wright. Warm-start strategies in interior-point methods for linear programming. *SIAM Journal on Optimization*, 12(3):782–810, 2002.

[95] Yin Zhang. On the convergence of a class of infeasible interior-point methods for the horizontal linear complementarity problem. *SIAM Journal on Optimization*, 4(1):208–227, 1994.