MULTISTAGE QUADRATIC STOCHASTIC PROGRAMMING

by

KAREN KARMAN LAU

Submitted to The University of New South Wales

for the degree of Doctor of Philosophy.

Submitted September, 1999

Acknowledgements

I would like to express my gratitude and appreciation to my supervisor Dr. R. S. Womersley whose advice and support has been invaluable. Without his guidance, this thesis would not have been possible.

I would like to thank all the staff member of the School of Mathematics for their help and support throughout my studies here. I would especially like to thank Dr. P Blennerhassett, the Director of Postgraduate Studies, his help during these sometimes difficult years is deeply appreciated.

Abstract

Multistage stochastic programming is an important tool in medium to long term planning where there are uncertainties in the data. In this thesis, we consider a special case of multistage stochastic programming in which each subproblem is a convex quadratic program. The results are also applicable if the quadratic objectives are replaced by convex piecewise quadratic functions. Convex piecewise quadratic functions have important application in financial planning problems as they can be used as very flexible risk measures. The stochastic programming problems can be used as multi-period portfolio planning problems tailored to the need of individual investors.

Using techniques from convex analysis and sensitivity analysis, we show that each subproblem of a multistage quadratic stochastic program is a polyhedral piecewise quadratic program with convex Lipschitz objective. The objective of any subproblem is differentiable with Lipschitz gradient if all its descendent problems have unique dual variables, which can be guaranteed if the linear independence constraint qualification is satisfied. Expressions for arbitrary elements of the subdifferential and generalized Hessian at a point can be calculated for quadratic pieces that are active at the point.

Generalized Newton methods with linesearch are proposed for solving multistage quadratic stochastic programs. The algorithms converge globally. If the piecewise quadratic objective is differentiable and strictly convex at the solution, then convergence is also finite.

A generalized Newton algorithm is implemented in Matlab. Numerical experiments have been carried out to demonstrate its effectiveness. The algorithm is

tested on random data with 3, 4 and 5 stages with a maximum of 315 scenarios. The algorithm has also been successfully applied to two sets of test data from a capacity expansion problem and a portfolio management problem. Various strategies have been implemented to improve the efficiency of the proposed algorithm. We experimented with trust region methods with different parameters, using an advanced solution from a smaller version of the original problem and sorting the stochastic right hand sides to encourage faster convergence. The numerical results show that the proposed generalized Newton method is a highly accurate and effective method for multistage quadratic stochastic programs. For problems with the same number of stages, solution times increase linearly with the number of scenarios.

Contents

Notation **x**

1 Introduction **1**

1.1 Random Variables and Modelling Uncertainty 2

1.1.1 Scenario Trees 3

1.2 Chance Constrained and Recourse Programs 4

1.2.1 Chance Constrained Programs 4

1.2.2 Recourse Programs 5

1.3 Modelling Real Life Problems and Applications 7

1.3.1 Stochastic Linear Programs 7

1.3.2 Portfolio Optimization 8

1.3.3 Asymmetric Risk Measures 10

1.3.4 Multistage Financial Planning 14

1.4 Multistage Quadratic Stochastic Programs (MQSPs) 15

1.4.1 Model Formulation 15

1.4.2 Large QP Equivalent 16

1.5 Overview of Thesis 17

2 Survey of Stochastic Programming Algorithms **19**

2.1 Large Scale Linear Programs 20

2.1.1 Simplex Methods 20

2.1.2 Interior Point Methods 22

2.2 Large Scale Quadratic Programs 25

2.2.1 Active Set Methods 26
2.2.2 Interior Point Methods 28
2.3 Convex Nonsmooth Programs 29
2.3.1 Lipschitz Functions, Subgradients and Generalized Hessians . 29
2.3.2 Subgradient Methods 32
2.3.3 Bundle Methods 32
2.3.4 LC^1 optimization 36
2.3.5 Minimizing Lipschitz Functions 38
2.4 Primal Decomposition Methods for Stochastic Programs 40
2.4.1 L-shaped Method 41
2.4.2 Nested Decomposition 47
2.4.3 Regularized Decomposition 49
2.4.4 Stochastic Decomposition and Importance Sampling 51
2.4.5 Piecewise Quadratic Form of the L-shaped Method 52
2.4.6 Two Stage Quadratic Stochastic Program 55
2.5 Dual Decomposition Methods for Stochastic Programs 58
2.5.1 Lagrangian Finite Generation Method 59
2.5.2 Diagonal Quadratic Algorithm 60
2.5.3 Progressive Hedging Algorithm 64
2.6 Stochastic Quasigradient 65
2.7 Stochastic Programs with Continuous Random Variables 66

3 Properties of Multistage Stochastic Quadratic Programs 67
3.1 Sensitivity Analysis 68
3.2 Piecewise Quadratic Programs (PQPs) 73
3.2.1 Piecewise Quadratic Functions 73
3.2.2 Piecewise Quadratic Programs 74
3.3 MQSPs as PQPs 78
3.3.1 Strictly Convex MQSPs 80
3.3.2 Convex Quadratic Programs 89

4 Generalized Newton Methods for MQSPs **96**

4.1 Strictly Convex LC^1 MQSPs 96

4.2 Feasibility Cuts 99

4.3 Modifications for Convex MQSPs 101

5 Convergence Analysis **107**

5.1 Strictly Convex LC^1 MQSPs 107

5.1.1 Global Convergence 108

5.1.2 Finite Convergence 110

5.2 Convex MQSPs 111

6 Implementation Issues and Numerical Results **120**

6.1 Implementation Issues 120

6.1.1 Random Data Generation 120

6.1.2 Linesearch Strategy and Trust Region Method 121

6.1.3 Presolve Strategies and other Speedup Schemes 127

6.1.4 Optimality Condition Verification 130

6.1.5 Degeneracy 131

6.2 Numerical Results 131

6.2.1 Trust Region Radius Updating 132

6.2.2 A Small Numerical Example 136

6.2.3 Random Data with Strictly Convex Subproblems 137

6.2.4 Random Data with Convex Subproblems 148

6.2.5 Stochastic Capacity Expansion Model 157

6.2.6 Portfolio Management Application 161

7 Conclusion **165**

Bibliography **169**

Index **181**

List of Tables

3.1 Solution to example PQP (3.15) 82

3.2 Example of a perturbed convex quadratic program 94

6.1 Size of equivalent deterministic QP (1.14) 133

6.2 Trust region method parameters 133

Numerical results for strictly convex random subproblems:

6.3 3 stage tree 141

6.4 4 stage tree 143

6.5 5 stage tree 145

Numerical results for convex random subproblems:

6.6 3 stage tree 150

6.7 4 stage tree 152

6.8 5 stage tree 154

6.9 Problem sizes of PLTEXP 158

6.10 Numerical results for problem PLTEXP: 2 to 4 stages 159

6.11 Problem sizes of SGPF 163

6.12 Numerical results for problem SGPF: 3 to 6 stages 163

List of Figures

1.1 Scenario tree 4
1.2 Typical efficient frontier 9
1.3 Linear-quadratic risk measure 11

3.1 Example of degenerate piecewise quadratic function 75
3.2 Example of perturbed PQP 81
3.3 Example of a perturbed convex quadratic program 95

6.1 Linesearch example 1 123
6.2 Linesearch example 2 123
6.3 Presolve tree 128
6.4 Cputime vs trust region schemes: Strictly convex random subproblems 134
6.5 Cputime vs trust region schemes: Convex random subproblems 134
6.6 Sparsity pattern of problem PLTEXP 138

Cputime vs number of variables: Strictly convex random subproblems:

6.7 3 stage tree 142
6.8 4 stage tree 144
6.9 5 stage tree 146
6.10 3 to 5 stage trees 147

Cputime vs number of variables: Convex random subproblems:

6.11 3 stage tree 151
6.12 4 stage tree 153
6.13 5 stage tree 155
6.14 3 to 5 stage trees 156

6.15 Cputime vs number of variables: PLTEXP, 2 to 4 stages 160
6.16 Cputime vs number of variables: SGPF, 3 to 6 stages 164

Notation

$\overset{=}{\leq}$	$=$ or $\leq$ as specified for each component of vector
$\|\cdot\|_B$	matrix norm, $\|x\|_B \equiv (x^\mathsf{T} B x)^{\frac{1}{2}}$
A	constraint matrix, usually of size $m \times n$
A_1, A_2	equality and inequality constraints in A respectively
$\bar{A}$	active constraint matrix, $\bar{A} \in \mathbb{R}^{\bar{m} \times n}$
$\mathcal{A}$	index set of active constraints
B	nonsingular basis matrix
$D_t(k)$	set of descendent in stage t of scenario k
E	the expectation operator
G	Hessian matrix of f
$\mathcal{G}$	generalized Hessian of f
I	identity matrix
$J(x_t^{k_t})$	the set of quadratic pieces active at $x_t^{k_t}$
K_t	total number of scenarios at stage t
$L_t^{k_t}$	the set of quadratic pieces encountered in the (t, k_t) subproblem
Q	recourse function, $Q_t(x_t) = E_{\xi_{t+1}} \mathcal{Q}_{t+1}(x_t, \xi_{t+1})$
$\mathcal{Q}$	optimal value function
T	number of stages
V_t	stage t technology matrix, $V_t \in \mathbb{R}^{m_t \times n_{t-1}}$
W_t	stage t recourse matrix, $W_t \in \mathbb{R}^{m_t \times n_t}$
W_{t1}, W_{t2}	the equality and inequality parts of W_t
e	the vector of ones $(1, 1, \ldots, 1)^\mathsf{T}$
f	convex function

g	gradient or any subgradient of f
$\tilde{g}^i$	$\sum_{j \in L^i} \lambda_j^i g_j$, aggregate subgradient
k	scenario index
m, $m_t, \bar{m}$	number of constraints; in stage t problem; number of active constraints.
n, n_t	number of variables; in stage t problem
p_t^k	probability of scenario (t, k_t)
q	a quadratic function
t	stage (or time) index
v^i	sufficient function decrease $v^i \equiv -\left(\|\tilde{g}^i + A^\intercal \mu\|^2_{(B^i)^{-1}} + \tilde{\alpha}^i + (b - Ax^i)^\intercal \mu^i\right)$ $= -\left((d^i)^\intercal B^i d^i + \tilde{\alpha}^i + (b - Ax^i)^\intercal \mu^i\right)$
w^i	stationarity measure $w^i \equiv \frac{1}{2}\|\tilde{g}^i + A^\intercal \mu\|^2_{(B^i)^{-1}} + \tilde{\alpha}^i + (b - Ax^i)^\intercal \mu^i$ $= \frac{1}{2}(d^i)^\intercal B^i d^i + \tilde{\alpha}^i + (b - Ax^i)^\intercal \mu^i$
Ω	support of ω
α_j^i	$\alpha_j^i = f(x^i) - f(y_j) - g_j^\intercal (x^i - y_j)$, linearization error
$\tilde{\alpha}^i$	$\tilde{\alpha}^i = \sum_{j \in L^i} \lambda_j^i \alpha_j^i$, aggregate linearization error
λ_l	dual variables to the lth linear supporting hyperplane, $l \in L$
μ	dual variables to constraints $Ax \stackrel{=}{\leq} b$ or $W_t x_t \stackrel{=}{\leq} h_t - T_t x_{t-1}$
ξ	random variable
ρ	step returned by linesearch routine
ω	realization of the random variable ξ

Chapter 1

Introduction

Medium to long term planning is essential to the success of businesses and project management. In these applications, the problems can often be divided into multiple stages, usually over time. Dynamic programming [2, 6], bilevel programming [110] and mathematical programming with equilibrium constraints [74] are useful modelling and solution techniques for problems with two or more stages. As a lot of the data is not available at the planning stage, the decisions need to be flexible enough to cope with different eventualities. Stochastic programming [10, 62, 105] is an increasingly important problem class for long term planning. Instead of treating the future as a certainty with known data as in classical optimization, stochastic programs incorporate information from a spectrum of possible future events. This gives decision makers the ability to quantify the risk in different scenarios. The stochastic model can be adjusted to reflect the relative importance of each scenario giving a solution strategy that is optimal overall with an acceptable level of risk.

Stochastic programming began in the mid 1950s, and was one of the motivations for Dantzig's seminal work on linear programming. Early work concentrated on the two-stage stochastic linear programs. For example, Van Slyke and Wets [121] developed the L-shaped method which is the basis of many algorithms used today. An important application is a multistage stochastic linear program designed for a large insurance company [13]. This is the first major commercial application of large scale multistage stochastic programming. While important, the linear case may not ade-

quately capture the usual risk averse attitude of decision makers, which is one of the main reasons for applying stochastic programming. With the advance of computing power, it is now possible to solve nonlinear problems with more stages. Nonlinear problems provide more flexibility in the types of applications and allow planners to take a more active role in controlling the return and risk profile of the project. Recent applications include a two-stage air force scheduling problem with a nonlinear objective using the convex DQA algorithm [4], a multistage economic model with general nonlinear subproblems using a nested decomposition algorithm [11] and multistage asset-liability management with quadratic program subproblems [20]. See also [8, 78, 79] for more recent applications of stochastic programming.

1.1 Random Variables and Modelling Uncertainty

Since stochastic programming deals with random data, we need to introduce some notation. The Greek letter ξ is used to denote a random variable. ω represents a possible realization of ξ out of the set of all possible outcomes Ω. A random variable is termed continuous or discrete depending on the set of values it can take on. Demand for electricity is a continuous random variable while the demand for cars is discrete. A discrete random variable has finite support if the set Ω has a finite number of elements. For example, if ξ is a random variables representing the demand for cars in a one month period, then $\Omega = \{0, 1, 2, \ldots, N\}$ where N is some fraction of the population, and ω can take on one of the $N + 1$ values in any month. However, for many random variables, the distinction between discrete and continuous is blurred. In economics and finance, prices of commodities and investment assets are routinely modelled as continuous random variables but in fact prices are quoted to a finite number of decimal places, usually to the nearest cent.

An event $\mathcal{A}$ is a combination of outcomes ω. Suppose ξ represents the state of the economy, then examples of events may be stock prices going up by 0 – 5%, 5 – 10% or more than 10%. A probability $P(\mathcal{A})$ can be associated with each event $\mathcal{A}$. For P to be a probability measure, we need $0 \leq P(\mathcal{A}) \leq 1$ for all $\mathcal{A} \subseteq \Omega$, $P(\Omega) = 1$

and $P(\mathcal{A}_1 \cup \mathcal{A}_2) = P(\mathcal{A}_1) + P(\mathcal{A}_2)$ if $\mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset$.

A continuous distribution is described by its density function $\rho(\xi)$. The probability of ξ being in an interval $[a, b]$ is

$$P(a \leq \xi \leq b) = \int_a^b \rho(\xi) d\xi.$$

The expectation of ξ is defined as $E[\,\xi\,] = \int_{-\infty}^{\infty} \xi \rho(\xi) d\xi$. For the discrete case, it can be simplified to $E[\,\xi\,] = \sum_{\omega \in \Omega} \omega\, P(\omega)$. The expectation of a function of ξ is defined similarly as $E[\,f(x,\xi)\,] = \int_{-\infty}^{\infty} f(x,\xi)\rho(\xi)d\xi$ or $E[\,f(x,\xi)\,] = \sum_{\omega \in \Omega} f(x,\omega)\, P(\omega)$. The variance of a random variable is $E[\,(\xi - E[\,\xi\,])^2\,]$.

1.1.1 Scenario Trees

In many applications, the distribution of the random variables is not known, or even if it is, it may be deemed too expensive to consider a discrete distribution with many possible outcomes or to handle a continuous distribution with numerical integration. It is common practice to select a relatively small set of representative outcomes called scenarios to represent the random events. The scenarios may be the quartiles of a known distribution or historic data, the prediction of some 'experts' or generated by simulations. Each scenario is then given a probability to reflect the likelihood of it happening. The generation of a set of representative scenarios is crucial for a balanced decision and is the topic of ongoing research, see for example [26, 56, 57, 69, 90]. For a multistage model, the scenario information can be organized into a tree structure. Figure 1.1 gives an example of a scenario tree for a 4-stage problem.

The ROOT node represents the present or the part of the data that is known. At stage 2, there are four different possibilities and each of these have various different possible outcomes in stage 3 and so on. A scenario consists of a complete path from the root node to a single leaf node.

Let the number of stages be T and the number of possible outcomes in each stage be K_t for $t = 1, \ldots, T$. The nodes at each stage can be labelled sequentially by $k_t = 1, \ldots, K_t$ for all t. Denote by $D_t(k)$ the immediate descendents in time t of node k. For example, in the scenario tree in Figure 1.1, $D_3(1)$ denotes the immediate

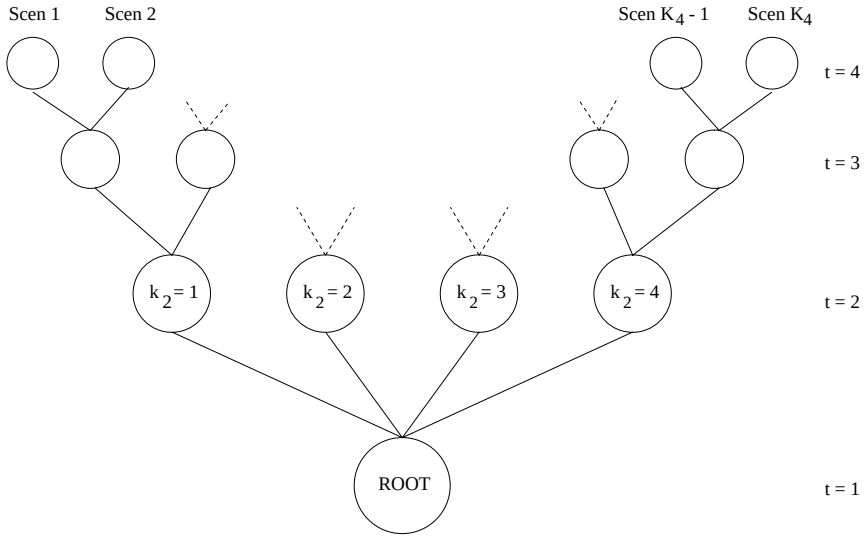

Figure 1.1: Scenario tree

descendents of node 1 which are the two leftmost nodes in time 3. For each leaf node k in stage T, let p_T^k be the associated probability of the scenario occurring. For $t = T-1, \ldots, 1$, p_t^k is given by $p_t^k = \sum_{l \in D_{t+1}(k)} p_{t+1}^l$ with $p_1 = 1$.

Scenario trees provide modellers with the flexibility to choose which scenarios need to be considered and their relative importance. However, since it is impractical to consider too many scenarios, it may limit the level of detail of the model. This is especially true for problems where many random factors are involved. In financial planning, since it is easier to predict trends in the near future, it is common to have more branches at each node in the earlier stages of the scenario trees and progressively fewer towards the end of the planning period.

1.2 Chance Constrained and Recourse Programs

While this thesis concentrates on recourse problems, both are outlined in this section.

1.2.1 Chance Constrained Programs

A *chance constrained program* or *probabilistically constrained program* is a mathematical program in which some constraints are stochastic and are required to be

satisfied with a specified minimum probability. It can take the following form:

$$\begin{aligned} \min_{x} \quad & \mathrm{E}_{\xi}[f(x,\xi)] \\ \text{s.t.} \quad & P\left(\{\xi | g_i(x,\xi) \leq 0\}\right) \geq \alpha_i, \quad i = 1, \ldots, m, \end{aligned}$$

where $0 \leq \alpha_i \leq 1$ for all i. This can be interpreted, for example, as requiring that a power supply system be able to cope with customer demand at least 95% of the time; or that a building can withstand at least 99% of the storms and earthquakes that the area is expected to encounter in the next 50 years without suffering serious structural damage. This formulation is most often used when it is impossible or prohibitively expensive to guarantee that all conditions can be satisfied at all times and the cost of failing those constraints is difficult to quantify. In the power supply example, equipment failure is unavoidable at times, but it is important to keep it at a reasonably low level to avoid the extreme inconvenience and high cost of a power outage. Chance constrained problems are often tackled by converting them into their deterministic equivalents. If the probability distribution of the random parameter satisfies some convexity requirements, then the feasible region is convex and closed. Techniques from mathematical programming can be used to solve them. A chance constraint implies a specific penalty on violation of the constraint. Although the penalty is not spelt out, there is no principle difference between setting a penalty and setting a required probability. See for example [10, 62] for a summary of properties and solution methods for chance constrained programs.

1.2.2 Recourse Programs

In *recourse programs*, the decision process is divided into two or more stages. We begin by considering a two stage recourse problem with linear constraints. It can be formulated as

$$\begin{aligned} \min_{x} \quad & f_1(x) + \mathrm{E}_{\xi}[\mathcal{Q}(x,\xi)] \\ \text{s.t.} \quad & Ax = b, \ x \geq 0, \end{aligned} \tag{1.1}$$

where for each realization ω of ξ,

$$\begin{aligned} \mathcal{Q}(x,\omega) = \min_{y} \quad & f_2(y,\omega) \\ \text{s.t.} \quad & W(\omega)y = h(\omega) - V(\omega)x,\ y \geq 0. \end{aligned} \tag{1.2}$$

The expected value of the second stage objective value $\mathrm{E}_\xi[\mathcal{Q}(x,\xi)]$ is the recourse term. At the first stage, a decision is made based on data currently available. In stage two, for each possible realization of the random variables ξ, a new decision is taken depending on the stage one decision. The expected cost over the two stages can then be calculated and the stage one decision may be revised to achieve a better overall balance of costs between stage 1 and 2. This process is repeated until the overall expected cost is optimal. As an example, consider an investor who is planning to invest in stocks and bonds for a one year period to maximize net return after tax. Suppose our investor decided to adopt a scenario approximation approach and believed that by the end of the year, it is equally likely that the stock price index will fall by 5%, stay the same or rise by 10%, while the return of bonds is known to be 2%. To model this problem, the first stage variables x in (1.1) represent the proportion of money invested in bonds and stock and $f_1(x)$ calculates the cost of setting up the investment portfolio. For any investment plan x and any realization ω of the three scenarios, the recourse variables y in (1.2) represent the return and $f_2(y,\omega)$ gives the return after tax.

Here, we list some important classifications of recourse programs. A recourse problem is said to have

fixed recourse if the recourse matrix W in (1.2) is fixed across all outcomes ω;

complete recourse if for all $v \in \mathbb{R}^m$, there exists $y \geq 0$ such that $Wy = v$;

relatively complete recourse if for all $x \geq 0$ such that $Ax = b$ and for all $\omega \in \Omega$, there exists $y \geq 0$ such that $W(\omega)y = h(\omega) - V(\omega)x$;

simple recourse if W can be expressed as $W = [I\ -I]$.

Simple recourse is a special case of complete recourse, which is in turn a special case of relatively complete recourse. Relatively complete recourse implies that for all x

that are feasible with respect to the first stage constraints, the recourse problem has a non-empty feasible region.

It is straightforward to generalize problem (1.1)–(1.2) to include nonlinear constraints as well.

1.3 Modelling Real Life Problems and Applications

Because of the wide applicability of linear programming models and the availability of reliable and efficient solution methods and software for linear programming, the majority of attention in stochastic program has been devoted to stochastic linear programs. However, we believe that the addition of nonlinear objectives, in particular quadratic and piecewise quadratic models will provide much greater power in modelling applications such as quadratic and piecewise quadratic risk measures in finance.

1.3.1 Stochastic Linear Programs

Stochastic linear programming (SLP) is a very important subclass of stochastic programming. Not surprisingly, the first reported commercial application of multistage stochastic programming was a SLP for the Japanese insurance company Yasuda [13] in the early 1990s. A piecewise linear objective was used to give the desired nonlinearity while keeping the problem solvable. Other recent applications of SLP include [14, 87, 91, 131].

A two-stage SLP can be formulated as

$$\begin{aligned} \min_{x_1} \quad & c_1^{\mathsf{T}} x_1 + E_{\xi_2} \mathcal{Q}_2(x_1, \xi_2) \\ \text{s.t.} \quad & Ax_1 = b, \quad x_1 \geq 0 \end{aligned} \tag{1.3}$$

where for each realization ω_2 of ξ_2

$$\begin{aligned} \mathcal{Q}_2(x_1, \omega_2) = \min_y \quad & q(\omega_2)^\intercal y \\ \text{s.t.} \quad & W(\omega_2)y + V(\omega_2)x_1 = h(\omega_2), \quad y \geq 0. \end{aligned} \tag{1.4}$$

To generalize a two stage SLP to T stages, (1.4) is replaced by

$$\begin{aligned} \mathcal{Q}_t(x_{t-1}, \omega_t) = \min_{x_t} \quad & q_t(\omega_t)^\intercal x_t + E_{\xi_{t+1}|\xi^t}\mathcal{Q}_{t+1}(x_t, \omega_{t+1}) \\ \text{s.t.} \quad & W_t(\omega_t)x_t + V_t(\omega_t)x_{t-1} = h_t(\omega_t), \quad x_t \geq 0, \end{aligned} \tag{1.5}$$

for $t = 2, \ldots, T$ and for each realization ω_t of ξ_t and $\mathcal{Q}(x_T, \cdot) = 0$. Here ξ^t is the history of the random variables up to time t and $E_{\xi_{t+1}|\xi^t}$ is the expected value with respect to ξ_{t+1} conditional on ξ^t.

SLP can be used to model problems with piecewise linear objective and constraints. This allows risk to be limited to an acceptable degree by controlling the maximum possible loss or through the use of piecewise linear risk measure.

SLP can be solved either as a large scale linear program, see Section 2.1, or by the L-shaped method and its variants which are discussed in Section 2.4.

1.3.2 Portfolio Optimization

Portfolio optimization is an important problem in financial management. It was first developed by Markowitz in his seminal work [77] in 1952. The goal is to maximize the expected return of a portfolio of investments while minimizing the risk posed to the investors. The standard measure of risk used in finance is the variance or standard deviation of return. Assume there are n asset classes with expected returns given by $\bar{r} \in \mathbb{R}^n$ and covariance matrix C where $C \in \mathbb{R}^{n\times n}$ is symmetric positive semi-definite. Let $x \in \mathbb{R}^n$ represent the proportion of fund invested in the n investments, so it satisfies $\sum_1^n x = e^\intercal x = 1$, where $e = (1, 1, \ldots, 1)^\intercal \in \mathbb{R}^n$. The portfolio return $r \in \mathbb{R}$ is $r = \bar{r}^\intercal x$ and the risk, as measured by the variance of r, is given by $x^\intercal C x$. Since investments with high expected returns usually have higher risk, some compromise is needed between striving for higher return and keeping risk under control. One

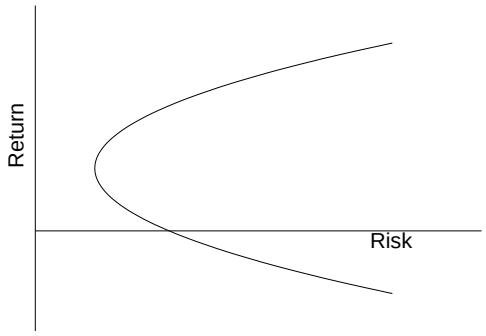

Figure 1.2: Typical efficient frontier

way to formulate this into a mathematical program is

$$\begin{aligned} \max_x \quad & \bar{r}^\mathsf{T} x \\ \text{s.t.} \quad & x^\mathsf{T} C x \leq c \\ & Ax \stackrel{=}{\leq} b, e^\mathsf{T} x = 1, x \geq 0, \end{aligned}$$

where the symbol $\stackrel{=}{\leq}$ means the constraints may be equalities or inequalities. This formulation maximizes the expected return while keeping the portfolio risk under a prespecified limit c. Alternatively, risk can be minimized while attaining an adequate expected return τ:

$$\begin{aligned} \min_x \quad & x^\mathsf{T} C x \\ \text{s.t.} \quad & \bar{r}^\mathsf{T} x \geq \tau \\ & Ax \stackrel{=}{\leq} b, e^\mathsf{T} x = 1, x \geq 0. \end{aligned} \tag{1.6}$$

It is also possible to combine the two objectives into one by introducing a weighting parameter α, $\alpha \geq 0$, giving

$$\begin{aligned} \min_x \quad & \alpha x^\mathsf{T} C x - \bar{r}^\mathsf{T} x \\ \text{s.t.} \quad & Ax \stackrel{=}{\leq} b, e^\mathsf{T} x = 1, x \geq 0. \end{aligned} \tag{1.7}$$

As C is positive semi-definite, (1.6) and (1.7) are convex quadratic programs and can be solved by standard QP software, see Section 2.2 for more details. By varying respectively the level of acceptable risk c, minimum expected return τ or the

weighting factor α in the three formulations, we can trace out what is known as the efficient frontier. A typical efficient frontier is shown in Figure 1.2. The top half of the curve depicts the set of portfolios which have the highest level of expected return for a given level of variance or the lowest variance for given expected return. For more detailed discussion on portfolio optimization, see for example [27].

1.3.3 Asymmetric Risk Measures

Although standard deviation (or variance) is the standard risk measure used in the finance industry, it has the drawback of penalizing above target gain as much as below target loss. It is well known from utility theory [125] that any risk measure is convex, non-decreasing with a decreasing rate of increase. Various authors have proposed the use of lower partial moments. For a random variable X, writing $\overline{X} \equiv E[\,X\,]$ as its expected value, the n-th lower partial moment [48] is defined as

$$L^n(X) = E[\,(\min(0, X - \overline{X}))^n\,].$$

This measures the n-th moment of the part of the random variable that is below its expected value. Of particular interest is the semi-variance defined as $E[\,(\min(0, X - \overline{X}))^2\,]$. It is a once continuously differentiable convex piecewise quadratic function in X. Piecewise quadratic functions will be defined and studied in Section 3.2.

The definitions of variance and lower partial moments can be generalized to measure deviation from a target value τ instead of the expected value $\overline{X}$,

$$L^n(X, \tau) = E[\,(\min(0, X - \tau))^n\,].$$

This may be viewed as the penalty for under-performing the target. Lower partial moments have the advantage that they are intuitively much more appealing as risk measures since people are only too happy to get a higher than expected return to their investment. However, opponents believe that, unlike the standard deviation, relatively little is known about the properties of lower partial moments and the resulting optimization program is much harder to solve for $n > 1$. There is also concern [64] that lower-partial moment measures over-estimate the investors' desire

for above target gain and put them into potentially more risky position than they are prepared to accept. King [64] proposed piecewise linear-quadratic measures as a compromise. This class of risk measures is defined as

$$\rho_{a,b}(r) = \begin{cases} ar - \frac{1}{2}a^2 & \text{if } r \leq a, \\ \frac{1}{2}r^2 & \text{if } a < r < b, \\ br - \frac{1}{2}b^2 & \text{if } b \leq r, \end{cases} \tag{1.8}$$

where $a \leq 0 \leq b$. Here the argument r can be $X - \overline{X}$ or $X - \tau$. The linear-quadratic risk measure is compared to variance in Figure 1.3.

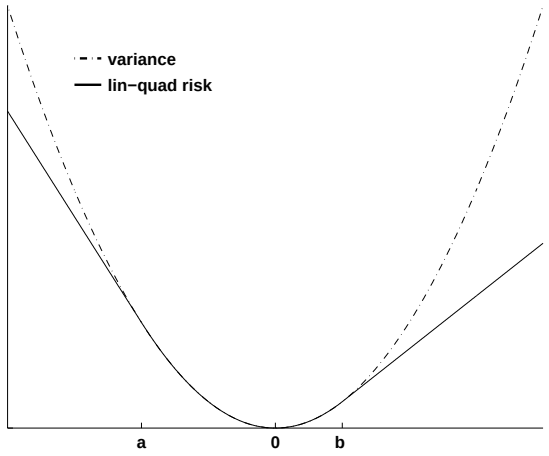

Figure 1.3: Linear-quadratic risk measure

The linear-quadratic risk measure can be seen as a generalized form of the M-estimator [55]. For a discrete random variable with K realizations and residuals $r_i, i = 1, \ldots, K$, the M-estimator is defined as

$$\sum_{i=1}^{K} w(r_i),$$

where

$$w(r) = \begin{cases} r^2/2, & |r| \leq c \\ c|r| - c^2/2, & |r| > c. \end{cases}$$

It is a robust version of the sum of square error measure $\sum_{i=1}^{K} r_i^2$. The linear-quadratic risk measure has the advantage that it is less sensitive to values outside the range $[a, b]$. It is equivalent to the M-estimator if $-a = b = c$. The possibility

of choosing different values for a and b make the linear-quadratic risk measure very versatile.

Lower partial moments and linear quadratic risk measures are more flexible than variance and give investors more control on how risk is measured. These general risk measures can be used instead of variance in portfolio optimization problems. Assume that a scenario approach is used and the forecast return in each scenario k is given by $\mu^k \in \mathbb{R}^n$ for the n assets and has probability p^k for $k = 1, \ldots, K$. Let $x \in \mathbb{R}^n$ be the proportion of fund invested in the different assets. The return in each scenario is $r^k = x^\intercal \mu^k$ and the expected return is $\bar{r} = \sum_{k=1}^K p^k r^k$. The target semi-variance of the portfolio is given by

$$\sum_{k=1}^{K} p^k \left(\min(0, r^k - \tau)\right)^2$$

The target linear-quadratic risk is

$$\sum_{k=1}^{K} p^k \rho_{a,b}(r^k - \tau),$$

which is equivalent to the semi-variance if $\tau = \bar{r}$ and $b = -a = \infty$. The mean-variance problem (1.7) can be converted to a mean target semi-variance problem

$$\begin{aligned} \min_{x,r} \quad & \alpha \sum_{k=1}^{K} p^k \left(\min(0, r^k - \tau)\right)^2 - \bar{r} \\ \text{s.t.} \quad & (\mu^k)^\intercal x = r^k, \quad k = 1, \ldots, K \\ & \bar{r} = \sum_{k=1}^{K} p^k r^k \\ & Ax \stackrel{=}{\leq} b, e^\intercal x = 1, x \geq 0. \end{aligned} \tag{1.9}$$

The target return τ can of course be set to equal the expected portfolio return $\bar{r}$. The objective is a separable piecewise quadratic function in r. The problem can be

converted to an ordinary QP by splitting $r^k - \tau$ into its positive and negative parts

$$\begin{aligned} \min_{x, r_-^k, r_+^k} \quad & \alpha \sum_{k=1}^{K} p^k (r_-^k)^2 - \bar{r} \\ \text{s.t.} \quad & (\mu^k)^\intercal x - \tau = r_+^k + r_-^k \\ & \bar{r} = \sum_{k=1}^{K} p^k (\mu^k)^\intercal x \\ & Ax \mathrel{\overline{\leq}} b, e^\intercal x = 1, x \geq 0 \\ & r_+^k \geq 0, r_-^k \leq 0. \end{aligned} \tag{1.10}$$

To use a linear-quadratic target risk instead of target semi-variance, we can replace $\sum_{k=1}^{K} p^k (\min(0, r^k - \tau))^2$ in the objective of (1.9) by $\sum_{k=1}^{K} p^k \rho_{a,b}(r^k - \tau)$. To change the problem to a QP, r^k can be split into $r^k = r_+^k + r_0^k - r_-^k$ where

$$r_-^k = -\min(r^k, a), \quad r_+^k = \max(r^k, b),$$
$$r_0^k = \begin{cases} a, & r^k < a, \\ r^k, & a \leq r^k \leq b, \\ b, & r^k > b. \end{cases}$$

The linear-quadratic risk function can be rewritten as

$$\rho_{a,b}(r^k - \tau) = -a r_-^k + \tfrac{1}{2} r_0^{k2} + b r_+^k - a^2 - b^2.$$

If we drop the constant terms $-a^2 - b^2$, the optimization program becomes

$$\begin{aligned} \min_{x, r_+^k, r_0^k, r_-^k} \quad & \alpha \sum_{k=1}^{K} p^k \left(b\, r_+^k + \tfrac{1}{2}(r_0^k)^2 - a\, r_-^k \right) - \bar{r} \\ \text{s.t.} \quad & (\mu^k)^\intercal x - \tau = r_+^k + r_0^k - r_-^k \\ & \bar{r} = \sum_{k=1}^{K} p^k (\mu^k)^\intercal x \\ & Ax \mathrel{\overline{\leq}} b, e^\intercal x = 1, x \geq 0. \\ & r_+^k \geq a,\ a \leq r_0^k \leq b,\ r_-^k \geq b. \end{aligned} \tag{1.11}$$

In the next section, we show how these risk measures can be extended to multi-stage problems.

1.3.4 Multistage Financial Planning

For medium to long term planning problems, it is often more appropriate to use a multistage model to adequately reflect the progression in time and the amount of information available. It also allows the inclusion of transaction costs for more realistic planning [88].

As an example, we now demonstrate how to formulate a multistage portfolio planning problem using a target semi-variance risk measure. Suppose that for T future periods, return forecasts $\mu_t^{k_t}$, for $t = 1, \ldots, S$, $k_t = 1, \ldots, K_t$ are available in the form of a scenario tree like Figure 1.1. The investor would like to maximize expected return and minimize transaction costs and risk as measured by target semi-variance over the whole planning horizon. The multistage stochastic program can be formulated as

$$\begin{aligned} \min_{x_1} \quad & c_1^\intercal x_1 + E[\, \mathcal{Q}_2(x_1, \xi_2)\,] \\ \text{s.t.} \quad & A x_1 \;\overline{\overline{\leq}}\; b_1, e^\intercal x_1 = 1, x_1 \geq 0, \end{aligned}$$

where c_t in the objective measures the costs associated with forming the portfolio and

$$\begin{aligned} \mathcal{Q}_t(x_{t-1}, \omega^k) = \min_{x_t, r_t} \quad & (c_t^k)^\intercal x_t - r_t + \alpha \left(\min(0, r_t - \tau_t^k)\right)^2 + E[\, \mathcal{Q}_t(x_t, \xi_{t+1})\,] \\ \text{s.t.} \quad & r_t = \mu_t^k x_{t-1} \\ & W_t^k x_t = h_t^k - V_t^k x_{t-1} \\ & A_t^k x_t \;\overline{\overline{\leq}}\; b_t^k, e^\intercal x_t = 1, x_t \geq 0, \end{aligned}$$

and the end terms are

$$Q_T(x_{T-1}, \omega_T) = -r_T + \alpha \left(\min(0, r_T - \tau_T^k)\right)^2$$

where $r_T = (\mu_T^k)^\intercal x_{T-1}$.

It is more complicated to use the expected return as the target value in the multistage problem. As the expected return for each stage depends on the returns of all nodes of that stage, the objective is no longer separable across the nodes. It is necessary to pass all the return variables from each node back to the root and calculate the expected return and variance for each stage there.

1.4 Multistage Quadratic Stochastic Programs (MQSPs)

This section formulates multistage quadratic stochastic programs where subproblems in each stage minimize a convex quadratic objective plus a recourse term subject to linear constraints. The multistage financial planning problems discussed in Section 1.3.4 are examples of this class of problems.

1.4.1 Model Formulation

For each stage t of a T-stage problem, let the random variable $\xi_t = (H_t, c_t, W_t, V_t, h_t)$ denote the stochastic input data to the multistage planning problem. Its value is given by the realization of an underlying random event ω_t defined over the event space Ω_t. A T-stage quadratic stochastic problem can be formulated as

$$\begin{aligned} \min_{x_1 \in \mathbb{R}^{n_1}} \quad & f_1(x_1) \equiv \tfrac{1}{2} x_1^{\mathsf{T}} H_1 x_1 + c_1^{\mathsf{T}} x_1 + Q_1(x_1) \\ \text{s.t.} \quad & W_1 x_1 \stackrel{=}{\leq} h_1, \end{aligned}$$

where for $t = 2, \ldots, T$, the recourse function is

$$Q_t(x_t, \xi_t) = E_{\xi_{t+1}|\xi^t} \mathcal{Q}_{t+1}(x_t, \xi_{t+1})$$

and

$$\begin{aligned} \mathcal{Q}_t(x_{t-1}, \xi_t) = \min_{x_t \in \mathbb{R}^{n_t}} \quad & f_t(x_t) \equiv \tfrac{1}{2} x_t^{\mathsf{T}} H_t(\omega_t) x_t + c_t(\omega_t)^{\mathsf{T}} x_t + Q_t(x_t, \xi_t) \\ \text{s.t.} \quad & W_t(\omega_t) x_t + V_t(\omega_t) x_{t-1} \stackrel{=}{\leq} h_t(\omega_t), \quad a.s., \end{aligned}$$

and $Q_T(x_T, \cdot) = 0$. Here $E_{\xi_{t+1}|\xi^t}$ denotes the expectation with respect to ξ_{t+1} conditional on the history of ξ up to time t, abbreviated as ξ^t. The symbol $\stackrel{=}{\leq}$ means the constraints may be equalities or inequalities. $H_t(\omega_t) \in \mathbb{R}^{n_t \times n_t}$ are symmetric positive semi-definite, $W_t(\omega_t) \in \mathbb{R}^{m_t \times n_t}$, $V_t(\omega_t) \in \mathbb{R}^{m_t \times n_{t-1}}$, $c_t(\omega_t) \in \mathbb{R}^{n_t}$ and $h_t(\omega_t) \in \mathbb{R}^{m_t}$. All data is fixed for $t = 1$, but some or all may be stochastic for $t = 2, \ldots, T$.

All variables are assumed bounded in this thesis. It is a natural assumption for many applications. We also assume that the random variables are discrete and are

arranged in the form of a scenario tree. Each node of the scenario tree is identified by the index pair (t,k). Since this formulation will be referred to constantly in the sequel, we will rewrite it with the explicit (t,k) indices for clarity.

$$\min_{x_1\in\mathbb{R}^{n_1}} \quad f_1(x_1) \equiv \tfrac{1}{2}x_1^\mathsf{T}H_1x_1 + c_1^\mathsf{T}x_1 + Q_1(x_1) \tag{1.12a}$$

$$\text{s.t.} \quad W_1x_1 \overset{=}{\leq} h_1, \tag{1.12b}$$

where for $t = 2,\dots,T$ and $k = 1,\dots,K_t$,

$$Q_t^k(x_t^k) = \sum_{\hat{k}\in D_{t+1}(k)} p_{t+1}^{\hat{k}}\mathcal{Q}_{t+1}^{\hat{k}}(x_t^k,\xi_{t+1}^{\hat{k}}),$$

and

$$\mathcal{Q}_t^k(x_{t-1},\xi_t^k) = \min_{x_t\in\mathbb{R}^{n_t}} \quad f_t^k(x_t) \equiv \tfrac{1}{2}x_t^\mathsf{T}H_t^kx_t + (c_t^k)^\mathsf{T}x_t + Q_t^k(x_t) \tag{1.13a}$$

$$\text{s.t.} \quad W_t^kx_t + V_t^kx_{t-1} \overset{=}{\leq} h_t^k, \quad a.s. \tag{1.13b}$$

From now on, we will write $Q(x_t^{k_t})$ instead of $Q_t^{k_t}(x_t^{k_t})$ and similarly for other variables to avoid unnecessary indices.

1.4.2 Large QP Equivalent

Problem (1.12) can be equivalently formulated as a large scale quadratic program (QP):

$$\begin{aligned}
\min \quad & \tfrac{1}{2}x_1^\mathsf{T}H_1x_1 + c_1^\mathsf{T}x_1 + \sum_{k=1}^{K_2} p_2^k\left(\tfrac{1}{2}(x_2^k)^\mathsf{T}H_2^kx_2^k + (c_2^k)^\mathsf{T}x_2^k\right) + \dots \\
& \qquad + \sum_{k=1}^{K_T} p_T^k\left(\tfrac{1}{2}(x_T^k)^\mathsf{T}H_T^kx_T^k + (c_T^k)^\mathsf{T}x_T^k\right) \\
\text{s.t.} \quad & W_1x_1 \overset{=}{\leq} h_1, \\
& V_2^kx_1 + W_2^kx_2^k \overset{=}{\leq} h_2^k, \quad k = 1,\dots,K_2, \\
& \qquad\qquad \ddots \qquad\qquad \vdots \\
& V_T^kx_{T-1}^i + W_T^kx_T^k \overset{=}{\leq} h_T^k, \quad i = 1,\dots,K_{T-1}, \\
& \qquad\qquad\qquad\qquad\qquad k \in D_T(i).
\end{aligned} \tag{1.14}$$

The QP (1.14) has $\sum_{t=1}^T K_tn_t$ variables and $\sum_{t=1}^T K_tm_t$ constraints. As the number of stages and scenarios increase, the dimension of the large scale QP increases

dramatically. It quickly becomes impractical to solve this problem directly if the number of scenarios in each stage K_t is too large. Therefore, it is necessary to compromise between the desire to model the problem more finely and the need to keep the problem tractable.

The objective Hessian of (1.14) is block diagonal. The highly structured constraint matrix can be exploited by the Dantzig-Wolfe decomposition [24] on its dual form. Benders' decomposition [3] can be used on the primal problem which is the basis of the L-shaped method [121].

1.5 Overview of Thesis

In Chapter 2, we survey techniques for solving linear programming and quadratic programming problems. They are the subproblems of (1.12)–(1.13) and it is important we solve them efficiently. This is followed by a study of convex nonsmooth programs. The techniques will be used in Chapter 4 when we develop a generalized Newton method to solve (1.12)–(1.13). Then we survey methods for solving different classes of stochastic programs.

Chapter 3 is concerned with the theoretical properties of the multistage quadratic stochastic program (1.12)–(1.13). We start by summarizing many useful results from sensitivity analysis. Then piecewise quadratic programs are defined and their sensitivity properties are studied. The results are applied to show that each subproblem of a multistage quadratic stochastic program is a piecewise quadratic program. We look at their differential properties and show how to obtain the quadratic expression of the component pieces of the piecewise quadratic objective.

Generalized Newton methods for solving (1.12)–(1.13) are developed in Chapter 4. The first algorithm looks at the simplest case where each subproblems are assumed to have strictly convex and differentiable objective and satisfy the relatively complete recourse assumption. Then we show how feasibility can be ensured by adding feasibility cuts to the basic algorithm. Finally, Algorithm 4.3 is obtained by applying techniques from nonsmooth optimization to the basic algorithm so that

it can handle problems with convex Lipschitz objectives.

In Chapter 5, the first generalized Newton algorithm is shown to converge globally and finitely if each piecewise quadratic objective of the multistage quadratic stochastic program is strictly convex and differentiable. The rate of convergence is shown to be superlinear. The same conclusions are extended to the case where feasibility cuts are included. Then we prove the global convergent property of Algorithm 4.3 for convex Lipschitz problems.

Chapter 6 gives details of the implementation and results of numerical experiments carried out to study the performance of Algorithm 4.3. We study the line-search strategy and trust region scheme especially developed for (1.12)–(1.13). A number of efficiency schemes studied in the literature are implemented. The numerical tests were performed on sets of random data with different scenario tree structures. The tests are repeated on two sets of publicly available test data for stochastic linear program, modified by adding small artificial quadratic terms to the objective. Conclusions and directions for future research are given in Chapter 7.

Chapter 2

Survey of Stochastic Programming Algorithms

This chapter surveys the different types of algorithms currently available to solve multistage quadratic stochastic programs. The two main approaches are to solve the deterministic equivalent (1.14) as a large scale mathematical program; or to decompose the multistage problem into smaller problems.

All recourse programs can be expressed as large scale mathematical programs. For multistage quadratic stochastic program (1.12)–(1.13), the deterministic equivalent (1.14) is a standard QP and can, in principle, be solved by the many available standard QP solvers. However, as the size of the deterministic equivalent increases sharply as more stages and scenarios are considered, it is important to exploit the highly structured constraint matrix and maintain sparsity as much as possible for an efficient algorithm. Scaling and pre-solving are essential to reduce the problem size and improve numerical stability.

Decomposition algorithms break up problem (1.12)–(1.13) into smaller subproblems. This enables us to solve problems of much larger size than by attacking the deterministic equivalents. The price is that each subproblem may have to be solved many times. There is a need to balance the advantage of having very small subproblems against reducing the number of iterations and subproblems solved. Decomposition algorithms can be classified into primal and dual. Primal decompo-

sition breaks up the problem along the stages. Information is passed back and forth between the stages to achieve feasibility and optimality. Dual decomposition solves the individual scenario problems. Information is exchanged between problems at the same time stage to ensure nodes with the same history make the same decisions.

2.1 Large Scale Linear Programs

For multistage stochastic linear programs (1.3), the deterministic equivalent is a large sparse linear program. This can be solved by either the simplex method or by an interior point method. The simplex method is generally considered to be very efficient for small to medium dense problems, while interior point methods are superior for large sparse problems and the cputime is expected to grow only linearly with the problem data size. There are many sophisticated commercial software like CPLEX [21] and OSL [60] for linear programming implementing both the simplex and interior point methods. For detailed discussion on both the simplex and interior point methods see, for example, [124, 128].

2.1.1 Simplex Methods

The simplex method considers the standard LP problem

$$\begin{aligned} \min \quad & c^{\mathsf{T}} x \\ \text{s.t.} \quad & Ax = b, x \geq 0. \end{aligned}$$

where $c \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. It uses the fact that $n - m$ variables must be zero at a vertex solution to partition the variables into basic, $x_i \geq 0$, and non-basic, $x_i = 0$. A and c are likewise partitioned: $A = [B|N], c = (c_B; c_N)$. The basic variables and the objective can be expressed in terms of x_N:

$$x_B = B^{-1}(b - Nx_N),$$

$$c^{\mathsf{T}} x = c_B^{\mathsf{T}} x_B + c_N^{\mathsf{T}} x_N = c_B^{\mathsf{T}} B^{-1} b + (c_N - N^{\mathsf{T}} B^{-\mathsf{T}} c_B)^{\mathsf{T}} x_N.$$

If any component i of the reduced cost vector $d_N = c_N - N^\intercal B^{-\intercal} c_B$ is negative, then the objective value can be decreased by letting x_{N_i} become positive, ie. basic. The change in x_B is $B^{-1}(b - N_i x_{N_i})$. The way i is chosen if there are more than one negative component is termed pricing. x_{N_i} is increased until one of the basic variables becomes zero and non-basic. The main computational cost of the simplex method is calculating $B^{-\intercal} c_B$ and $B^{-1} N_i$. This can be done by using a QR or LU factorization of B and updating the factors in each iteration. The simplex method moves between adjacent vertices until no improvement can be found. If all the simplex steps are non-degenerate, then the algorithm terminates after a finite number of iterations.

Large Sparse Simplex

A crash procedure [43] is used in all large simplex implementations to get a good starting basis and this can significantly reduce the number of iterations. Fourer [34] looked at specializing large LP technologies to the staircase constraint case. He demonstrated how the sparse LU updates and the backward and forward substitutions can exploit the staircase structure. Because of the special block structure of the constraints in (1.14), we can consider the large constraint matrix as composed of smaller sub-blocks and work with the associated bases. Only the inverses of the smaller bases of the sub-blocks need to be updated which is a significant saving. This is very similar to basis factorization [61, 116].

An efficient pricing strategy, usually Devex or steepest edge is also important for fast solution. Unlike the originally proposed method which chooses the most negative reduced cost, these two pricing schemes look at the gradient in the space of the structural variable instead of the space of the current basic variables. In steepest edge pricing, the gradient is calculated exactly, while Devex employs a heuristic. Although these two pricing schemes are computationally demanding, they can significantly reduce the total number of simplex steps required and lead to faster solution time for large problems. For really large problems, it may be beneficial to use partial pricing where only a subset of the reduced cost vector is calculated. While pricing strategies look at the rate of decrease of the objective if a non-basic

variable is increased, it doesn't take into account the maximum step and therefore the actual decrease. Multiple pricing selects a group of indices and calculate the actual decrease. It is very expensive but can reduce the number of iterations. These pricing schemes are discussed in [35] and techniques which adapt these schemes efficiently to staircase LPs are proposed.

Parallel Simplex

Simplex methods are not easily amenable to parallelization. In [112], the basic revised simplex method was parallelized, and almost linear speedup was achieved for the problems tested (with up to 500 variables). However, there is significant fill-in in the update scheme for the basis inverse B^{-1} which is clearly not suitable for large sparse problems. An implementation with a sparse LU factorization was also tested. However, very little speedup was achieved due to the sparsity of the matrices and heavy communications needed. Because of the difficulties in parallelizing individual operations in simplex methods, Hall and McKinnon [45,46] adopted an asynchronous approach. Different operations are performed in parallel, so some processors may be using outdated information. This approach is also more prone to numerical instability compared to revised simplex methods. Speedups between 2 to 5 using 2 to 12 processors are reported.

2.1.2 Interior Point Methods

Interior point methods are recognized as being very efficient for large sparse linear programs. One of their major advantages is that as the problem size increases, the number of iterations remains stable and the cputime grows linearly, see numerical results in Section 6.2. Here we briefly sketch the primal-dual variant of the interior point method. See [124, 128, 130] for excellent coverage of interior point methods. Consider the problem

$$\begin{aligned} \min \quad & c^{\mathsf{T}} x \\ \text{s.t.} \quad & Ax \leq b, x \geq 0. \end{aligned}$$

By adding a slack variable w and using a log barrier function with penalty parameter μ to enforce non-negativity of x and w, we have

$$\begin{aligned} \min \quad & c^\intercal x + \mu \sum_j \log x_j + \mu \sum_i \log w_i \\ \text{s.t.} \quad & Ax + w = b. \end{aligned}$$

The Lagrangian is

$$\mathcal{L}(x, w, y) = c^\intercal x + \mu \sum_j \log x_j + \mu \sum_i \log w_i + y^\intercal (b - Ax - w).$$

The first order optimality conditions are then

$$\begin{aligned} Ax + w &= b, \\ A^\intercal y - \mu X^{-1} e &= c, \\ y &= \mu W^{-1} e. \end{aligned}$$

Here X denotes the diagonal matrix whose diagonal entries are given by x, and similarly for W. We can rewrite the last set of equations as

$$Ax + w = b, \qquad A^\intercal y - z = c, \tag{2.1}$$

$$XZe = \mu e, \qquad WYe = \mu e. \tag{2.2}$$

The solution to this system of equations for decreasing μ is termed the central path. Starting from strictly positive primal and dual variables and slacks x, y, w, z, the path following algorithm finds $(\triangle x, \triangle y, \triangle w, \triangle z)$ that approximately lies on the central path. Substituting x by $x + \triangle x$ and similarly for y, z, w, (2.1) and (2.2) become

$$A\triangle x + \triangle w = b - Ax - w, \tag{2.3}$$

$$A^\intercal \triangle y - \triangle z = c - A^\intercal y + z, \tag{2.4}$$

$$Z\triangle x + X\triangle z = \mu e - XZe - \triangle X \triangle Z e, \tag{2.5}$$

$$W\triangle y + Y\triangle w = \mu e - YWe - \triangle Y \triangle W e.. \tag{2.6}$$

Since W and Z are diagonal, if the nonlinear terms $\triangle X \triangle Z e$ in (2.5) and $\triangle Y \triangle W e$ in (2.6) are dropped, we can solve for $\triangle w$ and $\triangle z$. (2.3) and (2.4) can be rearranged

to obtain the reduced KT system

$$\begin{bmatrix} -Y^{-1}W & A \\ A^{\intercal} & X^{-1}Z \end{bmatrix} \begin{bmatrix} \triangle y \\ \triangle x \end{bmatrix} = \begin{bmatrix} b - Ax - \mu Y^{-1}e \\ c - A^{\intercal}y + \mu X^{-1}e \end{bmatrix}. \tag{2.7}$$

It is possible to solve the reduced KT system further by explicitly solving for $\triangle x$ or $\triangle y$ to obtain the normal equations in the primal form

$$-(Y^{-1}W + AXZ^{-1}A^{\intercal})\triangle y = b - Ax - \mu Y^{-1}e - AXZ^{-1}(c - A^{\intercal}y + \mu X^{-1}e) \tag{2.8}$$

or the dual form

$$(A^{\intercal}YW^{-1}A + X^{-1}Z)\triangle x = c - A^{\intercal}y + \mu X^{-1}e + A^{\intercal}YW^{-1}(b - Ax - \mu Y^{-1}e). \tag{2.9}$$

The KT matrix in (2.7) is quasi-definite while the matrices in the normal equations (2.8) and (2.9) are positive definite. A matrix M is called quasi-definite [123] if it is of the form

$$M = \begin{bmatrix} -E & A^{\intercal} \\ A & F \end{bmatrix}$$

where E and F are symmetric positive definite. All three systems (2.7), (2.8) and (2.9) can be solved by using a $LDL^{\intercal}$ factorization where L is lower triangular and D is diagonal. The elements of D may be negative if the matrix is quasi-definite.

Large Sparse Interior Point Methods

Maintaining sparsity is the most important factor in an efficient implementation of an interior point method. This is achieved by permuting the rows and columns of the matrices in (2.7), (2.8) and (2.9) before factorization to minimize fill-in in the factor L. The most commonly used reordering schemes for large sparse matrices include the minimum degree ordering, its variant approximate minimum degree [37] and nested dissection ordering [1]. These three ordering schemes are more expensive than their competitors but they generally result in lower fill-in. As X, Y, W and Z are diagonal matrices, the sparsity pattern in the KT system (2.7) and the normal equations (2.8) and (2.9) remain the same throughout the algorithm, so it is only necessary to perform the reordering once in the beginning. It is worthwhile spending more time in reordering as the factorizations are computationally intensive operation

and need to be repeated many times. Dense columns in A can create significant fill-in in (2.8) and dense rows lead to a dense system in (2.9). This suggests factorizing the reduced KT matrix directly since it is sparse, although this may be less stable numerically. Dense columns can also be handled by the Schur complement which splits the constraint matrix into sparse and dense components. Lustig et al. [76] suggested splitting the variables to induce sparsity at the expense of increasing the problem size. They claim this method avoids the numerical difficulties sometimes associated with the Schur complement and can achieve higher accuracy, however, it is not as efficient.

Instead of dropping the nonlinear term completely, predictor-corrector methods [81] can be used to improve the accuracy of the new point. (2.7) is first solved to give $(\triangle x, \triangle y)$, then $(\triangle w, \triangle z)$ can be obtained by (2.3) and (2.4). The cross product term $\triangle X \triangle Z e$ and $\triangle Y \triangle W e$ are calculated and (2.7) with updated right hand side is solved again to give a more accurate estimate of $(\triangle x, \triangle y)$.

Parallel Interior Point Methods

Interior point methods are more readily parallelizable than simplex methods. The largest reduction in time can be obtained by parallelizing the Cholesky factorization routine followed by sparse triangular system solvers and sparse matrix multiplications. IBM's OSL [12] has reported speedup of typically 3 to 13 on 25 processors. A lot of work has been done in this area, see for example [22, 25, 75, 129].

2.2 Large Scale Quadratic Programs

Quadratic programs are required in the solution of (1.12)–(1.13), both for the individual subproblems or for solving the large scale equivalent problem (1.14). They are usually solved by active set methods or interior point methods.

2.2.1 Active Set Methods

Consider first the convex equality constrained QP

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \tfrac{1}{2} x^\intercal H x + c^\intercal x \\ \text{s.t.} \quad & Ax = b \end{aligned} \tag{2.10}$$

where $H \in \mathbb{R}^{n \times n}$ is positive semi-definite and $A \in \mathbb{R}^{m \times n}$. The null space method finds a basis Z for the null space of A, usually using a QR factorization for numerical stability or LU factorization on a submatrix of A if it is sparse. Any feasible solutions x can be expressed in term of another feasible point x_0 as $x = x_0 + Zw$ for some $w \in \mathbb{R}^{n-m}$. Then (2.10) is equivalent to

$$\min_{w \in \mathbb{R}^{n-m}} \tfrac{1}{2} w^\intercal (Z^\intercal H Z) w + (H x_0 + c)^\intercal Z w$$

and the solution is calculated by solving

$$(Z^\intercal H Z) w = -(H x_0 + c)^\intercal Z \tag{2.11}$$

using a factorization of $Z^\intercal H Z$.

For problems with inequality constraints, the active set method maintains a working list of active constraints J_i at iteration i. At a feasible point x_i, the following equality constrained QP is solved to give a descent direction d_i.

$$\{\min \tfrac{1}{2}(x_i + d)^\intercal H (x_i + d) + c^\intercal (x_i + d) : A_j (x_i + d) = b_j, j \in J_i\}.$$

If d_i is 0 and all Lagrange multipliers for inequality constraints are non-negative, then x_i is optimal for the original problem, otherwise, the constraint with the most negative multiplier is dropped from the active set. If $d_i \neq 0$, $x_i + d_i$ is the optimal solution in the current subspace. It is necessary to calculate the maximum possible step

$$\alpha = \max \left\{ \frac{b_j - a_j^\intercal x_i}{a_j^\intercal d_i} \mid a_j^\intercal d_i > 0, j \notin J_i \right\}$$

that does not violate any of the inactive constraints. The step is then given by

$$\rho_i = \min\{1, \alpha_i\},$$

and we set $x_{i+1} = x_i + \rho_i d_i$. If $\rho_i < 1$, the constraint that limits the step is added to the list of working active constraint list. As only one constraint is dropped or added each time, the LU factors of A can be updated efficiently as in the simplex method. Since the number of combinations of active constraints is finite, if the problem is not degenerate, then the sequence of function values is strictly decreasing and the active set method can be shown to converge finitely. See, for example, Fletcher [33] for more details on active set method.

Large Sparse Active Set Method

The problem with the null space method for large sparse equality constrained QP solver is that Z and therefore $Z^\intercal HZ$ may be dense if A is sparse but has dense columns. A more recent active set approach [42] aims to preserve sparsity by working directly on the KT conditions. The model problem is

$$\begin{aligned} \min \quad & \tfrac{1}{2} x^\intercal H x + c^\intercal x \\ \text{s.t.} \quad & Ax = b, l \leq x \leq u. \end{aligned} \tag{2.12}$$

Let λ and μ be the Lagrange multipliers for the equality and bound constraints respectively. For simplicity, we assume that no lower bounds are active since they can be treated in the same way as upper bounds. For each working active set, let the subscript FX denotes the variables fixed at their upper bounds. The first order optimality condition is

$$\begin{aligned} Hx + c + A^\intercal \lambda + I\mu = 0, \\ Ax = b, \\ x_{FX} = u_{FX}. \end{aligned}$$

Writing this in matrix form and splitting the fixed and free variables, denoted by the subscript FX and FR respectively, and using OD to denote off diagonal elements,

the system of equations become

$$\begin{bmatrix} H_{FX} & H_{OD} & A_{FX}^{\intercal} & I \\ H_{OD} & H_{FR} & A_{FR}^{\intercal} & 0 \\ A_{FX} & A_{FR} & 0 & 0 \\ I & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} x_{FX} \\ x_{FR} \\ \lambda \\ \mu \end{pmatrix} = \begin{pmatrix} -q_{FX} \\ -q_{FR} \\ b \\ u_{FX} \end{pmatrix}.$$

Clearly the last row is redundant as $x_{FX} = u_{FX}$, so we can eliminate the last row and first column. After swapping the two middle columns, the optimality condition is

$$\begin{bmatrix} A_{FX}^{\intercal} & H_{OD} & I \\ A_{FR}^{\intercal} & H_{FR} & 0 \\ 0 & A_{FR} & 0 \end{bmatrix} \begin{pmatrix} \lambda \\ x_{FR} \\ \mu \end{pmatrix} = \begin{pmatrix} -q_{FX} - H_{FX}x_{FX} \\ -q_{FR} - H_{OD}x_{FX} \\ b - A_{FX}x_{FX} \end{pmatrix}.$$

This is a $(n+m)$ linear system which is much larger than the $(n-m)$ reduced Hessian system in (2.11), but it is very sparse. Another advantage is that each time the active set changes, we only need to perform simple row permutation and only 2 out of the last n columns of the basis change. The LU factors can therefore be updated easily as in the simplex method.

One common criticism for active set method for large scale optimization is that many iterations is required if the starting active set is significantly different from the optimal one. To alleviate this problem, Goldfarb [40] proposed a method for deleting more than one constraints in each iteration.

Bound constrained QP

When only bound constraints are present, significant savings can be obtained. See [33, 38, 86] for detailed discussions. The projection gradient algorithm which allow for large changes in the active set can be especially beneficial for large scale bound constrained problem. See [5, 86].

2.2.2 Interior Point Methods

Interior point methods for QP follow closely those for LP. Following the derivation of (2.7), the reduced KT system can be expressed as

$$\begin{bmatrix} -(X^{-1}Z + H) & A^{\intercal} \\ A & Y^{-1}W \end{bmatrix} \begin{bmatrix} \triangle x \\ \triangle y \end{bmatrix} = \begin{pmatrix} c - A^{\intercal}y + \mu X^{-1}e + Hx \\ b - Ax - \mu Y^{-1}e \end{pmatrix}. \qquad (2.13)$$

The primal and dual normal equations are respectively

$$(Y^{-1}W + A(X^{-1}Z + H)^{-1}A^{\intercal})\triangle y \\ = b - Ax + \mu Y^{-1}e + A(X^{-1}Z + H)^{-1}(c - A^{\intercal}y + Hx - \mu X^{-1}e)$$

and

$$-(A^{\intercal}YW^{-1}A+X^{-1}Z+H)\triangle x = c-A^{\intercal}y+Hx-\mu X^{-1}e-A^{\intercal}YW^{-1}(b-Ax+\mu Y^{-1}e).$$

The primal normal equation has coefficient matrix $A(X^{-1}Z + H)^{-1}A^{\intercal}$ which is likely to have significant fill-in due to the inverse term. The dual form has sparsity pattern $A^{\intercal}YW^{-1}A + X^{-1}Z + H$ which is far more likely to be sparse unless A has dense columns. The system of equation (2.13) can be solved in exactly the same way as for LP. See for example [124] for more detailed discussion and examples of implementation.

2.3 Convex Nonsmooth Programs

This section gives an overview of algorithms for solving convex nonsmooth programs. The first part of this section is devoted to studying the differential properties of nonsmooth functions. This is followed by a review of subgradient methods, bundle methods for nonsmooth functions and lastly methods for minimizing LC^1 functions and Lipschitz continuous functions.

2.3.1 Lipschitz Functions, Subgradients and Generalized Hessians

This section summarizes some useful definitions and properties of convex Lipschitz functions. Details can be found in, for example, [53, 101].

A convex function f is *proper* if $f(x) > -\infty$ for all $x \in \mathbb{R}^n$ and there exists $\bar{x} \in \mathbb{R}^n$ such that $f(\bar{x}) < +\infty$. This means that f is proper if and only if dom f is nonempty and the restriction of f to dom f is finite. We will assume in the sequel that all convex functions are proper unless otherwise specified.

Let f be any function from $\mathbb{R}^n \to [-\infty, +\infty]$ and let $\bar{x}$ be a point such that $f(\bar{x})$ is finite. The *directional derivative* of f at $\bar{x}$ along a direction h is

$$f'(\bar{x}, h) = \lim_{\alpha \downarrow 0} \frac{f(\bar{x} + \alpha h) - f(\bar{x})}{\alpha}$$

if the limit exists ($+\infty$ and $-\infty$ are allowed as limits). The directional derivative of a convex function exists everywhere.

The *subdifferential* of a convex function f is

$$\partial f(\bar{x}) = \{v : f(\bar{x} + h) \geq f(\bar{x}) + v^{\mathsf{T}} h, \forall h \in \mathbb{R}^n\}.$$

Each element $v \in \partial f(\bar{x})$ is called a subgradient. For a proper convex function f, $\partial f(x)$ is empty if $x \notin \text{dom } f$ and $\partial f(x)$ is nonempty and bounded if and only if $x \in \text{int}(\text{dom } f)$.

The subdifferential and directional derivative of a convex function can be further characterized by

$$f'(\bar{x}, h) = \max_{v \in \partial f(\bar{x})} v^{\mathsf{T}} h \tag{2.14}$$

and

$$\partial f(\bar{x}) = \text{co} \left\{ \lim_{\substack{x \to \bar{x} \\ x \in D_f}} \nabla f(x) \right\},$$

where *co* denotes the convex hull and $D_f \subseteq \mathbb{R}^n$ is the set where f is differentiable.

The *ϵ-subdifferential* generalizes the subdifferential. It is defined as

$$\partial_\epsilon f(\bar{x}) = \{v : f(\bar{x} + h) \leq f(\bar{x}) + v^{\mathsf{T}} h + \epsilon, \forall h \in \mathbb{R}^n\}$$

for $\epsilon \geq 0$.

For a convex function $f(x) : \mathbb{R}^n \to \mathbb{R}$, its *conjugate* $f^*(x^*)$ is defined as

$$f^*(x^*) = \sup_{x \in \mathbb{R}^n} x^{\mathsf{T}} x^* - f(x).$$

The following results on convex conjugate functions are required in the sequel.

[P1] $f^*(x^*)$ is convex, closed and proper [101, Theorem 12.2].

[P2] If f is piecewise quadratic, then so is f^* [117,118, Proposition 2.1.7]. (Piecewise quadratic functions will be defined precisely in Section 3.2.)

[P3] If f is strictly convex, then int dom $f^* \neq \emptyset$ and f^* is continuously differentiable on int dom f^* [53, Theorem X4.1.1].

If f is also 1-coercive, that is f satisfies $\lim_{\|x\|\to+\infty} \frac{f(x)}{\|x\|} = +\infty$, then the following properties are also true:

[P4] If f is strictly convex, continuously differentiable and 1-coercive, then f^* is finite-valued on $\mathbb{R}^n$, strictly convex, continuously differentiable and 1-coercive [53, Corollary X4.1.4a].

[P5] If f is 1-coercive, then dom $f^* = \mathbb{R}^n$ [53, p.89].

A function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ is said to be *Lipschitzian* relative to X if there exists $L \geq 0$ such that

$$|f(y) - f(x)| \leq L\|y - x\|, \quad \forall x, y \in X.$$

By Rademacher's theorem, a Lipschitz function is differentiable almost everywhere, (see for example, [19]).

A function f is called LC^1 if f is continuously differentiable and ∇f is Lipschitz. An LC^1 function is twice continuously differentiable almost everywhere. Let D_f^2 be the set where f is twice continuously differentiable, Pang and Qi [95,97] introduced the *B-derivative*

$$\mathcal{G}_B(\bar{x}) = \left\{ \lim_{\substack{x\to\bar{x} \\ x\in D_f^2}} \nabla^2 f(x) \right\}.$$

This is the set of *generators* of the generalized Hessian $\mathcal{G}$ of f earlier proposed by Clarke [19]. The *generalized Hessian* $\mathcal{G}$ is the generalized Jacobian of ∇f and is given by

$$\mathcal{G}(x) = \mathrm{co}\, \mathcal{G}_B(x).$$

∇f is said to be *strongly BD-regular* [95,97] at x if all $G \in \mathcal{G}_B(x)$ are nonsingular.

A function $F : X \subseteq \mathbb{R}^n \to \mathbb{R}^m$ is *semismooth* [82,99] at $x \in \mathbb{R}^n$ if F is Lipschitz continuous in an open neighbourhood of x and the limit

$$\lim_{\substack{V\in\partial F(x+\tau h) \\ \tau\downarrow 0}} Vh$$

exists for all $h \in \mathbb{R}^n$, where $\partial F(y)$ is the Clarke's generalized Jacobian of F at y. Proposition 1 of [96] states that a continuous piecewise differentiable function is semismooth. An LC^1 function f defined on $X \subseteq \mathbb{R}^n$ is SC^1 at x if $\nabla f(x)$ is semismooth.

2.3.2 Subgradient Methods

Subgradient methods were the first practical methods for solving nonsmooth optimization problems. In their most basic form, they can be seen as nonsmooth versions of steepest descent methods. Consider the problem

$$\min_{x \in X} f(x)$$

each iteration of a subgradient method is given by $x^{i+1} = x^i + \rho^i d^i$ where ρ^i is an appropriate step, the search direction is $d^i = -g^i/\|g^i\|$ for some $g^i \in \partial f(x^i)$. Since only one subgradient is used at each iteration, d^i is not necessarily a descent direction and it is difficult to devise a practical stopping criterion. As d^i may not be a descent direction, steps have to be determined a priori. The method is globally convergent if the steps ρ^i satisfy

$$\rho^i > 0, \quad \lim_{i \to \infty} \rho^i = 0, \quad \sum_{i=1}^{\infty} \rho^i = \infty.$$

Subgradient methods have been modified with space dilation to improve the convergence rate. This is similar to quasi-Newton methods for smooth optimization. The search direction is given by

$$d^i = -\frac{H^i g^i}{(g^i)^\intercal H^i g^i}$$

where H^i is some nonsingular matrix. See [65, 111] for detailed discussions of subgradient methods.

2.3.3 Bundle Methods

Consider the convex nonsmooth program

$$\min_{x \in \mathbb{R}^n} f(x).$$

Suppose at each point $x \in \mathbb{R}^n$, a subgradient $g(x) \in \partial f(x)$ is available. One way to solve the problem is by forming a polyhedral approximation of $f(x)$ at each iteration i using previously obtained trial points x^l, and their subgradients $g(x^l), l = 1, \ldots, i-1$, namely

$$f(x) \geq \hat{f}(x) \equiv \max_{l<i} \left\{ f(x^l) + g(x^l)^\intercal (x - x^l) \right\}.$$

However, minimizing $\hat{f}(x)$ on $\mathbb{R}^n$ may be unbounded, therefore the basic cutting plane method [18, 63] obtains the next trial point by minimizing $\hat{f}(x)$ on some compact set C where C is known to contain the optimal solution. The function value and a subgradient is then calculated at x^i and the algorithm repeats itself until $f(x^i) \leq \hat{f}(x^i) + \epsilon$ for some given stopping tolerance $\epsilon \geq 0$. The above algorithm is globally convergent, however, it is unstable as iterates can vary wildly as cuts change the approximation and convergence can be very slow.

Suppose the set of subgradients (called a *bundle*) are calculated at some auxiliary sequence y^l, $l \leq i$ which can be thought of as some set similar to the sequence of iterates $\{x^l\}$. At each iterate x^i, the polyhedral approximation of f can be written as

$$\hat{f}(x) = \max_{l \leq i} \left\{ f(y^l) + g_l^\intercal (x - y^l) \right\}. \tag{2.15}$$

Define the *linearization error*

$$\alpha(x^i, y^l) = f(x^i) - f(y^l) - g_l^\intercal (x^i - y^l). \tag{2.16}$$

Substituting (2.16) into (2.15) gives

$$\hat{f}(x) = f(x^i) + \max_{l \leq i} \{ g_l^\intercal (x - x^i) - \alpha(x^i, y^l) \}. \tag{2.17}$$

Observe that

$$\begin{aligned} \alpha(x^j, y^l) &= f(x^j) - f(y^l) - g_l^\intercal (x^j - y^l) \\ &= f(x^i) - f(y^l) - g_l^\intercal (x^i - y^l) + f(x^j) - f(x^i) + g_l^\intercal (x^i - x^j) \\ &= \alpha(x^i, y^l) + f(x^j) - f(x^i) - g_l^\intercal (x^i - x^j). \end{aligned}$$

Therefore the linearization error can be updated easily without storing the points y^l where each g_l was first derived and $\alpha(x^i, y^l)$ can be denoted by α_l^i without any loss of information.

Denote the search direction by d. Since $\hat{f}(x)$ may not be a good approximation of the objective far away from x^i, a stabilizing term $\frac{1}{2}d^\mathsf{T}d$ is added to $\hat{f}$ to form the search direction finding subproblem.

$$\min_{d\in\mathbb{R}^n}\left\{\max_{l\le i} g_l^\mathsf{T}d - \alpha_l^i + \tfrac{1}{2}d^\mathsf{T}d\right\}. \tag{2.18}$$

This is a strictly convex problem and so has a unique solution. This removes the need for the arbitrary compact set C.

As we do not have full information about $\partial f(x^i)$, d^i may not be a descent direction. We need to distinguish two types of iterates.

1. If a positive step ρ^i can be found such that $f(x^i + \rho^i d^i)$ is sufficiently smaller than $f(x^i)$, then set $x^{i+1} = x^i + \rho^i d^i$ and $y^{i+1} = x^{i+1}$. This step is called a *serious step*.

2. Otherwise, set $x^{i+1} = x^i$ and $y^{i+1} = x^i + \rho d^i$ for some appropriate positive ρ such that $g_{i+1} \in \partial f(y^{i+1})$ is sufficiently different from the current bundle so that a better direction can be found in the next iteration. This step is termed a *null step* since the iterate x is not updated.

An algorithm of this type is given by Mifflin [83]. Let $0 < m_L < m_R < 1$ and $m_L < \frac{1}{2}$.

Algorithm 2.1

Step 0 Find $x^1 \in \mathbb{R}^n$, set $y^1 = x^1$ and $i = 1$. Calculate $g_1 \in \partial f(y^1)$.

Step 1 Solve

$$\begin{aligned} \min_{u,d} \quad & \tfrac{1}{2}d^\mathsf{T}G^i d + u \\ \text{s.t.} \quad & -\alpha_l^i + g_l^\mathsf{T}d \le u, \quad l = 1, \ldots, i \end{aligned} \tag{2.19}$$

for d and u. If $d = 0$, stop, x^i is optimal.

Step2 Linesearch

Calculate two steps $0 \le \rho_L \le \rho_R$ such that $x^{i+1} = x^i + \rho_L d^i$ and $y^{i+1} = x^i + \rho_R d^i$ satisfy

$$f(x^{i+1}) \le f(x^i) + m_L \rho_L u^i,$$

and

$$-\alpha_{i+1}^{i+1} + g_{i+1}^\intercal d^i \le m_R u^i.$$

Goto Step 1.

In Step 1, G^i are positive definite matrices. Mifflin [83] proved that the above algorithm is globally convergent.

However, the above algorithm is not practical as it requires storing all subgradients. One solution is the subgradient selection method [66] which uses results from generalized cutting plane method. In each iteration, it is only necessary to keep those subgradients whose corresponding Lagrange multiplier are positive. Since at most $n+1$ dual variables can be positive, the amount of storage required is bounded. The resulting algorithm is still globally convergent as only those subgradients with positive Lagrange multiplier are necessary in defining the current model of f. Therefore adding a new subgradient to this set is guaranteed to provide a better model of f. See Kiwiel [65].

If the number of variables is large, then the subgradient selection approach may still require too much work in each direction finding problem. To see how global convergence can be achieved with fewer cuts, we need to introduce an *aggregate subgradient*. Recall the definition of g_l as that

$$f(x) \ge f(y^l) + g_l^\intercal (x - y^l) \quad \text{for all } x \in \mathbb{R}^n.$$

Combine this with the linearization error (2.16) to give

$$\begin{aligned} f(x) &\ge f(x^i) + g_l^\intercal (x - x^i) - f(x^i) + f(y^l) + g_l^\intercal (x^i - y^l) \\ &= f(x^i) + g_l^\intercal (x - x^i) - \alpha(x^i, y^l). \end{aligned} \tag{2.20}$$

This shows that g_l is in the ϵ-subdifferential of f at x^i, $g_l \in \partial_\epsilon f(x^i)$ for $\epsilon = \alpha(x^i, y^l)$. As we will show in Lemma 5.3, the optimality condition of the ith subproblem (2.19)

requires that the dual variables $\lambda_l^i, l = 1, \ldots, i$ satisfy

$$\sum_{1 \le l \le i} \lambda_l^i = 1, \quad \lambda^i \ge 0.$$

Define

$$\tilde{g}^i = \sum_{1 \le l \le i} \lambda_l^i g_l, \quad \tilde{\alpha} = \sum_{1 \le l \le i} \lambda_l^i \alpha_l^i. \tag{2.21}$$

The aggregate subgradient $\tilde{g}^i$ is a convex combination of ϵ-subgradients of f at x^i. It is easy to verify that $\tilde{g}^i \in \partial_\epsilon f(x^i)$ for $\epsilon = \tilde{\alpha}$ by taking a convex combination of (2.20).

Kiwiel [65] showed that the ith search direction problem (2.19) can be replaced by

$$\begin{array}{ll} \min\limits_{u,d} & \frac{1}{2} d^{\mathsf{T}} G^i d + u \\ \text{s.t.} & -\alpha(x^i, y^i) + g_i^{\mathsf{T}} d \le u, \\ & -\tilde{\alpha}^{i-1} + (\tilde{g}^{i-1})^{\mathsf{T}} d \le u, \end{array}$$

while still maintaining global convergence. Of course, more subgradients can and should be included to give a finer approximation of f to achieve faster convergence.

The search direction problem is usually solved in its dual form since the number of subgradients is generally kept much smaller than the number of variables.

A lot of work has been done on bundle type methods. A trust region method is often used to replace the linesearch procedure. This allows the trust region radius to be updated depending on how accurately the current model approximates f, and so speed up convergence, see for example [68, 109]. Research has also been carried out in generalizing techniques from smooth optimization to incorporate second order information into bundle methods to achieve better convergence rate. Examples of recent papers in this area include [67, 70, 84, 85].

2.3.4 LC^1 optimization

In LC^1 optimization, both the objective function and the constraints are continuously differentiable but their gradients are only Lipschitz continuous. Qi [98] and

Pang and Qi [96] proposed using an approximate Newton method for solving this class of problems. Consider the problem

$$\min_x \quad f(x) \tag{2.22}$$

$$\text{s.t.} \quad g(x) \leq 0. \tag{2.23}$$

Let u denotes the Lagrange multiplier for the inequality constraints (2.23). The algorithm can be stated as follow. Start at some feasible (x^0, u^0). At each iteration i, solve

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \nabla f(x^i)^\intercal (x - x^i) + \tfrac{1}{2}(x - x^i)^\intercal G^i (x - x^i) \\ \text{s.t.} \quad & g(x^i) + \nabla g(x^i)^\intercal (x - x^i) \leq 0, \end{aligned}$$

where G^i is symmetric positive semi-definite. If $z^{i+1} = (x^{i+1}, u^{i+1})$ is not unique, choose a KTT point z^{i+1} which is closest to z^i in the 2-norm.

Qi [98] proved that if at a KT point (x^*, u^*), the linear independence constraint qualification, the second order sufficiency condition and the strict complementarity condition are satisfied, if $\nabla f(x^*)$ and $\nabla g(x^*)$ are semismooth and if there exists $V^i \in \partial_B(\nabla f(x^i))$ such that

$$\lim_{i\to\infty} \frac{\|(G^i - V^i)(x^{i+1} - x^i)\|}{\|z^{i+1} - z^i\|} = 0,$$

then the algorithm converges superlinearly. See Section 3.1 for the definitions of the linear independence constraint qualification and the strict complementarity condition. For the LC^1 program (2.22), define

$$I(z) = \{j | 1 \leq j \leq m, g_j(x) = 0\}$$

$$I_1(z) = \{j \in I(z) | u_j > 0\}, \quad I_2(z) = \{j \in I(z) | u_j = 0\}$$

and

$$D(z) = \{d \in \mathbb{R}^n | f'(x, d) = 0, g_j'(x, d) = 0 \text{ for } j \in I_1(z), g_j'(x, d) \leq 0 \text{ for } j \in I_2(z)\}.$$

A point z is said to satisfy the second order sufficiency condition [98] for (2.22) if it satisfies the Kuhn-Tucker conditions and if $d^\intercal V d > 0$ for all $d \in D(z)\backslash 0$ and $V \in \partial_B F_u$ where $F_u = \nabla f(x) + u^\intercal \nabla g(x)$.

If the constraints $g(x)$ are linear and a linesearch, such as an Armijo linesearch, is used to guarantee sufficient function value decrease in each iteration, then global convergence can also be achieved by assuming in addition that G^i are positive definite and bounded [98].

Han and Sun [47] improved on the superlinear convergence result of Qi [98] by replacing the linear independence constraint qualification and second order sufficiency condition assumptions at the solution point with the strong second order sufficiency condition.

2.3.5 Minimizing Lipschitz Functions

Pang et al. [94] proposed an algorithm to solve

$$\min_{x\in\mathbb{R}^n} f(x), \tag{2.24}$$

where $f(x)$ is locally Lipschitz but not necessarily convex. As f is not assumed to be convex, the directional derivative $f'(x,d)$ may not exist everywhere. A more general derivative concept is the *upper Dini directional derivative* of f at x in the direction of d, defined as

$$f^D(x,d) = \limsup_{\lambda\downarrow 0} \frac{f(x+\lambda d) - f(x)}{\lambda}. \tag{2.25}$$

The upper Dini directional derivative is everywhere defined if f is locally Lipschitz and it is equal to the usual directional derivative if $f'(x,d)$ exists. A point $x \in \mathbb{R}^n$ is called a Dini stationary point if for all $d \in \mathbb{R}^n$, $f^D(x,d) \geq 0$.

At each iterate x^i, the search direction subproblem is given by

$$\min_d \phi(x^i,d) + \tfrac{1}{2} d^\intercal B^i d, \tag{2.26}$$

where $B^i \in \mathbb{R}^{n\times n}$ is symmetric positive definite and contains second order information of f similar to a quasi-Newton method. The matrices $\{B^i\}$ are assumed to satisfy $c_1 x^\intercal x \leq x^\intercal B^i x \leq c_2 x^\intercal x$ for all i for constants $0 < c_1 < c_2$. The iteration function $\phi(x^i,d)$ is a linear approximation of f at x^i. It is assumed to be continuous in d for all x and $\phi(x,0) = 0$ for all x. $\phi(x^i,d)$ needs not be continuous in x since

the Dini directional derivative may be discontinuous in x. The iteration function $\phi(x^i, d)$ should also satisfy

$$\phi(x, d) \geq f^D(x, d) \text{ for all } (x, d) \in \mathbb{R}^n \times \mathbb{R}^n.$$

This condition and the positive definiteness of B^i guarantee that the solution to (2.26) is a descent direction at x^i.

Given linesearch parameters $\rho, \sigma \in (0, 1)$ and a stopping tolerance $\epsilon \geq 0$, the algorithm of Pang et al. [94] can be stated as

Algorithm 2.2

Step 0 Choose initial point $x^0 \in \mathbb{R}^n$. Set $i = 0$.

Step 1 Solve (2.26) for d^i.

Step 2 If $\|d^i\| = 0$, then stop, x^i is a Dini stationary point.

Step 3 Armijo linesearch: let $m_k \geq 0$ be the smallest integer that satisfies

$$f(x^i + \rho^m d^i) - f(x^i) \leq -\frac{\sigma}{2}\rho^m (d^i)^\top B^i d^i.$$

Set $x^{i+1} = x^i + \rho^{m_k} d^i$ and $i = i + 1$. Return to Step 1.

Under some technical assumptions on $\phi(x, d)$ at the limit point $\bar{x}$, the algorithm is proven to converge globally [94].

Qi and Sun [100] proposed another algorithm that also solves (2.24). Instead of using a linesearch, convergence is enforced by a trust region method. Once again, an iteration function is used for first order approximation of the objective. It satisfies $\phi(x, 0) = 0$ and $\phi(x, d)$ is lower semi-continuous for all $x \in \mathbb{R}^n$ and for all convergent sequences x^i

$$f(x^i + d^i) - f(x^i) \leq \phi(x^i, d^i) + o(\|d^i\|)$$

if $d^i \to 0$.

Given positive constants $c_0 \leq 1, c_2 < c_1 < 1, c_3 < 1 < c_4$. Their algorithm is

Algorithm 2.3

Step 0 Choose initial trust region radius $r_0 > 0$ and initial point x^0.

Step 1 Solve the search direction subproblem

$$\begin{aligned} Q^i(d^i) \equiv \min \quad & f(x^i) + \phi(x^i, d) + \tfrac{1}{2} d^T B^i d \\ \text{s.t.} \quad & \|d\| \leq r^i. \end{aligned}$$

If $\|d^i\| = 0$, then stop, x^i is a Dini stationary point of f.

Step 2 Let

$$a_k = \frac{f(x^i) - f(x^i + d^i)}{f(x^i) - Q^i(d^i)}.$$

Update

$$x^{i+1} = \begin{cases} x^i + d^i, & \text{if } a_k > c_2, \\ x^i, & \text{otherwise,} \end{cases}$$

$$r^{i+1} = \begin{cases} c_3 r^i, & \text{if } a_k \leq c_2, \\ r^i, & \text{if } c_2 < a_k \leq c_1, \\ \min\{c_4 r^i, r_0\}, & \text{otherwise,} \end{cases}$$

Goto Step 1.

Under some technical assumptions on $\phi(x, d)$, the algorithm is shown to be globally convergent [100]. Since convergence is enforced via the trust region radius updating, it is not necessary to have B^i positive definite.

For both Algorithms 2.2 and 2.3, if the objective is directionally differentiable, then $f'(x, d)$ satisfies all the required assumptions on $\phi(x, d)$.

2.4 Primal Decomposition Methods for Stochastic Programs

For multistage stochastic programs with hundreds or thousands of scenarios, the deterministic equivalents are very large and may require too much memory to be

solved efficiently. Decomposition algorithms break the problems down into smaller subproblems and therefore greatly reduce memory requirements. Similarity among different subproblems can also be exploited to produce more efficient algorithms. See the recent textbooks on stochastic programming [10, 62] for excellent coverage on decomposition methods and other topics on stochastic programming.

Primal decomposition methods break the stochastic program into smaller subproblems corresponding to the different time stages. This class of algorithms solves the individual time stage problems. At each node, the current solution is passed to the appropriate descendent nodes to form problems (1.13). Feasibility and optimality information is passed back to the ancestor problem to obtain a new trial point.

Decomposition methods are well suited to parallelization as subproblems can be solved independently on different processors. The main communication needed for primal decomposition is to broadcast the current solution to processors handling descendent problems and to pass feasibility and optimality information back to form a new current stage program. Depending on the ratio of time spent in each processor to the communications time and the efficiency of the communications network, varying levels of speed-up are possible.

Most work has been done on two stage stochastic linear programs (SLP-2) with fixed recourse. Many primal decomposition methods take advantage of the fixed feasibility set of the dual of the second stage subproblem induced by fixed recourse.

2.4.1 L-shaped Method

The earliest attempt to solve stochastic programs by decomposition is the L-shaped method of Van Slyke and Wets [121]. It is designed for two stage stochastic linear programs (1.3) with discrete random variables with a finite number of realizations. The L-shaped method exploits the structure of the constraint matrix in the large scale equivalent LP (1.14) by performing a Dantzig-Wolfe decomposition [24] on the dual problem or a Benders decomposition [3] on the primal. The result is a cutting plane method which builds an outer linearization of the recourse cost function. It

proceeds by first solving the first-stage (master) problem (1.3). For each realization $k, k = 1, \ldots, K$ of the stochastic parameters, (1.4) is tested for feasibility with respect to the current master solution. If an infeasible subproblem is encountered, a feasibility cut is generated. The cut is placed in the master problem in the form of an inequality constraint (2.27c), and the master problem is solved again to give a new first stage solution. Otherwise problems (1.4) are solved and if the current first stage solution is not optimal, an optimality cut (2.27d) is constructed and passed to the master problem.

Algorithm 2.4 *L-shaped Method*

Step 0 Set $r, s, \nu = 0$.

Step 1 Set $\nu = \nu + 1$. Solve the modified master problem

$$\begin{aligned} \min \quad & c^\intercal x + \theta && \text{(2.27a)} \\ \text{s.t.} \quad & Ax = b, && \text{(2.27b)} \\ & D_l x \geq d_l, \quad l = 1, \ldots, r, && \text{(2.27c)} \\ & E_l x + \theta \geq e_l, \quad l = 1, \ldots, s, && \text{(2.27d)} \\ & x \geq 0, && \text{(2.27e)} \end{aligned}$$

for (x^ν, θ^ν).

Step 2 For $k = 1, \ldots, K$

Solve the stage two feasibility problem

$$\begin{aligned} \min \quad & w = e^\intercal v^+ + e^\intercal v^- \\ \text{s.t.} \quad & Wy + v^+ - v^- = h_k - V_k x^\nu, \qquad (2.28) \\ & y, v^+, v^- \geq 0. \end{aligned}$$

If $w > 0$,

Let σ be the Lagrange multipliers, set $r = r + 1$,

Set $D_r = \sigma^\intercal V_k, \quad d_r = \sigma^\intercal h_k$.

Return to Step 1.

If $w = 0$ for all $k = 1, \ldots, K$, goto Step 3.

Step 3 For $k = 1, \ldots, K$ solve the stage two problem

$$\begin{aligned} \mathcal{Q}_k(x^\nu) = \min \quad & q_k^\intercal y \\ \text{s.t.} \quad & Wy = h_k - V_k x^\nu, \\ & y \geq 0. \end{aligned} \tag{2.29}$$

Let the Lagrange multiplier be π_k.

Set $s = s + 1$. $E_s = \sum_{k=1}^K p_k \pi_k^\intercal V_k, \quad e_s = \sum_{k=1}^K p_k \pi_k^\intercal h_k$.

If $\theta^\nu \geq e_s - E_s x^\nu$

x^ν is the optimal solution, STOP.

Else return to Step 1.

Feasibility Cuts

We now show how feasibility cut (2.27c) is derived. At a first stage solution x^ν, the feasible set for the stage two problem is $\{y | Wy = h_k - V_k x^\nu, y \geq 0\}$. Let pos $W = \{u | \exists y \geq 0, Wy = u\}$. Then a constraint right hand side $h_k - V_k x^\nu$ is feasible if and only if $h_k - V_k x^\nu \in$ pos W. If it is infeasible, then there must exist a hyperplane which separate $h_k - V_k x^\nu$ from pos W, that is a σ which satisfies $\sigma^\intercal(h_k - V_k x^\nu) > 0$ and $\sigma^\intercal u \leq 0$ for all $u \in$ pos W. This is satisfied by the dual variables of (2.28) as we will now demonstrate. The dual problem of (2.28) is

$$\begin{aligned} \max_\sigma \quad & \sigma^\intercal(h_k - V_k x^\nu) \\ \text{s.t.} \quad & \begin{bmatrix} W^\intercal \\ I \\ -I \end{bmatrix} \sigma \leq \begin{pmatrix} 0 \\ e \\ e \end{pmatrix}. \end{aligned} \tag{2.30}$$

Since $h_k - V_k x^\nu$ is assumed to be infeasible, the optimal value of (2.28) is strictly positive. By duality, the optimal values of the primal and dual problems are the same, that is $\sigma^\intercal(h_k - V_k x^\nu) > 0$ or $\sigma^\intercal h_k > \sigma^\intercal T_k x^\nu$. Clearly, x^ν does not satisfy the new cut (2.27c). From the first constraint of (2.30), σ satisfies $W^\intercal \sigma \leq 0$, that is $\sigma^\intercal W y \leq 0$ for all $y \geq 0$ or $\sigma^\intercal u \leq 0$ for all $u \in$ pos W. So σ satisfies the requirement of a separating hyperplane and (2.27c) only cuts off x that leads to infeasible stage two problems.

As each cut removes the current x, a new feasibility cut is created each time. Since problem (2.28) has a finite number of optimal bases, there can only be a finite number of feasibility cuts.

Optimality Cuts

Let π_k denote the dual variables of (2.29), so by duality,

$$\pi_k^\intercal(h_k - V_k x^\nu) = \mathcal{Q}_k(x^\nu).$$

Since $\mathcal{Q}_k$ is convex,

$$\begin{aligned} \mathcal{Q}_k(x) &\geq \mathcal{Q}_k(x^\nu) + \nabla\mathcal{Q}_k(x^\nu)^\intercal(x - x^\nu) \\ &= \mathcal{Q}_k(x^\nu) - \pi_k^\intercal V_k(x - x^\nu). \end{aligned}$$

Combining the last two equations gives

$$\mathcal{Q}_k(x) \geq \pi_k^\intercal h_k - \pi_k^\intercal V_k x.$$

Taking the expected value over all realizations gives

$$Q(x) = \sum_{k=1}^{K} \mathcal{Q}_k(x) \geq \sum_{k=1}^{K} p_k \pi_k^\intercal h_k - \sum_{k=1}^{K} p_k \pi_k^\intercal V_k x.$$

The master problem (2.27) is feasible for (x^ν, θ^ν) if $\theta^\nu \geq e_l - E_l x^\nu = Q(x^\nu)$. Since θ is unrestricted otherwise, the above holds as an equality. In this case, the algorithm terminates since x^ν minimizes the sum of the first stage problem and the linear lower bound of the second stage problem and the lower bound is exact at x^ν. If the new cut is not satisfied by (x^ν, θ^ν), then clearly the new cut is different from all previous cuts since none before excluded x^ν. As there are a finite number of optimal bases to problem (2.29), there can only be finitely many optimality cuts.

As a new cut is created in each iteration and there are finitely many feasibility and optimality cuts, the L-shaped method is finitely convergent. The drawback with this approach is that the number of cuts continues to grow as the algorithm proceed. There may be numerical difficulties if the gradients of the constraints become almost linearly dependent. The first few iterations are generally not efficient

as the approximation to the recourse function is very poor. As the early iterates of the cutting plane method tend to fluctuate wildly, the algorithm cannot make use of a good starting point even if one is available.

Parallel L-shaped Method

The L-shaped method can be parallelized naturally by assigning subsets of second stage problems to different processors. There may be significant idle time for slave processors if the master problem is time-consuming. There is also likely to be a communications bottleneck when all subproblems are trying to return cut information to the master processor. See for example [93] for a recent implementation.

Multi-cut L-shaped Method

The original L-shaped method creates only one optimality cut per iteration using the expected value from all scenario subproblems. It is possible to disaggregate it to form one cut per scenario or one for each subset of scenarios. The multi-cut version [7] leads to a larger master problem with more variables and constraints but it is expected to reduce the number of iterations required as the recourse function is approximated more finely. Computational experience in [87] suggests that the multi-cut approach reduces overall solution time.

Algorithm 2.5 *Multi-cut L-shaped Method*

Step 0 Set $r, \nu = 0, s_k = 0$ for $k = 1, \ldots, K$.

Step 1 Set $\nu = \nu + 1$. Solve the modified master problem

$$\begin{aligned} \min \quad & c^{\mathsf{T}} x + \sum_{k=1}^{K} \theta_k \\ \text{s.t.} \quad & Ax = b \\ & D_l x \geq d_l, \quad l = 1, \ldots, r \\ & E_{l_k} x + \theta_k \geq e_{l_k}, \quad l_k = 1, \ldots, s_k, k = 1, \ldots, K \\ & x \geq 0 \end{aligned}$$

for $(x^\nu, \theta_1^\nu, \ldots, \theta_K^\nu)$.

Step 2 As for Algorithm 2.4.

Step 3 For $k = 1, \ldots, K$ solve the stage two problems (1.4). Let π_k be the Lagrange multipliers.

If $\theta_k^\nu < p_k \pi_k^\intercal (h_k - V_k x^\nu)$

Set $s_k = s_k + 1$,

$E_{s_k} = p_k \pi_k^\intercal V_k, \quad e_{s_k} = p_k \pi_k^\intercal h_k$.

If $\theta_k^\nu \geq p_k \pi_k^\intercal (h_k - V_k x^\nu)$ for all $k = 1, \ldots, K$

x^ν is the optimal solution, STOP.

Else return to Step 1.

Bunching and Sifting

As we have to solve all the second stage problems (1.4) in each iteration at step 3 to obtain the dual variables, it is important to do it efficiently. For subproblems with fixed recourse, that is if W is fixed, a technique called bunching can be used. Bunching exploits the similarity of the recourse problems between scenarios by finding groups of realizations (called bunches) that share a common optimal basis. Let B be the optimal basis of W for some particular right hand side h_k and a subscript B denote components corresponding to the basic variables. Then $\pi_k = B^{-\intercal} q_{kB}$, $q_k - W^\intercal \pi_k \geq 0$ (optimality) and $B^{-1}(h_k - V_k x^\nu) \geq 0$ (feasibility). Assuming q is deterministic, define a bunch as all the right hand sides that are also feasible at this optimal basis, namely

$$\{h_k - V_k x^\nu | B^{-1}(h_k - V_k x^\nu) \geq 0, k = 1, \ldots, K\}.$$

Then nodes in the same bunch share the same dual variables. The optimal bases for other realizations can be obtained quickly from bases in previous bunches using dual simplex steps. This can significantly improve solution time if there are many scenarios sharing the same or similar bases.

Sifting is a similar technique requiring the right hand side to have component-wise independence in addition. For more general subproblems, warm starts from the optimal bases of other scenarios can also reduce solution time. See [10, 62] and

references therein for detailed discussions on bunching, sifting and other efficiency techniques.

2.4.2 Nested Decomposition

Nested decomposition [39, 54] extends the L-shaped method to solve multistage stochastic linear programming problems. It also assumes the stochastic parameters have a finite number of realizations. As the name suggests, nested decomposition solves the first stage problem and treats the remaining stages as another multistage stochastic program and solves them recursively. As the algorithm progresses, current solutions are passed forward to later stages as input to the right hand side (called forward passes), while information is passed back from later stages in the form of feasibility and optimality cuts to the ancestor problems (called backward passes). There are various schemes which determine the direction to proceed at an intermediate stage. In general, the fast forward fast backward approach seems to be the most efficient. This scheme suggests that the algorithm should proceed in the same direction as far as possible until an infeasible subproblem is found (when going forward) or when there are no more cuts to pass back (when going back). Information exchange among all stages is maximized in this scheme. See for example [87] for a description of these and other tree traversing schemes and a comparison of their performance.

As the number of subproblems increases rapidly with a longer planning horizon, efficiency schemes like bunching are even more important. Morton [87] suggested keeping a candidate list of bases when repeated optimal bases are unlikely. The technique was designed for systems with a large network component. In the first major iteration, the network flow is calculated using the candidate bases first and a feasible solution for the whole subproblem can then be calculated quickly from the network solution. The candidate with the minimum objective value is chosen as a starting basis. For subsequent major iterations, previous optimal bases from the same subproblem are used.

As with the L-shaped method, the initial iterations of the nested decomposition

are very inefficient and may result in infeasible input for later stages. It is helpful to be able to calculate preliminary cuts before starting the algorithm. If the stochastic parameters exhibit interstage independence and they only influence the right hand side of the nodal problems (h_t and V_t in (1.13)), then the expected value problem can be solved first and the feasibility and optimality cuts generated are valid for the original problem. If stochastic parameters across the stages are dependent, the cut sharing scheme of [59, 87] can be used to create preliminary cuts. This scheme looks at a single scenario at each stage and solves its descendents. The cuts created are passed back to the ancestor and modified using the cut sharing formula before passing to the ancestor's siblings. Prespecified solutions to all nodes are required which may be obtained by solving the expected value problem. It is suggested in [87] that more than one set of prespecified solutions, for example, three well spaced values within the lower and upper bounds of the variables, are used. The advanced cuts should be calculated in a backward order so that the maximum amount of information is passed back. Computational experiments in [87] suggest significant improvement in solution time on multistage stochastic linear programs with a large network component.

Parallelization of the nested decomposition method follows analogously from the L-shaped method. Sets of scenarios can be assigned to processors to create feasibility or optimality cuts simultaneously. Ruszczyński [107] suggested a different way to parallelize the nested decomposition algorithm. All subproblems are solved simultaneously to generate new cuts and current solutions. This new information is then passed to a special buffer so that the tasks can be performed asynchronously. This scheme significantly reduces communication bottlenecks and idle time in processors waiting for new information to come in. More information is generated at the same time, though it is possible that some processors may be using outdated information. Scheduling of processes will be important. Good speedup is reported [107] over serial nested decomposition, with speedups larger for longer time periods and more branches. The speedup was close to linear if the number of processors was small compared to the number of nodes in the scenario tree. See also [9] for another parallel implementation.

2.4.3 Regularized Decomposition

The L-shaped method is essentially a cutting plane algorithm and suffers from two major disadvantages: (1) the first few iterates are very unstable as the feasible set and the recourse function are poorly approximated, so the algorithm cannot take advantage of a good initial solution; (2) the number of cuts increases continuously as the algorithm proceeds, with the increase being more significant for the multicut version. Regularized decomposition [106, 108] generalizes the L-shaped method to address these two problems. In two stage stochastic linear programs (1.3)–(1.4), a convex proximal term $\sigma\|x-\hat{x}\|^2$ is added to the first stage objective function, where $\sigma > 0$ is a weighting parameter and $\hat{x}$ is a candidate solution. The proximal term forces the objective function to become strictly convex and reduces the variability of the solution, so a good starting point can be used to speed up the solution process. It also allows slack cutting planes to be dropped so the size of the master problem is well controlled. The algorithm either converges or proves the problem is infeasible in a finite number of iterations [106, 108].

The set of constraints used in the master problem is termed the committee. They may include constraints from the original first stage problem, feasibility and optimality cuts. The multicut approach is preferred because it approximates the piecewise linear objective more finely than aggregated cuts and since the proximal term allows the dropping of slack constraints, and the numbers of committee members is always bounded above by $n_1 + 2K$, where K is the number of scenarios.

Regularized decomposition is a two stage algorithm. The first stage identifies a feasible starting point and uses the same algorithm as the second stage main algorithm. The committee is initialized to contain the first stage constraints. In the next stage, the program starts at a feasible point and the committee contains a combination of the original constraints and feasibility and optimality cuts from the first stage algorithm.

Algorithm 2.6

Step 0 Set $\nu = 0$. Initialize committee and a feasible $\hat{x}^1$.

Step 1 $\nu = \nu + 1$. Solve the master problem with candidate solution $\hat{x}^\nu$

$$\begin{aligned} \min \quad & \sigma\|x - \hat{x}^\nu\|^2 + c^\intercal x + p^\intercal \theta \\ \text{s.t.} \quad & x \text{ satisfies constraints in committee,} \end{aligned}$$

for (x^ν, θ^ν) where p is the probability vector of the stage two scenarios.
$\hat{f}^\nu = c^\intercal x^\nu + p^\intercal \theta^\nu$.
If $\hat{f}^\nu = f(\hat{x}^\nu)$, optimal solution found, STOP.

Step 2 Delete the committee members inactive at (x^ν, θ^ν) so that no more than $n+K$ members remain.

Step 3 If x^ν violates any of the first stage constraints,
add to the committee at most K violated constraints.
Set $\hat{x}^{\nu+1} = \hat{x}^\nu$, goto Step 1.

Step 4 Solve (1.4) at x^ν for $k = 1, \ldots, K$
If (1.4) is infeasible, append feasibility cut to the committee.
If $Q_k(x^\nu) > \theta_k^\nu$, append optimality cut to the committee.

Step 5 If all subproblems are solvable, goto Step 6.
Else set $\hat{x}^{\nu+1} = \hat{x}^\nu$, goto Step 1.

Step 6 If either $f(x^\nu) = \hat{f}^\nu$ or $f(x^\nu) \leq \lambda f(\hat{x}^\nu) + (1-\lambda)\hat{f}^\nu$ and exactly $n+K$ members are active at (x^ν, θ^ν), then set $\hat{x}^{\nu+1} = x^\nu$, else set $\hat{x}^{\nu+1} = \hat{x}^\nu$.

Regularized decomposition is an active set method. It selects the optimality cuts that are active at the solution. Based on this observation, a specialized active set method is developed in [106] for solving the master problem. The method reduces the size of the linear system so that it is bounded by the number of first stage variables. This makes it practicable to solve problems with a huge number of realizations.

Numerical experiments in [108] suggest that regularized decomposition is a very efficient method for two stage stochastic linear program. It outperforms the standard L-shaped method with either aggregate or multicuts, both in terms of speed and the size of problems that can be solved.

The regularized decomposition method can be extended to the multistage case by using a nested decomposition. A proximal term can be added to all the subproblems except the terminal nodes to stabilize the current solution.

2.4.4 Stochastic Decomposition and Importance Sampling

Stochastic decomposition and importance sampling generalize the L-shaped method for two stage stochastic linear programming. Instead of solving one subproblem for each scenario realization in every iteration, only a subset of realizations are sampled. This is useful for problems having a very large number of branches or when the stochastic parameters are distributed continuously. The resulting algorithms have fewer cuts than their deterministic cousins, and the computing time required is less sensitive to the number of scenarios. However, as not all possible realizations are sampled, the optimality cuts generated are only statistically valid, and convergence can only be proven in a statistical sense.

In stochastic decomposition [50, 52], the authors propose using only one sample per iteration. The piecewise linear approximation to the recourse function is a statistically valid lower bound. As only a very small number of samples will be tested in the algorithm, there is no guarantee of recourse feasibility and it is only applicable to problems with relatively complete recourse. Optimality is achieved statistically and it is possible to obtain statistical estimates of lower and upper bounds on the optimal value [49]. As only one sample is used in each iteration, the solution time is not very sensitive to increases in the number of scenarios, although increased variability means more iterations are needed before the iterates converge. This scheme has also been combined with regularized decomposition (see Section 2.4.3) to bound the number of optimality cuts in the master problem [51]. This method is not worth parallelizing if only one sample is tested per iteration. However it is straightforward to parallelize it across a small number of processors and test the same number of random samples as processors and return one cut for each sample.

Importance sampling [23, 58] combines Monte Carlo sampling with an L-shaped method. Importance sampling improves on the crude Monte Carlo method and

reduces the variance of the solution estimate. In each iteration, a subset (about 50-200) of realizations is used to generate an exact feasibility cut or an approximate optimality cut. Optimality cuts are only statistically valid. Probabilistic lower and upper bounds of the solutions are kept and the algorithm terminates when the two are statistically indistinguishable. Confidence intervals on the optimal value can also be calculated. As not all the scenarios are tested, it does not always guarantee feasibility of the second stage. Forcing the algorithm to consider some rare but disastrous scenarios has been suggested as a way to protect the system against such risks. The L-shaped method with importance sampling can be parallelized as the L-shaped method. Numerical results [58] show that importance sampling is capable of solving problems with a massive number of scenarios to a high accuracy of within 95% confidence interval of the solution.

2.4.5 Piecewise Quadratic Form of the L-shaped Method

Louveaux [72, 73] adapted the nested L-shaped method to the multistage quadratic stochastic program (1.12)–(1.13). The algorithm has the same tree traversing strategy as the nested L-shaped method and uses a cutting plane method to force convergence. It takes advantage of the piecewise quadratic nature of the objective (piecewise quadratic function will be discussed further in Section 3.2) and uses a Newton direction in each step. For each stage t subproblem, the algorithm minimizes the current quadratic function with respect to a search domain S_t and the domain of the current quadratic function C_t (termed a cell). The current cell boundary is obtained by solving recursively the appropriate subtree. If the solution is not optimal, then a cut is placed on the search domain S_t to remove at least the current cell.

Algorithm 2.7

Step 0 Find $x_1 \in X_1 = \{x | W_1 x \overline{\leq} h_1\}$. Set $t = 1$, E_1 and d_1 vacuous.

Step 1 If $t = T$

Goto Step 2.

Else

For $i = t + 1$ to T

$k_i = (D_i(k_{i-1}))_1$,

$G_i = H_i^{k_i}, q_i = c_i^{k_i}, a_i = 0.$

$f_i = \frac{1}{2} x_i^{\mathsf{T}} G_i x_i + q_i^{\mathsf{T}} x_i + a_i,$

$X_i = \{x | W_i^{k_i} x + V_i^{k_i} x_{i-1} \stackrel{=}{\leq} h_i^{k_i}\}$, $C_i = S_i = X_i$, $x_i \in X_i$.

E_i and d_i vacuous.

Set $t = T$.

Step 2 Minimize the current quadratic function with respect to the search domain S_t and the current cell C_t:

$$v = \arg\min\{f_t | x \in S_t\},$$
$$w = \arg\min\{f_t | x \in C_t\}. \tag{2.31}$$

Step 3 If $\nabla f_t(w)^{\mathsf{T}}(v - w) = 0$

Current subtree optimal, goto Step 4.

Else

Update $S_t = S_t \cap \{x | \nabla f_t(w)^{\mathsf{T}}(x - w) \leq 0\}$, $C_t = X_t$.

E_t and d_t vacuous.

$x_t = v$.

$G_t = H_t^{k_t}, q_t = c_t^{k_t}, a_t = 0.$

$f_t = \frac{1}{2} x_t^{\mathsf{T}} G_t x_t + q_t^{\mathsf{T}} x_t + a_t,$

Goto Step 1.

Step 4 If t = 1, optimal solution found; STOP.

Otherwise, calculate boundary and quadratic expression of current cell.

Consider the Kuhn-Tucker condition for (2.31)

$$\begin{bmatrix} G_t & (W_{1t}^{k_t})^\top & (W_{2t}^{k_t})^\top & E_t^\top & 0 & 0 \\ W_{1t}^{k_t} & 0 & 0 & 0 & 0 & 0 \\ W_{2t}^{k_t} & 0 & 0 & 0 & I & 0 \\ E_t & 0 & 0 & 0 & 0 & I \end{bmatrix} \begin{pmatrix} x \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} -q_t \\ h_{1t}^{k_t} - V_{1t}^{k_t} x_{t-1} \\ h_{2t}^{k_t} - V_{2t}^{k_t} x_{t-1} \\ d_t \end{pmatrix} \tag{2.32}$$

$$\lambda_2, \lambda_3, s_1, s_2 \geq 0. \tag{2.33}$$

Here W_1, and W_2 represent the equality and inequality constraints respectively. Let B be an optimal complementary basis of (2.32). Calculate

$$r = B^{-1} \begin{pmatrix} -q_t \\ h_t^{k_t} - V_t^{k_t} x_{t-1} \\ d_t \end{pmatrix}, \quad E = B^{-1} \begin{pmatrix} 0 \\ V_t^{k_t} \\ 0 \end{pmatrix}. \tag{2.34}$$

Let $\bar{r}$ and $\bar{E}$ be the rows of r and E that correspond to the non-negative variables $\lambda_2, \lambda_3, s_1, s_2$. Update

$$C_{t-1} \leftarrow C_{t-1} \cap \{x | \bar{E}x \leq \bar{r} + \bar{E}x_{t-1}\}, \tag{2.35}$$

$$E_{t-1} \leftarrow \begin{bmatrix} E_{t-1} \\ \bar{E} \end{bmatrix}, \quad d_{t-1} \leftarrow \begin{pmatrix} d_{t-1} \\ \bar{r} + \bar{E}x_{t-1} \end{pmatrix}.$$

Find $Q_t = \frac{1}{2} x_t^\top G x_t + q^\top x_t + a$ by substituting the value of the basic variable x from (2.32) into f_t.

Update $G_{t-1} \leftarrow G_{t-1} + p_t^{k_t} G,\ q_{t-1} \leftarrow q_{t-1} + p_t^{k_t} q,\ a_{t-1} \leftarrow a_{t-1} + p_t^{k_t} a,$

$f_{t-1} = \frac{1}{2} x_{t-1}^\top G_{t-1} x_{t-1} + q_{t-1}^\top x_{t-1} + a_{t-1}.$

If $k_t + 1 \notin D_t(k_{t-1})$

$t = t - 1$, goto Step 2.

Else solve subtree of next sibling:

$k_t = k_t + 1,$

$G_t = H_t^{k_t}, q_t = c_t^{k_t}, a_t = 0.$

$f_t = \frac{1}{2} x_t^\top G_t x_t + q_t^\top x_t + a_t,$

$X_t = \{x | W_t^{k_t} x + V_t^{k_t} x_{t-1} \ \overline{\leq}\ h_t^{k_t}\}$, $C_t = S_t = X_t$, $x_t \in X_t$.

E_t and d_t vacuous.

Goto Step 1.

In Step 4, the boundary of the current cell is constructed by adding cuts from each direct descendent problem. The variable r in (2.34) gives the optimal stage t primal and dual solution at the current x_{t-1}. For arbitrary $\tilde{x}_{t-1}$, the primal and dual variables using the same basis is

$$r - E(\tilde{x}_{t-1} - x_{t-1}).$$

This is optimal if and only if the non-negative dual variables λ_2, λ_3, and slack variables s_1 and s_2 remains non-negative, or

$$\bar{r} - \bar{E}(\tilde{x}_{t-1} - x_{t-1}) \geq 0.$$

hence the update formula (2.35).

If the recourse matrix W is fixed and all descendents have similar active sets, then some cuts may be redundant and can be removed. However, if the problems have stochastic recourse or very dissimilar active sets, then the number of constraints required to define a cell may at worst be the sum of the inequality constraints, including bounds, of all direct descendents. This may lead to numerical difficulty as the constraint gradients become almost linearly dependent.

Since the number of cell is finite and at least one cell is removed in each iteration, Louveaux [72] proved the algorithm converges finitely.

2.4.6 Two Stage Quadratic Stochastic Program

Chen et al. [16, 17] studied a two stage stochastic program with quadratic recourse and continuous random variables. The problem is formulated as

$$\begin{aligned} \min_{x} \quad & f(x) \equiv P(x) + Q(x) \\ \text{s.t.} \quad & Ax \leq b \end{aligned} \tag{2.36}$$

where

$$Q(x) = \int_{\Omega} \mathcal{Q}(x, \omega)\rho(\omega)d\omega$$

and

$$\begin{aligned} \mathcal{Q}(x, \omega) = \max_{y} \quad & -\tfrac{1}{2}y^{\mathsf{T}}Hy + y^{\mathsf{T}}(h(\omega) - Vx) \\ \text{s.t.} \quad & Wy \leq q. \end{aligned}$$

Here $P(x)$ is a convex twice differentiable function and H is positive definite.

Since H is positive definite, the optimal value function $\mathcal{Q}(x, \omega_i)$ for each realization ω_i is convex and continuously differentiable [16, 17]. Its gradient is given by

$$\nabla \mathcal{Q}_i(x, \omega_i) = -V^\intercal \arg\min_y \{-\tfrac{1}{2} y^\intercal H y + y^\intercal (h(\omega_i) - Tx), Wy \leq q\},$$

and the integral is approximated by numerical integration using N sample points.

$$\nabla f(x) = \nabla P(x) + \frac{1}{N} \sum_{i=1}^{N} \nabla \mathcal{Q}_i(x, \omega_i) \tilde{\rho}(\omega_i)$$

where $\tilde{\rho}(\omega_i)$ is some appropriate probability.

Let Y denotes the feasible set of the stage two problem. Let $\Pi_{H,Y}(u)$ be the projection of u onto Y induced by H, that is $\Pi_{H,Y}(u) = \arg\min_{y \in Y} \|u - y\|_H^2$. $\Pi_{H,Y}$ without any argument denotes the $n_1 \times n_2$ matrix which maps a vector from $\mathbb{R}^{n_1}$ to $Y \subseteq \mathbb{R}^{n_2}$. Let $\mathcal{L}(u)$ be the linearity space $\mathcal{L}(u) = \{y \in \mathbb{R}^m | (\Pi_{H,Y}(u) - u)^\intercal H y = 0, \overline{W}_u y = 0\}$ where $\overline{W}_u$ is the set of active constraints at u.

Let $u_i = H^{-1}(h(\omega_i) - Tx)$. An element of the generalized Hessian of the optimal value function $\mathcal{Q}(x, \omega_i)$ is given by

$$V(x, \omega_i) = \begin{cases} V^\intercal H^{-1} V, & \text{if } u^i \in \text{int } Y, \\ V^\intercal H^{-\frac{1}{2}} \Pi_{H, \mathcal{L}(u^i)} H^{-\frac{1}{2}} V, & \text{otherwise} \end{cases}$$

and an element of the generalized Hessian of $f(x)$ is

$$B(x) = \nabla^2 P(x) + \frac{1}{N} \sum_{i=1}^{N} V(x, \omega_i) \tilde{\rho}(\omega_i).$$

Using the above differential information, the authors proposed a generalized Newton method for solving (2.36). Let

$$g(x, \lambda) = \begin{pmatrix} \nabla f(x) + A^\intercal \lambda \\ \min(\lambda, b - Ax) \end{pmatrix}$$

and

$$g^i(d, \lambda) = \begin{pmatrix} \nabla f(x) + A^\intercal \lambda + B^i d \\ \min(\lambda, b - A(x^i + d)) \end{pmatrix},$$

where min is the component-wise minimum of the two vectors. Let $\{\eta^i\}$ be a non-negative forcing sequence which control the accuracy of the quadratic programs. The algorithm can be stated as

Algorithm 2.8

Step 0 Choose scalars $\rho, \sigma, \alpha, \epsilon \in (0,1)$. Choose feasible x^0 and feasible first stage Lagrange multipliers $\lambda^0 \geq 0$.

Step 1 For some appropriate positive definite matrix, find approximate generalized Newton direction d^i

$$\begin{aligned} \min_d \quad & \tfrac{1}{2} d^\intercal B^i d + \nabla f(x^i)^\intercal d \\ \text{s.t.} \quad & A(x^i + d) \leq b. \end{aligned} \tag{2.37}$$

Let d^i be a feasible solution to (2.37) and λ^i be an approximate Lagrange multiplier such that they satisfy

$$(B^i d^i + \nabla f(x^i))^\intercal d^i \leq 0$$

and

$$\|g^i(d^i, \lambda^{i+1})\| \leq \eta^i \|g(x^i, \lambda^i)\|.$$

Step 2 If $i = 0$

Let $\delta = \|g(x^0, \lambda^0)\|$. Perform Armijo linesearch:

Find the smallest integer $\tau^i \geq 0$ such that

$$f(x^i + \rho^\tau d^i) \leq f(x^i) - \sigma/2 \rho^\tau (d^i)^\intercal B^i d^i.$$

Let $x^{i+1} = x^i + \rho^{\tau_i} d^i$.

Else

If $\|g(x^i + d^i, \lambda^{i+1})\|/\delta \leq \epsilon$

$x^{i+1} = x^i + d^i$,

$\delta = \|g(x^{i+1}, \lambda^{i+1})\|$, $\epsilon = \alpha\epsilon$.

Else

Perform Armijo linesearch to find τ^i and set $x^{i+1} = x^i + \rho^{\tau_i} d^i$.

Step 3 If x^{i+1} satisfies a prespecified stopping rule, stop;

Else return to Step 1 and set $i = i + 1$.

If all the B^i used in the descent direction search subproblem are positive definite and the forcing sequence $\{\eta^i\}$ satisfy $\lim_{i\to\infty} \eta_i \|g(x^i, \lambda^i)\| = 0$, then the algorithm converges globally [16,17]. Moreover, if the solution satisfies the linear independence constraint qualification and the strict complementarity condition (see Section 3.1 for their definitions) and $f(x)$ is BD-regular at x^* (see Section 2.3.1), then the solution is unique and the rate of convergence is Q-superlinear.

2.5 Dual Decomposition Methods for Stochastic Programs

Dual decomposition works with subproblems corresponding to scenarios. Each scenario consists of a complete path from the root to one leaf node. That is each scenario problem is a deterministic multistage problem. A scenario problem can be formulated either as a large scale problem analogous to (1.14) or recursively as in (1.12)–(1.13).

Dual decomposition schemes are Lagrangian based methods aimed directly at nonlinear stochastic programming problems. This class of methods takes advantage of the weak link between scenarios. As each scenario problem is deterministic and has a separate set of variables, they are only coupled by the *non-anticipativity* requirement which stipulates that nodes sharing the same stochastic parameter history up to and including that stage must make the same decisions. Dual decomposition methods relax the non-anticipativity constraints linking the scenarios and iteratively solve all individual scenarios while non-anticipativity is enforced gradually through a penalty term in the objective.

Dual decomposition algorithms are well suited to parallelization as each scenario problem can be stored and solved on separate processors. Generally, the only communication requirements is for processors to share scenario solutions to enforce non-anticipativity. As with all decomposition schemes, there is a need to balance the advantage of having very small subproblems with the increase in the number of subproblems and iterations needed for convergence and the increasing communications to computation time ratio.

2.5.1 Lagrangian Finite Generation Method

Lagrangian finite generation [102, 103] is a Lagrangian based method applicable to two stage stochastic programs with linear or convex quadratic subproblems with relatively complete recourse.

Consider the problem

$$\min_{x \in X} c^{\mathsf{T}} x + \tfrac{1}{2} x^{\mathsf{T}} C x + E\psi_\omega(x) \tag{2.38}$$

where

$$\psi_\omega(x) = \max_{z_\omega \in Z_\omega} z_\omega^{\mathsf{T}}(h_\omega - V_\omega x) - \tfrac{1}{2} z_\omega^{\mathsf{T}} H_\omega z_\omega, \tag{2.39}$$

C and H_ω are positive semi-definite, and X and Z_ω are nonempty convex polyhedra.

The Lagrangian is

$$\mathcal{L}(x, z) = c^{\mathsf{T}} x + \tfrac{1}{2} x^{\mathsf{T}} C x + E\{z_\omega^{\mathsf{T}}(h_\omega - V_\omega x) - \tfrac{1}{2} z_\omega^{\mathsf{T}} H_\omega z_\omega\} \tag{2.40}$$

for $x \in X, z \in Z$. Here $Z = \otimes_{\omega \in \Omega} Z_\omega$ and $z \in Z$ is formed by appending all z_ω together. $\mathcal{L}(x, z)$ is convex in x and concave in z.

The Lagrangian finite generation method works by finding the saddle point of $\mathcal{L}(x, z)$ in $X \times Z^\nu$ where $Z^\nu \subseteq Z, \nu = 1, 2, \ldots$ is the convex hull of a small number of points in Z. The sets Z^ν are used to approximate Z. Given a termination parameter $\epsilon \geq 0$, the Lagrangian finite generation method can be stated as

Algorithm 2.9

Step 0 Choose an initial convex polytope $Z^1 \subset Z$. Set $\nu = 1$.

Step 1 Find a saddle point $(\bar{x}^\nu, \bar{z}^\nu)$ of $\mathcal{L}(x, z)$ relative to $X \times Z^\nu$ and set $\bar{\alpha}_\nu = \mathcal{L}(\bar{x}^\nu, \bar{z}^\nu)$.

Step 2 For $\omega \in \Omega$, solve the following for z_ω^ν

$$\alpha_\omega^\nu = \max_{z_\omega \in Z_\omega} z_\omega^{\mathsf{T}}(h_\omega - V_\omega \bar{x}^\nu) - \tfrac{1}{2} z_\omega^{\mathsf{T}} H_\omega z_\omega.$$

Form $z^\nu \in Z$ by appending z_ω^ν and set

$$\alpha_\nu = c^{\mathsf{T}} \bar{x}^\nu + \tfrac{1}{2} (\bar{x}^\nu)^{\mathsf{T}} C \bar{x}^\nu + E\alpha_\omega^\nu = \mathcal{L}(\bar{x}^\nu z^\nu).$$

Step 3 If $\alpha_\nu - \bar{\alpha}_\nu \leq \epsilon$, STOP.

Step 4 Choose $Z^{\nu+1}$ containing $\{\bar{z}^\nu, z^\nu\}$. $\nu = \nu + 1$. Goto Step 1.

The dual variable space is approximated by Z^ν which is the convex hull of a subsequence $\{\tilde{z}^{\nu_i}\}$ of $\{\tilde{z}^\nu\}$, that is

$$Z^\nu = \text{co}\{\tilde{z}^{\nu_i} | i = 1, \ldots, m_\nu\} = \left\{ \sum_{i=1}^{m_\nu} \lambda_i \tilde{z}^{\nu_i} \,\middle|\, \lambda_i \geq 0, \sum_{i=1}^{m_\nu} \lambda_i = 1 \right\},$$

where $m_\nu \geq 2$ is defined by the user. Since all points in Z^ν can be described by a convex combination of all the extreme points $\tilde{z}^{\nu_i}$, the minimax problem in (x, z) in Step 1 is equivalent to a minimax problem in (x, λ) which is of a much lower dimension if m_ν is kept reasonably small. This problem can also be converted into a standard QP. See [103] for details.

The algorithm converges to the optimal solution linearly if C and H_ω are positive definite [102]. The rate of convergence can be estimated in advance and is valid for the whole sequence of $\{x^\nu\}$. For problems that are only convex, not strictly convex, a positive definite term can be added to the Hessian. The overall algorithm then converges linearly.

2.5.2 Diagonal Quadratic Algorithm

The diagonal quadratic algorithm (DQA) [4, 89] is an augmented Lagrangian based decomposition scheme that exploits the sparsity of the linking constraints connecting subsets of variables. It is well suited to nonlinear multistage stochastic programming and can handle problems with nonlinear equations linking the successive stages.

The DQA can be applied as either a primal or dual decomposition scheme, although it is more natural to take the dual approach. Consider a T-stage, K_T-scenario stochastic program. Associate with each scenario $k, k = 1, \ldots, K_T$, a set of variables x_t^k for each stage $t, t = 1, \ldots, T$. The scenario k solution is $x^k = (x_1^k, x_2^k, \ldots, x_T^k)$ and the complete solution to the original problem is given by $x = (x^1, x^2, \ldots, x^{K_T})$. To enforce non-anticipativity, the idea of a sibling is used. Suppose the tree is organized so that for each scenario k, $k + 1$ has the most recent common ancestor

with scenario k among all scenarios j, $j > k$. For each node on the scenario tree, the first node on its right which shares its immediate ancestor is called its sibling. If no such node exist, the leftmost node will become the sibling. The mapping is a rotation map of nodes sharing the same immediate ancestor and each node has a unique sibling. Non-anticipativity can be restated as: each node and its sibling must have identical values, or

$$x_t^k = x_t^{s(k,t)}, \quad k = 1, \ldots, K_t, t = 1, \ldots, T \tag{2.41}$$

where $s(k,t)$ denote the index of the sibling of scenario k at stage t. If we also denote the scenario k feasible set by

$$X^k = \{x^k | x^k \text{ satisfies (1.13b) for } \xi = \omega^k\} \tag{2.42}$$

and the problem feasible set as $X = X_1 \times X_2 \times \ldots \times X_{K_T}$. Then (1.12) can be rewritten as

$$\min_{x \in X} \quad \sum_{k=1}^{K_T} p_k \sum_{t=1}^{T} f_t^k(\omega_t^k, x_t^k) \tag{2.43}$$

$$\text{s.t.} \quad x_t^k = x_t^{s(k,t)}, \quad k = 1, \ldots, K_t, t = 1, \ldots, T. \tag{2.44}$$

Problem (2.43) is separable in the variable x^k except for constraints (2.41). The DQA proceeds by relaxing these non-separable linking constraints. The problem is reformulated using the augmented Lagrangian

$$\mathcal{L}(x, \pi) = \sum_{k=1}^{K_T} \sum_{t=1}^{T} p_k f_t^k(x_t^k) + \sum_{k=1}^{K_T} \sum_{t=1}^{T-1} (\pi_t^k)^\intercal (x_t^k - x_t^{s(k,t)}) + \frac{1}{2}\rho \sum_{k=1}^{K_T} \sum_{t=1}^{T-1} \|x_t^k - x_t^{s(k,t)}\|^2 \tag{2.45}$$

where π denotes the Lagrange multipliers and $\rho > 0$ is a penalty parameter. The augmented Lagrangian method is chosen as it is easy to apply and does not suffer from ill-conditioning. For an arbitrary initial multiplier π^0, the algorithm can be stated as:

Algorithm 2.10

Step 0 Choose starting multiplier π^0, set $\nu = 0$.

Step 1 Calculate

$$x^{\nu} = \arg\min_{x \in X} \mathcal{L}(x, \pi^{\nu}). \tag{2.46}$$

Step 2 If $x_t^k = x_t^{s(k,t)}$ for all $k = 1, \ldots K_T$, $t = 1, \ldots, T$,

stop, optimal solution found;

Else

Let $\pi_t^{k,\nu+1} = \pi_t^{k,\nu} + \rho(x_t^{k,\nu} - x_t^{s(k,t),\nu})$.

Set $\nu = \nu + 1$. Return to Step 1.

Clearly, the quadratic term in (2.45) is non-separable in the x^k. To completely decompose the problem, $\mathcal{L}(x, \pi)$ is approximated separately for each x^k by temporarily holding the other variables $x^j, j \neq k$ constant at $\hat{x}$.

$$\mathcal{L}(x, \pi) \simeq \sum_{k=1}^{K_T} \mathcal{L}^k(x^k, \hat{x}, \pi), \tag{2.47}$$

where

$$\begin{aligned} \mathcal{L}^k(x^k, \hat{x}, \pi) &= p_k \sum_{t=1}^{T} f_t(\omega_t^k, x_t^k) + \sum_{t=1}^{T-1} (x_t^k)^{\intercal}(\pi_t^k - \pi_t^{s^{-1}(k,t)}) \\ &\quad + \frac{1}{2}\rho \sum_{t=1}^{T-1} \left(\|x_t^k - \hat{x}_t^{s(k,t)}\|^2 + \|x_t^k - \hat{x}_t^{s^{-1}(k,t)}\|^2 \right). \end{aligned} \tag{2.48}$$

Each x^k can now be minimized separately given $\hat{x}$ and π. Given a relaxation parameter $0 < \tau < 1$ and stopping parameter $\epsilon > 0$, a nonlinear Jacobi iteration with under relaxation is then used to enforce non-anticipativity and obtain a solution for (2.46):

Algorithm 2.11

Step 0 Choose $0 < \tau < 1, \epsilon > 0$. Set $\pi = \pi^{\nu}$, $\hat{x}^{\nu,m} = x^{\nu-1}$ and $m = 1$.

Step 1 For $k = 1, \ldots, K_T$, find

$$x_k^{\nu,m} = \arg\min_{x^k \in X^k} \mathcal{L}^k(x^k, \hat{x}^{\nu,m}, \pi). \tag{2.49}$$

Step 2 If $\|x^{\nu,m} - \hat{x}^{\nu,m}\| < \epsilon$ then stop; otherwise set

$$\hat{x}^{\nu,m+1} = \hat{x}^{\nu,m} + \tau(x^{\nu,m} - \hat{x}^{\nu,m}), \tag{2.50}$$

set $m = m + 1$ and return to Step 1.

It is not necessary to solve (2.49) to optimality. The iterative nature of both the Jacobi steps and interior point methods can be exploited by performing only a few steps of the interior point method for (2.49) before updating using (2.50).

The DQA scheme converges linearly. It can also be applied to induce a primal decomposition. However, it is shown in [4] that the rate of convergence depends on the sparsity of the constraints linking the partially separable subsets of variables. Since each non-anticipativity constraint links only two variables while the equations of dynamics linking nodes in successive time stages can be more complex, it is expected that the method converges faster for dual decomposition than when applied to its primal counterpart. The different convergent rates when applied to dual and primal decomposition have been demonstrated in [104]. The dual decomposition scheme is also more attractive from a modelling point of view. The non-anticipativity constraints have the same form in all applications and can be easily relaxed automatically while the equations of dynamics are problem dependent and relaxing them needs to be dealt with case by case.

DQA is highly suited to a parallel environment. DQA does not need a central coordinating unit as only siblings need to exchange solutions in step (2.49). By choosing an appropriate computer architecture and allocating the subproblems suitably, communication is restricted to the nearest neighbours only. This removes the communication bottlenecks commonly encountered in other parallel implementations (an example is when all processors try to return cut information to the master problem in the L-shaped method and its variants). With the DQA scheme, the stochastic program can be decomposed into smaller problems consisting of one or more scenarios in a very flexible manner (nodal problems for primal decomposition) depending on the capacity of the machines and efficiency of the algorithm used to solve the subproblems.

2.5.3 Progressive Hedging Algorithm

The progressive hedging algorithm (PHA) [103] (also referred to as scenario aggregation) is very similar to DQA applied as a dual decomposition scheme. It also uses an augmented Lagrangian approach and induces a full decomposition of the stochastic program into scenario problems by relaxing the non-anticipativity constraints. The major difference between DQA and PHA is the way non-anticipativity is expressed. Instead of using a Jacobi iterations to force all sibling problems to have the same solution, PHA takes the expected value of all the scenario optimal solutions to form a non-anticipativity solution (termed an implementable policy) which is not necessarily feasible in all scenarios. That is

$$\text{implementable policy: } \bar{x} = \sum_{k=1}^{K} p_k x^k. \tag{2.51}$$

This implementable policy is then used to form approximate scenario problems in the next iteration in the same way as the sibling solution was used in the DQA. The only difference is that now every scenario has the same 'sibling'. Analogous to (2.48), the subproblems can be written as

$$\min_{x^k \in X^k} f_t^k(\omega_t^k, x_t^k) + (\pi^k)^\intercal (x^k - \bar{x}) + \frac{1}{2}\rho \|x^k - \bar{x}\|^2. \tag{2.52}$$

PHA can then be formulated as

Algorithm 2.12

Step 0 Choose $\bar{x}^0$ and π^0 and set $\nu = 1$.

Step 1 For $k = 1, \ldots, K$, $\bar{x} = \bar{x}^{\nu-1}$ and $\pi^k = \pi^{k,\nu-1}$, solve (2.52) for $x^{k,\nu}$.
If $x^{k,\nu} = \bar{x}^{\nu-1}$ for all $k = 1, \ldots, K$, then STOP; optimal solution found.

Step 2 Calculate $\bar{x}^\nu = \sum_{k=1}^{K} p_k x^{k,\nu}$.

Step 3 $\pi^{k,\nu+1} = \pi^{k,\nu} + \rho(x^{k,\nu} - \bar{x}^\nu)$. Set $\nu = \nu + 1$. Return to Step 1.

This procedure can easily be parallelized. The only communication requirement is to pass the subproblem solutions to the master processor which then returns the

implementable policy as the next trial solution. The multipliers are easy to calculate. The method is stable and arbitrary starting points can be used. Each iteration is relatively simple, however, it may take many iterations and convergence can be slow.

The algorithm is proven [103] to show strict improvement at each step for convex programs and it converges to a solution linearly.

2.6 Stochastic Quasigradient

Stochastic quasigradient [28, 29] is one of the earliest practical method to solve stochastic programs. It is the stochastic analogue of the steepest descent method in deterministic optimization. The problem can be stated as

$$\begin{aligned} \min \quad & E[q(x,\xi)] \\ \text{s.t.} \quad & x \in X, \end{aligned}$$

where X is a closed convex set and the random variable ξ may be discrete or continuous.

In each iteration, one or more sample observations are made and an estimate of the objective gradient at the current point x is calculated either analytically or using forward/backward/central differences. A step is then taken along the steepest descent direction and the new point is projected into the feasible region if necessary. This method is easy to apply, however, convergence is relatively slow especially for problems in higher dimension. It is not easy to decide on an appropriate step or a stopping criterion as the sequence of optimal value estimates is not monotonic. It can be proven that the sequence of iterates converges to the optimal solution with probability 1 under suitable conditions.

2.7 Stochastic Programs with Continuous Random Variables

For stochastic programs with continuous random variables, calculating the recourse function often involves multidimensional integration. This can be achieved by Monte-Carlo type methods such as stochastic decomposition and importance sampling discussed in Section 2.4.4 or stochastic quasigradient method in Section 2.6. Numerical integration technique can also be used if the number of continuous random variables is small (no more than 10), see [17] for details.

Another approach is to discretize the continuous random variables and calculate lower and upper bounds of the expected value. If the difference between the lower and upper bound is higher than some prespecified tolerance, then the discretization of the random variables need to be refined to tighten the bounds.

A simple lower bound is the Jensen's inequality. Suppose the support of the random variable ξ is partitioned into a number of regions $C^k, k = 1, \ldots, K$. Let $p^k = P\{\xi|C^k\}$ and $\xi^k = E[\,\xi|C^k\,]$ be the probability and expected value of ξ with respect to a region C^k respectively. The Jensen's lower bound can be stated as

$$\mathrm{E}_\xi[g(x,\xi)] \geq \sum_{k=1}^{K} p^k g(x,\xi^k).$$

If ξ has polyhedral support, then the Edmundson-Madansky inequality provides an upper bound to the expected value if $g(x,\xi)$ is separable in ξ or if ξ is stochastically independent. Suppose the support of ξ is $\Xi = [a_1, b_1] \times \ldots \times [a_n, b_n]$ and let extΞ denotes the extreme points of Ξ. The Edmundson-Madansky inequality is

$$\mathrm{E}_\xi[(g(x,\xi)] \leq \sum_{e \in \mathrm{ext}\Xi} \prod_{i=1}^{n} \frac{|\bar{\xi}_i - e_i|}{b_i - a_i} g(x,e).$$

For a thorough treatment of these and other bounds and how they can be incorporated into efficient algorithms, see [10, 62].

Chapter 3

Properties of Multistage Stochastic Quadratic Programs

Multistage stochastic quadratic programs (MQSPs) defined in (1.12)–(1.13) are highly structured mathematical programming problems. The objective of each subproblem is, although not C^2, a convex, Lipschitz piecewise quadratic function. This can be exploited in algorithmic development leading to significant savings over general multistage stochastic programs. Section 3.1 summarises many results from sensitivity analysis which are useful to our investigations. Piecewise quadratic functions and piecewise quadratic programs are then defined in Section 3.2. We study the properties of the optimal value functions of the piecewise quadratic programs when their right hand sides are perturbed. Section 3.3 applies the above results to show that MQSPs can be expressed as piecewise quadratic programs. We examine the structure of the piecewise quadratic objective functions and derive expressions for component quadratic functions active at a point. The differential information obtained is used in the next chapter to develop generalized Newton method for solving MQSPs.

3.1 Sensitivity Analysis

Consider the convex programming problem with right hand side perturbation

$$\phi(\xi) = \min_{x \in \mathbb{R}^n} f(x) \tag{3.1a}$$

$$\text{s.t. } g_i(x) \leq \xi_i, \quad i = 1, \ldots, r, \tag{3.1b}$$

$$h_j(x) = \xi_j, \quad j = r+1, \ldots, m. \tag{3.1c}$$

Here f and $g_i, i = 1, \ldots, r$ are convex functions and finite on the feasible region. The equality constraint functions h_j are affine for $j = r+1, \ldots, m$. The feasible region is clearly closed and it is assumed to be nonempty and bounded which is natural for many applications. The *unperturbed* problem is obtained when $\xi_i = 0$, $i = 1, \ldots, m$.

Let u_i and w_j be the Lagrange multipliers for the inequality and equality constraints respectively. The *Kuhn-Tucker (KT) condition* is satisfied for the unperturbed problem at a feasible point x^* if there exists $u_i^*, i = 1, \ldots, r$ and $w_j^*, j = r+1, \ldots, m$ such that

$$0 \in \partial f(x^*) + \sum_{i=1}^{r} u_i^* \, \partial g_i(x^*) + \sum_{j=r+1}^{m} w_j^* \, \partial h_j(x^*) \tag{3.2a}$$

and

$$u_i^* g_i(x^*) = 0, u_i^* \geq 0 \text{ for } i = 1, \ldots, r. \tag{3.2b}$$

Next we list some constraint qualifications which are necessary to guarantee the existence of the Lagrange multipliers. Details can be found in [32, 53]. For the following constraint qualifications, we assume $\xi = 0$. Let $X \subseteq \mathbb{R}^n$ denote the feasible region of the unperturbed problem. Let $\mathcal{A}(x) = \{i | g_i(x) = 0, i = 1, \ldots, r\}$ be the set of active inequality constraints at x.

[MFCQ] The *Mangasarian-Fromovitz* constraint qualification is said to hold at $x^* \in X$ if

1. the gradients $\{\nabla h_j(x^*), j = r+1, \ldots, m\}$ are linearly independent.

2. there exists a vector $z \in \mathbb{R}^n$ such that $\nabla g_i(x^*)^\intercal z < 0$ for all $i \in \mathcal{A}(x^*)$, $\nabla h_j(x^*)^\intercal z = 0$ for $j = r+1, \ldots, m$;

[SLCQ] The *Slater condition* is satisfied at $x^* \in X$ if $h_j, j = r+1, \ldots, m$ are affine, $g_i, i = 1, \ldots, r$ are pseudo-convex and there exists a point $\bar{x} \in X$ with $g_i(\bar{x}) < 0$ for all $i \in \mathcal{A}(x^*)$.

[SSLCQ] The *strong Slater condition* is satisfied if $h_j, j = r+1, \ldots, m$ are affine and linearly independent and there exists a point $\bar{x} \in X$ such that $h(\bar{x}) = 0$ and $g_i(\bar{x}) < 0$ for $i = 1, \ldots, r$.

[LICQ] The *linear independence* constraint qualification is satisfied at x^* if the set of active constraint gradients $\{\nabla h_j(x^*), j = r+1, \ldots, m, \nabla g_i(x^*), i \in \mathcal{A}(x^*)\}$ are linearly independent.

If all the constraint functions are affine, then the *basic constraint qualification* is satisfied. This implies that there exists Lagrange multipliers such that the KT condition holds.

MFCQ is both necessary and sufficient for the set of Lagrange multiplier vectors corresponding to a given local solution of a general NLP problem to be nonempty, compact and convex. MFCQ is preserved under right hand side perturbations.

If the equality constraints are affine and the inequalities are pseudo-convex functions, which is satisfied by any convex program, then MFCQ is equivalent to SLCQ.

SSLCQ extends MFCQ to convex nonsmooth programming. It is a necessary and sufficient condition for ensuring the set of Lagrange multipliers to be nonempty, compact and convex.

LICQ implies MFCQ. If LICQ holds and all the problem data is continuously differentiable at a solution x^*, then the Lagrange multipliers corresponding to x^* exists and is unique. However, if the objective is not differentiable at x^*, then LICQ is not enough to guarantee uniqueness of the Lagrange multiplier as Example 3.1 shows.

Example 3.1 *Consider the problem*

$$\begin{array}{ll} \min\limits_{x} & f(x) = \begin{cases} x, & x \leq 0 \\ 2x, & x \geq 0 \end{cases} \\ \text{s.t.} & -x \leq 0 \end{array}$$

Clearly the solution is $x^* = 0$ and the constraint is active. Let u be the Lagrange multiplier. The optimality condition is:

$$\exists v \in \partial f(0) = [1, 2] \text{ and } u \geq 0 \text{ such that } v - u = 0.$$

Therefore $u \in [1, 2]$ which is not unique even though LICQ is satisfied at the solution.

Definition 3.2 (Strict complementarity condition (SCC)) *The strict complementarity condition is satisfied at x^* if $u_i^* > 0$ for all $i \in \mathcal{A}(x^*)$.*

Definition 3.3 (Second order sufficiency condition (SOSC), [32, Lemma 3.2.1]) *An optimal solution x^* satisfies the second order sufficiency condition for the unperturbed problem (3.1) if*

1. *the functions f, g_i and h_j are twice continuously differentiable in a neighbourhood of x^*;*
2. *there exists Lagrange multipliers u^* and w^* such that the first order Kuhn-Tucker conditions (3.2) are satisfied;*
3. *$z^\intercal \nabla^2 f(x^*) z > 0$ for all $z \neq 0$ such that*
   - *$\nabla f(x^*)^\intercal z = 0$;*
   - *$\nabla g_i(x^*)^\intercal z \leq 0, i \in \mathcal{A}(x^*)$;*
   - *$\nabla h_j(x^*)^\intercal z = 0$ for $j = r + 1, \ldots, m$.*

Lemma 3.4 ([32, Lemma 3.2.1]) *If a feasible point x^* of the unperturbed problem (3.1) satisfies the second order sufficiency condition, then x^* is a strict local minimizer of the problem, that is there exists a neighbourhood $B(x^*)$ around x^* such that $f(x) > f(x^*)$ for all $x \in B(x^*) \cap X$.*

For a convex program, Lemma 3.4 implies that an optimal solution is unique if the second order sufficiency condition is satisfied. For a convex quadratic program, this result can be strengthened to read that an optimal solution is unique if and only if SOSC holds. To see this, suppose x^* is the unique optimal solution that satisfies the KT condition but does not satisfy SOSC. Then there exists a feasible direction $z \neq 0$ such that $\nabla f(x^*)^\intercal z = 0$ and $z^\intercal \nabla^2 f(x^*)\, z = 0$. Since a quadratic function is completely characterized by its second order Taylor series, $f(x^*) = f(x^* + \alpha z)$ for all $\alpha \in \mathbb{R}$. This contradicts the assumption that x^* is the unique optimal solution.

When SOSC fails, it is possible to get multiple optimal solutions which form a convex set for convex programs. However, the next proposition states that all solutions must share the same Lagrange multiplier. So we can refer to the Lagrange multiplier of a convex program without reference to a particular optimal solution.

Proposition 3.5 ([53, Proposition VII 3.1.1]) *Let $\bar{x}$ and $\hat{x}$ be two solutions of the convex program (3.1) and $U(\bar{x})$ and $U(\hat{x})$ the sets of associated Lagrange multipliers, then $U(\bar{x}) = U(\hat{x})$.*

The following results study the differential property of the optimal value function $\phi(\xi)$ of (3.1).

Theorem 3.6 ([32, Theorem 5.4.1]) *If the following conditions are satisfied at x^* with Lagrange multipliers u^* and w^* for $\xi = 0$:*

1. *the second order sufficiency condition;*
2. *the linear independence constraint qualification;*
3. *strict complementarity condition;*

then

1. *x^* is a local isolated minimizer of (3.1) and u^* and w^* are unique;*
2. *for ξ in a neighbourhood of 0, there exists a once continuously differentiable vector function $y(\xi) = [x(\xi)^\intercal\, u(\xi)^\intercal\, w(\xi)^\intercal]^\intercal$ that satisfies the second order sufficiency condition for (3.1) with $y(0) = [x^{*\intercal}\, u^{*\intercal}\, w^{*\intercal}]^\intercal$;*

3. strict complementarity and LICQ hold at $y(\xi)$ for ξ near 0;

4. for ξ in a neighbourhood of 0, the gradient of the optimal value function is

$$\nabla_\xi \phi(\xi) = \begin{pmatrix} -u(\xi) \\ -w(\xi) \end{pmatrix}; \tag{3.3}$$

5. for ξ in a neighbourhood of 0, the Hessian of the optimal value function is

$$\nabla^2_\xi \phi(\xi) = \begin{pmatrix} -\nabla_\xi u(\xi) \\ -\nabla_\xi w(\xi) \end{pmatrix}. \tag{3.4}$$

Rockafellar [101, §28,§29] discusses the relationship between the optimal value function $\phi(\xi)$ and the Lagrange multiplier for the convex program (3.1).

Theorem 3.7 ([101, Theorem 29.1]) *If the optimal value $\phi(0)$ is finite, then the Kuhn-Tucker vectors for problem (3.1) are precisely the vectors u such that $-u$ is a subgradient of $\phi(\xi)$ at $\xi = 0$, that is*

$$-u \in \partial\phi(0).$$

Corollary 3.8 ([101, Corollaries 29.1.1, 29.1.3]) *The one-sided directional derivative*

$$\phi'(0, \xi) = \lim_{\lambda \downarrow 0} \frac{\phi(\lambda\xi) - \phi(0)}{\lambda}$$

exists for every $\xi \in \mathbb{R}^m$ and is a positively homogeneous convex function of ξ. The Kuhn-Tucker vectors u form a closed convex set $U \in \mathbb{R}^m$ and its support function is the closure of the function $\xi \to \phi'(0, -\xi)$, that is

$$\max_{u \in U} u^\intercal \xi = cl\ \phi'(0, -\xi).$$

The unperturbed problem (3.1) has a unique Kuhn-Tucker vector if and only if $\phi(\xi)$ is differentiable at $\xi = 0$ and $u^* = -\nabla\phi(\xi)|_{\xi=0}$.

3.2 Piecewise Quadratic Programs (PQPs)

We now turn our attention to piecewise quadratic functions. Section 3.2.2 looks at a minimizing piecewise quadratic function subject to linear constraints, in particular, sensitivity information when the right hand side of the constraints is perturbed. These results are used to show that each stage $t+1$ subproblem of (1.13) is a piecewise quadratic program with right hand side perturbation $V_{t+1}x_t$ and obtain an expression for elements of the subdifferential and generalized Hessian of the stage t recourse function.

3.2.1 Piecewise Quadratic Functions

Definition 3.9 (Piecewise quadratic function) *A function is called* piecewise quadratic *if it is continuous on its effective domain and its effective domain is a nonempty polyhedron that can be decomposed into a finite number of convex polyhedra, on each of which the function is quadratic. Let* $\{P_l, l \in L\}$ *be a polyhedral partition of the domain* $X \subseteq \mathbb{R}^n$ *of* f *where* L *is a finite index set with* $|L|$ *elements. That is,* $X = P_1 \cup P_2 \cup \ldots \cup P_{|L|}$ *and* $P_i \cap P_l \subseteq \mathbb{R}^{n-1}$ *if* $i \neq l$*. Then on each polyhedron* P_l, $f(x) = \frac{1}{2}x^\top G_l x + c_l^\top x + f_{0l}$ *where* $G_l \in \mathbb{R}^{n\times n}$ *are symmetric,* $c_l \in \mathbb{R}^n$ *and* $f_{0l} \in \mathbb{R}$*.*

Linear, piecewise linear and quadratic functions defined on convex polyhedra are special cases of piecewise quadratic functions. Semi-variance and piecewise linear-quadratic risk measures discussed in Section 1.3.3 are also examples of piecewise quadratic functions.

From the definition, the sum of a finite number of piecewise quadratic functions is also piecewise quadratic. Note however that, the maximum of two quadratic functions may not satisfy this definition of a piecewise quadratic function since the boundary is in general curved. For example, let $f(x) = \max\{x_1^2 + x_2^2,\, x_2^2 + x_2\}$, f is clearly convex but the boundary of the quadratic pieces is the curve $x_2 = x_1^2$. Thus strictly speaking, we are only dealing with what should be called 'polyhedral piecewise quadratic' functions.

Piecewise quadratic functions are an example of a class of functions called piecewise smooth or piecewise C^k functions. These functions are highly structured, it is known that they are locally Lipschitz and semismooth. See [15] for more detailed discussion.

If a piecewise quadratic function f is convex, then its subgradient exists everywhere and is given by

$$\partial f(x) = \text{co}\{\nabla f_j(x),\, j \in J(x)\}. \tag{3.5}$$

Similarly, the generalized Hessian is given by

$$\mathcal{G}(x) = \text{co}\{\nabla^2 f_j(x),\, j \in J(x)\}. \tag{3.6}$$

A piece $f_l, l \in L$ of the piecewise quadratic function is *active* at x if f_j coincides with the piecewise quadratic function on a set around x with positive measure. The set of active pieces $J(x)$ at x can be defined as

$$J(x) = \{l \in L | P_l \cap B_\delta(x) \text{ has positive measure for all } \delta > 0\},$$

where $B_\delta(x)$ is the ball around x with radius δ.

In a piecewise quadratic function, we call a piece *degenerate* if its active domain, which is where it coincides with the piecewise quadratic function, has an empty interior in $\mathbb{R}^n$.

In Figure 3.1, $f_2(x)$ is active for $x \leq -10/7$. It crosses $f(x)$ again at $x = 0$ but is clearly degenerate at that point. This example shows a potential difficulty caused by degeneracy. Observe that the tangent of $f_2(x)$ at the origin is not a supporting plane for $f(x)$. If it is wrongly classified as active, then it can cause an algorithm to consider the minimum to be at the origin instead of $x = 1$. This is a very serious problem for algorithmic development if degeneracy is not clearly excluded.

3.2.2 Piecewise Quadratic Programs

A piecewise quadratic program (PQP) minimizes a piecewise quadratic function subject to linear constraints. Consider the perturbation problem (3.1) again, with

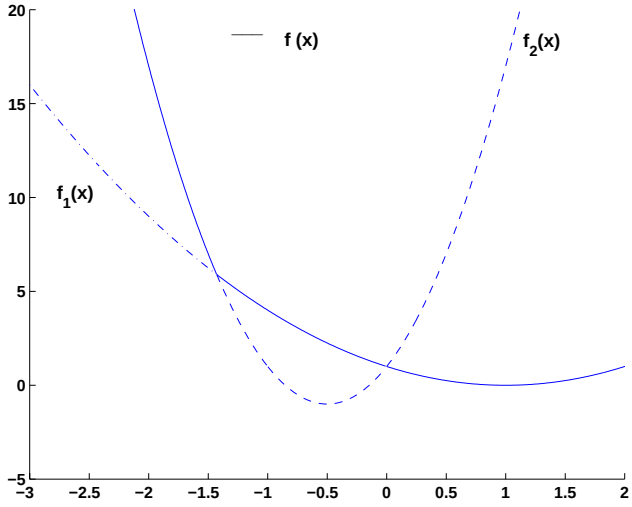

Figure 3.1: Example of degenerate piecewise quadratic function

a convex and piecewise quadratic objective. A linear perturbation in the objective is also considered to demonstrate the symmetry between perturbations in the right hand side and the objective. Let

$$\begin{aligned} \phi(\xi,\eta) = \min_{x\in\mathbb{R}^n}\ & f(x) + \eta^{\mathsf{T}}x \\ \text{s.t. } & A_1x = b_1 + \xi_1 \\ & A_2x \le b_2 + \xi_2. \end{aligned} \tag{3.7}$$

Here $f(x) : \mathbb{R}^n \to \mathbb{R}$ is convex and piecewise quadratic, $\eta \in \mathbb{R}^n$, $\xi_1 \in \mathbb{R}^{m_1}$ and $\xi_2 \in \mathbb{R}^{m_2}$ are perturbation vectors with $\xi = (\xi_1^{\mathsf{T}}, \xi_2^{\mathsf{T}})^{\mathsf{T}} \in \mathbb{R}^m$.

The Lagrangian of (3.7) is

$$\mathcal{L}(x,u,w,\xi,\eta) = f(x) + \eta^{\mathsf{T}}x + u^{\mathsf{T}}(A_1x - b_1 - \xi_1) + w^{\mathsf{T}}(A_2x - b_2 - \xi_2), \tag{3.8}$$

where $u \in \mathbb{R}^{m_1}$ and $0 \le w \in \mathbb{R}^{m_2}$ are the Lagrange multipliers for the equality and inequality constraints respectively. As the constraints are all linear, the basic constraint qualification [53] guarantees the existence of optimal Lagrange multipliers.

The primal problem associated with $\mathcal{L}(x,u,w,\xi,\eta)$ is

$$\inf_{x\in\mathbb{R}^n} F(x,\xi,\eta), \tag{3.9}$$

where

$$\begin{aligned} F(x,\xi,\eta) &= \sup_{u\in\mathbb{R}^{m_1},w\in\mathbb{R}^{m_2}_+} \mathcal{L}(x,u,w,\xi,\eta) \\ &= \begin{cases} f(x)+\eta^{\mathsf{T}}x, & A_1x = b_1+\xi_1, A_2x \le b_2+\xi_2 \\ +\infty, & \text{otherwise,} \end{cases} \end{aligned}$$

and the corresponding dual problem is

$$\sup_{u\in\mathbb{R}^{m_1},w\in\mathbb{R}^{m_2}_+} G(u,w,\xi,\eta) \tag{3.10}$$

where

$$\begin{aligned} G(u,w,\xi,\eta) &= \inf_{x\in\mathbb{R}^n} \mathcal{L}(x,u,w,\xi,\eta) \\ &= -u^{\mathsf{T}}(b_1+\xi_1) - w^{\mathsf{T}}(b_2+\xi_2) - \sup_{x\in\mathbb{R}^n}\{(-A_1^{\mathsf{T}}u - A_2^{\mathsf{T}}w-\eta)^{\mathsf{T}}x - f(x)\} \\ &= -u^{\mathsf{T}}(b_1+\xi_1) - w^{\mathsf{T}}(b_2+\xi_2) - f^*(-A_1^{\mathsf{T}}u - A_2^{\mathsf{T}}w-\eta). \end{aligned} \tag{3.11}$$

First we need to establish the conditions under which the saddle value of (3.8) exists. If it exists, then the value of (3.9) and (3.10) are both equal to this saddle value. Assume that ξ and η are fixed and (3.7) has an nonempty bounded feasible region. Denote the domain of the primal variable x by X and the domain of the dual variable $y^{\mathsf{T}} \equiv (u^{\mathsf{T}}, w^{\mathsf{T}})$ by Y.

Theorem 3.10 ([101, Corollary 37.3.1]) *Let $\mathcal{L}$ be a closed proper concave-convex function on $\mathbb{R}^m \times \mathbb{R}^n$ with effective domain $X \times Y$. If either X or Y is bounded, the saddle value of $\mathcal{L}$ exists.*

The next theorem shows that the optimal value function of (3.7) is a piecewise quadratic function of the perturbations ξ and η. It is a direct extension of Sun [118, Proprosition 2.2.4] which deals with monotropic piecewise quadratic programming subject to equality constraints.

Theorem 3.11 *If the feasible region of (3.7) is bounded and $\phi(\xi^0,\eta^0)$ is finite for some (ξ^0,η^0), then $\phi(\xi,\eta^0)$ and $-\phi(\xi^0,\eta)$ are proper, closed, convex and piecewise quadratic functions of ξ and η respectively.*

Proof: Since the feasible region of (3.7) is bounded, a saddle value exists for $\mathcal{L}$ from Theorem 3.10. This implies that the optimal values of (3.9) and (3.10) coincide and are equal to the saddle value.

First consider a right hand side perturbation in (3.7). Using strong duality,

$$\begin{aligned}
\phi(\xi,\eta^0) &= \inf_{x\in\mathbb{R}^n} F(x,\xi,\eta^0) \\
&= \sup_{u\in\mathbb{R}^{m_1},w\in R_+^{m_2}} G(u,w,\xi,\eta^0) \\
&= \sup_{u\in\mathbb{R}^{m_1},w\in R^{m_2}} -u^\mathsf{T}\xi_1 - w^\mathsf{T}\xi_2 - G_0(u,w) \\
&= G_0^*(-\xi_1,-\xi_2),
\end{aligned}$$

where

$$G_0(u,w) = u^\mathsf{T}b_1 + w^\mathsf{T}b_2 + f^*(-A_1^\mathsf{T}u - A_2^\mathsf{T}w - \eta^0) + \begin{cases} 0, & w \geq 0, \\ +\infty, & w \ngeq 0. \end{cases} \tag{3.12}$$

$G_0(u,w)$ is proper, convex and piecewise quadratic since the conjugate function f^* is convex as a result of properties P1 and P2 (page 30). Applying P1 and P2 again, $\phi(\xi,\eta^0) = G_0^*(-\xi_1,-\xi_2)$ is proper, piecewise quadratic and convex in ξ.

Next we prove the claim for $-\phi(\xi^0,\eta)$. Since by assumption $\phi(\xi,\eta)$ is finite for some (ξ^0,η^0), (3.7) remains feasible for all η for fixed ξ^0, so $\phi(\xi^0,\eta)$ is proper. From strong duality,

$$\begin{aligned}
-\phi(\xi^0,\eta) &= -\inf_{x\in\mathbb{R}^n} F(x,\xi^0,\eta) = \sup_{x\in\mathbb{R}^n}\{-F(x,\xi^0,\eta)\} \\
&= \sup_{x\in\mathbb{R}^n}\inf_{u\in\mathbb{R}^{m_1},w\in R_+^{m_2}} -f(x) - \eta^\mathsf{T}x - u^\mathsf{T}(A_1x - b_1 - \xi_1^0) - w^\mathsf{T}(A_2x - b_2 - \xi_2^0) \\
&= \sup_{x\in\mathbb{R}^n}\{-\eta^\mathsf{T}x - F_0(x)\} = F_0^*(-\eta)
\end{aligned}$$

where

$$F_0(x) = f(x) + \begin{cases} 0, & A_1x = b_1 + \xi_1^0, A_2x \leq b_2 + \xi_2^0, \\ +\infty, & \text{otherwise}, \end{cases}$$

is a convex piecewise quadratic function. It follows that $-\phi(\xi^0,\eta)$ is convex, proper and piecewise quadratic. ∎

3.3 MQSPs as PQPs

The results in previous sections will now be gathered to demonstrate that all subproblems in the MQSP (1.12)–(1.13) are convex piecewise quadratic programs. We examine the structure of the piecewise quadratic recourse functions to show when they change representation. Their differential properties are also studied leading to expressions for elements of subgradients and generalized Hessians.

Proposition 3.12 *Each objective functions $f_t^{k_t}$ in (1.12) and (1.13) is convex, piecewise quadratic and Lipschitz continuous.*

Proof: We will prove the claim on a single branched deterministic tree to simplify notation. As each stage t stochastic parameter ξ_t has finite support, and the recourse function of each subproblem is a probability weighted sum of the optimal value functions of its descendents. It is straight forward to extend the result from a single branched tree to one with a more general tree structure.

Consider the stage T subproblem. Since a convex quadratic function is piecewise quadratic, and the right hand side perturbation $V_T x_{T-1}$ is a linear transformation of x_{T-1}, Theorem 3.11 implies that the stage $T-1$ recourse function $Q(x_{T-1})$ is convex and piecewise quadratic in x_{T-1}. It is also locally Lipschitz since it is piecewise smooth. By assumption, H_{T-1} is positive semi-definite, so the stage $T-1$ objective is convex, piecewise quadratic and Lipschitz. By repeatedly applying Theorem 3.11 and observing that all H_t are positive semi-definite, the above argument shows that the objectives of all stages have the stated properties. ∎

Corollary 3.8 can be used to strengthen Proposition 3.12 to show when $f_t^{k_t}$ is continuously differentiable.

Proposition 3.13 *Each objective function $f_t^{k_t}$ of subproblem (t, k_t) in (1.12) and (1.13) is convex, piecewise quadratic,* LC^1 *and* SC^1 *at $x_t^{k_t}$ if all its descendent problems $(t+1, k), k \in D_{t+1}(k_t)$ with constraint right hand side $h_{t+1}^k - V_{t+1}^k x_t^{k_t}$ have unique optimal Lagrange multipliers. A sufficient condition is if all descendent problems satisfy the linear independence constraint qualification.*

Proof: By Corollary 3.8, the optimal value function of a convex program with perturbed right hand side is differentiable if and only if it has unique optimal Lagrange multiplier. So, if for all $k \in D_{t+1}(k_t)$, problem $(t+1, k)$ has unique Lagrange multiplier, then the optimal value functions $\mathcal{Q}_{t+1}^k$ are all differentiable at $x_t^{k_t}$. Combining this with Proposition 3.12, it follows that $f_t^{k_t}$ is convex, piecewise quadratic and differentiable at $x_t^{k_t}$. Since the derivatives of differentiable piecewise quadratic functions are continuous piecewise linear and therefore Lipschitz, the objective function is LC^1 at $x_t^{k_t}$. $f_t^{k_t}(x_t^{k_t})$ is also SC^1 because a piecewise smooth function is semismooth.

To prove the second half of the proposition, recall that the LICQ is a sufficient condition for ensuring unique Lagrange multiplier for smooth optimization. Since the stage T problems of (1.12)–(1.13) are convex quadratic programs, LICQ is enough to guarantee that they have unique Lagrange multipliers and therefore differentiable optimal value functions. Clearly, the objective functions of the stage $T-1$ ancestor problem are differentiable, so LICQ is sufficient to ensure unique multipliers. The conclusion follows by repeating the argument backward to the stage t problem. ∎

LICQ always holds if the constraint matrix has full row rank. Unfortunately this is not true if all variables are bounded. It is straight forward to verify that if the quadratic part of any nodal problem is replaced by a convex piecewise quadratic function, Proposition 3.12 is still valid. If the new convex piecewise quadratic function is differentiable, then Proposition 3.13 also holds.

Writing $u(x_t^{k_t})$ as an optimal Lagrange multiplier of the (t, k_t) subproblem, an element of the subgradient of $f(x_t^{k_t})$ is given by (3.3)

$$H_t^{k_t} x_t^{k_t} + c_t^{k_t} + \sum_{k \in D_{t+1}(k_t)} p_{t+1}^k (V_{t+1}^k)^\intercal u(x_{t+1}^k) \in \partial f(x_t^{k_t}). \tag{3.13}$$

This of course simplifies to become the gradient if $u(x_t^{k_t})$ is unique for all subproblems.

As each optimal value function $\mathcal{Q}$ is a piecewise quadratic function, it is twice differentiable almost everywhere. An element of the generalized Hessian of $f(x_t^{k_t})$

can be calculated by

$$H_t^{k_t} + \sum_{k \in D_{t+1}(k_t)} p_{t+1}^k (V_{t+1}^k)^\intercal \mathcal{G}(x_{t+1}^k) V_{t+1}^k, \tag{3.14}$$

where $\mathcal{G}(x_{t+1}^k)$ is an element of the generalized Hessian of the optimal value function $\mathcal{Q}(x_{t+1}^k)$. The calculations of $\mathcal{G}(x_{t+1}^k)$ will be discussed in details in Sections 3.3.1 and 3.3.2.

3.3.1 Strictly Convex MQSPs

If H_t in (1.12) and (1.13) are assumed to be positive definite, then the objective function is strictly convex.

More insight on the structure of the piecewise quadratic objective of (1.12) and (1.13) can be gained by applying Theorem 3.6. For each stage t subproblem with ancestor solution $\hat{x}_{t-1}$, if LICQ is not satisfied at the solution $x_t^*(\hat{x}_{t-1})$, then there may exist multiple optimal Lagrange multipliers. In this case, the optimal value function is not differentiable and $\hat{x}_{t-1}$ clearly lies on the boundary of different quadratic pieces. Since all constraints are linear, LICQ can only fail on a set of $\hat{x}_{t-1}$ with measure zero. In the following, we assume LICQ holds at $x_t^*(\hat{x}_{t-1})$. Since H_t is positive definite, the second order sufficiency condition is satisfied if the recourse term in the objective is C^2 at $x_t^*(\hat{x}_{t-1})$. The conditions of Theorem 3.6 are satisfied if strict complementarity holds at $x_t^*(\hat{x}_{t-1})$. It follows that the optimal value function $\mathcal{Q}_t(x_{t-1})$ is C^2 at $\hat{x}_{t-1}$ and $\hat{x}_{t-1}$ is in the interior of a quadratic piece. $\mathcal{Q}_t(x_{t-1})$ can therefore only change representation when 1) the active set changes which is indicated by loss of strict complementarity or LICQ; or 2) the optimal solution $x_t^*(\hat{x}_{t-1})$ lies on the boundary of different quadratic pieces of the stage t objective. As there are finitely many combinations of active constraints in each stage and the constraints are linear, this partitions the feasible region of each subproblem into a finite number of convex polyhedra.

Before discussing how to calculate the generalized Hessian of the optimal value function, we give an example of a strictly convex PQP subject to right hand side

perturbation to demonstrate the piecewise quadratic nature of the optimal value function.

Example 3.14 *Consider the following perturbed piecewise quadratic program*

$$\begin{aligned} Q(x) \equiv \min_{y \in \mathbb{R}^2} \quad & \tfrac{1}{2}y_2^2 - y_1 - 2y_2 + \begin{cases} \tfrac{1}{2}y_1^2, & P1 : y_1 \geq 0 \\ y_1^2, & P2 : y_1 \leq 0 \end{cases} \\ \text{s.t.} \quad & y_1 + y_2 \leq x_1, \\ & y_1 \leq x_2, \\ & y_2 \leq x_2. \end{aligned} \tag{3.15}$$

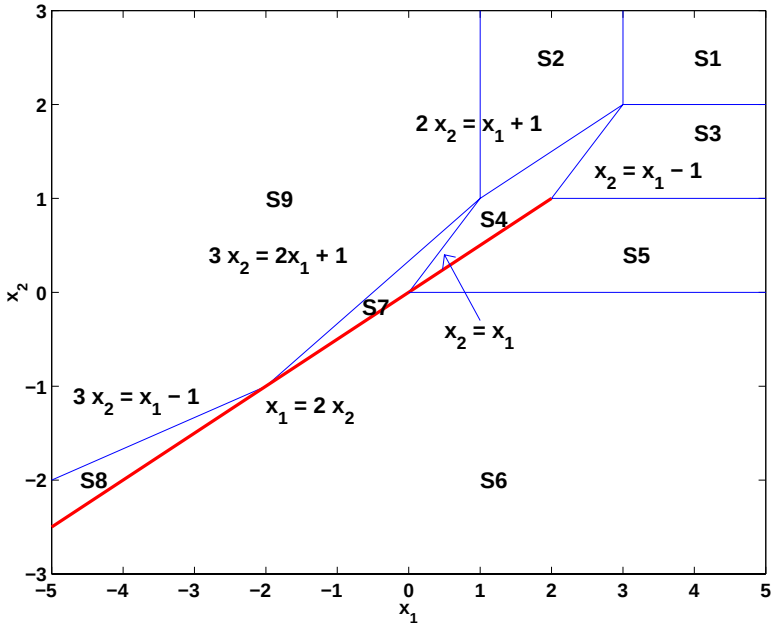

Figure 3.2: Example of perturbed PQP

This can be thought of as an intermediate stage problem of a MQSP. The objective is strictly convex and piecewise quadratic as it contains a recourse function from subsequent stages. The linear constraints are perturbed by the ancestor solution x. The piecewise quadratic optimal value function $Q(x)$ is shown in Figure 3.2. In Table 3.1, the first row identifies the convex polyhedral convex set, labelled S1 to S9, which partitions the domain of $Q(x)$; the row 'obj' gives the piece of the piecewise quadratic objective the optimal solution $y^*(x)$ belongs to; $\mathcal{A}(x)$ is the active constraint set, $y^*(x)$ is the optimal solution and $u^*(x)$ is the Lagrange multiplier.

	S1	S2	S3	S4	S5
obj.	P1	P1	P1	P1	P1
$\mathcal{A}(x)$	$\emptyset$	1	3	1,3	2,3
$y^*(x)$	$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$	$\frac{1}{2}\begin{pmatrix} x_1 - 1 \\ x_1 + 1 \end{pmatrix}$	$\begin{pmatrix} 1 \\ x_2 \end{pmatrix}$	$\begin{pmatrix} x_1 - x_2 \\ x_2 \end{pmatrix}$	$\begin{pmatrix} x_2 \\ x_2 \end{pmatrix}$
$u^*(x)$	$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$	$\frac{1}{2}\begin{pmatrix} 3 - x_1 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 2 - x_2 \end{pmatrix}$	$\begin{pmatrix} 1 - x_1 + x_2 \\ 0 \\ 1 + x_1 - 2x_2 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 1 - x_2 \\ 2 - x_2 \end{pmatrix}$

	S6	S7	S8	S9
obj.	P2	P2	P2	P2
$\mathcal{A}(x)$	2,3	1,3	1,2	1
$y^*(x)$	$\begin{pmatrix} x_2 \\ x_2 \end{pmatrix}$	$\begin{pmatrix} x_1 - x_2 \\ x_2 \end{pmatrix}$	$\begin{pmatrix} x_2 \\ x_1 - x_2 \end{pmatrix}$	$\frac{1}{3}\begin{pmatrix} x_1 - 1 \\ 2x_1 + 1 \end{pmatrix}$
$u^*(x)$	$\begin{pmatrix} 0 \\ 1 - 2x_2 \\ 2 - x_2 \end{pmatrix}$	$\begin{pmatrix} 1 - 2x_1 + 2x_2 \\ 0 \\ 1 + 2x_1 - 3x_2 \end{pmatrix}$	$\begin{pmatrix} 2 - x_1 + x_2 \\ x_1 - 3x_2 \\ 0 \end{pmatrix}$	$\frac{1}{3}\begin{pmatrix} 5 - 2x_1 \\ 0 \\ 0 \end{pmatrix}$

Table 3.1: Solution to example PQP (3.15)

This example shows clearly that changes in quadratic pieces in $Q(x)$ occur when the active set changes and one or more Lagrange multipliers becomes zero or when the optimal solution $y^*(x)$ switches from P1 to P2 in the objective. $Q(x)$ is continuously differentiable everywhere except for the ray $\{x_1 = 2x_2, x_1 \leq 2\}$ when the active constraint gradients are linearly dependent and $Q(x)$ is only Lipschitz continuous.

As each objective f in (1.12) and (1.13) is piecewise quadratic, the generalized Hessian is either the singleton $\nabla^2 f(x)$ if f is twice differentiable at x or the convex hull of the Hessians of all the quadratic pieces x belongs to. In this example, we see that

$$\mathcal{G}(\begin{bmatrix} 0 \\ 2 \end{bmatrix}) = \begin{bmatrix} 2/3 & 0 \\ 0 & 0 \end{bmatrix},$$

$$\mathcal{G}(\begin{bmatrix} 1 \\ 2 \end{bmatrix}) = co\left\{ \begin{bmatrix} 2/3 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1/2 & 0 \\ 0 & 0 \end{bmatrix} \right\}.$$

Since all the recourse functions in MQSP (1.12)–(1.13) are piecewise quadratic, the second derivatives exist almost everywhere and the generalized Hessian is everywhere well defined. The following discusses how the generalized Hessian can be calculated under various assumptions.

If the conditions of Theorem 3.6 are satisfied at the solution $x_t^*(\hat{x}_{t-1})$, then the optimal value function is twice continuously differentiable and the Hessian is given by [32, Equation 5.5.3]

$$\nabla^2 \mathcal{Q}(\hat{x}_{t-1}) = V_t^\intercal \left(\bar{A}_t G_t^{-1} \bar{A}_t^\intercal\right)^{-1} V_t, \tag{3.16}$$

where G_t is the Hessian of the stage t objective at $x_t^*(\hat{x}_{t-1})$ and $\bar{A}_t$ is the matrix of active constraint gradients.

If SCC fails, then it is likely that a constraint is switching from active to inactive and $\hat{x}_{t-1}$ lies on the boundary of the piecewise quadratic optimal value function. In this case, (3.16) will give the element of the generalized Hessian that corresponds to the piece that has the same active set.

Now consider the case when LICQ is not satisfied at $x_t^*(\hat{x}_{t-1})$. Since the constraints are linear, this can only happen on a set of x_{t-1} with measure zero. Since each quadratic piece has closed domain, there exists an $\epsilon > 0$ such that for all x_{t-1} in an ϵ neighbourhood of $\hat{x}_{t-1}$ that lead to feasible stage t problems, if the active constraints at $x_t^*(x_{t-1})$ are linearly independent then the stage t optimal active sets satisfy

$$\mathcal{A}(x_t^*(x_{t-1})) \subseteq \mathcal{A}(x_t^*(\hat{x}_{t-1})).$$

Therefore all non-degenerate quadratic pieces active at x_{t-1} are given by active sets in stage t with linearly independent active constraint gradients and each of these active set are subset of the active set at $x_t^*(x_{t-1})$. When LICQ is not satisfied, $\bar{A}$ does not have full row rank and $(\bar{A}G^{-1}\bar{A}^\intercal)$ in (3.16) is singular. There are possibly more than one linearly independent subsets of the active constraints such that the solution is still optimal if only those constraints are present which can lead to different optimal Lagrange multipliers. Such a linearly independent subset of active constraints is returned if the problem is solved by an active set method. From Theorem 3.7, the Lagrange multiplier associated with this active set is a subgradient of the optimal value function of the stage t problem. Equation (3.16) will give an element of the generalized Hessian associated with that subgradient. Although unlikely, the subgradient and generalized Hessian may not correspond to a non-

degenerate active quadratic piece at $x_t^*(x_{t-1})$. This is undesirable as the quadratic piece obtained is only valid on a set of measure zero and can lead to problem in calculating recourse information of ancestor problem as we will discuss later in this section. Louveaux [72] described a finite procedure which can guarantee that the gradient and Hessian generated are in fact those of a non-degenerate active quadratic piece.

Next consider the case when SOSC fails. Since the stage t objective function f_t is strictly convex by assumption, this can only happen if the solution $x_t^*(\hat{x}_{t-1})$ lies on a boundary between pieces of the piecewise quadratic objective. The optimality conditions are

$$\sum_{j \in J(x)} \lambda_j (G_j x + g_j) + A^\intercal u = 0$$

$$\sum_{j \in J(x)} \lambda_j = 1$$

$$\lambda \geq 0, u_i \geq 0, i = 1, \ldots, r,$$

where $J(x)$ is the set of quadratic pieces that are active at x and constraints 1 to r are the inequalities.

Suppose the objective f_t is LC^1 at $x_t^*(\hat{x}_{t-1})$. Since all active quadratic pieces have the same gradient as f_t at $x_t^*(\hat{x}_{t-1})$, the solution satisfies the optimality condition of all QPs with any active quadratic piece as the objective. By letting G_t be the Hessian of one of the active quadratic piece, (3.16) yields the Hessian of the optimal value function of the QP formed with the chosen quadratic objective. The resulting Hessian is an element of the generalized Hessian of $\mathcal{Q}_t$.

However, if the objective f_t is only Lipschitz continuous at the solution, $x_t^*(\hat{x}_{t-1})$ may not satisfy the optimality condition of QPs formed by any single one of the active quadratic functions. Substituting the Hessian of these quadratic functions into (3.16) will return the Hessian of the optimal value function of that QP calculated at its solution, which in general does not belong to the generalized Hessian of $\mathcal{Q}_t$. To obtain a correct generalized Hessian, it is necessary to restrict the domain of x_t by including the boundary constraints of the current quadratic piece on which f_t is

nonsmooth. This ensures that the recourse Hessian is calculated at a solution to the nonsmooth piecewise quadratic program. This can be achieved by applying results from Louveaux [72] discussed in Section 2.4.5. Recall the Kuhn-Tucker conditions for the stage t problem (2.32) and (2.33)

$$\begin{bmatrix} G_t & (W_{1t})^\intercal & (W_{2t})^\intercal & E_t^\intercal & 0 & 0 \\ W_{1t} & 0 & 0 & 0 & 0 & 0 \\ W_{2t} & 0 & 0 & 0 & I & 0 \\ E_t & 0 & 0 & 0 & 0 & I \end{bmatrix} \begin{pmatrix} x \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} -q_t \\ h_{1t} - V_{1t}\hat{x}_{t-1} \\ h_{2t} - V_{2t}\hat{x}_{t-1} \\ d_t \end{pmatrix} \tag{3.17}$$

$$\lambda_2, \lambda_3, s_1, s_2 \geq 0.$$

The scenario index k has been dropped to simplify notation. Here W_{1t} and W_{2t} represent the equality and inequality parts of the stage t constraints. The constraint $E_t x \leq d_t$ gives the domain of the current quadratic piece, s_1 and s_2 are slack variables for the inequality constraints. Denote by $r = (x, \lambda_1, \lambda_2, \lambda_3, s_1, s_2)$ the primal, dual and slack variables to the stage t problem. Let B be an optimal basis of (3.17) for right hand side $h_t - V_t\hat{x}_{t-1}$, so B is square and nonsingular. For $\hat{x}_{t-1}$ a solution is given by

$$r(\hat{x}_{t-1}) = B^{-1} \begin{pmatrix} -q_t \\ h_t - V_t\hat{x}_{t-1} \\ d_t \end{pmatrix}. \tag{3.18}$$

For other x_{t-1}, the primal and dual variables using the same basis are

$$r(x_{t-1}) = r(\hat{x}_{t-1}) + B^{-1} \begin{pmatrix} 0 \\ -V_t(x_{t-1} - \hat{x}_{t-1}) \\ 0 \end{pmatrix}.$$

This is optimal if and only if λ_2, λ_3, s_1 and s_2 remain non-negative. Let

$$E = B^{-1} \begin{pmatrix} 0 \\ V_t \\ 0 \end{pmatrix}$$

and $\bar{r}$ and $\bar{E}$ be rows of r and E that correspond to the non-negative variables λ_2, λ_3, s_1 and s_2. Then the basis B remains optimal if

$$\bar{r}(x_{t-1}) = \bar{r}(\hat{x}_{t-1}) - \bar{E}(x_{t-1} - \hat{x}_{t-1}) \geq 0$$

which is the polyhedron

$$\{x_{t-1} | \bar{E} x_{t-1} \leq \bar{r}(\hat{x}_{t-1}) + \bar{E}\hat{x}_{t-1}\}. \tag{3.19}$$

Clearly, the boundary constraints that are active at $\hat{x}_{t-1}$ are given by the rows of $\bar{r}(\hat{x}_{t-1})$ that are zero.

If the stage t problem is solved by an active set method, then the optimal basis can be identified very easily by removing the columns associated with the dual variables of constraints not considered active by the active set solver and the columns associated with the slack variables of those constraints considered active.

As the Kuhn-Tucker conditions involve the gradient information but not second order information, the only boundary constraints of the piecewise quadratic piece that are necessary in determining the minimum of a piecewise quadratic program are those on which the objective is nondifferentiable. These constraints influence where the solution set lies and therefore the recourse information. The boundary constraint on which the objective is smooth does not play any role in determining the solution set since the objective gradient is continuous across it and it does not change the Kuhn-Tucker condition whether it is present or not. This leads us to speculate that in calculating the Hessian of the stage $t-1$ optimal value function when f_t is nonsmooth at x^*_{t-1}, it is only necessary to include the explicit boundary of each stage t descendent if $x^*_t(x^*_{t-1})$ also lies on a nonsmooth part of f_{t-1}. Otherwise, if f_t is smooth at $x^*_t(x^*_{t-1})$, then we do not need to include E_t and d_t in (2.33). Another way to see why this is plausible is this: the nonsmoothness of the stage $t-1$ objective is caused by non-uniqueness of the stage t Lagrange multiplier and therefore the loss of LICQ in stage t. If the stage t objective is smooth at $x^*_t(x^*_{t-1})$, the boundary of the stage t pieces play no role in the Kuhn-Tucker conditions or in determining the Lagrange multipliers. Therefore we believe it is possible to show that in finding the boundary of a quadratic piece on which the function is nonsmooth, it is necessary to include the boundary of the descendent problems only if a solution of the descendent problem also lies on a nonsmooth part of the objective.

The following example demonstrates the potential difficulties with calculating the optimal value function Hessian when the piecewise quadratic objective func-

tion is not continuously differentiable at the solution. It also shows how boundary constraints of a quadratic piece can be calculated.

Example 3.15

$$\begin{aligned} \phi(x) = \min_y \quad & f(y) \\ \text{s.t.} \quad & y_1 + y_2 \le x, \end{aligned}$$

where

$$f(y) = \begin{cases} f_1(y) \equiv (y_1-1)^2 - \frac{1}{2} + \frac{1}{2}(y_2-1)^2, & P1 : y_1 \le 0, \\ f_2(y) \equiv \frac{1}{2}(y_1+1)^2 + \frac{1}{2}(y_2-1)^2, & P2 : y_1 \ge 0 \end{cases}$$

The minimizer lies on the boundary $\{y \in \mathbb{R}^2 : y_1 = 0\}$ *between* P_1 *and* P_2*. On the boundary,* f *is nonsmooth with*

$$\partial f(y) = \operatorname{conv}\left\{ \begin{pmatrix} -2 \\ y_2 - 1 \end{pmatrix}, \begin{pmatrix} 1 \\ y_2 - 1 \end{pmatrix} \right\}.$$

The optimality conditions are

$$\begin{gathered} \lambda_1 \begin{pmatrix} 2(y_1-1) \\ y_2 - 1 \end{pmatrix} + \lambda_2 \begin{pmatrix} y_1 + 1 \\ y_2 - 1 \end{pmatrix} + \mu \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 0, \\ \lambda_1 + \lambda_2 = 1, \\ \lambda_1, \lambda_2, \mu \ge 0. \end{gathered}$$

The solution is

	$x < -1$	$-1 < x < 1$	$x > 1$
$\mathcal{A}$	1	1	0
μ	$(4-2x)/3$	$1-x$	0
y^*	$\frac{1}{3}\begin{pmatrix} x+1 \\ 2x-1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ x \end{pmatrix}$	$\begin{pmatrix} 0 \\ 1 \end{pmatrix}$
λ	$\begin{pmatrix} 1 \\ 0 \end{pmatrix}$	$\frac{1}{3}\begin{pmatrix} 2-x \\ 1+x \end{pmatrix}$	$\frac{1}{3}\begin{pmatrix} 1 \\ 2 \end{pmatrix}$
$f^*(x)$	$\frac{1}{3}(x^2 - 4x + 5/2)$	$\frac{1}{2}x^2 - x + 1$	$\frac{1}{2}$
$\nabla^2 f^*(x)$	2/3	1	0

Let $x = 0$*, then* $\mathcal{A} = \{1\}$ *and* $\nabla^2 f(x) = 1$*. As* $y_1^*(x) = 0$*, the solution lies on the boundary between the quadratic pieces, where* f *is not differentiable. It is incorrect*

to apply (3.16) without taking into account the boundary of the piecewise quadratic objective as we will now demonstrate.

Since the constraint right hand side is simply x, V in (3.16) is 1 and can be ignored in the calculations below. Applying (3.16) directly using the Hessian of $f_1(y)$ as G, we have

$$\left(\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)^{-1} = \frac{2}{3}.$$

If $f_2(y)$ is used, (3.16) gives

$$\left(\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)^{-1} = \frac{1}{2}.$$

These are clearly incorrect for $x = 0$ and are in fact the Hessians of the optimal value functions if $f_1(y)$ and $f_2(y)$ are respectively minimized subject to the constraint $y_1 + y_2 = x$.

To get the correct optimal value function Hessian, we need to restrict y to the boundary between $P1$ and $P2$, that is, add the constraint $[1\ 0]\, y = 0$ to the set of active constraint before applying (3.16). Repeating the above calculations give:

$$\left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \right)^{-1} = \begin{bmatrix} 1 & -1/4 \\ 1 & 3/4 \end{bmatrix}.$$

$$\left(\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \right)^{-1} = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}.$$

Since only the first constraint involves the variable x, we find that $\nabla^2 f^*(x) = 1$ which is the true Hessian.

Next we show how to obtain the boundary of a quadratic piece. Let $x = 0$, then $y = (0, 0)$ and the constraint $y_1 + y_2 \leq x$ is active. Using the quadratic piece P1, the optimality condition (2.32) is

$$\begin{bmatrix} 2 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \lambda_1 \\ \lambda_2 \\ s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

where the last constraint correspond to the boundary $y_1 = 0$. *Since both constraints are active, the optimal basis is obtained by setting the slack variables zero and choosing the first four columns of the left hand side matrix. From (3.18),*

$$r(0) = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \text{ and } E = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \end{pmatrix}.$$

Since only the Lagrange multipliers are nonnegative, $\bar{r}$ *and* $\bar{E}$ *correspond to the last two rows of* r *and* E *respectively. Equation (3.19) then gives*

$$\{x \in \mathbb{R} | x \leq 1, -x \leq 1\} = \{x \in \mathbb{R} | -1 \leq x \leq 1\}$$

which is the correct boundary of the quadratic piece that contains $x = 0$.

Note also that at $x = 1$, *the Lagrange multiplier is* 0 *even though the constraint is active, that is SCC fails at this point. Using the calculation above will give us* 1 *as an element of the generalized Hessian.*

Despite the objective being only Lipschitz continuous at the solution, the optimal value function $\phi(x)$ *is still* C^2 *for* $-1 < x < 1$ *or* $x > 1$.

3.3.2 Convex Quadratic Programs

If the solution of a convex program satisfies LICQ, SCC and SOSC, standard sensitivity theory [32] gives the Hessian of the optimal value function in a formula similar to (3.16). For a convex QP, SOSC is satisfied if the objective is strictly convex on the null space of the optimal active constraint gradients which in general may not hold. The following calculations show how second order derivative information of a convex QP can be obtained under the assumption of LICQ and SCC. The results can then be extended in the same manner as in Section 3.3.1 to obtain elements of the generalized Hessian of the optimal value function of (1.12)–(1.13) when LICQ or SCC fails or when the objective is not differentiable at the solution.

Consider the convex QP

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \tfrac{1}{2} x^{\mathsf{T}} G x + c^{\mathsf{T}} x \\ \text{s.t.} \quad & Ax \stackrel{=}{\leq} \xi. \end{aligned} \tag{3.20}$$

Denote the set of m binding constraints at an optimal solution $\bar{x}$ as $\bar{A}\bar{x} = \xi$ and the Lagrange multiplier as $\bar{\lambda}$. The optimality conditions are

$$\begin{aligned} \bar{A}\bar{x} &= \xi \\ G\bar{x} + c + \bar{A}^\intercal\bar{\lambda} &= 0. \end{aligned} \tag{3.21}$$

Since $\bar{A} \in \mathbb{R}^{m\times n}$ has linearly independent rows and $m \leq n$, we can perform a QR decomposition.

$$\bar{A}^\intercal = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = [Y\ Z] \begin{bmatrix} R \\ 0 \end{bmatrix}, \tag{3.22}$$

where $R \in \mathbb{R}^{m\times m}$ is nonsingular, $Y \in \mathbb{R}^{n\times m}$ and $Z \in \mathbb{R}^{n\times n-m}$ are the orthonormal basis to the range space of $\bar{A}^\intercal$ and null space of $\bar{A}^\intercal$ respectively. They satisfy the relations $\bar{A}^\intercal = YR, AZ = 0, Y^\intercal Y = I, Z^\intercal Z = I$ and $Y^\intercal Z = 0$.

Writing $\bar{x} = Yy + Zz$, we have

$$\xi = \bar{A}\bar{x} = \bar{A}Yy + \bar{A}Zz = \bar{A}Yy = R^\intercal Y^\intercal Yy = R^\intercal y.$$

This gives $y = R^{-\intercal}\xi$. All points satisfying $\bar{A}x = \xi$ can be expressed as $x = YR^{-\intercal}\xi + Zz$ for $z \in \mathbb{R}^{n-m}$.

We can now reformulate (3.20) as an unconstrained QP

$$\min_{z\in\mathbb{R}^{n-m}} \tfrac{1}{2}(YR^{-\intercal}\xi + Zz)^\intercal G(YR^{-\intercal}\xi + Zz) + c^\intercal(YR^{-\intercal}\xi + Zz).$$

Simplifying and dropping the constant terms gives

$$\min_{z\in\mathbb{R}^{n-m}} \tfrac{1}{2}z^\intercal Z^\intercal GZz + (Z^\intercal c + Z^\intercal GYR^{-\intercal}\xi)^\intercal z.$$

The optimal solution satisfies

$$Z^\intercal GZz = -(Z^\intercal c + Z^\intercal GYR^{-\intercal}\xi) \equiv \hat{b}. \tag{3.23}$$

If SOSC is not satisfied, $Z^\intercal GZ$ is positive semi-definite, but singular. A spectral decomposition gives

$$Z^\intercal GZ = [V\ W]^\intercal D[V\ W], \tag{3.24}$$

where

$$D = \begin{bmatrix} \bar{D} & & & \\ & 0 & & \\ & & \ddots & \\ & & & 0 \end{bmatrix}, \quad \bar{D} = \begin{bmatrix} \alpha_1 & & & \\ & \ddots & & \\ & & & \alpha_r \end{bmatrix}.$$

$V \in \mathbb{R}^{(n-m)\times r}$ contains in each column the eigenvector corresponding to the positive eigenvalues $\alpha_i, i = 1, \ldots, r$ and the columns of $W \in \mathbb{R}^{(n-m)\times(n-m-r)}$ give an orthonormal basis for the eigenspace corresponding to the zero eigenvalue. Since $[V\,W]$ forms an orthonormal basis for $\mathbb{R}^{(n-m)\times(n-m)}$, rewrite $z = Vv + Ww$. The vector $\hat{b}$ can also be decomposed into components in the spaces spanned by the columns of V and W, namely $\hat{b} = Vb_v + Wb_w$ where $b_v = V^\intercal \hat{b}$ and $b_w = W^\intercal \hat{b}$. Equation (3.23) can then be rewritten as

$$Z^\intercal GZ(Vv + Ww) = \hat{b} = Vb_v + Wb_w. \tag{3.25}$$

As W contains the eigenvectors of $Z^\intercal GZ$ for the zero eigenvalues, $Z^\intercal GZW = 0$, so $w \in \mathbb{R}^{n-m-r}$ is free. Equations (3.23) and (3.25) are solvable if and only if $b_w = 0$, that is if $\hat{b}$ is wholly contained in the space spanned by the columns of V. The components v can be obtained from (3.25) by multiplying by $V^\intercal$ to give

$$V^\intercal Z^\intercal GZVv = \bar{D}v = V^\intercal \hat{b},$$
$$v = \bar{D}^{-1} V^\intercal \hat{b}.$$

Retracing the steps, the optimal solution is

$$\begin{aligned} x &= Yy + Zz \\ &= YR^{-\intercal}\xi + Z(Vv + Ww) \\ &= YR^{-\intercal}\xi + Z(V\bar{D}^{-1}V^\intercal \hat{b} + Ww) \\ &= YR^{-\intercal}\xi - ZV\bar{D}^{-1}V^\intercal Z^\intercal (c + GYR^{-\intercal}\xi) + ZWw. \end{aligned}$$

Since $w \in \mathbb{R}^{n-m-r}$ is free, the optimal solution x is not unique if $n > m + r$, that is when SOSC fails. As $\bar{A}^\intercal = YR$, from (3.21),

$$\begin{aligned} \bar{A}^\intercal \bar{\lambda} &= YR\bar{\lambda} = -(G\bar{x} + c). \\ \bar{\lambda} &= -R^{-1}Y^\intercal (G\bar{x} + c) \\ &= -R^{-1}Y^\intercal (GYR^{-\intercal}\xi - GZV\bar{D}^{-1}V^\intercal Z^\intercal (c + GYR^{-\intercal}\xi) + GZWw + c). \end{aligned}$$

The expression for $\bar{\lambda}$ contains the term $-R^{-1}Y^{\mathsf{T}}GZWw$ which at first glance implies that $\bar{\lambda}$ is also not unique. This cannot be true since LICQ is assumed to hold and we now show why $GZWw$ is zero. Since W is an orthonormal basis for the eigenvectors corresponding to the zero eigenvalues of $Z^{\mathsf{T}}GZ$, $w^{\mathsf{T}}W^{\mathsf{T}}(Z^{\mathsf{T}}GZ)Ww = 0$ for all $w \in \mathbb{R}^{n-m-r}$. We can expand that further by taking the eigenvalue decomposition for G, say $G = PD_0P^{\mathsf{T}}$. Then

$$w^{\mathsf{T}}W^{\mathsf{T}}Z^{\mathsf{T}}(PD_0P^{\mathsf{T}})ZWw = (\sqrt{D_0}P^{\mathsf{T}}ZWw)^{\mathsf{T}}(\sqrt{D_0}P^{\mathsf{T}}ZWw) = 0,$$
$$\sqrt{D_0}P^{\mathsf{T}}ZWw = 0.$$

That is $GZWw = P\sqrt{D_0}(\sqrt{D_0}P^{\mathsf{T}}ZWw) = 0$ for all w, or $GZW = 0$.

The Lagrange multiplier becomes

$$\bar{\lambda} = -R^{-1}Y^{\mathsf{T}}(I - GZV\bar{D}^{-1}V^{\mathsf{T}}Z^{\mathsf{T}})(GYR^{-\mathsf{T}}\xi + c), \tag{3.26}$$

which does not depend directly on the solution $\bar{x}$ and is affine in ξ.

Since $\bar{\lambda} = -\nabla\phi(\xi)$, the Hessian of the optimal value function $\phi(\xi)$ is

$$\nabla^2\phi(\xi) = -\nabla_\xi\lambda = R^{-1}Y^{\mathsf{T}}(I - GZV\bar{D}^{-1}V^{\mathsf{T}}Z^{\mathsf{T}})GYR^{-\mathsf{T}}. \tag{3.27}$$

If SOSC is satisfied at the solution, then $Z^{\mathsf{T}}GZ$ is nonsingular, W in (3.24) is vacuous and $D = \bar{D}$. $V\bar{D}^{-1}V^{\mathsf{T}}$ in (3.27) is equivalent to $(Z^{\mathsf{T}}GZ)^{-1}$. It is not necessary to perform the spectral decomposition and (3.27) can be rewritten as

$$\nabla^2\phi(\xi) = R^{-1}Y^{\mathsf{T}}(I - GZ(Z^{\mathsf{T}}GZ)^{-1}Z^{\mathsf{T}})GYR^{-\mathsf{T}}. \tag{3.28}$$

Using the above results, an element of the generalized Hessian of the recourse function can be calculated in an analogous way to (3.16)

$$\nabla^2\mathcal{Q}(x_{t-1}) = V_t^{\mathsf{T}}R^{-1}Y^{\mathsf{T}}(I - G_tZ(Z^{\mathsf{T}}G_tZ)^{-1}Z^{\mathsf{T}})G_tYR^{-\mathsf{T}}V_t. \tag{3.29}$$

where Y, Z and R are obtained from a QR decomposition of $\bar{A}_t$.

The discussion following Example 3.14 can be applied in the convex case to obtain an element of the generalized Hessian of the recourse function in (1.12)–(1.13) if LICQ or SCC fails or if the objective is nonsmooth at the solution.

It is much more difficult to carry out sensitivity analysis on a convex program if the solution does not satisfy the second order sufficiency condition. This is because the optimal solution need not be unique and is in general a convex set. Different solutions can lead to different active sets. Therefore, a change in the active set does not necessarily mean a change in the quadratic pieces in $\phi(\xi)$. However, all optimal solutions must all share the same Lagrange multiplier as a result of Proposition 3.5. This is correct intuitively as the set of Lagrange multipliers is the subdifferential of the optimal value function which by definition must have the same value for all optimal solutions. The following example illustrates the difficulties.

Example 3.16 *Consider a convex QP subject to right hand side perturbations.*

$$\begin{aligned} \phi(x) = \min_{y \in \mathbb{R}^3} \quad & \tfrac{1}{2}y_1^2 - y_2 - y_3 \\ \text{s.t.} \quad & y_2 + y_3 \leq x_1 \\ & y_2 \leq x_2 \\ & y_3 \leq x_2 \\ & y_2, y_3 \geq 0 \end{aligned}$$

where $x_1, x_2 > 0$.

The Kuhn-Tucker condition is

$$\begin{pmatrix} y_1 \\ -1 \\ -1 \end{pmatrix} + \lambda_1 \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} + \lambda_2 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \lambda_3 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \lambda_4 \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix} + \lambda_5 \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.30}$$

The piecewise quadratic optimal value function is shown in Figure 3.3 and details are given in Table 3.2.

This example demonstrates clearly the loss of structure when SOSC fails. Although the three regions $S1, S2, S3$ *all share the same optimal value function, between them, there are seven different optimal active sets. Each of LICQ, SCC and SOSC fails in some active sets in the interior of a piece of the piecewise optimal value function. The optimal value function is continuously differentiable (*C^1*) everywhere except on the ray* $\{x \in \mathbb{R}^2 : x_2 = x_1/2, x_1 \geq 0\}$ *where LICQ is not satisfied.*

	S1			S2		
$\lambda^*(x)$	$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$			$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$		
$\mathcal{A}(x)$	1	1, 4	1, 5	1	1, 3, 4	1, 2, 5
$y^*(x)$	$\begin{pmatrix} 0 \\ \alpha \\ x_1 - \alpha \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ x_1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ x_1 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ \alpha \\ x_1 - \alpha \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ x_1 \end{pmatrix}$	$\begin{pmatrix} 0 \\ x_1 \\ 0 \end{pmatrix}$
C^1	Y	Y	Y	Y	Y	Y
SCC	Y	N	N	Y	N	N
SOSC	N	Y	Y	N	Y	Y
LICQ	Y	Y	Y	Y	N	N

	S3			S4	S5
$\lambda^*(x)$	$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$			$\begin{pmatrix} \beta \\ 1-\beta \\ 1-\beta \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}$
$\mathcal{A}(x)$	1	1, 2	1, 3	1, 2, 3	2, 3
$y^*(x)$	$\begin{pmatrix} 0 \\ \alpha \\ x_1 - \alpha \end{pmatrix}$	$\begin{pmatrix} 0 \\ x_2 \\ x_1 - x_2 \end{pmatrix}$	$\begin{pmatrix} 0 \\ x_1 - x_2 \\ x_2 \end{pmatrix}$	$\begin{pmatrix} 0 \\ x_2 \\ x_2 \end{pmatrix}$	$\begin{pmatrix} 0 \\ x_2 \\ x_2 \end{pmatrix}$
C^1	Y	Y	Y	N	Y
SCC	Y	N	N	N	Y
SOSC	N	Y	Y	Y	Y
LICQ	Y	Y	Y	N	Y

in S1–S3, $x_2 \leq \alpha \leq x_1 - x_2$ in S4, $0 \leq \beta \leq 1$

Table 3.2: Example of a perturbed convex quadratic program

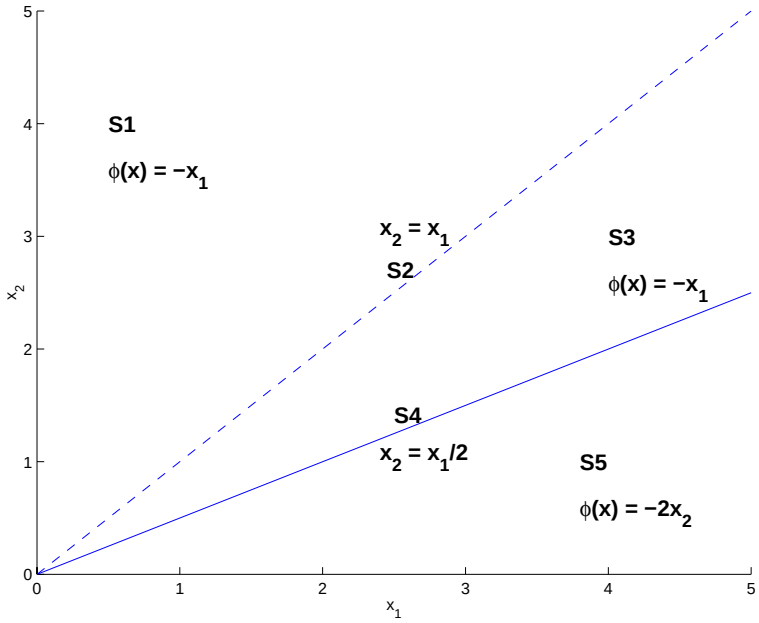

Figure 3.3: Example of a perturbed convex quadratic program

Chapter 4

Generalized Newton Methods for MQSPs

This chapter presents generalized Newton algorithms for solving problem (1.12)–(1.13). We start by examining the special case where all subproblems have strictly convex LC^1 objective and relatively complete recourse. In Section 4.2, we look at how feasibility can be ensured for problems without relatively complete recourse by adding feasibility cuts. In Section 4.3 the first algorithm is modified to handle convex problems that are only Lipschitz continuous.

4.1 Strictly Convex LC^1 MQSPs

The generalized Newton algorithm we present for strictly convex MQSPs combines the nested decomposition method (Section 2.4.2) and the generalized Newton method of Section 2.4.6. As in the nested decomposition method, our algorithm starts from the root node and passes the current iterate to all appropriate descendents. This is continued until the last stage where new QPs are formed and solved. Instead of passing optimality cuts, which are linear supporting hyperplanes to ancestor nodes, gradients and elements of the generalized Hessians of the recourse functions are passed back.

As discussed in Section 3.2.2, the subproblem at each node is a strictly convex

PQP. Each subproblem objective is also assumed to be LC^1, for instance, if each $W_t^{k_t}$ has full row rank. As the objective is piecewise quadratic and second order information is available, a sequential quadratic program algorithm which uses a quadratic model at each iteration is ideally suited to our purpose. At each iterate $x_t^{k_t}$ the current subtree is solved recursively to give the gradient and an element of the generalized Hessian of the recourse function using (3.3) and (3.16) and the discussion afterwards (p.83–86). A Newton direction is then calculated and a linesearch is used to find the next iterate.

Algorithm 4.1 Given stopping criterion $\epsilon_0 : 0 < \epsilon_0 < 1$.

Step 0. $V_1 = 0$, $D_1(k_0) =$. Initialize $Q_t^{k_t} = 0$ for $t = 1, \ldots, T, k_t = 1, \ldots, K_t$.
Set $t = 1$, $k_1 = 1$.

Step 1. Solve (1.13) for $x_t^{k_t}$.

Step 2. Recursively solve the subtree at $x_t^{k_t}$.
If $t = T - 1$
For $k_T \in D_T(k_t)$
Solve the stage T scenario k_T problem (1.13) with $x_t^{k_t}$ as input.
Elseif $t < T - 1$
For $k_{t+1} \in D_{t+1}(k_t)$
$t = t + 1$.
Goto Step 1.
(**) Continue.

Step 3. Calculate the gradient $g_t^{k_t}$ and an element $G_t^{k_t}$ of the generalized Hessian of the objective function at $x_t^{k_t}$ by

$$g_t^{k_t} = H_t^{k_t} x_t^{k_t} + c_t^{k_t} + \sum_{k \in D_{t+1}(k_t)} p_{t+1}^k \nabla \phi_{t+1}^k (x_t^{k_t}), \tag{4.1}$$

$$G_t^{k_t} = H_t^{k_t} + \sum_{k \in D_{t+1}(k_t)} p_{t+1}^k \mathcal{G}_{t+1}^k (x_t^{k_t}), \tag{4.2}$$

where $\nabla\phi^k_{t+1}(x^{k_t}_t)$ and $\mathcal{G}^k_{t+1}(x^{k_t}_t)$, given by (3.3) and (3.16) with $\xi = V_{t+1}x^{k_t}_{t+1}$, are the gradient and an element of the generalized Hessian of the optimal value function of the immediate descendents with respect to $x^{k_t}_t$.

Step 4. Calculate a descent direction $d^{k_t}_t$ by solving

$$\begin{aligned} \min_{d\in\mathbb{R}^{n_t}} \quad & \tfrac{1}{2}d^\intercal G^{k_t}_t d + d^\intercal g^{k_t}_t \\ \text{s.t.} \quad & W^{k_t}_t(x^{k_t}_t + d) \le h^{k_t}_t - V^{k_t}_t x^{k_{t-1}}_{t-1}. \end{aligned} \tag{4.3}$$

Step 5. Optimality check

If $\|d^{k_t}_t\| \le \epsilon_0$

dual variables to (4.3) are the dual variables to the current subproblem.

If $t > 1$

Current subtree optimal.

$t = t - 1$, return to label (**) in Step 2.

Else

optimal solution found; STOP.

Step 6. Linesearch to find a feasible and acceptable step length ρ^k.

Update $x^{k_t}_t = x^{k_t}_t + \rho^k d^{k_t}_t$. Goto Step 3.

In Step 3, p^k_{t+1} is the probability conditional on its immediate ancestor being realized, which is slightly different from that used in (1.14).

A commonly used linesearch criterion for smooth optimization is the Wolfe-Powell condition. For each subproblem, the step $\rho^{k_t}_t > 0$ is chosen such that

$$f(x^{k_t}_t + \rho^{k_t}_t d^{k_t}_t) \le f(x^{k_t}_t) + \sigma_1\rho^{k_t}_t \nabla f(x^{k_t}_t)^\intercal d^{k_t}_t \tag{4.4}$$

$$|\nabla f(x^{k_t}_t + \rho^{k_t}_t d^{k_t}_t)^\intercal d^{k_t}_t| \le -\sigma_2 \nabla f(x^{k_t}_t)^\intercal d^{k_t}_t \tag{4.5}$$

for some given $0 < \sigma_1 < \sigma_2 < 1$. It is well known that if f does not goes to $-\infty$, then there exists an interval such that (4.4) and (4.5) hold. See Fletcher [33] for a detailed discussion on this and other linesearch conditions.

Feasibility cuts have been omitted in the above algorithm. We show how to implement feasibility cuts in Section 4.2 for problems that do not have relatively

complete recourse. However, the feasibility cuts generated may create linearly dependent constraints or cause numerical ill-conditioning.

4.2 Feasibility Cuts

Algorithm 4.1 is now modified to ensure feasibility when relatively complete recourse is not guaranteed. Each time an infeasible subproblem is encountered, a feasibility cut is generated for the immediate ancestor problem and the algorithm returns to that node and finds a feasible solution satisfying the new cut as well. Derivation of feasibility cut was discussed in Section 2.4.1.

Algorithm 4.2 Given stopping criterion $\epsilon_0 : 0 < \epsilon_0 < 1$.

Step 0. $V_1 = 0$, $D_1(k_0) = 1$. Initialize $C_t^{k_t}, d_t^{k_t}$ to be empty, $Q_t^{k_t} = 0$ for $t = 1, \ldots, T, k = 1, \ldots, K_t$. Set $t = 1$, $k_1 = 1$.

Step 1. Find a feasible $x_t^{k_t}$ by solving (1.13) with feasibility cuts

$$C_t^{k_t} x \leq d_t^{k_t}.$$

If this problem is infeasible

If $t = 1$, program (1.12)–(1.13) is infeasible, STOP;

Else

Solve corresponding LP:

$$\begin{aligned}
\min \quad & [y_1\ y_2\ y_3\ y_4]^\intercal e && \text{(4.6a)}\\
\text{s.t.} \quad & W_{1t}^{k_t} x_t + V_{1t}^{k_t} x_{t-1} + y_1 - y_2 = h_{1t}^{k_t}, && \text{(4.6b)}\\
& W_{2t}^{k_t} x_t + V_{2t}^{k_t} x_{t-1} - y_3 \leq h_{2t}^{k_t}, && \text{(4.6c)}\\
& C_t^{k_t} x_t - y_4 \leq d_t^{k_t}, && \text{(4.6d)}\\
& y_1, y_2, y_3, y_4 \geq 0, && \text{(4.6e)}
\end{aligned}$$

where $e = (1, \ldots, 1)^\intercal$. Let μ be the dual variables to (4.6b) and (4.6c). Append $\mu^\intercal \begin{bmatrix} V_t^{k_t} \\ 0 \end{bmatrix}$ and $\mu^\intercal \begin{bmatrix} h_t^{k_t} \\ d_t^{k_t} \end{bmatrix}$ to $C_{t-1}^{k_{t-1}}$ and $d_{t-1}^{k_{t-1}}$ respectively.
Goto label (**) in Step 2.

Step 2. Recursively solve the subtree at $x_t^{k_t}$.

If $t = T - 1$

For $k_T \in D_T(k_t)$

Solve the stage T node k_T problem (1.13) with $x_t^{k_t}$ as input.

If subproblem is infeasible

Solve corresponding LP (4.6a) with C_T, d_T and y_4 vacuous.

Append $\mu^\intercal V_T^{k_T}$ and $\mu^\intercal h_T^{k_T}$ to $C_{T-1}^{k_{T-1}}$ and $d_{T-1}^{k_{T-1}}$ respectively.

Goto Step 1.

Elseif $t < T - 1$

For $k_{t+1} \in D_{t+1}(k_t)$

$t = t + 1$, Goto Step 1.

(**) Continue.

If any subproblem is infeasible

Set $t = t - 1$. Goto Step 1.

Step 3. Calculate the gradient $g_t^{k_t}$ and an element $G_t^{k_t}$ of the generalized Hessian at $x_t^{k_t}$ using (4.1) and (4.2).

Step 4. Calculate a descent direction $d_t^{k_t}$ by solving

$$\begin{aligned} \min_{d \in \mathbb{R}^{n_t}} \quad & \tfrac{1}{2} d^\intercal G_t^{k_t} d + d^\intercal g_t^{k_t} \\ \text{s.t.} \quad & W_t^{k_t}(x_t^{k_t} + d) \leq h_t^{k_t} - V_t^{k_t} x_{t-1}^{k_{t-1}} \\ & C_t^{k_t}(x_t^{k_t} + d) \leq d_t^{k_t}. \end{aligned}$$

Step 5. Optimality check; as in Algorithm 4.1.

Step 6. Linesearch to find feasible and acceptable step length ρ.

If a subproblem is infeasible,

Goto Step 4 to find new descent direction.

Else

Set $x_t^{k_t} = x_t^{k_t} + \rho d_t^{k_t}$. Goto Step 3.

In (4.6a), the equality $\{W_1, V_1, h_1\}$ and inequality $\{W_2, V_2, h_2\}$ constraints are shown separately to clarify the dependence on the slack variables y. Algorithm 4.2 returns

control to the parent node with a new feasible cut as soon as an infeasible descendent is encountered. By using an appropriate cut sharing formula, new cuts can be created for all nodes in the same stage (see the discussion in Section 2.4.2).

4.3 Modifications for Convex MQSPs

We now proceed to modify Algorithm 4.1 for the convex MQSP case. Instead of dealing with strictly convex, piecewise quadratic, LC^1 function in each subproblem, the objectives are in general only convex, piecewise quadratic and Lipschitz continuous. Feasibility cuts can be added in the same way as in Algorithm 4.2 and will not be dealt with here.

As the objective function f may be nonsmooth, we follow the approach in bundle method (Section 2.3.3) and form a piecewise linear approximation to f by collecting appropriate set of subgradients. Suppose we have a set $\{y(l), f(y_l), g(y_l)\}_{l\in L}$ where $g(y_l) \in \partial f(y_l)$. Then the linear approximation of f using the lth piece is

$$f(x) \geq f(y_l) + g(y_l)^\intercal (x - y_l). \tag{4.7}$$

For some arbitrary $x^i \in \mathbb{R}^n$, define

$$\alpha(x^i, y_l) = \alpha_l^i \equiv f(x^i) - f(y_l) - g(y_l)^\intercal (x^i - y_l) \tag{4.8}$$

to be the linearization error of the lth piece at x^i. Since f is convex, $\alpha(x^i, y_l) \geq 0$ for all $x^i, y_l \in \mathbb{R}^n$. Substituting (4.8) into (4.7) gives

$$\begin{aligned} f(x) &\geq f(x^i) - \alpha(x^i, y_l) - g(y_l)^\intercal (x^i - y_l) + g(y_l)^\intercal (x - y_l) \\ &= f(x^i) - \alpha(x^i, y_l) + g(y_l)^\intercal (x - x^i). \end{aligned} \tag{4.9}$$

That is $g(y_l) \in \partial_\epsilon f(x^i)$ with $\epsilon = \alpha(x^i, y_l)$.

Let $\lambda_l, l \in L$ satisfy

$$\lambda \geq 0, \quad \sum_{l\in L} \lambda_l = 1. \tag{4.10}$$

Define the aggregate linearization error $\tilde{\alpha}^i = \sum_{l \in L} \lambda_l^i \, \alpha(x^i, y_l)$ and the aggregate subgradient $\tilde{g}^i = \sum_{l \in L} \lambda_l^i \, g(y_l)$, where λ^i satisfies (4.10). Taking the convex combination of (4.9) gives

$$f(x) \geq f(x^i) - \tilde{\alpha}^i + (\tilde{g}^i)^\intercal (x - x^i). \tag{4.11}$$

Therefore $\tilde{g}^i \in \partial_\epsilon f(x^i)$ with $\epsilon = \tilde{\alpha}^i$.

Since the objective function of each (t, k_t) subproblem is known to be piecewise quadratic, we can strengthen the approximation of f around x^i by using

$$g_l(x^i) = g_l(y_l) + G_l(x^i - y_l) \tag{4.12}$$

if the quadratic piece q_l is active at x^i and $G_l = \nabla^2 q_l$.

Define L as the set of known quadratic pieces and let $J(x^i) \subseteq L$ be the set of quadratic pieces active at x^i. Combining (4.9), (4.11) and (4.12), a linear lower bound of f around x^i is given by

$$f(x) \geq f(x^i) + \max \left(\begin{array}{c} g_j(x^i)^\intercal (x - x^i), \; j \in J(x^i), \\ -\alpha(x^i, y_l) + g_l(y_l)^\intercal (x - x^i), \; l \in L \setminus J(x^i), \\ -\tilde{\alpha}^i + (\tilde{g}^i)^\intercal (x - x^i) \end{array} \right)$$

for λ^i that satisfies (4.10).

Since f is nonsmooth, the search direction may not be a descent direction. The linesearch routine needs to find either a point satisfying the sufficient function value decrease condition or ensure that the new subgradient is different enough from the current bundle to give a better search direction on the next iteration. Given $0 < \sigma < \frac{1}{2}$, $0 < \bar{\rho} < 1$, $\rho_{max} > 1$, a linesearch scheme that satisfies the above requirement, which we will verify in Section 5.2, is given by Kiwiel [65]. Find steps $0 \leq \rho_L^i \leq \rho_R^i \leq \rho_{max}$ such that for $x^{i+1} = x^i + \rho_L^i d^i$ and $y^{i+1} = x^i + \rho_R^i d^i$

$$f(x^{i+1}) \leq f(x^i) + \sigma \rho_L^i v^i. \tag{4.13}$$

Either $\rho_L^i \geq \bar{\rho}$ and $\rho_R^i = \rho_L^i$, (a serious step) or $\rho_L^i = 0$ and $\rho_R^i \geq \bar{\rho}$ (a null step). For the case of linearly constrained problem with constraints $Ax \stackrel{=}{\leq} b$, the sufficient function decrease measure v^i is

$$v^i = -\left((d^i)^\intercal B^i d^i + \tilde{\alpha}^i + (b - Ax^i)^\intercal \mu^i \right)$$

where μ is the vector of Lagrange multiplier for the linear constraints.

Algorithm 4.3 Given the stopping criterion $\epsilon_0 : 0 < \epsilon_0 < 1$.

Step 0. $V_1 = 0$, $D_1(k_0) = 1$. Initialize $Q_t^{k_t} = 0$, $\tilde{\alpha}_t^{k_t}$ and $\tilde{g}_t^{k_t}$ to be vacuous, $L_t^{k_t} = \emptyset$ for $t = 1, \ldots, T$, $k_t = 1, \ldots, K_t$. Set $t = 1$, $k_1 = 1$.

Step 1. Solve (1.13) for $x_t^{k_t}$.

Step 2. Recursively solve the subtree at $x_t^{k_t}$.

If $t = T - 1$

For $k_T \in D_T(k_t)$

Solve the stage T scenario k_T problem (1.13) with $x_t^{k_t}$ as input.

Elseif $t < T - 1$

For $k_{t+1} \in D_{t+1}(k_t)$

$t = t + 1$. Goto Step 1.

(**) Continue.

Step 3. If $L_t^{k_t}$ is empty

Calculate $f_{t,1}^{k_t}$, $g_{t,1}^{k_t}$ and $G_{t,1}^{k_t}$ for an active quadratic piece at $x_t^{k_t}$.

Set $J(x_t^{k_t}) = L_t^{k_t} = \{1\}$.

Else

Calculate $f_l(x_t^{k_t})$ and $g_l(x_t^{k_t})$ for all $l \in L_t^{k_t}$.

Identify the set $J(x_t^{k_t})$ of quadratic pieces that are active at $x_t^{k_t}$.

Step 4. Find search direction $d_t^{k_t}$

$$\min_{d \in \mathbb{R}^n, u \in \mathbb{R}} \quad u + \tfrac{1}{2} d^\intercal B_t^{k_t} d \tag{4.14a}$$

$$\text{s.t.} \quad g_j(x_t^{k_t})^\intercal d \le u, \quad j \in J(x_t^{k_t}) \tag{4.14b}$$

$$-\alpha_l(x_t^{k_t}, y_{t,l}^{k_t}) + g_l(y_{t,l}^{k_t})^\intercal d \le u, \quad l \in L_t^{k_t} \setminus J(x_t^{k_t}) \tag{4.14c}$$

$$-\tilde{\alpha}_t^{k_t} + (\tilde{g}_t^{k_t})^\intercal d \le u \tag{4.14d}$$

$$W_t^{k_t}(x_t^{k_t} + d) \le h_t^{k_t} - V_t^{k_t} x_{t-1}^{k_{t-1}}. \tag{4.14e}$$

Step 5. Optimality check

If $\|d_t^{k_t}\| \leq \epsilon_0$

Current subtree optimal.

Dual variables to (4.14e) are the dual variables to current subproblem.

If $t > 1$

$t = t - 1$, return to label (**) in Step 2.

Else

optimal solution found; STOP.

Step 6. Perform linesearch to get $0 \leq \rho_L \leq \rho_R$ that satisfy (4.13).

Let $f_{t,\bar{l}}^{k_t}$, $g_{t,\bar{l}}^{k_t}$ and $G_{t,\bar{l}}^{k_t}$ be the function value, gradient and Hessian of a quadratic function $q_{\bar{l}}$ that is active at $x_t^{k_t} + \rho_R d_t^{k_t}$.

If $\rho_L \geq \bar{\rho}$ (sufficient function value decrease)

Update $x_t^{k_t} = x_t^{k_t} + \alpha^k d_t^{k_t}$ (Serious step).

Set $\tilde{g}_t^{k_t}$ and $\tilde{\alpha}_t^{k_t}$ vacuous.

Else

$x_t^{k_t} = x_t^{k_t}$ (Null step).

Let λ be the Lagrange multiplier of constraints (4.14b)–(4.14d).

Calculate the new aggregate subgradient and its linearization error.

$$\tilde{g}_t^{k_t} = \lambda^\intercal \begin{pmatrix} g_j(x_t^{k_t}) \\ g_l(y_{t,l}^{k_t}) \\ \tilde{g}_t^{k_t} \end{pmatrix}, \ \tilde{\alpha}_t^{k_t} = \lambda^\intercal \begin{pmatrix} 0 \\ \alpha_l(x_t^{k_t}, y_{t,l}^{k_t}) \\ \tilde{\alpha}_t^{k_t} \end{pmatrix}, \quad \begin{matrix} j \in J(x_t^{k_t}) \\ l \in L_t^{k_t} \setminus J(x_t^{k_t}) \end{matrix} . \quad (4.15)$$

End

If $\bar{l} \notin L_t^{k_t}$

Update $L_t^{k_t} = L_t^{k_t} \cup \bar{l}$

Store $f_{t,\bar{l}}^{k_t}$, $g_{t,\bar{l}}^{k_t}$, $G_{t,\bar{l}}^{k_t}$ and $y_{t,\bar{l}}^{k_t} = x_t^{k_t} + \rho_R d_t^{k_t}$.

Else

Update $f_{t,\bar{l}}^{k_t}$, $g_{t,\bar{l}}^{k_t}$, and $y_{t,\bar{l}}^{k_t}$.

Goto Step 3.

The matrix $B_t^{k_t}$ in the descent direction calculation (4.14) is a positive definite matrix. It can be any element of the generalized Hessian $\mathcal{G}(x_t^{k_t})$ if it is positive

definite, or $\mathcal{G}(x_t^{k_t}) + \delta I$ for some $\delta > 0$. As $x_t^{k_t}$ is not changed after a null step, we can use the same $B_t^{k_t}$ during a sequence of consecutive null steps which is necessary for proving global convergence of the algorithm.

Since each nodal problem objective is only assumed to be convex Lipschitz continuous, the calculation of an element of the recourse generalized Hessian requires more care than when the objective is strictly convex and LC^1. If the objective is not strictly convex on the active constraint manifold, then results in Section 3.3.2 are needed to generate an element of the generalized Hessian of the optimal value function.

If there exist two positive dual variables $\lambda_{\hat{\jmath}}$ and $\lambda_{\tilde{\jmath}}$ to (4.14b) with $\hat{\jmath}, \tilde{\jmath} \in J(x_t^{k_t})$ and $\hat{\jmath} \neq \tilde{\jmath}$, then the objective is only Lipschitz continuous at the solution. Results from Louveaux [72] (see page 84–86) can be used to calculate a correct element of the generalized Hessian. The matrix E in (2.33) is in the worst case of size $c \times n_t$ where c is the sum of the number of inequality constraints including bounds of all direct descendents. This is likely to cause linear dependence of the active constraint gradients. Since we do not include the current piece boundary in the descent direction problem, it is not clear how to identify an optimal basis when these boundaries are added unless the modified QP is solved again using an active set method. Another difficulty is that to calculate a correct generalized Hessian of the stage t optimal value function, we need to guarantee that the subgradient and element of the generalized Hessian returned by stage $t+1$ belong to a non-degenerate active piece. However, since the stage t problem does not satisfy LICQ, this requires an extra procedure to find a non-degenerate active piece. To circumvent these problems, the recourse Hessian can be omitted whenever the objective is nonsmooth at the solution. The subgradient returned is still valid since it is calculated from the Lagrange multipliers, and can therefore be used in subsequent descent direction finding problems. However, since no Hessian information is returned, we cannot test if the current piece is active at later iterates in Step 3 and cannot update the gradient to other iterates if it is active. This may slow down the convergence rate but will not affect the verification of optimality since only the subgradients are required in

Step 4 to calculate the descent direction.

As will be shown in Lemma 5.3, λ satisfies $e^\intercal \lambda = 1$ and $\lambda \geq 0$. So the aggregate subgradient $\tilde{g}_t^{k_t}$ calculated after the first in a consecutive sequence of null steps satisfies $\tilde{g}_t^{k_t} \in \partial_\epsilon f(x_t^{k_t})$ with $\epsilon = \tilde{\alpha}$ as (4.11) showed. Since $x_t^{k_t}$ is not changed after a null step, the updated $\tilde{g}_t^{k_t}$ in subsequent null step is also given by a convex combination of elements of $\partial_\epsilon f(x_t^{k_t})$. The same calculation as (4.11) shows that the updated $\tilde{g}_t^{k_t}$ is an element of the ϵ-subdifferential of f at $x_t^{k_t}$ with ϵ being the updated $\tilde{\alpha}_t^{k_t}$.

Chapter 5

Convergence Analysis

The theoretical properties of Algorithm 4.1, 4.2 and 4.3 are studied in this chapter. We show that all three algorithms converge globally, that is all accumulation points of the sequence generated by these algorithms are stationary points of the multistage quadratic stochastic program (1.12)–(1.13). If the piecewise quadratic objective is differentiable and strictly convex at the solution, then the algorithms can locate the solution in a finite number of iterations. In this case, the iterates converge locally to the solution at a superlinear rate, that is $\lim_{i\to\infty} \frac{\|x^{i+1}-x^*\|}{\|x^i-x^*\|} = 0$. Note that the assumption of strict convexity excludes multistage stochastic linear programming.

5.1 Strictly Convex LC^1 MQSPs

We first prove Algorithm 4.1 converges globally and finitely for problem (1.12)–(1.13) if each subproblem has relatively complete recourse and the piecewise quadratic objective is strictly convex and LC^1 . These conclusions are then extended to Algorithm 4.2 without the relatively complete recourse assumption.

Because of the recursive nature of the algorithms, each nodal problem, except for those in the terminal stage, is treated in the same way. The following proofs apply to each subproblem and in particular to the original problem (1.12) by considering the root node. We can represent each nodal subproblem as

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & Ax \mathrel{\overline{\leq}} b \end{aligned} \tag{5.1}$$

where f is strictly convex, piecewise quadratic and LC^1, $A \in \mathbb{R}^{m\times n}$ and $X = \{x \in \mathbb{R}^n : Ax \mathrel{\overline{\leq}} b\}$ is nonempty.

5.1.1 Global Convergence

The following global convergence proof is adapted from Pang et al. [94] which was concerned with minimizing an unconstrained locally Lipschitz function.

Theorem 5.1 *Let f be strictly convex, piecewise quadratic and LC^1. If the line-search conditions (4.4) and (4.5) are satisfied in step 6 of Algorithm 4.1, that is in each iteration i, the step $\rho^i > 0$ is chosen such that*

$$f(x^i + \rho^i d^i) \leq f(x^i) + \sigma_1 \rho^i g(x^i)^\intercal d^i \tag{5.2}$$

$$|g(x^i + \rho^i d^i)^\intercal d^i| \leq -\sigma_2\, g(x^i)^\intercal d^i \tag{5.3}$$

for some $0 < \sigma_1 < \sigma_2 < 1$ and $g \equiv \nabla f$, then the sequence $\{x^i\}$ generated by Algorithm 4.1 converges globally to the unique minimum of (5.1).

Proof: First, note that as f is strictly convex, there exist constants $c_2 > c_1 > 0$ such that for all quadratic pieces l,

$$0 < c_1\|d\|^2 \leq d^\intercal G_j d \leq c_2\|d\|^2, \text{ for all } d \in \mathbb{R}^n, d \neq 0. \tag{5.4}$$

Since X is nonempty, we can find a feasible starting point x^0 in step 1 of Algorithm 4.1. As only feasible steps are accepted, $x^i \in X$ for all i. Therefore $d = 0$ is a feasible solution to (4.3), and any d^i that solves (4.3) satisfies

$$\tfrac{1}{2}(d^i)^\intercal G^i d^i + (g^i)^\intercal d^i \leq 0. \tag{5.5}$$

If $d^i = 0$, x^i is optimal as will be shown at the end of this proof. If $d \neq 0$, then

$$(g^i)^\intercal d^i \leq -\tfrac{1}{2}(d^i)^\intercal G^i d^i < 0,$$

showing that it is a descent direction. Since f is LC^1, there exists $\rho^i > 0$ which satisfies the approximate linesearch condition (5.2) and (5.3) and $\{f(x^i)\}$ is a strictly decreasing sequence. As f is strictly convex and piecewise quadratic, it is bounded from below. The level set $X^0 = \{x \in X : f(x) \leq f(x^0)\}$ is a compact set, so the sequence $\{x^i\}$ lies in X^0 and has an accumulation point $\bar{x} \in X^0 \subseteq X$.

Next we show that the sequence $\{d^i\}$ is bounded. As $\{x^i\}$ is bounded and each g^i is the gradient of a quadratic function active at x^i, $\{g^i\}$ is also a bounded sequence and there exist a constant $c_0 > 0$ such that

$$|(g^i)^\intercal d^i| < c_0 \|d^i\| \text{ for all } i. \tag{5.6}$$

Combining this with (5.4) and (5.5), we have

$$\tfrac{1}{2} c_1 \|d^i\|^2 \leq \tfrac{1}{2} (d^i)^\intercal G^i d^i \leq |(g^i)^\intercal d^i| \leq c_0 \|d^i\|,$$

so

$$\|d^i\| \leq \frac{2c_0}{c_1}.$$

From (5.6), the sequence $\{(g^i)^\intercal d^i\}$ is also bounded. Now taking limit in (5.2),

$$0 \geq \lim_{i \to \infty} \sigma_1 \rho^i (g^i)^\intercal d^i \geq \lim_{i \to \infty} f(x^{i+1}) - f(x^i) = 0.$$

If $\liminf_{i \to \infty} \rho^i > 0$, then $\lim_{i \to \infty} (g^i)^\intercal d^i = 0$. If $\liminf_{i \to \infty} \rho^i = 0$, consider a subsequence such that $\{g^i\} \to \bar{g}$, $\{d^i\} \to \bar{d}$ and $\{G^i\} \to \bar{G}$, then $\bar{G}$ satisfies (5.4). Taking the limit $i \to \infty$ in (5.3) gives $|\bar{g}^\intercal \bar{d}| \leq -\sigma_2 \bar{g}^\intercal \bar{d}$, that is $\bar{g}^\intercal \bar{d} = 0$ since $\sigma_2 < 1$.

As $\bar{g}^\intercal \bar{d} = 0$, regardless of $\liminf_{i \to \infty} \rho_i$, from (5.5),

$$0 \geq -\tfrac{1}{2} \bar{d}^\intercal \bar{G} \bar{d} \geq \bar{g}^\intercal \bar{d} = 0,$$

so, as $\bar{G}$ is positive definite, $\bar{d} = 0$.

Now let $x \in X$ be arbitrary, as d^i is the unique minimum of (4.3), we have for each i,

$$(x - x^i - d^i)^\intercal (g^i + G^i d^i) \geq 0.$$

As G^i are bounded and f is LC^1, taking the limit $i \to \infty$, we have

$$(x - \bar{x})^\intercal \nabla f(\bar{x}) \geq 0,$$

that is $\bar{x}$ satisfies the optimality condition of the convex program (5.1). Moreover, as f is strictly convex, the minimum is unique, therefore the sequence $\{x^i\}$ converges globally to $\bar{x}$ which is the unique minimum of (5.1). ∎

5.1.2 Finite Convergence

The next theorem shows that Algorithm 4.1 converges finitely. The proof follows closely that of Theorem 3 of Sun [119] which applies to equality constrained problem and assumes an exact linesearch is used.

Theorem 5.2 *Let f in (5.1) be convex and piecewise quadratic. Suppose the iterates $\{x^i\}$ of Algorithm 4.1 converges to a solution x^* of (5.1). Let f be differentiable at x^* and let all quadratic pieces active at x^* be strictly convex. If the unit step is always tested in the linesearch routine in Algorithm 4.1, then $\{x^i\}$ converges to the unique minimum x^* of (5.1) in a finite number of iterations.*

Proof: Recall the partition of the piecewise quadratic objective in Definition 3.9. Suppose $x^* \in P_1 \cap \ldots \cap P_r$ and $x^* \notin P_{r+1} \cup \ldots \cup P_{|L|}$. Since $x^i \to x^*$, there exists $\bar{\imath}$ such that for all $i \geq \bar{\imath}$, $x^i \notin P_{r+1} \cup \ldots \cup P_{|L|}$.

By the optimality of x^*, there exists Lagrange multiplier λ^* such that $\nabla f(x^*) + A^\mathsf{T}\lambda^* = 0$. As ∇f is continuous, $\nabla f(x^*) = G_l x^* + c_l$, for $l = 1, \ldots, r$. So x^* also satisfies the Kuhn-Tucker conditions and is therefore the unique minimizers of the following strictly convex programs:

$$\begin{array}{ll} \min & \frac{1}{2}x^\mathsf{T}G_l x + c_l^\mathsf{T}x \\ \text{s.t.} & Ax \stackrel{=}{\leq} b, \end{array}$$

for $l = 1, \ldots, r$. If follows that $d^* = x^* - x^{\bar{\imath}}$ is the unique minimum of

$$\begin{array}{ll} \min & \frac{1}{2}d^\mathsf{T}G_l d + \nabla f(x^{\bar{\imath}})^\mathsf{T}d \\ \text{s.t.} & Ad \stackrel{=}{\leq} b - Ax^{\bar{\imath}}. \end{array}$$

Therefore a unit step will move from $x^{\bar{\imath}}$ to x^*. The optimal solution of the next descent direction subproblem is then $d^{\bar{\imath}+1} = 0$ and Algorithm 4.1 terminates. ∎

Algorithm 4.1 can be shown to converge superlinearly if an Armijo linesearch is used in step 6, that is, for some $\gamma \in (0,1), \sigma_2 \in (0,1/2)$, x is updated by $x^{i+1} = x^i + \gamma^{r_i} d^i$ where r_i is the smallest nonnegative integer r such that

$$f(x^i + \gamma^r d^i) - f(x^i) \leq \sigma_2 \gamma^r \nabla f(x^i) d^i. \tag{5.7}$$

This follows as ∇f is semismooth and f is strictly convex, so by Theorem 3 of Pang and Qi [96], $\{x^i\}$ converges to x^* Q-superlinearly, that is,

$$\lim_{i \to \infty} \frac{\|x^{i+1} - x^*\|}{\|x^i - x^*\|} = 0.$$

Theorems 5.1 and 5.2 can be extended to Algorithm 4.2 for problems that do not have relatively complete recourse. In Algorithm 4.2, the feasibility cuts are added in the same way as in the nested decomposition method. As noted in Section 2.4.1, there are only finitely many such cuts and a new cut is generated each time. Therefore Algorithm 4.2 either proves problem (5.1) is infeasible, or adds a finite number of constraints to the subproblems. If the assumptions of Theorem 5.1 and 5.2 hold, then Algorithm 4.2 will converge globally to the optimal solution in a finite number of iterations.

5.2 Convex MQSPs

As in the strictly convex case, each nodal subproblem of (1.12)–(1.13) can be represented by (5.1) where f is convex and piecewise quadratic. This section shows that Algorithm 4.3 converges globally to a solution of (5.1). The proof is adapted from Kiwiel [65, Chapter 2]. The original proof, concerned with convex unconstrained minimization problems, is modified to take into account the linear constraints.

Recall the search direction subproblem on the ith iteration:

$$\begin{aligned}
\min_{u,d} \quad & u + \tfrac{1}{2} d^\intercal B^i d && \text{(5.8a)}\\
\text{s.t.} \quad & g_j(x^i)^\intercal d \le u, \quad j \in J(x^i) && \text{(5.8b)}\\
& -\alpha(x^i, y_l) + g_l(y_l)^\intercal d \le u, \quad l \in L^i \setminus J(x^i) && \text{(5.8c)}\\
& -\tilde{\alpha}^{i-1} + (\tilde{g}^{i-1})^\intercal d \le u && \text{(5.8d)}\\
& Ad \mathrel{\overset{=}{\leq}} b^i && \text{(5.8e)}
\end{aligned}$$

where $b^i \equiv b - Ax^i$ and B^i is positive definite. Constraint (5.8d) is only present if the previous step was a null step. Let the dual variables be $\lambda_l^i, l \in L^i$ for (5.8b) and (5.8c), $\tilde{\lambda}^i$ for (5.8d) and μ^i for (5.8e). We will show in Lemma 5.3 that λ satisfies

$$\sum_{l \in L^i} \lambda_l^i + \tilde{\lambda}^i = 1, \quad \tilde{\lambda}^i \ge 0, \quad \lambda_l^i \ge 0, l \in L^i. \tag{5.9}$$

For clarity, all relevant definitions are summarized here:

$$\begin{aligned}
g_l &\in \partial f(y_l),\\
\alpha_l^i &\equiv \alpha(x^i, y_l) \equiv f(x^i) - f(y_l) - g_l^\intercal (x^i - y_l),\\
\tilde{g}^i &\equiv \sum_{l \in L^i} \lambda_l^i g_l + \tilde{\lambda} \tilde{g}^{i-1},\\
\tilde{\alpha}^i &\equiv \sum_{l \in L^i} \lambda_l^i \alpha_l^i + \tilde{\lambda} \tilde{\alpha}^{i-1},\\
v^i &\equiv -\left(\|(B^i)^{-1/2} \tilde{g}^i + (B^i)^{-1/2} A^\intercal \mu^i\|^2 + \tilde{\alpha}^i + (b - Ax^i)^\intercal \mu^i \right)\\
&= -\left(\|\tilde{g}^i + A^\intercal \mu^i\|^2_{(B^i)^{-1}} + \tilde{\alpha}^i + (b - Ax^i)^\intercal \mu^i \right), && \text{(5.10)}\\
w^i &\equiv \tfrac{1}{2} \|\tilde{g}^i + A^\intercal \mu^i\|^2_{(B^i)^{-1}} + \tilde{\alpha}^i + (b - Ax^i)^\intercal \mu^i. && \text{(5.11)}
\end{aligned}$$

As x^i is a feasible solution to (5.1), $b - Ax^i \ge 0$ and $\mu_I^i \ge 0$ where the subscript I denote the inequality constraints, so $(b - Ax^i)^\intercal \mu^i \ge 0$. By convexity of f, $\tilde{\alpha}^i \ge 0$. Therefore $w^i = 0$ if and only if all three terms equal 0. From (4.11) and the discussion at the end of Chapter 4, $\tilde{g}^i \in \partial_\epsilon f(x^i)$ for $\epsilon = \tilde{\alpha}^i$. If $w^i = 0$, then $0 \in \partial f(x^i) + A^\intercal \mu^i$ and $(b - Ax^i)^\intercal \mu^i = 0$, that is μ^i satisfies the complementarity condition of (5.1). Therefore $w^i = 0$ implies that x^i is optimal and w^i can be interpreted as an optimality measure.

The next lemma establishes the dual problem to the search direction finding problem (5.8).

Lemma 5.3 *The variables (d^i, u^i) solve the search direction finding problem (5.8) if and only if the dual variables (λ^i, μ^i) solve*

$$\begin{aligned} \min_{\lambda,\tilde{\lambda},\mu} \quad & \Lambda(\lambda, \tilde{\lambda}, \mu) && \text{(5.12a)} \\ \text{s.t.} \quad & \sum_{l \in L^i} \lambda_l + \tilde{\lambda} = 1 && \text{(5.12b)} \\ & \tilde{\lambda}, \lambda_l, \mu_I \geq 0, && \text{(5.12c)} \end{aligned}$$

where

$$\Lambda(\lambda, \tilde{\lambda}, \mu) = \tfrac{1}{2} \left\| \sum_{l \in L^i} \lambda_l g_l^i + \tilde{\lambda} \tilde{g}^{i-1} + A^\intercal \mu \right\|^2_{(B^i)^{-1}} + \sum_{l \in L^i} \lambda_l \alpha_l^i + \tilde{\lambda} \tilde{\alpha}^{i-1} + \mu^\intercal b^i.$$

Proof: First consider the optimality conditions of (5.8)

$$\begin{gathered} \begin{pmatrix} B^i d^i \\ 1 \end{pmatrix} + \sum_{l \in L^i} \lambda_l^i \begin{pmatrix} g_l^i \\ -1 \end{pmatrix} + \tilde{\lambda}^i \begin{pmatrix} \tilde{g}^{i-1} \\ -1 \end{pmatrix} + \begin{pmatrix} A^\intercal \mu^i \\ 0 \end{pmatrix} = 0, \\ -\alpha_l^i + (g_l^i)^\intercal d^i \leq u^i, \quad \lambda_l^i (u^i + \alpha_l^i - (g_l^i)^\intercal d^i) = 0, \; l \in L^i \\ -\tilde{\alpha}^i + (\tilde{g}^{i-1})^\intercal d^i \leq u^i, \quad \tilde{\lambda}^i (u^i + \tilde{\alpha}^i - (\tilde{g}^{i-1})^\intercal d^i) = 0, \\ A d^i \; \overset{=}{\leq} \; b^i, \quad (\mu^i)^\intercal (b^i - A d^i) = 0, \\ \lambda_l^i, \tilde{\lambda}^i, \mu_I^i \geq 0. \end{gathered} \tag{5.13}$$

The first equation can be simplified as

$$\begin{gathered} B^i d^i + \sum_{l \in L^i} \lambda_l^i g_l^i + \tilde{\lambda}^i \tilde{g}^{i-1} + A^\intercal \mu^i = B^i d^i + \tilde{g}^i + A^\intercal \mu^i = 0, \\ \sum_{l \in L^i} \lambda_l^i + \tilde{\lambda}^i = 1. \end{gathered} \tag{5.14}$$

Now consider problem (5.12),

$$\begin{aligned} \nabla_\mu \Lambda(\lambda^i, \tilde{\lambda}, \mu^i) &= A (B^i)^{-1} \left(\sum_{l \in L^i} \lambda_l^i g_l^i + \tilde{\lambda}^i \tilde{g}^{i-1} + A^\intercal \mu^i \right) + b^i \\ &= -A d^i + b^i, \end{aligned}$$

where the last equality is due to (5.14). For $l \in L^i$, we have

$$\begin{aligned}\frac{\partial \Lambda(\lambda^i, \tilde{\lambda}, \mu^i)}{\partial \lambda_l} &= (g_l^i)^\intercal (B^i)^{-1} \left(\sum_{l \in L^i} \lambda_l^i g_l^i + \tilde{\lambda} \tilde{g}^{i-1} + A^\intercal \mu^i \right) + \alpha_l^i \\ &= -(g_l^i)^\intercal d^i + \alpha_l^i.\end{aligned}$$

Similarly

$$\frac{\partial \Lambda(\lambda^i, \tilde{\lambda}, \mu^i)}{\partial \tilde{\lambda}} = -(\tilde{g}^{i-1})^\intercal d^i + \tilde{\alpha}^{i-1}.$$

Since μ satisfies $\mu_I \geq 0$, for optimality, we must have either $\frac{\partial \Lambda(\lambda^i, \tilde{\lambda}, \mu^i)}{\partial \mu_p} = 0, p = 1, \ldots, m$, or both of $\mu_p^i = 0$ and $\frac{\partial \Lambda(\lambda^i, \tilde{\lambda}, \mu^i)}{\partial \mu_p} \geq 0$ for $p \in I$ as we are dealing with a minimization problem.

Denote the Lagrange multiplier to (5.12b) by u. Since $\lambda_l \geq 0$ for $l \in L^i$, optimality implies that either $\frac{\partial \Lambda(\lambda^i, \tilde{\lambda}, \mu^i)}{\partial \lambda_l} + u^i = 0$, or both of $\lambda_l^i = 0$ and $\frac{\partial \Lambda(\lambda^i, \tilde{\lambda}, \mu^i)}{\partial \lambda_l} + u^i \geq 0$. And an analogous set of relationships holds for $\tilde{\lambda}$.

Summarizing, the optimality conditions for (5.12) can be written as

$$\begin{gathered} u^i \geq (g_l^i)^\intercal d^i - \alpha_l^i, \quad \lambda_l^i (u^i - g_l^\intercal d^i + \alpha_l^i) = 0, \quad l \in L^i, \\ u^i \geq (\tilde{g}^{i-1})^\intercal d^i - \tilde{\alpha}^{i-1}, \quad \tilde{\lambda}^i (u^i - (\tilde{g}^{i-1})^\intercal d^i + \tilde{\alpha}^{i-1}) = 0, \\ A d^i \ \overline{\leq} \ b^i, \quad (\mu^i)^\intercal (b^i - A d^i) = 0, \\ \sum_{l \in L^i} \lambda_l^i + \tilde{\lambda}^i = 1, \\ \lambda_l^i, \ \tilde{\lambda}^i, \ \mu_I^i \geq 0. \end{gathered}$$

As both (5.12) and (5.8) are convex programming problem and they satisfy the same set of optimality conditions, therefore (u^i, d^i) solve (5.8) if and only if the dual variables $(\lambda^i, \tilde{\lambda}^i, \mu^i)$ solve (5.12). ∎

Note that from (5.14), $d^i = -(B^i)^{-1}(\tilde{g}^i + A^\intercal \mu^i)$, so

$$(d^i)^\intercal B^i d^i = \|\tilde{g}^i + A^\intercal \mu^i\|^2_{(B^i)^{-1}}. \tag{5.15}$$

Next we verify that the linesearch condition (4.13) ensures either a sufficient function value decrease or the bundle of subgradients is significantly improved by

adding a new subgradient. At a null step, the linesearch rule (4.13) implies that $\rho_L^i = 0$, $\rho_R^i \geq \bar{\rho}$ and

$$f(y^{i+1}) \geq f(x^i) + \sigma \rho_R^i v^i$$

where $y^{i+1} = x^i + \rho_R^i d^i$ and v^i is given by (5.10). Let $g \in \partial f(y^{i+1})$, then

$$\begin{aligned} -\alpha(x^i, y^{i+1}) + g^\intercal d^i &= -\big(f(x^i) - (f(y^{i+1}) + \rho_R^i\, g^\intercal d^i)\big) + g^\intercal d^i \\ &= f(y^{i+1}) - f(x^i) + (1 - \rho_R^i) g^\intercal d^i \\ &> \sigma\, \rho_R^i v^i + (1 - \rho_R^i) g^\intercal d^i. \end{aligned}$$

By convexity, $\alpha(x^i, y^{i+1}) \geq 0$, so $\rho_R^i g^\intercal d^i > \sigma\, \rho_R^i v^i$. Combining the last two inequalities gives

$$-\alpha(x^i, y^{i+1}) + g^\intercal d^i > \sigma\, \rho_R^i v^i + (1 - \rho_R^i)\sigma v^i = \sigma v^i. \tag{5.16}$$

This observation helps to establish Lemma 5.4 which will be used in the main theorem to show that in a sequence of null steps, the optimality measure w^i is monotonically decreasing.

Lemma 5.4 *Let $\nu \in [0,1]$, $g \in \partial f(y^{i+1})$ and*

$$U(\nu) = \tfrac{1}{2}\|(1-\nu)\tilde{g}^i + \nu g + A^\intercal \mu^i\|^2_{B^{-1}} + (1-\nu)\tilde{\alpha}^i + \nu\alpha + (b^i)^\intercal \mu^i \tag{5.17}$$

and

$$\bar{w} = \min\{U(\nu) | 0 \leq \nu \leq 1\}.$$

If (5.13), (5.14) and (5.16) hold, then

$$\bar{w} \leq w^i - \frac{(1-\sigma)^2 (w^i)^2}{8C^2},$$

where $C = \max\big\{\|\tilde{g}^i + A^\intercal \mu^i\|^2_{B^{-1}},\, \|g\|_{B^{-1}},\, \|\tilde{g}^i\|_{B^{-1}},\, \tilde{\alpha}^i,\, (b^i)^\intercal \mu^i,\, 1\big\}$.

Proof:

$$\begin{aligned} U(\nu) &= \tfrac{1}{2}\|\tilde{g}^i + A^\intercal \mu^i\|^2_{B^{-1}} + \nu(\tilde{g}^i + A^\intercal \mu^i)^\intercal B^{-1}(g - \tilde{g}^i) + \frac{\nu^2}{2}\|g - \tilde{g}^i\|^2_{B^{-1}} \\ &\quad + \tilde{\alpha}^i + \nu(\alpha - \tilde{\alpha}^i) + (b^i)^\intercal \mu^i \\ &= w^i - \nu(\tilde{g}^i + A^\intercal \mu^i)^\intercal B^{-1}(\tilde{g}^i + A^\intercal \mu^i) + \nu(\tilde{g}^i + A^\intercal \mu^i)^\intercal B^{-1}(g + A^\intercal \mu^i) \\ &\quad + \frac{\nu^2}{2}\|g - \tilde{g}^i\|^2_{B^{-1}} + \nu(\alpha - \tilde{\alpha}^i) \\ &= w^i - \nu\|\tilde{g}^i + A^\intercal \mu^i\|^2_{B^{-1}} - \nu(g + A\mu^i)^\intercal d^i + \frac{\nu^2}{2}\|g - \tilde{g}^i\|^2_{B^{-1}} + \nu(\alpha - \tilde{\alpha}^i), \end{aligned}$$

where the last equality follows from (5.14). From (5.16)

$$-g^\tau d^i < -\sigma v^i - \alpha = \sigma\left(\|\tilde{g}^i + A^\tau \mu^i\|^2_{B^{-1}} + \tilde{\alpha}^i + (b^i)^\tau \mu^i\right) - \alpha.$$

Therefore

$$\begin{aligned} U(\nu) &< w^i - \nu\|\tilde{g}^i + A^\tau\mu^i\|^2_{B^{-1}} + \sigma\nu\left(\|\tilde{g}^i + A^\tau\mu^i\|^2_{B^{-1}} + \tilde{\alpha}^i + (b^i)^\tau\mu^i\right) \\ &\quad - \nu\alpha - \nu(\mu^i)^\tau A d^i + \frac{\nu^2}{2}\|g - \tilde{g}^i\|^2_{B^{-1}} + \nu(\alpha - \tilde{\alpha}^i) \\ &= w^i - \nu(1-\sigma)\left(\|\tilde{g}^i + A^\tau\mu^i\|^2_{B^{-1}} + \tilde{\alpha}^i + (b^i)^\tau\mu^i\right) \\ &\quad + \nu(b^i)^\tau\mu^i - \nu(\mu^i)^\tau A d^i + \frac{\nu^2}{2}\|g - \tilde{g}^i\|^2_{B^{-1}} \\ &= w^i - \nu(1-\sigma)(-v^i) + \frac{\nu^2}{2}\|g - \tilde{g}^i\|^2_{B^{-1}}. \end{aligned}$$

Since $\nu(1-\sigma) \geq 0$, $w^i \leq -v^i$ and (5.13) stated that $(\mu^i)^\tau(b^i - Ad^i) = 0$

$$\begin{aligned} U(\nu) &< w^i - \nu(1-\sigma)w^i + \frac{\nu^2}{2}\|g - \tilde{g}^i\|^2_{B^{-1}} \\ &< w^i - \nu(1-\sigma)w^i + \frac{\nu^2}{2}(2C)^2. \end{aligned}$$

Let $\overline{U}(\nu) = w^i - \nu(1-\sigma)w^i + \frac{\nu^2}{2}(2C)^2$. The minimum of $\overline{U}(\nu)$ is attained at $\bar{\nu} = \frac{(1-\sigma)w^i}{4C^2}$. Since $0 \leq w^i \leq \frac{1}{2}C + 2C$ and $C \geq 1$, $\bar{\nu}$ satisfies $0 \leq \bar{\nu} < 1$. Therefore

$$\bar{w} \leq U(\bar{\nu}) < \overline{U}(\bar{\nu}) = w^i - \frac{(1-\sigma)^2(w^i)^2}{8C^2},$$

as claimed. ∎

Now we are ready to prove Algorithm 4.3 converges globally to a solution. The stopping criterion ϵ_0 is set to 0.

Theorem 5.5 *Let $f : \mathbb{R}^n \to \mathbb{R}$ in (5.1) be convex and piecewise quadratic. Let the feasible region X be convex, compact and nonempty. Suppose B^i in the descent direction subproblem (5.8) are uniformly positive definite and uniformly bounded, that is, there exist $0 < c_1 < c_2$ such that $c_1\|d\|^2 \leq \frac{1}{2}d^\tau B^i d \leq c_2\|d\|^2$ for all i and all $d \in \mathbb{R}^n$. Then any accumulation point of the sequence $\{x^i\}$ generated by Algorithm 4.3 is a solution of (5.1).*

Proof: If Algorithm 4.3 stops at the ith iteration, then $d^i = 0$. From the definition of (5.8), the only linear supporting hyperplanes that can be active are those with linearization error $\alpha = 0$, so $\tilde{\alpha}^i = 0$. From (5.15), $d^i = 0$ is equivalent to $\|\tilde{g}^i + A^\intercal \mu^i\| = 0$. Since the dual variables to (5.8) satisfy $0 = (b^i - Ad^i)^\intercal \mu^i$, if $d = 0$, then $(b - Ax^i)^\intercal \mu^i = 0$. This demonstrates that if Algorithm stops, $w^i = 0$ and so x^i is indeed optimal.

Now assume that Algorithm 4.3 does not stop. We need to consider the case when there is an infinite number of serious steps and the case of infinitely many null steps.

Suppose there is an infinite number of serious steps indexed by $i \in I$. Since the linesearch condition (4.13) implies that $\{f(x^i)\}$ is a non-increasing sequence and the feasible set X is assumed to be compact, for any converging subsequence $\{x^i\}_{i\in I_0}$ where $I_0 \subseteq I$, there exists $\bar{x} \in X$ such that $\{x^i\}_{i\in I_0} \to \bar{x}$. Since X is compact, there exists a finite number c such that $f(x) \geq c$ for all $x \in X$. From the linesearch rule (4.13),

$$\begin{aligned} f(x^0) - f(x^i) &= f(x^0) - f(x^1) + \ldots + f(x^{i-1}) - f(x^i) \\ &\geq \sum_{p=0}^{i-1} \sigma\, \rho_L^p(-v^p) \\ &\geq \sum_{p=0, p\in I_0}^{i-1} \sigma\, \rho_L^p(-v^p) \\ &\geq \sigma\bar{\rho} \sum_{p=0, p\in I_0}^{i-1} (-v^p) \\ &\geq \sigma\bar{\rho} \sum_{p=0, p\in I_0}^{i-1} w^p. \end{aligned}$$

Taking the limit as $i \in I_0$ goes to infinity,

$$f(x^0) - c \geq f(x^0) - f(\bar{x}) \geq \sigma\bar{\rho} \sum_{i=0, i\in I_0}^{\infty} w^i.$$

Since the left hand side is bounded and $w^i \geq 0$ for all i, $w^i \xrightarrow{i\in I_0} 0$ and any accumulation point of $\{x^i\}_{i\in I_0}$ is a solution to (5.1).

Now consider the case of a finite number of serious steps but an infinite number of null steps, that is, there exists $\bar{\imath} > 1$ such that for all $i \geq \bar{\imath}, x^i = x^{\bar{\imath}}$. This implies $b^i = b^{\bar{\imath}}$ and the algorithm requires that $B^i = B^{\bar{\imath}}$.

Since the dual variables of (5.8) solve problem (5.12), comparing (5.11) and (5.12a) shows that w^i is equal to the optimal value of (5.12). This can be used to show that $w^{i+1} < w^i$ for $i > \bar{\imath}$. For the $(i+1)$th iteration, the objective of the dual problem is

$$\tfrac{1}{2}\left\| \sum_{l \in L^{i+1}} \lambda_l g_l^{i+1} + \tilde{\lambda}\tilde{g}^i + A^\intercal \mu \right\|^2_{(B^i)^{-1}} + \sum_{l \in L^{i+1}} \lambda_l \alpha_l^{i+1} + \tilde{\lambda}\tilde{\alpha}^i + \mu^\intercal b^{i+1}. \tag{5.18}$$

Since B does not change after a null step, $B^{i+1} = B^i$, so all calculations in a consecutive sequence of null steps are carried out in the same matrix norm. Let $\bar{l} \in L^{i+1}$ be the index such that $g_{\bar{l}}^{i+1}$ is the subgradient added in the ith iteration. For any $\nu \in [0,1]$, the variables (λ, μ) given by

$$\lambda_{\bar{l}} = \nu, \quad \tilde{\lambda} = (1-\nu) \tag{5.19a}$$

$$\lambda_l = 0, l \in L^{i+1} \setminus \bar{l}, \quad \mu = \mu^i \tag{5.19b}$$

are feasible for (5.12) for the $i+1$st dual subproblem. Substituting (5.19) into (5.18) and writing $g = g_{\bar{l}}^{i+1}$, $\alpha = \alpha_{\bar{l}}^{i+1}$, (5.18) becomes $U(\nu)$ defined in (5.17). Therefore w^{i+1} is bounded above by $U(\nu)$ for $\nu \in [0,1]$. From Lemma 5.4, $w^{i+1} \leq w^i - \frac{(1-\sigma)^2 (w^i)^2}{8(C^i)^2}$, where $C^i = \max\left\{ \|\tilde{g}^i + A^\intercal \mu^i\|^2_{(B^i)^{-1}}, \|g_{\bar{l}}^{i+1}\|_{(B^i)^{-1}}, \|\tilde{g}^i\|_{(B^i)^{-1}}, \tilde{\alpha}^i, (b^i)^\intercal \mu^i, 1 \right\}$. Since $\frac{1}{2}\|\tilde{g}^i + A^\intercal \mu^i\|^2_{(B^i)^{-1}} + \tilde{\alpha}^i + (b - Ax^i)^\intercal \mu^i = w^i \leq w^{\bar{\imath}}$ for all $i > \bar{\imath}$, there exists a constant C_1 independent of i such that

$$C_1 \geq \{\|\tilde{g}^i + A^\intercal \mu^i\|^2_{(B^i)^{-1}}, \tilde{\alpha}^i, (b - Ax^i)^\intercal \mu^i, 1\}, \quad i \geq \bar{\imath}.$$

By assumption, X is compact, so $\{x^i\}$ and $\{y^i\}$ are bounded. Therefore $\{g(y^i)\}$ is bounded, $\{\tilde{g}^i\}$ is also bounded as it is formed recursively by convex combinations of $\{g(y^i)\}$. Hence there exists a constant C_2 such that $\max\{\|\tilde{g}^i\|, \|g(y^i)\|\} \leq C_2$ for all $i > \bar{\imath}$. Therefore for $C = \max\{C_1, C_2\} \geq 1$, w satisfies $w^{i+1} \leq w^i - \frac{(1-\sigma)^2 (w^i)^2}{8C^2}$ for all $i \geq \bar{\imath}$. As $\sigma \in (0, \frac{1}{2})$ is fixed and $w^i \geq 0$ for all i, this implies that $w^i \downarrow 0$. This combined with the fact that $x^i = x^{\bar{\imath}}$ for all $i > \bar{\imath}$, implies that $\{x^i\}$ has an

accumulation point $\bar{x} = x^{\bar{l}}$ which is a solution of (5.1). Therefore any accumulation point of the iterates generated by Algorithm 4.3 is a solution of (5.1). ∎

Let the piecewise quadratic objective f be differentiable at x^* and all the active pieces at x^* be strictly convex. If B in the descent direction subproblem (5.8) is given by an element of the generalized Hessian of f whenever it is positive definite, then Theorem 5.2 can be used to show Algorithm 4.3 converges to the solution of (5.1) finitely by replacing all references to Algorithm 4.1 with Algorithm 4.3. Since f is assumed SC^1 and strictly convex at x^*, by using the same argument at the end of Section 5.1, the local rate of convergence of Algorithm 4.3 can also be shown to be superlinear following Theorem 3 of Pang and Qi [96].

Chapter 6

Implementation Issues and Numerical Results

In this chapter, we discuss details of an implementation of Algorithm 4.3. The algorithm was implemented in Matlab 5.3 [80] on a SGI Origin 2000 running IRIX 6.5. Numerical results for random data with different tree structures are presented in Section 6.2. The algorithm was also tested on publicly available stochastic linear programming test problems modified by adding small convex quadratic terms to the objective functions.

6.1 Implementation Issues

6.1.1 Random Data Generation

Small random data sets were generated to test the performance of Algorithm 4.3. Only random right hand sides are considered in the current implementation, though stochasticity in other problem data can be added with no changes to the properties of the algorithm. Given a tree structure and subproblem sizes, random matrices $H_t, c_t, V_t, W_t, t = 1, \ldots, T$ with specified density and distribution of singular values or eigenvalues were generated. This is done by calling Matlab's random matrix generators. The entries are normally distributed and the desired density is achieved by

performing random plane or Jacobi rotations. Each stochastic right hand side $h(\omega_t)$ is calculated from two component, the expected value $\bar{h}_t$ and a random perturbation vector v_t, both of which have normally distributed entries. The set of stochastic right hand sides are given by $h(\omega_t) \in \{\bar{h}_t, \bar{h}_t \pm v_t, \bar{h}_t \pm 2v_t, \bar{h}_t \pm 3v_t\}$. The stochastic right hand sides are temporally independent, so each sibling in the same stage has the same set of descendents. Normally distributed random lower and upper bounds are also generated for each variable.

6.1.2 Linesearch Strategy and Trust Region Method

Function evaluations, especially for early stage problems, are very expensive in Algorithm 4.3 as each function evaluation involves solving a branch of the scenario tree to optimality. A linesearch routine that takes advantage of the piecewise quadratic nature of the objective and the structure of MQSPs is essential to an efficient implementation. A one dimensional piecewise quadratic model is used in the linesearch in the numerical experiments. A step of one is always tested first. Once the optimal step has been bounded, either by the unit step or by extrapolation, the one dimensional quadratic functions and their tangents at the two end points are constructed. The tangents provide piecewise linear lower bounds on the objective function. The next trial step is chosen from the two minima of the two quadratic pieces and the intersections of the quadratics, if they lie within the two end points. The function values at the candidate points are predicted using the quadratic functions and are considered valid if they lie above the piecewise linear lower bound. The point with the lowest valid function value prediction is chosen as the next step. If all of the trial points are invalid, then the midpoint between the two endpoints is used instead (See Figures 6.1 and 6.2).

Let $\hat{q}(\rho) \equiv q(x + \rho d)$ denote a quadratic function that is active at $x + \rho d$ for some $\rho \geq 0$. Then $\hat{q}'(\rho) = \nabla q(x + \rho d)^\top d$. For $0 \leq \rho_L < \rho_R$ with $\hat{q}'(\rho_L) < 0$ and $\hat{q}'(\rho_R) > 0$, let the convex piecewise linear function

$$\underline{f}(\rho) \equiv \max \left(\begin{array}{l} \hat{q}(\rho_L) + (\rho - \rho_L)\hat{q}'(\rho_L), \\ \hat{q}(\rho_R) + (\rho - \rho_R)\hat{q}'(\rho_R) \end{array} \right)$$

be the piecewise linear lower approximation of f defined at $x + \rho_L d$ and $x + \rho_R d$. Denote by $\underline{f}_0 = \min_{\rho_L \le \rho \le \rho_R} \underline{f}(\rho)$, the minimum of the piecewise linear lower bound.

The linesearch is terminated when sufficient function value decrease (4.13) is achieved or if the current function value is very close to the minimum $\underline{f}_0$ of the current linear lower bound $\underline{f}(\rho)$.

Given parameters $c_1 > 1$, $0 < c_2, c_3 < 1$, $0 < \sigma < \frac{1}{2}$, maximum feasible step $\rho_{\max}$ and required function value decrease v, the linesearch algorithm can be stated as

Algorithm 6.1

Step 0. Let $q_L(x)$ be a quadratic function active at $\rho = 0$. Initialize $\rho = 1$, $l = 1$. While $l <$ maximum iteration count

Step 1. Calculate $f(\rho) = q(\rho)$, $g(\rho) = \nabla q(\rho)$ and $G(\rho) = \nabla^2 q(\rho)$.

Step 2. If $g^\intercal d > 0$ or $f > f(x)$, set $\rho_R = \rho$, $q_R = q$.
Else, set $\rho_L = \rho$, $q_L = q$.

Step 3. If $f \le f(x) + \sigma\rho v$, set $\rho_L = \rho$, $q_L = q$, stop.

Step 4. If ρ_R is undefined (Extrapolation)
If $\rho = \rho_{\max}$, Set $\rho_L = \rho_R = \rho$, $q_L(x) = q_R(x) = q$. Stop.
Set $l = l + 1$, $\rho = \min(c_1\rho, \rho_{\max})$.

Step 5. Else (Interpolation)
If $\min(\hat{q}(\rho_L), \hat{q}(\rho_R)) - \underline{f}_0 \le c_3 \underline{f}_0$, stop.
Set $l = l + 1$.
Let $\bar{\rho}_1$ and $\bar{\rho}_2$ correspond to the minima of q_L and q_R
and $\bar{\rho}_3$ and $\bar{\rho}_4$ correspond to the intersections of q_L and q_R.
Calculate $q_L(\bar{\rho}_1)$, $q_R(\bar{\rho}_2)$, $q_L(\bar{\rho}_3)$ and $q_L(\bar{\rho}_4)$
Choose $\bar{\rho}$ corresponding to the minimum $q_p(\bar{\rho}_k)$ such that
$\rho_L + c_2 \le \bar{\rho}_k \le \rho_R - c_2$ and $q_p(\bar{\rho}_k) \ge \underline{f}(\rho)$,
where $p = R$ if $k = 2$, otherwise, $p = L$.
If no such $\bar{\rho}$ exists, set $\rho = \frac{1}{2}(\rho_L + \rho_R)$.

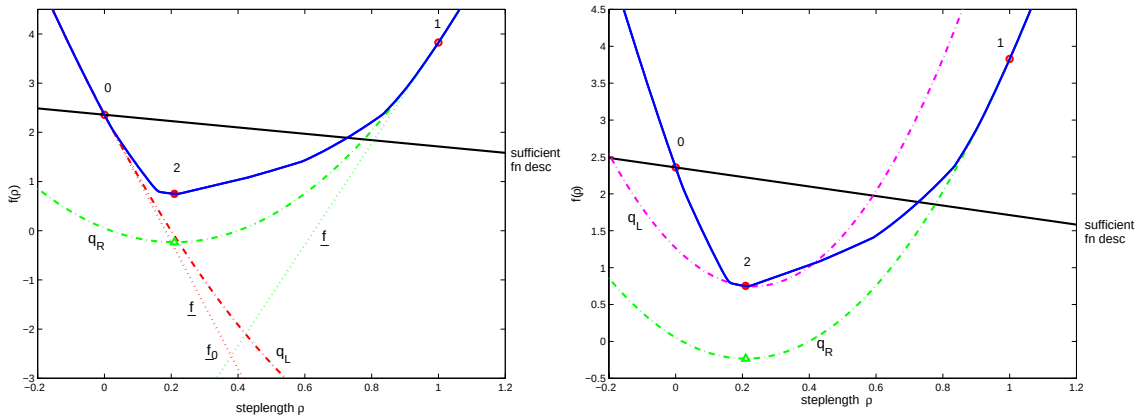

Figure 6.1: Linesearch example 1

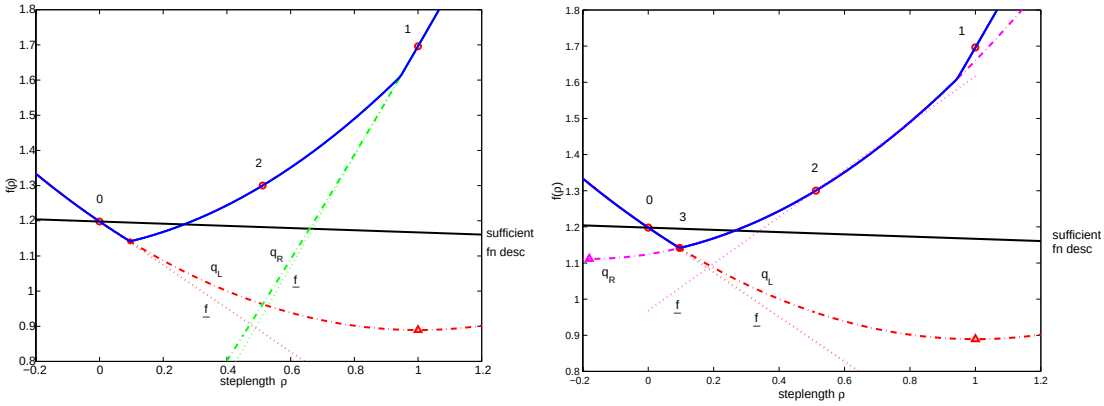

Figure 6.2: Linesearch example 2

Figures 6.1 and 6.2 give two examples of the linesearch. In the first example, the unit step is too large, therefore the new quadratic piece at $\rho = 1$ is identified as q_R. One intersection of q_L and q_R, and the minimum of q_R lie in (ρ_L, ρ_R) and have predicted function values above the linear lower bound (the dotted lines). Since the minimum of q_R has a lower predicted value, it is used as the next trial step. After calculating the function value at the new trial step, the new point is found to have negative slope and satisfies the sufficient function value decrease condition, so the new quadratic piece becomes the new q_L and the linesearch terminates. In the second example, both the minima of q_L and q_R are below the linear lower bound. So

the intersection of q_L and q_R that lies between $\rho_L = 0$ and $\rho_R = 1$ becomes the next trial step. However, the second trial point is still too high and the new quadratic piece is identified as q_R. The third trial point is given by the intersection of q_L and q_R and is the exact minimum of the line.

The value of the parameters used in the linesearch routine are $c_1 = 5$, $c_2 = 0.01$, $c_3 = 0.1$ and $\sigma = 0.01$. As each function evaluation is very expensive, the maximum number of iteration in the linesearch routine is limited to 5. If the linesearch fails to find sufficient function decrease, then $\rho_L = 0$ and $\rho_R > 0$. As discussed in the proof of Theorem 5.5, in this case, the subgradient generated at ρ_R is significantly different from those already in the set leading to a better descent direction.

Recent research in nonmonotone linesearch techniques [44, 120] suggests that they may be beneficial in decreasing the number of function evaluations and cputime usage.

Trust Region Method

A trust region method is also used in conjunction with linesearch when implementing Algorithm 4.3. The trust region radius tries to approximate the size of the current quadratic piece. This is to prevent the algorithm from taking steps that are too large and so that the function is poorly approximated by the current quadratic piece.

From (5.8), the predicted function value decrease is

$$\tilde{\triangle} f = \tfrac{1}{2}(d^i)^\intercal B^i d^i + u^i.$$

and the true function value decrease is

$$\triangle f = f(x^i + \rho^i d^i) - f(x^i).$$

Calculate the ratio

$$r = \frac{\triangle f}{\tilde{\triangle} f}.$$

Let `trr` denotes the current trust region radius, $0 <$ `mintrr` $<$ `maxtrr` are given minimum and maximum allowable values. The following segment of MATLAB [80] code shows how `trr` is commonly updated in the literature

```
r = df/fpred;
if r < c1
   trr = max(trr/5, mintrr);
elseif r > c2
   trr = min(maxtrr, trr*2);
end
```

where $0 < c1 < c2 < 1$ are given parameters. Typical values are $c1 = 0.25$ and $c2 = 0.75$. The problem in the numerical experiments may not be well scaled, so the variables may have vastly different order of magnitude. To compensate for the possible differences in scale of different variables, each variable has its own trust region radius. The above updating scheme is modified to allow `trr` to be a vector if the same size as the variable giving

Algorithm 6.2

```
r = df/fpred;
if r < c1
   trr = max(trr/5, mintrr);
elseif r > c2
   index = find(abs(d)./trr) >= c5;
   trr(index) = trr(index) * 2;
end
```

We introduce the parameter $c5 \in [0, 1]$ to control individual trust region radius. If $c5 = 1$ then only trust region radii that were active in the last direction finding problem are increased. If $c5 = 0$, then all trust region radii are updated uniformly.

Results from preliminary numerical experiments show that using the above trust region updating scheme, the algorithm often still produces very poor search directions that require several interpolation steps in the linesearch and therefore are very expensive in terms of function evaluations.

Algorithm 4.3 is an active set method. Solving a subproblem to optimality requires identifying the right active set in the current problem and the correct set of

active quadratic pieces in the piecewise quadratic objective, which are in turn given by the appropriate active sets in descendent problems. Since all QP subproblems are solved by an active set solver, they can be sped up significantly if good starting points and active sets are provided. As we have seen in Section 3.3, changes of pieces in the piecewise quadratic objective involves constraints changing from active to inactive. This suggest that pieces close to each other should have similar active sets. Suppose the current point is x^i and denote the sequence of trial points tested by the linesearch routine as $\{x^{i,0}, x^{i,1}, \dots\}$ with $x^{i,0} = x^i$. If a step much larger than the size of the quadratic piece at x^i is used, then all the descendent subproblems are likely to have very different active sets when they are solved with right hand side $x^{i,1}$. Using the previous optimal solutions and active sets with $x^{i,0}$ probably would not speed up the solution time by much as the starting points may not even be feasible anymore. Therefore, the function evaluation at $x^{i,1}$ is much more expensive than at a point closer to $x^{i,0}$. Moreover, the quadratic piece at $x^{i,0}$ is possibly a very poor approximation to the objective at $x^{i,1}$ if the step is much larger than the distance between $x^{i,0}$ and the boundary of its associated quadratic piece. Linesearch may have to reduce the step by a significant amount, leading to another very expensive function evaluation since the new point $x^{i,2}$ is far away from $x^{i,1}$ and so is likely to have very different active sets in the descendent problems. On the other hand, if the trust region radius is too small, the algorithm will take more small steps. However, these steps are likely to be relatively cheap since the descendent problems are expected to have similar active sets as in the previous function evaluation.

Since small steps involves few active set changes and should be cheap, we prefer a trust region updating scheme that is more sensitive to poor function approximation. Instead of increasing `trr` for any $r > c2$, `trr` is only increased if r is reasonably close to one and is decreased if r is much bigger than one.

Given $0 < c1 < c2 < 1 < c3 < c4$ and $0 \le c5 \le 1$. Let the step ρ returned by linesearch algorithm 6.1 be `step`. The trust region radius is updated by

Algorithm 6.3

```
step1 = max(step, 0.1);
```

```
trrold = trr;
r = df/fpred;
if r < c1
   trr = trr*max(r,0.1)*0.5;
   trr = min(trr, norm(d,inf)*step1*0.5);
elseif (r > c2) & (r < c3)
   index = find(abs(d)./trr >= c5);
   trr(index) = trr(index)*5;
   trr = min(maxtrr, trr);
elseif r > c4
   trr = min(trr, norm(d,inf)) *step1*0.8;
else
   trr = min(step1*max(1,abs(d)*1.5), trr);
end
trr = max([trr trrold/50 mintrr]);
```

When the actual function value decrease is very small compared to the predicted decrease, `trr` is reduced by a factor depending on r and the new `trr` satisfies `trr` $\leq \frac{1}{2}\max(\rho, 0.1)\|d\|_\infty$. For $c_1 \leq r \leq c_2$ or $c_3 \leq r \leq c_4$, `trr` is updated so that it has approximately the same magnitude as the last d. A comparison of the two different trust region updating schemes with different parameters will be presented in Section 6.2.1.

6.1.3 Presolve Strategies and other Speedup Schemes

When Algorithm 4.3 is applied to (1.12)–(1.13), each subproblem is initially solved without any recourse information. The decisions generated are myopic and maybe infeasible with respect to subsequent stages. Solving the expected value problem first to produce good starting point and possibly preliminary cuts is a commonly used strategy to speed up the solution process. Morton [87] generalized this approach by considering a few representative subproblems from each node to obtain

a good starting point. Valid feasibility and optimality cuts are generated by using a dual sharing formula. Both these strategies are explored in the numerical experiments. The solution process is divided into two phases. In the first phase, either the expected value problem or a smaller version of the original problem is solved. The expected value problem is obtained by replacing the stochastic variables in each stage by their expected values. The resulting problem is a multistage deterministic problem. Alternatively, a small set of representative nodes from the original tree can be used to give a more balanced approximation to the stochastic variables. One way to obtain the set is by first sorting the stochastic right hand sides by some heuristic. From each node that is contained in the small presolve tree, a small number of samples, say two, that are reasonably well spread and are in some sense typical of other siblings, are chosen. In the numerical experiments, if the number of siblings is less than four, then only the one closest to the expected value is chosen. For nodes with more descendents, siblings on either side of the one closest to the expected value are used. This is shown in Figure 6.3 as nodes marked with 'P2'.

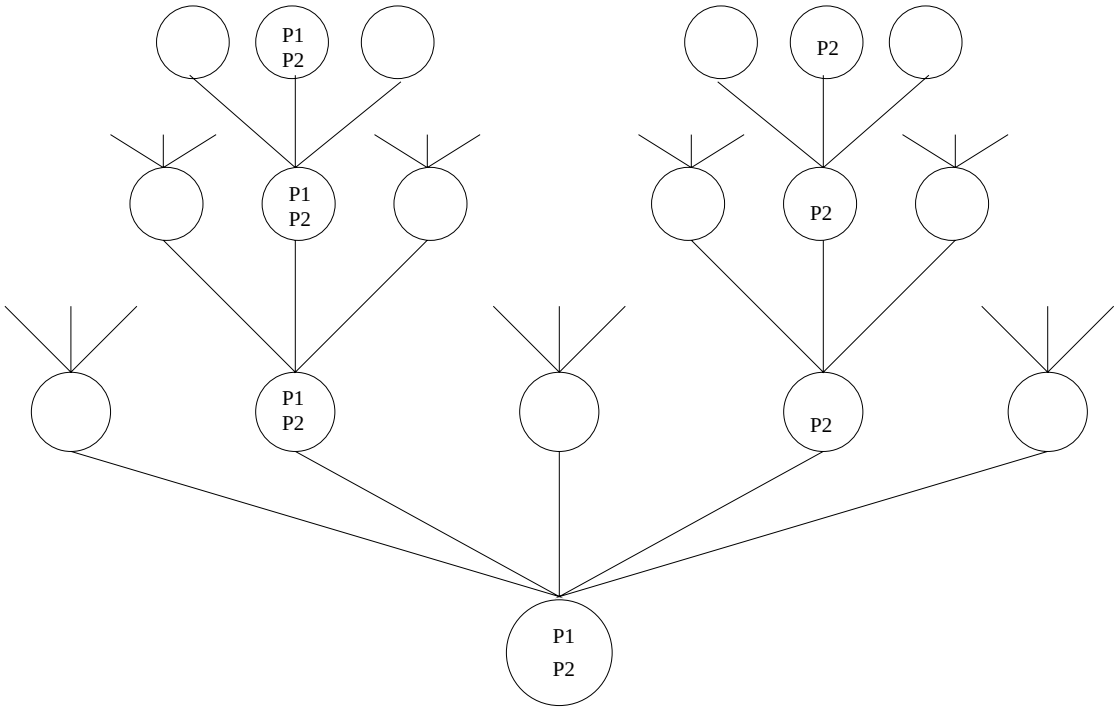

Figure 6.3: Presolve tree

There are two ways the expected value tree and the small representative tree can be solved. Since each subproblem in the experiment is very small and the number of nodes in the presolve tree is bounded by a small multiple of the number of stages, the equivalent QP problem (1.14) is not very large and can be solved efficiently using interior point methods. The solution can be used as starting point to the original problem. This has the advantage of not producing any feasibility cuts which are often generated in the early stages of Algorithm 4.3 when the iterates tend to be more extreme. Since the solution from an interior point solver is in general not as accurate as that from an active set method and does not provide any active set information, while Algorithm 4.3 is an recursive active method and can be sped up by accurate active set estimate, it is advantageous if the interior point solution is first converted to an active set solution before being used as starting point in Algorithm 4.3. Many recent papers are devoted to extending results from LP and general nonlinear problems with good properties to general problems. See [31,71,127] and references therein for more discussion.

Alternatively, the presolve problems can be solved by Algorithm 4.3. This provides approximate recourse functions, accurate starting point and active set estimate. The trust region radii will also be better tuned to the current quadratic piece. If there are more than one sibling at a stage in the presolve tree, only nodes marked with 'P1' in Figure 6.3 are solved in the first few iterations. This avoids solving too many branches while the iterates are still far away from the solution. The algorithm switches to the full presolve tree only when the optimality condition is below a prespecified tolerance.

If a subproblem repeatedly fails to find a solution feasible with respect to its descendents, it and its immediate descendents are aggregated to a large QP similar to (1.14). Either a feasible solution is found or a feasibility cut is created for the ancestor stage using information from the combined problem. This strategy has also been implemented in [11] in a nested Benders decomposition scheme. They solved subtrees with progressively more stages until a feasible solution is found thus avoiding using feasibility cuts altogether.

6.1.4 Optimality Condition Verification

Algorithm 4.3 stops when $\|d\| \leq \epsilon$ for some prespecified tolerance $\epsilon > 0$. This requires solving a convex QP (4.14) to confirm the current x is optimal. It is also possible to check optimality by explicitly calculating the dual variables to the active constraints and subgradients of active pieces. Consider the problem

$$\begin{aligned} \min_{\lambda,\mu} \quad & \tfrac{1}{2}\left\| \sum_{j\in J(x^i)} \lambda_j g_j^i + \bar{A}^\intercal \mu \right\|^2 \\ \text{s.t.} \quad & \sum_{j\in J(x^i)} \lambda_j = 1 \\ & \lambda, \mu_I \geq 0 \end{aligned} \tag{6.1}$$

where $\bar{A} \in \mathbb{R}^{\bar{m}\times n}$ is the set of active constraints, μ is the corresponding vector of multipliers, and the subscript I indicate the inequality constraints. This is a constrained linear least square problem with $\bar{m} + |J|$ variables. Since the number of active constraints and active pieces is expected to be smaller than the number of variables, (6.1) is smaller than (4.14). Its special structure also means that (6.1) is much easier to solve than (4.14) which is a general convex QP. For the majority of problems, the number of iterations for each subproblem is only significant (approximately 5) in the first one or two stage one iterations and is usually 1 or 2 towards the end. This makes using (6.1) more competitive for problems that require few iterations per subproblem.

The special structure of (6.1) also makes it more numerically stable than (4.14). This is especially important if the subproblems are convex but not strictly convex. In this case, B in (4.14) is made positive definite by adding a small multiple of an identity matrix to an element of the generalized Hessian if it is singular, so B is likely to be badly-conditioned. As the subdifferential of each recourse function is given by the dual variables of its descendents problems, it is of vital importance that the dual variables are calculated as accurately as possible. Therefore, in our implementation, (6.1) is used. Algorithm 4.3 needs to be modified by moving Step 5 to before Step 4. The optimality test

$$\text{If} \quad \|d_t^{k_t}\| \leq \epsilon$$

is replaced by

$$\text{If} \quad \left\| \sum_{j \in J(x_t^{k_t})} \lambda_{t,j}^{k_t} g_{t,j}^{k_t} + \bar{W}_t^{k_t} \mu_t^{k_t} \right\| \leq \epsilon$$

where $\lambda_t^{k_t}$ and $\mu_t^{k_t}$ are the solution to (6.1) in the (t, k_t) subproblem.

6.1.5 Degeneracy

As we have seen in Section 3.2.1, the presence of a degenerate quadratic piece may lead to false convergence. However, it should be clear that it is a rare occurrence. To show that a quadratic piece $q_j(x)$ is indeed active at $x \in \mathbb{R}^n$, it is necessary to have n linearly independent directions d such that $q_j(x + d) = f(x + d)$. Without loss of generality, we can use the basis vectors $\{\pm\delta e_i, i = 1, \ldots n\}$ for some small positive δ. Clearly, this is very expensive and impractical as it involves many function evaluations. This test is not carried out in the numerical experiments. To minimize the possibility of degenerate piece causing premature convergence and to reduce storage, not all quadratic pieces are kept. Each time a subproblem is resolved with a new right hand side, only a few (three in the numerical experiments) quadratic pieces with the lowest function values from previous iteration are kept. These are used to give a rough linear lower bound. Since all inactive pieces are used to produce the linear lower bound, this value can be used to check against the optimal value. If the difference is small, then we can be confident that the solution obtained is indeed optimal.

6.2 Numerical Results

Algorithm 4.3 is applied to random data generated as described in Section 6.1.1 for problems with 3, 4 and 5 stages. The root node has 3, 5 or 7 branches and each node in later stages has 3 or 5 branches. The 4-stage tree in Figure 6.3 with 5 branches at the root node and 3 descendents for each stage 2 and 3 node is denoted as a $5 \times 3 \times 3$ tree. Each nodal subproblem has 50 variables, 2 equality and 8 inequality constraints plus lower and upper bound on each variable. The density

of the matrices H_t and W_t are set to 40% and V_t is 20% dense. The eigenvalues of H_t are in the range $[0, 10]$, the singular values of W_t lie in $[-10, 10]$. The tests are performed for the case where all H_t are positive definite and then repeated with each H_t having 10 zero eigenvalues.

The stochastic right hand sides are generated in the order of $\bar{h} - 3v$, $\bar{h} - 2v$, $\bar{h} - v$, $\bar{h}$, $\bar{h} + v$, $\bar{h} + 2v$, $\bar{h} + 3v$. We compare the effect of solving the siblings in this order and also sorting them so that they are solved in increasing distance from the expected value, ie. $\bar{h}, \bar{h} + v, \bar{h} - v, \bar{h} + 2v, \bar{h} - 2v, \bar{h} + 3v, \bar{h} - 3v$. We also compare the performance of Algorithm 4.3 when different presolve strategies are used. The strategies are identified by the following keys:

ES: expected value problem solved by Algorithm 4.3; stochastic RHS sorted;

EL: expected value problem solved by an interior point method; RHS sorted;

PS: small presolve tree solved by Algorithm 4.3; RHS sorted;

PL: small presolve tree solved by an interior point method; RHS sorted;

NS: no presolve phase; RHS sorted;

NN: no presolve phase; RHS not sorted;

The large scale equivalent QP (1.14) is also generated for comparison. It is solved using the interior point solver SPSOLQP of Ye [130] and the corresponding active set solutions are then calculated whenever possible. Table 6.1 gives the size and characteristics of the equivalent QP.

6.2.1 Trust Region Radius Updating

The two trust region radius updating schemes 6.2 and 6.3 are tested with different parameters to establish appropriate values for the numerical experiments. The test was performed on a 4 stage $5 \times 3 \times 3$ tree. The values of the parameters are given in Table 6.2. The results are given in Figure 6.4 and 6.5 for problems with strictly convex and convex subproblems respectively. The graphs plot the geometric mean of the cputime used to solve a set of 20 random problems. The geometric mean of

Table 6.1: Size of equivalent deterministic QP (1.14)

# Stages	3			
Tree	3×3	5×3	7×3	
# nodes	(1 3 9) 13	(1 5 15) 21	(1 7 21) 29	
large QP size	130×650	210×1050	290×1450	
# entries in A	3800	6200	8600	
Density of A	.045	.028	.020	
# Stages	4			
Tree	$3 \times 3 \times 3$	$5 \times 3 \times 3$	$7 \times 3 \times 3$	$7 \times 5 \times 3$
# nodes	(1 3 9 27) 40	(1 5 15 45) 66	(1 7 21 63) 92	(1 7 35 105) 148
large QP size	400×2000	660×3300	920×4600	1480×7400
# entries in A	11900	19700	27500	44300
Density of A	.015	.0090	6.5e-3	4.0e-3
# Stages	5			
Tree	$3 \times 3 \times 3 \times 3$	$5 \times 3 \times 3 \times 3$	$7 \times 3 \times 3 \times 3$	$7 \times 5 \times 3 \times 3$
# nodes	(1 3 9 27 81) 121	(1 5 15 45 135) 201	(1 7 21 63 189) 281	(1 7 35 105 315) 463
large QP size	1210×6050	2010×10050	2810×14050	4630×23150
# entries in A	36200	60200	84200	138800
Density of A	4.9e-3	3.0e-3	2.1e-3	1.3e-3

	Algorithm 6.2		Algorithm 6.3			
parameters	P1	P2	M1	M2	T1	T2
c1	0.25	0.25	0.5	0.5	0.7	0.7
c2	0.75	0.75	0.8	0.8	0.95	0.95
c3	Inf	Inf	1.2	1.2	1.05	1.05
c4	Inf	Inf	1.5	1.5	1.3	1.3
c5	0.8	0	0.8	0	0.8	0

Table 6.2: Trust region method parameters

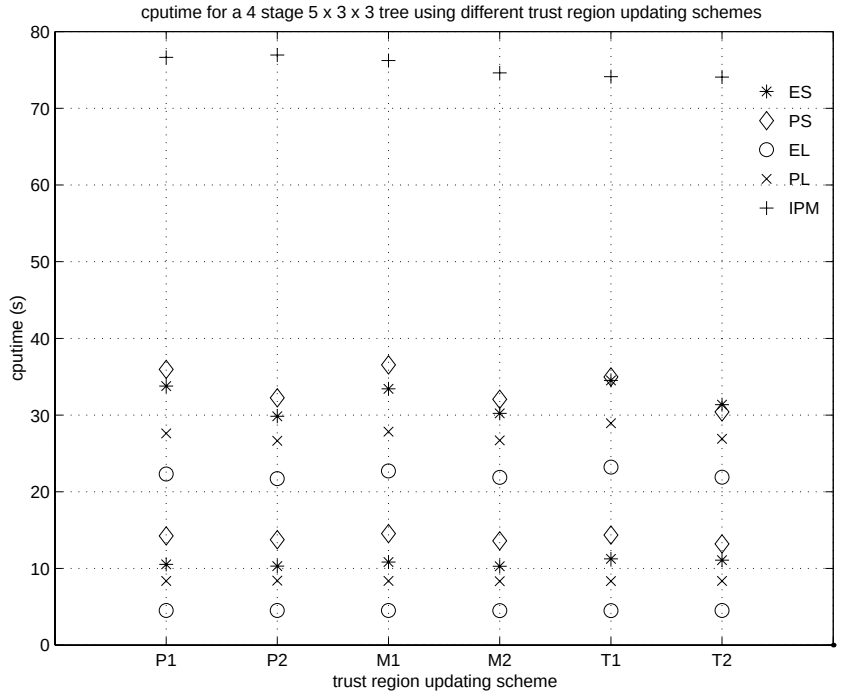

Figure 6.4: Cputime vs trust region schemes: Strictly convex random subproblems

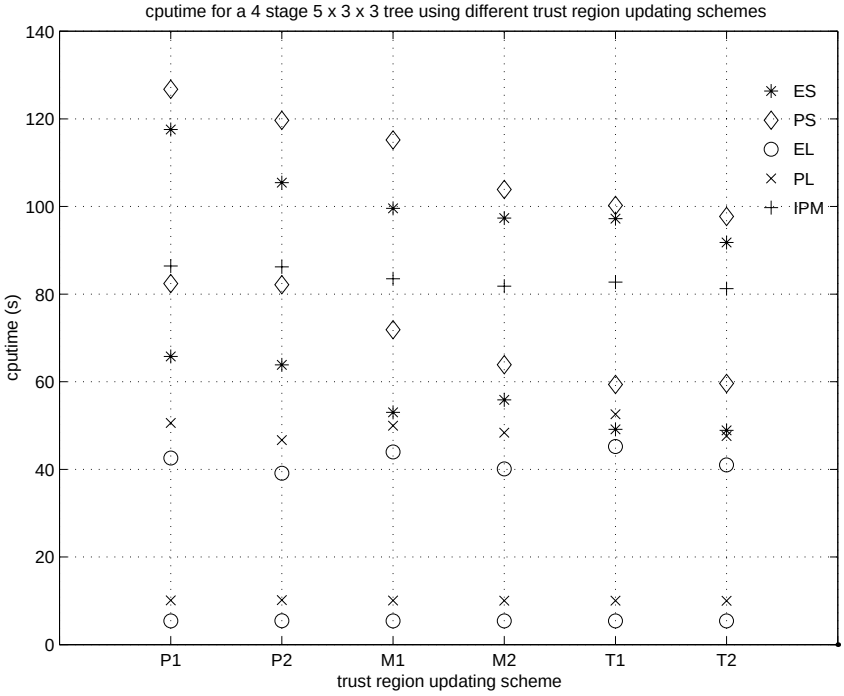

Figure 6.5: Cputime vs trust region schemes: Convex random subproblems

a random vector $x \in \mathbb{R}^n$ is given by

$$\left(\prod_{i=1}^{n} x_i\right)^{(1/n)}.$$

The geometric mean is a more robust measure of the average as it is less sensitive to outliers in the data. For strictly convex problems, there is no significant difference in performance when parameters $c1$ to $c4$ are changed. Relaxing $c5$ from 0.8 to 0 has decreased cputime slightly, corresponding to increasing the trust region radii of all variables instead of only those which are almost active in the search direction problem. When the subproblems are only convex, a similar pattern holds if the presolve phase was performed by an interior point method. However, when the presolve problem was solved by Algorithm 4.3, the difference in the different trust region updating schemes becomes much more prominent. Scheme 'T' is clearly far more efficient than 'M' which is in turn faster than 'P'. This shows that a trust region updating scheme that is more sensitive to poor function approximation is much more efficient. The difference in behaviour between the strictly convex and not strictly convex problems is probably because when the objective is strictly convex, subproblem solutions are more likely to be determined by the curvature information and are less dependent on the trust region radii. When the presolve tree is solved by Algorithm 4.3, the iterates need to move between different quadratic pieces to reach the solution. A trust region updating scheme that is sensitive to poor function approximation can reduce the chance of the iterates going to regions that are poorly approximated by the current quadratic function which generally lead to more function evaluations that are likely to be expensive. In the second phase, the iterate is already in a neighbourhood not too far from the solution and therefore it requires fewer changes in quadratic pieces and the trust region radii plays a less important role than in the presolve phase.

Based on these results, we decided to use the updating scheme 'T' in the following numerical experiments. Although setting $c5$ to be 0 performs better overall than using $c5 = 0.8$, for the most difficult problems, using $c5 = 0.8$ proves to be far more efficient. Therefore, as a compromise, parameter $c5$ is set to be 0.5.

6.2.2 A Small Numerical Example

We begin our numerical experiments by running Algorithm 4.3 on a small example. It is a two stage stochastic quadratic program taken from Louveaux [72]. The problem is

$$\begin{aligned} \min_{x\in\mathbb{R}^2} \quad & 2x_1 + 3x_2 + Q(x) \\ \text{s.t.} \quad & 3x_1 + 2x_2 \leq 15 \\ & x_1 + 2x_2 \leq 8 \\ & x_1, x_2 \geq 0 \end{aligned}$$

where $Q(x) = \mathrm{E}_\xi[\mathcal{Q}(x,\xi)]$ and

$$\begin{aligned} \mathcal{Q}(x,\xi) = \min_{y\in\mathbb{R}^2} \quad & \tfrac{1}{2}y_1^2 + y_1y_2 + \tfrac{1}{2}y_2^2 - 6.5y_1 - 7y_2 \\ \text{s.t.} \quad & y_1 \leq x_1 \\ & y_2 \leq x_2 \\ & y_1 \leq h_1(\xi) \\ & y_2 \leq h_2(\xi) \\ & y_1, y_2 \geq 0. \end{aligned}$$

The stochastic variables h_1 and h_2 are independent. The stochastic variable h_1 can take on the values $\{2,4,6\}$ each with probability $1/3$ and h_2 can take on the values $\{1,3,5\}$ also with equal probability.

The problem was solved without preprocessing (strategy NN) since it is so small. Algorithm 4.3 took 17 iterations in the first stage and 5.32 cpu second to reach optimality. The first stage solution obtained is $(2.49964, 1.00000)$ with objective value -8.58333 with 5 quadratic pieces active at the solution. The relative error in the optimal value is $5.8e-08$. The number of iterations taken is quite large, because the first stage objective is linear and all recourse Hessian are zero matrices, so convergence is solely enforced by the subgradient cutting planes. This example shows that Algorithm 4.3 can handle problems with both linear and quadratic objectives.

6.2.3 Random Data with Strictly Convex Subproblems

The performance of Algorithm 4.3 is first tested on problem (1.12)–(1.13) with strictly convex subproblems. Tables 6.3 to 6.5 summarize the results of the numerical experiments. We report the average number of iterations for the root node problem, cputime, the total number of times any one of the nodes in each stage are solved (for example, if the five stage 2 nodes are solved 5, 3, 4, 2 and 5 times respectively, then the total number we report is 19) and the number of feasibility cuts created for each node. Note that as control is returned to the parent node as soon as an infeasible node is detected, the total number of times nodes are solved may not be a multiple of the number of nodes. However, nodes in the same stage always have the same number of cuts because cuts are shared within a stage. The number of iterations in the second column of DL and PL include only the iteration used in Algorithm 4.3 but not those used in the interior point solver. The cputime tabulated are the minimum, geometric mean and the maximum over 20 randomly generated problems. The average cputime required per node is also calculated. Since the numerical experiments were performed with MATLAB, the cputime is not comparable to a Fortran or C implementation. As an indication, the first stage problem of PLTEXP (see Section 6.2.5) is a QP of size 62×126 with no equality constraint. The sparsity pattern of the constraint matrix and the Hessian is given in Figure 6.6. With no active set estimate and using the zero vector, which is feasible, as a starting point, the active set method QP routine used in our numerical experiments took 5.9 cpu second. For the same problem, SPSOLQP [130], which is an interior point method written also in Matlab, took 1.2 cpu second while LOQO [122], which is an interior point code written in C, took 0.025 cpu second.

To give an indication of the accuracy of Algorithm 4.3, the first stage optimal solution x_1 and function value f_1 are compared to $\bar{x}_1$ and $\bar{f}_1$ obtained in the large scale problem. The relative errors reported are $\epsilon_x = \|x_1 - \bar{x}_1\|/\|\bar{x}_1\|$ and $\epsilon_f = |f_1 - \bar{f}_1|/|\bar{f}_1|$. If a presolve phase is also used, then ϵ_f under the presolve column gives the difference between the optimal value of the presolve problem and the full problem. That is $\epsilon_f = |f_{\text{presolve}} - f_{\text{full}}|/|f_{\text{full}}|$. The optimality norm of the root

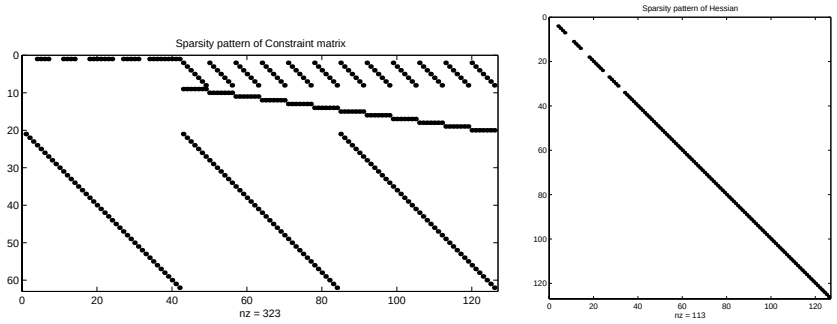

Figure 6.6: Sparsity pattern of problem PLTEXP

node problem is $\epsilon_L = \|g_1 - W_1^T \lambda_1\|$. The median and maximum are tabulated. The large scale deterministic equivalent (1.14) is solved by the interior point solver SPSOLQP for comparison. The interior point solution is converted to an active set solution whenever possible. However, as the problem size increases, this becomes increasingly difficult. To give an indication of the accuracy of the interior point solution, the maximum infeasibility is calculated. It is defined as

$$\delta = \max \begin{pmatrix} Ax - b \\ b_1 - A_1 x \\ 0 \end{pmatrix}. \tag{6.2}$$

The equality constraints are $A_1 x = b_1$. The median and maximum of δ are tabulated. The cputime shown in the IPM column includes only time spent in the interior point solver, not in the conversion to an active set solution.

For problems with only three branches at the root node, the expected value problem and the small presolve tree are identical. Therefore only the expected value problem is used in the presolve phase, strategies PS and PL are not used.

Figures 6.7 to 6.9 plot the geometric mean of the cputime versus the size of the deterministic equivalent QP (1.14) for problems with 3, 4 and 5 stages respectively. Time spent in the presolve phase is shown as dotted lines, while the total time is shown as solid lines. Figure 6.10 shows the geometric mean of the total cputime for all problem sizes.

The numerical results show that having a presolve stage significantly improves the performance of Algorithm 4.3, and the saving becomes much more prominent as the number of branches and number of stages increase. Due to the extremely

small size of the expected value problem, an interior point method can solve it much more quickly than Algorithm 4.3 and strategy DL is clearly the most effective solution scheme. Using a small scenario tree in PS and PL yielded more information about the original problem, this is confirmed by the smaller differences between the function values of the presolve problems and the full problems. However, strategies PS and PL did not lead to any reduction in total cputime.

For strategies with a presolve phase, all feasibility cuts are generated in the first phase. The reason is that in the presolve phase, iterates need to move between many pieces before reaching the solution region. The larger change in iterates make them much more likely to produce infeasible descendent problems. Solving the deterministic equivalent of the presolve problem has eliminated the need for feasibility cuts completely in these experiments.

As the number of stages increases, the cputime required per node increases, because subtrees need to be solved many more times as the level of recursion increases. On the other hand, cputime per node decreases as the number of branches increases due to economy of scale. The first time the first subproblem in each stage is solved is the most expensive while other subproblems and subsequent iterations can take advantage of good starting points and active set information.

The expected value problems with the same number of stages and different number of branches are identical due to the way the stochastic right hand sides are generated. However, Algorithm 4.3 performs differently on them. This is because when feasibility cuts are generated, all stochastic right hand sides are included in the calculation which leads to some differences in the cuts returned.

Sorting stochastic right hand sides in increasing distance from the expected value has led to an increase in cputime. This is probably because right hand sides far away from the expected value are more likely to lead to infeasible subproblems. By solving them first, the necessary feasibility cuts are identified earlier. This is confirmed by the fact that strategy NS on average creates more feasibility cuts than NN.

The accuracy of Algorithm 4.3 is very high. If the interior point solution can be converted to an active set solution, then the relative errors in function values

and solution points of all subproblems are very close to the machine error (2e-16). For the other problems, the relative error is of the same order of magnitude as the maximum infeasibility of the interior point solution.

As all variables in the experimental data have lower and upper bounds, the linearly independence constraint qualification may not be satisfied for all subproblems. It is possible for active constraint gradients to be linearly dependent leading to nonsmooth objective in ancestor problem.

Due to the large size of the deterministic equivalent (1.14). The interior point solver was unable to solve the 5 stages $7 \times 5 \times 3 \times 3$ problem.

Tree Structure	Method	E S		E L		P S		P L		N S	N N	IPM
		Expected	Total (incl. expected)	IPM	Total (incl. IPM)	Presolve	Total (incl. presolve)	IPM	Total (incl. IPM)			
3×3	Iteration	6.2	9.2	29.9	3.0	0.0	0.0	0.0	0.0	6.5	5.9	31.1
	cpu (s): min/geo. mean/max	0.9/ 4.6/ 22.5	2.3/ 7.9/ 26.2	2.6/ 3.0/ 3.5	4.2/ 5.9/ 8.2	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	2.4/ 13.3/ 70.4	3.0/ 13.7/ 47.8	11.5/ 13.2/ 16.6
	mean cpu/node	1.95	0.69	1.00	0.46	0.00	0.00	0.00	0.00	1.32	1.26	1.02
	number of Node	(1 1 1) 3	(1 3 9) 13	(1 1 1) 3	(1 3 9) 13	(1 1 1) 3	(1 3 9) 13	(1 1 1) 3	(1 3 9) 13	(1 3 9) 13	(1 3 9) 13	
	Node visited	1 7 24	2 16 71		1 9 43	0 0 0	0 0 0		0 0 0	1 19 160	1 18 150	
	Cut	0.2 1.4	0.2 1.4		0.0 0.0	0.0 0.0	0.0 0.0		0.0 0.0	0.3 1.5	0.3 1.6	
	ϵ_x med		2.15e-15		2.20e-15		0.00e+00		0.00e+00	2.66e-15	2.20e-15	
	ϵ_x max		5.62e-15		6.05e-15		0.00e+00		0.00e+00	1.28e-14	8.24e-15	
	ϵ_f med	2.5e-02	8.25e-16	2.5e-02	6.89e-16	0.0e+00	0.00e+00		0.00e+00	9.23e-16	7.83e-16	median(δ)
	ϵ_f max	2.9e-01	3.86e-15	2.9e-01	4.27e-15	0.0e+00	0.00e+00		0.00e+00	4.00e-15	4.41e-15	2.8e-14
	ϵ_L med		5.30e-14		5.31e-14		0.00e+00		0.00e+00	4.23e-14	6.05e-14	max(δ)
	ϵ_L max		4.19e-13		3.01e-13		0.00e+00		0.00e+00	2.21e-13	2.15e-13	7.2e-14
5×3	Iteration	6.5	10.1	29.9	3.5	11.2	14.0	30.3	2.8	6.3	6.2	31.2
	cpu (s): min/geo. mean/max	0.9/ 4.7/ 27.4	3.4/ 10.0/ 32.8	2.6/ 3.0/ 3.5	5.0/ 7.8/ 13.3	1.8/ 6.5/ 31.8	5.0/ 11.3/ 37.4	4.6/ 5.3/ 6.6	7.9/ 10.1/ 14.2	4.2/ 19.3/ 89.2	4.8/ 19.6/ 54.6	19.2/ 21.0/ 24.6
	mean cpu/node	2.04	0.54	1.00	0.38	1.63	0.60	1.06	0.49	1.15	1.11	1.00
	number of Node	(1 1 1) 3	(1 5 15) 21	(1 1 1) 3	(1 5 15) 21	(1 2 2) 5	(1 5 15) 21	(1 2 2) 5	(1 5 15) 21	(1 5 15) 21	(1 5 15) 21	
	Node visited	1 7 23	2 25 107		1 18 77	2 18 38	3 32 109		1 14 72	1 32 241	1 31 233	
	Cut	0.2 1.3	0.2 1.3		0.0 0.0	0.1 1.1	0.1 1.1		0.0 0.0	0.3 1.4	0.3 1.4	
	ϵ_x med		2.29e-15		2.60e-15		2.50e-15		2.52e-15	2.63e-15	2.65e-15	
	ϵ_x max		7.79e-07		7.79e-07		7.79e-07		7.79e-07	7.79e-07	7.79e-07	
	ϵ_f med	3.0e-02	9.59e-16	3.0e-02	8.34e-16	5.9e-03	9.01e-16	5.9e-03	9.10e-16	9.31e-16	7.86e-16	median(δ)
	ϵ_f max	3.3e-01	3.48e-08	3.3e-01	3.48e-08	2.2e-01	3.48e-08	2.2e-01	3.48e-08	3.48e-08	3.48e-08	3.0e-14
	ϵ_L med		5.07e-14		7.11e-14		3.96e-14		4.60e-14	4.49e-14	4.46e-14	max(δ)
	ϵ_L max		2.14e-13		2.03e-13		2.88e-13		1.70e-13	3.05e-13	1.21e-13	2.9e-06
7×3	Iteration	6.5	10.3	29.9	3.7	10.8	13.9	30.3	3.0	6.5	6.6	31.8
	cpu (s): min/geo. mean/max	0.9/ 4.7/ 22.9	4.2/ 12.2/ 30.8	2.6/ 3.0/ 3.5	5.9/ 9.8/ 17.1	1.8/ 6.4/ 27.2	5.8/ 12.8/ 34.8	4.6/ 5.3/ 6.6	8.5/ 11.9/ 16.4	6.0/ 26.4/ 188.4	6.5/ 26.1/ 109.8	26.0/ 29.7/ 34.2
	mean cpu/node	1.99	0.46	1.00	0.35	1.57	0.49	1.06	0.42	1.26	1.10	1.03
	number of Node	(1 1 1) 3	(1 7 21) 29	(1 1 1) 3	(1 7 21) 29	(1 2 2) 5	(1 7 21) 29	(1 2 2) 5	(1 7 21) 29	(1 7 21) 29	(1 7 21) 29	
	Node visited	1 7 24	2 34 148		1 26 115	2 18 36	3 39 138		1 21 103	1 46 338	1 46 324	
	Cut	0.2 1.2	0.2 1.2		0.0 0.0	0.1 1.2	0.1 1.2		0.0 0.0	0.3 1.5	0.3 1.4	
	ϵ_x med		4.03e-15		3.22e-15		3.45e-15		2.93e-15	3.74e-15	3.49e-15	
	ϵ_x max		8.22e-06		8.22e-06		8.22e-06		8.22e-06	8.22e-06	8.22e-06	
	ϵ_f med	3.1e-01	3.80e-15	3.1e-01	3.86e-15	6.6e-03	3.60e-15	6.6e-03	3.70e-15	3.60e-15	3.64e-15	median(δ)
	ϵ_f max	3.3e-01	1.64e-07	3.3e-01	1.64e-07	2.3e-01	1.64e-07	2.3e-01	1.64e-07	1.64e-07	1.64e-07	4.3e-14
	ϵ_L med		4.59e-14		5.65e-14		4.31e-14		4.80e-14	4.70e-14	4.55e-14	max(δ)
	ϵ_L max		3.60e-10		3.60e-10		3.60e-10		3.60e-10	3.60e-10	3.60e-10	9.2e-06

Table 6.3: Numerical results for 3 stage tree with strictly convex subproblems

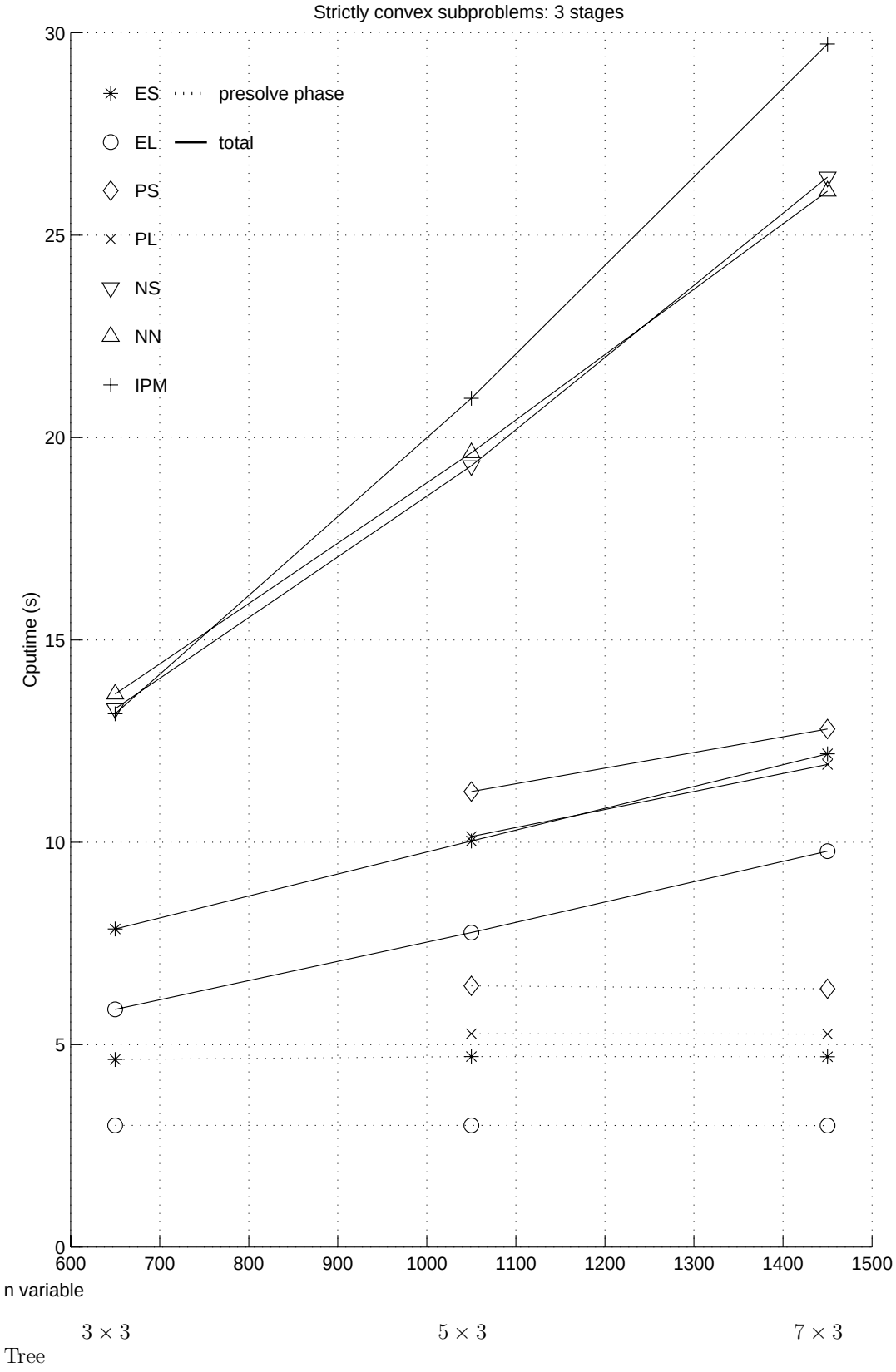

Figure 6.7: Cputime vs number of variables: 3 stage tree, strictly convex subproblems

Tree Structure	Method	E S		E L		P S		P L		N S	N N	IPM
		Expected	Total (incl. expected)	IPM	Total (incl. IPM)	Presolve	Total (incl. presolve)	IPM	Total (incl. IPM)			
$3 \times 3 \times 3$	Iteration	6.2	9.3	30.2	3.1	0.0	0.0	0.0	0.0	6.6	6.2	32.1
	cpu (s): min/geo. mean/max	3.0/ 11.4/ 55.4	12.5/ 25.8/ 65.4	3.9/ 4.5/ 5.2	11.4/ 15.5/ 27.8	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	19.3/ 68.5/ 286.0	24.2/ 67.6/ 211.4	39.8/ 45.5/ 66.1
	mean cpu/node	3.62	0.69	1.14	0.40	0.00	0.00	0.00	0.00	2.14	1.97	1.14
	number of Node	(1 1 1 1) 4	(1 3 9 27) 40	(1 1 1 1) 4	(1 3 9 27) 40	(1 1 1 1) 4	(1 3 9 27) 40	(1 1 1 1) 4	(1 3 9 27) 40	(1 3 9 27) 40	(1 3 9 27) 40	
	Node visited	1 7 24 52	2 16 71 234		1 9 43 157	0 0 0 0	0 0 0 0		0 0 0 0	1 20 168 904	1 19 155 825	
	Cut	0.2 1.6 1.9	0.2 1.6 1.9		0.0 0.0 0.0	0.0 0.0 0.0	0.0 0.0 0.0		0.0 0.0 0.0	0.3 1.8 2.0	0.3 1.5 1.9	
	ϵ_x med		3.78e-07		3.78e-07		0.00e+00		0.00e+00	3.78e-07	3.78e-07	
	ϵ_x max		1.49e-05		1.49e-05		0.00e+00		0.00e+00	1.49e-05	1.49e-05	
	ϵ_f med	1.6e-02	5.28e-09	2.1e-01	5.28e-09	0.0e+00	0.00e+00	0.0e+00	0.00e+00	5.28e-09	5.28e-09	median(δ)
	ϵ_f max	1.6e-02	3.51e-07	2.1e-01	3.51e-07	0.0e+00	0.00e+00	0.0e+00	0.00e+00	3.51e-07	3.51e-07	5.2e-06
	ϵ_L med		6.03e-14		5.14e-14		0.00e+00		0.00e+00	7.48e-14	1.02e-13	max(δ)
	ϵ_L max		8.83e-13		8.53e-13		0.00e+00		0.00e+00	5.56e-13	4.68e-13	3.4e-05
$5 \times 3 \times 3$	Iteration	6.2	9.8	30.2	3.6	9.5	12.3	30.6	2.8	6.3	6.3	32.2
	cpu (s): min/geo. mean/max	3.1/ 11.2/ 35.4	21.1/ 33.8/ 53.0	4.0/ 4.5/ 5.2	14.3/ 23.2/ 36.6	4.8/ 13.9/ 44.1	18.4/ 34.7/ 63.9	7.2/ 8.4/ 9.6	17.5/ 28.8/ 51.0	31.2/ 97.6/ 399.1	35.9/ 97.7/ 317.3	67.0/ 77.1/ 114.0
	mean cpu/node	3.38	0.53	1.14	0.36	2.35	0.55	1.20	0.45	1.86	1.74	1.18
	number of Node	(1 1 1 1) 4	(1 5 15 45) 66	(1 1 1 1) 4	(1 5 15 45) 66	(1 2 2 2) 7	(1 5 15 45) 66	(1 2 2 2) 7	(1 5 15 45) 66	(1 5 15 45) 66	(1 5 15 45) 66	
	Node visited	1 7 23 50	2 25 105 348		1 18 78 272	2 15 35 64	3 30 106 331		1 14 70 279	1 32 258 1336	1 32 245 1269	
	Cut	0.2 1.4 2.0	0.2 1.4 2.0		0.0 0.0 0.0	0.1 1.4 1.8	0.1 1.4 1.8		0.0 0.0 0.0	0.3 1.6 2.1	0.3 1.4 1.9	
	ϵ_x med		4.78e-06		4.78e-06		4.78e-06		4.78e-06	4.78e-06	4.78e-06	
	ϵ_x max		2.50e-05		2.50e-05		2.50e-05		2.50e-05	2.50e-05	2.50e-05	
	ϵ_f med	2.0e-02	3.66e-08	2.0e-02	3.66e-08	7.8e-03	3.66e-08	7.8e-03	3.66e-08	3.66e-08	3.66e-08	median(δ)
	ϵ_f max	2.1e-01	1.26e-06	2.1e-01	1.26e-06	2.1e-01	1.26e-06	2.1e-01	1.26e-06	1.26e-06	1.26e-06	6.0e-06
	ϵ_L med		7.07e-14		8.61e-14		6.26e-14		6.23e-14	4.50e-14	7.29e-14	max(δ)
	ϵ_L max		4.00e-13		3.54e-13		3.46e-13		6.65e-13	4.89e-13	3.34e-13	3.6e-05
$7 \times 3 \times 3$	Iteration	6.2	10.1	30.2	3.8	11.4	14.6	30.6	2.9	6.5	6.5	32.5
	cpu (s): min/geo. mean/max	3.1/ 11.6/ 38.1	26.2/ 43.2/ 68.8	3.9/ 4.5/ 5.2	17.1/ 31.2/ 54.8	4.8/ 14.2/ 46.6	21.9/ 44.3/ 168.6	7.2/ 8.4/ 9.6	22.2/ 36.1/ 63.1	38.4/ 132.2/ 750.9	44.5/ 127.8/ 388.6	96.4/ 108.8/ 162.0
	mean cpu/node	3.57	0.49	1.14	0.35	2.43	0.53	1.20	0.41	1.91	1.65	1.19
	number of Node	(1 1 1 1) 4	(1 7 21 63) 92	(1 1 1 1) 4	(1 7 21 63) 92	(1 2 2 2) 4	(1 7 21 63) 92	(1 2 2 2) 4	(1 7 21 63) 92	(1 7 21 63) 92	(1 7 21 63) 92	
	Node visited	1 7 24 53	2 33 145 482		1 27 115 398	2 19 38 67	3 42 155 505		1 20 98 384	1 47 361 1881	1 46 346 1759	
	Cut	0.2 1.4 2.0	0.2 1.4 2.0		0.0 0.0 0.0	0.2 1.4 1.8	0.2 1.4 1.8		0.0 0.0 0.0	0.3 1.8 2.1	0.3 1.5 1.5	
	ϵ_x med		4.87e-06		4.87e-06		4.87e-06		4.87e-06	4.87e-06	4.87e-06	
	ϵ_x max		3.13e-05		3.13e-05		3.13e-05		3.13e-05	3.13e-05	3.13e-05	
	ϵ_f med	2.1e-02	4.52e-08	2.1e-02	4.52e-08	8.3e-03	4.52e-08	8.3e-03	4.52e-08	4.52e-08	4.52e-08	median(δ)
	ϵ_f max	2.1e-01	4.43e-07	2.1e-01	4.43e-07	2.1e-01	4.43e-07	2.1e-01	4.43e-07	4.43e-07	4.43e-07	9.6e-06
	ϵ_L med		6.11e-14		6.20e-14		6.93e-14		6.14e-14	5.59e-14	7.00e-14	max(δ)
	ϵ_L max		7.62e-13		6.81e-13		2.67e-09		2.67e-09	4.39e-12	4.40e-12	4.3e-05
$7 \times 5 \times 3$	Iteration	6.2	10.2	30.2	4.0	9.2	12.5	30.9	3.0	6.7	6.7	33.0
	cpu (s): min/geo. mean/max	3.1/ 11.6/ 38.0	38.9/ 63.0/ 97.0	4.0/ 4.5/ 5.2	22.9/ 48.4/ 81.7	7.4/ 22.1/ 99.6	36.9/ 68.0/ 167.7	11.7/ 13.4/ 15.4	32.1/ 58.6/ 95.0	71.9/ 201.0/ 666.3	76.1/ 209.4/ 687.4	160.5/ 177.2/ 310.4
	mean cpu/node	3.64	0.44	1.14	0.34	2.46	0.49	1.22	0.41	1.72	1.71	1.21
	number of Node	(1 1 1 1) 4	(1 7 35 105) 148	(1 1 1 1) 4	(1 7 35 105) 148	(1 2 4 4) 11	(1 7 35 105) 148	(1 2 4 4) 11	(1 7 35 105) 148	(1 7 35 105) 148	(1 7 35 105) 148	
	Node visited	1 7 25 55	2 35 251 854		1 28 213 740	2 15 73 109	3 39 266 804		1 21 180 683	1 48 637 3086	1 48 628 3057	
	Cut	0.2 1.2 1.9	0.2 1.2 1.9		0.0 0.0 0.0	0.2 1.4 1.8	0.2 1.4 1.8		0.0 0.0 0.0	0.3 1.6 1.7	0.3 1.6 1.4	
	ϵ_x med		2.40e-06		2.40e-06		2.40e-06		2.40e-06	2.40e-06	2.40e-06	
	ϵ_x max		1.16e-05		1.16e-05		1.16e-05		1.16e-05	1.16e-05	1.16e-05	
	ϵ_f med	2.2e-02	9.45e-08	2.2e-02	9.45e-08	6.9e-03	9.45e-08	6.9e-03	9.45e-08	9.45e-08	9.45e-08	median(δ)
	ϵ_f max	2.8e-01	1.92e-06	2.8e-01	1.92e-06	1.3e-01	1.92e-06	1.3e-01	1.92e-06	1.92e-06	1.92e-06	3.2e-05
	ϵ_L med		4.73e-14		6.48e-14		5.73e-14		6.77e-14	4.82e-14	4.82e-14	max(δ)
	ϵ_L max		5.09e-10		5.09e-10		3.16e-13		6.59e-13	5.66e-10	5.66e-10	1.5e-04

Table 6.4: Numerical results for 3 stage tree with strictly convex subproblems

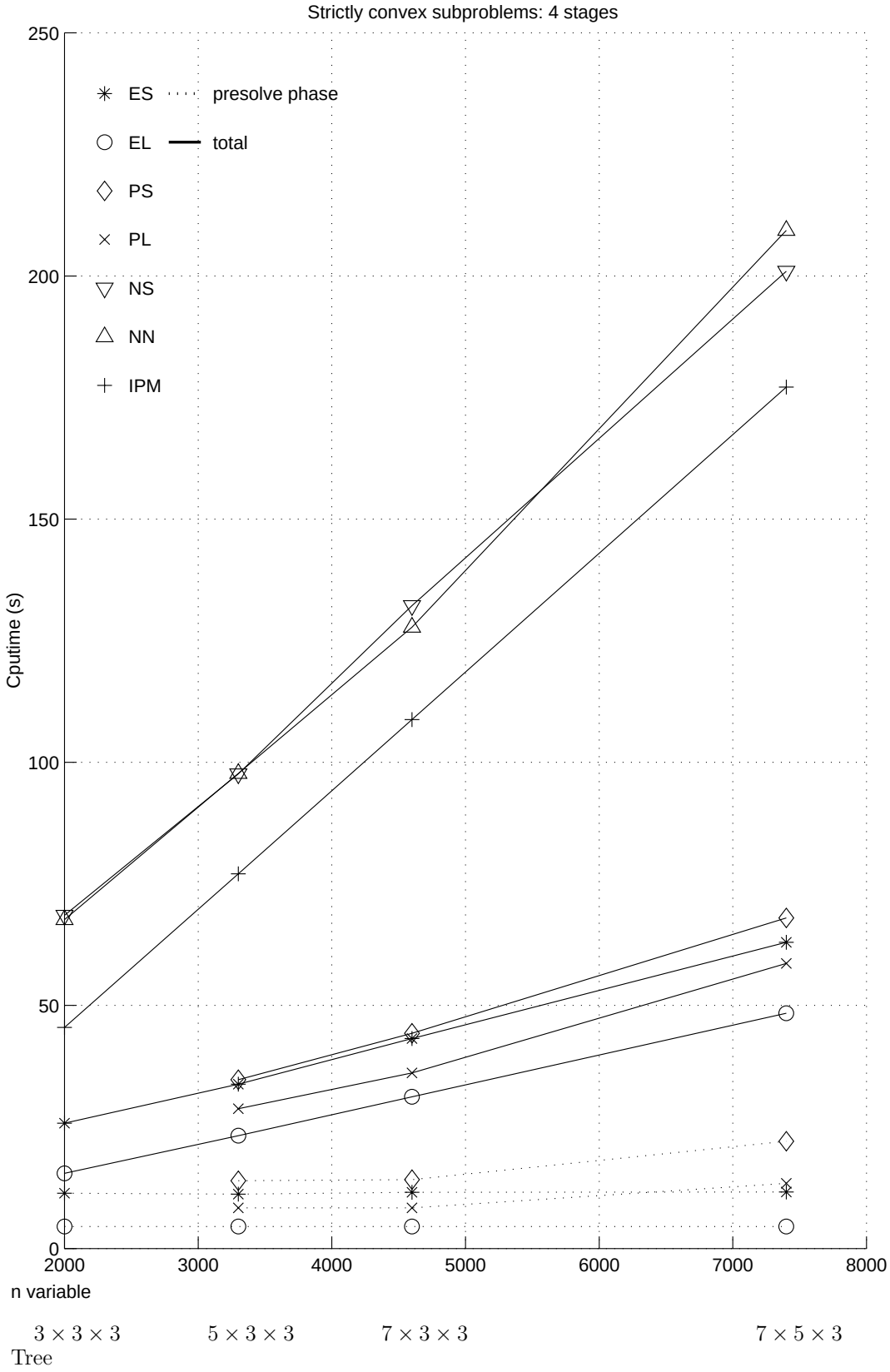

Figure 6.8: Cputime vs number of variables: 4 stage tree, strictly convex subproblems

Tree Structure	Method	E S		E L		P S		P L		IPM
		Expected	Total (incl. expected)	IPM	Total (incl. IPM)	Presolve	Total (incl. presolve)	IPM	Total (incl. IPM)	
$3\times3\times3\times3\times3$	Iteration	6.1	9.2	30.4	3.0	0.0	0.0	0.0	0.0	33.0
	cpu (s): min/geo. mean/max	5.9/ 23.6/ 80.2	48.2/ 75.4/ 146.7	5.4/ 6.1/ 7.1	29.1/ 44.5/ 104.4	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	131.5/ 149.0/ 195.3
	mean cpu/node	5.75	0.65	1.23	0.39	0.00	0.00	0.00	0.00	1.24
	number of Node	(1 1 1 1 1) 5	(1 3 9 27 81) 121	(1 1 1 1 1) 5	(1 3 9 27 81) 121	(1 1 1 1 1) 5	(1 3 9 27 81) 121	(1 1 1 1 1) 5	(1 3 9 27 81) 121	
	Node visited	1 7 23 55 98	2 16 68 234 711		1 9 42 160 527	0 0 0 0 0	0 0 0 0 0		0 0 0 0 0	
	Cut	0.2 1.6 2.1 2.4	0.2 1.6 2.1 2.4		0.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0		0.0 0.0 0.0 0.0	
	ϵ_x med		4.03e-06		4.03e-06		0.00e+00		0.00e+00	
	ϵ_x max		3.29e-05		3.29e-05		0.00e+00		0.00e+00	
	ϵ_f med	1.5e-02	4.29e-08	1.5e-02	4.29e-08	0.0e+00	0.00e+00	0.0e+00	0.00e+00	median(δ)
	ϵ_f max	1.5e-01	8.42e-07	1.5e-01	8.42e-07	0.0e+00	0.00e+00	0.0e+00	0.00e+00	2.5e-05
	ϵ_L med		1.16e-13		8.57e-14		0.00e+00		0.00e+00	max(δ)
	ϵ_L max		4.78e-13		6.72e-13		0.00e+00		0.00e+00	1.1e-04
$5\times3\times3\times3\times3$	Iteration	6.2	10.1	30.4	3.8	9.4	12.5	30.9	3.0	33.5
	cpu (s): min/geo. mean/max	5.9/ 23.5/ 68.4	64.8/ 105.9/ 207.9	5.3/ 6.1/ 7.1	41.9/ 73.4/ 166.1	9.8/ 27.3/ 117.9	64.4/ 103.8/ 185.5	10.2/ 11.9/ 13.9	43.9/ 90.3/ 330.7	230.8/ 260.0/ 354.9
	mean cpu/node	5.59	0.55	1.23	0.39	3.71	0.53	1.32	0.51	1.30
	number of Node	(1 1 1 1 1) 5	(1 5 15 45 135) 201	(1 1 1 1 1) 5	(1 5 15 45 135) 201	(1 2 2 2 2) 9	(1 5 15 45 135) 201	(1 2 2 2 2) 9	(1 5 15 45 135) 201	
	Node visited	1 7 23 53 96	2 26 109 374 1153		1 19 83 300 961	2 15 35 68 117	3 31 110 349 1023		1 15 73 300 1062	
	Cut	0.2 1.4 2.2 2.5	0.2 1.4 2.2 2.5		0.0 0.0 0.0 0.0	0.1 1.4 2.1 2.4	0.1 1.4 2.1 2.4		0.0 0.0 0.0 0.0	
	ϵ_x med		3.47e-06		3.47e-06		3.47e-06		3.47e-06	
	ϵ_x max		2.76e-05		2.76e-05		2.76e-05		2.76e-05	
	ϵ_f med	1.7e-02	2.91e-07	1.7e-02	2.91e-07	1.1e-02	2.91e-07	1.1e-02	2.91e-07	median(δ)
	ϵ_f max	1.8e-01	7.84e-07	1.8e-01	7.84e-07	1.3e-01	7.84e-07	1.3e-01	7.84e-07	7.7e-05
	ϵ_L med		1.13e-13		6.88e-14		6.94e-14		7.35e-14	max(δ)
	ϵ_L max		1.34e-10		1.34e-10		3.92e-13		6.92e-13	2.2e-04
$7\times3\times3\times3\times3$	Iteration	6.2	9.9	30.4	3.7	11.4	14.7	30.9	3.1	33.9
	cpu (s): min/geo. mean/max	6.0/ 23.6/ 68.5	80.3/ 135.3/ 256.9	5.4/ 6.2/ 7.1	58.5/ 101.3/ 205.6	9.9/ 27.1/ 73.3	97.3/ 135.4/ 470.5	10.2/ 11.9/ 14.0	60.3/ 118.7/ 367.2	340.5/ 370.5/ 448.6
	mean cpu/node	5.64	0.50	1.23	0.38	3.51	0.52	1.32	0.47	1.32
	number of Node	(1 1 1 1 1) 5	(1 7 21 63 189) 281	(1 1 1 1 1) 5	(1 7 21 63 189) 281	(1 2 2 2 2) 9	(1 7 21 63 189) 281	(1 2 2 2 2) 9	(1 7 21 63 189) 281	
	Node visited	1 7 23 54 97	2 33 144 495 1521		1 26 115 410 1298	2 19 38 69 107	3 43 159 511 1516		1 22 103 409 1408	
	Cut	0.2 1.4 2.2 2.4	0.2 1.4 2.2 2.4		0.0 0.0 0.0 0.0	0.2 1.4 2.0 2.4	0.2 1.4 2.0 2.4		0.0 0.0 0.0 0.0	
	ϵ_x med		2.78e-06		2.78e-06		2.78e-06		2.78e-06	
	ϵ_x max		3.56e-05		3.56e-05		3.56e-05		3.56e-05	
	ϵ_f med	1.7e-02	3.44e-07	1.7e-02	3.44e-07	1.2e-02	3.44e-07	1.2e-02	3.44e-07	median(δ)
	ϵ_f max	1.8e-01	3.15e-06	1.8e-01	3.15e-06	1.3e-01	3.15e-06	1.3e-01	3.15e-06	1.5e-04
	ϵ_L med		4.63e-14		6.08e-14		9.93e-14		1.05e-13	max(δ)
	ϵ_L max		5.80e-13		5.38e-13		3.77e-10		3.77e-10	3.7e-04
$7\times5\times3\times3$	Iteration	6.2	10.2	0.0	4.0	9.2	12.7	0.0	3.1	0.0
	cpu (s): min/geo. mean/max	6.1/ 23.6/ 65.1	127.5/ 214.4/ 460.0	5.6/ 6.4/ 7.4	91.0/ 173.9/ 412.7	16.9/ 38.8/ 139.2	141.7/ 220.3/ 430.3	18.6/ 21.6/ 25.5	121.8/ 211.7/ 471.0	0.0/ 0.0/ 0.0
	mean cpu/node	5.77	0.48	1.29	0.40	3.16	0.50	1.45	0.48	0.00
	number of Node	(1 1 1 1 1) 5	(1 7 35 105 315) 463	(1 1 1 1 1) 5	(1 7 35 105 315) 463	(1 2 4 4 4) 15	(1 7 35 105 315) 463	(1 2 4 4 4) 15	(1 7 35 105 315) 463	
	Node visited	1 7 23 55 98	2 35 249 856 2655		1 28 214 743 2340	2 15 75 116 166	3 39 274 844 2508		1 22 186 720 2440	
	Cut	0.2 1.2 1.9 2.4	0.2 1.2 1.9 2.4		0.0 0.0 0.0 0.0	0.2 1.4 1.9 2.3	0.2 1.4 1.9 2.3		0.0 0.0 0.0 0.0	
	ϵ_x med		NaN		NaN		NaN		NaN	
	ϵ_x max		NaN		NaN		NaN		NaN	
	ϵ_f med	1.9e-02	NaN	1.9e-02	NaN	9.1e-03	NaN	9.1e-03	NaN	median(δ)
	ϵ_f max	2.1e-01	NaN	2.1e-01	NaN	1.6e-01	NaN	1.6e-01	NaN	NaN
	ϵ_L med		7.33e-14		6.82e-14		9.90e-14		8.75e-14	max(δ)
	ϵ_L max		4.08e-10		4.08e-10		3.09e-10		4.08e-10	NaN

Table 6.5: Numerical results for 3 stage tree with strictly convex subproblems

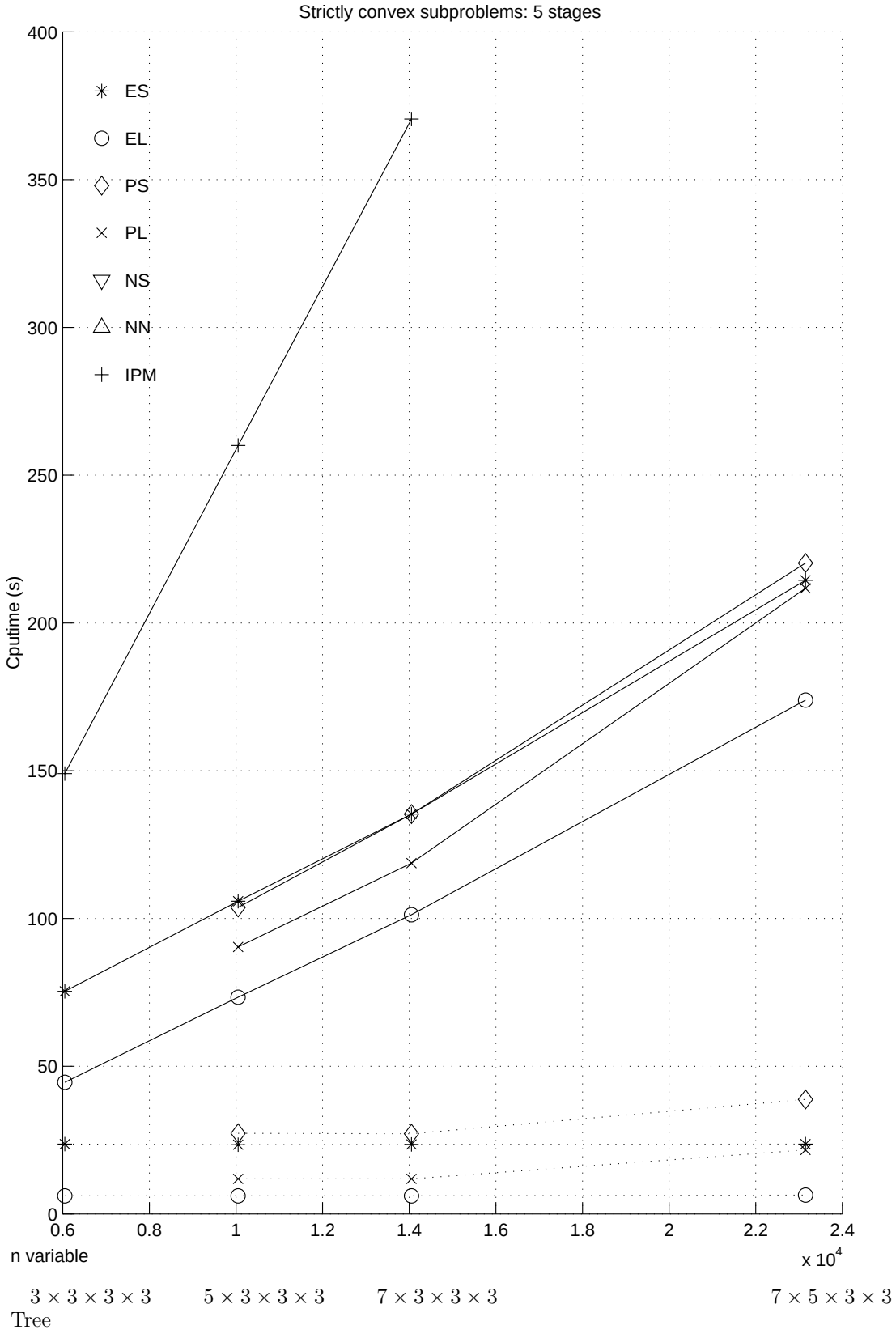

Figure 6.9: Cputime vs number of variables: 5 stage tree, strictly convex subproblems

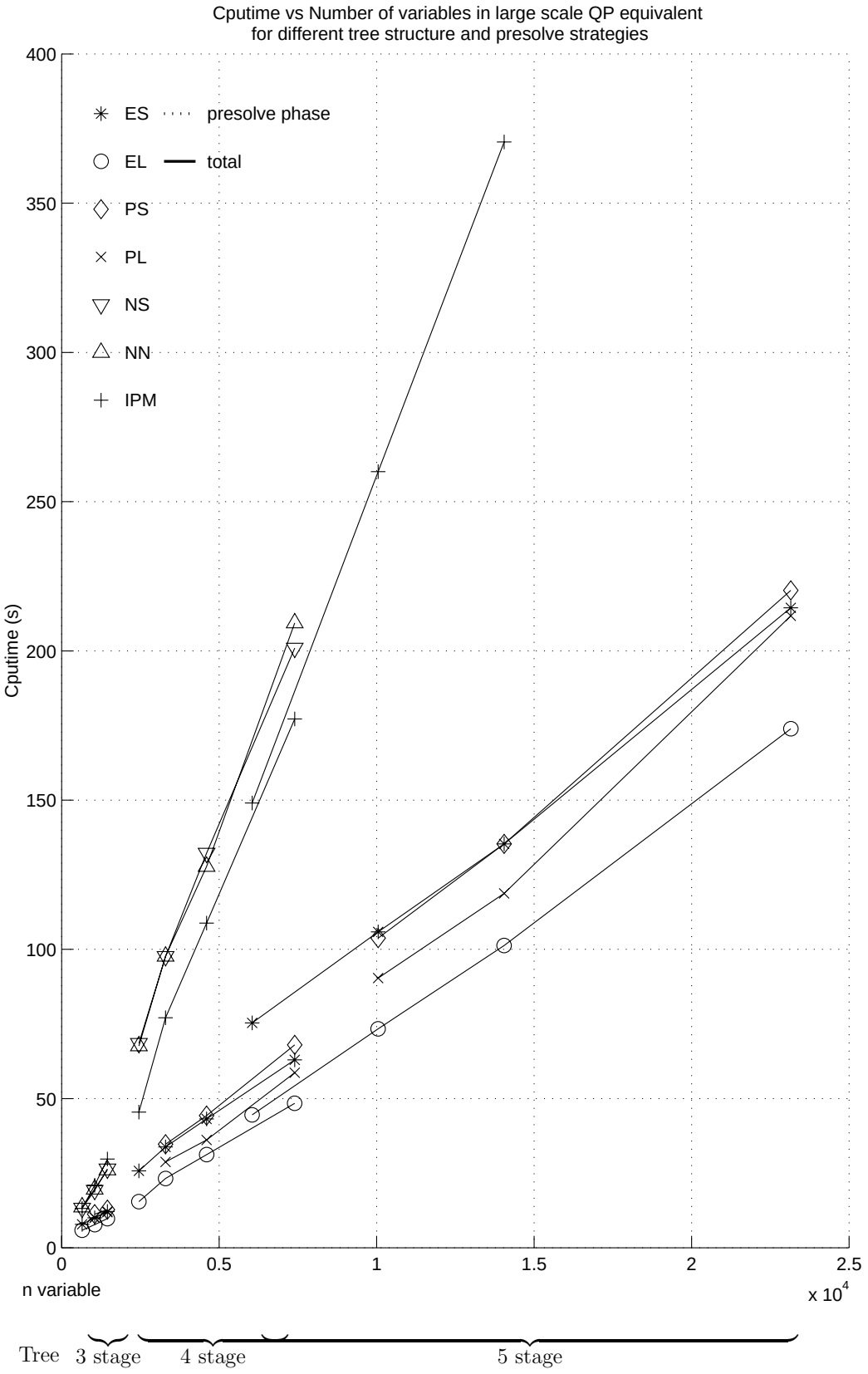

Figure 6.10: Cputime vs number of variables: strictly convex subproblems

6.2.4 Random Data with Convex Subproblems

In many applications, a significant proportion of the variables only appear linearly in the objective. The Hessian matrix has a number of zero rows and columns and is only positive semi-definite. The numerical experiments have been repeated for the case where each Hessian matrix is positive semi-definite with 10 zero eigenvalues. To distinguish this from problems with strictly convex objective, we sometimes refer to this class of problems as singular convex problems. Since a presolve phase is shown to reduce the amount of cputime required significantly, we do not test strategies NS and NN in this section. Tables 6.6 to 6.8 summarize the results. The relative error in x is not calculated since convex problems with singular Hessians do not in general have unique solutions. Figure 6.11 to 6.13 plot the geometric mean of cputime spent in problems with 3, 4 and 5 stages and Figure 6.14 plots the cputime taken for all problems tested.

The most obvious difference between solving strictly convex subproblems and singular convex subproblems is the increase in cputime. This is due to several factors. When the subproblems are convex but not strictly convex, the solutions are more dependent on the constraints and bounds. This makes it more likely to have more combinations of active sets and an increased number of quadratic pieces. Since Algorithm 4.3 is a recursive active set method, this leads to an increase in the number of iterations. The number of feasibility cuts also increased as the iterates are more likely to be at extreme points of the feasible regions making descendents problems more likely to be infeasible. The linear independence constraint qualification is also more likely to be violated, therefore recourse functions may become nondifferentiable. The loss of differentiability can lead to significant increase in cputime. Lastly, singular convex QPs require more time to solve than strictly convex QPs. The amount of cputime used by the interior point solver also increased for problems which are not strictly convex, but the difference is much smaller.

Comparing the different presolve strategies, it is clear that using an interior point solver to solve the presolve problem is much more efficient than using Algorithm 4.3. It also eliminates almost entirely the need for feasibility cuts except for two random

problems. For the 3 stage problems, solving the deterministic equivalent of the expected value problem in the presolve phase is the most efficient solution strategy. The same is true for the 4 and 5 stages problems with few branches. However, as the number of branches increases, solving a small presolve tree becomes more efficient overall as it provides a better starting point for the full problem.

One 5 stage random problem with tree structure $7 \times 3 \times 3 \times 3$ and $7 \times 5 \times 3 \times 3$ required more than 10 times more cputime to solve the full problem than the next slowest problem when the expected value solution is used as starting point. The same problem presolved with a small tree (PS and PL) required only a tiny fraction of the cputime used by DS and DL. This illustrates the sensitivity of active set methods to initial solutions and the advantage of using a small presolve tree which looks at a more balanced subset of the stochastic parameters and hence returns a better starting point.

Tree Structure	Method	E S		E L		P S		P L		IPM
		Expected	Total (incl. expected)	IPM	Total (incl. IPM)	Presolve	Total (incl. presolve)	IPM	Total (incl. IPM)	
3×3	Iteration	14.8	18.8	30.9	3.8	0.0	0.0	0.0	0.0	31.9
	cpu (s)	8.4/ 17.7/ 50.7	12.3/ 23.0/ 59.5	2.8/ 3.3/ 4.7	4.9/ 8.3/ 14.2	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	12.1/ 13.5/ 16.6
	cpu/node	6.53	1.90	1.10	0.66	0.00	0.00	0.00	0.00	1.04
	# Node	(1 1 1) 3	(1 3 9) 13	(1 1 1) 3	(1 3 9) 13	(1 1 1) 3	(1 3 9) 13	(1 1 1) 3	(1 3 9) 13	
	Node visited	1 17 78	2 29 152		1 12 69	0 0 0	0 0 0		0 0 0	
	Cut	0.5 2.0	0.5 2.0		0.0 0.0	0.0 0.0	0.0 0.0		0.0 0.0	
	ϵ_f med	4.1e-03	1.69e-15	4.1e-03	9.55e-16	0.0e+00	0.00e+00	0.0e+00	0.00e+00	median(δ)
	max		7.5e-02 1.94e-09	7.5e-02	1.94e-09	0.0e+00	0.00e+00	0.0e+00	0.00e+00	5.2e-14
	ϵ_L med		7.45e-14		4.72e-14		0.00e+00		0.00e+00	max(δ)
	max		5.26e-13		2.90e-13		0.00e+00		0.00e+00	4.4e-07
5×3	Iteration	14.2	18.3	30.9	4.0	22.8	26.6	31.1	3.2	32.5
	cpu (s)	8.5/ 17.0/ 46.6	13.3/ 25.2/ 60.2	2.8/ 3.3/ 4.7	5.6/ 11.0/ 19.1	10.1/ 22.5/ 46.6	15.4/ 30.1/ 61.7	4.9/ 5.9/ 9.5	8.9/ 13.8/ 26.4	20.6/ 22.3/ 28.5
	cpu/node	6.30	1.28	1.10	0.55	4.90	1.54	1.21	0.68	1.06
	# Node	(1 1 1) 3	(1 5 15) 21	(1 1 1) 3	(1 5 15) 21	(1 2 2) 5	(1 5 15) 21	(1 2 2) 5	(1 5 15) 21	
	Node visited	1 17 74	2 38 194		1 21 115	2 41 108	3 61 219		1 17 103	
	Cut	0.5 1.9	0.5 1.9		0.0 0.0	0.6 1.8	0.6 1.8		0.0 0.0	
	ϵ_f med	5.2e-03	1.72e-09	5.2e-03	1.72e-09	2.5e-03	1.72e-09	2.5e-03	1.72e-09	median(δ)
	max	1.0e-01	5.68e-08	1.0e-01	5.68e-08	4.5e-02	5.68e-08	4.5e-02	5.68e-08	9.0e-07
	ϵ_L med		4.25e-14		4.95e-14		4.18e-14		5.19e-14	max(δ)
	max		2.13e-13		1.39e-13		1.55e-13		1.18e-11	6.4e-06
7×3	Iteration	13.9	18.1	30.9	4.2	20.2	24.1	31.1	3.5	32.7
	cpu (s)	8.2/ 16.9/ 34.8	15.4/ 27.9/ 48.5	2.8/ 3.3/ 4.7	6.7/ 13.9/ 29.0	10.0/ 21.8/ 41.6	16.2/ 32.0/ 58.9	5.0/ 5.9/ 9.5	9.9/ 16.4/ 35.8	27.4/ 30.8/ 35.6
	cpu/node	6.13	1.01	1.10	0.51	4.64	1.16	1.21	0.59	1.06
	# Node	(1 1 1) 3	(1 7 21) 29	(1 1 1) 3	(1 7 21) 29	(1 2 2) 5	(1 7 21) 29	(1 2 2) 5	(1 7 21) 29	
	Node visited	1 16 72	2 46 236		1 30 160	2 36 100	3 65 255		1 26 143	
	Cut	0.5 1.9	0.5 1.9		0.0 0.0	0.6 1.8	0.6 1.8		0.0 0.0	
	ϵ_f med	5.3e-03	7.86e-09	5.3e-03	7.86e-09	2.8e-03	7.86e-09	2.8e-03	7.86e-09	median(δ)
	max	1.1e-01	3.33e-07	1.1e-01	3.33e-07	4.5e-02	3.33e-07	4.5e-02	3.33e-07	1.5e-06
	ϵ_L med		4.41e-14		4.54e-14		4.87e-14		5.67e-14	max(δ)
	max		1.14e-13		2.37e-13		7.57e-13		2.84e-13	1.3e-05

Table 6.6: Numerical results for 3 stage tree with convex subproblems

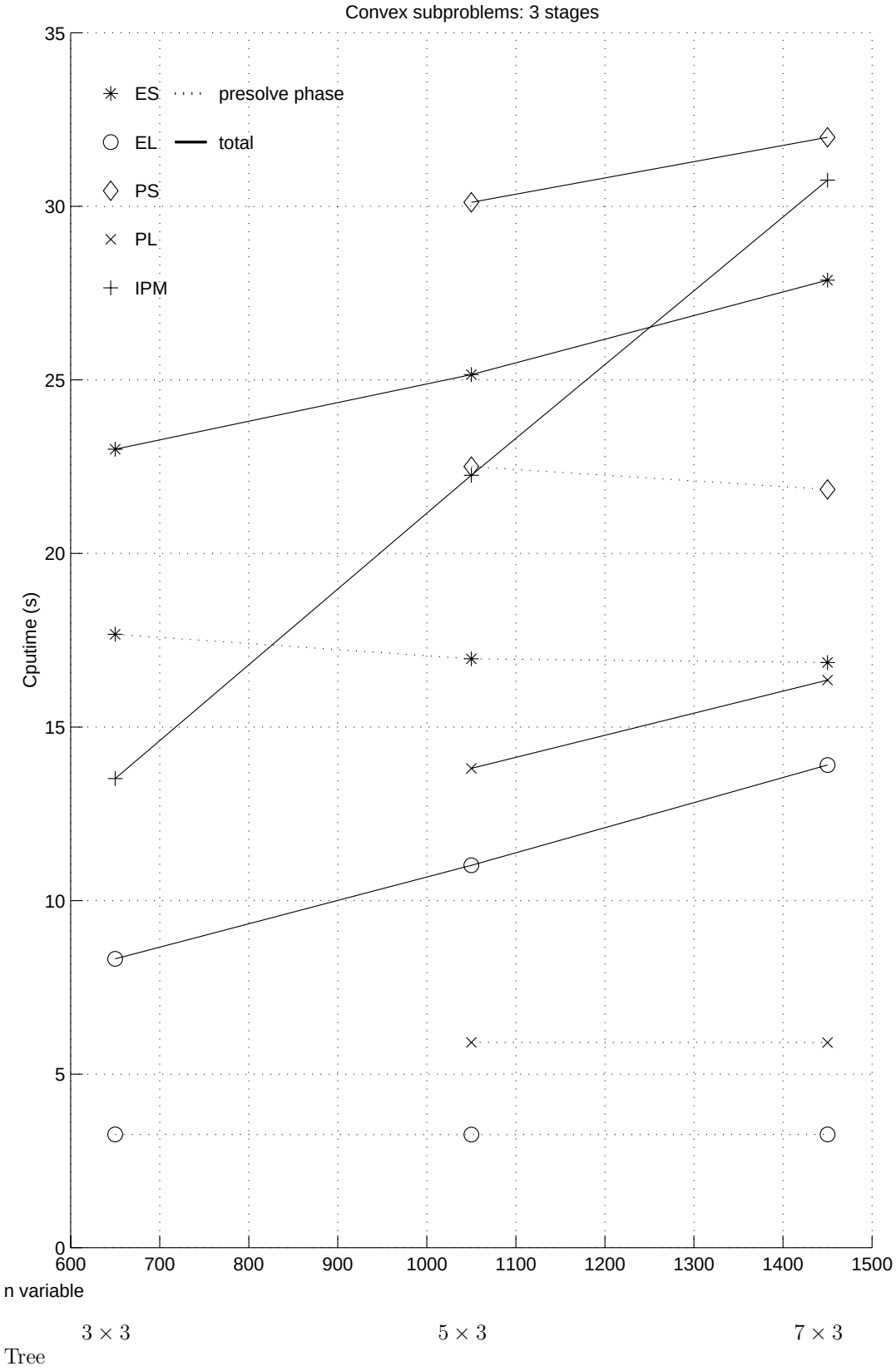

Figure 6.11: Cputime vs number of variables: 3 stage tree, convex subproblems

Tree Structure	Method	E S		E L		P S		P L		IPM
		Expected	Total (incl. expected)	IPM	Total (incl. IPM)	Presolve	Total (incl. presolve)	IPM	Total (incl. IPM)	
$3 \times 3 \times 3$	Iteration	13.9	18.1	30.8	4.0	0.0	0.0	0.0	0.0	33.0
	cpu (s)	18.8/ 49.9/ 283.7	34.6/ 78.8/ 318.1	4.1/ 5.4/ 8.2	15.2/ 30.5/ 59.6	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	40.0/ 48.2/ 74.2
	cpu/node	16.37	2.34	1.39	0.83	0.00	0.00	0.00	0.00	1.21
	# Node	(1 1 1 1) 4	(1 3 9 27) 40	(1 1 1 1) 4	(1 3 9 27) 40	(1 1 1 1) 4	(1 3 9 27) 40	(1 1 1 1) 4	(1 3 9 27) 40	
	Node visited	1 16 69 258	2 29 151 606		1 13 79 336	0 0 0 0	0 0 0 0		0 0 0 0	
	Cut	0.6 1.9 2.5	0.6 1.9 2.5		0.0 0.0 0.0	0.0 0.0 0.0	0.0 0.0 0.0		0.0 0.0 0.0	
	ϵ_f med	1.1e-02	1.18e-08	1.1e-02	1.18e-08	0.0e+00	0.00e+00	0.0e+00	0.00e+00	median(δ)
	max	1.2e-00	9.40e-08	1.2e-00	9.40e-08	0.0e+00	0.00e+00	0.0e+00	0.00e+00	3.7e-06
	ϵ_L med		9.59e-14		5.13e-14		0.00e+00		0.00e+00	max(δ)
	max		2.28e-12		6.58e-13		0.00e+00		0.00e+00	2.1e-05
$5 \times 3 \times 3$	Iteration	13.3	17.9	30.8	4.3	15.8	19.4	31.4	3.4	33.5
	cpu (s)	18.9/ 49.0/ 282.9	46.6/ 96.3/ 329.8	4.1/ 5.4/ 8.1	20.1/ 46.9/ 119.5	28.8/ 59.3/ 262.7	55.3/ 100.2/ 346.6	7.8/ 10.1/ 16.5	26.6/ 51.6/ 116.9	74.2/ 82.4/ 116.4
	cpu/node	15.77	1.67	1.39	0.79	10.76	1.74	1.48	0.87	1.26
	# Node	(1 1 1 1) 4	(1 5 15 45) 66	(1 1 1 1) 4	(1 5 15 45) 66	(1 2 2 2) 7	(1 5 15 45) 66	(1 2 2 2) 7	(1 5 15 45) 66	
	Node visited	1 15 67 252	2 38 204 837		1 23 134 574	2 26 91 300	3 45 202 783		1 18 114 538	
	Cut	0.5 1.8 2.5	0.5 1.8 2.5		0.0 0.0 0.0	0.6 1.7 2.9	0.6 1.7 2.9		0.0 0.0 0.0	
	ϵ_f med	1.3e-02	4.87e-08	1.3e-02	4.87e-08	8.9e-03	4.87e-08	8.9e-03	4.87e-08	median(δ)
	max	1.2e+00	2.03e-06	1.2e+00	2.03e-06	6.6e-01	2.03e-06	6.6e-01	2.03e-06	9.9e-06
	ϵ_L med		6.81e-14		7.29e-14		9.12e-14		6.48e-14	max(δ)
	max		5.98e-13		4.47e-13		6.31e-13		4.45e-13	4.3e-05
$7 \times 3 \times 3$	Iteration	13.7	18.4	30.8	4.5	15.8	19.6	31.4	3.2	34.4
	cpu (s)	18.8/ 48.4/ 284.2	55.3/ 115.3/ 355.3	4.1/ 5.4/ 8.1	24.2/ 63.7/ 175.6	28.7/ 58.3/ 263.2	62.5/ 113.0/ 363.3	7.8/ 10.1/ 16.5	30.9/ 62.8/ 141.4	103.3/ 119.3/ 184.9
	cpu/node	15.61	1.42	1.39	0.79	10.46	1.38	1.48	0.75	1.31
	# Node	(1 1 1 1) 4	(1 7 21 63) 92	(1 1 1 1) 4	(1 7 21 63) 92	(1 2 2 2) 4	(1 7 21 63) 92	(1 2 2 2) 4	(1 7 21 63) 92	
	Node visited	1 15 64 249	2 50 263 1100		1 34 195 832	2 26 87 286	3 53 241 943		1 24 149 682	
	Cut	0.6 1.8 2.5	0.6 1.8 2.5		0.0 0.0 0.0	0.6 1.8 2.8	0.6 1.8 2.8		0.0 0.0 0.0	
	ϵ_f med	1.3e-02	3.48e-08	1.3e-02	3.48e-08	9.2e-03	3.48e-08	9.2e-03	3.48e-08	median(δ)
	max	1.2e+00	2.63e-06	1.2e+00	2.63e-06	6.7e-01	2.63e-06	6.7e-01	2.63e-06	6.9e-06
	ϵ_L med		7.72e-14		1.11e-13		9.95e-14		8.43e-14	max(δ)
	max		4.91e-13		4.79e-13		5.52e-13		6.50e-13	4.0e-05
$7 \times 5 \times 3$	Iteration	13.7	18.6	30.8	4.5	16.1	20.2	31.6	3.4	34.7
	cpu (s)	20.5/ 47.6/ 249.7	89.7/ 159.1/ 349.5	4.1/ 5.5/ 8.2	38.0/ 97.5/ 267.3	38.9/ 94.8/ 295.7	82.9/ 189.4/ 471.3	12.6/ 14.4/ 20.0	49.5/ 94.8/ 230.5	169.7/ 203.3/ 354.8
	cpu/node	14.87	1.19	1.40	0.74	11.03	1.45	1.32	0.70	1.39
	# Node	(1 1 1 1) 4	(1 7 35 105) 148	(1 1 1 1) 4	(1 7 35 105) 148	(1 2 4 4) 11	(1 7 35 105) 148	(1 2 4 4) 11	(1 7 35 105) 148	
	Node visited	1 15 65 239	2 51 444 1824		1 33 348 1447	2 28 228 523	3 57 532 1793		1 24 246 1131	
	Cut	0.6 1.6 2.5	0.6 1.6 2.5		0.0 0.0 0.0	0.6 1.9 3.0	0.6 1.9 3.0		0.0 0.0 0.0	
	ϵ_f med	1.5e-02	9.97e-08	1.5e-02	9.97e-08	6.9e-03	9.97e-08	6.9e-03	9.97e-08	median(δ)
	max	1.1e+00	7.98e-06	1.1e+00	7.98e-06	2.4e-01	7.98e-06	2.4e-01	7.98e-06	1.8e-05
	ϵ_L med		8.36e-14		8.39e-14		1.18e-13		1.12e-13	max(δ)
	max		1.43e-12		7.72e-13		4.57e-13		6.26e-13	5.9e-05

Table 6.7: Numerical results for 4 stage tree with convex subproblems

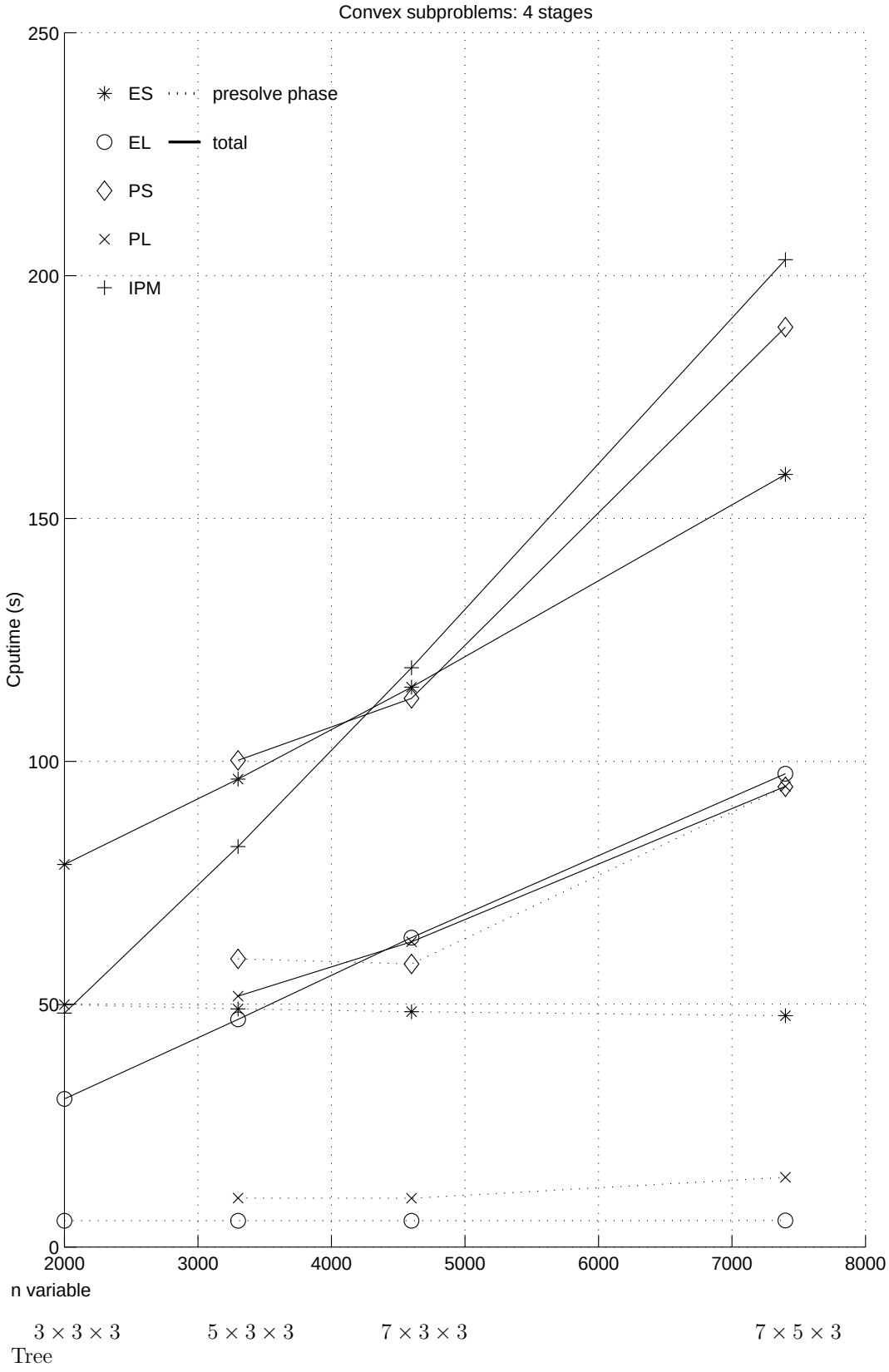

Figure 6.12: Cputime vs number of variables: 4 stage tree, convex subproblems

Tree Structure	Method	E S		E L		P S		P L		IPM
		Expected	Total (incl. expected)	IPM	Total (incl. IPM)	Presolve	Total (incl. presolve)	IPM	Total (incl. IPM)	
$3 \times 3 \times 3 \times 3 \times 3$	Iteration	13.7	18.1	31.0	4.2	0.0	0.0	0.0	0.0	34.9
	cpu (s)	58.0/ 150.8/ 891.4	131.0/ 279.5/ 973.9	5.9/ 7.1/ 11.4	56.6/ 102.4/ 238.7	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	0.0/ 0.0/ 0.0	142.9/ 162.9/ 346.3
	cpu/node	43.16	2.77	1.47	0.92	0.00	0.00	0.00	0.00	1.37
	# Node	(1 1 1 1 1) 5	(1 3 9 27 81) 121	(1 1 1 1 1) 5	(1 3 9 27 81) 121	(1 1 1 1 1) 5	(1 3 9 27 81) 121	(1 1 1 1 1) 5	(1 3 9 27 81) 121	
	Node visited	1 16 69 241 877	2 30 152 609 2293		1 13 77 335 1262	0 0 0 0 0	0 0 0 0 0		0 0 0 0 0	
	Cut	0.6 1.8 3.0 2.8	0.6 1.8 3.0 2.8		0.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0		0.0 0.0 0.0 0.0	
	ϵ_f med	9.8e-03	8.30e-08	9.8e-03	8.30e-08	0.0e+00	0.00e+00	0.0e+00	0.00e+00	median(δ)
	max	1.6e-01	4.02e-06	1.6e-01	4.02e-06	0.0e+00	0.00e+00	0.0e+00	0.00e+00	2.9e-05
	ϵ_L med		1.26e-13		8.86e-14		0.00e+00		0.00e+00	max(δ)
	max		3.92e-10		3.92e-10		0.00e+00		0.00e+00	8.8e-05
$5 \times 3 \times 3 \times 3 \times 3$	Iteration	13.7	18.3	31.0	4.3	18.0	21.9	31.7	3.5	34.6
	cpu (s)	40.1/ 159.6/ 886.4	168.3/ 368.0/ 1001.1	5.9/ 7.2/ 11.4	79.9/ 168.6/ 605.5	73.0/ 178.7/ 777.5	204.3/ 361.7/ 1111.7	11.0/ 14.1/ 27.7	97.5/ 186.0/ 559.5	244.4/ 277.8/ 346.7
	cpu/node	47.04	2.19	1.47	0.96	24.95	2.06	1.63	1.07	1.39
	# Node	(1 1 1 1 1) 5	(1 5 15 45 135) 201	(1 1 1 1 1) 5	(1 5 15 45 135) 201	(1 2 2 2 2) 9	(1 5 15 45 135) 201	(1 2 2 2 2) 9	(1 5 15 45 135) 201	
	Node visited	1 16 72 256 959	2 40 212 893 3407		1 23 134 600 2289	2 31 94 281 893	3 51 221 858 3103		1 18 115 567 2381	
	Cut	0.5 1.8 3.3 2.8	0.5 1.8 3.3 2.8		0.0 0.0 0.0 0.0	0.6 1.7 3.1 3.6	0.6 1.7 3.1 3.6		0.0 0.0 0.0 0.0	
	ϵ_f med	1.1e-02	2.86e-07	1.1e-02	2.86e-07	8.3e-03	2.86e-07	8.3e-03	2.86e-07	median(δ)
	max	1.6e-01	1.28e-06	1.6e-01	1.28e-06	1.5e-01	1.28e-06	1.5e-01	1.28e-06	8.4e-05
	ϵ_L med		1.07e-13		1.07e-13		9.01e-14		1.04e-13	max(δ)
	max		5.60e-10		5.60e-10		1.39e-12		1.22e-12	2.7e-04
$7 \times 3 \times 3 \times 3 \times 3$	Iteration	13.8	18.6	31.0	4.7	19.6	23.5	31.7	3.6	35.4
	cpu (s)	61.6/ 170.6/ 880.9	206.0/ 512.8/ 9380.5	6.0/ 7.2/ 11.4	121.3/ 278.8/11915.6	60.5/ 177.4/ 824.9	243.2/ 427.7/ 1257.5	11.1/ 14.1/ 27.8	124.8/ 245.6/ 635.6	361.3/ 412.6/ 493.3
	cpu/node	48.54	3.38	1.47	2.97	25.16	1.72	1.63	0.98	1.47
	# Node	(1 1 1 1 1) 5	(1 7 21 63 189) 281	(1 1 1 1 1) 5	(1 7 21 63 189) 281	(1 2 2 2 2) 9	(1 7 21 63 189) 281	(1 2 2 2 2) 9	(1 7 21 63 189) 281	
	Node visited	1 17 72 259 989	2 51 278 1336 8093		1 34 204 1111 8285	2 34 92 267 895	3 63 265 1042 3844		1 26 157 746 3037	
	Cut	0.6 1.7 3.4 3.0	0.6 1.7 3.4 3.1		0.0 0.0 0.0 0.0	0.6 1.7 3.2 3.6	0.6 1.7 3.2 3.6		0.0 0.0 0.0 0.0	
	ϵ_f med	1.1e-02	2.51e-07	1.1e-02	2.51e-07	8.4e-03	2.51e-07	8.4e-03	2.51e-07	median(δ)
	max	1.6e-01	2.79e-05	1.6e-01	2.79e-05	1.5e-01	2.79e-05	1.5e-01	2.79e-05	9.8e-05
	ϵ_L med		1.74e-13		9.98e-14		1.23e-13		1.79e-13	max(δ)
	max		4.87e-10		4.92e-10		1.38e-10		1.38e-10	2.8e-04
$7 \times 5 \times 3 \times 3$	Iteration	13.8	18.9	0.0	5.0	18.6	22.2	0.0	3.4	0.0
	cpu (s)	62.0/ 167.1/ 1052.8	346.8/ 733.2/ 9422.7	6.3/ 7.4/ 11.7	202.2/ 477.3/ 9992.6	82.4/ 212.5/ 1031.2	322.9/ 619.4/ 1359.6	20.3/ 25.1/ 44.7	244.7/ 425.2/ 951.4	0.0/ 0.0/ 0.0
	cpu/node	46.00	2.49	1.53	2.01	18.60	1.47	1.71	1.00	0.00
	# Node	(1 1 1 1 1) 5	(1 7 35 105 315) 463	(1 1 1 1 1) 5	(1 7 35 105 315) 463	(1 2 4 4 4) 15	(1 7 35 105 315) 463	(1 2 4 4 4) 15	(1 7 35 105 315) 463	
	Node visited	1 17 71 256 939	2 53 465 2150 10938		1 36 383 1891 10667	2 28 181 431 1128	3 55 455 1652 5849		1 25 260 1217 5069	
	Cut	0.6 1.6 3.2 3.4	0.6 1.6 3.2 3.5		0.0 0.0 0.0 0.0	0.6 2.0 3.2 3.0	0.6 2.0 3.2 3.0		0.0 0.0 0.0 0.0	
	ϵ_f med	9.9e-03	NaN	9.9e-03	NaN	4.2e-03	NaN	4.2e-03	NaN	median(δ)
	max	2.1e-01	NaN	2.1e-01	NaN	3.1e-02	NaN	3.1e-02	NaN	9.8e-05
	ϵ_L med		2.04e-13		1.21e-13		2.27e-13		1.70e-13	max(δ)
	max		1.25e-09		2.01e-10		3.54e-09		3.89e-09	2.8e-04

Table 6.8: Numerical results for 5 stage tree with convex subproblems

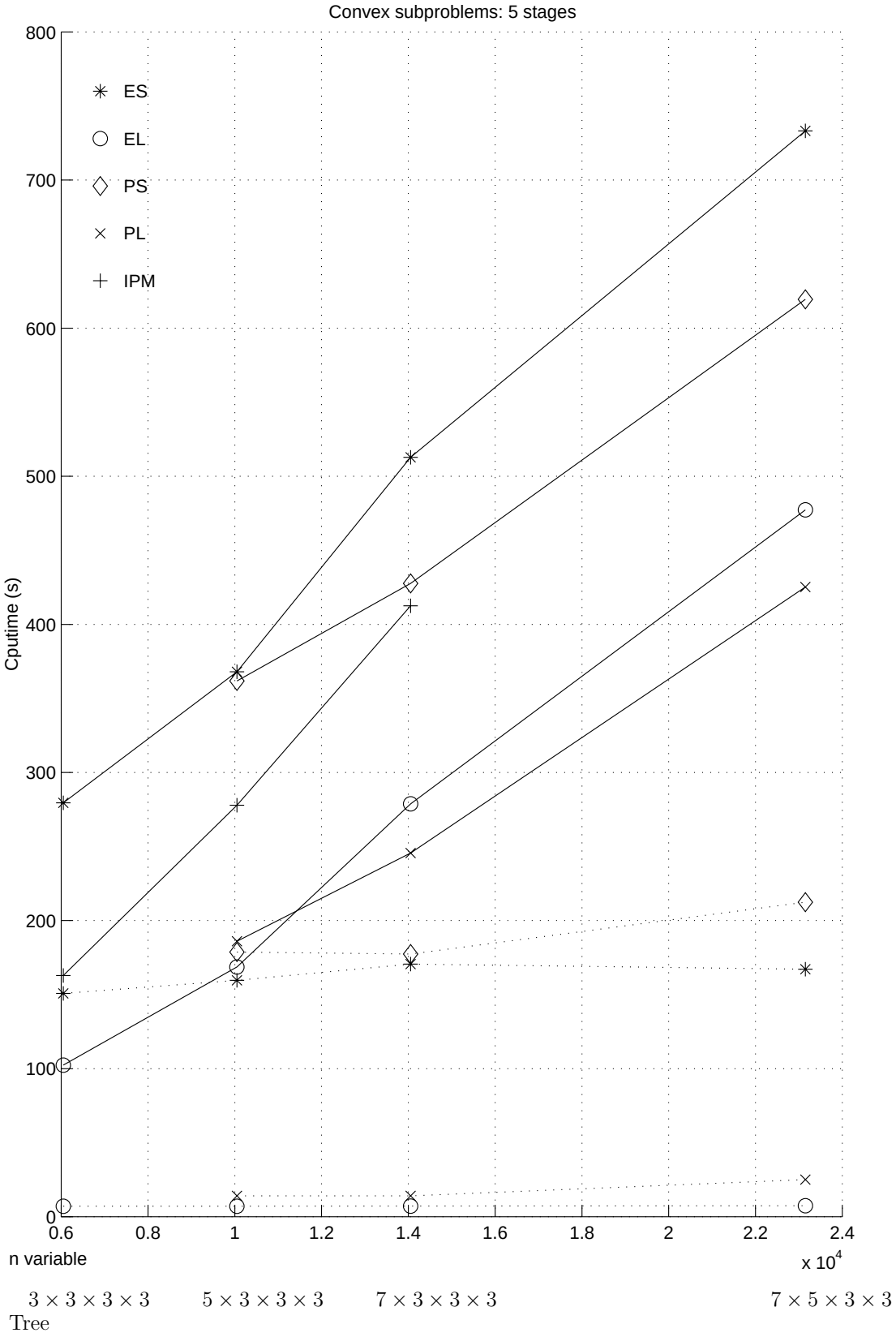

Figure 6.13: Cputime vs number of variables: 5 stage tree, convex subproblems

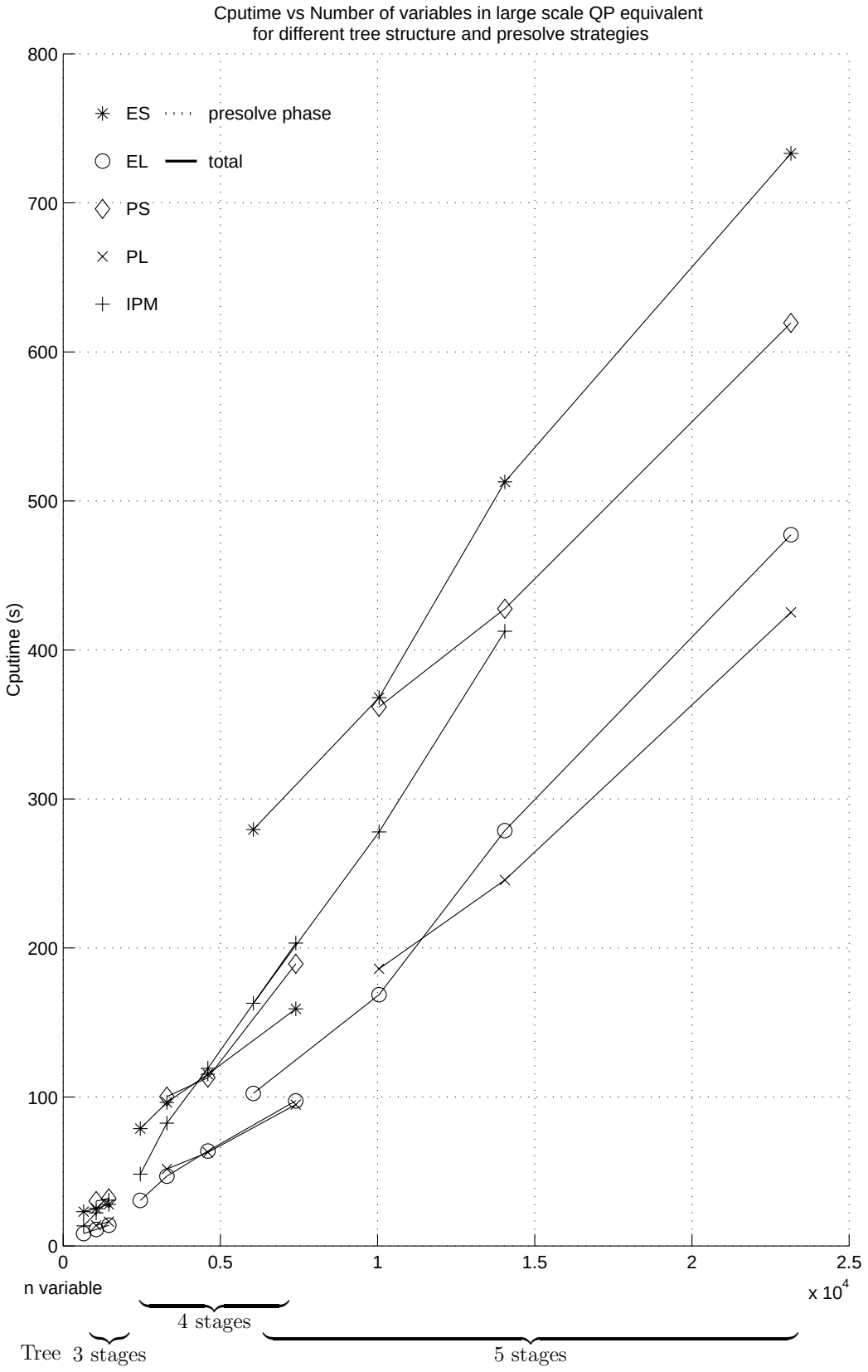

Figure 6.14: Cputime vs number of variables: convex subproblems

6.2.5 Stochastic Capacity Expansion Model

Due to the popularity of stochastic linear program, there are a number of publicly available test problems [36, 89, 113–115]. Unfortunately, there are not many test sets available for quadratic stochastic programming. Algorithm 4.3 is now tested on a test problem set called PLTEXP [113][1]. It is a linear relaxation of a stochastic capacity expansion problem. The problem is a stochastic linear program extendable to an arbitrary number of stages and scenarios. The problem was expressed in standard form where all constraints are equalities. We removed the slack variables and converted the appropriate equations to inequalities. The original data has two distinct sets of variables. The first 44 variables in each subproblem have coefficients in the cost function with order of magnitude 10^2, while the rest of the variables have coefficients of the order of 10^{-3} to 10^{-2}. In the numerical experiment, the second set of variables are rescaled by multiplying their coefficients by a factor of 10^3 to bring the two scales closer. To convert the problem to a multistage quadratic stochastic program, an artificial Hessian matrix is added to each subproblem. It has an entry of 10^{-3} on the diagonal for every variable that has a nonzero coefficient in the cost function, see Figure 6.6.

The problem set is tested for trees with 2 to 4 stages. The root node has 6 branches and other nodes have either 2 or 6 branches. The first stage problem has size 62×126 and each later stage problem has size 104×197. The size of the deterministic QP equivalent is shown in Table 6.9. To show the effect of the artificial Hessian term, we list the value of the optimal function value $f_{\text{lp}}(x_{\text{lp}})$ of the original LP, the optimal function value $f_{\text{qp}}(x_{\text{qp}})$ of the QP and the function value $(f_{\text{lp}}(x_{\text{qp}}))$ of the LP using the QP solution. The results are given in Table 6.10 and Figure 6.15. This problem set has very few optimal bases in the second and subsequent stages and this is reflected in the very small number of iterations required. Most problems only require one or two iterations before reaching optimality. This makes solving the multistage quadratic stochastic program directly without any presolve phase a very competitive option. Using an interior point solution as a starting point typically

[1] Available electronically at http://www-personal.umich.edu/~jrbirge/dholmes/main.html

Number of Stages	Tree Structure	size of deterministic QP	$f_{\text{lp}}(x_{\text{lp}})$	$f_{\text{qp}}(x_{\text{qp}})$	$f_{\text{lp}}(x_{\text{qp}})$
2	6	686×1308	-9.47935	-9.47910	-9.47936
	16	1726×3278	-9.66331	-9.66304	-9.66331
3	6×2	1934×3672	-13.64323	-13.64286	-13.64323
	6×6	4430×8400	-13.96937	-13.96898	-13.96937
4	$6 \times 2 \times 2$	4430×8400	-17.92819	-17.92773	-17.92819
	$6 \times 6 \times 6$	26894×50952	–	-19.59884	-19.59941

Table 6.9: Problem sizes of PLTEXP

requires one more iteration than using the solution from Algorithm 4.3. This is because the solution has many zero Lagrange multipliers making conversion to an active set solution very difficult, so the starting point is not as accurate as that from Algorithm 4.3.

There were no feasibility cuts in any problems so they were omitted in Table 6.10. Similar to the results for the random problems, the cputime required per node increases with the number of stages, but decreases as the number of siblings increases.

The large scale QP equivalent proved difficult for SPSOLQP [130]. The maximum infeasibility (defined in (6.2)) is up to 9.9e-5 for the 2 and 3 stages problems. In Table 6.10, we report the cputime used by SPSOLQP but the deterministic equivalent is solved again using LOQO [122] which was far more accurate. The accuracy of the solution from Algorithm 4.3 is calculated by comparing to the LOQO solution. The solution times for LOQO were not reported because it is programmed in C and the cputime is not comparable to Algorithm 4.3 which is written in MATLAB. The deterministic equivalent of the 6×6 and $6 \times 2 \times 2$ problems are too large for SPSOLQP and were solved only by LOQO. The deterministic equivalent of the 4 stage $6 \times 6 \times 6$ problem was not solved due to memory requirements in forming the QP.

Tree Structure	Method	E S		E L		P S		P L		N S	N N	IPM
		Expected	Total (incl. expected)	IPM	Total (incl. IPM)	Presolve	Total (incl. presolve)	IPM	Total (incl. IPM)			
6	Iteration	1.0	2.0	19.0	3.0	2.0	3.0	15.0	2.0	1.0	1.0	0.0
	cpu (s)	22.8	59.9	2.2	87.0	42.1	65.0	10.6	34.8	48.1	47.8	142.9
	cpu/node	11.42	8.56	1.10	12.43	14.02	9.29	3.52	4.98	6.87	6.83	20.41
	number of Node	(1 1) 2	(1 6) 7	(1 1) 2	(1 6) 7	(1 2) 3	(1 6) 7	(1 2) 3	(1 6) 7	(1 6) 7	(1 6) 7	
	Node visited	1 1	2 7		1 18	2 3	3 9		1 12	1 6	1 6	
	ϵ_f		5.68e-07		5.68e-07		5.68e-07		5.68e-07	5.68e-07	5.68e-07	
	ϵ_L		6.80e-14		3.53e-09		6.80e-14		1.86e-12	6.81e-14	9.70e-14	δ = 9.9e-05
16	Iteration	1.0	2.0	18.0	2.0	2.0	3.0	16.0	2.0	1.0	1.0	0.0
	cpu (s)	24.1	97.3	2.3	67.3	43.4	104.3	21.3	105.9	100.5	107.1	1486.1
	cpu/node	12.07	5.72	1.13	3.96	14.45	6.14	7.11	6.23	5.91	6.30	87.42
	number of Node	(1 1) 2	(1 16) 17	(1 1) 2	(1 16) 17	(1 2) 3	(1 16) 17	(1 2) 3	(1 16) 17	(1 16) 17	(1 16) 17	
	Node visited	1 1	2 17		1 32	2 4	3 20		1 32	1 16	1 16	
	ϵ_f		1.91e-07		1.91e-07		1.91e-07		1.91e-07	Inf	Inf	
	ϵ_L		6.80e-14		7.49e-13		6.81e-14		1.27e-09	6.80e-14	1.79e-11	δ = 3.8e-05
6×2	Iteration	1.0	2.0	20.0	2.0	2.0	3.0	22.0	2.0	1.0	1.0	0.0
	cpu (s)	55.4	436.7	8.7	471.5	124.7	405.5	26.2	505.3	341.4	332.0	0.0
	cpu/node	18.46	22.98	2.90	24.81	24.94	21.34	5.24	26.60	17.97	17.47	0.00
	number of Node	(1 1 1) 3	(1 6 12) 19	(1 1 1) 3	(1 6 12) 19	(1 2 2) 5	(1 6 12) 19	(1 2 2) 5	(1 6 12) 19	(1 6 12) 19	(1 6 12) 19	
	Node visited	1 1 1	2 7 15		1 12 26	2 3 3	3 9 15		1 12 28	1 6 12	1 6 12	
	ϵ_f		1.32e-07		1.32e-07		1.32e-07		1.32e-07	1.32e-07	1.32e-07	
	ϵ_L		6.80e-14		6.86e-10		3.33e-13		7.38e-11	3.16e-13	6.80e-14	δ = 0.0e+00
6×6	Iteration	1.0	2.0	20.0	2.0	2.0	3.0	21.0	2.0	1.0	3.0	0.0
	cpu (s)	76.7	571.1	8.9	500.7	138.2	574.1	62.1	461.0	513.4	1778.1	0.0
	cpu/node	25.56	13.28	2.98	11.64	27.64	13.35	12.41	10.72	11.94	41.35	0.00
	number of Node	(1 1 1) 3	(1 6 36) 43	(1 1 1) 3	(1 6 36) 43	(1 2 2) 5	(1 6 36) 43	(1 2 2) 5	(1 6 36) 43	(1 6 36) 43	(1 6 36) 43	
	Node visited	1 1 1	2 7 43		1 12 78	2 3 5	3 9 41		1 12 84	1 6 36	1 30 210	
	ϵ_f		1.62e-11		2.36e-11		Inf		Inf	Inf	Inf	
	ϵ_L		2.77e-13		4.44e-09		6.80e-14		1.45e-09	4.06e-13	1.33e-08	δ = 0.0e+00
$6 \times 2 \times 2$	Iteration	1.0	2.0	21.0	4.0	2.0	3.0	25.0	2.0	1.0	1.0	0.0
	cpu (s)	100.8	1420.9	14.4	1581.2	264.7	1502.8	51.5	3228.8	1196.3	1065.0	0.0
	cpu/node	25.19	33.04	3.61	36.77	37.82	34.95	7.36	75.09	27.82	24.77	0.00
	number of Node	(1 1 1 1) 4	(1 6 12 24) 43	(1 1 1 1) 4	(1 6 12 24) 43	(1 2 2 2) 7	(1 6 12 24) 43	(1 2 2 2) 7	(1 6 12 24) 43	(1 6 12 24) 43	(1 6 12 24) 43	
	Node visited	1 1 1 1	2 7 15 31		1 42 94 190	2 3 3 3	3 9 15 27		1 12 58 120	1 6 12 24	1 6 12 24	
	ϵ_f	6.0e-03	5.62e-12	6.0e-03	1.12e-10	1.9e-02	3.22e-10	1.9e-02	8.79e-11	6.74e-11	2.19e-11	
	ϵ_L		2.19e-13		5.72e-09		6.82e-14		1.57e-10	6.80e-14	6.81e-14	δ = 2.1e-15
$6 \times 6 \times 6$	Iteration	1.0	2.0	21.0	2.0	2.0	3.0	25.0	2.0	1.0	1.0	0.0
	cpu (s)	123.9	3907.7	14.8	3754.5	594.9	3709.8	1080.0	5214.6	4030.4	3605.8	0.0
	cpu/node	30.98	15.09	3.69	14.50	84.98	14.32	154.28	20.13	15.56	13.92	0.00
	number of Node	(1 1 1 1) 4	(1 6 36 216) 259	(1 1 1 1) 4	(1 6 36 216) 259	(1 2 2 2) 7	(1 6 36 216) 259	(1 2 2 2) 7	(1 6 36 216) 259	(1 6 36 216) 259	(1 6 36 216) 259	
	Node visited	1 1 1 1	2 7 43 259		1 12 78 474	2 3 7 13	3 9 43 235		1 12 84 534	1 6 42 258	1 6 36 216	
	ϵ_f	2.0e-02	NaN	2.0e-02	NaN	7.3e-02	NaN	7.3e-02	NaN	NaN	NaN	
	ϵ_L		6.81e-14		2.49e-10		6.81e-14		8.33e-12	2.76e-13	3.00e-13	δ = 0.0e+00

Table 6.10: Numerical results for problem PLTEXP: 2 to 4 stages

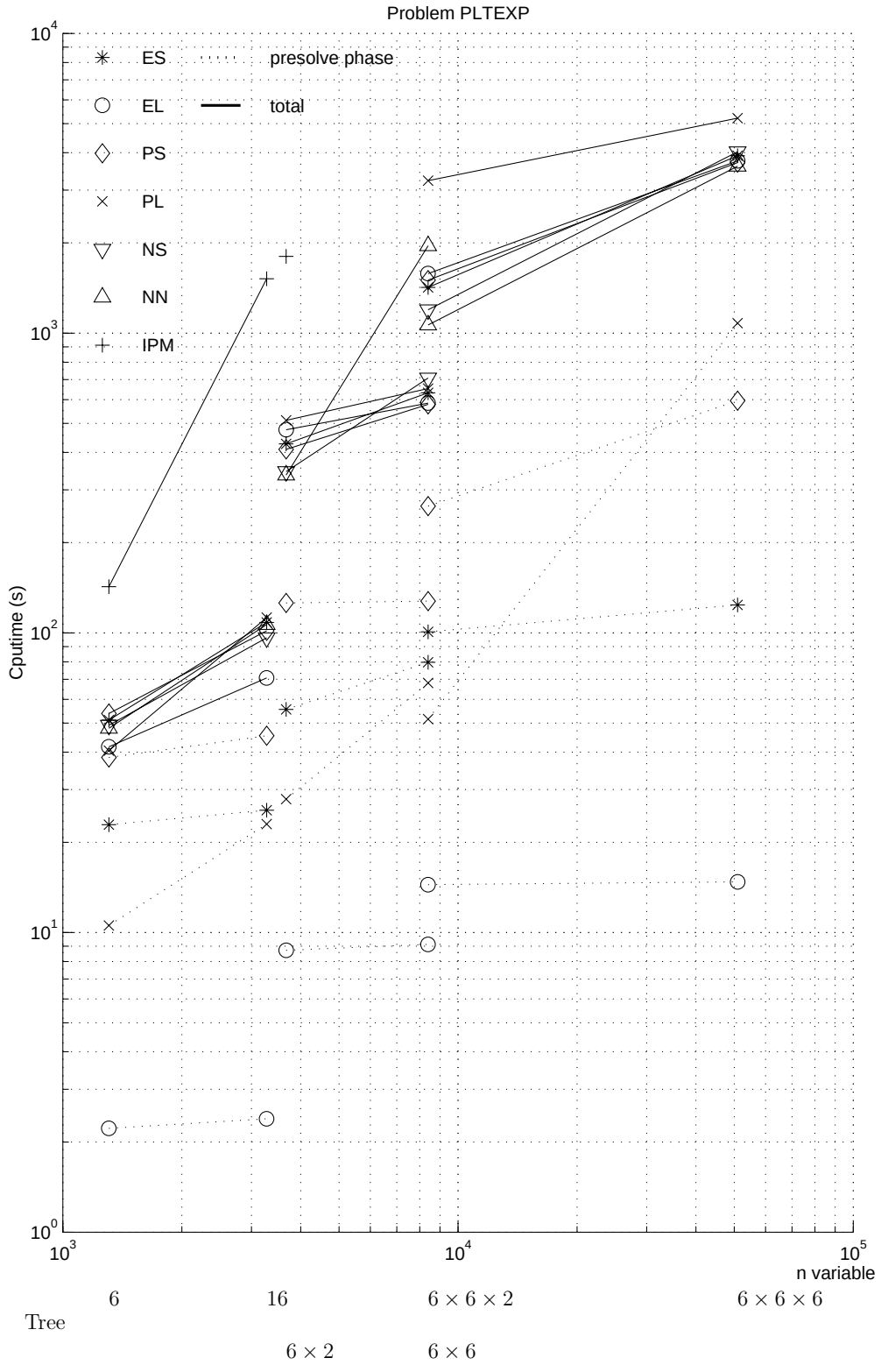

Figure 6.15: Cputime vs number of variables: PLTEXP, 2 to 4 stages

6.2.6 Portfolio Management Application

In this section, we test the performance of Algorithm 4.3 on a set of portfolio mangement problems called SGPF. The problem data is due to Frauendorfer [36] [2]. It is a portfolio optimization problems with stochastic parameters in both the linear cost function and the constraint right hand side. The original problems are stochastic linear programs with 3 to 6 stages. The magnitude of the variables of the original data is very large, so they are rescaled to have magnitude of 1 in the numerical experiments. The problems are converted into multistage quadratic stochastic programs by adding an artificial Hessian to each subproblem. Each Hessian has a small positive entry (10^{-4}) on the diagonal for every variable that has a nonzero entry in the linear cost function. Careful inspection of the problem structure reveals that many variables are in fact fixed by the equality constraints to be zero. Since all variables also have lower bounds of zero, this lead to a very large number of linearly dependent active constraints. Applying Algorithm 4.3 directly to the data proved to be extremely slow due to the large number of quadratic pieces and boundary constraints on which the objective is nonsmooth. A lot of the variables in stage 2 and above are also fixed by equality constraints with right hand side given by ancestor solution and cannot be eliminated easily. However, since all $h_t \geq 0$ and all entries in the technology matrices satisfy $V_t \leq 0$, the non-negativity constraint for these variables are clearly redundant and they are eliminated to reduce the possibility of having linearly dependent active constraints. Table 6.11 gives the size of each subproblem of the original problems, the size after redundant constraints and lower bounds are removed and the size of the equivalent large scale QP. The values of the optimal function value $f_{\text{lp}}(x_{\text{lp}})$ of the original LP, the optimal function value $f_{\text{qp}}(x_{\text{qp}})$ of the QP and the function value $(f_{\text{lp}}(x_{\text{qp}}))$ of the LP using the QP solution are also listed to show the effect of the artificial Hessian term.

The numerical results are given in Table 6.12 and Figure 6.16. The large scale equivalent of the 4 stage problem was very large and was only solved by LOQO [122]. The QP equivalent of the 5 and 6 stage problem are too large and were not solved

[2] Available electronically at http://www-personal.umich.edu/~jrbirge/dholmes/main.html

in the experiments. Algorithm 4.3 successfully solved the SGPF problems for 3 to 6 stages. The number of iterations is very small for all numbers of stages and solution strategies. This indicates that the first stage solution is very close to the solution of the stage one QP problem. The amount of cputime required increases approximately linearly with the size of the deterministic equivalent. The performance of the different solution strategies is mixed. Sorting the stochastic right hand side only lead to a reduction in cputime in the 3 stage problems. The difference in optimal function value between the presolve problem and the original problem for strategies PS and PL are much bigger than when the expected valued problem was used. These indicate the simple heuristic which only sorts the right hand side but not the stochastic linear cost function is inadequate for this problem. This also explains why the presolve trees have much larger differences in function values to the full problems than the expected value trees and the relatively poor performance of strategies PS and PL especially for the 5 and 6 stage problem.

	Original			Eliminated					
Stage	size of W_t	number of lower bound	size of deterministic QP	size of W_t	number of lower bound	size of deterministic QP	$f_{\text{lp}}(x_{\text{lp}})$	$f_{\text{qp}}(x_{\text{qp}})$	$f_{\text{lp}}(x_{\text{qp}})$
1	62×78	78		11×27	27				
2	63×79	79		17×33	27				
3	63×79	79	1952×2448	23×39	27	671×1167	-3.027604e+03	-3.027603e+03	-3.027604e+03
4	63×79	79	9827×12323	28×44	27	4171×6667	-4.031391e+03	-4.031389e+03	-4.031391e+03
5	63×79	79	49202×61698	33×49	27	24796×37292	–	-5.201260e+03	-5.201265e+03
6	63×79	79	246077×308573	38×54	27	143546×206042	–	-6.484472e+03	-6.484479e+03

Table 6.11: Problem sizes of SGPF

Tree Structure	Method	E S		E L		P S		P L		N S	N N	IPM
		Expected	Total (incl. expected)	IPM	Total (incl. IPM)	Presolve	Total (incl. presolve)	IPM	Total (incl. IPM)			
5 × 5	Iteration	2.0	3.0	15.0	1.0	2.0	4.0	13.0	3.0	2.0	2.0	27
	cpu (s)	3.2	10.9	0.4	8.2	2.6	10.8	1.5	13.5	10.6	12.8	5.4
	mean cpu/node	1.08	0.35	0.14	0.26	0.52	0.35	0.29	0.44	0.34	0.41	0.17
	number of Node	(1 1 1) 3	(1 5 25) 31	(1 1 1) 3	(1 5 25) 31	(1 2 2) 5	(1 5 25) 31	(1 2 2) 5	(1 5 25) 31	(1 5 25) 31	(1 5 25) 31	
	Node visited	1 2 5	2 7 75		1 5 70	2 3 17	3 13 112		1 15 95	1 10 95	1 10 110	
	Cut	0.0 1.0	0.0 1.0		0.0 0.0	0.0 0.0	0.0 0.0		0.0 0.0	0.0 0.0	0.0 0.0	
	ϵ_f	1.9e-04	4.37e-09	1.9e-04	4.37e-09	1.5e-01	4.37e-09	1.5e-01	5.34e-09	4.37e-09	4.37e-09	
	ϵ_L		4.56e-11		3.25e-11		3.91e-16		6.82e-16	1.62e-10	4.10e-16	$\delta = 1.3e\text{-}09$
5 × 5 × 5	Iteration	2.0	4.0	13.0	2.0	2.0	3.0	15.0	1.0	1.0	1.0	0.0
	cpu (s)	4.6	83.2	0.7	73.5	15.1	59.8	4.4	73.3	73.5	58.8	0.0
	mean cpu/node	1.16	0.53	0.18	0.47	2.16	0.38	0.63	0.47	0.47	0.38	0.00
	number of Node	(1 1 1 1) 4	(1 5 25 125) 156	(1 1 1 1) 4	(1 5 25 125) 156	(1 2 2 2) 7	(1 5 25 125) 156	(1 2 2 2) 7	(1 5 25 125) 156	(1 5 25 125) 156	(1 5 25 125) 156	
	Node visited	1 2 3 9	2 12 68 469		1 10 65 435	2 3 17 39	3 8 57 364		1 5 75 495	1 5 70 495	1 5 45 350	
	Cut	0.0 1.0 1.0	0.0 1.0 1.0		0.0 0.0 0.0	0.0 0.0 0.0	0.0 0.0 0.0		0.0 0.0 0.0	0.0 0.0 0.0	0.0 0.0 0.0	
	ϵ_f	9.5e-02	1.82e-10	9.5e-02	1.82e-10	2.9e-01	1.82e-10	2.9e-01	2.25e-09	2.25e-09	1.82e-10	
	ϵ_L		7.59e-16		6.25e-16		1.83e-15		1.31e-12	7.28e-16	2.38e-15	$\delta = 9.6e\text{-}13$
5 × 5 × 5 × 5	Iteration	1.0	2.0	14.0	1.0	4.0	6.0	15.0	1.0	2.0	1.0	0.0
	cpu (s)	3.8	461.4	1.2	615.7	72.8	780.7	12.7	684.7	836.2	640.7	0.0
	mean cpu/node	0.75	0.59	0.25	0.79	8.09	1.00	1.41	0.88	1.07	0.82	0.00
	number of Node	(1 1 1 1 1) 5	(1 5 25 125 625) 781	(1 1 1 1 1) 5	(1 5 25 125 625) 781	(1 2 2 2 2) 9	(1 5 25 125 625) 781	(1 2 2 2 2) 9	(1 5 25 125 625) 781	(1 5 25 125 625) 781	(1 5 25 125 625) 781	
	Node visited	1 1 2 2 2	2 6 47 317 2097		1 5 70 495 3010	2 11 21 59 123	3 21 116 699 3833		1 5 100 630 3620	1 20 155 850 4725	1 5 50 435 2570	
	Cut	0.0 1.0 1.0 1.0	0.0 1.0 1.0 1.0		0.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0	0.0 3.0 0.0 0.0		0.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0	
	ϵ_f	1.6e-01	NaN	1.6e-01	NaN	4.4e-01	NaN	4.4e-01	NaN	NaN	NaN	
	ϵ_L		1.67e-15		1.54e-15		9.31e-16		1.36e-12	9.59e-16	5.04e-16	$\delta = 0.0e\text{+}00$
5	Iteration	1.0	2.0	11.0	1.0	2.0	3.0	18.0	1.0	1.0	1.0	0.0
	cpu (s)	6.5	5034.2	1.9	5650.4	362.1	10814.7	35.7	8450.4	7249.3	5278.1	0.0
	mean cpu/node	1.09	1.61	0.31	1.81	32.92	3.46	3.24	2.70	2.32	1.69	0.00
	number of Node	(1 1 1 1 1 1) 6	(1 5 25 125 625 3125) 3906	(1 1 1 1 1 1) 6	(1 5 25 125 625 3125) 3906	(1 2 2 2 2 2) 11	(1 5 25 125 625 3125) 3906	(1 2 2 2 2 2) 11	(1 5 25 125 625 3125) 3906	(1 5 25 125 625 3125) 3906	(1 5 25 125 625 3125) 3906	
	Node visited	1 1 2 2 2 2	2 6 62 487 3477 19422		1 5 70 560 3545 19330	2 3 5 23 58 115	3 8 120 903 5663 29795		1 5 50 410 3135 17905	1 5 50 425 2585 14335	1 5 25 175 1485 9090	
	Cut	0.0 1.0 1.0 1.0 1.0	0.0 1.0 1.0 1.0 1.0		0.0 0.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0 0.0		0.0 0.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0 0.0	0.0 0.0 0.0 0.0 0.0	
	ϵ_f	1.6e-01	NaN	1.6e-01	NaN	6.2e-01	NaN	6.2e-01	NaN	NaN	NaN	
	ϵ_L		5.37e-16		7.96e-16		7.12e-15		6.34e-16	1.60e-15	7.73e-16	$\delta = 0.0e\text{+}00$

Table 6.12: Numerical results for problem SGPF: 3 to 6 stages

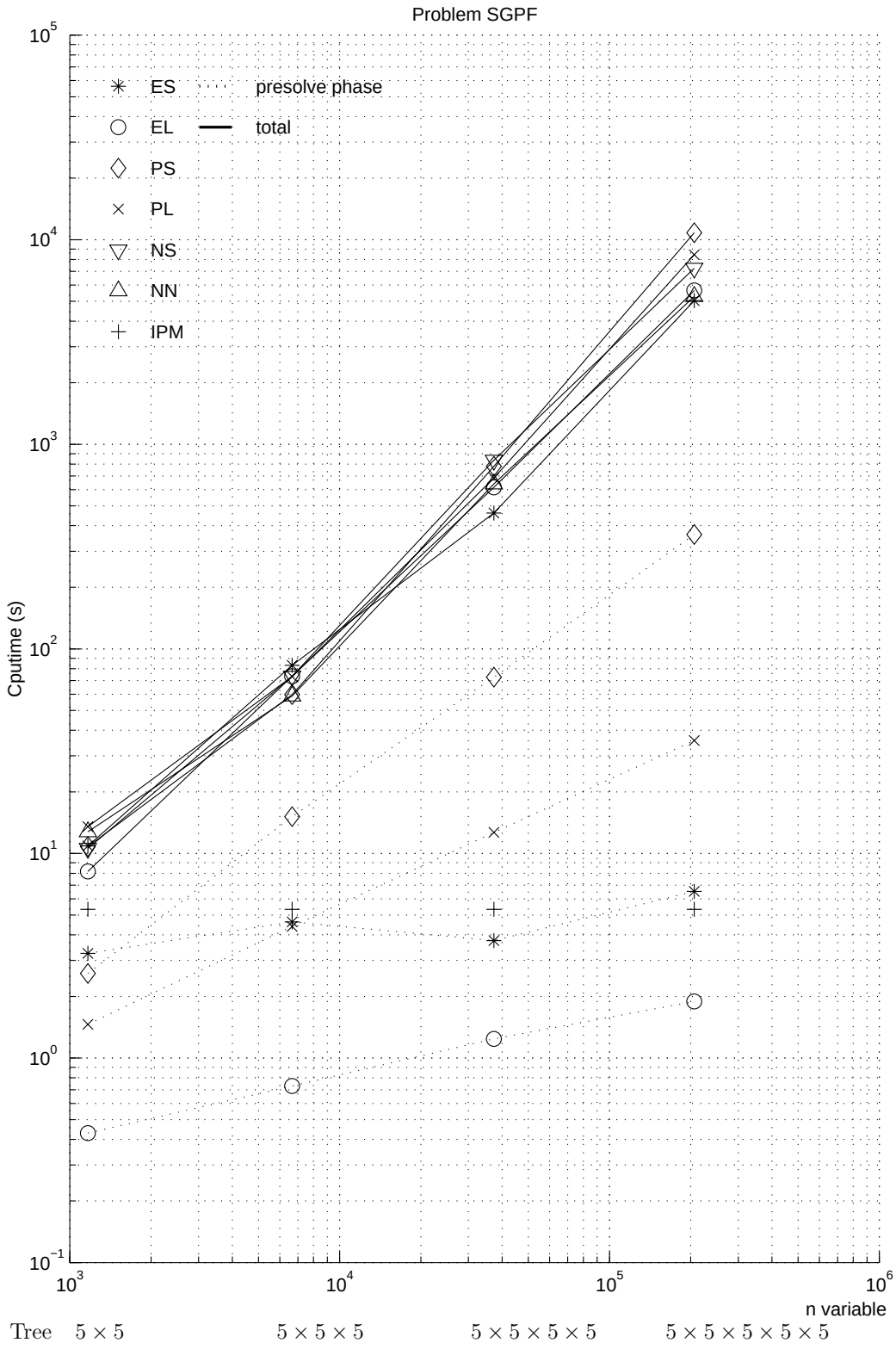

Figure 6.16: Cputime vs number of variables: SGPF, 3 to 6 stages

Chapter 7

Conclusion

Uncertainty is an integral part of planning for the future. Multistage stochastic programming allows decision makers to use mathematical programming techniques to design strategies that are 'optimal' taking into account a number of possible future scenarios.

In this thesis, we considered a class of multistage stochastic programs where each subproblem is a convex quadratic program. Unlike the success of quadratic programming and stochastic linear programming, there has been relatively little work on quadratic stochastic programming. We believe this is an important class of problems as it generalizes stochastic linear program, allowing for some degree of nonlinearity, while remaining structured enough to be solved efficiently. This is particularly relevant to the finance industry where the standard risk measure is the variance. The theory and algorithms developed here are applicable if the convex quadratic objectives are replaced by piecewise quadratic functions. This extension has important applications in financial planning as convex piecewise quadratic functions can serve as very flexible risk measure. This allows investors to design risk measures and therefore investment portfolios that suit their individual circumstances.

Each recourse function of a multistage quadratic stochastic program (1.12)–(1.13) is a convex piecewise quadratic function. Using results from convex analysis and sensitivity analysis, we studied the structure of the piecewise quadratic recourse

function. This enabled us to derive expressions for component quadratic functions which provide elements of subdifferentials and generalized Hessians of the piecewise quadratic objectives in (1.12)–(1.13).

Generalized Newton methods were investigated for solving problem (1.12)–(1.13). The algorithms were proven to converge globally. If the piecewise quadratic objective is strictly convex and differentiable at the solution, the convergence is finite.

Numerical experiments were carried out to test the accuracy and efficiency of Algorithm 4.3. The results show that the generalized Newton method is a highly accurate method for solving multistage quadratic stochastic programs. Comparison of the different solution strategies show that using the expected value problem solutions as starting points is a very efficient way to speed up solution time if the number of scenarios is moderate and they are not too different from each other. For problems with many diverse scenarios, solving a small representative subset of them proves to be more efficient by generating a better starting point that takes into account of the different stochastic outcomes.

The expected value problem and the small presolve tree of a multistage quadratic stochastic program are much smaller than the original problem. Hence, they can often be solved as large sparse deterministic problems using standard QP software. With the capacity of modern sophisticated QP software, especially with interior point methods which have proven to be very efficient for large sparse problem, this can be a very fast and reliable option. On the other hand, the multistage quadratic stochastic program formulation is in general not as smooth as the QP. It is usually necessary in the early stages of the presolve phase to switch quadratic pieces many times which may be expensive especially if the number of stages is large. Unlike the full problem, where Algorithm 4.3 can use siblings' solutions as starting points when there are many siblings, there is little comparative advantage over the QP formulation in the presolve phase. This suggests that solving the deterministic equivalent may be a faster presolve option if the size of the problem is not too large. If the equivalent problem is very large and requires too much memory to solve efficiently, then a combination of the two alternatives, where subproblems across

a few stages of the multistage quadratic stochastic program are aggregated into a single QP can be used. This reduces (1.12)–(1.13) to a quadratic stochastic program with only a few stages. Algorithm 4.3 can then solve it very efficiently.

For multistage quadratic stochastic programs with the same number of stages, the amount of cputime required by Algorithm 4.3 increases approximately linearly with the size of the equivalent large scale QP problem. Cputime required per node increases as the number of stages increases since the level of recursion is increased. This suggests that it may be more efficient to aggregate problems from two or more consecutive stages to form larger subproblems and reduce the number of stages. This can be done most easily in the root node since we do not need to calculate recourse information, so interior point methods can be used without converting back to an active set solution.

For multistage quadratic stochastic program, subgradients and generalized Hessians of the recourse functions depend on the active sets of the descendent problems. This requires each subproblem be solved to optimality in every iteration. This may not be very efficient when the iterates are still very far from the solution. It may be possible to relax this requirement by further research. We may be able to use approximate dual solution to obtain ϵ-subdifferential information. Second order information may be obtained by ways similar to the quasi-Newton method. This would allow the use of an interior point method for solving subproblems which may be more efficient if each subproblems is large and sparse, for example, when they are obtained by aggregating subproblems from two or more stages. We can switch back to using an exact active set solution when the iterate is close to a solution.

One difficulty in combining interior point solvers with Algorithm 4.3 is that Algorithm 4.3 is a recursive active set method. The starting point from an interior point solver is usually not as accurate as that of an active set solution and does not provide any active set information. Algorithm 4.3 may need to perform more iterations than when an active set solution is used as starting point. Interior point methods are also less easy to warm start than active set methods, making them less attractive when a sequence of similar problems needs to be solved. Combining

interior point algorithm with active set method and warm starting [41] interior point method are very active research topics. Advances in these areas would allow Algorithm 4.3 to take advantage of the efficiency of interior point methods for large sparse subproblems.

Algorithm 4.3 can be used to solve multistage stochastic program where each problem in (1.12)–(1.13) is a piecewise linear-quadratic program. For this class of problems to be solved efficiently, we need data structures that can store the data efficiently and an algorithm that can take advantage of the special structure of piecewise quadratic program. This will allow more general problems to be solved such as using piecewise linear-quadratic risk measures to model the risk in different scenarios.

Bibliography

[1] A. Agrawal, P. Klein, and R. Ravi, *Cutting down on fill using nested dissection: provably good elimination orderings*, in Graph theory and sparse matrix computation, Springer, New York, 1993, pp. 31–55.

[2] R. Bellman, *Dynamic Programming*, Princeton University Press, New Jersey, 1957.

[3] J. F. Benders, *Partitioning procedures for solving mixed-variables programming problems*, Numerische Mathematik, 4 (1962), pp. 238–252.

[4] A. J. Berger, J. M. Mulvey, and A. Ruszczyński, *An extension of the DQA algorithm to convex stochastic programs*, SIAM Journal on Optimization, 4 (1994), pp. 735–753.

[5] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*, Academic Press, New York and London, 1982.

[6] ——, *Dynamic Programming and Optimal Control*, Prentice Hall, Englewood Cliffs, NJ, 1995.

[7] J. R. Birge, *Decomposition and partitioning methods for multi-stage stochastic lineat programs*, Operations Research, (1985), pp. 989–1007.

[8] ——, *Stochastic programming computation and applications*, INFORMS Journal on Computing, 9 (1997), pp. 111–133.

[9] J. R. Birge, C. J. Donohue, D. F. Holmes, and O. G. Svintsitski, *A parallel implementation of the nested decomposition algorithm for multistage*

stochastic linear programs, Mathematical Programming, 75 (1996), pp. 327–352.

[10] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*, Springer-Verlag, New York, 1997.

[11] J. R. Birge and C. H. Rosa, *Parallel decomposition of large scale stochastic nonlinear programs*, Annals of Operations Research, 64 (1996), pp. 39–65.

[12] R. H. Bisseling, T. M. Doup, and L. D. J. C. Loyens, *A parallel interior point algorithm for linear programming on a network of transputers*, Annals of Operations Research, 43 (1993), pp. 51–86. Applied mathematical programming and modelling (Uxbridge, 1991).

[13] D. R. Cariño, T. Kent, D. H. Myers, C. Stacy, M. Sylvanus, A. L. Turner, K. Watanabe, and W. T. Ziemba, *The Russell-Yasuda Kasai model: An asset/liability model for a Japanese insurance company using multistage stochastic prgramming*, Interfaces, 24 (1994), pp. 29–49.

[14] D. R. Cariño and A. L. Turner, *Multiperiod Asset Allocation with Derivative Assets*, in Mulvey and Ziemba [92], 1998, pp. 182–204.

[15] R. W. Chaney, *Piecewise C^k functions in nonsmooth analysis*, Nonlinear Analysis, 15 (1990), pp. 649–660.

[16] X. Chen, L. Qi, and R. S. Womersley, *Newton's method for quadratic stochastic programs with recourse*, Journal of Computational and Applied Mathematics, 60 (1995), pp. 29–46.

[17] X. Chen and R. S. Womersley, *A parallel inexact Newton method for stochastic programs with recourse*, Annals of Operations Research, 64 (1996), pp. 113–141.

[18] E. W. Cheney and A. A. Goldstein, *Newton's method for convex programming and Tchebycheff approximation*, Numerische Mathematik, 1 (1959), pp. 253–268.

[19] F. H. Clarke, *Optimization and Nonsmooth Analysis*, John Wiley, Chichester and New York, 1983.

[20] G. Consigli and M. A. H. Dempster, *Dynamic stochastic progrmming for asset-liability management*, to appear in Annals of Operations Research.

[21] CPLEX Optimization, Inc., *Using the CPLEX Callable Library*, Incline Village, NV, USA, 1995.

[22] J. Czyzyk, R. Fourer, and S. Mehrotra, *Using a massively parallel processor to solve large sparse linear programs by an interior-point method*, SIAM Journal on Scientific Computing, 19 (1998), pp. 553–565.

[23] G. B. Dantzig and P. W. Glynn, *Parallel processors for planning under uncertainty*, Annals of Operations Research, 22 (1990), pp. 1–21.

[24] G. B. Dantzig and P. Wolfe, *The decomposition principle for linear programs*, Operations Research, 8 (1960), pp. 101–111.

[25] A. de Silva and D. Abramson, *A parallel interior point method and its application to facility location problems*, Computational Optimization and Applications, 9 (1998), pp. 249–273.

[26] J. Dupačová, M. Bertocchi, and V. Moriggia, *Postoptimality for scenario based financial planning models with an application to bond portfolio management*, in Mulvey and Ziemba [92], 1998, pp. 263–285.

[27] E. J. Elton and M. J. Gruber, *Modern Portfolio Theory and Investment Analysis*, John Wiley & Sons, Toronto, 1991.

[28] Y. Ermoliev, *Stochastic quasigradient methods and their application to systems optimization*, Stochastics, 9 (1983), pp. 1–36.

[29] ——, *Stochastic quasigradient methods*, in Ermoliev and Wets [30], pp. 143–185.

[30] Y. Ermoliev and R. J.-B. Wets, eds., *Numerical techniques for stochastic optimization*, Springer-Verlag, Berlin, 1988.

[31] F. Facchinei, A. Fischer, and C. Kanzow, *On the accurate identification of active constraints*, SIAM Journal on Optimization, 9 (1999), pp. 14–32 (electronic).

[32] A. V. Fiacco, *Introduction to Sensitivity and Stability Analysis in Nonlinear Programming*, vol. 165 of Mathematics in Science and Engineering, Academic Press, New York, 1983.

[33] R. Fletcher, *Practical Methods of Optimization*, John Wiley, Chichester and New York, 1987.

[34] R. Fourer, *Solving staircase linear programs by the simplex method, 1: Inversion*, Mathematical Programming, 23 (1982), pp. 274–313.

[35] ——, *Solving staircase linear programs by the simplex method, 1: Pricing*, Mathematical Programming, 25 (1983), pp. 251–292.

[36] K. Frauendorfer, *Stochastic two-stage programming*, Springer-Verlag, Berlin, 1992. Habilitationsschrift, University of Zürich, Zürich, 1992.

[37] A. George and J. W. H. Liu, *The evolution of the minimum degree ordering algorithm*, SIAM Review, 31 (1989), pp. 1–19.

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

[39] C. R. Glassey, *Nested decomposition and multistage linear programs*, Management Science, 20 (1973), pp. 282–292.

[40] D. Goldfarb, *Strategies for constraint deletion in active set algorithms*, in Numerical analysis (Dundee, 1985), Longman Sci. Tech., Harlow, 1986, pp. 66–81.

[41] J. Gondzio and J.-P. Vial, *Warm start and ε-subgradients in the cutting plane scheme for block-angular linear programs*, Tech. Rep. Logilab Technical Report 97.1, Section of Management Studies, University of Geneva, June 1997.

[42] N. I. M. Gould, *An algorithm for large-scale quadratic programming*, IMA Journal on Numerical Analysis, 11 (1991), pp. 299–324.

[43] N. I. M. Gould and J. K. Reid, *New crash procedures for large systems of linear constraints*, Mathematical Programming, 45 (1989), pp. 475–501.

[44] L. Grippo, F. Lampariello, and S. Lucidi, *A nonmonotone line search technique for Newton's method*, SIAM Journal on Numerical Analysis, 23 (1986), pp. 707–716.

[45] J. A. J. Hall and K. I. M. McKinnon, *An asynchronous parallel revised simplex algorithm*, Tech. Rep. MS 95-50, Department of Mathematics and Statistics, University of Edinburgh, 1995.

[46] ——, *ASYNPLEX, an asynchronous parallel revised simplex algorithm*, Annals of Operations Research, 81 (1998), pp. 27–49.

[47] J. Han and D. Sun, *Superlinear convergence of approximate Newton methods for LC^1 optimization problems without strict complementarity*, in Recent advances in nonsmooth optimization, World Scientific Publishing, River Edge, NJ, 1995, pp. 141–158.

[48] W. V. Harlow, *Asset allocation in a downside-risk framework*, Financial Analysts Journal, 47 (1991), pp. 28–40.

[49] J. L. Higle and S. Sen, *Statistical verification of optimality conditions for stochastic programs with recourse*, Annals of Operations Research, 30 (1991), pp. 215–240.

[50] ——, *Stochastic decomposition: An algorithm for two-stage linear programs with recourse*, Mathematics of Operations Research, 16 (1991), pp. 650–669.

[51] ——, *Finite master programs in regularised stochastic decomposition*, Mathematical Programming, 67 (1994), pp. 143–168.

[52] ——, *Stochastic Decomposition*, Kluwer Academic Publishers, Dordrecht, 1996. A statistical method for large scale stochastic linear programming.

[53] J.-B. Hiriart-Urruty and C. Lemaréchal, *Convex Analysis and Minimization Algorithms I and II*, Grundlehren der Mathematischen Wissenschaften 306, Springer-Verlag, Berlin, 1993.

[54] J. K. Ho and A. S. Manne, *Nested decomposition for dynamic models*, Mathematical Programming, 6 (1974), pp. 121–140.

[55] P. J. Huber, *Robust statistics*, John Wiley & Sons, Inc., New York, 1981. Wiley Series in Probability and Mathematical Statistics.

[56] K. H yland, *Asset liability management for a life insurance company*, PhD thesis, The Norwegian University of Science and Technology, 1998.

[57] K. H yland and S. W. Wallace, *Generating scenario trees for multistage problems*, to appear in Management Science.

[58] G. Infanger, *Monte Carlo (importance) sampling within a Benders decomposition algorithm for stochastic linear programs*, Annals of Operations Research, 39 (1992), pp. 69–95.

[59] G. Infanger and D. P. Morton, *Cut sharing for multistage stochastic linear programs with interstage dependency*, Mathematical Programming, 75 (1996), pp. 241–256.

[60] International Business Machines Co., *IBM Optimization Subroutine Library Guide and Reference, form number SC23-0519*, Armonk, NY, USA, 1992.

[61] P. Kall, *Computational methods for solving two-stage stochastic linear programming problems*, Zeitschrift für Angewandte Mathematik und

Physik. ZAMP. Journal of Applied Mathematics and Physics. Journal de Mathématiques et de Physiques Appliquées, 30 (1979), pp. 261–271.

[62] P. Kall and S. W. Wallace, *Stochastic Programming*, John Wiley, Chichester and New York, 1994.

[63] J. Kelley, J. E., *The cutting-plane method for solving convex programs*, Journal of the Society Industrial and Applied Mathematics, 8 (1960), pp. 703–712.

[64] A. J. King, *Asymmetric risk measures and tracking models for portfolio optimization under uncertainty*, Annals of Operatinos Research, 45 (1993), pp. 165–178.

[65] K. C. Kiwiel, *Methods of descent for nondifferentiable optimization*, Springer-Verlag, Berlin-New York, 1985.

[66] ——, *A subgradient selection method for minimizing convex functions subject to linear constraints*, Computing. Archiv für Informatik und Numerik, 39 (1987), pp. 293–305.

[67] ——, *Proximal level bundle methods for convex nondifferentiable optimization, saddle-point problems and variational inequalities*, Mathematical Programming, 69 (1995), pp. 89–109.

[68] ——, *Restricted step and Levenberg-Marquardt techniques in proximal bundle methods for nonconvex nondifferentiable optimization*, SIAM Journal on Optimization, 6 (1996), pp. 227–249.

[69] R. Kouwenberg, *Scenario generation and stochastic programming models for asset liability management*, to appear in European Journal of Operational Research.

[70] C. Lemaréchal and C. Sagastizábal, *An approach to variable metric bundle methods*, in System Modelling and Optimization (Compiègne, 1993), J. Henry and J.-P. Yvon, eds., Springer, London, 1994, pp. 144–162.

[71] R. Levkovitz and G. Mitra, *Experimental investigations in combining primal dual interior point method and simplex based LP solvers*, Annals of Operations Research, 58 (1995), pp. 19–38. Applied mathematical programming and modeling, II (APMOD 93) (Budapest, 1993).

[72] F. Louveaux, *Piecewise quadratic programs with applications to stochastic quadratic programming*, Tech. Rep. DP7713, Centre for Operations Research & Econometrics, Universite Catholique De Louvain, May 1977.

[73] ——, *A solution method for multistage stochastic programs with recourse with application to an energy investment problem*, Operations Research, 28 (1980), pp. 889–902.

[74] Z.-Q. Luo, J.-S. Pang, and D. Ralph, *Mathematical programs with equilibrium constraints*, Cambridge University Press, Cambridge, 1996.

[75] I. J. Lustig and G. Li, *An implementation of a parallel primal-dual interior point method for block-structured linear programs*, Computational Optimization and Applications., 1 (1992), pp. 141–161.

[76] I. J. Lustig, J. M. Mulvey, and T. J. Carpenter, *Formulating two-stage stochastic programs for interior point methods*, Operations Research, 39 (1991), pp. 757–771.

[77] H. M. Markowitz, *Portfolio selection*, Journal of Finance, 8 (1952), pp. 77–91.

[78] K. Marti and P. Kall, eds., *Stochastic programming*, Springer-Verlag, Berlin, 1995. Numerical techniques and engineering applications.

[79] ——, eds., *Stochastic programming methods and technical applications*, Springer-Verlag, Berlin, 1998.

[80] The MathWorks, Inc., *http://www.mathworks.com*, Natick, MA 01760-1500.

[81] S. Mehrotra, *On the implementation of a primal-dual interior point method*, SIAM Journal on Optimization, 2 (1992), pp. 575–601.

[82] R. B. Mifflin, *Semismooth and semiconvex functions in constrained optimization*, SIAM Journal on Control and Optimization, 15 (1977), pp. 957–972.

[83] ——, *A modification and extension of Lemarechal's algorithm for nonsmooth minimization*, Mathematical Programming Study, (1982), pp. 77–90. Nondifferential and variational techniques in optimization (Lexington, Ky., 1980).

[84] ——, *A quasi-second-order proximal bundle algorithm*, Mathematical Programming, 73 (1996), pp. 51–72.

[85] R. B. Mifflin, D. Sun, and L. Qi, *Quasi-Newton bundle-type methods for nondifferentiable convex optimization*, SIAM Journal on Optimization, 8 (1998), pp. 583–603 (electronic).

[86] J. J. Moré, *On the performance of algorithms for large-scale bound constrained problems*, in Large-scale numerical optimization (Ithaca, NY, 1989), SIAM, Philadelphia, PA, 1990, pp. 32–45.

[87] D. P. Morton, *An enhanced decomposition algorithm for multistage stochastic hydroelectric scheduling*, Annals of Operations Research, 64 (1996), pp. 211–235.

[88] J. M. Mulvey, *Financial planning via multi-stage stochastic programs*, in Mathmatical Programming: State of the Art 1994, J. R. Birge and K. G. Murty, eds., The University of Michigan, 1994, pp. 151–171.

[89] J. M. Mulvey and A. Ruszczyński, *A new scenario decomposition method for large-scale stochastic optimization*, Operations Research, 43 (1995), pp. 477–490.

[90] J. M. Mulvey and A. E. Thorlacius, *The Towers Perrin global capital market scenario generation system*, in Mulvey and Ziemba [92], 1998, pp. 286–312.

[91] J. M. Mulvey and H. Vladimirou, *Stochastic newtwork programming for financial planning problems*, Management Science, 38 (1992), pp. 1642–1664.

[92] J. M. Mulvey and W. T. Ziemba, eds., *Worldwide Asset and Liability Modeling*, Cambridge University Press, Cambridge, UK, 1998.

[93] S. S. Nielsen and S. A. Zenios, *Scalable parallel Benders decomposition for stochastic linear programming*, Parallel Computing, 23 (1997), pp. 1069–1088.

[94] J.-S. Pang, S.-P. Han, and N. Rangaraj, *Minimization of locally Lipschitzian functions*, SIAM Journal on Optimization, 1 (1991), pp. 57–82.

[95] J.-S. Pang and L. Qi, *Nonsmooth equations: motivation and algorithms*, SIAM Journal on Control and Optimization, 3 (1993), pp. 443–465.

[96] ——, *A globally convergent Newton method for convex SC^1 minimization problems*, Journal of Optimiaztion Theory and Applications, 85 (1995), pp. 633–648.

[97] L. Qi, *Convergence analysis of some algorithms for solving nonsmooth equations*, Mathematics of Operations Research, 18 (1993), pp. 227–244.

[98] ——, *Superlinearly convergent approximate Newton methods for LC^1 optimization problems*, Mathematical Programming, 64 (1994), pp. 277–294.

[99] L. Qi and J. Sun, *A nonsmooth version of Newton's method*, Mathematical Programming, 58 (1993), pp. 353–367.

[100] ——, *A trust region algorithm for minimization of locally Lipschitzian functions*, Mathematical Programming, 66 (1994), pp. 25–43.

[101] R. T. Rockafellar, *Convex Analysis*, Princeton University Press, Princeton, New Jersey, 1970.

[102] R. T. Rockafellar and R. J.-B. Wets, *A Lagrangian finite generation technique for solving linear-quadratic problems in stochastic programming*, Mathematical Programming Study, 28 (1986), pp. 63–93.

[103] ——, *Scenario and policy aggregation in optimization under uncertainty*, Mathematics of Operations Research, 16 (1991), pp. 119–47.

[104] C. H. Rosa and A. Ruszczyński, *On augmented Lagrangian decomposition methods for multistage stochastic programs*, Annals of Operations Research, 64 (1996), pp. 289–309.

[105] S. Ross, *Introduction to Stochastic Dynamic Programming*, Academic Press, New York and London, 1983.

[106] A. Ruszczyński, *A regularized decomposition method for minimizing a sum of polyhedral functions*, Mathematics of Operations Research, 35 (1986), pp. 309–333.

[107] ——, *Parallel decomposition of multistage stochastic programming problems.*, Mathematical Programming, 58A (1993), pp. 201–228.

[108] ——, *Regularized decomposition for stochastic programs: algorithmic techniques and numerical results*, Tech. Rep. WP-93-21, IIASA, 1993.

[109] H. Schramm and J. Zowe, *A version of the bundle idea for minimizing a nonsmooth function: conceptual idea, convergence analysis, numerical results*, SIAM Journal on Optimization, 2 (1992), pp. 121–152.

[110] K. Shimizu, Y. Ishizuka, and J. F. Bard, *Nondifferentiable and two-level mathematical programming*, Kluwer Academic Publishers, Boston, MA, 1997.

[111] N. Z. Shor, *Minimization methods for nondifferentiable functions*, Springer-Verlag, Berlin-New York, 1985. Translated from the Russian by K. C. Kiwiel and A. Ruszczyński.

[112] W. Shu and M. Wu, *Sparse implementation of revised simplex algorithm on parallel computers*, in Proceedings of Sixth SIAM conference on Parallel Processing for Scientific Computing, 1993, pp. 501–509.

[113] M. J. Sims, *Use of a stochastic capacity planning model to find the optimal level of flexibility for a manufacturing system.* Senior Design Project, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, 1992.

[114] *Stochastic programming.*
http://www-personal.umich.edu/~jrbirge/dholmes/main.html.

[115] *Stochastic programming ftp archive.*
http://www.jims.cam.ac.uk/research/papers/frg/stoch.htm.

[116] B. Strazicky, *Some results concerning an algorithm for the discrete recourse problem*, in Stochastic Programming, M. A. H. Dempster, ed., Academic Press, 1980, pp. 263–274.

[117] J. Sun, *On Monotropic Piecewise Quadratic Programming*, PhD thesis, Department of Applied Mathematics, University of Washington, 1986.

[118] ——, *A study on monotropic piecewise quadratic programming*, in Recent Developments in Mathematical Programming, S. Kumar, ed., Gordon and Breach Science Publishers, 1991, pp. 213–235.

[119] ——, *On piecewise quadratic Newton and trust region problems*, Mathematical Programming, 76 (1997), pp. 451–467.

[120] P. L. Toint, *An assessment of nonmonotone linesearch techniques for unconstrained optimization*, SIAM Journal on Scientific Computing, 17 (1996), pp. 725–739.

[121] R. Van Slyke and R. J.-B. Wets, *L-shpaed linear programs with application to optimal control and stochastic programming*, SIAM Journal on Applied Mathematics, 17 (1969), pp. 638–663.

[122] R. Vanderbei, *LOQO User's Manual – Version 3.03*, Princeton, 1997. Statistics and Operations Research Technical Report No. SOR-97-07.

[123] R. J. Vanderbei, *Symmetric quasidefinite matrices*, SIAM Journal on Optimization, 5 (1995), pp. 100–113.

[124] ——, *Linear programming: foundations and extensions*, Kluwer Academic Publishers, Boston, MA, 1996.

[125] J. Von Neumann and O. Morgenstern, *Theory of games and economic behavior*, Princeton University Press, Princeton, 1953.

[126] S. W. Wallace, J. Higle, and S. Sen, eds., *Stochastic programming, algorithms and models*, Baltzer Science Publishers BV, Amsterdam, 1996. Papers from the IFIP Workshop held in Lillehammer, January 1994, Ann. Oper. Res. **64** (1996).

[127] M. H. Wright, *Identifiable surfaces in constrained optimiazation*, SIAM Journal on Control and Optimization, 31 (1993), pp. 1063–1079.

[128] ——, *Primal-dual interior-point methods*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.

[129] D. Yang and S. A. Zenios, *A scalable parallel interior point algorithm for stochastic linear programming and robust optimization*, Computational Optimization and Applications, 7 (1997), pp. 143–158. Computational issues in high performance software for nonlinear optimization (Capri, 1995).

[130] Y. Ye, *Interior Point Algorithms: Theory and Analysis*, John Wiley, Chichester and New York, 1997.

[131] S. A. Zenios, *Asset and Liability Management under Uncertaunty for Fixed Income Securities*, in Mulvey and Ziemba [92], 1998, pp. 537–557.

Index

active set method, 26–28
aggregate linearization error, 102
aggregate subgradient, 35, 102, 106

B-derivative, 31
bunching, 46
bundle method
    linearization error, 33
bundle method, convex nonsmooth program, 32–36

chance constrained program, 5
complete recourse, 7
conjugate function, 30
constraint qualification, 68
convex nonsmooth program, 29–40
    bundle method, 32–36
    subgradient method, 32

degeneracy, MQSP, 74, 131
diagonal quadratic algorithm, 60–63
directional derivative, 30
dual decomposition, 20, 58–66
    diagonal quadratic algorithm, 60–63
    Lagrangian finite generation method, 59–60
    progressive hedging algorithm, 64–65

expected value tree, MQSP, 127

feasibility cut, 111, 129
    L-shaped method, 43
finite convergence, MQSP, 110, 119
fixed recourse, 6, 41

generalized Hessian, 31
generalized Hessian, MQSP, 79, 83–86, 89–92
geometric mean, 132
global convergence
    convex MQSP, 111
    strictly convex, LC^1 MQSP, 108

importance sampling, 51–52
interior point method
    linear program, 22–25
    quadratic program, 28–29

KKT condition, 68

L-shaped method, 41–47
    feasibility cut, 43
    optimality cut, 44
Lagrangian finite generation method, 59–60
LC^1, 31
linear independence CQ, 69
linear program, 20–25
    interior point method, 22–25
    simplex method, 20–22
linearization error, 101
    bundle method, 33
linesearch, 121–124
Lipschitz function, 31

Mangasarian-Fromovitz CQ, 68
MQSP
    algorithm
        convex Lipschitz problem, 101–106
        strictly convex LC^1 problem, 96–99
        with feasibility cut, 99–101
    degeneracy, 74, 131
    equivalent QP, 16
    expected value tree, 127
    finite convergence, 110, 119

formulation, 15
generalized Hessian, 79, 83–86, 89–92
global convergence, 108, 111
presolve strategies, 127–129, 132
properties, 78–93
small presolve tree, 128
subgradient, 79
superlinear convergence, 111, 119
multistage quadratic stochastic programs, *see* MQSP

nested decomposition, 47–48
non-anticipativity constraint, 58
null step, 102

optimality cut, L-shaped method, 44

piecewise quadratic form of L-shaped method, 52
piecewise quadratic function, 73
piecewise quadratic program, 74
presolve strategies, MQSP, 127–129, 132
primal decomposition, 19, 40–58
importance sampling, 51–52
L-shaped method, 41–47
nested decomposition, 47–48
piecewise quadratic form of L-shaped method, 52
regularized decomposition, 49–51
stochastic decomposition, 51–52
two stage quadratic SP, 55
probabilistically constrained program, 5
progressive hedging algorithm, 64–65
proximal term, 49

quadratic program, 25–29
active set method, 26–28
interior point method, 28–29

random variables, 2
recourse
fixed, 6
relatively complete, 7
simple, 7
recourse program, 5–7
regularized decomposition, 49–51
relatively complete recourse, 7
risk
asymmetric risk, 10
linear-quadratic risk measure, 11
lower partial moments, 10
standard deviation, 8

scenario tree, 3–4
second order sufficiency condition, 70
semismooth function, 31
sensitivity analysis, 68–72
serious step, 102
simple recourse, 7
simplex method, 20–22
Slater condition, 69
small presolve tree, MQSP, 128
stochastic decomposition, 51–52
stochastic linear program, 7–8
stochastic quasigradient, 65
strict complementarity condition, 70
strong Slater condition, 69
subdifferential, 30
subgradient method, convex nonsmooth program, 32
superlinear convergence, MQSP, 111, 119

trust region method, 124–127, 132
two stage quadratic SP, 55