A NULL-SPACE PRIMAL-DUAL ALGORITHM FOR NONLINEAR NETWORK OPTIMIZATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF MANAGEMENT SCIENCE AND ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Chih-Hung Lin Candidate
March 2002

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Walter Murray
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Richard W. Cottle

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Michael A. Saunders

Approved for the University Committee on Graduate Studies:

Preface

To Feng-Yin, Winnie, Dora, and my parents,
For their love, affection, and support.

Acknowledgements

I express my sincere appreciation to my advisor, Prof. Walter Murray, for his constant guidance, encouragement, and support during my time at Stanford. He has modeled many of the qualities and characteristics to which I aspire, and I have been greatly inspired by his enthusiasm for both teaching and research.

I am deeply indebted to Prof. Michael A. Saunders, my associate advisor, for being on my orals and dissertation reading committees and suggesting numerous improvements in this thesis. I would also like to thank Prof. Richard W. Cottle for reading my dissertation and agreeing to be the final member of my orals committee. Thank you for such kind comments!

I want to thank my friends who make these years at Stanford more enjoyable. I would like to mention just a few in particular: Chao-Hua Lin, Chao-Yu Lin, Chih-Tung Chan, Eric Wang, Kien-Ming Ng, Paul Yang, Tsung-Ning Liu, Wei-Je Huang and people from volleyball club of Stanford Taiwanese Student Association. Thanks to all my fellow students in System Optimization Lab and professors at Stanford, of which there are too many to mention by name.

Finally, I wish to thank my parents and sisters for their endless support and tremendous love. Most of all, I would like to thank my wife, Feng-Yin Liu, for her encouragement, understanding, patience, and love, and my beloved daughter, Winnie and Dora, for their inspiration in the years of my PhD study.

Contents

Preface **iv**

Acknowledgements **v**

1 Introduction **1**

1.1 Problem Formulation 3

1.2 Logarithmic Barrier Algorithm 7

1.3 Classes of Nonlinear Network Optimization 8

1.3.1 Hydropower generation management 9

1.3.2 Large data bases and statistics estimate 12

1.3.3 Production and inventory planning 15

1.3.4 Transportation and communication network 18

2 Truncated Primal-Dual Algorithm **20**

2.1 Primal-Dual Algorithm 22

2.1.1 Central path and path-following algorithm 22

2.1.2 Primal-dual equation 25

2.2 Null-Space Algorithm 27

2.2.1 Reduced Newton direction 29

2.2.2 Directions of negative curvature and the modified CG-Lanczos method 30

2.2.3 Spanning tree variable reduction basis 31

2.2.4 Preconditioning the Conjugate Gradient method 32

2.2.5 Affine scaling 37

2.2.6 Maximal spanning tree basis 39

3 Linesearch Procedures **44**

3.1 Sufficient decrease conditions 46

3.1.1 Goldstein rule 47

3.1.2 Gamma rule 48

3.2 Linesearch iteration 50

3.2.1 Interval of uncertainty and interval reducing procedure 51

3.2.2 Safeguarded-steplength algorithm 53

3.3 Linesearch for Objective with Logarithmic Barrier Function 54

3.3.1 Initial estimate 59

3.3.2 The procedure for generating subsequent trial steps 63

3.4 Linesearch on the Lagrange Multiplier 67

4 Implementation **70**

4.1 Other Implementation Issues 70

4.2 Experimental Results and Comparisons 75

4.2.1 Test problems 76

4.2.2 Computational results 79

4.2.3 Comparisons 90

5 Summary and Conclusions **93**

Bibliography **95**

List of Tables

2.1 A primal-dual interior-point algorithm 25
2.2 Algorithm for computing multiplication $Z \times v$ 33
2.3 Algorithm for computing multiplication $Z^T \times v$ 34
2.4 A modified PCG-Lanczos algorithm 36
2.5 A spanning tree updating algorithm 41
2.6 Algorithm for updating the spanning tree by inserting an arc 42
2.7 Algorithm for updating the spanning tree by deleting an arc 43

3.1 Algorithm for linesearch iterate 51
3.2 Algorithm for updating the interval of uncertainty 53
3.3 Safeguarded steplength algorithm 55
3.4 Algorithm for initial estimate 64
3.5 Algorithm for generating subsequent Trial Steps 68

4.1 Truncated null-space primal-dual algorithm 75
4.2 Hardware and system configuration 76
4.3 The Buckley test set 77
4.4 Computational results for DSP set 1 81
4.5 Computational results for DSP set 2 82
4.6 Computational results for DSP set 3 83
4.7 Computational results for DSP set 4 84
4.8 Computational results for NFP set 1 85
4.9 Computational results for NFP set 2 86
4.10 Computational results for NFP set 3 87

4.11 Computational results for NFP set 4 88
4.12 Computational results for large DSP 89
4.13 CG iterations comparison 90
4.14 Performance for NPDNET and SNOPT 91
4.15 Performance for NPDNET and SNOPT 92

List of Figures

1.1 The matrix and graph representation for network constraints 5
1.2 Hydropower generation network with 3 reservoirs and 4 time periods 11
1.3 Social Accounting Matrix with three agents 13
1.4 A three time period production model without backlogging 16
1.5 A three time period production model with backlogging 17
1.6 Multi-echelon production model 18

3.1 Illustration for Goldstein rule 48
3.2 Illustration for Gamma rule 50
3.3 EXAMPLE illustrating inefficiency for standard interpolation. 56
3.4 Example 1 of quadratic interpolating for a barrier function 58
3.5 Example 2 of quadratic interpolating for a barrier function 59

4.1 Network interpretation for doubly stochastic problem 78

Chapter 1

Introduction

Network constraints occur commonly in optimization. They arise because networks themselves are a common feature of models. The network is used to describe such infrastructure as a gas network, the web, public highways, etc. Network constraints arise within an optimization problem when one seeks an optimal way to move flow (e.g., goods, signals, cars, electrical currents) on a network (transshipment network, computer network, transportation grid, power grid). Network optimization problems appear in several applications in operations research, transportation, communication network design, hydroelectric power system scheduling, air traffic control, economics, finance, engineering design, manufacturing, and other areas are modeled in the models with network constraints. Some other problems without a network formulation have been solved efficiently only after being reformulated as network optimization problems. The problem formulation is given in section 1.1.

These problems are usually characterized by their very large size. Several theoretical and numerical studies have produced algorithms and software for linearly constrained network programs. As a result of their form, linearly constrained network programs are some of the largest optimization problems solved in practice today. It has been extensively discussed that network-structured optimization problems can be solved substantially faster than the general linearly constrained optimization. As shown by the mid-seventies survey [85], several codes and studies, for example [44] and [75], developed based on the network simplex algorithm for pure network problems were 150–200 times faster than general purpose LP codes of the time. In the early eighties, the research concentration on network optimization moved to the generalized network problem. The network simplex algorithm for the generalized network problem, see [10, 76], was shown to be approximately 50 times faster than LP codes. For linear programming, this observation does not only hold for Simplex type algorithms but also for the recent developments of Karmarkar's algorithm [58], and the ensuing research on interior-point methods. Early attempts to apply interior-point methods to the linear network flow problems can be found in [2, 54, 57, 90, 91]. Recently, Portugal et al. [86] proposed and implemented a truncated primal-infeasible dual-feasible network interior-point method.

Nonlinearities arise due to physical phenomena and economic considerations. In most circumstances, fortunately, only the objective function is affected. In section 1.3, we will discuss some of the real-world problems that lead to nonlinear and, possibly, nonseparable, nonconvex objective functions with network constraints. The highly sparse nature of these problems allows for the solution of very large nonlinear networks. Several authors, includes Beck, Lasdon and Engquist [5], Dembo [20] and [21], Kamesam and Meyer [55], Ahlfeld et al. [1] and Dembo, Mulvey and Zenios [23] have designed nonlinear programming algorithms to exploit the special structure of network problems.

In this thesis, we apply a null-space truncated Primal-Dual linesearch method to solve the nonlinear network problem. It begins with an initial interior solution, and each iterate computes a direction Δx along which some potential function improves. The new iteration $\tilde{x}$ to the problem is moving along the direction Δx from the current iteration $\bar{x}$ and can be written as $\tilde{x} = \bar{x} + \alpha \times \Delta x$, where the steplength α is chosen to maintain the strictly interior and to achieve a reduction in a merit function. The detailed algorithm for finding a suitable steplength is described in chapter 3. An efficient algorithm for computing a truncated primal-dual direction via a null-space method is described in chapter 2.2. In chapter 4, we discuss the details of the implementation, test problems and the computational results. Finally, we give our conclusion in chapter 5 and present some possible extensions to our current work .

1.1 Problem Formulation

Network optimization problems are characterized by their network constraints. Given a network flow problem, there is an associated directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of m nodes and $\mathcal{E}$ is a set of n directed edges. Let (i, j) denote a directed edge from node i to node j. Each arc has associated with it an origin node and a destination node, implying a direction for flow to follow. Usually, arcs have limitations (capacities) on how much flow can move through them. In spite of the capacity limitations on the arcs, flows must satisfy Kirchhoff's Law (conservation of flow), which states that for each node in the network, the sum of all incoming flow plus the flow produced at the node must equal the sum of all outgoing flow plus the flow consumed at the node. The decision variables x_{ij} represent units of flow on the arc (i, j). Therefore, these optimization problems usually can be formulated as linearly constrained network programs of the form

$$\begin{aligned} \text{minimize} \quad & F(x) \\ \text{subject} \quad & \sum_{(j,h)\in\mathcal{E}} x_{hj} - \sum_{(k,j)\in\mathcal{E}} f_{jk}x_{jk} = b_j, \quad j \in \mathcal{V}, && (1.1) \\ & l_{ij} \le x_{ij} \le u_{ij}\ , && (1.2) \end{aligned}$$

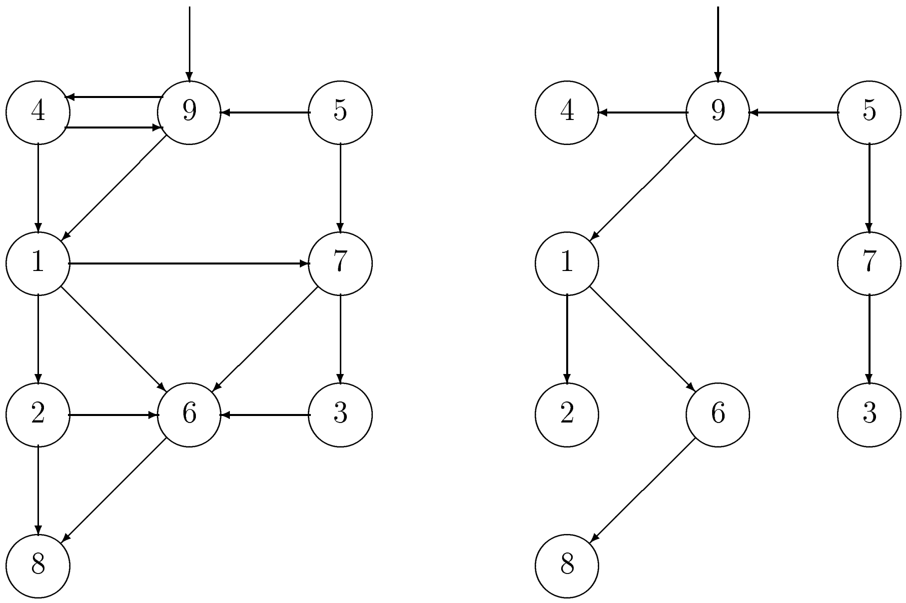
where f_{jk} are the multipliers of the flows on the arc (j, k). If all of the f_{jk} are unity then the problem is termed a pure network optimization problem; otherwise it is termed a generalized network optimization problem. In this thesis, we focus on the algorithm for pure network optimization problems, and the algorithm for generalized network problems will be discussed in chapter 5 as an extension to our current work. The constraints (1.1) are called network constraints (the conservation of flow equations) and constraints (1.2) are called capacity constraints. Based on the constraint set (1.1), nodes with $b_j > 0$ correspond to source nodes in the network, and nodes with $b_j < 0$ correspond to sink nodes. In matrix notation, the network flow problem can be reformulated as the following:

$$\begin{aligned} \text{minimize} \quad & F(x) \\ \text{subject} \quad & Ax = b, \\ & l \leq x \leq u, \end{aligned} \tag{1.3}$$

where A is an $m \times n$ vertex-edge incidence matrix of the network node arc graph $\mathcal{G}$; b, l and $u \in \mathcal{R}^n$ are given vectors; and $x \in \mathcal{R}^n$ is the vector of decision variables. In the other word, each column represents an arc on the network. Arc (i, j) denotes the arc from node i to node j. If (i, j) belongs to the network, there is an associated column of A with exactly two nonzero elements, one in the i-th row is "+1" and the other in the$j$-th row is "-1". The details of the structure of the constraint matrices of network problems is described in [7] and [59]. It is easy to see that the matrix A is not of full row rank because $e^T A = 0$, where $e = (1, \ldots, 1)^T$. One of the most popular and easy ways to make the constraint of matrix full rank is to introduce a root arc from outside to the root node. Suppose that we set the node r as the root node. The modified constraint matrix A will then be

$$A = [A \quad e_r] \ . \tag{1.4}$$

The modified constraint matrix has full row rank and the modified graph is usually called a "rooted network". It is well known [19] that any basis B of a rooted network corresponds to a rooted spanning tree. After nodes reordering (row permutation), the basis can be formed as a lower triangular matrix. Figure 1.1 depicts the matrix and graph representation for some particular network constraints and an associated basis.



The original network structure (matrix as shown in (1.5))

The rooted tree of the network (matrix as shown in (1.6))

Figure 1.1: The matrix and graph representation for network constraints

$$
\begin{pmatrix}
 & & & & 1 & 1 & & -1 & -1 & -1 & & & & & & \\
 & & & & & & & & 1 & & & & -1 & & -1 & \\
 & & & & & & & & & & & 1 & & -1 & & \\
 & 1 & -1 & & -1 & & & & & & & & & & & \\
 & & & -1 & & & -1 & & & & & & & & & \\
 & & & & & & & & & 1 & 1 & & 1 & 1 & & -1 \\
 & & & & & & 1 & 1 & & & -1 & -1 & & & & \\
 & & & & & & & & & & & & & & 1 & 1 \\
1 & -1 & 1 & 1 & & -1 & & & & & & & & & &
\end{pmatrix} \tag{1.5}
$$

$$
\begin{array}{cc}
\begin{pmatrix}
1 & & & & & & & & \\
-1 & 1 & & & & & & & \\
 & & 1 & & & & & & \\
 & -1 & -1 & 1 & & & & & \\
 & & & & 1 & & & & \\
 & & & & -1 & 1 & & & \\
 & & & & & -1 & -1 & & \\
 & & & & & & & 1 & \\
 & & & -1 & & & 1 & -1 & 1
\end{pmatrix}
&
\begin{array}{l}
\to 8 \\ \to 6 \\ \to 2 \\ \to 1 \\ \to 3 \\ \to 7 \\ \to 5 \\ \to 4 \\ \to 9
\end{array}
\end{array}
\tag{1.6}
$$

Note that there are only $2n$ nonzeros out of mn entries of A; the node-arc incidence matrix is not a space-efficient representation of the network. There are many other ways to represent a network. A popular representation is the *node-node adjacency matrix* C, an $m \times m$ matrix with an entry 1 in position (i, j) if arc $(i, j) \in \mathcal{E}$ and 0 otherwise. Such a representation is efficient for dense networks, but is inefficient for sparse networks. A more efficient representation for sparse networks is the *adjacency list*, where for each node $i \in \mathcal{V}$ there exists a list of arcs emanating from node i, i.e. a list of nodes j such that $(i, j) \in \mathcal{E}$. The *forward star* representation is a multi-array implementation of the adjacency list data structure. An adjacency list allows easy access to the arcs emanating from a given node, but no easy access to the incoming arcs. The *reverse star* representation allows easy access to the incoming arcs, but no easy access to the arcs emanating from a given node. Another representation that is much used in interior-point network flow implementations is a simple *arc list*, where the arcs are stored in two linear arrays,"*fromnode*" and "*tonode*". In our implementation, we use the arc list representation to represent a network.

In network optimization, the structure of the matrix (A) is special but the number of variables and constraints are typically extremely large. It is essential to use the special structure to get good performance for solving network optimization problems. The aim of this thesis is to present an efficient algorithm to solve large-scale network optimization problems with a general nonconvex, nonseparable nonlinear objective function.

1.2 Logarithmic Barrier Algorithm

Starting in 1984 with the work of Karmarkar [58], a resurgence of interest has taken place in barrier methods for constrained optimization. Classical barrier methods, which were popular in the 1960s and early 1970s, treat inequality constraints by creating a transformed function containing a positive singularity ("barrier") at the boundary of the feasible region. Consider the constrained optimization problem

$$\begin{array}{rl} \text{minimize} & f(x) \\ \text{subject} & g_i(x) = 0, \quad i = 1, \ldots, m_1, \\ & h_j(x) \leq 0, \quad j = 1, \ldots, m_2. \end{array} \tag{1.7}$$

Suppose that the objective function f and the set of constraint function $\{g_i\}$ and $\{h_j\}$ are smooth. The corresponding logarithmic barrier problem is

$$\begin{array}{rl} \text{minimize} & f_B(x, \mu) = f(x) - \mu \sum_{i=1}^{m_2} \ln h_i(x) \\ \text{subject} & g_i(x) = 0, \quad i = 1, \ldots, m_1, \end{array} \tag{1.8}$$

where μ is a positive scalar called the *barrier parameter*. Note that this barrier problem is defined only at strictly feasible points $\bar{x}$ for which $h_j(\bar{x}) < 0, j = 1, \ldots, m_2$. Let x^* denote a local solution of (1.7) and $x(\mu)$ denote a minimizer of (1.8). Under certain conditions, it can be shown that

$$\lim_{\mu \to 0} x(\mu) = x^* \ .$$

For detail about background and theory, please refer to [28, 30, 100].

Most barrier methods proposed since 1984 are designed for special cases of linear programming (LP) and convex quadratic programming, and the vast majority of papers on these topics have concentrated on proofs of polynomial complexity (see [45] and [48] for more bibliographies). After that, several barrier methods for convex nonlinear programming have been proposed and analyzed from the viewpoint of complexity. In recent years, interior-point methods for general nonlinear programming have received considerable attention because of their close relationship with the "new" polynomial approaches to linear and quadratic programming. On the practical side, barrier techniques are increasingly being applied to various general nonlinear optimization problems, including nonconvex and large-scale problems [79, 92].

Barrier methods fell from favor during the 1970s partly because inherent ill-conditioning in the Hessian matrices created difficulties for standard unconstrained methods. A major issue today is the development of strategies for dealing with this ill-conditioning; for a discussion of some approaches, see [100]. A second inconvenient feature of barrier functions is that they tend to cause inefficiencies for general-purpose linesearch techniques. In chapter 3, we discuss the role of a linesearch in optimization algorithms, and explain why a special-purpose linesearch procedure may be helpful in dealing with barrier functions.

The method we consider in this thesis is motivated by the application of the logarithmic barrier function technique to problem (1.3). In a barrier function method to (1.3), the problem to be solved is (NLP$_\mu$):

$$\begin{array}{ll} \text{minimize} & F(x) - \mu \sum_{i=1}^{n} (\ln(x_i - l_i) + \ln(u_i - x_i)) \\ \text{subject to} & Ax = b, \\ & l < x < u, \end{array} \tag{1.9}$$

where $\mu > 0$ is the barrier parameter. Here, $\ln x_i$ denotes the logarithm of x_i to the natural base. To simplify the notation, we let

$$F_B(x) = F(x) - \mu \sum_{i=1}^{n} (\ln(x_i - l_i) + \ln(u_i - x_i)) \; . \tag{1.10}$$

1.3 Classes of Nonlinear Network Optimization

In this section we establish a general taxonomy for nonlinear programming problems with network structure. We survey in detail the primary areas of application for the nonlinear network optimization model, mainly from [23] and [50]. Most of the examples discussed here are large scale and are difficult or inefficient to solve using general-purpose software.

1.3.1 Hydropower generation management

Network problems arise in planning the operation of a hydro system. The problem is that of maximizing the hydropower generated along a time horizon by a multi-reservoir power system. The nodes represent the reservoirs and the arcs correspond to the sections of the river that connect the reservoir. This basic network is then replicated on a number of segments, corresponding to the time periods (usually weeks) included in the planning horizon, which are linked by special arcs on some of the nodes.

The decision variables in the optimization problem are the amount of water to be released from each reservoir to its direct downstream reservoirs in a given period (corresponding to the arcs in the basic network), or the amount of water to be stored in the reservoir from one period to the next (the special arcs linking the segments). This amount becomes available to be released from storage in the next period.

The constraints are the linear network constraints that ensure flow balance in each reservoir, and simple bounds on the variables. The purposes of these bounds are

1. to ensure that the water released serves the desired flood control, irrigation and navigation purposes;
2. to ensure that the amount of water released from a given reservoir to any of its directly downstream reservoirs does not exceed the canal capacity;
3. to penalize the amount of stored water that exceeds a safety capacity for any given reservoir;
4. to force the amount of water stored in the reservoirs to remain below a given upper bound.

The objective consists of the maximization of a nonlinear function, typically the amount of hydroelectricity generated from the system, or the saving in the cost of thermal generated power to the revenue from hydroelectricity generated from the system over the planning horizon.

We now introduce a formal model from [27] for the problem described above. Here, we denote

T the time periods set,

R the set of reservoirs,

E the set of reservoirs that are not used for hydropower generation ($E \subset R$),

U_j the set of reservoirs directly upstream from reservoir $j \in R$,

D_j the set of reservoirs directly downstream from reservoir $j \in R$,

r_{tji} the amount of water released from reservoir j to reservoir $i \in D_j$ in period $t \in T$,

s_{tj} the amount of water stored in reservoir j at the beginning of period t,

b_{tj} the exogenous inflow to reservoir j at the beginning of period t.

The flow balance equations in each reservoir for each time period are as follows:

$$-\sum_{i \in U_j} r_{tij} - s_{tj} + \sum_{i \in D_j} r_{tji} + s_{t+1,j} = b_{tj} \quad \forall j \in J,\ t \in T, \tag{1.11}$$

where s_{tj} is fixed for $t = 1$ and $t = |T| + 1$. Furthermore, the bounds on the arcs of the replicated network are given by

$$l_{tji} \leq r_{tji} \leq u_{tji} \quad \forall i \in D_j,\ j \in J,\ t \in T, \text{ and} \tag{1.12}$$

$$m_{tj} \leq s_{tj} \leq M_{tj} \quad \forall j \in J,\ t \in T. \tag{1.13}$$

The constraints (1.11)–(1.13) can be written as the network constraints (1.3). An example of the graph interpretation for a Hydropower generation network with 3 reservoirs and 4 time periods is given in Figure 1.2.

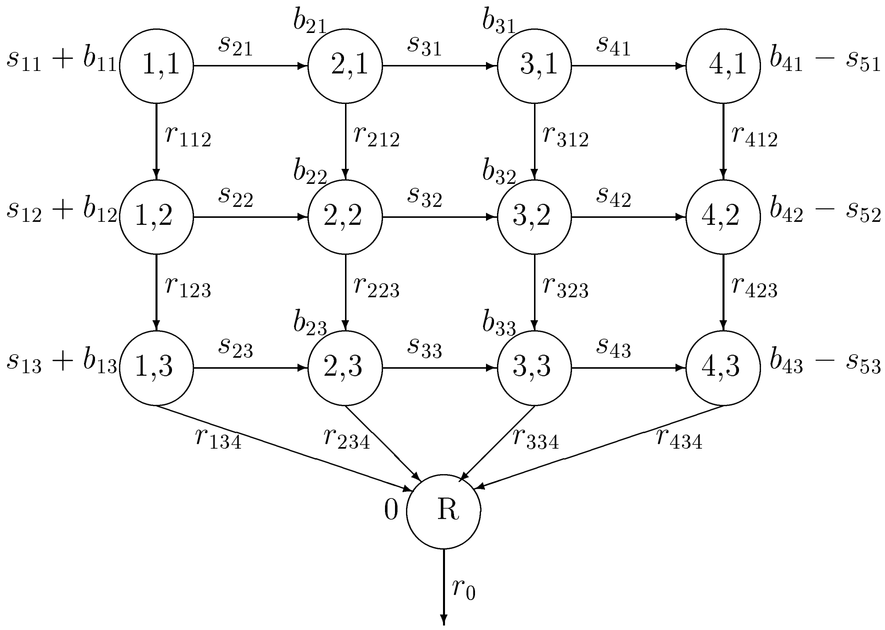


Figure 1.2: Hydropower generation network with 3 reservoirs and 4 time periods

According to the objective, the hydropower generated from the water released from reservoir j to reservoir i at time period t can be express as

$$h_{tji} = K_{tji} r_{tji} \quad \forall\, t \in T,\ j \in J \backslash E,\ i \in D_j\ , \tag{1.14}$$

where K_{tji} may be a linear or nonlinear function of the s-variables for the reservoirs in set $J \backslash E$, that can be approximated by

$$K_{tji} r_{tji} \approx g_{tji}(s_{tj},\ s_{t+1,j}) \quad \forall\, t \in T,\ j \in J \backslash E,\ i \in D_j\ . \tag{1.15}$$

Therefore, the objective can be approximated by some polynomial function of r and s.

The formulation given above is under the assumption that the exogenous incoming flow for each reservoir and each time period is known and given. In the real world, those parameters are usually considered as random variables. Decomposition approaches have been used by researchers, including [84], [83], [93], and [49], to solve this problem with remarkable success. Most of these approaches are based on Benders decomposition. In [27], an augmented Lagrangian based algorithm was proposed and implemented with good success.

This model can become large rather quickly when used for long or even medium term planning. The problem solved in [21] was a nonlinear and nonconvex network problem with over 18,000 bounded variables and 5,000 network flow-conservation constraints. Currently, there are a number of utilities that use nonlinear network optimization to schedule dam releases. At present, there is one company, namely Hidroelectrica Española S.A., that does all its long-term scheduling and planning using a nonlinear network optimization model solved by the specialized code NLPNET [20].

1.3.2 Large data bases and statistics estimate

Timely collection of very large data bases has become increasingly important over the past two decades. Many government agencies and private companies routinely depend upon these files for maintaining their operation. One important class of data bases is known as microdata, whereby the file consists of a large number of individual decision units: individuals, families, corporations, etc. Typically, the size of a microdata file ranges from several thousand to billions of observations. Once collected, the raw data must be processed before it can be presented to the computer user. In the data handling process, there are numerous steps involved, several of which employ network optimization. In this section, we discuss one of the most prominent examples: Estimating Social Accounting Matrices.

A Social Accounting Matrix ($X = [x_{ij}]_{i=1,\ldots,n}^{j=1,\ldots,n}$), or SAM, is a square matrix whose components represent the flow of funds between the national income accounts of a country's economy at a fixed point of time. Each index (a row and a column) represents an account or an agent in the economy. Component x_{ij} is positive if agent j receives funds from agent i. A SAM is a snapshot of the critical variables in a general equilibrium model describing the circular flow of financial transactions in an economy.

Typically, the agents of an economy include institutions, factors of production, households, and the rest of the world to account for transactions with the economies of other countries. For example, "the production activities" generate added value that flows to the factor of production, land, labor and capital. The factor "income" is the primary source of income for institutions, including households, government and firms, who purchase "goods and service" supplied by production activities, thereby completing the cycle. This simple example produces a SAM with three agents and three nonzero components as shown in Figure 1.3. Of course, to be useful for equilibrium modeling, this highly aggregated model must be disaggregated into sub-accounts for each sector of the economy. The table may reach a few hundred to a few thousand agents in size.

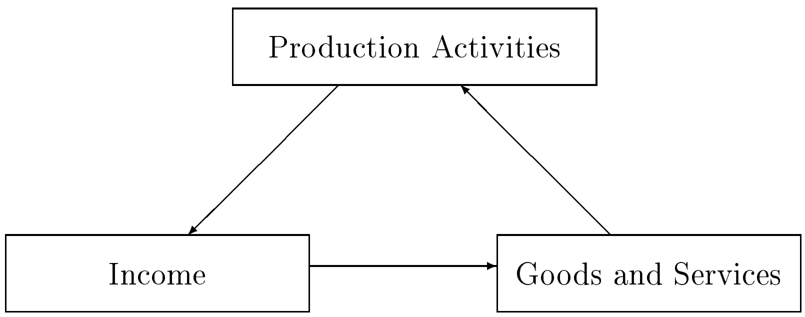


Figure 1.3: Social Accounting Matrix with three agents

One problem encountered in using these tables is the following:

1. the elements in the table are usually calculated from different agents in an economy and are often obtained through sampling procedures,

2. the total expenditure (or income) of the agents is normally available through government sources and is more accurate and up-to-date.

Therefore, the model should be formulated as follows: given a SAM whose elements are out-of-date, compute updated values for these elements that satisfy a pre-specified set of row and column sums. Such problems arise not only in economics but also in contingency table analysis, in statistical application, in analysis of congestion in telecommunication networks, and in demographic studies in the social sciences.

We now discuss a formal model from [23] for the problem described above. First, we denote

G the set of index with positive components,

$\bar{x}_{ij}$ the given value of entries in the matrix,

x_{ij} updated new value of the entries in the matrix,

r_i the given new row sum for the i-th row,

c_j the given new column sum for the j-th column.

The new entries in the matrix must therefore satisfy

$$\sum_{(i,j)\in G} x_{ij} = r_i \ \forall i = 1, \ldots, n \tag{1.16}$$

$$\sum_{(i,j)\in G} x_{ij} = c_j \ \forall j = 1, \ldots, n \tag{1.17}$$

$$x_{ij} \geq 0 \ \forall (i,j) \in G \tag{1.18}$$

The manner in which the new entries are postulated to relate to the old entries depends on the behavioral assumptions that are most plausible in the particular model under study. Here, we give a popular example of the objective function. Suppose that our objective is to find x_{ij} as "close" as possible to $\bar{x}_{ij}$ in a least square criterion. This can be formulated as

$$\min_{x_{ij}} \sum_{(i,j)\in G} (x_{ij} - \bar{x}_{ij})^2 \ . \tag{1.19}$$

Therefore, (1.16)–(1.19) constitute a quadratic optimization with network flow conservation (transportation) constraints. Introducing a weighting mechanism of the form $(1/\bar{x}_{ij})(x_{ij} - \bar{x}_{ij})$ does not change the quadratic nature of the model. The optimal solution will be a chi-square estimate of the new matrix.

According to the size of the problem encountered in the matrix estimation, one of the test case in [23] is derived from the national economy of Thailand. It results in a nonlinear network optimization problem with about 2,200 decision variables and about 1,100 constraints. On the other hand, network models have been used to estimate SAMs for development planning by the World Bank. A modeling system, GAMS/SAMBAL of [107], provides the interfacing of network software with high level modeling languages and the interaction with the SAM data base. The GAMS/SAMBAL system is also being used for compiling regional tables for national data by researchers at the Social Sciences Data Center of the University of Pennsylvania. A comprehensive survey of matrix balancing applications and related algorithmic work is given in [94].

1.3.3 Production and inventory planning

Another major area in which network models arise is production and inventory planning. This includes the popular economic lot size problem and the capacity expansion problem. In this area numerous models are found for optimal production and inventory scheduling. These models originate from viewing the production processes dynamically, i.e., as occurring over discrete time intervals. The resulting models involve establishing flows over structured networks, with the network structure varying according to the modeling assumptions. The networks tend to be staged, with the stages corresponding to time periods. Units flowing through the network correspond to production and inventory, with node requirements corresponding to customer demands. Production demands for all time periods are assumed to be known in advance. Concave objective functions occur naturally for this application. Start-up costs reflect facility and machine start-up, and storage requirements. Economies of scale are reflected in storage requirements, shipping, and material purchases. Note that start-up costs can occur at any time boundary, resulting in objective functions that are piecewise concave. The following is a series of production-based models that demonstrate the correspondence between network structure and model assumptions.

The basic model in this area is the Wagner-Whitin model for a production and inventory system with no backlogging of demand [106]. The variables for the model are defined as follows:

n number of time periods i

x_i amount of the product produced in period i,

r_i market requirements for time period i,

I_i inventory of the product in period i, $I_i = \sum_{k=1}^{i}(x_k - r_k)$

$P_i(x_i)$ cost of producing x_i units in period i,

$H_i(I_i)$ cost of inventory holdings I_i, in period i.

Restrictions on the model include nonnegative production and production capacity $0 \le x \le u$, and no backlogging of unsatisfied demand and facility capacity, $0 \le I_i \le U$. The resulting model becomes

$$\begin{aligned} \min_{x_i,\, I_i} \quad & \sum_{i=1}^{n} P_i(x_i) + \sum_{i=1}^{n-1} H_i(I_i) \\ s.t. \quad & x_i + I_{i-1} - I_i = r_i,\ i = 1, \ldots, n \\ & 0 \le x \le u,\ 0 \le I_i \le U\ , \end{aligned} \tag{1.20}$$

with given I_0 and I_n. The network interpretation for a three time periods model is depicted in Figure 1.4.

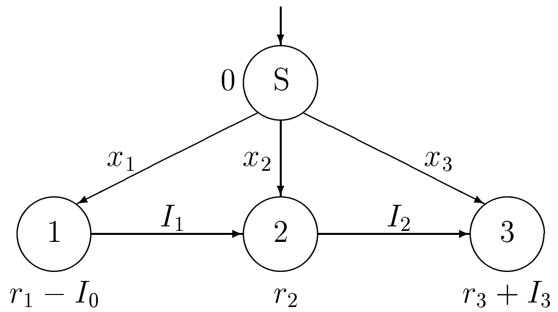


Figure 1.4: A three time period production model without backlogging

The effect of allowing backlogging of demand on the previous model is explored by Zangwill [105, 106]. Here we introduced the extra variables B_i to be the amount of shortage for period i, indicating that demands in a time period can be satisfied in a later period. The resulting extended network is presented in Figure 1.5; the backward arcs correspond to backlogged demand. The objective functions should add the shortage cost $\sum_{i=1}^{n-1} T_i(B_i)$.

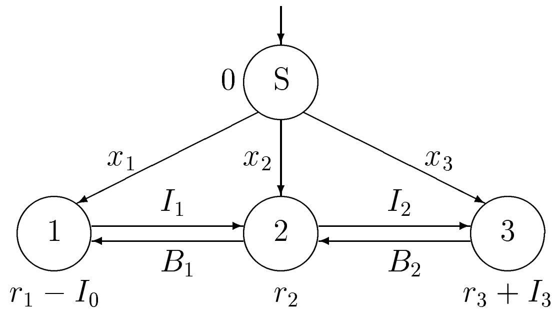


Figure 1.5: A three time period production model with backlogging

Both of the previous models assumed a single production facility and a single commodity without gains or losses. Generalizing the model to a product that requires a series of processes, each process performed in a separate facility, results in a multi-echelon economic lot size model [106]. The network for a three-facility, three-time period model (without backlogged demand) is shown in Figure 1.6. Facility 0 is the origin of the material required for processing. In this model, flow from $(t, i-1)$ to (t, i) corresponds to production in time period t at facility i and flow from (t, i) to $(t+1, i)$ corresponds to inventory in period t at facility i. Konno considers extensions to the multi-echelon model, including backlogging [65] and a time-lag for processing [66].

A natural extension to the basic dynamic lot size problem is to allow for multiple production facilities for the same product and account for shipping requirements between production regions and demand regions. Location becomes important in this model as the option to manufacture in one region to satisfy demand in another region becomes valid due to considerations such as production start-up costs. Included in this class is the capacity expansion model. This model permits the cost comparison of expanding production capability at one site versus shipping products from another site. Manne and Veinott [73] present a single production region mode with linear transportation costs and concave expansion costs. Erlenkotter [26] considers two production regions with similar cost functions. Fong and Rao [35] extend this two

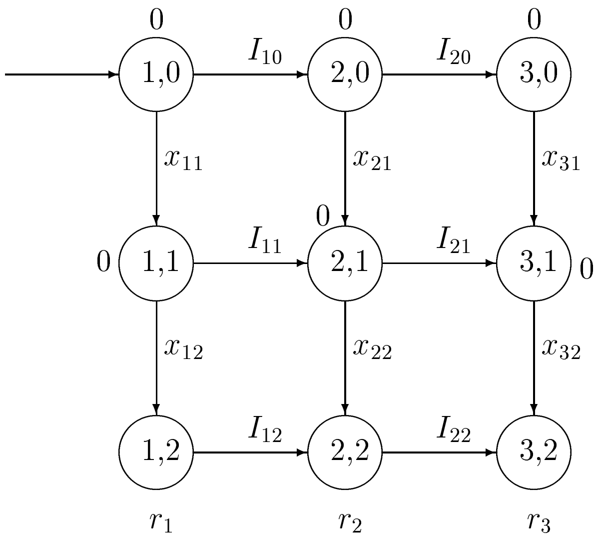


Figure 1.6: Multi-echelon production model

production region model to include concave costs for both transportation costs and capacity expansion.

1.3.4 Transportation and communication network

One of the most natural network models arises in the area of transportation planning and communication network design. Transportation planning arises in a number of general models, moving equipment to a road construction site, or leasing a truck for transporting goods. The resulting networks are as general as the applications, with the common feature that the networks tend to be sparse. This application area also includes problems that have become well known in their linear version, including the *Hitchcock Transportation Problem*. Applications for this case include problems original from Navy supply systems [68], and a cotton gin plant location problem [63]. If the Hitchcock problem is generalized to include intermediate nodes with zero demand and the uncapacitated case is considered, the transshipment problem is obtained [81]. A significant amount of effort in transportation modeling has been performed by Florian et al. [18, 33, 34], who develop transportation models for application including freight transportation by rail, traffic analysis, and packet switching networks. Other transportation models are discussed by Magnanti and Wong [72], and Klincewicz [61] considers freight transport problems in the context of consolidation areas of product transport. LeBlanc [70] also discusses a rail network design problem. Hall [51] discusses a graphical interpretation of some transportation problems and the effect of nonlinear costs. A specific pipeline transportation model has been studied by Bulatov [11].

The Transportation Problem is an often used model for transportation applications. Since the basic network is fixed for this model, applications give rise to a range of objective functions. The objective functions are chosen to reflect various assumptions concerning the cost of transporting items throughout the network.

Communications networks have a natural interpretation in the context of transportation and transshipment models. In this setting the traffic consists of point-to-point messages, and the networks consist of communication media instead of roads or routes. Economies of scale could correspond to reduced cost per message at higher utilization for a link, or to the communication medium used. A major assumption in using this type of model is that customer demands are constant for the time period of the model. These models are characterized by their size, which is usually large, and the generality of the underlying networks. Specific models for communication network design have been developed in [4, 53, 103, 104].

Chapter 2

Truncated Primal-Dual Algorithm

In this chapter, we describe a Newton-type algorithm model for solving network constrained optimization problems (1.3) with network constraints and bounded variables. The algorithm model is based on the definition of a continuously differentiable exact merit function that follows an exact penalty approach for the box constraints.

The proposed algorithm for the network constrained optimization problems is to solve a sequence of linearly equality constrained minimization problems (1.9) parameterized by the positive barrier parameter μ. The primal-dual methods described here involve outer and inner iterations. Each outer iteration is associated with an element of a decreasing positive sequence of parameters $\{\mu_j\}$ such that $\lim_{j\to\infty} \mu_j = 0$. The inner iterations correspond to an iterative process for solving the linearly equality constrained minimization problem (1.9) for a given μ. The first-order optimality conditions for the linearly equality constrained minimization problem are discussed in section 2.1. They imply that there exist vectors $(x(\mu), \lambda(\mu))$ satisfying the conditions. The vector $\lambda(\mu)$ can be interpreted as an estimate of λ^*, the Lagrange multipliers of the original network optimization problem (1.3). Fiacco and McCormick [29] give the conditions under which local solutions $(x(\mu), \lambda(\mu))$ converge to (x^*, λ^*) as $\mu \to 0$. When $(x(\mu), \lambda(\mu))$ is regarded as a function of the parameter μ, the set of minimizers $x(\mu)$ defines a continuously differentiable path known as the trajectory or the central path.

For large-scale problems, it may be impractical to use the interior-point method based on Newton's method. In this thesis, we use a truncated primal-dual algorithm [79, 80, 100, 9, 52], in which an approximation to the Newton search direction is used. The essential features of the truncated primal-dual algorithm are that each barrier subproblem (1.9) is solved approximately using a truncated-Newton method. Then the solutions to the subproblems are extrapolated to obtain an initial guess for a new subproblem. Many of the enhancements for the truncated primal-dual algorithm, such as preconditioning, a special matrix-vector product, and a numerically stable formula for the search direction, are discussed in [79]. A convergence proof and complexity analysis of a truncated primal-dual algorithm were given by Nash and Sofer [80].

Because Newton's method is based on a Taylor's series expansion near the current solution estimate, there is no guarantee that the search direction it computes will be as crucial far away from (x^*, λ^*). As discussed in Dembo, Eisenstat and Steihaug [22], at the beginning of the solution process, a reasonable approximation to the Newton's direction may be almost as effective as the Newton's direction itself. It is only gradually, as the solution is approached, that Newton's direction takes on more and more meaning. This suggests using an iterative method to solve the Newton's equation. Moreover, it should be an iterative method with a variable tolerance, so that far away from the solution, the Newton's equation is not solved to undue accuracy. Only when the solution is approached, we should consider expending enough effort to compute something like the exact Newton's direction.

Sherman [95] suggested using Successive-Over-Relaxation (SOR), one of the simplest of a whole class of methods that have been found to be effective for solving linear systems arising in partial differential equations. However, it is difficult to get SOR methods to perform well on general problems. Also, they appear to be prohibitively expensive to use in the context of truncated-Newton's methods.

In this application, Conjugate Gradient (CG) and Preconditioning Conjugate Gradient (PCG) are much better suited and have become popular. Although CG is ideal for problems where the matrix has only one whose eigenvalues lie is a small number of clusters, it is guaranteed to converge (in exact arithmetic) in at most n iterations. However, the requirement of both of these methods is that the matrix must be positive-definite. The (reduced) Hessian matrix is only guaranteed to be positive semi-definite at the solution and may be indefinite elsewhere. There is a close relation between CG and Lanczos, so it is possible to recover the Lanczos vectors and the tridiagonal matrix from CG vectors and coefficients. One can then compute the smallest Ritz value and vector, which are optimal in the Krylov subspace we work in. However, the work and storage requirements then become similar to those of Lanczos. Here, we use the Modified CG-Lanczos Method proposed by Boman [8]. In this method, we first use CG to solve the linear system, called reduced Newton system, and switch to the Lanczos method to find a better direction of negative curvature when that is necessary.

In section 2.2.3, we present a special null-space basis-Spanning Tree Variable Reduction basis, which can take an advantage of the network constraints and save computation on the matrix vector multiplication, the most computationally intensive part of CG. To reduce the number of CG iterates, we describe the PCG method with a nonbasic diagonal preconditioner, with respect to the maximal spanning tree basis, to solve the system of equations in section 2.2.4 and a maximal spanning tree updating scheme in section 2.2.6.

2.1 Primal-Dual Algorithm

We first assume that F is twice-continuously differentiable ($\in C^2$), and that A is a matrix of full row rank.

2.1.1 Central path and path-following algorithm

By introducing slack variables, we can rewrite problem (1.9) as follows:

$$\begin{array}{rl} \text{minimize} & F(x) - \mu \sum_{i=1}^{n} (\ln(s_{li}) + \ln(s_{ui})) \\ \text{subject} & Ax = b, \\ & x - s_l = l \\ & x + s_u = u \\ & s_l > 0, \; s_u > 0 \; . \end{array} \tag{2.1}$$

To characterize the solution of the barrier problem (2.1), we introduce its Lagrangian function,

$$\begin{aligned} L(x, s, \lambda_A, \lambda_l, \lambda_u) &= F(x) - \mu \sum_{i=1}^{n} (\ln(s_{li}) + \ln(s_{ui})) + \lambda_A^T (Ax - b) \\ &\quad + \lambda_l^T (-x + s_l + l) + \lambda_u^T (x + s_u - u) \ , \end{aligned} \tag{2.2}$$

where λ_A, λ_l and λ_u are Lagrange multipliers with respect to the equality constraints $Ax - b = 0$, $-x + s_l + l = 0$ and $x + s_u - u = 0$. The optimality (KKT) conditions for (2.1) are

$$\begin{aligned} \frac{\partial L}{\partial x} &= g + A^T \lambda_A - \lambda_l + \lambda_u = 0 \\ \frac{\partial L}{\partial s_l} &= \lambda_l - \mu (S_l)^{-1} e = 0 \\ \frac{\partial L}{\partial s_u} &= \lambda_u - \mu (S_u)^{-1} e = 0 \\ \frac{\partial L}{\partial \lambda_A} &= Ax - b = 0 \\ \frac{\partial L}{\partial \lambda_l} &= -x + s_l + l = 0 \\ \frac{\partial L}{\partial \lambda_u} &= x + s_u - u = 0 \\ & \quad s_l > 0, \ s_u > 0, \ \lambda_l > 0, \ \lambda_u > 0 \ , \end{aligned} \tag{2.3}$$

with $S_l = \mathrm{diag}(s_1^l, \dots, s_n^l)$ and $S_u = \mathrm{diag}(s_1^u, \dots, s_n^u)$. Several papers including [101] and [17] have given arguments suggesting that the primal search direction will often cause the slack variables to become negative, and that can be inefficient. Research in linear programming [102] and nonlinear programming [12] has shown that more effective interior-point methods can be obtained by using the modified KKT system

$$\begin{aligned} & g + A^T \lambda_A - \lambda_l + \lambda_u = 0 \\ & S\lambda - \mu e = 0 \\ & S = \begin{pmatrix} S_l & \\ & S_u \end{pmatrix} \ , \quad \lambda = \begin{pmatrix} \lambda_l \\ \lambda_u \end{pmatrix} \end{aligned}$$

$$\begin{aligned}
&Ax - b = 0 \\
&-x + s_l + l = 0 \\
&x + s_u - u = 0 \\
&s_l > 0,\ s_u > 0,\ \lambda_l > 0,\ \lambda_u > 0\ ,
\end{aligned} \tag{2.4}$$

which is obtained by multiplying both sides of second and third conditions of (2.3) by S_l and S_u respectively. Although (2.3) and (2.4) theoretically have the same solution, applying Newton's method to them will produce different iterates. Furthermore, system (2.4) has the advantage that the derivatives of $S\lambda - \mu e = 0$ are bounded as any slack variables approach zero, which is not the case with (2.3). As in [98, 102, 101, 12], analysis of the primal-dual step, as well as computational experience with linear and nonlinear programs, has shown that it overcomes the drawbacks of the primal-step.

Rather than solving each barrier subproblem (2.1) accurately, we will be content with an approximate solution $(\hat{x}, \hat{s})$ satisfying $M(x, s, \lambda_l, \lambda_u, \mu) < \epsilon_\mu$, where $M(\cdot)$ measures the optimality condition of the barrier problem (2.1) and is defined by

$$M(x, s, \lambda_l, \lambda_u, \mu) = \max\left\{\|Z^T(g - \lambda_l + \lambda_u)\|_\infty,\ \|Ax - b\|_\infty,\ \|S\lambda - \mu e\|_\infty\ \right\}\ , \tag{2.5}$$

where Z is a basis for the null-space of A. The form of Z is discussed later in section 2.2.3. The tolerance ϵ_μ, which determines the accuracy in the solution of barrier subproblems, is decreased from one subproblem to the next and must converge to zero. In this paper, we set $\epsilon_\mu = \vartheta_1 \mu$ and $\mu^{k+1} = \vartheta_3 \mu^k$ with $0 < \vartheta_3 < 1$ if $M(x^k, s^k, \lambda_l^k, \lambda_u^k, \mu^k) < \epsilon_{\mu^k}$ and $mineig(Z^T HZ) > -\vartheta_2 \mu$. We test optimality for the nonlinear network optimization problem (1.3) by means of $M(x^k, s^k, \lambda_l^k, \lambda_u^k, 0)$ and $mineig(Z^T HZ) \geq 0$. The primal-dual interior-point algorithm is described in Table 2.1 and a proof of convergence for this algorithm is given in Theorem 1.

Theorem 1 *The primal-dual interior-point algorithm described in Table 2.1 will terminate in a finite number of steps.*

Proof In [36], Forsgren and Murray defined a class of algorithms that generate a sequence of iterates x_μ^k such that $\lim x_\mu^k = x^*(\mu)$. Consequently, such an algorithm will

Table 2.1: A primal-dual interior-point algorithm

Input: x_0, μ_0, A
Output: x^*, λ^*

set s_0, λ_0
$k = 0$
repeat
if $(M(x^k, s^k, \lambda_l^k, \lambda_u^k, \mu^k) < \vartheta_1 \mu^{k-1}$ and $mineig(Z^T HZ) > -\vartheta_2 \mu)$
$\mu^k = \vartheta_3 \mu^{k-1}$
else
$\mu^k = \mu^{k-1}$
end if
find either a sufficient descent directions
or a direction of sufficient negative curvature Δx_k
find Δs^k and $\Delta \lambda^k$
find a suitable steplength α_x^k
$x^{k+1} = x^k + \alpha_x^k \Delta x^k$
$s^{k+1} = s^k + \alpha_x^k \Delta s^k$
find a suitable steplength α_λ^k
$\lambda^{k+1} = \lambda^k + \alpha_\lambda^k \Delta \lambda^k$
$k = k + 1$
until $M(x^k, s^k, \lambda_l^k, \lambda_u^k, 0) < \epsilon_{PD}$ **and** $mineig(Z^T HZ) \geq 0$

find $x^*(\mu)$ after a finite number of iterations. Since we generate a sequence of sufficient descent directions and/or directions of sufficient negative curvature, the primal-dual interior-point algorithm described in Table 2.1 falls into the class of algorithms that are defined in [36]. Therefore, it will terminate in a finite number of steps. ■

2.1.2 Primal-dual equation

Suppose that the current iterate is $(\bar{x}, \bar{s}, \bar{\lambda})$ and the search direction $(\Delta x, \Delta s, \Delta \lambda)$ is computed by applying Newton's method to the system of equations (2.4). Under the assumption that F is twice-continuously differentiable ($\in C^2$) and that A is a matrix of full row rank, the search direction will be the unique solution that satisfies the linear system

$$\begin{bmatrix} H & 0 & 0 & A^T & -I & I \\ A & 0 & 0 & 0 & 0 & 0 \\ -I & I & 0 & 0 & 0 & 0 \\ I & 0 & -I & 0 & 0 & 0 \\ 0 & \bar{\Lambda}_l & 0 & 0 & \bar{S}_l & 0 \\ 0 & 0 & \bar{\Lambda}_u & 0 & 0 & \bar{S}_u \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta s_l \\ \Delta s_u \\ \Delta\lambda_A \\ \Delta\lambda_l \\ \Delta\lambda_u \end{bmatrix} = - \begin{bmatrix} g_{\bar{x}} + A^T\bar{\lambda}_A - \bar{\lambda}_l + \bar{\lambda}_u \\ A\bar{x} - b \\ -\bar{x} + \bar{s}_l + l \\ \bar{x} + \bar{s}_u - u \\ \bar{S}_l\bar{\lambda}_l - \mu e \\ \bar{S}_u\bar{\lambda}_u - \mu e \end{bmatrix} . \tag{2.6}$$

Since $-x + s_l + l = 0$ and $x + s_u - u = 0$ at each iteration, the linear system can be simplified to

$$\begin{bmatrix} H & A^T & -I & I \\ A & 0 & 0 & 0 \\ \bar{\Lambda}_l & 0 & \bar{S}_l & 0 \\ -\bar{\Lambda}_u & 0 & 0 & \bar{S}_u \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda_A \\ \Delta\lambda_l \\ \Delta\lambda_u \end{bmatrix} = - \begin{bmatrix} g_{\bar{x}} + A^T\bar{\lambda}_A - \bar{\lambda}_l + \bar{\lambda}_u \\ A\bar{x} - b \\ \bar{S}_l\bar{\lambda}_l - \mu e \\ \bar{S}_u\bar{\lambda}_u - \mu e \end{bmatrix} . \tag{2.7}$$

The third and fourth equation of (2.7) implies

$$\Delta\lambda_l = -\bar{S}_l^{-1}\bar{\Lambda}_l\Delta x - (\bar{\lambda}_l - \mu\bar{S}_l^{-1}e) \tag{2.8}$$

$$\Delta\lambda_u = \bar{S}_u^{-1}\bar{\Lambda}_u\Delta x - (\bar{\lambda}_u - \mu\bar{S}_u^{-1}e) . \tag{2.9}$$

Replacing $\Delta\lambda_l$ and $\Delta\lambda_u$ in the first equation of (2.7) by (2.8) and (2.9), we get

$$(H + \Omega)\Delta x + A^T\Delta\lambda_A = -g_{\bar{x}} - A^T\bar{\lambda}_A + \mu\bar{S}_l^{-1}e - \mu\bar{S}_u^{-1}e , \tag{2.10}$$

where $\Omega = \bar{S}_l^{-1}\Lambda_l + \bar{S}_u^{-1}\Lambda_u$. From the definition of F_B in (1.10), we know

$$g_{\bar{x}}^B = \nabla F_B(\bar{x}) = g_{\bar{x}} - \mu\bar{S}_l^{-1}e + \mu\bar{S}_u^{-1}e . \tag{2.11}$$

Therefore, we can get the search direction by first solving

$$\begin{bmatrix} (H+\Omega) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda_A \end{bmatrix} = - \begin{bmatrix} g_{\bar{x}}^B + A^T\bar{\lambda}_A \\ A\bar{x} - b \end{bmatrix} \tag{2.12}$$

and then obtaining Δs_l, Δs_u, $\Delta\lambda_l$ and $\Delta\lambda_u$ from $\Delta s_l = \Delta x$, $\Delta s_u = -\Delta x$ and (2.8) and (2.9). The system of equations (2.12) is usually called the *primal-dual equation*. Hereafter, we refer to the matrix on the left-hand side of (2.12) as the *KKT-matrix* and denote it by K.

2.2 Null-Space Algorithm

One of the most fundamental requirements of an interior-point implementation for network optimization is an efficient implementation of an iterative method to compute the search direction at each iteration. One can solve (2.12) for the search direction or solve the mathematically equivalent system,

$$Z^T(H+\Omega)Zd_Z = -Z^T\xi \ , \tag{2.13}$$

for some $\xi \in \mathcal{R}^n$. The actual equation is derived in section 2.2.1.

Although (2.12) and (2.13) are mathematically equivalent, they differ in the amount of work required to obtain the search direction. To obtain the search direction from (2.12), the KKT-matrix is required. If H and A are sparse, they yield a sparse K and the nonzero elements of K keep the same pattern. It is possible to take advantage of the sparsity of the problem if equations involving K are solved. In [36], Forsgren and Murray consider the LBL^T factorization of K. Such a factorization computes

$$\Pi^T K \Pi = LBL^T \ , \tag{2.14}$$

where Π is a permutation matrix, L is a unit lower-triangular matrix, and B is a symmetric block-diagonal matrix whose diagonal blocks are size 1×1 or 2×2.

To obtain the search direction for (2.13), the reduced Hessian is required. In the general linear equality-constrained context, a matrix Z may be obtained by forming the LU-factorization of A; see [42] for details. As discussed in [41], the matrix $Z^T(H+\Omega)Z$ may be completely dense, and the amount of work to form $Z^T(H+\Omega)Z$ may be prohibitive even if Z is sparse. However, we intend to solve (2.13) for search direction via an iterative method in our null-space algorithm. We use the modified CG-Lanczos algorithm described in [8] to solve (2.13). Although CG has been widely used for primal-dual algorithms for LP and SQP, etc, a key difference is that we are using it neither on a KKT system nor on a range-space system but on a reduced-Hessian system. When we use it on a range-space system, it is necessary to recover the search direction in x-space if we solve the system of equations approximately, whereas when we use it on the null-space system and get the search direction in x-space, it is not necessary to recover the search direction in multiplier-space even if we do not solve the reduced-Hessian system accurately. In the modified CG-Lanczos algorithm, it is not required to form the matrix $Z^T(H+\Omega)Z$. Furthermore, the matrix Z for network contraints can be obtained without a factorization and we retain the sparsity of the network constraints. In the modified CG-Lanczos algorithm, the matrix-vector multiplication of $Z\times v$, $(H+\Omega)\times v$ and $Z^T\times v$ are the most computationally intensive parts for each CG-Lanczos iteration. Therefore, it is predictable that the sparsity of the null-space basis matrix Z will play a very important role. It is extremely important to reduced the computation for the matrix-vector multiplication of $Z\times v$ and $Z^T\times v$. In section 2.2.3, we describe a *spanning tree variable reduction basis* and an efficient algorithm ($2n$ additions/subtractions) to compute the matrix-vector multiplication of $Z\times v$ or $Z^T\times v$.

At the same time, it is essential to reduce the number of CG iterations. It is well known and discussed [69] that either (2.12) or (2.13) will become ill-conditioned when the solution approaches the optimal solution. In section 2.2.4, we present a preconditioned CG method with a maximal spanning tree diagonal preconditioner and the mathematically equivalent affine scaling methods. It makes the preconditioned matrix less ill-conditioned and improves efficiency by reducing the number of CG iterations. Since the maximal spanning tree is computationally expensive and is not essential, an efficient approximate maximal spanning updating method is derived, as described in section 2.2.6.

2.2.1 Reduced Newton direction

Since $span(A^T) \bigcup span(Z) = \mathcal{R}^n$, the search direction Δx can be formed by

$$\Delta x = A^T d_A + Z d_Z \ . \tag{2.15}$$

Substituting for Δx from (2.15) into (2.12), we get

$$(H+\Omega)(A^T d_A + Z d_Z) + A^T \Delta\lambda_A = -(g^B_{\bar{x}} + A^T \bar{\lambda}_A) \ , \tag{2.16}$$

$$AA^T d_A = -(A\bar{x} - b) \ . \tag{2.17}$$

By the assumption that A has full row rank, AA^T is a nonsingular matrix and (2.17) defines d_A uniquely. Substituting it into (2.16), we get

$$(H+\Omega)Zd_Z + A^T \Delta\lambda_A = -(g^B_{\bar{x}} + A^T \bar{\lambda}_A + (H+\Omega)A^T d_A) \ . \tag{2.18}$$

Multiplying both sides of (2.18) by Z^T, we obtain

$$Z^T(H+\Omega)Zd_Z = -Z^T(g^B_{\bar{x}} + (H+\Omega)A^T d_A) \ , \tag{2.19}$$

which is usually called the *Reduced Newton system.* To obtain the search direction Δx, it seems that we need to solve two linear systems (2.17) and (2.19). However, we can update d_A at each iteration without solving any linear system. Assume we have d^k_A satisfying (2.17). Therefore, we know

$$A(x^k + \Delta x^k) = Ax^k - (Ax^k - b) = b \tag{2.20}$$

and

$$x^{k+1} = x^k + \alpha^k \Delta x^k \ . \tag{2.21}$$

(2.20) and (2.21) imply

$$\begin{aligned} AA^T d_A^{k+1} &= b - Ax^{k+1} \\ &= A(x^k + \Delta x^k) - A(x^k + \alpha^k \Delta x^k) \\ &= (1-\alpha^k) A \Delta x^k \\ &= (1-\alpha^k) AA^T d_A^k \ . \end{aligned} \tag{2.22}$$

Under the assumption that A has full row rank, we can conclude

$$d_A^{k+1} = (1-\alpha^k) d_A^k \ . \tag{2.23}$$

2.2.2 Directions of negative curvature and the modified CG-Lanczos method

If the reduced Hessian $Z^T \bar{H} Z$ is indefinite, there exists a direction d with the property that

$$d^T Z^T \bar{H} Z d < 0 \ . \tag{2.24}$$

Any direction satisfying (2.24) is called a direction of negative curvature at $\bar{x}$.

As demonstrated in [8], any method for linearly constrained minimization that can be shown to converge to a point satisfying the second-order necessary conditions must explicitly or implicitly compute a direction of negative curvature of an indefinite reduced Hessian. The reason to compute a direction of negative curvature d is to move away from a saddle point, or avoid becoming close to such point. Using a direction of negative curvature in an algorithm when it exists not only helps to avoid the saddle point but also aids faster convergence to a second-order KKT point.

Unlike the direction of descent, we can define the best direction of negative curvature as a direction minimizing the Rayleigh quotient:

$$\frac{d^T Z^T \bar{H} Z d}{d^T d}. \tag{2.25}$$

Computing a direction of negative curvature is very similar to computing an eigenvector corresponding to the most negative eigenvalue.

The dominating computation for a Newton-type algorithm is that one has to solve a (reduced Hessian) linear system (2.12) and/or to solve an eigenvalue problem to get the search direction at each iteration. To solve the linear system (2.12), one can apply either a direct method or an iterative method. However, it will lose the sparsity of the problem if we apply a direct method. In this section, we describe an efficient algorithm to solve for the search direction approximately by an iterative method.

In the present context, the algorithm we are interested in first computes a truncated reduced Newton direction d_Z by solving (2.19) approximately. A good direction of negative curvature is also computed if $Z^T(\bar{H}+\Omega)Z$ is determined to be indefinite. We use a modified CG-Lanczos algorithm proposed in [8]. The first stage of the algorithm modifies the standard Conjugate Gradient (CG) algorithm such that we can obtain either a satisfactory descent direction or a good initial solution for finding a direction of negative curvature in the second stage. In the case that we want to solve (2.19) by CG but $Z^T(\bar{H}+\Omega)Z$ is determined to be indefinite, the CG method might fail. If all the CG iterates lie in a subspace where the quadratic is convex, CG may find a satisfactory descent direction and we will not determine whether the matrix $Z^T(\bar{H}+\Omega)Z$ is indefinite or not. On the other hand, the CG process breaks down if the CG search direction is a direction of negative curvature. In such a case, we have found a direction of negative curvature, though it may be is poor. However, it is a good initial solution for finding a good direction of negative curvature because it is a direction of negative curvature already. If the CG process breaks down before finding a satisfactory descent direction, we switch to the Lanczos algorithm, to find a good direction of negative curvature. As in the experiment of [8], we can expect that only a few iterations will provide a efficiently good direction of negative curvature.

2.2.3 Spanning tree variable reduction basis

For a node-arc incidence matrix, Dantzig [19] has shown that the arcs in any basis form a spanning tree of the network. Therefore, following [60], [14] and [78], there exists permutation matrices $P_{row} \in \mathcal{R}^{m\times m}$ and $P_{col} \in \mathcal{R}^{n\times n}$ such that the matrix $P_{row}AP_{col}$ can be partitioned as

$$P_{row}AP_{col} = [\ B\ N\ ]\ , \tag{2.26}$$

where the $m \times m$ submatrix B is nonsingular and lower triangular matrix. Using a variable reduction matrix Z as in Murtagh and Saunders [78], we can get a matrix with columns that span the null-space of A by setting

$$Z = P_{col} \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} . \tag{2.27}$$

Let index arrays “ib”, “in1”, “in2” and the sign vector “sb” be as follows:

ib(k): the index of the subdiagonal nonzero element in k-th column of B.

sb(k): the sign of the diagonal element in k-th column of B.

in1(k): the index of element 1 in k-th column of S.

in2(k): the index of element -1 in k-th column of S.

In the CG procedure, we have to compute Z times a vector and Z^T times a vector. With the variable reduction basis matrix (2.27), the procedure and the complexity analysis (number of “$\times$” , number of “$+$”) for computing such product is shown in Table 2.2 and Table 2.3. Both procedures take 0 multiplications and $2n$ additions/subtractions.

2.2.4 Preconditioning the Conjugate Gradient method

As mentioned in [47], the method of conjugate gradients works well on matrices that are either well conditioned or have a few distinct eigenvalues. In our case, it is predictable that the reduced Hessian matrix $Z^T(H + \Omega)Z$ will become very ill-conditioned because of some diagonal elements of Ω that are extremely large (when the solution approaches the boundary, some of the elements of s are closed to zero)

Table 2.2: Algorithm for computing multiplication $Z \times v$

Input: v, $in1$, $in2$, ib, sb, d
Output: Zv

$Nv = zeros(m, 1)$;
for $h = 1 : n$ (0,2(n-m))
    $Nv(in1(h)) = Nv(in1(h)) + v(h)$;
    $Nv(in2(h)) = Nv(in2(h)) - v(h)$;
end
for $h = 1 : m$(0,2m)
    $Nv(ib(h)) = Nv(ib(h)) + Nv(h)$;
    if $(sb(h) == -1)$
        $Nv(h) = -Nv(h)$;
    end if end
$Zv(m + 1 : n) = v$;
$Zv = P_{col} * Zv$

Table 2.3: Algorithm for computing multiplication $Z^T \times v$

Input: v, $in1$, $in2$, ib, sb, d
Output: ZTv

$Pv = P_{col}^T * v$
$Dv(m) = sb(m) * Pv(m)$
for $h = m - 1 : 1$ (0,2m)
    $Dv(h) = (Dv(ib(h)) + Pv(h));$
    if $(sb(h) == -1)$
        $Dv(h) = -Dv(h);$
end
for $h = 1 : n - m$ (0,m)
    $Nv(h) = Dv(in1(h)) - Dv(in2(h));$
end
$ZTv = Nv + Pv;$ (0,n-m)

and the others are of reasonable size. The number of CG iterations may be extremely large even for an approximate solution.

For the *Preconditioned Conjugate Gradient method*, in place of (2.19) we solve the modified linear system

$$C^{-1}Z^T(H+\Omega)ZC^{-T}\tilde{d}_Z = -C^{-1}Z^T(g_{\bar{x}}^B + (H+\Omega)A^T d_A)\ , \tag{2.28}$$

where C is nonsingular and CC^T is symmetric positive definite. To be practical, the computation of $C^{-1}v$ must be "simple". The aim is to make the preconditioned matrix

$$C^{-1}Z^T(H+\Omega)ZC^{-T} \tag{2.29}$$

less ill-conditioned than $Z^T(H+\Omega)Z$, with more clustered eigenvalues. The choice of preconditioner can have a dramatic effect upon the number of CG iterations required. A spanning tree preconditioner will be used and discussed in the following section. Preconditioning for linear systems is well understood, especially the positive definite case. However, it is not easy to precondition eigenvalue problems; see [64] for a detailed discussion.

The modified PCG-Lanczos algorithm for obtaining a search direction in x-space is presented in Table 2.4. Within this algorithm, the matrix-vector multiplications Zv and Z^Tv are given by Table 2.2 and 2.3.

As described early in this chapter, a Preconditioned CG method with a variable tolerance is suggested to solve for a truncated Newton's direction. In our modified PCG-Lanczos algorithm, the first criterion monitors the residual of the system of normal equations. Let r_i be the residual at the i-th CG iteration and μ_k be the barrier parameter at the k-th interior-point iteration. If

$$\|r_i\| \le \tau_1\sqrt{\mu_k}\|r_0\|\ , \tag{2.30}$$

the first stopping criterion is triggered. Criterion (2.30) becomes tighter when μ becomes smaller. It means we will obtain more accurate primal-dual directions as the iterates get closer to the optimal solution.

Table 2.4: A modified PCG-Lanczos algorithm

Input: Z, $\bar{H}$, ξ
Output: Zd_Z is either a truncated reduced Newton direction or a direction of negative curvature

$k = 0;\quad d_Z = 0;\quad r = -Z^T * \xi$
repeat
  Solve $CC^T v = r$
  $k = k + 1;$
  if $k == 1$
    $p = v$; and $rv1 = r' * v$;
  else
    $b = rv1/rv0;$
    $p = r + b * p;$
  end
  $Ap = Z^T * (\bar{H} * (Z * p));$
  $pAp = p' * Ap;$
  if $pAp < 0$
    $d_Z = p;$
    $\beta = \| d_Z \|;$
    $q = 0;$
    repeat
      $q = d_Z/\beta;$
      $x = Z^T * (\bar{H} * (Z * q));$
      $\alpha = q^T * d_Z;$
      $d_Z = d_Z + \alpha * q;$
      $\beta = \| d_Z \|;$
    until *stopping criterion is satisfied*
    return;
  end
  $a = rv1/pAp;$
  $d_Z = d_Z + a * p;$
  $r = r - a * Ap;$
  $rv0 = rv1$; and $rv1 = r' * v$;
until *stopping criterion is satisfied*

As discussed in Theorem 1, we required the search direction to be a "sufficient" descent direction to ensure convergence. When the current iterate is feasible, the search direction is $\Delta x^k = Zd_Z$, where d_Z is obtained from the modified PCG method. However, the normal stopping criterion for PCG (2.30) can not guarantee a sufficient descent direction in finite-precision computation. To ensure the sufficient descent property, we impose the relative descent stopping criterion

$$\frac{(Zd_Z^{(i)})^T g_B^k}{\|Zd_Z^{(i)}\| \cdot \|g_B^k\|} \leq -\tau_2 \ , \tag{2.31}$$

as the second stopping criterion when the current iterate is feasible. Geometrically, the relative descent $\dfrac{(Zd_Z^{(i)})^T g_B^k}{\|Zd_Z^{(i)}\| \cdot \|g_B^k\|}$ is the cosine of the angle between the gradient g_B^k and the potential search direction $Zd_Z^{(i)}$. By bounding the cosine away from zero we are insisting that Δx^k is not arbitrarily close to being orthogonal to g_B^k. Also, as a safeguard, a hard limit of $\tau_3(n-m)$ CG iterations is imposed.

2.2.5 Affine scaling

Applying the affine scaling matrix D, which is a diagonal matrix, to the original x-space, will transfer to the other vector space call x'-space. In other words, we let

$$x' = D^{-1}x \tag{2.32}$$

and solve for $\Delta x'$ rather than Δx. Substituting (2.32) into (2.7) gives the new system

$$\begin{bmatrix} HD & A^T & -I & I \\ AD & 0 & 0 & 0 \\ \bar{\Lambda}_l D & 0 & \bar{S}_l & 0 \\ -\bar{\Lambda}_u D & 0 & 0 & \bar{S}_u \end{bmatrix} \begin{bmatrix} \Delta x' \\ \Delta\lambda_A \\ \Delta\lambda_l \\ \Delta\lambda_u \end{bmatrix} = - \begin{bmatrix} g_{\bar{x}} + A^T\bar{\lambda}_A - \bar{\lambda}_l + \bar{\lambda}_u \\ A\bar{x} - b \\ \bar{S}_l\bar{\lambda}_l - \mu e \\ \bar{S}_u\bar{\lambda}_u - \mu e \end{bmatrix} . \tag{2.33}$$

The third and fourth equation of (2.33) implies

$$\Delta\lambda_l \ = \ -\bar{S}_l^{-1}\bar{\Lambda}_l D\Delta x' - (\bar{\lambda}_l - \mu\bar{S}_l^{-1}e) \tag{2.34}$$

$$\Delta\lambda_u = \bar{S}_u^{-1}\bar{\Lambda}_l D\Delta x' - (\bar{\lambda}_u - \mu\bar{S}_u^{-1}e) \ . \tag{2.35}$$

Replacing $\Delta\lambda_l$ and $\Delta\lambda_u$ in the first equation of (2.33) by (2.34) and (2.35), we can get the search direction $\Delta x'$ by solving

$$\begin{bmatrix} D(H+\Omega)D & DA^T \\ AD & 0 \end{bmatrix} \begin{bmatrix} \Delta x' \\ \Delta\lambda_A \end{bmatrix} = - \begin{bmatrix} D(g_{\bar{x}}^B + A^T\bar{\lambda}_A) \\ A\bar{x} - b \end{bmatrix} , \tag{2.36}$$

Let

$$Z_{AD} = P_{col} \begin{bmatrix} -D_B^{-1}B^{-1}ND_N \\ I \end{bmatrix} \tag{2.37}$$

be the null-space basis matrix for matrix AD. We can follow the same reasoning given in section 2.2.1 to get

$$\Delta x' = Z_{AD}d'_Z + DA^Td'_A \ . \tag{2.38}$$

We can obtain d'_Z from

$$Z_{AD}^T D(H+\Omega)DZ_{AD}d'_Z = -Z_{AD}^T(Dg_{\bar{x}}^B + D(H+\Omega)D^2A^Td'_A) \ . \tag{2.39}$$

Furthermore, $AA^Td_A = -(A\bar{x} - b)$ and $AD^2A^Td'_A = -(A\bar{x} - b)$ implies that we can substitute $D^2A^Td'_A = A^Td_A$ and update it by (2.23).

If we use the spanning tree variable reduction basis (2.27) as the null-space basis, the matrix appearing in the left-hand side of the linear system of equations (2.39) can be expanded as

$$\begin{aligned} & Z_{AD}^T D(H+\Omega)DZ_{AD} \\ = \ & \left[-D_NN^TB^{-T}D_B^{-1} \ I\right] P_{col}^T D(H+\Omega)DP_{col} \begin{bmatrix} -D_B^{-1}B^{-1}ND_N \\ I \end{bmatrix} \\ = \ & \left[-D_NN^TB^{-T}D_B^{-1} \ I\right] \\ & \begin{bmatrix} D_B(H_{BB}+\Omega_B)D_B & D_B(H_{BN}D_N \\ D_NH_{NB}D_B & D_N(H_{NN}+\Omega_N)D_N \end{bmatrix} \begin{bmatrix} -D_B^{-1}B^{-1}ND_N \\ I \end{bmatrix} \end{aligned}$$

$$\begin{aligned} &= D_N N^T B^{-T}(H_{BB}+\Omega_B)B^{-1}ND_N - D_N N^T B^{-T} H_{BN} D_N \\ &\quad D_N H_{NB} B^{-1} N D_N + D_N (H_{NN}+\Omega_N) D_N \ . \end{aligned} \tag{2.40}$$

For the preconditioned matrix shown in (2.29), we can expand it as follows:

$$\begin{aligned} & C^{-1}Z^T D(H+\Omega)DZC^{-T} \\ =\ & C^{-1}\left[-N^T B^{-T}\ I\right] P_{col}^T D(H+\Omega)DP_{col} \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} C^{-T} \\ =\ & C^{-1}N^T B^{-T}(H_{BB}+\Omega_B)B^{-1}NC^{-T} - C^{-1}N^T B^{-T} H_{BN} C^{-T} \\ & C^{-1}H_{NB}B^{-1}NC^{-T} + C^{-1}(H_{NN}+\Omega_N)C^{-T} \ . \end{aligned} \tag{2.41}$$

Comparing (2.40) and (2.41), we see that the affine scaling scheme is equivalent to the PCG method with the preconditioner $CC^T = D_N^{-2}$.

2.2.6 Maximal spanning tree basis

In linear network programming, the spanning tree preconditioner for the range space normal equation was introduced in [89] and used in several codes, such as [54] and [86]. In that research, a "maximal spanning tree" of the graph $\mathcal{G}$ is identified, using as the weights

$$w = \Omega^k e \ . \tag{2.42}$$

This approach is theoretically equivalent to forming the scaled variable reduction matrix Z_{AD} and choosing the basis B as the maximal spanning tree for weights w. In our algorithm, we apply a different scaling,

$$D = (\Omega e + (diag(H))^+ e)^{-\frac{1}{2}} \ , \tag{2.43}$$

to the original x-space and compute the maximal spanning tree based on the weights

$$\bar{w} = D^{-1} e \ . \tag{2.44}$$

Kruskal's algorithm [67], implemented with the data structures in [97], is used to compute the maximal spanning tree. The complexity of the algorithm for finding a maximal spanning tree is $O(n^2)$. However, we do not need the exact maximal spanning tree. Our goal is to keep indices with large components of $\bar{w}$ on the spanning tree. Therefore, an approximated maximal spanning tree is acceptable. Chin [16] and Spira and Pan [96] present an $O(n)$ algorithm for updating the maximal spanning tree if a new arc is inserted into the graph, and an $O(n^2)$ algorithm if an arc is deleted.

In the primal-dual algorithm, the change in the maximum spanning tree is usually very small from one iteration to the next iteration. Therefore, we conduct a spanning tree updating procedure as shown in Table 2.5. Basically, we first check if there is any arc (variable) whose weight is too small to be in the spanning tree (basis) but is currently in it or there is any arc we should definitely include in the spanning tree. Suppose that $w_{sort}(m)$ is the m-th largest element of the weight w. Then the threshold weight for adding a non-basic arc to the spanning tree is

$$thred_{add} = \zeta_1 * w_{sort}(m) \ , \tag{2.45}$$

with $\zeta_1 > 1$, and the threshold weight for deleting a basic arc from the spanning tree is

$$thred_{del} = \zeta_2 * w_{sort}(m) \ , \tag{2.46}$$

with $0 < \zeta_2 < 1$. We insert the arcs we want to include one by one and then delete the arcs we do not want in the basis if they are still in the basis. The reason that we first update the spanning tree by inserting an arc is that it costs less computation and it often removes the arc we want to delete in the process. According to our numerical results, only 15 percent of the updates are deleting an arc.

When an arc is inserted into a spanning tree a cycle is created. We can get a new tree by removing any arc within the cycle. To update the maximal spanning tree, we can remove the arc with minimum weight of the arcs in the cycle. The algorithm for updating the spanning tree by inserting an arc is described in Table 2.6.

When we delete an arc from a spanning tree, a subtree disconnected from the root is created. We can get a new spanning tree by adding an arc with one end

Table 2.5: A spanning tree updating algorithm

Input: $\mathcal{I}_B$, $\mathcal{I}_N$, w
Output: new $\mathcal{I}_B$, $\mathcal{I}_N$

Given $1 \leq \zeta_1$ and $0 < \zeta_2 \leq 1$
$\mathcal{I}_{del} = \emptyset$ and $\mathcal{I}_{add} = \emptyset$
$w_{sort} = sort(w)$
$thred_{add} = \zeta_1 * w_{sort}(m)$ and $thred_{del} = \zeta_2 * w_{sort}(m)$
for $(i \in \mathcal{I}_B)$
    if $(w(i) < thread_{del})$
        $i \in \mathcal{I}_{del}$
    end if
end
for $(i \in \mathcal{I}_N)$
    if $(w(i) > thread_{add})$
        $i \in \mathcal{I}_{add}$
    end if
end
for $(i \in \mathcal{I}_{add})$
    Update the Spanning by Inserting Arc i
end
for $(i \in \mathcal{I}_{del})$
    if $(i \in \mathcal{I}_B)$
        Update the Spanning by Deleting Arc i
    end if
end

Table 2.6: Algorithm for updating the spanning tree by inserting an arc

Input: $\mathcal{I}_B$, $\mathcal{I}_N$, w, i_{add}
Output: new $\mathcal{I}_B$, $\mathcal{I}_N$

Find the index set $\mathcal{I}_{cycle} \subset \mathcal{I}_B \bigcup i_{add}$
$i_{remove} = \arg\min \{w_i,\ i \in \mathcal{I}_{cycle}\}$
$\mathcal{I}_N = \mathcal{I}_N \bigcup \{i_{remove}\} \setminus \{i_{add}\}$
$\mathcal{I}_B = \mathcal{I}_B \bigcup \{i_{add}\} \setminus \{i_{remove}\}$

node in the rooted tree and another in the subtree. To update the maximal spanning tree, we can insert the arc with maximum weight over the arcs with one end node in the rooted tree and another in the subtree. However, we do not need the exact maximal spanning tree. We need to put the arc with large weight into the spanning tree and remove the arc with a tiny weight out of the spanning tree. Therefore, we propose that an appropriate arc with weight greater than $\zeta_3 w(i_{del})$, where $\zeta_3 > 1$ and $w(i_{del})$ is the weight of the arc that we would like to delete, is acceptable to insert into the spanning tree rather than searching for the entire arcs set. The algorithm for updating the spanning tree by deleting an arc is described in Table 2.7.

Table 2.7: Algorithm for updating the spanning tree by deleting an arc

Input: $\mathcal{I}_B$, $\mathcal{I}_N$, w, i_{del}
Output: new $\mathcal{I}_B$, $\mathcal{I}_N$

```
Find the node set V_root ⊂ V and V_sub = V\V_root
i_ins = i_del
thred = ζ_3 w_{i_del}
for (i ∈ E)
    if ( (fromnode(i) ∈ V_root and tonode(i) ∈ V_sub) or
          (tonode(i) ∈ V_root and fromnode(i) ∈ V_sub) )
        if w_i > thred
            i_ins = i
            break
        end if
    end if
end
I_N = I_N ∪ {i_del} \ {i_ins}
I_B = I_B ∪ {i_ins} \ {i_del}
```

Chapter 3

Linesearch Procedures

In a linesearch algorithm, a new point w_{k+1} is defined by taking a positive step α_k from the current point w_k along a search direction p_k. In other words,

$$w_{k+1} = w_k + \alpha_k p_k \ . \tag{3.1}$$

It is assumed that the objective function $\Psi(w)$ is a differentiable function and p_k is a search direction with the property that p_k is a "sufficient" descent direction for the objective function. Given the current point w_k and the search direction p_k, we denote the univariate function $\psi(\alpha)$ as

$$\psi(\alpha) = \Psi(w_k + \alpha p_k) \ . \tag{3.2}$$

It implies that $\psi(\alpha)$ represents the variation in Ψ as a function of the steplength α from w_k along p_k and has the properties

$$\psi(0) = \Psi(w_k) \ , \tag{3.3}$$

$$\psi'(0) = p_k^T \nabla \Psi(w_k) \ , \tag{3.4}$$

$$\psi'(\alpha) = p_k^T \nabla \Psi(w_k + \alpha p_k) \ . \tag{3.5}$$

Since p_k is a descent direction of Ψ, it follows that $\psi'(0) < 0$.

The process of finding a suitable steplength α_k is called the *linesearch procedure*. In general, linesearch procedures generate a closed interval $[a, b]$, called the *interval of uncertainty*, such that an acceptable steplength α_k lies in $[a, b]$. Then by generating and testing a new trial steplength in the current enclosing interval, either an acceptable steplength is obtained or the interval of uncertainty reduced.

In the linesearch procedure, α_k is required to satisfy some conditions, as discussed in section 3.1. Depending on the choice of parameter used in the conditions, the linesearch procedure can perform an accurate linesearch or an inexact linesearch. In the early years of linesearch algorithms for unconstrained optimization, most implementations used an accurate linesearch. Recently, the topic of an inexact linesearch is widely used and studied by many authors such as [3, 99, 88, 40, 24, 71, 31]. However, an accurate linesearch remains desirable in some situations, such as in large-scale problems where the numerical computation of a search direction p_k is substantially more time-consuming than the evaluation of $\Psi(\cdot)$ at trial steps. For additional discussion and comparison, see [24, 31, 43].

For generating the trial steplength, one of the popular techniques is the Armijo or pure backtracking linesearch [82, 24], in which each trial step $\alpha^{(j)}$, $j = 1, \ldots$ is determined as a fixed fraction ϖ (typically $\varpi = \frac{1}{2}$ or $\frac{1}{10}$) of the previous step $\alpha^{(j-1)}$. Under reasonably mild conditions, this procedure with modification is guaranteed to produce a sufficient decrease (see [13]). The advantage of an Armijo linesearch is its simplicity. In the situation where the initial step is almost always accepted, such as the case when applying Newton's method to minimize a well-behaved function, an Armijo linesearch is also efficient. However, a pure backtracking procedure is not appropriate when an accurate linesearch is needed, since the fixed and strictly decreasing sequence of trial steps $\{\varpi^j \alpha^{(0)}\}$ is unaffected by the behavior of $\psi(\cdot)$.

A second standard linesearch technique is to define the trial steps as iterates from a procedure for finding an approximate minimizer of a univariate function $\psi(\cdot)$ within a specified interval of uncertainty. The interval of uncertainty is modified to reflect information about $\psi(\cdot)$ accumulated as the linesearch proceeds. The details about updating the interval of uncertainty are described in section 3.2.1. Here, it should be stressed that the univariate minimization is used only as a model subproblem for generating trial steps in the linesearch; the sequence of trial steps terminates as soon as the conditions on the step length are satisfied.

For smooth objective functions, low-order polynomial (linear, quadratic or cubic) interpolation is widely used to conduct the trial step based on previous trial steps and the associated value of $\psi(\cdot)$ (see [37, 43, 24, 31, 25]). This approach is sometime combined with backtracking (see [25]), and can be used when an accurate linesearch is needed. To ensure robustness, a safeguarded algorithm is used to guarantee a certain reduction in the interval of uncertainty. In a safeguarded algorithm, the interpolant step α_{int} can be rejected under some circumstances, for example, if α_{int} fails to lie within the current interval of uncertainty. This issue and related topics are discussed in [37, 31, 25]. In section 3.2.2, an efficient safeguarded steplength algorithm is discussed and described.

In the linesearch procedure for an objective with a logarithmic barrier function, low-order polynomial interpolation usually cannot reflect the singularity of the barrier function. This problem tends to cause inefficiency for the barrier method. Therefore, several special-purpose linesearch strategies have been proposed. In [32], separate approximations were made to the objective function and the barrier term. In the approach of [31], a special function is developed by substituting linearized versions of the objective function and constraints into the form of the barrier term. The techniques of both [32] and [31] apply to a broad class of barrier functions. A special approximating function designed specially for the logarithmic barrier function was proposed in [77]. In section 3.3, we propose a double iterative linesearch procedure, in which each trial step length is produced by a special iterative procedure to find an approximate local minimizer of a lower-order polynomial interpolant plus the barrier term.

3.1 Sufficient decrease conditions

A fundamental requirement for a linesearch algorithm associated with a descent method involves the reduction in $\Psi(w)$ at each iteration. If convergence is to be assured, the steplength α_k has to satisfy conditions of sufficient reduction in $\Psi(w)$.

Sufficient decrease conditions not only require a reduction in Ψ but have restrictions on the character of the decrease. The sufficient decrease requirement can be achieved by several alternative sets of restrictions on α_k. In this section, two of the most popular rules for sufficient decrease, the Goldstein rule and the Gamma rule, are described. For other sufficient decrease conditions, see [82, 43, 24, 31, 25].

3.1.1 Goldstein rule

In [46], Goldstein and Price proposed a sufficient decrease condition that has been widely used and discussed. This condition ensures that α_k is neither "too large" nor "too small". A suitable step, α_k, is required to satisfy

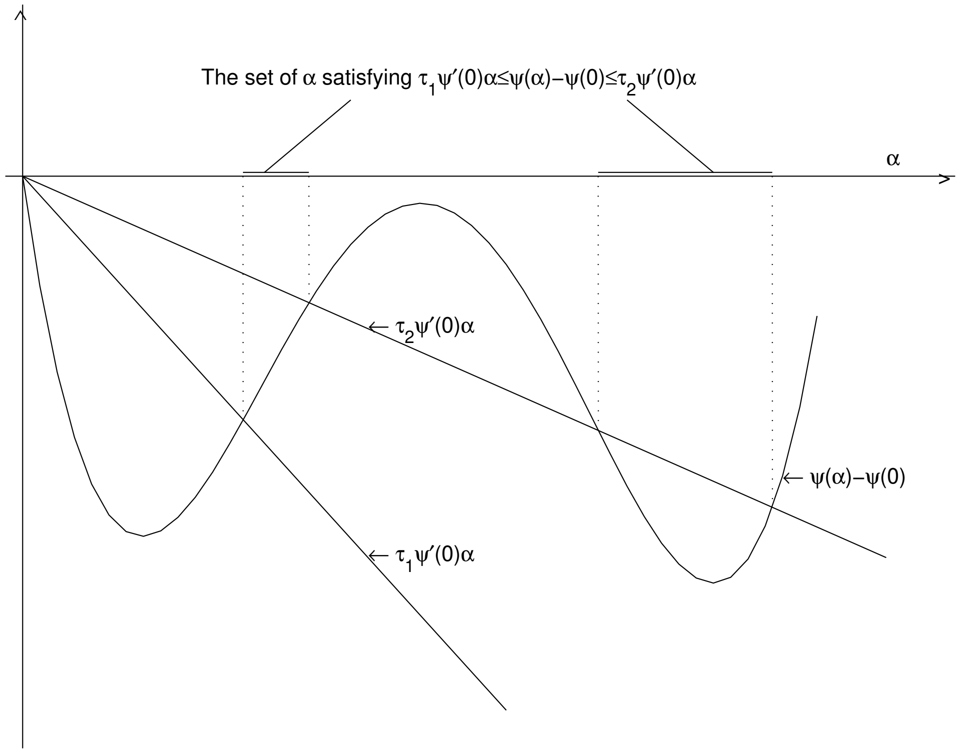
$$0 < -\tau_1 \alpha_k \psi'(0) \leq \psi(0) - \psi(\alpha_k) \leq -\tau_2 \alpha_k \psi'(0) \ , \tag{3.6}$$

where τ_1 and τ_2 are scalars satisfying $0 < \tau_1 < \tau_2 < 1$. This is now known as the *Goldstein Rule*. The upper bound in (3.6) prevents α_k from being too large, and the lower bound prevents α_k from being too small. Figure 3.1 illustrates the rule.

The Goldstein rule does not in itself constitute an algorithm because how to find a point satisfying it is not immediately suggested by the rule. One possibility is to adopt a *backtracking linesearch algorithm*, that is, the trial values of steplength are defined in terms of an initial step $\alpha^{(0)}$ and a positive scalar ϖ with $\varpi < 1$. The value of α_k is taken as the first member of sequence $\{\varpi^j \alpha^{(0)}\}_{j=0,\ldots}$ for which (3.6) is satisfied. The performance of this type of algorithm is highly dependent on the choice of $\alpha^{(0)}$, rather than on any particular merits of condition (3.6). Typically $\alpha^{(0)} = 1$ in order to achieve superlinear convergence with Newton-based and quasi-Newton methods rather than to maximize the probability of satisfying (3.6).

As emphasized in [6], the Goldstein rule (3.6) alone does not guarantee a "good" value of α_k. Note that for almost all functions encountered in practice, choosing $\alpha^{(0)}$ as 10^{-5} would satisfy (3.6) for appropriate small values of τ_1 and τ_2. This strategy would be "efficient" finding a suitable α_k, but, any descent method that included such a steplength algorithm would be extremely inefficient. It is essential to consider the performance of a steplength algorithm not merely in terms of the number of function

Figure 3.1: Illustration for Goldstein rule



evaluations per iteration, but in terms of the overall reduction in $\Psi(\cdot)$ achieved at each iteration.

3.1.2 Gamma rule

An alternative class of conditions on α_k can be derived from interpreting the steplength in terms of univariate minimization:

$$\underset{\alpha}{\text{minimize}} \;\; \psi(\alpha) \;. \tag{3.7}$$

A step to the first minimizer of $\psi(\cdot)$ yields a "significant" reduction in $\psi(\cdot)$, and this choice of α_k is important for many theoretical convergence results. The following widely used criterion for steplength acceptance is based on interpreting α_k in terms of univariate minimization:

$$| \psi'(\alpha_k) | \leq -\gamma_1 \psi'(0) \ , \tag{3.8}$$

where $0 \leq \gamma_1 < 1$. The value of γ_1 in (3.8) determines the accuracy with which α_k approximates a stationary point of $\psi(\cdot)$, and consequently provides a means of controlling the balance of effort to be expended in computing α_k. When γ_1 is small, a linesearch procedure based on (3.8) can be described as an "accurate linesearch"; when $\gamma_1 = 0$, the procedure will terminate at a stationary point of $\psi(\alpha)$ so that it will be termed an "exact linesearch". Furthermore, it may be shown that the condition (3.8) prevents α_k from being too small. It will also usually prevent α_k from being too large.

Because (3.8) does not restrict the function value of $\psi(\cdot)$, an additional condition is necessary to ensure a sufficient decrease in the function value of $\psi(\cdot)$. In particular, many steplength algorithms include the following condition:

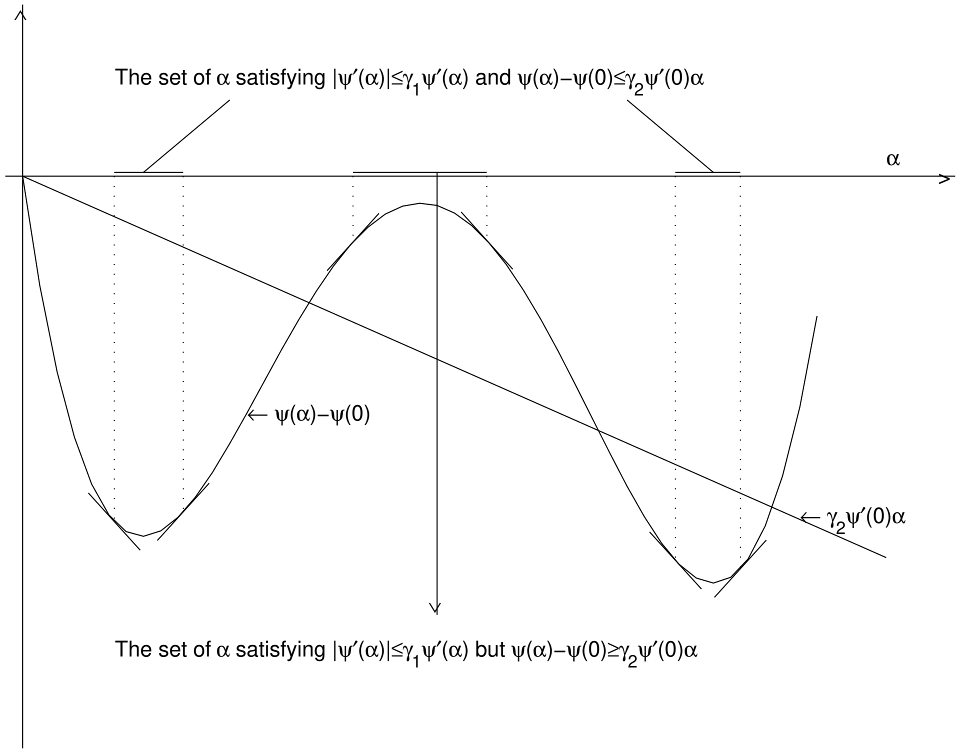
$$\psi(\alpha_k) \leq \psi(0) + \gamma_2 \alpha_k \psi'(0) \ , \tag{3.9}$$

where $0 < \gamma_1 < \frac{1}{2}$. Satisfaction of (3.9) implies that the point $(\alpha_k, \psi(\alpha_k))$ lies on or below the straight line $\ell(\alpha) = \psi(0) + (\gamma_2 \psi'(0))\alpha$. Usually, γ_2 is chosen to be small, which increases the probability that an approximate minimizer will satisfy the condition. Combining conditions (3.8) and (3.9) as termination criteria gives the so-called *Gamma Rule*.

The set of points satisfying conditions (3.8) and (3.9) will be denote by $\Gamma(\gamma_1, \gamma_2)$. If $0 < \gamma_2 \leq \gamma_1 < 1$, it can be shown [43] that $\Gamma(\gamma_1, \gamma_2)$ contains a nontrivial interval (i.e. there exists a point belonging to $\Gamma(\gamma_1, \gamma_2)$). Figure 3.2 illustrates the gamma rule.

The advantage of using (3.8) as an acceptance criterion is that its interpretation in terms of a local minimum suggests efficient methods for computing a good value of α_k, as discussed in section 3.2 and 3.3.

Figure 3.2: Illustration for Gamma rule



3.2 Linesearch iteration

A suitable value of α_k is obtained by an iterative procedure that generates a sequence of trial steplength $\{\alpha_k^{(j)}\}$ for $j = 0, 1, \ldots$ with termination condition that accepts the first $\alpha_k^{(j)}$ for which the sufficient decrease conditions hold. A certain amount of computation is required to calculate and to test for each trial steplength $\alpha_k^{(j)}$. For example, to check the sufficient decrease condition we have to evaluate $\Psi(w)$ and $\nabla\Psi(w)$ at $w_k + \alpha_k^{(j)} p_k$. The function evaluations of $\Psi(w)$ and $\nabla\Psi(w)$ are usually time-consuming.

In the early years of linesearch algorithms for unconstrained optimization, most implementations used an accurate linesearch, i.e., α_k was choose to be an approximation close to the minimizer of ψ. We can easily achieve this goal by setting the

Table 3.1: Algorithm for linesearch iterate

Input: w_k, p_k
Output: α_k is the step length which will be taken in k-th iteration

```
j = 0;
compute α_k^(0) (see Table 3.4)
if Ψ(α_k^(0)) − Ψ(0) > γ_3 α_k^(0) ψ'(0)
        initialize the interval of uncertainty (α_lo, α_hi)
        repeat
                j = j + 1;
                compute α_trial^(j) (see Table 3.5)
                α_k^(j) = safeguard(α_trial^(j), α_lo, α_hi) (see Table 3.3)
                evaluate Ψ(α_k^(j)) and Ψ'(α_k^(j))
                updating the interval of uncertainty (α_lo, α_hi) (see Table 3.2)
        until α_k^(j) satisfies the condition (3.8) and (3.9)
end if
return α_k^j;
```

parameters γ_1 and γ_2 in the Gamma rule to be small. A steplength satisfying the terminating criteria will be very close to the minimizer of ψ. Today, it is customary to perform only an inexact or relaxed linesearch algorithm with small γ_1 and large γ_2 (for example, $\gamma_1 = 10^{-4}$ and $\gamma_2 = 0.9$).

Typical relaxed sufficient decrease criteria are satisfied by a set of acceptable values for α_k. A good linesearch algorithm should produce a good step, usually a large reduction in Ψ relative to what can be achieved in a small number of trial steps.

As demonstrated in [87], the efficient linesearch iteration will be as in Table 3.1.

3.2.1 Interval of uncertainty and interval reducing procedure

As discussed in section 3.1.2, the Gamma rule is derived from interpreting the linesearch iterate in terms of univariate minimization. The techniques used to solve the problem of univariate minimization in a bounded interval are analogous to those for zero-finding. To develop an interval-reduction procedure, we need to define a condition that ensures that there is a proper minimum α^* in a given interval, say $[\alpha_{low}, \alpha_{high}]$, which is defined as the *interval of uncertainty* for this problem. The interval reducing procedure is based upon computing a sequence of intervals $\{I_j\}$, each containing a minimum of the objective function, such that the interval I_j lies wholly within I_{j-1}. Given an interior-point α_{trial} for $[\alpha_{low}, \alpha_{high}]$, the next interval of uncertainty will be the reduced interval $[\alpha_{low}, \alpha_{trial}]$ or $[\alpha_{trial}, \alpha_{high}]$, which can be determined by using the value and the first derivative of the objective function at α_{trial}.

For linesearch iterations using Gamma rule as a terminating criterion, the interval reducing procedure is modified as each interval of uncertainty $\{I_j\}$ must contain points of $\Gamma(\gamma_1, \gamma_2)$. The sequence of intervals can be computed using the method of safeguarded interpolation described in section 3.2.2 and 3.3. The linesearch iterate will end up with the first point generated by safeguarded interpolation that lies in $\Gamma(\gamma_1, \gamma_2)$.

To ensure that the interval of uncertainty $[\alpha_{low}, \alpha_{trial}]$ contains points of $\Gamma(\gamma_1, \gamma_2)$, $\alpha_{low}, \alpha_{trial}$ are required to satisfy the following conditions:

$$\begin{aligned} \psi'(\alpha_{low}) &\leq \gamma_1 \psi'(0) \qquad \text{and} \\ \psi(\alpha_{low}) &\leq \psi(0) + \gamma_2 \alpha_{low} \psi(0) \ , \end{aligned} \tag{3.10}$$

and

$$\begin{aligned} \psi'(\alpha_{high}) &\geq -\gamma_1 \psi'(0) \qquad \text{or} \\ \psi(\alpha_{high}) &\geq \psi(0) + \gamma_2 \alpha_{high} \psi'(0) \ . \end{aligned} \tag{3.11}$$

Based on conditions (3.10) and (3.11), we can use the algorithm described in Table 3.2 to update the interval of uncertainty.

Table 3.2: Algorithm for updating the interval of uncertainty

Input: α_{low}, α_{high}, α_{trial},
Output: $[\alpha_{low}, \alpha_{high}]$ is the reduced interval of uncertainty

```
if ψ(α_trial) ≥ ψ(0) + γ_2 α_trial ψ'(0)
        α_high = α_trial
else if ψ'(α_trial) > 0
        α_high = α_trial
else
        α_low = α_trial
end
return [α_low, α_high]
```

3.2.2 Safeguarded-steplength algorithm

The idea of a safeguarded steplength algorithm is to conduct the trial steplength by the combination of a reliable method (such as the method of bisection) and a rapidly-convergent method (such as polynomial or special interpolation, see section 3.3.1, 3.3.2) that has the following properties:

(i) If $\Psi(\cdot)$ is well behaved, the linesearch algorithm will converge rapidly.

(ii) In the worst case, it is not much less efficient than the guaranteed method.

At each iteration, the safeguarded method is given an interval of uncertainty, say $[a, b]$. The known values of $\psi(\alpha)$ and $\psi'(\alpha)$ are used to obtain an estimate of minimizer α_{trial} by interpolation of the linesearch function. Without safeguards, the next step would involve evaluating $\psi(\alpha_{trial})$ and $\psi'(\alpha_{trial})$. However, a safeguarded procedure ensures that α_{trial} is a "reasonable" steplength before evaluating $\psi(\alpha_{trial})$ and $\psi'(\alpha_{trial})$.

It is complicated to define what is "reasonable". However, there are some apparent required characteristics. First of all, α_{trial} must lie in the interval of uncertainty $[a, b]$. Secondly, it should not be too close to a or b even if α_{trial} is in $[a, b]$. The reason for this is that the sequence of trial steplengths might converge to one of the end points or the reduction of the interval of uncertainty may become extremely slow. The most common techique used to prevent these is to specify a fraction $\eta_1 \in (0, 0.5)$, often chosen small (say 0.05), and force α_{trial} at least $\eta_1(b-a)$ away from the two end points a and b. That is,

$$\begin{cases} \alpha_{trial} = a + \eta_1(b-a) & \text{if } \alpha_{trial} < a + \eta_1(b-a) \\ \alpha_{trial} = b - \eta_1(b-a) & \text{if } \alpha_{trial} > b - \eta_1(b-a) \\ \alpha_{trial} = \alpha_{trial} & \text{otherwise} \end{cases} . \tag{3.12}$$

Note that an absolute bound is not required since $b-a$ is bounded away from zero. Furthermore, an efficient safeguarded algorithm should also take into account the situations that cause inefficiency and often occur in the interpolating method of the linesearch procedure. We should consider the following four cases:

1. The minimum is close to the right-end point of the interval of uncertainty, but the trial step α_{trial} obtained from interpolating method is less than the lower bound and the interval of uncertainty keeps updating the left-end point. For example, the function in Figure 3.3.

2. The minimum is close to the left-end point of the interval of uncertainty, but the trial step α_{trial} obtained from interpolating method is great than the upper bound and the interval of uncertainty keeps updating the right-end point. (The opposite to the previous case)

After considering those cases, we conduct an efficient safeguarded steplength algorithm that prevents those slow convergence cases, as shown in Table 3.3.

3.3 Linesearch for Objective with Logarithmic Barrier Function

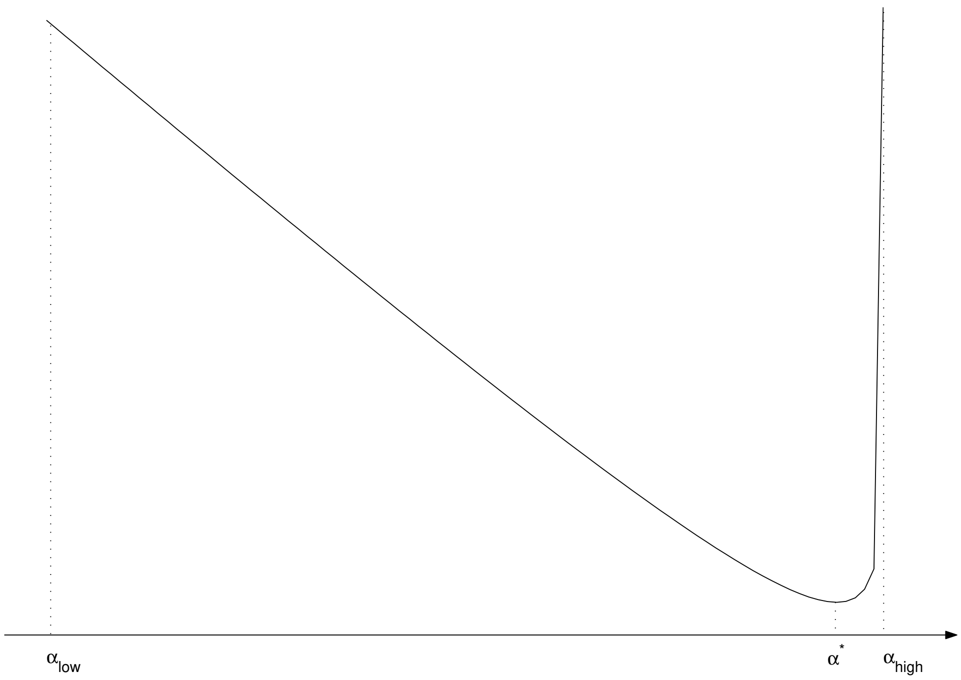
In the 1960s and 1970s, several special-purpose linesearch strategies were proposed for barrier functions. In [32], separate approximations are made to the objective

Table 3.3: Safeguarded steplength algorithm

Input: α_{low}, α_{high}, α_{trial}, I_{low}, I_{high}, J_{low} and J_{high}
Output: α_{trial} is the suitable trial steplength

```
LB = η₁α_high + (1 − η₁)α_low
UB = (1 − η₁)α_high + η₁α_low
if α_trial < LB
    α_trial = LB
    J_low = J_low + 1
    J_high = 0
    if J_low ≥ 3
        if I_high ≥ 3
            α_trial = η₂α_high + (1 − η₂)α_low
        else if I_low ≥ 3
            α_trial = UB
        end
    end
else if α_trial > UB
    α_trial = UB
    J_high = J_high + 1
    J_low = 0
    if J_high ≥ 3
        if I_low ≥ 3
            α_trial = η₂α_low + (1 − η₂)α_high
        else if I_high ≥ 3
            α_trial = LB
        end
    end
end
return α_trial
```

Figure 3.3: EXAMPLE illustrating inefficiency for standard interpolation.



function and the barrier term. The technique can be applied to different kinds of barrier functions. In [77], the approximating function are designed specially for the logarithmic barrier function.

Since Karmarkar [56] proposed his projective algorithm in 1984 and its relationship to the barrier method was shown in [39], the interest in barrier methods was renewed but largely for the linear programming problem. Such algorithms typically rely on a fixed step and do not in practice require a linesearch. Typically, the linesearch in the barrier-related interior-point algorithms for linear programming take α_k as a fixed fraction (say, 0.95 or 0.99) of the maximum allowable steplength, the step from x_k along p_k to the boundary of the feasible region. Basically, this kind of simplified strategy succeeds for the linear programming case because in many instances singularity at the boundary causes a minimizer to occur near the boundary in a function that would otherwise behave like a monotonically decreasing linear function. For more discussion and references for linesearches in interior-point methods for linear programming, please refer to [48].

However, as discussed in [77], it tends to be inadequate or inefficient to use such a simple linesearch approach for nonlinear optimization. For minimizing a nonlinear objective function with a logarithmic barrier function, we suggest modifications to two aspects of a traditional general linesearch:

(i) The choice of initial step $\alpha^{(0)}$, described in section 3.3.1.

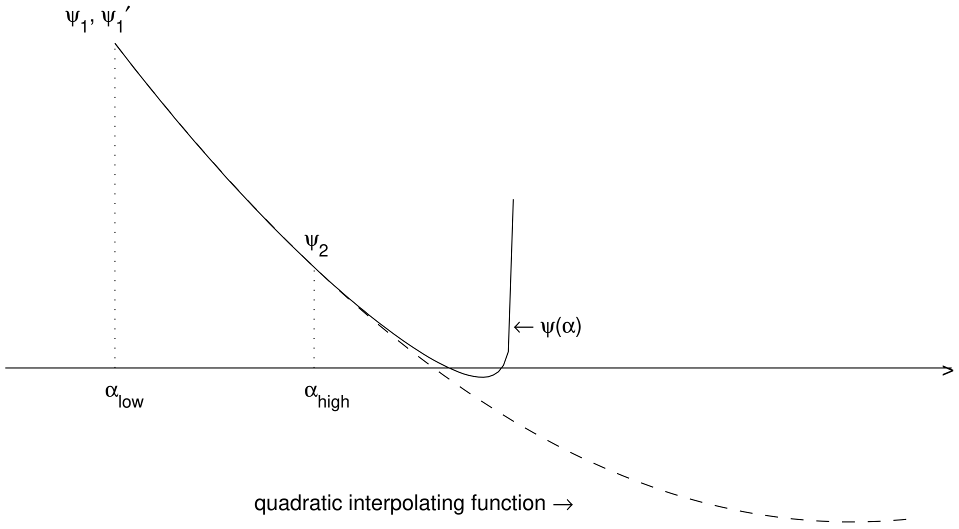
(ii) The procedure for generating subsequent trial steps $\alpha^{(j)}$, described in section 3.3.2.

In this section, the discussion involves a single linesearch and we denote $\bar{w} = (\bar{x}^T, \bar{s}^t)^T$ as the current iterate and $p = (\Delta x^T, \Delta s^T)^T$ as the descent search direction. Furthermore, we define the univariate linesearch function as follows:

$$\begin{aligned}
\phi(\alpha) &= F(\bar{x} + \alpha\Delta x) \ , \\
\varphi(\alpha) &= B(\bar{s} + \alpha\Delta s) \ , \\
\psi(\alpha) &= F_B(\bar{w} + \alpha p, \mu) = \phi(\alpha) - \mu\varphi(\alpha) \ .
\end{aligned}$$

The second issue arises because a low-order polynomial interpolant may not reflect the characteristic dramatic behavior of a barrier function in a neighborhood of a singularity. In Figure 3.4, a barrier function $\psi(\alpha)$ with barrier parameter $\mu = 1$ is shown as a solid line and the dashed line represents the quadratic function that interpolates $\psi(\alpha_{low})$, $\psi'(\alpha_{low})$ and $\psi(\alpha_{high})$. The minimizer of the quadratic function is a substantial and infeasible overestimate of the minimizer of the barrier function. In Figure 3.5, another barrier function $\psi(\alpha)$ with barrier parameter $\mu = 1$ is shown as a solid line and the dashed line represents the quadratic function that interpolates $\psi(\alpha_{low})$, $\psi'(\alpha_{low})$ and $\psi'(\alpha_{high})$. The minimizer of the quadratic function is a substantial underestimate of the minimizer of the barrier function.

Figure 3.4: Example 1 of quadratic interpolating for a barrier function



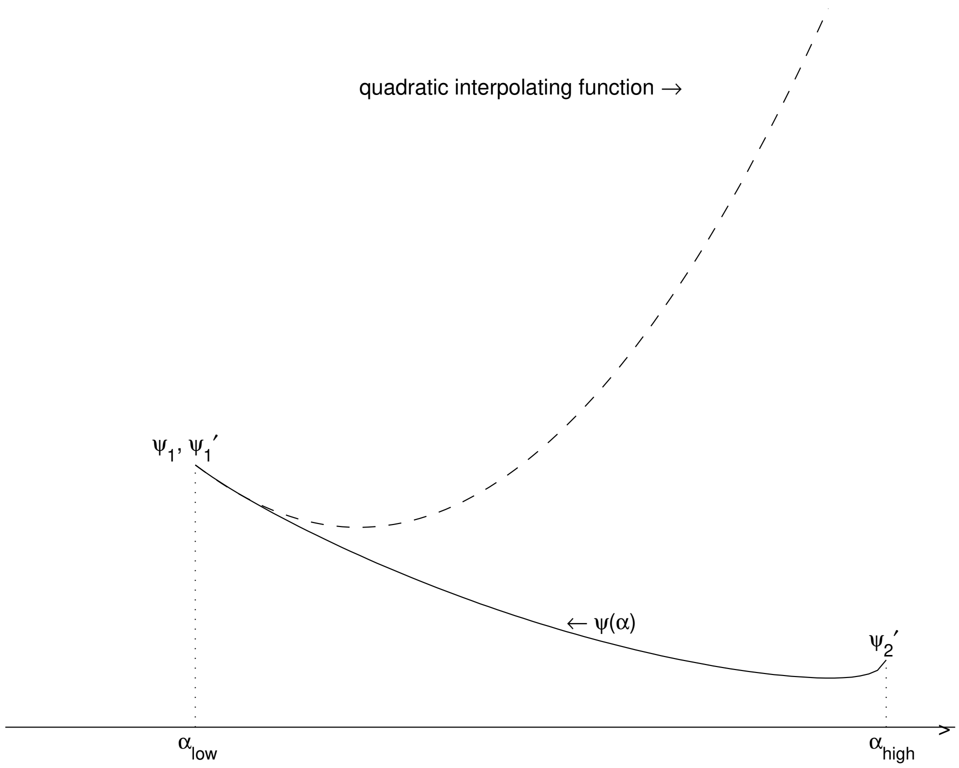
Both examples occur often in the linesearch for barrier function and illustrate the disparity of function interpolation using low-order polynomial interpolant. Instead, we use a low-order polynomial $\bar{\phi}$ to interpolate the function ϕ but not for the barrier function φ. The interpolating function becomes

$$\bar{\psi}(\alpha) = \bar{\phi}(\alpha) - \mu\varphi(\alpha) \, . \tag{3.13}$$

We propose an iterative procedure to find the next trial step that is an "approximate" local minimizer for the univariate function $\bar{\psi}(\alpha)$. Note that the function evaluation for $\varphi(\cdot)$ only involves some logarithmic function ($\ln s_i$) and some reciprocal ($\frac{1}{s_i}$), and the effort to compute these quantities is negligible compared to the evaluation of $\phi(\alpha)$.

For univariate minimization, a general-purpose algorithm (for example, Newton's

Figure 3.5: Example 2 of quadratic interpolating for a barrier function



method) might be inefficient because of the singularity of the barrier function. Therefore, an efficient iterative univariate minimization algorithm based on a special interpolating function containing a logarithmic singularity is used and described later in this section.

3.3.1 Initial estimate

In the algorithm described in Table 3.1, it is important to find a good initial step because the hope is that most of the linesearch iterates terminate at the initial step. In this section, we describe an efficient algorithm to estimate the minimizer α^* of $\psi(\alpha)$ based on the current information and the special properties of barrier functions. For the initial step, the first ingredient in the process is to calculate the maximum allowable steplength, the step from $(\bar{x}^T, \bar{s}^T)^T$ along $(\Delta x^T, \Delta s^T)^T$ to the boundary of the feasible region. It can be easily computed by a ratio test:

$$\alpha_{\max} = \begin{cases} \min\left\{ -\frac{\bar{s}_i}{(\Delta s)_i} : \quad (\Delta s)_i < 0 \right\} & \text{if } (\Delta s)_i < 0 \text{ for some i} \\ \infty & \text{if } (\Delta s)_i \geq 0 \text{ for all i} \end{cases} . \tag{3.14}$$

When $\alpha_{\max} = \infty$, which indicates that there is no any singularity for moving $\bar{w}$ along p, we set $\alpha^{(0)} = 1$.

For the case that $\alpha_{\max} < \infty$, we know there is a singularity of $\psi(\alpha)$ at $\alpha = \alpha_{\max}$. The interpolating function for ψ has the form

$$\bar{\psi}(\alpha) = \bar{\phi}(\alpha) - \mu \cdot \varphi(\alpha) \ , \tag{3.15}$$

where

$$\bar{\phi}(\alpha) \quad = \quad a + b\alpha$$

is the linear interpolation function for ϕ at $\alpha = 0$, that is, $\bar{\phi}'(0) = b = \phi'(0)$. So, we derive a strategy to find the initial estimate linesearch step $\alpha^{(0)}$ as the minimizer of $\bar{\psi}(\cdot)$. With the descent direction property, we can claim the following.

Lemma 1 *The sufficient condition that a unique minimizer $\alpha_{\bar{\psi}}$ of $\bar{\psi}$ exists on $[0, \alpha_{\max}]$ is $\phi'(0) < 0$.*

Proof If the search direction is a descent direction, then

$$\bar{\psi}'(0) \quad = \quad b - \mu\varphi'(0) = \phi'(0) - \mu\varphi'(0) = \psi'(0) < 0 \ .$$

The first and second derivatives of $\bar{\Psi}(\cdot)$ are given by

$$\begin{aligned} \bar{\Psi}'(\alpha) &= b - \mu \sum_{i=1}^{N_s} \frac{(\Delta s)_i}{s + \alpha \cdot (\Delta s)_i} \ , \\ \bar{\Psi}''(\alpha) &= \mu \sum_{i=1}^{N_s} \frac{(\Delta s)_i^2}{(s + \alpha \cdot (\Delta s)_i)^2} \ . \end{aligned}$$

Since its second derivative is positive and $\lim_{\alpha \to \alpha_{\max}^{-}} \bar{\Psi}'(\alpha) = +\infty$, we know there exists a unique minimizer $\alpha_{\bar{\psi}}$ of $\bar{\psi}$ on $[0, \alpha_{\max}]$. $\blacksquare$

When only $\psi(0)$, $\psi'(0)$ and $\phi'(0)$ are known, we use the first interpolation function of the form

$$\Theta_1(\alpha) = a + b\alpha - \mu \cdot (\ln(\alpha_{\max} - \alpha) + c\alpha + f) \,. \tag{3.16}$$

The first and second derivatives of Θ_1 are given by

$$\begin{aligned} \Theta_1'(\alpha) &= (b - \mu c) + \frac{\mu}{\alpha_{\max} - \alpha} \,, \\ \Theta_1''(\alpha) &= \frac{\mu}{(\alpha_{\max} - \alpha)^2} \,. \end{aligned}$$

Because its second derivative is always positive, Θ_1 is strictly convex. A minimizer of Θ_1 occurs at α_{Θ_1} satisfying

$$\Theta_1'(\alpha_{\Theta_1}) = (b - \mu c) + \frac{\mu}{\alpha_{\max} - \alpha_{\Theta_1}} = 0 \,.$$

Hence,

$$\alpha_{\Theta_1} = \alpha_{\max} + \frac{\mu}{b - \mu c} \,.$$

Determination of c depends on fitting $\Theta_1(\alpha)$ to the values $\psi(\cdot)$ and $\psi'(\cdot)$ at $\alpha = 0$. It implies $b - c\mu = \psi'(0) - \dfrac{\mu}{\alpha_{\max}}$. Therefore,

$$\alpha_{\Theta_1} = \frac{\psi'(0)}{\psi'(0) - \alpha_{\max}} \alpha_{\max} \,. \tag{3.17}$$

Suppose an interval of uncertainty $[\alpha_{low}, \alpha_{high}]$ is known for $\bar{\psi}(\cdot)$ with $\bar{\psi}'(\alpha_{low}) < 0$ and $\bar{\psi}'(\alpha_{high}) > 0$. Then we use the second interpolation function of the form

$$\Theta_2(\alpha) = a + b\alpha - \mu \cdot (d \cdot \ln(\alpha_{\max} - \alpha) + c\alpha + f) \,. \tag{3.18}$$

The first and second derivatives of Θ_2 are given by:

$$\Theta_2'(\alpha) = (b - \mu c) + \frac{\mu d}{\alpha_{\max} - \alpha} \, , \tag{3.19}$$

$$\Theta_2''(\alpha) = \frac{\mu d}{(\alpha_{\max} - \alpha)^2} \, . \tag{3.20}$$

By fitting $\Theta_2(\alpha)$ to the value of $\bar{\psi}'(\cdot)$ at α_{low} and α_{high}, we have

$$b - c\mu + \frac{\mu d}{\alpha_{\max} - \alpha_{low}} = \bar{\psi}'(\alpha_{low}) \, , \tag{3.21}$$

$$b - c\mu + \frac{\mu d}{\alpha_{\max} - \alpha_{high}} = \bar{\psi}'(\alpha_{high}) \, . \tag{3.22}$$

Subtracting (3.21) from (3.22), we get

$$\mu d[\frac{1}{\alpha_{\max} - \alpha_{high}} - \frac{1}{\alpha_{\max} - \alpha_{low}}] = \bar{\psi}'(\alpha_{high}) - \bar{\psi}'(\alpha_{low}) \, ,$$

which implies

$$\mu d = \frac{(\alpha_{\max} - \alpha_{high})(\alpha_{\max} - \alpha_{low})}{\alpha_{high} - \alpha_{low}}[\bar{\psi}'(\alpha_{high}) - \bar{\psi}'(\alpha_{low})] > 0 \, . \tag{3.23}$$

It follows from (3.20) and $d > 0$ implied by (3.23) that Θ_2 is strictly convex. From the assumption that $\bar{\psi}'(\alpha_{low}) < 0$ and $\bar{\psi}'(\alpha_{high}) > 0$, a unique minimizer of Θ_2, say $\alpha_{\Theta_2} \in [\alpha_{low}, \alpha_{high}]$, satisfies

$$\Theta_2'(\alpha_{\Theta_2}) = (b - \mu c) + \frac{\mu d}{\alpha_{\max} - \alpha_{\Theta_2}} = 0 \, ,$$

which implies

$$\alpha_{\Theta_2} = \alpha_{\max} + \frac{\mu d}{b - \mu c} \, . \tag{3.24}$$

From (3.22) and (3.23), we obtain

$$b - c\mu = \bar{\psi}'(\alpha_{high}) - \frac{\mu d}{\alpha_{\max} - \alpha_{high}}$$

$$\begin{aligned} &= \bar{\psi}'(\alpha_{high}) - \frac{\alpha_{\max} - \alpha_{low}}{\alpha_{high} - \alpha_{low}}[\bar{\psi}'(\alpha_{high}) - \bar{\psi}'(\alpha_{low})] \\ &= \frac{\alpha_{\max} - \alpha_{low}}{\alpha_{high} - \alpha_{low}}\bar{\psi}'(\alpha_{low}) - \frac{\alpha_{\max} - \alpha_{high}}{\alpha_{high} - \alpha_{low}}\bar{\psi}'(\alpha_{high}) \ . \end{aligned} \tag{3.25}$$

Hence from (3.24) and (3.25), we know that α_{Θ_2} is given by

$$\alpha_{\Theta_2} = \alpha_{\max} + \frac{(\alpha_{\max} - \alpha_{high})(\alpha_{\max} - \alpha_{low})[\bar{\psi}'(\alpha_{high}) - \bar{\psi}'(\alpha_{low})]}{(\alpha_{\max} - \alpha_{low})\bar{\psi}'(\alpha_{low}) - (\alpha_{\max} - \alpha_{high})\bar{\psi}'(\alpha_{high})} \ . \tag{3.26}$$

Using these two analytical solutions for the interpolating functions, the initial estimate $\alpha^{(0)}$ can be obtained from the algorithm in Table 3.4.

3.3.2 The procedure for generating subsequent trial steps

When the initial estimate $\alpha^{(0)}$ does not satisfy the condition given in Table 3.1, we need to generate subsequent trial steps based on the information for $\psi(\cdot)$. Since the search direction is always a descent direction, we can choose the end points of the interval of uncertainty such that the function $\psi(\cdot)$ is decreasing at its left endpoint ($\psi'(\alpha_{low}) < 0$). Under this situation, the interpolating function for ψ has the form

$$\tilde{\psi}(\alpha) = \tilde{\phi}(\alpha) - \mu \cdot \varphi(\alpha) \ , \tag{3.27}$$

where

$$\tilde{\phi}(\alpha) \quad = \quad a_1 + b_1(\alpha - \alpha_{low}) + \frac{c_1}{2}(\alpha - \alpha_{low})^2$$

is the quadratic interpolating function for ϕ at $\alpha = \alpha_{low}$, the current left end point of the interval of uncertainty with $\psi'(\alpha_{low}) < 0$, and one of the steplengths $\alpha = \alpha_s$, which has been evaluated in the current linesearch procedure. Therefore, we have

$$b_1 = \phi'(\alpha_{low}) \ , \tag{3.28}$$

$$c_1 = \frac{1}{\alpha_s - \alpha_{low}}(\phi'(\alpha_s) - \phi'(\alpha_{low})) \ . \tag{3.29}$$

Table 3.4: Algorithm for initial estimate

Input: $\bar{w} = (\bar{x}, \bar{s})$ and $p = (\Delta x, \Delta s)$
Output: $\alpha^{(0)}$ is the initial step length in the linesearch iteration

$j = 0;$
$\alpha_{\max} = \min\{-\dfrac{\bar{s}_i}{(\Delta s)_i} : \quad (\Delta s)_i < 0\}$
if $\alpha_{\max} = \infty$ (no restriction on α)
    $\alpha^{(0)} = 1$
else
    $h = 1$
    $b = \nabla F(\bar{x})^T p$
    $\alpha_h^{(0)} = \dfrac{\nabla F_B(\bar{w})^T p}{\nabla F_B(\bar{w})^T p - \dfrac{\mu}{\alpha_{\max}}}$
    $\bar{\psi}'(\alpha_h^{(0)}) = b - \mu \nabla B(\bar{s} + \alpha_h^{(0)} \Delta s)^T \Delta s$
    while $\bar{\psi}'(\alpha_h^{(0)}) < -\rho$
        $h = h + 1$
        $\alpha_h^{(0)} = \eta \alpha_{\max} + (1 - \eta) \alpha_{h-1}^{(0)}$
        $\bar{\psi}'(\alpha_h^{(0)}) = b - \mu \nabla B(\bar{s} + \alpha_h^{(0)} \Delta s)^T \Delta s$
    end
    $\alpha_{low} = \alpha_{h-1}^{(0)}$ and $\bar{\psi}'(\alpha_{low}) = \bar{\psi}'(\alpha_{h-1}^{(0)})$
    $\alpha_{high} = \alpha_h^{(0)}$ and $\bar{\psi}'(\alpha_{high}) = \bar{\psi}'(\alpha_h^{(0)})$
    repeat
        $h = h + 1$
        $\alpha_{trial} = \alpha_{\max} + \dfrac{(\alpha_{\max} - \alpha_{low})(\alpha_{\max} - \alpha_{high})(\bar{\psi}'(\alpha_{high}) - \bar{\psi}'(\alpha_{low}))}{(\alpha_{\max} - \alpha_{low})\bar{\psi}'(\alpha_{low}) - (\alpha_{\max} - \alpha_{high})\bar{\psi}'(\alpha_{high})}$
        $\alpha_h^{(0)} =$ safeguard$(\alpha_{trial}, \alpha_{low}, \alpha_{high})$ (see Table 3.3)
        $\bar{\psi}'(\alpha_h^{(0)}) = b - \mu \nabla B(\bar{s} + \alpha_h^{(0)} \Delta s)^T \Delta s$
        Update the interval of uncertainty $(\alpha_h^{(0)}, \alpha_{low}, \alpha_{high})$ (see Table 3.2)
    until $|\ \bar{\psi}'(\alpha_h^{(0)})\ | > \rho$
    $\alpha^{(0)} = \alpha_h^{(0)}$
end
return $\alpha^{(0)}$

We now need to derive a strategy to find the subsequent linesearch step $\alpha^{(j)}$ as the approximate minimizer of $\tilde{\psi}(\cdot)$. Here, we also apply the technique of interval reduction. Suppose that the current interval of uncertainty for minimizing $\tilde{\psi}(\cdot)$ is $(\tilde{\alpha}_l, \tilde{\alpha}_h)$. From the choice of interval of uncertainty, we can guarantee that $\tilde{\psi}'(\tilde{\alpha}_l) < 0$. We use a third interpolating function of the form

$$\begin{aligned} \Theta_3(\alpha) &= a_1 + b_1(\alpha - \alpha_{low}) + \frac{c_1}{2}(\alpha - \alpha_{low})^2 \\ &\quad -\mu(\ln(\alpha_{\max} - \alpha) + a_2 + b_2(\alpha - \tilde{\alpha}_l) + \frac{c_2}{2}(\alpha - \tilde{\alpha}_l)^2) . \end{aligned} \tag{3.30}$$

To simplify the calculation, denote

$$\begin{aligned} a &= a_1 + b_1(\tilde{\alpha}_l - \alpha_{low}) + \frac{c_1}{2}(\tilde{\alpha}_l - \alpha_{low})^2 - \mu a_2 , \\ b &= b_1 + c_1(\tilde{\alpha}_l - \alpha_{low}) - \mu b_2 , \\ c &= c_1 - \mu c_2 . \end{aligned}$$

Then, the interpolating function $\Theta_3(\cdot)$ becomes

$$\Theta_3(\alpha) = a + b(\alpha - \tilde{\alpha}_l) + \frac{c}{2}(\alpha - \tilde{\alpha}_l)^2 - \mu \ln(\alpha_{\max} - \alpha). \tag{3.31}$$

The first and second derivatives of Θ_3 are given by

$$\Theta_3'(\alpha) = b + c(\alpha - \tilde{\alpha}_l) + \frac{\mu}{\alpha_{\max} - \alpha} , \tag{3.32}$$

$$\Theta_3''(\alpha) = c + \frac{\mu}{(\alpha_{\max} - \alpha)^2} . \tag{3.33}$$

Because of the singularity of $\tilde{\psi}(\cdot)$ at $\alpha_{\max}$, it is not appropriate to use $\tilde{\alpha}_h$ to interpolate the function if $\tilde{\alpha}_h = \alpha_{\max}$. By fitting $\Theta_3'(\alpha)$ to the value of $\tilde{\psi}'(\cdot)$ at $\tilde{\alpha}_l$ and $\tilde{\alpha}_s$, we have

$$b + \frac{\mu}{\alpha_{\max} - \tilde{\alpha}_l} = \tilde{\psi}'(\tilde{\alpha}_l) , \tag{3.34}$$

$$b + c(\tilde{\alpha}_s - \tilde{\alpha}_l) + \frac{\mu}{\alpha_{\max} - \tilde{\alpha}_s} = \tilde{\psi}'(\tilde{\alpha}_s) . \tag{3.35}$$

Subtracting (3.34) from (3.35), we get

$$c = \frac{1}{\tilde{\alpha}_s - \tilde{\alpha}_l}(\tilde{\psi}'(\tilde{\alpha}_s) - \tilde{\psi}'(\tilde{\alpha}_l) + \frac{\mu}{\alpha_{\max} - \tilde{\alpha}_l} - \frac{\mu}{\alpha_{\max} - \tilde{\alpha}_s}) \tag{3.36}$$

and

$$b = \tilde{\psi}'(\tilde{\alpha}_l) - \frac{\mu}{\alpha_{\max} - \tilde{\alpha}_l} . \tag{3.37}$$

Because $\Theta_3'(\tilde{\alpha}_l) < 0$ based on the choice of the interval of uncertainty and $\Theta_3'(\alpha)$ is unbounded above as $\alpha \to \alpha_{\max}$, at least one minimizer of $\Theta_3(\cdot)$ must exist in the interval of $(\tilde{\alpha}_l, \alpha_{\max})$. To simplify the computation, we denote $\alpha = z + \tilde{\alpha}_l$ and $\bar{\alpha}_{\max} = \alpha_{\max} - \tilde{\alpha}_l$.

A stationary point $\alpha_{\Theta_3} = z_* + \tilde{\alpha}_l$ of $\Theta_3(\cdot)$ occurs when $\Theta_3'(\cdot) = 0$. It implies that z_* satisfies the following quadratic equation:

$$cz_*^2 + (b - c\bar{\alpha}_{\max})z_* - \mu - b\bar{\alpha}_{\max} = 0 . \tag{3.38}$$

If $c = 0$, then there is no quadratic term and the minimizer of $\Theta_3(\cdot)$ is $\alpha_{\Theta_3} = \tilde{\alpha}_l + z_* = \tilde{\alpha}_l + \dfrac{\mu + b\bar{\alpha}_{\max}}{b - c\bar{\alpha}_{\max}}$. Otherwise, we have the following result.

Lemma 2 *If $c \neq 0$, then $\dfrac{c\bar{\alpha}_{\max} - b - \sqrt{(b - c\bar{\alpha}_{\max})^2 + 4c(\mu + b\bar{\alpha}_{\max})}}{2c}$ is the unique root of (3.38) in $(0, \bar{\alpha}_{\max})$ under the assumption that $\tilde{\psi}'(\tilde{\alpha}_l) < 0$.*

Proof. If $c \neq 0$, there are two roots r_1 and r_2 that can be expressed in two forms:

$$r_1, r_2 = \frac{c\bar{\alpha}_{\max} - b \pm \sqrt{(b - c\bar{\alpha}_{\max})^2 + 4c(\mu + b\bar{\alpha}_{\max})}}{2c} \tag{3.39}$$

$$= \frac{c\bar{\alpha}_{\max} - b \pm \sqrt{(b + c\bar{\alpha}_{\max})^2 + 4c\mu}}{2c} , \tag{3.40}$$

and the product of r_1 and r_2 satisfies

$$r_1 r_2 = \frac{-(b\bar{\alpha}_{\max} + \mu)}{c} . \tag{3.41}$$

Case 1. If $c > 0$, r_1 and r_2 both are real (from (3.40)) and have the same sign (from (3.41)). Furthermore,

$$\sqrt{(b + c\bar{\alpha}_{\max})^2 + 4c\mu} > \mid b + c\bar{\alpha}_{\max} \mid = \max\left\{b + c\bar{\alpha}_{\max}, -b - c\bar{\alpha}_{\max}\right\} . \tag{3.42}$$

If $b + c\bar{\alpha}_{\max} > 0$, using (3.41) and (3.42) we can obtain

$$r_1 \;\; > \;\; \frac{c\bar{\alpha}_{\max} - b + (b + c\bar{\alpha}_{\max})}{2c} > \bar{\alpha}_{\max} . \tag{3.43}$$

If $b + c\bar{\alpha}_{\max} < 0$, it implies that $\bar{\alpha}_{\max} < -\frac{b}{c}$. Combining (3.41) and (3.42) we have

$$r_1 \;\; > \;\; \frac{c\bar{\alpha}_{\max} - b - (b + c\alpha_{\max})}{2c} = -\frac{b}{c} > \bar{\alpha}_{\max} . \tag{3.44}$$

From (3.43), (3.44) and the fact that at least one root of (3.38) must exist in the interval $(0, \bar{\alpha}_{\max})$, it follows that $r_2 \in (0, \bar{\alpha}_{\max})$.

Case 2. If $c < 0$, r_1 and r_2 both are real (from (3.40)) and (3.41) shows that they have the opposite sign. We also know that $4c(\mu + b\bar{\alpha}_{\max}) > 0$ because $c < 0$ and $\mu + b\bar{\alpha}_{\max} < 0$. Therefore, from (3.39) we have

$$\begin{aligned} & c\bar{\alpha}_{\max} - b + \sqrt{(b - c\bar{\alpha}_{\max})^2 + 4c(\mu + b\bar{\alpha}_{\max})} \\ & > c\bar{\alpha}_{\max} - b + \mid b - c\bar{\alpha}_{\max} \mid > 0 . \end{aligned} \tag{3.45}$$

It follows that $r_1 < 0$ and $r_2 \in (0, \bar{\alpha}_{\max})$ by the same argument as in case 1. $\blacksquare$

From (3.37), (3.36) and Lemma 2, we can generate the subsequent trial step $\alpha^{(j)}$ by using the algorithm in Table 3.5.

3.4 Linesearch on the Lagrange Multiplier

In most primal-dual algorithms, the steplength on the Lagrange multiplier is either the same as for x and s, i.e., $\alpha_\lambda = \alpha_x$ if that is acceptable, or a certain fraction of the maximal allowable steplength to keep the Lagrange multiplier positive, i.e., $\alpha_\lambda = \eta\alpha_\lambda^{\max}$. However, we believe we should be able to do better than that. First of

Table 3.5: Algorithm for generating subsequent Trial Steps

Input: $\bar{w} = (\bar{x}, \bar{s})$, $p = (\Delta x, \Delta s)$, $\alpha_{\max}$, α_{low}, α_s, $\psi'(\alpha_{low})$ and $\psi'(\alpha_s)$
Output: $\alpha^{(j)}$ is the j^{th} trial step length in the linesearch iteration

$h = 0$
$b_1 = \phi'(\alpha_{low})$ and $c_1 = \dfrac{1}{\alpha_s - \alpha_{low}}(\phi'(\alpha_s) - \phi'(\alpha_{low}))$
$\tilde{\alpha}_l = \alpha_{low}$ and $\tilde{\alpha}_s = \alpha_s$
repeat
  $h = h + 1$
  $\tilde{\alpha}_{\max} = \alpha_{\max} - \tilde{\alpha}_l$
  $b = \tilde{\psi}'(\tilde{\alpha}_l) - \dfrac{\mu}{\tilde{\alpha}_{\max}}$
  $c = \dfrac{1}{\tilde{\alpha}_s - \tilde{\alpha}_l}\left(\tilde{\psi}'(\tilde{\alpha}_s) - \tilde{\psi}'(\tilde{\alpha}_l) + \dfrac{\mu}{\tilde{\alpha}_{\max}} - \dfrac{\mu}{\alpha_{\max} - \tilde{\alpha}_s}\right)$
  if $c = 0$
    $$\alpha_{trial} = \tilde{\alpha}_l + \frac{\mu + b\tilde{\alpha}_{\max}}{b}$$
  else
    $$\alpha_{trial} = \tilde{\alpha}_l + \frac{c\tilde{\alpha}_{\max} - b - \sqrt{(b + c\tilde{\alpha}_{\max})^2 + 4c\mu}}{2c}$$
  end
  $\alpha_h^{(j)} = \text{safeguard}(\alpha_{trial}, \tilde{\alpha}_l, \tilde{\alpha}_{high})$ (see Table 3.3)
  $\bar{\psi}'(\alpha_h^{(j)}) = b_1 + c_1(\alpha_h^{(j)} - \alpha_{low}) - \mu \nabla B(\bar{s} + \alpha_h^{(j)}\Delta s)^T \Delta s$
  Update the interval of uncertainty $(\alpha_h^{(j)}, \tilde{\alpha}_l, \tilde{\alpha}_{high})$ (see Table 3.2)
until $\mid \tilde{\psi}'(\alpha_h^{(j)}) \mid > \rho$
$\alpha^{(j)} = \alpha_h^{(j)}$

all, the linesearch on x and s is independent of the Lagrange multiplier λ. Therefore, we can decide the steplength α_λ on λ after the linesearch procedure for x and s. In the latter, the gradient of the objective function for the new iterate is obtained. In each primal-dual iteration, we try to find a point that is close to the optimality condition (2.4) of the barrier subproblem (2.1). Since the first and second equations of the optimality condition (2.4) involve the Lagrange multiplier λ, we consider choosing the steplength α_λ as the minimizer of

$$\rho_1 \|S(\lambda^k + \alpha_\lambda \Delta\lambda^k) - \mu_k e\|^2 + (1-\rho_1)\|Z^T(g_{x^{k+1}} - (\lambda_l^k + \alpha_\lambda \Delta\lambda_l^k) + (\lambda_u^k + \alpha_\lambda \Delta\lambda_u^k))\|^2 \, , \tag{3.46}$$

a convex combination of the two-norm of the first and second equations of (2.4).

The minimizer $\tilde{\alpha}_\lambda$ of (3.46) is given by

$$\tilde{\alpha}_\lambda = \frac{\rho_1(\Delta\lambda^k)^T S(S\lambda^k - \mu_k e) + (1-\rho_1)(g_{x^{k+1}} - \lambda_l^k + \lambda_u^k)^T Z Z^T(\Delta\lambda_u^k - \Delta\lambda_l^k)}{\rho_1\|S\Delta\lambda^k)\|^2 + (1-\rho_1)\|Z^T(\Delta\lambda_u^k - \Delta\lambda_l^k)\|^2} \, . \tag{3.47}$$

The computation of (3.47) needs matrix-vector multiplications $Z^T(\Delta\lambda_u^k - \Delta\lambda_l^k)$ and $Z^T(g_{x^{k+1}} - \lambda_l^k + \lambda_u^k)$, which are cheap to compute (2n additions/subtractions).

We should note that the new multiplier iterate obtained from $\lambda^{k+1} = \lambda^k + \tilde{\alpha}_\lambda \Delta\lambda^k$ may not be strictly positive as we need. To ensure the strictly interior property, we set the steplength α_λ as

$$\alpha_\lambda = \min\{\rho_2 \alpha_\lambda^{\max}, \ \tilde{\alpha}_\lambda\} \ . \tag{3.48}$$

Chapter 4

Implementation

In this chapter, we describe NPDNET, an implementation of the null-space truncated primal-dual algorithm for solving nonlinear network problems. We first specify the initial estimates and the parameter settings and describe the complete version of the NPDNET algorithm in section 4.1. Secondly, we conduct some numerical experiments on problems of the form (1.3). In section 4.2, we describe the test problems, the performance results and the comparisons with a general-purpose large-scale nonlinear optimization solver, SNOPT [38].

NPDNET is a collection of Fortran 77 subroutines for solving nonlinear pure network optimization problems of the form (1.3). The user must provide a subroutine to define the function value, the gradient and the Hessian matrix of the objective, and four arrays, $node_{from}$, $node_{to}$, l and u, to define the network. The format of the user-supplied subroutine is referred to as SNOPT format, which can be found in [38].

4.1 Other Implementation Issues

To complete the algorithm description, we make some remarks on other important implementation issues of the truncated null-space primal-dual algorithm, namely the starting solution and parameter settings.

The primal-dual algorithm starts with any solution $\{x^0, s_l^0, s_u^0, \lambda_l^0, \lambda_u^0\}$ satisfying

$$s_l^0 > 0,\ s_u^0 > 0,\ \lambda_l^0 > 0,\ \lambda_u^0 > 0 \tag{4.1}$$

and

$$x^0 - s_l^0 = l\ , \tag{4.2}$$

$$x^0 - s_u^0 = u\ . \tag{4.3}$$

Additionally, it is desirable that the initial point also satisfy the remaining equation in the modified KKT system (2.4) that defines the central path. Consequently, we also require that the starting solution satisfy the following equations:

$$S_l^0 \lambda_l^0 = \mu^0 e \quad \text{and} \quad S_u^0 \lambda_u^0 = \mu^0 e\ , \tag{4.4}$$

for $\mu_0 > 0$.

In NPDNET, we let

$$\begin{aligned} x^0 &= (u+l)/2\ , \\ s_l^0 &= (u+l)/2\ , \\ s_u^0 &= (u+l)/2\ , \\ \lambda_l^0 &= \mu^0 (S_l^0)^{-1} e\ , \\ \lambda_u^0 &= \mu^0 (S_u^0)^{-1} e\ , \end{aligned} \tag{4.5}$$

be the starting solution.

In NPDNET, the primal-dual barrier parameter has an initial value

$$\mu_0 = 500\ . \tag{4.6}$$

Subsequently, for iteration $k \geq 1$, the procedure for updating μ^k is

$$\begin{aligned} &\text{if } (M(x^k, s^k, \lambda_l^k, \lambda_u^k, \mu^k) < \vartheta_1 \mu^{k-1} \text{and } \min \operatorname{eig}(Z^T H Z) > -\vartheta_2 \mu) \\ &\qquad \mu^k = \vartheta_3 \mu^{k-1} \end{aligned}$$

else

$\mu^k = \mu^{k-1}$

end if

as described in Table 2.1 with

$$\vartheta_1 = 1 \ , \quad \vartheta_2 = 1 \quad \text{and} \quad \vartheta_3 = 0.02 \ . \tag{4.7}$$

As described in Table 2.1, the stopping criterion is

$$M(x^k, s^k, \lambda_l^k, \lambda_u^k, 0) < \epsilon_{PD} \ , \tag{4.8}$$

where we set the optimality tolerance

$$\epsilon_{PD} = 10^{-8} \ . \tag{4.9}$$

In the Preconditioned Conjugate-Gradient method, we terminate the PCG process when one of the criteria

$$\|r_i\| \le \tau_1 \sqrt{\mu_k} \|r_0\| \ , \tag{4.10}$$

$$\frac{(Zy_i)^T g_B^k}{\|Zy_i\| \cdot \|g_B^k\|} \le -\tau_2 \ , \tag{4.11}$$

or

$$\text{CGiter} > \tau_3(n-m) \tag{4.12}$$

is triggered. In NPDNET, we set

$$\tau_1 = 0.02 \ , \quad \tau_2 = 10^{-8} \quad \text{and} \quad \tau_3 = 0.2 \ . \tag{4.13}$$

In the spanning tree updating algorithm, see Table 2.5, the threshold weight for adding a non-basic arc to the spanning tree is

$$thred_{add} = \zeta_1 * w_{sort}(m) \ , \tag{4.14}$$

and the threshold weight for deleting a basic arc from the spanning tree is

$$thred_{del} = \zeta_2 * w_{sort}(m) \ , \tag{4.15}$$

where $w_{sort}(m)$ is the m-th largest element of the weight w. In NPDNET, we set

$$\zeta_1 = 10 \quad \text{and} \quad \zeta_2 = \frac{1}{10} \ . \tag{4.16}$$

In the algorithm for updating the spanning tree by deleting an arc, see Table 2.7, the criterion for an acceptable arc is an arc with one end node in the rooted tree and another in the subtree whose weight is greater than

$$\zeta_3 w(i_{del}) \ . \tag{4.17}$$

In NPDNET, we set

$$\zeta_3 = 10. \tag{4.18}$$

For the stopping rule in the algorithm for linesearch iterate, see Table 3.1, the initial trial steplength $\alpha_k^{(0)}$ is acceptable when

$$\Psi(\alpha_k^{(0)}) - \Psi(0) \leq \gamma_3 \alpha_k^{(0)} \psi'(0) \ , \tag{4.19}$$

and the subsequent trial step $\alpha_k^{(j)}$ is acceptable when it satisfies the Gamma rule:

$$\mid \psi'(\alpha_k^{(j)}) \mid \leq -\gamma_1 \psi'(0) \ , \tag{4.20}$$

and

$$\psi(\alpha_k^{(j)}) \leq \psi(0) + \gamma_2 \alpha_k^{(j)} \psi'(0) \ . \tag{4.21}$$

In NPDNET, we set

$$\gamma_1 = 0.85 \ , \quad \gamma_2 = \gamma_3 = 0.1 \ . \tag{4.22}$$

If x_k is close to a saddle point then the direction obtained may not be a direction of sufficient descent. What is happening is the magnitude of the direction of negative curvature may be large compared to the direction of descent. The likelihood of this occurrence is small because saddle points are not points of attraction for the algorithm. Therefore, it is not worthwhile having a special search under these circumstances or having complex termination criteria. It has sufficient to have a simple back-tracking algorithm that terminates at the first point lower than the current iterate.

For a given interval of uncertainty $(\alpha_{low}, \alpha_{high})$, we define a safeguarded interval in the safeguarded steplength algorithm, see Table 3.3, as

$$[(1-\eta_1)\alpha_{low} + \eta_1\alpha_{high}, \eta_1\alpha_{low} + (1-\eta_1)\alpha_{high}] \ . \tag{4.23}$$

To prevent the slow convergence cases caused by the barrier function, we take the trial steplength

$$\eta_2\alpha_{low} + (1-\eta_2)\alpha_{high} \tag{4.24}$$

if it happens three times in a row that the trial step α_{trial} obtained from the interpolation method is less than the lower bound of the safeguarded interval and the interval of uncertainty keeps updating the left-end point. In NPDNET, we set

$$\eta_1 = 0.2 \ , \quad \eta_2 = 0.02 \ . \tag{4.25}$$

In the algorithm for initial trial step, see Table 3.4, and the algorithm for generating subsequent trial steps, see Table 3.5, we use an iterative procedure to find the minimizer of the interpolating functions $\bar{\psi}(\cdot)$ and $\tilde{\psi}(\cdot)$, respectively. The stopping criterion for the iterative procedure is

$$\mid \bar{\psi}'(\alpha^{(0)}) \mid < \xi_1 \tag{4.26}$$

and

$$\mid \tilde{\psi}'(\alpha^{(h)}) \mid < \xi_2 \ . \tag{4.27}$$

In NPDNET, we set

$$\xi_1 = 0.1 \ , \quad \xi_2 = 0.1 \ . \tag{4.28}$$

Table 4.1: Truncated null-space primal-dual algorithm

Initialize x_0, μ_0, s_0 and λ_0 by (4.5) and (4.6)
$k = 0$
repeat
  if $(M(x^k, s^k, \lambda_l^k, \lambda_u^k, \mu^k) < \vartheta_1 \mu^{k-1}$ and $\min \mathrm{eig}(Z^T H Z) > -\vartheta_2 \mu)$
    $\mu^k = \vartheta_3 \mu^{k-1}$
  else
    $\mu^k = \mu^{k-1}$
  end if
  call spanning tree updating algorithm, Table 2.5
  call modified CG-Lanczos algorithm, Table 2.4, for Δx_k
  compute Δs^k and $\Delta \lambda^k$ from (2.6)
  call linesearch procedure, Table 3.1, for α_x^k
  $x^{k+1} = x^k + \alpha_x^k \Delta x^k$
  $s^{k+1} = s^k + \alpha_x^k \Delta s^k$
  compute α_λ^k from (3.48)
  $\lambda^{k+1} = \lambda^k + \alpha_\lambda^k \Delta \lambda^k$
  $k = k + 1$
until $M(x^k, s^k, \lambda_l^k, \lambda_u^k, 0) < \epsilon_{PD}$ **and** $\min \mathbf{eig}(Z^T H Z) \geq 0$

To conclude, we give a complete version of truncated null-space primal-dual algorithm in Table 4.1.

4.2 Experimental Results and Comparisons

In the context of network optimization, most researchers are interested in problems with either a linear objective or some special nonlinear objective, e.g., a separable function, convex/quadratic function or concave function. General-purpose network optimization has not received much attention in the literature. As a result, there does not exist any test problem set for general nonlinear network optimization in the public domain. To test our algorithm, we constructed two sets of test problems

Table 4.2: Hardware and system configuration

CPU :	AMD Athlon processor at 1.2 GHZ
Memory size :	768 Mbytes
Operating system :	Microsoft Windows 2000
Fortran Compiler :	Fortran PowerStation 4.0

by combining unconstrained minimization test problems and two network constraint generators. Details are given in section 4.2.1.

The experiments were performed on a PC with the configuration summarized in Table 4.2. Though this hardware is fast for integer arithmetic, it is less efficient for floating-point computation. We expect NPDNET to benefit from systems with improved floating-point performance relative to integer arithmetic. A large set of computational results is presented and discussed in section 4.2.2. We then compare some of our results to SNOPT in section 4.2.3.

4.2.1 Test problems

In our test problem set, we generate the objective function and the constraints separately. For the objective functions $F(x)$, we use the Buckley test set containing 142 unconstrained test problems. Table 4.3 lists the function name, problem name, and the size of each implemented test problem. The function value, the gradient and the Hessian are provided for all test problems.

For the network constraints, we consider two different sets of problems. The first set of network constraints is from netgen [62], a pure minimum cost linear network flow problem generator. The generator can be retrieved from the ftp site **dimacs.rutgers.edu**. For the first set of test problems—Network Flow Problems (NFP)—we generate the input file for the minimum cost linear network flow problem from netgen and replace the linear objective function by one of the nonlinear functions from the Buckley test set.

Table 4.3: The Buckley test set

function	problem	size	function	problem	size	function	problem	size
ARGAUS	ARGAUS	3	GENTSN	GENT2B	2	PENAL3	PENL3GM5	1000
ARGQDN	ARGQDN50	5	GENTSN	GENT50A	3	POWBSC	POWBSC	2
ARGQDO	ARGQO10	5	GENTSN	GENT500A	3	POWBSC	POWBSC50	50
ARGQDZ	ARGQDZ10	3	GENTSN	GENT1000	10	POWBSC	POWBS500	100
ARTRIG	ARTRIG10	10	GOTTFR	GOTTFR	12	POWBSC	POWB1000	1000
AVRIEL	AVRIEL3	2	GULF	GULFSH2	2	POWER	POWER10	10
BARD70	BARD70	3	HELIX	HELIX	2	POWER	POWER75	75
BEAL58	BEAL58KO	2	HILBRT	HILBR10A	2	POWQUD	POWQUD8A	4
BIGGS	BIGGS6	6	HILBRT	HILBRT12	2	POWSSQ	POWSSQ	2
BOOTH	BOOTH	2	HIMLN3	HIMLN3	2	PWSING	PWSING5	4
BOX66	BOX662HL	2	HIMM1	HIMM1	2	PWSING	PWSING60	60
BRKMCC	BRKMCC	2	HIMM25	HIMM25	2	PWSING	PWSIN100	100
BROWNB	BROWNB	2	HIMM27	HIMM27	2	PWSING	PWSI1000	1000
BROWND	BROWND	4	HIMM28	HIMM28	2	QUARTC	QUARTC	25
BROY7D	BROY7D	60	HIMM29	HIMM29	2	RECIPE	RECIPE	3
BRWNAL	BRWNAL10	10	HIMM30	HIMM30	3	ROSENB	ROSENB	2
BRWNAL	BRWNL100	100	HIMM32	HIMM32	4	SHNRSN	SHNRSN10	10
BRYBND	BRYBND	10	HIMM33	HIMM33A	2	SARSEB	SARSEB	4
BRYBND	BRYBND18	100	HYPCIR	HYPCIR	2	SCHMVT	SCHMVT	3
BRYTRI	BRYTRI2	5	JENSMP	JENSMP	2	SCHMVT	SCHMVT50	50
BRYTRI	BRYTRI6	20	KOWOSB	KOWOSB1	4	SCHMVT	SCHMV500	500
BRYTRI	BRYTRI10	600	MANCIN	MANCIN10	10	SCHMVT	SCHV1000	1000
CHEBYQ	CHEBYQ8	8	MANCIN	MANCIN50	50	SISSER	SISSER	2
CHEBYQ	CHEBYQ10	10	MEYER	MEYER	3	SQRTMX	MSQRTB9	9
CHNRSH	CHNRSH10	10	NMSRF1	NMSURF64	36	SQRTMX	MSQRTB49	49
CLIFF	CLIFF	2	NMSRF1	NMSUR484	400	TDQUAD	TDQ10	10
CLUSTR	CLUSTR	2	NMSRF2	SNMSUR64	36	TDQUAD	TDQ500	500
CRGLVY	CRGLVY	4	NMSRF2	SNMSR484	400	TDQUAD	TDQ1000	1000
CRGLVY	CRGLVY10	10	MORCIN	MORCIN10	10	TOIN2	TOIN2	3
CRGLVY	CRGLY500	500	MOREBV	MOREBV10	10	TOIN4	TOIN4	4
CRGLVY	CRGL1000	1000	MOREBV	MOREBV18	18	TOINT	PSPTOINT	50
DIX7DG	DIX7DGA	15	MOREBV	MORBV998	998	TRIDIA	TRIDIA10	10
DIXON	DIXON	10	NONDIA	NONDIA10	10	TRIDIA	TRLN100	100
ENGVL1	ENGVL1A	2	NONDIA	NONDIA20	20	TRIDIA	TRLN1000	1000
ENGVL1	ENGVL1B2	10	NONDIA	NONDI500	500	TRIGTO	TRIGT50	50
ENGVL1	ENGVL1B4	100	NONDIA	NOND1000	1000	TRIGTO	TRIGT100	100
ENGVL1	ENGVL1B6	1000	OSBRN1	OSBRN1	5	VARDIM	VARDIM	10
ENGVL2	ENGVL2	3	OSBRN2	OSBRN2	11	VARDIM	VARDIM100	100
EXTRSN	EXTRAR10	10	PENAL1	PEN1GM6	10	VAROSE	VAROSEBG1	50
EXTRSN	EXTRAR50	50	PENAL1	PEN1LN1	50	VAROSE	VAROSEBG2	100
EXTRSN	EXTRA100	100	PENAL1	PEN1LN2	100	WATSON	WATSON6	6
EXTRSN	EXTR1000	1000	PENAL1	PEN1LN3	1000	WATSON	WATSON12	12
FRANK	FRANK8	8	PENAL2	PEN2GM6	4	WOODS	WOODS	4
FRANK	FRANK12	12	PENAL2	PEN2GM1	50	WOODS	WOODS80	80
FRDRTH	FRDRTH	2	PENAL2	PEN2GM2	100	XTX	XTX2	2
FRDRTH	FRDRTHB3	50	PENAL3	PENL3GM3	50	XTX	XTX16	16
FRDRTH	FRDRTHB4	100	PENAL3	PENL3GM4	100	ZANGWL	ZANGWL1	3
FRDRTH	FRDRTHB7	1000						

The second set of network constraints that we consider is doubly stochastic constraints. These arise in the content of statistical estimation problems, where both the row sum and the column sum of a square matrix are equal to one. In mathematical form,

$$\sum_{j=1}^{j=m} x_{ij} = 1, \qquad \forall\, 1 \le i \le m \tag{4.29}$$

$$\sum_{i=1}^{i=m} x_{ij} = 1, \qquad \forall\, 1 \le j \le m \tag{4.30}$$

$$0 \le x_{ij} \le 1, \qquad \forall\, 1 \le i \le m \text{ and } 1 \le j \le m\ . \tag{4.31}$$

The doubly stochastic constraints can be reformulated as network constraints by multiplying both sides of constraints (4.30) by -1. The network interpretation for doubly stochastic constraints is as shown in Figure 4.1, where the node r_i represents row i and the node c_i represents column j. The arc between node r_i and node c_j represents component x_{ij}. This set of network contains $2m$ nodes and m^2 arcs. The second set of test problems—Doubly Stochastic Problems (DSP)—have a nonlinear objective function from the Buckley test set and a set of doubly stochastic constraints. For any integer m, an interior feasible point for DSP always exists because the matrix with all components $1/m$ is feasible.

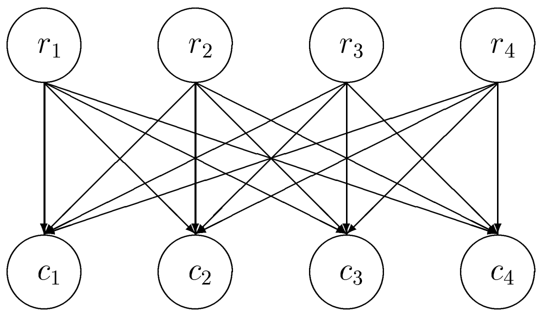


Figure 4.1: Network interpretation for doubly stochastic problem

4.2.2 Computational results

In this section, we present preliminary results with our code NPDNET. For every class of problems, we summarize the runs of the problems solved by NPDNET in tables that include the following information:

Name the name of the objective function,

N_a the number of arcs in the network,

N_n the number of nodes in the network,

N_{obj} the number of variables involved in the objective function evaluation,

iter the number of primal-dual iterations,

func the number of function evaluations,

CB the number of spanning tree updates,

CG the number of CG iterations,

N_{act} the number of active variables at the optimal solution,

T_{fun} the CPU time (in seconds) that NPDNET spends in the function evaluations,

T_{total} the total CPU time (in seconds) that NPDNET spends to solve the problem.

For the test set DSP, we first set the size of the problem to be $m = \sqrt{4N_{obj}} + 1$. Tables 4.4–4.7 summarize the runs for this DSP test set. We make the following remarks regarding the computational results:

- NPDNET required an average of 29.4 primal-dual iterations to converge to the optimal solution, and the number of iterations does not grow as the size of the problem grows.

- On average, it takes 1.828 function evaluations per primal-dual iteration. This result indicates that our linesearch algorithm works very well in the primal-dual algorithm.

- The average ratio of $\frac{\mathrm{CB}}{iter \times N_n}$ is 0.029. This means that only 3 percent of the basis variables were updated in each iteration and confirms that the number of basis variables that we need to change is small from one iteration to the next.

For the test set NFP, we first generated 5 network flow problems with different sizes from netgen. We created a code that reads the standard netgen output file to fit our data structure. For each test problem, we decided which set of constraints to choose based on N_{obj}, the number of variables involved in the function evaluation. Tables 4.8–4.11 summarize the runs for this NFP test set. For the NFP runs, we make the following remarks:

- NPDNET required an average of 32.16 primal-dual iterations to converge to the optimal solution. This is a little more than for the DSP problems because NPDNET spends more iterations in phase I.

- On average, it takes 1.46 function evaluations per primal-dual iteration. The average ratio of $\frac{\mathrm{CB}}{iter \times N_n}$ is 0.035.

- NPDNET fails to converge on some of the NFP test problems, for instance CRGL1000, PENL3GM5, POWBS500, POWB1000, SCHV1000 and VARDIM100, because the objective function is not well defined on the feasible region.

In order to test NPDNET on the large problems, we chose 9 functions, including CRGL1000, ENGVL1B6, EXTR1000, GENT1000, NOND1000, PEN1LN3, PWSI1000 and TDQ1000, with 1000 variables involved in the function evaluation, and we generated large DSP constraint sets of different sizes. Table 4.12 summarizes the runs for the large DSP test set. For those runs, we make the following remarks:

- The number of primal-dual iterations stays the same as the problem size grows.

- The growth rate of CPU time for NPDNET is close to a linear rate.

Table 4.4: Computational results for DSP set 1

Name	N_a	N_n	N_{obj}	iter	func	CB	CG	N_{act}	T_{fun}	T_{total}
AVRIEL3	17	8	2	19	23	5	61	2	0.00	0.02
BEAL58KO	17	8	2	16	20	5	51	7	0.00	0.05
BOOTH	17	8	2	16	18	2	40	7	0.00	0.02
BOX662HL	17	8	2	17	21	2	59	7	0.00	0.02
BRKMCC	17	8	2	16	20	3	42	2	0.00	0.04
BROWNB	26	10	2	18	18	7	53	9	0.00	0.02
CLIFF	17	8	2	16	31	5	61	0	0.00	0.02
CLUSTR	17	8	2	15	21	3	44	2	0.00	0.04
ENGVL1A	17	8	2	20	36	5	78	6	0.00	0.03
FRDRTH	17	8	2	18	20	5	62	7	0.00	0.02
GENT2B	17	8	2	17	23	4	65	2	0.00	0.03
GOTTFR	17	8	2	27	54	2	117	0	0.00	0.04
HIMLN3	17	8	2	21	44	5	85	2	0.00	0.02
HIMM1	17	8	2	16	20	3	39	7	0.00	0.02
HIMM25	17	8	2	16	19	5	50	7	0.00	0.02
HIMM27	17	8	2	14	29	3	38	0	0.00	0.06
HIMM28	17	8	2	18	21	2	53	7	0.00	0.03
HIMM29	17	8	2	73	223	2	353	0	0.00	0.07
HIMM33A	17	8	2	20	37	2	78	2	0.00	0.03
HYPCIR	17	8	2	19	24	5	51	7	0.00	0.03
JENSMP	17	8	2	10	58	2	44	0	0.00	0.02
POWBSC	17	8	2	59	120	4	226	5	0.00	0.06
POWSSQ	17	8	2	21	32	2	88	1	0.00	0.02
ROSENB	17	8	2	20	26	4	78	2	0.00	0.02
SISSER	17	8	2	15	25	2	54	0	0.00	0.03
XTX2	17	8	2	17	32	2	65	0	0.00	0.03
ARGAUS	26	10	3	17	29	5	77	3	0.00	0.03
ARGQDZ10	26	10	3	17	63	3	99	0	0.01	0.03
BARD70	26	10	3	19	19	6	105	9	0.00	0.02
ENGVL2	26	10	3	24	53	3	102	9	0.00	0.04
GULFSH2	26	10	3	25	56	7	95	9	0.02	0.06
HELIX	26	10	3	25	54	7	141	9	0.00	0.01
HIMM30	26	10	3	16	18	5	76	2	0.00	0.03
MEYER	37	12	3	19	19	9	64	11	0.00	0.03
SCHMVT	26	10	3	16	25	5	83	2	0.00	0.04

Table 4.5: Computational results for DSP set 2

Name	N_a	N_n	N_{obj}	iter	func	CB	CG	N_{act}	T_{fun}	T_{total}
TOIN2	26	10	3	16	25	3	87	0	0.00	0.02
ZANGWL1	26	10	3	15	27	3	71	0	0.00	0.02
CRGLVY	26	10	4	16	20	5	81	2	0.00	0.03
HIMM32	26	10	4	23	45	7	67	9	0.00	0.05
KOWOSB1	26	10	4	16	41	4	93	0	0.01	0.04
PEN2GM6	26	10	4	88	222	4	631	0	0.00	0.09
POWQUD8A	26	10	4	10	64	0	25	0	0.00	0.03
PWSING4	26	10	4	23	32	4	149	2	0.00	0.02
SARSEB	26	10	4	18	23	4	103	2	0.00	0.02
TOIN4	26	10	4	14	23	4	72	0	0.00	0.02
WOODS	26	10	4	25	32	5	126	2	0.00	0.03
ARGQDN50	37	12	5	15	21	4	43	11	0.00	0.02
ARGQO10	37	12	5	21	61	5	151	0	0.00	0.04
BRYTRI2	37	12	5	22	34	5	164	2	0.00	0.03
OSBRN1	37	12	5	35	59	10	349	1	0.00	0.06
BIGGS6	37	12	6	26	29	5	172	4	0.00	0.04
WATSON6	37	12	6	18	21	6	155	3	0.01	0.03
CHEBYQ8	50	14	8	65	104	20	597	1	0.02	0.12
FRANK8	50	14	8	16	22	12	104	5	0.00	0.03
MSQRTB9	50	14	9	16	22	12	119	6	0.00	0.03
ARTRIG10	65	16	10	13	37	8	150	0	0.00	0.04
BRWNAL10	65	16	10	17	19	13	85	15	0.00	0.05
BRYBND	65	16	10	24	36	9	288	3	0.01	0.06
CHEBYQ10	65	16	10	51	81	14	547	6	0.00	0.13
CHNRSH10	65	16	10	16	22	7	149	7	0.00	0.03
CRGLVY10	65	16	10	18	23	12	139	10	0.00	0.04
DIXON	65	16	10	17	23	14	217	9	0.00	0.03
ENGVL1B2	65	16	10	21	33	13	290	0	0.00	0.05
EXTRAR10	65	16	10	24	41	17	275	6	0.00	0.07
HILBR10A	65	16	10	15	22	7	154	16	0.00	0.03
MANCIN10	65	16	10	19	19	8	125	4	0.03	0.07
MORCIN10	65	16	10	14	20	8	132	2	0.00	0.04
MOREBV10	65	16	10	14	26	8	200	0	0.00	0.04
NONDIA10	65	16	10	16	66	9	120	0	0.00	0.04

Table 4.6: Computational results for DSP set 3

Name	N_a	N_n	N_{obj}	iter	func	CB	CG	N_{act}	T_{fun}	T_{total}
PEN1GM6	65	16	10	18	59	8	168	0	0.00	0.04
POWER10	65	16	10	19	39	8	199	2	0.00	0.05
SHNRSN10	65	16	10	19	24	8	159	6	0.01	0.04
TDQ10	65	16	10	20	29	13	114	2	0.00	0.03
TRIDIA10	65	16	10	19	49	8	256	0	0.00	0.04
VARDIM	101	20	10	34	65	12	196	19	0.00	0.10
OSBRN2	65	16	11	23	30	9	274	4	0.19	0.24
FRANK12	65	16	12	17	25	14	205	6	0.01	0.05
HILBRT12	65	16	12	15	22	7	151	19	0.00	0.03
WATSON12	65	16	12	20	22	8	190	9	0.02	0.04
DIX7DGA	82	18	15	13	24	9	111	5	0.00	0.04
XTX16	82	18	16	17	35	9	109	0	0.00	0.04
MOREBV18	101	20	18	16	25	10	399	1	0.00	0.06
BRYTRI6	101	20	20	18	23	9	262	12	0.00	0.05
NONDIA20	101	20	20	15	60	10	64	0	0.00	0.04
QUARTC	122	22	25	32	40	12	217	53	0.00	0.09
NMSURF64	170	26	36	22	35	15	356	30	0.01	0.11
SNMSUR64	170	26	36	25	48	15	408	30	0.01	0.15
MSQRTB49	226	30	49	35	47	30	345	78	0.15	0.43
EXTRAR50	257	32	50	27	45	34	346	14	0.00	0.19
FRDRTHB3	257	32	50	94	107	97	573	88	0.04	0.69
GENT50A	257	32	50	22	59	16	357	0	0.01	0.20
MANCIN50	290	34	50	100	646	37	1659	76	14.86	15.92
PEN1LN1	257	32	50	17	58	16	181	0	0.00	0.15
PEN2GM1	257	32	50	26	37	31	329	91	0.02	0.21
PENL3GM3	257	32	50	138	189	114	566	112	0.22	1.25
POWBSC50	257	32	50	61	136	37	2434	0	0.02	0.93
SCHMVT50	257	32	50	27	50	31	2185	15	0.05	0.54
PSPTOINT	257	32	50	18	22	21	207	80	0.05	0.14
TRIGT50	257	32	50	27	34	33	317	34	0.31	0.59

Table 4.7: Computational results for DSP set 4

Name	N_a	N_n	N_{obj}	iter	func	CB	CG	N_{act}	T_{fun}	T_{total}
BROY7D	290	34	60	19	24	27	1041	31	0.02	0.37
PWSING60	290	34	60	27	49	22	1464	17	0.03	0.42
POWER75	362	38	75	20	26	18	215	15	0.01	0.22
WOODS80	362	38	80	20	23	33	275	54	0.02	0.21
BRWNL100	442	42	100	17	19	20	130	6	0.06	0.24
BRYBND18	442	42	100	27	33	26	986	60	0.27	0.76
ENGVL1B4	442	42	100	14	20	34	238	6	0.03	0.19
EXTRA100	442	42	100	26	50	31	521	5	0.03	0.34
FRDRTHB4	442	42	100	169	175	179	668	164	0.36	2.19
PEN1LN2	442	42	100	26	53	22	290	0	0.06	0.38
PEN2GM2	442	42	100	31	52	29	397	132	0.09	0.53
PENL3GM4	442	42	100	188	307	203	1782	185	1.06	3.96
PWSIN100	442	42	100	27	47	27	1419	29	0.04	0.59
TRLN100	442	42	100	23	43	36	1413	8	0.01	0.55
TRIGT100	442	42	100	31	41	26	406	69	2.78	3.16
NMSUR484	1682	82	400	26	49	62	865	355	2.35	3.70
SNMSR484	1682	82	400	26	49	62	866	355	2.28	3.64
CRGLY500	2117	92	500	17	23	56	174	258	1.23	1.86
GENT500A	2117	92	500	20	64	46	437	0	0.94	2.11
NONDI500	2117	92	500	18	28	46	121	6	1.19	1.76
POWBS500	2117	92	500	90	146	78	30553	6	7.49	47.52
TDQ500	2117	92	500	16	24	81	128	75	0.67	1.31
BRYTRI10	2305	96	600	122	135	62	7386	520	11.61	26.84
MORBV998	4226	130	998	16	40	65	5892	0	4.39	29.81
CRGL1000	4226	130	1000	19	33	104	224	542	5.53	7.78
ENGVL1B6	4226	130	1000	16	26	87	428	41	2.98	5.81
EXTR1000	4226	130	1000	23	42	114	339	540	4.69	7.56
FRDRTHB7	4226	130	1000	93	118	370	917	1736	26.72	57.77
GENT1000	4226	130	1000	21	67	65	362	0	4.05	7.02
NOND1000	4226	130	1000	19	32	89	145	40	5.11	6.95
PEN1LN3	4226	130	1000	44	108	67	360	0	11.75	34.45
POWB1000	4226	130	1000	164	308	154	164096	40	56.29	739.94
PWSI1000	4226	130	1000	23	42	85	2098	256	4.81	14.77
TDQ1000	4226	130	1000	22	52	113	241	25	3.83	6.30
TRLN1000	4226	130	1000	21	32	113	1891	64	4.18	13.18

Table 4.8: Computational results for NFP set 1

Name	N_a	N_n	N_{obj}	iter	func	CB	CG	N_{act}	T_{fun}	T_{total}
ARGAUS	17	10	3	20	20	4	65	5	0	0.05
ARGQDN50	17	10	5	18	21	4	66	3	0	0.03
ARGQDZ10	17	10	3	19	19	5	64	5	0	0.02
ARGQO10	17	10	5	20	20	5	50	8	0	0.02
ARTRIG10	17	10	10	22	22	4	80	4	0	0.03
AVRIEL3	17	10	2	15	15	3	60	1	0	0.03
BARD70	17	10	3	17	17	4	52	5	0.01	0.03
BEAL58KO	17	10	2	24	24	7	79	4	0	0.01
BIGGS6	17	10	6	17	18	6	68	4	0	0.02
BOOTH	17	10	2	16	19	3	62	1	0	0.03
BOX662HL	17	10	2	19	22	5	76	1	0	0.02
BRKMCC	17	10	2	18	20	4	72	1	0	0.03
BROWNB	17	10	2	24	24	6	56	5	0	0.03
BROWND	17	10	4	21	25	6	71	2	0	0.02
BRWNAL10	17	10	10	29	56	6	107	6	0	0.02
BRYBND	17	10	10	43	80	13	149	4	0	0.04
BRYTRI2	17	10	5	35	83	4	139	2	0	0.03
CHNRSH10	17	10	10	26	49	5	104	3	0.01	0.03
CLIFF	17	10	2	15	31	4	59	1	0	0.03
CLUSTR	17	10	2	29	31	10	88	1	0	0.07
DIXON	17	10	10	17	19	4	68	1	0	0.02
ENGVL1A	17	10	2	16	16	3	61	1	0	0.04
ENGVL1B2	17	10	10	24	35	4	96	3	0	0.03
ENGVL2	17	10	3	23	141	3	88	0	0.01	0.05
EXTRAR10	17	10	10	32	77	7	128	4	0	0.03
FRANK8	17	10	8	17	26	5	68	1	0	0.02
FRDRTH	17	10	2	17	17	4	65	1	0	0.01
GENT2B	17	10	2	20	21	5	80	1	0	0.01
GOTTFR	17	10	2	25	30	8	85	1	0	0.04
HILBR10A	17	10	10	17	17	4	58	8	0	0.02
HIMLN3	17	10	2	17	17	5	68	1	0	0.02
HIMM1	17	10	2	15	15	4	60	0	0	0.02
HIMM25	17	10	2	16	16	3	63	1	0	0.01

Table 4.9: Computational results for NFP set 2

Name	N_a	N_n	N_{obj}	iter	func	CB	CG	N_{act}	T_{fun}	T_{total}
HIMM29	17	10	2	32	35	7	64	5	0	0.03
HIMM30	17	10	3	22	27	5	78	1	0	0.04
HIMM32	17	10	4	46	145	8	167	2	0.02	0.06
HIMM33A	17	10	2	12	17	4	48	0	0	0.04
HYPCIR	17	10	2	15	15	3	55	1	0	0.03
KOWOSB1	17	10	4	20	20	5	69	7	0.01	0.04
MANCIN10	17	10	10	23	24	5	90	3	0.04	0.06
MEYER	17	10	3	22	22	6	40	5	0.01	0.02
MORCIN10	17	10	10	17	17	3	68	3	0	0.02
MOREBV10	17	10	10	25	39	3	100	3	0	0.04
MSQRTB9	17	10	9	27	42	5	107	2	0	0.03
NONDIA10	17	10	10	25	25	5	96	4	0	0.04
OSBRN1	17	10	5	17	17	5	62	4	0.01	0.04
PEN1GM6	17	10	10	17	17	3	68	3	0	0.02
PEN2GM6	17	10	4	30	47	4	106	2	0.01	0.03
POWBSC	17	10	2	90	244	18	187	5	0	0.05
POWER10	17	10	10	19	19	5	73	3	0	0.01
POWQUD8A	17	10	4	10	20	2	40	0	0	0.01
POWSSQ	17	10	2	20	27	5	74	5	0	0.02
PWSING4	17	10	4	22	25	7	82	5	0	0.03
RECIPE	17	10	3	15	19	3	57	1	0	0.03
ROSENB	17	10	2	21	23	7	84	1	0	0.02
SCHMVT	17	10	3	50	99	6	128	1	0	0.03
SISSER	17	10	2	16	16	3	60	1	0	0.02
TDQ10	17	10	10	20	24	3	80	4	0	0.04
TOIN2	17	10	3	17	18	3	66	1	0	0.03
TOIN4	17	10	4	14	14	4	56	1	0	0.03
TRIDIA10	17	10	10	16	16	3	64	3	0	0.02
VARDIM	17	10	10	33	67	5	114	9	0	0.02
WATSON6	17	10	6	24	24	12	55	6	0	0.03
WOODS	17	10	4	20	20	5	80	1	0	0.02
XTX2	17	10	2	20	22	3	80	1	0	0.02
ZANGWL1	17	10	3	18	19	3	72	1	0	0.02

Table 4.10: Computational results for NFP set 3

Name	N_a	N_n	N_{obj}	iter	func	CB	CG	N_{act}	T_{fun}	T_{total}
BRYTRI6	42	20	20	21	19	16	149	15	0.01	0.04
CRGLVY	42	20	4	17	44	14	130	0	0	0.03
CRGLVY10	42	20	10	22	20	14	192	2	0	0.04
DIX7DGA	42	20	15	63	223	19	610	11	0.02	0.11
FRANK12	42	20	12	23	26	15	215	6	0	0.04
HILBRT12	42	20	12	20	18	18	193	19	0	0.04
HIMM27	42	20	2	14	63	11	101	0	0	0.03
HIMM28	42	20	2	16	40	12	140	0	0	0.03
MOREBV18	42	20	18	20	18	16	152	17	0	0.03
NONDIA20	42	20	20	30	28	23	125	17	0	0.04
OSBRN2	42	20	11	32	48	15	307	4	0.25	0.31
WATSON12	42	20	12	28	26	17	194	19	0.02	0.06
XTX16	42	20	16	19	17	14	172	6	0	0.03
EXTRAR50	84	40	50	31	41	29	526	26	0.02	0.11
FRDRTHB3	84	40	50	63	93	67	310	30	0.05	0.17
GENT50A	84	40	50	25	26	25	281	21	0	0.08
GULFSH2	84	40	3	27	39	10	310	3	0.03	0.08
HELIX	84	40	3	22	30	9	277	3	0	0.07
HILBR10A	84	40	10	24	30	20	435	14	0	0.08
MSQRTB49	84	40	49	29	31	33	468	29	0.17	0.25
NMSURF64	84	40	36	20	18	32	377	25	0.01	0.09
PEN1LN1	84	40	50	22	21	19	301	18	0	0.07
PEN2GM1	84	40	50	22	20	22	267	21	0.01	0.10
POWER75	84	40	75	33	33	25	278	10	0.02	0.18
PSPTOINT	84	40	50	26	36	23	463	27	0.01	0.12
PWSING60	84	40	60	26	34	22	454	24	0.01	0.09
QUARTC	84	40	25	36	100	28	376	19	0.01	0.12
SCHMVT50	84	40	50	46	65	39	638	12	0.04	0.20
SNMSUR64	84	40	36	21	19	33	400	23	0.01	0.07
TRIGT50	84	40	50	38	58	29	412	14	0.42	0.59
VAROSBG1	84	40	50	21	22	19	351	17	0.01	0.07
WOODS80	84	40	80	23	21	24	271	26	0.04	0.14
BROY7D	102	40	60	26	28	39	397	34	0	0.10

Table 4.11: Computational results for NFP set 4

Name	N_a	N_n	N_{obj}	iter	func	CB	CG	N_{act}	T_{fun}	T_{total}
BRWNL100	1010	200	100	32	28	241	3634	17	0.13	3.15
BRYBND18	1010	200	100	37	36	239	2507	27	0.30	2.24
BRYTRI10	1010	200	600	51	74	333	11258	306	4.07	11.47
ENGVL1B4	1010	200	100	34	36	231	5911	8	0.02	3.65
ENGVL1B6	1010	200	1000	37	36	292	2169	397	5.59	7.42
EXTR1000	1010	200	1000	89	324	363	25557	461	16.77	32.02
EXTRA100	1010	200	100	43	61	237	8616	10	0.02	5.18
FRDRTHB4	1010	200	100	93	103	263	1602	82	0.23	2.70
FRDRTHB7	1010	200	1000	152	165	652	3309	694	39.43	44.06
GENT1000	1010	200	1000	48	49	371	2125	425	7.61	9.65
GENT500A	1010	200	500	57	138	314	6214	210	2.31	6.77
MORBV998	1010	200	998	30	23	262	3603	382	6.38	9.03
NMSUR484	1010	200	400	42	48	302	9657	152	3.16	9.16
NOND1000	1010	200	1000	55	90	385	2199	475	13.09	15.33
NONDI500	1010	200	500	85	160	342	4221	235	5.07	9.27
PEN1LN2	1010	200	100	26	19	231	1646	8	0.03	1.66
PEN1LN3	1010	200	1000	37	33	237	1557	263	7.97	63.13
PEN2GM2	1010	200	100	42	51	247	1938	36	0.11	2.24
PENL3GM3	1010	200	50	67	94	240	1366	54	0.07	2.14
PENL3GM4	1010	200	100	199	302	281	5168	109	1.05	8.78
POWBSC50	1010	200	50	135	183	301	9080	31	0.05	6.98
PWSI1000	1010	200	1000	41	52	322	4078	514	7.10	10.04
PWSIN100	1010	200	100	34	36	237	4069	34	0.05	2.69
SNMSR484	1010	200	400	40	43	302	9187	152	2.94	8.61
TDQ1000	1010	200	1000	32	30	237	1655	263	4.27	5.62
TDQ500	1010	200	500	31	29	304	2913	158	1.06	3.28
TRIGT100	1010	200	100	72	75	272	1551	33	5.73	8.23
TRLN100	1010	200	100	30	26	234	3308	7	0.04	2.24
TRLN1000	1010	200	1000	43	59	264	2878	325	7.22	9.46
VAROSBG2	1010	200	100	31	27	234	3671	16	0.09	2.49

Table 4.12: Computational results for large DSP

Name	N_a	N_n	N_{obj}	iter	func	CB	CG	N_{act}	T_{fun}	T_{total}
CRGL1000	10001	200	1000	18	27	109	176	502	5.04	10.53
ENGVL1B6	10001	200	1000	13	24	99	159	1	2.40	6.22
EXTR1000	10001	200	1000	19	36	99	123	500	3.90	8.98
GENT1000	10001	200	1000	20	92	99	338	0	4.26	12.19
NOND1000	10001	200	1000	24	123	100	110	1	6.65	13.74
PEN1LN3	10001	200	1000	61	152	97	387	0	15.22	138.33
PWSI1000	10001	200	1000	31	78	99	727	250	7.12	21.05
TDQ1000	10001	200	1000	20	67	188	144	0	3.48	9.17
TRLN1000	10001	200	1000	27	52	99	1530	81	4.99	27.82
CRGL1000	108901	660	1000	23	49	496	374	822	7.43	151.88
ENGVL1B6	108901	660	1000	20	42	16	336	321	4.61	119.38
EXTR1000	108901	660	1000	25	67	342	396	815	6.04	155.92
GENT1000	108901	660	1000	19	60	331	220	1	4.71	93.56
NOND1000	108901	660	1000	26	76	339	200	321	7.99	123.35
PEN1LN3	108901	660	1000	39	132	18	437	0	11.40	390.78
PWSI1000	108901	660	1000	24	65	413	1101	500	6.41	308.67
TDQ1000	108901	660	1000	28	88	330	270	10	6.05	134.15
TRLN1000	108901	660	1000	26	76	330	2092	319	5.83	507.32
CRGL1000	1000001	2000	1000	18	37	999	107	502	10.28	645.81
ENGVL1B6	1000001	2000	1000	11	25	1	40	1	5.46	301.89
EXTR1000	1000001	2000	1000	19	43	999	73	500	9.74	661.32
GENT1000	1000001	2000	1000	15	39	999	59	1	8.22	407.21
NOND1000	1000001	2000	1000	22	87	1	72	1	16.90	1018.97
PEN1LN3	1000001	2000	1000	16	90	1	20	0	14.98	747.71
PWSI1000	1000001	2000	1000	20	59	999	192	500	12.12	822.39
TDQ1000	1000001	2000	1000	22	91	999	82	0	15.78	1080.23
TRLN1000	1000001	2000	1000	27	63	999	3861	987	13.56	8192.04

Table 4.13: CG iterations comparison

N_a	17	26	37	65	101	257
$\frac{CG}{iter \times N_a}$	2.06E-01	1.89E-01	1.78E-01	1.56E-01	1.23E-01	9.52E-02
N_a	442	2117	4226	10001	108901	1000001
$\frac{CG}{iter \times N_a}$	4.80E-02	3.65E-02	3.27E-02	7.51E-03	2.20E-04	2.18E-05

4.2.3 Comparisons

In this section, we compare NPDNET to one of the most efficient general-purpose large-scale nonlinear optimization solvers, SNOPT [38], which is a Fortran 77 package for large-scale constrained optimization problems. The version of SNOPT that we compare with is version 5.3 and it runs with default parameter settings. We compare NPDNET and SNOPT on the DSP problem with $m = 33$ and $m = 46$ and 9 objective functions, including CRGL1000, ENGVL1B6, EXTR1000, GENT1000, NOND1000, PEN1LN3, PWSI1000 and TDQ1000. Table 4.14 and 4.15 summarize the runs for these DSP test sets on both Fortran codes, NPDNET and SNOPT. We make the following remarks regarding the computational results:

- On most of the problems except PWSI1000 and TRLN1000, NPDNET and SNOPT take a similar number of iterations and function evaluations. However, the CPU time taken by SNOPT on these problems is about 20 times more for $N_a = 1090$ and about 100 times more for $N_a = 2017$ than by NPDNET.

- Over all the test problems, NPDNET runs 28 times faster on the problems with $N_a = 1090$ and 220 times faster on the problems with $N_a = 2017$. Consequently, we expect to gain more as the size of the problem grows.

- Since NPDNET needs to evaluate the Hessian matrix of the objective function but SNOPT does not, NPDNET spends more time on the function evaluations

Table 4.14: Performance for NPDNET and SNOPT

		NPDNET			SNOPT		
Name	N_a	iter	func	T_{total}	iter	func	T_{total}
CRGL1000	1090	24	35	7.36	37	40	15.45
ENGVL1B6	1090	16	24	3.54	49	62	102.17
EXTR1000	1090	23	38	5.24	58	66	62.29
GENT1000	1090	26	51	5.50	71	115	284.47
NOND1000	1090	27	55	7.99	81	86	141.65
PEN1LN3	1090	17	39	12.79	30	38	123.11
PWSI1000	1090	25	44	7.92	407	471	296.87
TDQ1000	1090	17	23	3.22	35	41	124.60
TRLN1000	1090	19	23	4.78	293	340	400.38
Average	1090	21.55	36.88	6.48	117.88	139.88	172.33

than SNOPT.

Table 4.15: Performance for NPDNET and SNOPT

		NPDNET			SNOPT		
Name	N_a	iter	func	T_{total}	iter	func	T_{total}
CRGL1000	2017	22	36	7.24	33	37	304.04
ENGVL1B6	2017	13	19	3.09	23	26	236.24
EXTR1000	2017	27	57	6.76	37	40	645.04
GENT1000	2017	26	70	6.31	43	46	1348.25
NOND1000	2017	19	26	5.77	54	61	661.13
PEN1LN3	2017	19	54	14.07	30	34	695.61
PWSI1000	2017	25	50	8.96	240	261	3101.18
TDQ1000	2017	20	29	4.18	27	31	859.42
TRLN1000	2017	22	43	8.08	418	473	6365.36
Average	2017	21.44	42.67	7.16	100.56	112.11	1579.59

Chapter 5

Summary and Conclusions

The code described in this paper can be further improved. For instance, great benefits were obtained from parallel implementation of the conjugate-gradient code in a parallel implementation of a dual affine scaling network flow algorithm [91]. We expect NPDNET to benefit similarly from such computer architectures.

We have shown that our linesearch algorithm performs very well for the primal-dual algorithm for network optimization. We expect that it will also prove useful when used within other interior-point algorithms, for example, box-constrained nonlinear optimization, general linearly constrained nonlinear optimization with box constraints, and linear inequality constrained optimization.

For generalized network problems, the constraint structure is the same as for pure network problems but some components of the node-arc incidence matrix A are neither 1 nor -1. To solve the generalized network problem, the procedure that must be modified is the algorithm for computing $Z \times v$ and $Z^T \times v$. The spanning tree variable reduction null-space basis can easily extend to generalized network problems by using the flow multiplier $-f_{ij}$ rather than -1 in the algorithm for computing multiplication $Z \times v$ and $Z^T \times v$.

As discussed in [85], [74] and [15], some of the real-world applications have an embedded network structure of the form

$$\text{Minimize} \qquad F(x)$$

$$\begin{aligned} Ax &= b \\ Cx &= d \\ l \leq x &\leq u \ , \end{aligned} \tag{5.1}$$

where $Ax = b$ represents network constraints and $Cx = d$ ($C \in R^{r \times n}$) represents general linear constraints. We can also use our algorithm with a different variable reduction basis. Suppose that $P_{row} A P_{col} = [B\ N]$, where B is a basis (rooted spanning tree) of A and the associated partition of C is $CP_{col} = [C_1\ C_2]$. We can apply LU decomposition with column interchanges to $D = C_2 - C_1 B^{-1} N$ and get

$$D\tilde{P}_{col} = L\,[U_1\ U_2] \ , \tag{5.2}$$

where $U_1 \in R^{r \times r}$ is an upper-triangular matrix. The variable reduction basis for the constraint matrix $[A^T\ C^T]^T$ can be obtained by

$$Z = P_{col} \begin{bmatrix} -B^{-1}N \\ I \end{bmatrix} \tilde{P}_{col} \begin{bmatrix} -U_1^{-1}U_2 \\ I \end{bmatrix} . \tag{5.3}$$

In this class of problems, the number of the general linear constraints r (the number of rows in C) is usually small comparing to the number of network constraints. Therefore, the computation of the LU decomposition and the multiplication $Z \times v$ and $Z^T \times v$ will be not too computationally intensive.

Finally, we hope this thesis will lead to further research both on the implementation of null-space truncated methods for other specially structured mathematical programming problems, as well as on the theoretical analysis of null-space truncated variants of other interior-point techniques.

Bibliography

[1] D. P. Ahlfeld, R. S. Dembo, J. M. Mulvey, and S. A. Zenios. Nonlinear programming on generalized networks. *ACM Trans. Math. Software*, 13:350–367, 1987.

[2] A. Armacost and S. Mehrotra. Computational comparison of the network simplex method with the affine scaling method. *Opsearch*, 28:26–43, 1991.

[3] L. Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific J. of Mathematics*, 16:1–3, 1966.

[4] I. Baybars and R. H. Edahl. A heuristic method for facility planning in telecommunications networks with multiple alternate routes. *Naval Research Logistics Quarterly*, 35(4):503–528, 1988.

[5] P. Beck, L. Lasdon, and M. Engquist. A reduced gradient algorithm for nonlinear networks. *ACM Trans. Math. Software*, 9:57–70, 1983.

[6] D. P. Bertsekas. *Nonlinear Optimization.* Athena Scientific, Belmont, MA, 1982.

[7] D. P. Bertsekas. *Linear Network Optimization: Algorithm and Codes.* The MIT Press, London, England, 1991.

[8] E. G. Boman. *Infeasibility and Negative Curvature in Optimization.* PhD thesis, The program in Scientific Computing and Computational Mathematics, Stanford University, 1999.

[9] M. G. Breitfeld and D. Shanno. Computational experience with penalty/barrier methods for nonlinear programming. *Annals of Operations Research*, 62:439–464, 1996.

[10] G. G. Brown and R. D. McBride. Solving generalized networks. *Management Science*, 30:1497–1523, 1984.

[11] V. P. Bulatov. The immersion method for the global minimization of functions on the convex polyhedra. *International Symposium on Engineering, Mathematics, and Applications, Beijing, China*, pages 335–338, 1988.

[12] R. H. Byrd, M. B. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. Technical report, 1999.

[13] R. H. Byrd and J. Nocedal. A tool for the analysis of quasi-Newton methods with application to unconstrained optimization. *SIAM J. Numer. Anal.*, 26:727–739, 1989.

[14] R. H. Byrd and R. B. Schnabel. Continuity of the null space basis and constrained optimization. *Math. Prog.*, 35:32–41, 1986.

[15] S. Chen and R. Saigal. A primal algorithm for solving a capacitated network flow problem with additional linear constraints. *Networks*, 7:59–79, 1977.

[16] F. Chin. Algorithm for updating minimal spanning tree. *J. of Computer and System Sciences*, 16:333–344, 1978.

[17] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A note on using alternative second-order models for the subproblems arising in barrier function methods for minimization. *Numer. Math.*, 68:17–33, 1994.

[18] T. G. Crainic, M. Florian, and J. Leal. A model for the strategic planning of national freight transportation by rail. *Transportation Science*, 24(1):1–24, 1990.

[19] G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, New Jersey, 1963.

[20] R. S. Dembo. The performance of nlpnet, a large-scale nonlinear optimizer. *Math. Program. Study*, 26:245–248, 1986.

[21] R. S. Dembo. A primal truncated Newton algorithm with application to large-scale nonlinear network optimization. *Math. Program. Study*, 31:43–72, 1987.

[22] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.

[23] R. S. Dembo, J. M. Mulvey, and S. A. Zenios. Large-scale nonlinear network models and their application. *OR Practice*, 37(3):353–372, 1989.

[24] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

[25] J. E. Dennis, Jr. and R. B. Schnabel. A view of unconstrained optimization. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Handbooks in Operations Research and Management Science, volume 1. Optimization*, chapter 1, pages 1–72. North Holland, Amsterdam, New York, Oxford and Tokyo, 1989.

[26] D. Erlenkotter. Two producing areas—dynamic programming solutions. In A.S. Manne, editor, *Investment for Capacity Expansion: Size, Location and Time Phasing*, pages 210–227. MIT Press, Cambridge, MA, 1967.

[27] L. F. Escudero, J. L. de la Fuente, C. Garcia, and F. J. Prieto. Hydropower generation management under uncertainty via scenario analysis and parallel computation. *IEEE Transactions on Power Systems*, 11(2):683–689, 1996.

[28] A. V. Fiacco. Barrier methods for nonlinear programming. In A. Holzman, editor, *Operations Research Support Methodology*, pages 377–440, New York, 1979. Marcel Dekker.

[29] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* John Wiley & Sons, Inc., New York, 1968.

[30] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* Classics in Applied Mathematics. SIAM, Philadelphia, 1990.

[31] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, Chichester, second edition, 1987.

[32] R. Fletcher and A. P. McCann. Acceleration techniques for nonlinear programming. In *Optimization.* Academic Press, New York, 1969.

[33] M. Florian. An introduction to network models used in transportation planning. In *Transportation Planning Models.* Elsevier Publishers, New York, 1984.

[34] M. Florian. Nonlinear cost network models in transportation analysis. *Mathematical Programming Study 26*, pages 167–196, 1986.

[35] C. O. Fong and M. R. Rao. Capacity expansion with two producing regions and concave costs. *Management Science*, 22(3):331–339, 1975.

[36] A. Forsgren and W. Murray. Newton methods for large-scale linear equality-constrained minimization. *SIAM J. Matrix Anal. Appl.*, 14:560–587, 1993.

[37] P. E. Gill and W. Murray. Newton-type methods for unconstrained and linearly constrained optimization. *Math. Prog.*, 7:311–350, 1974.

[38] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. Numerical Analysis Report 97-2, Department of Mathematics, University of California, San Diego, La Jolla, CA, 1997.

[39] P. E. Gill, W. Murray, M. A. Saunders, J. Tomlin, and M. H. Wright. On projected Newton methods for linear programming and an equivalence to Karmarkar's projective method. *Math. Prog.*, 36:183–209, 1986.

[40] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A note on a sufficient-decrease criterion for a non-derivative step-length procedure. *Math. Prog.*, 23:349–352, 1982.

[41] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Software and its relationship to methods. In P. T. Boggs, R. H. Byrd, and R. B. Schnabel, editors, *Numerical Optimization 1984*, pages 139–159. SIAM, Philadelphia, 1985.

[42] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Constrained nonlinear programming. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Handbooks in Operations Research and Management Science, volume 1. Optimization*, chapter 3, pages 171–210. North Holland, Amsterdam, 1989.

[43] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, London and New York, 1981.

[44] F. Glover, J. Hultz, D. Klingman, and J. Stutz. Generalized network: A fundamental computer based planning tool. *Management Science*, 24:1209–1220, 1978.

[45] D. Goldfarb and M. J. Todd. Linear programming. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Handbooks in Operations Research and Management Science, volume 1. Optimization*, chapter 2, pages 73–170. North Holland, Amsterdam, 1989.

[46] A. A. Goldstein. On steepest descent. *SIAM J. on Control*, 3:147–151, 1965.

[47] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, Maryland, 1983.

[48] C. C. Gonzaga. Path-following methods for linear programming. *SIAM Review*, 34:167–224, 1992.

[49] B. G .Gorenstin and N. M. Campodonico. Power system expansion planning under uncertainty. *IEEE Transactions on Power Systems*, 8(1):129–136, 1993.

[50] G. Guisewite. Network problems. In R. Horst and P.M. Pardalos, editors, *Handbook of Global Optimization*. Kluwer Academic Publisher, 1995.

[51] R. W. Hall. Graphical interpretation of the transportation problem. *Transportation Science*, 23(1):34–75, 1989.

[52] F. Jarre and M. A. Saunders. A practical interior-point method for convex programming. *SIAM J. Optimization*, 5:149–171, 1995.

[53] B. Yaged Jr. Minimum cost routing for static network models. *Networks*, 1:139–172, 1971.

[54] J. Kaliski and Y. Ye. A decomposition variant of potential reduction algorithm for linear programming. *Management Science*, 39:757–776, 1993.

[55] P. V. Kamesam and R. R. Meyer. Multipoint methods for separable nonlinear network. *Math. Program. Studt*, 22:185–205, 1984.

[56] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[57] N. Karmarkar and K. G. Ramakrishnan. Computational results of an interior point algorithm for large scale linear programming. *Math. Prog.*, 52:555–586, 1991.

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

[59] J. L. Kennington and R. V. Helgason. *Algorithms for Network Programming.* John Wiley & Sons, New York and Toronto, 1980.

[60] K. Kim and L. Nazareth. Implementation of a primal null-space affine scaling method and its extensions. Technical Report 92-1, Department of Pure and Applied Mathematics, Washington State University, 1992.

[61] J. G. Klincewicz. Solving a freight transport problem using facility location techniques. *Operations Research*, 38(1):99–109, 1989.

[62] D. Klingman, A. Napier, and J. Stutz. Netgen: A program for generating large scale capacitated assignment, transportation, and minimum-cost flow network problems. *Management Science*, 20:814–820, 1974.

[63] D. Klingman, P. H. Randolph, and S. W. Fuller. A cotton ginning problem. *Operations Research*, 24(4):700–717, 1976.

[64] A. V. Knyazev. Preconditioned eigensolvers—an oxymoron. *Electronic Transcations on Numerical Analysis*, 7:104–123, 1998.

[65] H. Konno. Minimum concave series production system with deterministic demands: A backlogging case. *J. of the Operations Research Society of Japan*, 16:246–253, 1973.

[66] H. Konno. Minimum concave series production system : Multi-echelon model. *Math. Prog.*, 41:185–193, 1988.

[67] J. Kruskal. On the shortest spanning tree of graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[68] H. W. Kuhn and W. J. Baumol. An approximate algorithm for the fixed-charge transportation problem. *Naval Research Logistics Quarterly*, 9(1):1–16, 1962.

[69] M. Lalee, J. Nocedal, and T. Plantenga. On the implementation of an algorithm for large-scale equality constrained optimization. *SIAM J. Optimization*, 8:682–706, 1998.

[70] L. J. LeBlanc. Global solutions for a nonconvex, nonconcave rail network model. *Mathematical Science*, 23:131–139, 1978.

[71] D. G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley Publishing Company, Reading, MA, second edition, 1984.

[72] T. L. Magnanti and R. T. Wong. Network design and transportation planning: Models and algorithms. *Transportation Science*, 18(1):1–55, 1984.

[73] A. S. Manne and A. F. Veinott Jr. Optimal plant size with arbitray increasing time paths of demand. In A.S. Manne, editor, *Investment for Capacity Expansion: Size, Location and Time Phasing*, pages 178–190. MIT Press, Cambridge, MA, 1967.

[74] R. D. McBride. Solving embedded generalized network problems. *European J. of Operational Research*, 21:82–92, 1985.

[75] J. M. Mulvey. Testing of a large scale network optimization program. *Math. Prog.*, 15:291–315, 1978.

[76] J. M. Mulvey and S. A. Zenios. Solving large scale generalized networks. *J. of Information and Optimization Sciences*, 6:95–112, 1985.

[77] W. Murray and M. H. Wright. Line search procedures for the logarithmic barrier function. *SIAM J. Optim.*, 4:229–246, 1994.

[78] B. A. Murtagh and M. A. Saunders. Large-scale linearly constrained optimization. *Math. Prog.*, 14:41–72, 1978.

[79] S. G. Nash and A. Sofer. A barrier method for large-scale constrained optimization. *ORSA J. on Computing*, 5:40–53, 1993.

[80] S. G. Nash and A. Sofer. On the complexity of a practical interior-point method. *SIAM J. Optim.*, 8(3):833–849, 1993.

[81] A. Orden. The transshipment problem. *Management Science*, 2(3):276–285, 1956.

[82] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, New York, 1970.

[83] M. V. F. Pereira. Optimal stochastic operations scheduling of large hydroelectric system. *International J. of Electrical Power and Energy Systems*, 11:161–169, 1989.

[84] M. V. F. Pereira and L. M. V. G. Pinto. Stochastic optimization of a hydroelectric system: a decomposition approach. *Water Resource Research*, 21(6), 1985.

[85] M. Pinar and S. A. Zenios. Solving nonlinear programs with embedded network structures. In *Network Optimization Problems: Algorithm, Application and Complexity*, pages 177–202. World Scientific, New Jersey, 1999.

[86] L. F. Portugal, M. G. C. Resende, G. Veiga, and J. J. Judice. A truncated primal-infeasible dual-feasible network interior point method. *Networks*, 35(2):91–108, 2000.

[87] F. A. Potra and Y. Shi. Efficient line search algorithm for unconstrained optimization. *J. of Optimization Theory and Applications*, 85:677–704, 1995.

[88] M. J. D. Powell. Some global convergence properties of a variable metric algorithm for minimization without exact line searches. In R. W. Cottle and C. E. Lemke, editors, *SIAM-AMS Proceedings*, volume IX, Philadelphia, 1976. SIAM Publications.

[89] M. G. C. Resende and G. Veiga. An efficient implementation of a network interior point method. Technical report, 1992.

[90] M. G. C. Resende and G. Veiga. Computing the projection in an interior point algorithm: An experimental comparison. *Investigacion Operativa*, 3:81–92, 1993.

[91] M. G. C. Resende and G. Veiga. An implementation of the dual affine scaling algorithm for minimum cost flow on bipartite uncapacitated networks. *SIAM J. on Optimization*, 3:516–537, 1993.

[92] U. T. Ringertz. Optimal design of nonlinear shell structures. Report FFA TN 91-18, The Aeronautical Research Institute of Sweden, 1991.

[93] T. A. Rottimg and A. Gjelsvik. Stochastic dual programming for seasonal scheduling in the Norwegian power system. *IEEE Transactions on Power Systems*, 7:273–279, 1992.

[94] M. H. Schneider and S. A. Zenios. A comparative study of algorithms for matrix balancing. *Operations Research*, 38(3):439–455, 1990.

[95] A. H. Sherman. On Newton-iterative methods for the solution of system of nonlinear equations. *SIAM J. Numerical Analysis*, 15:755–771, 1978.

[96] P. M. Spira and A. Pan. On finding and updating spanning trees and shortest paths. *SIAM J. Computing and System Sciences*, 4:375–380, 1975.

[97] R. Tarjan. Data structures and network algorithm. *Society for Industrial and Applied Mathematics*, 1983.

[98] R. J. Vanderbei. *Linear Programming*. Kluwer, Dordrecht, the Netherlands, 1996.

[99] P. Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11:226–235, 1968.

[100] M. H. Wright. Interior methods for constrained optimization. In A. Iserles, editor, *Acta Numerica 1992*, pages 341–407. Cambridge University Press, New York, USA, 1992.

[101] M. H. Wright. Why a pure primal Newton barrier step may be infeasible. *SIAM J. Optim.*, 5:1–12, 1995.

[102] S. J. Wright. *Primal-dual interior-point methods*. SIAM, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[103] N. Zadeh. On building minimum cost communication networks. *Networks*, 3:315–331, 1973.

[104] N. Zadeh. On building minimum cost communication networks over time. *Networks*, 4:19–34, 1974.

[105] W. I. Zangwill. A deterministic multi-period productin scheduling model with backlogging. *Management Science*, 13(1):105–119, 1966.

[106] W. I. Zangwill. A backlogging model and a multi-echelon model of a economic lot size production system - a network approach,. *Management Science*, 15(9):506–527, 1969.

[107] S. A. Zenios, A. Drud, and J. M. Mulvey. Balancing large social accounting matrices with nonlinear network programming. *Networks*, 19(5):569–585, 1989.