General Solution Methods for Mixed Integer Quadratic Programming and Derivative Free Mixed Integer Non-Linear Programming Problems

Eric Newby

School of Computational and Applied Mathematics,
University of the Witwatersrand,
Johannesburg.

April 27, 2013

Declaration

I declare that this project is my own, unaided work. It is being submitted to the Faculty of Science, University of the Witwatersrand, in fulfilment of the requirements for the degree of Doctor of Philosophy. It has not been submitted before for any degree or examination in any other University.

Eric Newby
April 27, 2013

I dedicate this thesis to Siegfried Maske

Abstract

In a number of situations the derivative of the objective function of an optimization problem is not available. This thesis presents a novel algorithm for solving mixed integer programs when this is the case. The algorithm is the first developed for problems of this type which uses a trust region methodology. Three implementations of the algorithm are developed and deterministic proofs of convergence to local minima are provided for two of the implementations.

In the development of the algorithm several other contributions are made. The derivative free algorithm requires the solution of several mixed integer quadratic programming subproblems and novel methods for solving non-convex instances of these problems are developed in this thesis. Additionally, it is shown that the current definitions of local minima for mixed integer programs are deficient and a rigorous approach to developing possible definitions is proposed. Using this approach we propose a new definition which improves on those currently used in the literature.

Other components of this thesis are an overview of derivative based mixed integer non-linear programming, extensive reviews of mixed integer quadratic programming and deterministic derivative free optimization and extensive computational results illustrating the effectiveness of the contributions mentioned in the previous paragraphs.

Acknowledgements

I express my deepest gratitude to everyone who helped me complete this thesis. Firstly, I would like to thank my supervisor Prof. Montaz Ali, who was constantly available for guidance, motivation and helpful comments even when he was on sabbatical. I also thank my parents and my brother for the constant support they provided and for the many optimization jokes they came up with to keep me light hearted. Additionally, I would like to thank the National Research Foundation for providing funding during the duration of my thesis. I am also grateful to all of my friends, both in the CAM department and outside of it, for keeping me amused and entertained and supplying encouragement and support where it was needed. Finally, I would like to thank Adéle Rossouw for making the final months of writing this thesis far more enjoyable than they would otherwise have been.

Contents

Nomenclature **xiii**

1 Introduction **1**
1.1 Problem overview 1
1.2 Motivation 2
1.3 Structure of the thesis 4

2 Review of Mixed Integer Non-Linear Programming **5**
2.1 Convex MINLPs 5
2.1.1 Branch and Bound 6
2.1.2 Outer Approximation 7
2.1.3 Generalised Benders Decomposition 8
2.1.4 Extended Cutting Plane 8
2.1.5 Branch and Cut 8
2.1.6 Disjunctive Programming 9
2.2 Non-convex MINLPs 10
2.2.1 The $\alpha - BB$ method 10
2.2.2 BARON 11
2.2.3 Couenne 12
2.2.4 SCIP 13
2.2.5 Global optimization of non-convex GDP 14
2.2.6 Suitability of traditional MINLP methods 14
2.3 Mixed integer quadratic programming 15
2.3.1 Convex mixed integer quadratic programming 15
2.3.2 Non-convex mixed integer quadratic programming . . 16
2.4 Conclusion 23

3 Review of Derivative Free Programming **24**
3.1 Introduction to derivative free optimization 24
3.2 Derivative free optimization of continuous problems 25
3.2.1 Metaheuristics 25

3.2.2 Surrogate based optimization . . . 27
3.2.3 Local sampling methods . . . 28
3.3 The BOBYQA algorithm . . . 34
3.3.1 Overview of BOBYQA . . . 34
3.3.2 The initial quadratic model . . . 35
3.3.3 Updating the quadratic model . . . 38
3.3.4 Choosing the direction vector . . . 44
3.3.5 The RESCUE procedure . . . 51
3.3.6 Further details of BOBYQA . . . 55
3.4 Derivative free optimization of mixed integer problems . . . 60
3.4.1 Metaheuristics and surrogate optimization . . . 60
3.4.2 Local sampling . . . 62
3.5 Conclusion . . . 64

4 Methods for Solving Non-Convex MIQPs 65
4.1 The linear transformation . . . 66
4.2 Approach used when H_{cc} is invertible . . . 67
4.2.1 Choosing the elements of V . . . 67
4.2.2 The Branch and Bound Procedure . . . 74
4.3 Approach used when H_{cc} is positive definite . . . 77
4.3.1 MIQCR . . . 77
4.3.2 MIQTCR . . . 81
4.3.3 MIQTBC . . . 84
4.4 Approach used when H_{cc} is singular . . . 91
4.5 Conclusion . . . 95

5 Derivative Free Methods for MINLPs 96
5.1 Local minima of mixed integer programs . . . 96
5.2 Mixed integer algorithm . . . 104
5.2.1 Initialisation . . . 104
5.2.2 Trust Region and Alternative Iterations . . . 107
5.2.3 Further details on the implementation of HEMBOQA 108
5.2.4 SEMBOQA and COMBOQA . . . 115
5.3 Conclusion . . . 119

6 Computational Results and Discussion 120
6.1 Mixed integer quadratic programming . . . 120
6.1.1 Results obtained with H_{cc} invertible . . . 121
6.1.2 Results obtained with H_{cc} positive definite . . . 136
6.1.3 Results obtained with H_{cc} singular . . . 147
6.2 Mixed integer non-linear programming . . . 148

6.3 Conclusion 160

7 Conclusion **161**
7.1 Summary 161
7.2 Future Work 163

A Pseudocode for the HEMBOQA Algorithm **166**

B Results for the derivative free MINLP solvers **170**

C Test Problems **184**
C.1 Method used to generate MIQPs 184
C.1.1 Generation of the objective function 184
C.1.2 Generation of the constraints 185
C.2 MINLP test functions 186

Bibliography **195**

List of Figures

3.1 A flow chart showing the processes followed to choose between trust region and alternative iterations and to find ρ_{k+1}. . . . 49

5.1 A figure illustrating important features of the objective function of problem (5.6). 103

6.1 Performance profile examining the effectiveness of the transformation when solving problems with $n_c = n_d$ using SCIP. 124

6.2 Performance profile examining the effectiveness of the transformation when solving problems with $n_c > n_d$ using SCIP. 125

6.3 Performance profile examining the effectiveness of the transformation when solving problems with $n_c < n_d$ using SCIP. 126

6.4 Performance profile examining the effect of the choice of U_{dd} when solving problems with $n_c = n_d$ using SCIP. Bad U_{dd} denotes the upper triangular choice of U_{dd}. 128

6.5 Performance profile examining the effectiveness of the transformation when solving problems with $n_c = n_d$ using BARON. 130

6.6 Performance profile examining the effectiveness of the transformation when solving problems with $n_c > n_d$ using BARON. 131

6.7 Performance profile examining the effectiveness of the transformation when solving problems with $n_c < n_d$ using BARON. 132

6.8 Performance profile examining the effectiveness of the transformation when solving problems with $n_c = n_d$ and H_{cc} invertible using Algorithm 2. 133

6.9 Performance profile examining the effectiveness of the transformation when solving problems with $n_c > n_d$ and H_{cc} invertible using Algorithm 2. 134
6.10 Performance profile examining the effectiveness of the transformation when solving problems with $n_c < n_d$ and H_{cc} invertible using Algorithm 2. 135
6.11 Performance profile examining the effectiveness of the reformulations schemes MIQCR and MIQTCR when $n_c = n_d$. . . 138
6.12 Performance profile examining the effectiveness of the reformulations schemes MIQCR and MIQTCR when $n_c > n_d$. . . 140
6.13 Performance profile examining the effectiveness of the reformulations schemes MIQCR and MIQTCR when $n_c < n_d$. . . 141
6.14 Performance profile comparing the solution times of the reformulated problem and the original problem for $n \leq 6$. . . . 143
6.15 Performance profile comparing the solution times of the reformulated problem and the original problem for $n \geq 8$. . . . 144
6.16 Performance profile examining the effectiveness of the transformation when solving problems with $n_c = n_d$ and H_{cc} singular using Algorithm 2. 149
6.17 Performance profile examining the effectiveness of the transformation when solving problems with $n_c > n_d$ and H_{cc} singular using Algorithm 2. 150
6.18 Performance profile examining the effectiveness of the transformation when solving problems with $n_c < n_d$ and H_{cc} singular using Algorithm 2. 151
6.19 Performance profile comparing the derivative free algorithms using number of function evaluations as a performance measure. 155
6.20 Cumulative probability distribution of the m-fold improvement of the derivative free algorithms. 156
6.21 A plot showing the effect of increasing n on the average number of function evaluations required by HEMBOQA to solve the test problems. 158
6.22 A plot showing the effect of increasing n on the average number of function evaluations required by SEMBOQA to solve the test problems. 158
6.23 A plot showing the effect of increasing n on the average number of function evaluations required by COMBOQA to solve the test problems. 159

List of Tables

6.1 The results obtained when solving problem (1.2) and problem (4.27) with constraints of type 1, $n_c = n_d$ and H_{cc} invertible using Algorithm 2. The number of unsolved problems is denoted by us. . . . 136
6.2 The average relaxation gap and number of variables for problems with constraints of type 2 when n_c is varied with $n = 8$. The ratio in the fourth column is the relaxation gap of MIQCR over that of MIQTCR. The ratio in the seventh column is the number of variables used by MIQTCR over the number used by MIQCR. . . . 139
6.3 The results obtained when solving problems with constraints of type 1, $n_c = n_d$ and H_{cc} positive definite. The reformulated problems are solved using CPLEX. The number of unsolved problems is denoted by us. . . . 145
6.4 The time taken to solve problems with constraints of type 1 with H_{cc} positive definite using Couenne. . . . 146
6.5 The time taken to solve problems with constraints of type 2 with H_{cc} positive definite using Couenne. . . . 146
6.6 The time taken to solve problems with constraints of type 3 with H_{cc} positive definite using Couenne. . . . 147
6.7 The results obtained when solving problem (1.2) and problem (4.27) with constraints of type 1, $n_c = n_d$ and H_{cc} singular using Algorithm 2. The number of unsolved problems is denoted by us. . . . 152
6.8 The test functions used to examine the effectiveness of the derivative free MINLP algorithms. . . . 153
6.9 Rankings of the four derivative free algorithms using the objective function value and the number of function evaluations as performance criteria. . . . 156
6.10 Percentage of test problems for which the global minimum was found. . . . 160

B.1 Results for $n = 2$ 173
B.2 Results for $n = 4$ 176
B.3 Results for $n = 6$ 178
B.4 Results for $n = 8$ 180
B.5 Results for $n = 10$ 183

List of Algorithms

1 Generation of the orthogonal matrix used to update Z_k . . . 44
2 The Branch and Bound algorithm 76
3 The U_{dd} selection algorithm for MIQTBC 88
4 The U_{cc} selection algorithm for singular H_{cc} 94
5 General structure of the deterministic implementations of the derivative free algorithm 116
6 Generation of Hessians with H_{cc} invertible 185
7 Generation of Hessians with H_{cc} positive definite 192
8 Generation of Hessians with H_{cc} singular 192
9 Generation of sparse linear inequality constraints 193
10 Generation of dense linear inequality constraints 194

Nomenclature

A large amount of notation is defined in this thesis. We only give descriptions of recurring notation in this list. Notation is generally taken to be recurring if it occurs on more than one page. The acronyms and mathematical symbols are listed separately.

Abbreviations

$\alpha - BB$ α Branch and Bound 10

BARON Branch and reduce optimization navigator 11

BOBYQA Bound optimization by quadratic approximation 3

COMBOQA Combined mixed bound constrained optimization by quadratic approximation 104

Couenne Convex over and under envelopes for nonlinear estimation 12

GBD Generalised Benders decomposition 8

GDP Generalised disjunctive program 9

GPS Generalised pattern search 30

HEMBOQA Heuristic mixed bound constrained optimization by quadratic approximation 104

LP Linear program 13

MADS Mesh adaptive direct search 31

MILP Mixed integer linear program 6

MINLP Mixed integer non-linear program 1

MIQCP Mixed integer quadratically constrained program 14

MIQCR Mixed integer quadratic convex reformulation 77

MIQP Mixed integer quadratic program 2

MIQTBC Mixed integer quadratic transformation and bilinear convexification 77

MIQTCR Mixed integer quadratic transformation and convex reformulation 77

MP Master problem 7

NLP Non-linear program 7

NOMAD Nonsmooth optimization by mesh adaptive direct search 62

SCIP Solving constraint integer programs 13

SEMBOQA Separate mixed bound constrained optimization by quadratic approximation 104

Symbols

$\mathcal{X}_i$ The feasible continuous manifold obtained by fixing the integer variables to have the values of y_i^d 108

α_i Non-zero real number used to generate interpolation points from x_0 36

α_k The step size used by the direct search algorithms 29

β_i Non-zero real number used to generate interpolation points from x_0 36

β_{ij} Coefficient of a term used to perturb the objective function in MIQCR and MIQTCR 78

Δ_k Outer trust region radius or trust region vector used by the trust region algorithms 34

δ_k The greatest distance between x_k and an interpolation point . . 50

δ_{it} The Kronecker delta 46

γ_{in} The number of interpolation points on $\mathcal{X}_s$ 109

γ_{req} The number of interpolation points required to be on $\mathcal{X}_s$ 109

γ_i The possible replacement interpolation points generated by RESCUE 52

$\hat{\mathcal{L}}$ Feasible region of the linear relaxation of problem (2.7) 17

$\hat{\Omega}$ The feasible region of problem (2.6), the lifted MIQP 17

$\hat{x}_i$ The branching point 6

κ_{ij} Additional variable introduced by the convex envelope of $x_i x_j$. 22

Λ The matrix $\Lambda = H_{dd} - H_{cd}^T H_{cc}^{-1} H_{cd}$ which is used in MIQTBC . 84

λ The m unknowns in the linear system used to update Q which are used to update the Hessian 37

$\Lambda_t(x)$ Quadratic function satisfying the Lagrange interpolation conditions 46

$\langle H, X \rangle$ Trace(H, X) 17

$\mathcal{D}$ The set of directions used by the direct search algorithms 29

$\mathcal{N}_{\text{comb}}(x)$ The neighbourhood used in the definition of a combined local minimum 100

$\mathcal{N}_d(x_d)$ A discrete, user defined neighbourhood 97

$\mathcal{N}_m(x)$ A neighbourhood used in the definition of a local minimum of a mixed integer problem 97

$\mathcal{N}_r(x)$ A commonly used subset of $\mathcal{N}_m(x)$ 97

$\mathcal{S}^n$ Space of real, symmetric matrices of order n 2

$\mathcal{T}$ The set of integers such that the new point generated by RESCUE $\widehat{y}_t$, $t \in \mathcal{T}$ is not in the original set $\{y_i : i = 1, \ldots, m\}$ 54

μ Parameter used in generation of U_{cc} for singular H_{cc} 93

μ_j^k The multipliers used in the representation of $\nabla^2 Q_k$ 40

ν Parameter used in the generation of U_{cc} for singular H_{cc} 93

ω Parameter used in the generation of U_{cc} for singular H_{cc} 93

Ω_c The feasible region of the MINLP when $n_d = 0$ 97

Ω_d The feasible region of the MINLP when $n_c = 0$ 97

Ω_k A submatrix of W_k 37

ω_k A measure of the distance between interpolation points on the manifold $\mathcal{X}_s$ 111

Ω_m The feasible region of the general MINLP 97

Ω_q The feasible region of problem (1.2) 17

ϕ Scalar used in the updating procedure for W_k 42

ψ Scalar used in the updating procedure for W_k 42

ρ_1 The initial inner trust region radius 35

ρ_{end} The final inner trust region radius 35

ρ_k Inner trust region radius or trust region vector used by the trust region algorithms 35

σ Scalar used in the updating procedure for W_k 42

τ Scalar used in the updating procedure for W_k 42

all$(h(a,b))$ all$(h(a,b))$ is true if $h(a_i, b_i)$ holds for all i 112

any$(h(a,b))$ any$(h(a,b))$ is true if $h(a_i, b_i)$ holds for any i 112

clconv$(\cdot)$ Closed convex hull 17

$\Theta^{(1)}$ One of two terms into which Θ is split when H_{cc} is singular ... 91

$\Theta^{(1)}_{cc}$ A submatrix of $\Theta^{(1)}$ 91

$\Theta^{(1)}_{dd}$ A submatrix of $\Theta^{(1)}$ 91

$\Theta^{(2)}$ One of two terms into which Θ is split when H_{cc} is singular ... 91

Θ_{cc} A submatrix of the Hessian of the transformed MIQP 73

Θ_{dd} A submatrix of the Hessian of the transformed MIQP 73

$\| \cdot \|_F$ The Frobenius norm 33

$\widehat{y}_i$ The new set of interpolation points generated by the trust region algorithms 35

Ξ_k	A submatrix of W_k 37
A	Matrix describing the linear inequality constraints, $Ax \leq b$ 2
a	Number of linear inequality constraints 2
b	Vector describing the linear inequality constraints, $Ax \leq b$ 2
$B_\varepsilon(x_c)$	The open ball $\{y_c \in \mathbb{R}^{n_c} : \|y_c - x_c\| < \varepsilon\}$ 97
c_i	Non-linear constraints of the general MINLP 6
D	Matrix describing the linear equality constraints, $Dx = e$ 2
d_k	The direction vector generated during the kth iteration of the trust region algorithms 35
E	The set of indices $\{(i,k) : i \in I,\ k = 0, \ldots, \lfloor \log_2(u_i) \rfloor\}$ 78
e	Vector describing the linear equality constraints, $Dx = e$ 2
e_i	The ith unit vector 36
F	$F = \nabla^2 Q_{k+1} - \nabla^2 Q_k$ 38
f	Non-linear objective function 2
g	Gradient of the quadratic objective function h 2
G_k	The matrix used in the representation of $\nabla^2 Q_k$ 40
H	Hessian of the quadratic function h 2
h	Quadratic objective function 2
H_{cc}	A submatrix of the Hessian of a quadratic objective function ..66
H_{cd}	A submatrix of the Hessian of a quadratic objective function ..66
H_{dd}	A submatrix of the Hessian of a quadratic objective function ..66
h_{jk}	Constraints in the disjunctions of a GDP 9
I	The set of indices $\{n_c + 1, n_c + 2, \ldots, n\}$ 68
I_n	The $n \times n$ identity matrix 30
J	The set of indices $\{1, 2, \ldots, n_c\}$ 68

K The set of integers $\{1, 2, \ldots, m\}$ 34

l Vector of lower bounds 2

looplength The number of previous interpolation points which are stored by the trust region algorithms 110

m The number of interpolation points used by the trust region algorithms 34

M_k The mesh of points used by the direct search algorithms 29

maxevals The maximum number of function evaluations allowed in HEMBOQA 104

maxtime The maximum time HEMBOQA is allowed to run 104

n Number of decision variables 2

n_{cons} The number of non-linear constraints in the general MINLP . . . 6

$n_{\max}$ The maximum number of function evaluations allowed in SEMBOQA and COMBOQA 115

n_c Number of continuous decision variables 2

n_d Number of discrete decision variables 2

P The set of indices $\{(i, j) \in I^2 \cup (I \times J) \cup (J \times I)\}$ 78

p Number of linear equality constraints 2

PSD The region defined by (2.9) 18

q The n unknowns in the linear system used to update Q which are used to update the gradient 37

Q_k The quadratic model in the kth iterations of the trust region algorithms 34

q_k The gradient of Q_k in the trust region algorithms 41

r The non-zero elements on the right hand side of the linear system used to update Q 37

R_2 The region defined by the rank-2 linear inequalities 17

r_k Quantity used to measure the quality of a quadratic model ... 50

s The integer in $\{1, 2, \ldots, m\}$ such that $y_s = x_k$ ... 43

t Integer such that y_t is the interpolation point discarded by the trust region algorithms ... 35

$t_{\max}$ The maximum time that SEMBOQA and COMBOQA are allowed to run ... 115

t_{ik} Binary variables used in MIQCR, MIQTCR and MIQTBC ... 79

u Vector of upper bounds ... 2

U_{cc} A submatrix of the transformation matrix V ... 66

U_{cd} A submatrix of the transformation matrix V ... 66

U_{dd} A submatrix of the transformation matrix V ... 66

V The matrix used in the linear transformation of the MIQPs. ... 66

v Vector used in the updating procedure for W_k ... 43

W_k The matrix used to update the quadratic model in the trust region algorithms ... 37

w_{ij} Variables used in MIQCR and MIQTCR with $w_{ij} = x_i x_j$ ... 78

x Vector of decision variables ... 2

x_0 The user supplied initial point ... 35

x_c Vector of continuous decision variables ... 2

x_d Vector of discrete decision variables ... 2

x_k The point with the lowest function value that has been found by the kth iteration of an algorithm ... 29

x_n The point generated by SEMBOQA and COMBOQA when x_k is not a minimum ... 116

X_{ij} Auxiliary variable satisfying $X_{ij} = x_i x_j$ ... 16

y^L The lower bounds on the variables in the transformed MIQP ... 69

y^U The upper bounds on the variables in the transformed MIQP ... 69

y_δ The interpolation which is furthest from x_k 111

y_i The interpolation points used by the trust region algorithms . . 34

Z A factorisation of Ω such that $\Omega = ZZ^T$ 38

z Vector used in the updating procedure for W_k 42

z_{ijk} Variables used in MIQCR and MIQTCR with $z_{ijk} = t_{ik}x_j$ 79

Chapter 1

Introduction

This chapter is intended to give a brief introduction to the work done in this thesis. Section 1.1 contains a description of the problems considered. In section 1.2 we list applications of the problems described in section 1.1 and a give brief summary of the current literature related to the solution of these problems. Finally we describe the gaps in the literature that will be filled by the approaches developed in this thesis. Section 1.3 gives an outline of the structure of the remainder of the thesis.

1.1 Problem overview

Optimization problems involve the minimization of an objective function subject to some set of constraints. One of the most general classes of optimization problems are the mixed integer non-linear programs (MINLPs). MINLPs are problems whose objective functions and constraints can be non-linear and which involve both continuous and integer variables. A large number of real world problems from many different disciplines can be formulated as MINLPs [120]. Accordingly, a large amount of work has been done on developing both local and global solution approaches for this type of problem [67]. Both heuristic and deterministic approaches have been developed, in this work we will mainly concern ourselves with the deterministic case. Almost all of these approaches rely on knowledge of the derivative of the objective function [67]. However, there are a number of problems in which the required derivatives may be unavailable. These problems all have one or more of the following characteristics; the objective function is expensive, the objective function is noisy or the objective function is obtained using a black box simulation [46]. Problems of this type can only be solved using derivative free optimization methods. The development of a deterministic

derivative free algorithm for solving MINLPs is the main aim of this thesis. However, rather than considering general MINLPs, as described above, we shall consider the simpler class of bound constrained MINLPs. As is clear from the name, the variables in bound constrained MINLPs may only be restricted by upper and lower bounds rather than the general non-linear constraints described above. The problem described above can be formulated quantitatively as follows:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & l \leq x \leq u, \\ & x = \left[x_c^T, x_d^T\right]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}, \end{aligned} \tag{1.1}$$

where $f(x)$ is some non-linear function of x, n_d is the number of discrete integer variables and n_c is the number of continuous variables. Problem (1.1) must be solved without using the derivative of f.

A second problem considered in this thesis involves the solution of another subclass of general MINLPs, namely mixed integer quadratic programs (MIQPs). A MIQP is an optimization problem with a quadratic objective function, linear constraints and both continuous and integer variables. Obviously a problem with these characteristics can be formulated mathematically as follows:

$$\begin{aligned} \min_{x} \quad & h(x) = \frac{1}{2} x^T H x + g^T x \\ \text{s.t.} \quad & Ax \leq b, \\ & Dx = e, \\ & l \leq x \leq u, \\ & x = \left(x_c^T, x_d^T\right)^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}, \end{aligned} \tag{1.2}$$

where $H \in \mathcal{S}^n$ (space of real, symmetric matrices of order n), $g \in \mathbb{R}^n$, $A \in \mathbb{R}^{(a,n)}$, $b \in \mathbb{R}^a$, $D \in \mathbb{R}^{(p,n)}$ and $e \in \mathbb{R}^p$. Mature solution approaches have been developed for convex MIQPs and commercial software packages are available for solving these problems. Far less work has been done on solving the non-convex version of problem (1.2) and the non-convex case is the focus of the work done in this thesis.

1.2 Motivation

Derivative free optimization procedures for continuous problems have advanced considerably in the past two decades. This is due to a dramatic

increase in the number of applications requiring derivative free methods [46]. Very little work has been done on extending these approaches to the mixed integer case. The approaches which have been developed are all based on extending continuous direct search procedures to the mixed integer case. One concern about the implementation of these algorithms is that they consider the continuous and discrete variables separately, this may cause then to ignore information that could prove useful in the minimization of f. In addition they all follow a similar basic form so the effect of using different algorithmic structures has not been well explored for derivative free MINLPs. It is in an attempt to overcome these problems that we develop a novel derivative free mixed integer algorithm by extending the trust region based, continuous, derivative free algorithm BOBYQA (Bound Optimization BY Quadratic Approximation) [126].

The characteristics of problems requiring the use of derivative free optimization methods were listed in section 1.1. Problems with these characteristics arise in engineering, finance, industrial design, dynamic pricing, medical imaging, groundwater supply problems and operations research, among others [46, 68, 71, 72]. These problem characteristics occur most commonly in combination with mixed variables in engineering applications [17, 71]. Most of the real world derivative free, mixed integer problems have more complicated constraints than the bound constraints considered here. It is hoped that the bound constrained method proposed in this thesis will lead to the development of trust region methods capable of handling more complicated constraints. A number of methods have been developed for continuous problems which allow complicated constraints to be handled by methods originally developed for bound constrained problems. These approaches include augmented Lagrangians [53, 96], inexact restoration [39], extreme barrier approaches [20], filter methods [18] and auxiliary functions [168]. A modification of any of these methods to the mixed integer case would allow our method to be used in solving real world problems with complicated constraints.

We now summarise the reasons for our focus on the development of solution methods for non-convex MIQPs. The main reason is that a number of MIQPs need to be solved as sub-problems during our derivative free algorithm for solving problem (1.1). MIQPs are also important in their own right, having a number of other applications including the mean variance problem in portfolio optimization [60], model predictive control [22], hybrid systems control [139], the optimal sizing and citing of substations in a network routing problem [57] and chaotic mapping of complete multipartite graphs [33]. As was noted in section 1.1 the algorithms for solving problem (1.2) when H is positive semidefinite are relatively mature and commercial

software is available to solve these problems. On the other hand, relatively little work considers the non-convex version of problem (1.2). Most of the work that has been done involves the development of convex relaxations and valid inequalities. Solution methods for special cases of problem (1.2) have been developed in [33, 38, 40, 113, 114]. None of these methods can solve the most general form of problem (1.2). However, the most general form of problem (1.2) can be solved using general MINLP solvers such as BARON [135], the αBB method [8] and Couenne [28]. Problem (1.2) can also be solved using a recently developed modification of the MINLP solver SCIP [30]. We propose modifications to these existing general methods which allow us to solve problem (1.2) more efficiently then the unmodified general methods would allow.

More specifically problem (1.2) is simplified by using carefully chosen linear transformations to reduce the number of bilinear terms in the objective function. As will be explained in chapter 2, each bilinear term requires additional variables and constraints to be added to the problem during the solution process which increases the time taken to solve the problem. The reduction in the number of bilinear terms is especially large when the problem contains more continuous variables than integer variables; it is in this case that our methods are most effective.

1.3 Structure of the thesis

The remainder of this thesis is structured as follows. In chapter 2 we give a summary of the most common methods used for solving MINLPs as well as a review of the current methods available for solving MIQPs. In chapter 3 we give a review of derivative free optimization methods. The review covers the most common methods used to solve continuous problems and the current state of mixed integer, derivative free methods with an emphasis on the direct search procedures mentioned in section 1.2. Chapter 3 also contains a detailed description of BOBYQA. In chapter 4 we develop methods for solving problem (1.2). Chapter 5 contains the description of the derivative free algorithm for solving problem (1.1). Computational results illustrating the effectiveness of the methods developed in chapters 4 and 5 are given in chapter 6. Chapter 7 contains concluding remarks and a discussion of possible future work.

Chapter 2

Review of Mixed Integer Non-Linear Programming

This chapter contains a review of the deterministic techniques most commonly used to solve MINLPs as well as a review the literature relating to MIQPs. The MINLP review does not go into detail concerning state of the art methods based on these techniques since, as detailed in section 2.2.6, they are not suitable for derivative free applications. The MINLP review is divided into two parts; in section 2.1 we discuss methods developed for convex MINLPs while section 2.2 reviews the non-convex case. The MIQP review is contained in section 2.3. The material contained in the MIQP review is relevant to the solution of problem (1.2). Some concluding remarks are given in section 2.4.

2.1 Convex MINLPs

MINLPs are problems with non-linear objective functions and constraints in which both continuous and integer variables are present. This is one of the most general classes of optimization problems and a large number of real world problems from many different disciplines can be formulated as MINLPs; a list of examples is given in [34]. In this section we consider MINLPs with the following form:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & c_i(x) \leq 0, \quad i = 1, \ldots, n_{\text{cons}}, \\ & x = \left[x_c^T, x_d^T\right]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}, \end{aligned} \tag{2.1}$$

where $f(x)$ and $c_i(x)$ are convex functions of x and n_{cons} is the number of constraints. In the following sections we review the most commonly used deterministic procedures for solving problem (2.1). Good reviews of this topic can be found in [34, 67, 120].

2.1.1 Branch and Bound

Branch and Bound methods were initially introduced for solving mixed integer linear programs (MILPs) in [50] and were extended to the non-linear case in [36, 69, 142]. The basic idea behind Branch and Bound algorithms is to use the continuous relaxation of problem (2.1) to obtain valid lower bounds and explore the space of the integer variables using a tree search. Specifically, this method proceeds by solving the continuous relaxation of problem (2.1). If the solution of the continuous relaxation is feasible for problem (2.1) then the optimal solution has been found and the algorithm is stopped. Otherwise the solution of the continuous relaxation provides a lower bound on the solution of problem (2.1) and the algorithm selects and branches on one of the integer variables, say x_i. Branching is achieved by choosing a branching point, say $\hat{x}_i$, and forming two new problems by adding the constraints $x_i \leq \lfloor \hat{x}_i \rfloor$ and $x_i \geq \lceil \hat{x}_i \rceil$ respectively to problem (2.1). One of the new problems is then selected, its continuous relaxation is solved and the branching process is repeated. In this way a search tree is generated in which each node corresponds to a MINLP obtained from problem (2.1) using bound constraints. At each iteration nodes are pruned if they satisfy one of three fathoming rules:

i) The continuous relaxation of the MINLP is infeasible.

ii) The lower bound obtained using the continuous relaxation is greater than the current upper bound on the solution.

iii) The solution of the continuous relaxation is feasible for problem (2.1).

If a node is fathomed using iii) then the solution of the continuous relaxation provides an upper bound on the solution of problem (2.1). No branching occurs on a fathomed node. The algorithm stops once all of the leaf nodes of the search tree have been fathomed and returns the lowest upper bound as the solution. Generally the Branch and Bound method is only effective when the size of the search tree is small or when the continuous relaxations are easy to solve.

In [36, 98] an extension to the Branch and Bound method is proposed based on using sequential quadratic programming (SQP) to solve the continuous relaxations. The change to the basic Branch and Bound procedure is

that early branching is allowed. Specifically, after a certain number of SQP iterations a decision is made whether to branch before solving the continuous relaxation completely. While this method reduces the computational cost of the continuous relaxations it precludes fathoming using ii).

2.1.2 Outer Approximation

The Outer Approximation method was first introduced in [55] for problems which are linear in the integer variables. The method was extended to the general non-linear case in [56]. The outer approximation algorithm proceeds by solving a finite alternating sequence of non-linear programming (NLP) subproblems and relaxations of a MILP master problem (MP). The MP is constructed by reformulating problem (2.1) as a MILP and relaxing the problem by removing some of the constraints. This relaxed problem is the MP and its solution provides a lower bound on the solution of problem (2.1). The MILP reformulation is obtained using supporting hyperplanes. At each iteration of the algorithm the MP is used to obtain a vector of values for the discrete variables; the vector used is the discrete component of the solution of the MP. An NLP subproblem is then generated by fixing the values of the discrete variables in problem (2.1) to be equal to the vector generated by the MP. The NLP subproblem is used to find values for the continuous variables. If the NLP is infeasible a feasibility problem is solved and a new master problem is generated by adding constraints to the current MP which make the vector of discrete variables used in the construction of the NLP infeasible. Otherwise, if the NLP is feasible its solution gives an upper bound on the solution of problem (2.1) and the MP for the next iteration is constructed by adding cuts to the current MP which ensure that a better lower bound is obtained during the next iteration. The algorithm is either stopped when the MP becomes infeasible or when the upper and lower bounds are within a specified tolerance.

Outer Approximation performs poorly on functions which cannot be well approximated by linear functions. To overcome this limitation Fletcher and Leyffer [56] developed the quadratic outer approximation method by adding a second order Lagrangian term to the objective function of the MP. This new master problem is a MIQP. This approach allows a better representation of the non-linearities in problem (2.1). The disadvantages of quadratic outer approximation are that the MP no longer provides a lower bound on problem (2.1) and that the MP takes longer to solve since it is now a MIQP rather than a MILP.

Another possible flaw of the Outer Approximation method is that it does not exploit the strong relationship which exists between successive master

problems. In an attempt to overcome this limitation the LP/NLP based Branch and Bound method was developed in [97, 129]. This method solves the master problem generated in the first iteration of the outer approximation algorithm using a Branch and Bound procedure. The NLP subproblems used in Outer Approximation are solved at each node where a feasible solution for the discrete variables is produced.

2.1.3 Generalised Benders Decomposition

Benders Decomposition method was developed for MILP by Benders [29]. The extension of this method to the non-linear case is known as Generalised Benders Decomposition (GBD) [58, 64]. The GBD method is similar to the Outer Approximation algorithm discussed in section 2.1.2 in that it proceeds by solving a finite alternating sequence of NLP subproblems and relaxations of a MILP master problem. The MP used by GBD is not derived using outer approximation, rather it is derived using non-linear duality theory. Another difference is that GBD adds fewer cuts to the MP at each iteration than Outer Approximation. As a result the GBD MP is easier to solve than the Outer Approximation MP. The trade-off is that the MP used in Outer Approximation supplies a tighter lower bound.

2.1.4 Extended Cutting Plane

The Extended Cutting Plane (ECP) method was developed in [158] by extending the continuous cutting plane algorithm [79] to the mixed integer case. The ECP method can only be applied to problems with linear objective functions. Problems with non-linear objectives can easily be transformed to the required form by moving the objective into the constraints. Unlike the previous methods ECP does not require the solution of NLPs, rather it proceeds by solving a sequence of MILPs. At each iteration a linearisation of the most violated constraint is added to the MILP. This causes the series of MILPs solved to produce a non-decreasing sequence of lower bounds on problem (2.1). The algorithm is terminated when the greatest constraint violation lies within a user defined tolerance. In Westerlund and Pörn [159] the ECP method was extended to handle pseudo-convex problems.

2.1.5 Branch and Cut

The Branch and Cut method was originally developed for combinatorial optimization. It was extended to mixed binary programming in [142] and to the general integer case by Kesavan and Barton [80]. The Branch and Cut

procedure is an extension of the Branch and Bound procedure outlined in section 2.1.1. Like Branch and Bound it explores the discrete space using a search tree and uses continuous relaxations of problem (2.1) to obtain valid lower bounds. However, each time a node which cannot be fathomed, is generated the Branch and Cut procedure either branches to create two new nodes or it adds cuts to the continuous relaxation which make the current solution infeasible. If cuts are added the new continuous relaxation is solved and the decision between branching and cutting on the current node is made again.

2.1.6 Disjunctive Programming

In this section we discuss an alternative method of formulating problem (2.1) using algebraic constraints, logic disjunctions and logic relations. This formulation is known as a generalised disjunctive program (GDP) and it takes the following form [67]:

$$
\begin{aligned}
\min_{x_c, c} \quad & \sum_{k \in SD} c_k + f(x_c) \\
\text{s.t.} \quad & c_i(x_c) \leq 0, \quad i = 1, \ldots, n_{\text{cons}}, \\
& \bigvee_{j \in D_k} \begin{bmatrix} Y_{jk} \\ h_{jk}(x_c) \leq 0 \\ c_k = \gamma_{jk} \end{bmatrix}, \quad k \in SD, \\
& \Omega(Y) = \text{True}, \\
& x_c \in \mathbb{R}^{n_c},\ c_k \in \mathbb{R},\ Y_{ik} \in \{\text{True}, \text{False}\},
\end{aligned}
\tag{2.2}
$$

where SD is the set of disjunctions and D_k is the set of terms in the kth disjunction. Y_{jk} are boolean variables which control the space in which the continuous variables can lie. If Y_{jk} is true then term $j \in D_k$ is true in disjunction $k \in SD$. In other words the constraint $h_{jk}(x_c) \leq 0$ must be satisfied by the continuous variables and the variable c_k is given a value γ_{jk}. $\Omega(Y)$ are logical relations, in the form of propositional logic, which must be satisfied by the boolean variables. As in problem (2.1), f and c_i are convex non-linear functions. However in this case they are functions only of $x_c \in \mathbb{R}^{n_c}$ rather than of x [67].

Real world problems are modelled as GDPs when logical conditions are involved in the problem formulation and when the integer variables are used to model conditions where different sets of constraints are valid [120]. In addition, any problem expressed as a MINLP can be transformed to a GDP and vice-versa. Thus, any GDP can be solved by transforming it to a MINLP

and applying one of the procedures described above. However it is often more efficient to solve problem (2.2) directly. A number of the methods described above have been altered to solve GDP rather than MINLPs. Specifically a Branch and Bound procedure has been developed in [91], an Outer Approximation algorithm has been developed in [153] and a GBD algorithm has been developed in [153]. In [155] an Outer Approximation algorithm has been developed which can solve hybrid problems involving both disjunctions and mixed integer constraints.

2.2 Non-convex MINLPs

In this section we consider methods which find a global optimum of problem (2.1) when we remove the constraint that f and c_i must be convex functions. All of the methods discussed in this section are based on the Branch and Bound procedure. Due to the non-convexity of the problem lower bounds are obtained by using convex envelopes or under-estimators to construct lower bounding MINLPs. It is also necessary to branch on both the continuous and the integer variables. Branching on the continuous variables results in search trees which are not finite and accordingly only ε-convergence can be guaranteed. This makes the methods potentially very computationally expensive. The major differences between the methods discussed in this section lie in the methods used to perform the branching and in the procedures used to obtain the lower bounds.

2.2.1 The $\alpha - BB$ method

The $\alpha - BB$ (α Branch and Bound) method was first developed for continuous twice differentiable non-convex NLPs in [14] and was extended to non-convex MINLPs by Adjiman et al. [8]. The $\alpha - BB$ algorithm is a Branch and Bound algorithm which uses convex underestimators of f and c_i to obtain lower bounds. The lower bounds are obtained by replacing f and c_i with their convex underestimators in problem (2.1), relaxing the integrality constraint and solving the resulting convex NLP. The convex underestimators are constructed by underestimating each term in f and c_i. Three types of terms must be considered during this procedure; convex terms, non-convex terms with special structure and arbitrary non-convex terms. Obviously convex terms do not need to be underestimated. Non-convex terms with special structure are terms for which specific convex bounding procedures have been developed. These terms include bilinear terms, trilinear terms, fractional terms and univariate concave functions [107]. Finally, it has been shown in

[14] that any twice differentiable term $r(x)$ is underestimated by the following expression

$$r(x) + \alpha \sum_{i=1}^{n} (u_i - x_i)(l_i - x_i), \tag{2.3}$$

where $\alpha \in \mathbb{R}_+$. It can be shown that (2.3) is convex if and only if α satisfies

$$\alpha \geq \max\left\{0, -\frac{1}{2}\min_{x,k} \lambda_k(x)\right\}, \tag{2.4}$$

where λ_k are the eigenvalues of the Hessian of $r(x)$. Clearly, using (2.3) and (2.4) a convex underestimator of any twice differentiable function can be constructed. This completes the discussion of the lower bounding procedure.

The $\alpha - BB$ method converges to a global minimum of problem (2.1) with finite ε-convergence when f and c_i are twice differentiable. The algorithm is terminated when the upper and lower bounds are within a user specified tolerance. Clearly some procedure is required to determine the upper bounds. Any feasible solution of problem (2.1) provides an upper bound. These values could be obtained when the solution of the convex relaxation is feasible, by using heuristics or by fixing the integer variables and solving the resulting NLP.

Finally, we discuss the branching procedure used by $\alpha - BB$. The branching variable is chosen to be the variable with the most fractional solution at the optimum of the lower bounding problem. If it is possible to use knowledge of the problem to create a priority branching list then the information in the list is incorporated into the choice of branching variable using a hybrid approach. The branching point is chosen to be the value of the branching variable at the optimum of the lower bounding problem.

2.2.2 BARON

BARON (Branch And Reduce Optimization Navigator) is a Branch and Bound based procedure developed by Sahinidis [135]. BARON guarantees ε-convergence to a global optimum of problem (2.1) when f and c_i are factorable functions. Factorable functions can be expressed as recursive sums and products of univariate functions. The lower bounds in BARON are obtained using a combination of convex underestimators and linearisation. The convex underestimators are constructed using factorable programming techniques [110]. The underestimators are then linearised using a two stage outer approximation technique which automatically detects and makes use of convexity in the objective function, further details are given in [147, 148]. In addition to upper and lower bounding BARON also makes use of a number

of range reduction techniques. These techniques are applied to every node of the search tree in pre and post processing procedures to reduce both the search space and the relaxation gap. Further details on the various range reduction techniques can be found in [135].

We now discuss the branching procedure used by BARON. The branching variable x_i is chosen using one of two strategies. The first strategy is developed in [134] and simply chooses the branching variable to be the variable with the greatest domain on the current node. The second strategy is the violation transfer strategy developed in [147]. This procedure assigns each variable a measure of the violation of the non-convexities of the objective function by the current convex relaxation. The variables are then weighted using a priority branching list, should one be available. The branching variable is then chosen to be the variable with the greatest weighted violation measure. For specific details on the violation measure the interested reader is referred to [146] and [147]. Once the branching variable has been chosen the branching point $\hat{x}_i$ is chosen using the following procedure. If the ith coordinate of the current upper bound lies in the domain of the branching variable on the current node then $\hat{x}_i$ is set to the ith coordinate of the upper bound. Otherwise $\hat{x}_i$ is set to the ith coordinate of the solution of the continuous relaxation of the current node. In addition, once in every specified number of iterations the branching point is chosen to be the mid-point of the domain of the branching variable on the current node. This accelerates convergence and is necessary to ensure exhaustiveness.

2.2.3 Couenne

Couenne (Convex Over and Under ENvelopes for Nonlinear Estimation) is a Branch and Bound procedure developed in [28]. Couenne guarantees ε-convergence to a global optimum of problem (2.1) when f and c_i are factorable functions. The lower bounds used by Couenne are obtained in two steps. Firstly, problem (2.1) is reformulated, using a number of auxiliary variables, to a problem whose constraints have a simpler form than problem (2.1). In the second step the reformulated problem is linearised. The linearisation is similar to that used in BARON. This linearised problem is solved to obtain a valid lower bound. The reformulation is only applied to the original problem, i.e. to the problem in the first node of the search tree. The linearisation of the first node is calculated from scratch using a procedure described in [28]. The linearisation used by every other node is calculated by refining the linearisation of its parent node. Upper bounds are obtained by fixing the values of the discrete variables using the solution of the lower bounding problem and solving the resulting NLP. Like BARON, Couenne

makes use of a number of range reduction techniques, details of which can be found in [28].

We now describe the branching procedure used by Couenne. The branching variable can be chosen using one of two strategies. The first strategy chooses the variable with the greatest degree of infeasiblity at the solution of the lower bounding problem, the measure of infeasiblity used to make this choice is given in [28]. The second strategy is the violation transfer strategy [147] which was described in the previous section. Once the branching variable has been selected the branching point is chosen using the following formula

$$\hat{x}_i = \max\left\{l_i + \beta, \min\left\{u_i - \beta, \alpha\tilde{x}_i + \frac{(1+\alpha)(u_i + l_i)}{2}\right\}\right\}, \tag{2.5}$$

where $\beta = 0.2(u_i - l_i)$, $\alpha = 0.25$ and $\tilde{x}$ is the solution of the lower bounding problem. Except that $\hat{x}_i$ is set to zero if the value given by (2.5) is sufficiently close to zero.

2.2.4 SCIP

SCIP (Solving Constraint Integer Programs) is a Branch and Bound procedure, developed in Achterberg [7], based on a paradigm called constraint integer programming (CIP). CIP combines the modelling and solving techniques available in constraint programming, MILP and the satisfiability problem. SCIP cannot solve the general version of problem (2.1), it can only solve problems which have linear constraints in the continuous variables once the integer variables have been fixed. This restriction makes it possible to prove that SCIP converges to the global optimum in a finite number of function evaluations. SCIP prunes the Branch and Bound tree using linear programming (LP) relaxations of problem (2.1), cutting planes and domain propagation. Domain propagation is a concept used in constraint programming which uses the constraints on a subproblem in the Branch and Bound tree to determine domain reductions which can be used to reduce the search space. A final procedure used to prune nodes is conflict analysis, a concept adapted from the satisfiability problem. Conflict analysis generates valid constraints by analysing the infeasible subproblems in the Branch and Bound tree.

A number of branching rules are implemented in SCIP; most infeasible branching, pseudocost branching, strong branching, hybrid strong/pseudo–cost branching, pseudocost branching with strong branching initialization and reliability branching. Further details on these branching procedures are given in Achterberg [7]. SCIP also contains a number of node selection policies; best first search, depth first search, best estimate search, hybrid best

estimate/best bound search and depth first search with periodical selection of the best node. Once again, further details are given in Achterberg [7].

In Berthold et al. [30] SCIP has been extended such that it can be used to solve mixed integer quadratically constrained programs (MIQCP). MIQCPs have the same form as problem (1.2), with the addition of quadratic constraints to the problem. This extension is discussed further in section 2.3.2.

2.2.5 Global optimization of non-convex GDP

In [92] a Branch and Bound procedure is developed for solving a class of non-convex GDPs. This procedure can solve problem (2.2) when f, c_i and h_{jk} are made up of terms that are either convex, bilinear, linear fractional or concave separable. A convex relaxation is constructed using convex underestimators of f, c_i and h_{jk}. This convex relaxation provides lower bounds on the solution of the problem. This algorithm also makes use of range reduction techniques. Specifically, the range reduction techniques developed in [165] are applied.

The Branch and Bound procedure uses two sets of branching rules. The initial set of branching rules only allow branching on the boolean variables. Once a node is produced on which all of the boolean variables are fixed the second set of branching rules is applied. The second set of branching rules is used to fix the continuous variables for given values of the boolean variables. Once the second set of branching rules have been used to fix the continuous variables the solution obtained is used to update the upper bound. A cut is then added to the problem making the choice of boolean variables fixed by the first set of branching rules infeasible. The second set of branching rules is then replaced by the first set. The algorithm terminates when the upper and lower bounds are within a user specified tolerance. This algorithm guarantees ε-convergence to a global minimum of problem (2.2).

2.2.6 Suitability of traditional MINLP methods

Besides Branch and Bound all of the methods discussed in the sections 2.1 and 2.2 make use of the derivative of the objective function which clearly makes them unsuitable for solving derivative free problems. While the Branch and Bound method can be implemented without using the derivative of the objective function it does rely on continuous relaxations of the objective function to obtain lower bounds. There are a number of situations in which these relaxations are not available and these situations require derivative free solution methods [3, 46, 71]. Clearly none of the traditional methods described here can be used to solve problem (1.1) in the derivative free case.

2.3 Mixed integer quadratic programming

In this section we consider in more detail the literature on a specific class of MINLPs, namely mixed integer quadratic programs. This is relevant to the second problem considered in this thesis; the solution of problem (1.2). Most of the literature concerning this problem considers convex MIQPs and solution methods in this case have advanced to the point where reliable, efficient commercial software packages are available. In section 2.3.1 we briefly review the literature related to convex MIQPs and mention some of the software available to solve problem (1.2) in this case. Far less work considers non-convex MIQPs and it is this case that the methods developed in chapter 4 consider. The literature on non-convex MIQPs is reviewed in section 2.3.2.

2.3.1 Convex mixed integer quadratic programming

As noted above reliable, efficient commercial software packages are available for solving convex MIQPs. Accordingly, the amount of work published on the convex case has decreased in recent years with focus shifting to the non-convex case. We now briefly discuss some of the work done on convex MIQPs in the past. In [9] a Cutting Plane algorithm for convex MIQPs is developed. The algorithm adds valid inequalities to the problem until the optimum of the continuous relaxation is integer feasible. In [31] a Branch and Cut algorithm is developed specifically for solving problems arising from portfolio optimization. A Benders Decomposition algorithm is proposed in [88, 89]. The algorithm includes a preprocessing step involving a linear transformation which simplifies the master problem used by the algorithm. Improvements to the Benders Decomposition algorithm are suggested in [58]. A Branch and Bound algorithm using piecewise linear underestimators to obtain lower bounds is developed in [11]. In [97] a Branch and Bound method using a dual active set algorithm with warm starts[1] to solve the continuous relaxations is proposed. A procedure for obtaining improved lower bounds in a Branch and Bound framework is given in [57]. More recently Axehill et al. [23] proposed novel relaxations for use in model predicative control applications and Buchheim et al. [37] present a Branch and Bound procedure that uses a preprocessing step to ensure that the nodes can be processed quickly. The commercial packages available for solving convex MIQPs include Gurobi, CPLEX, MOSEK and XPRESS. CPLEX is used to solve convex MIQPs in

[1]The term warm start is used to describe a situation where information from a previous optimization problem is used to speed up the reoptimization after a minor change has been made to the problem [21].

this thesis. It uses a Branch and Cut algorithm with a Simplex Method used to solve the continuous subproblems [62].

2.3.2 Non-convex mixed integer quadratic programming

The work that has been done on solving non-convex MIQPs can be divided into two classes. The first class considers the derivation of tight convex relaxations and strong cuts which can be used to obtain tight lower bounds on the solution of problem (1.2). The relaxations and cuts can be included in non-convex MINLP algorithms such as those described in section (2.2). The work done in this class does not consider the development of algorithms specifically for solving MIQPs. The second class considers the development of algorithms specifically for solving MIQPs. In chapter 4 we propose modifications to these methods which improve their efficiency.

Inequalities and relaxations

Before beginning our discussion we note that many of the procedures described in this section have been developed for MIQCPs. However, as we consider MIQPs in this thesis all of the techniques in this section shall be discussed in that context. Readers interested in the extension to MIQCP are referred to the relevant references. A good review of all but the most recent techniques discussed in this section can be found in [41].

A procedure used by a large proportion of the inequalities that have been developed to date is known as lifting [41]. This involves the use of auxiliary variables to lift the problem to a higher dimensional space in which it has a simplified structure. The most common form of lifting is to use the auxiliary variables X_{ij} to represent the quadratic terms in the objective using the constraints $X_{ij} = x_i x_j$. These constraints can be represented using the symmetric matrix equation $X = xx^T$. This lifting procedure results in the following reformulation of problem (1.2) [41]:

$$\begin{aligned} \min_{x} \quad & h(x) = \frac{1}{2}\langle H, X\rangle + g^T x \\ \text{s.t.} \quad & Ax \leq b, \\ & Dx = e, \\ & l \leq x \leq u, \\ & X = xx^T, \\ & x = \left(x_c^T, x_d^T\right)^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d},\ X \in \mathcal{S}^n, \end{aligned} \tag{2.6}$$

where $\langle H, X\rangle = \text{Trace}(H, X)$. The objective function is now linear but we have added the non-convex constraint $X = xx^T$ to the problem. It can be shown that problem (2.6) is equivalent to the following problem [41]:

$$\begin{aligned} \min_x \quad & h(x) = \frac{1}{2}\langle H, X\rangle + g^T x \\ \text{s.t.} \quad & (x, X) \in \text{clconv}(\hat{\Omega}), \end{aligned} \tag{2.7}$$

where $\hat{\Omega}$ is the feasible region of problem (2.6) and $\text{clconv}(\hat{\Omega})$ denotes the closed convex hull of $\hat{\Omega}$. A linear relaxation of problem (2.7) is obtained by relaxing the integrality constraint and removing the constraint $X = xx^T$. Denote the feasible region of this problem by $\hat{\mathcal{L}}$. In the following paragraphs we discuss a number of inequalities that can be used to strengthen $\hat{\mathcal{L}}$ by making the feasible region a closer approximation of $\hat{\Omega}$.

We first describe the most common method of constructing inequalities used to strengthen $\hat{\mathcal{L}}$. Suppose $\alpha^T x \leq \alpha_0$ and $\beta^T x \leq \beta_0$ are two valid inequalities for Ω_q, where Ω_q denotes the feasible region of problem (1.2). The following equality is then also valid for Ω_q [41]

$$-(\alpha^T x - \alpha_0)(\beta^T x - \beta_0) \leq 0, \tag{2.8}$$

and it follows that the following inequality is valid for $\text{clconv}(\hat{\Omega})$

$$\alpha_0\beta^T x + \beta_0\alpha^T x - \alpha_0\beta_0 - \langle \beta\alpha^T, X\rangle \leq 0.$$

The inequalities contained explicitly in (2.6) can be used to derive a number of inequalities with the form of (2.8). The linearised versions of this set of inequalities are considered in [85, 104] and are known as the rank-2 linear inequalities. Denote the region defined by the rank-2 linear inequalities by R_2. The rank-2 inequalities include the well known RLT inequalities described in [138].

Another important class of strengthening inequalities are the semidefinite inequalities [41]. These inequalities arise from the fact that for any $x \in \mathbb{R}^n$ the following relation holds

$$\begin{bmatrix} 1 & x^T \\ x & xx^T \end{bmatrix} = \begin{bmatrix} 1 \\ x \end{bmatrix} \begin{bmatrix} 1 \\ x \end{bmatrix}^T \succcurlyeq 0.$$

Considering the constraint $X = xx^T$ in problem (2.6) we see that the following linear constraint is valid for $\text{clconv}(\hat{\Omega})$ [41]

$$\begin{bmatrix} 1 & x^T \\ x & X \end{bmatrix} \succcurlyeq 0. \tag{2.9}$$

The region defined by this constraint is denoted PSD. It is also possible to make use of relaxations of (2.9). For example, in Kim and Kojima [82] all of the principal 2×2 submatrices of the matrix in (2.9) are constrained to by positive semidefinite without requiring that (2.9) is satisfied.

Now we have $\text{clconv}(\hat{\Omega}) \subseteq \hat{\mathcal{L}} \cap R_2 \cap PSD$. Equality has been proven for various special cases, these are not considered here as we are concerned with the general problem. The interested reader is referred to [41]. Replacing the feasible region of problem (2.7) with any combination of $\hat{\mathcal{L}}$, R_2 and PSD gives a convex relaxation of problem (1.2) which can be used to obtain valid lower bounds [41].

In [136] a different approach is used to handle the non-convex constraint $X = xx^T$. This constraint can be represented using the following relations

$$X - xx^T \succcurlyeq 0, \tag{2.10}$$

$$X - xx^T \preccurlyeq 0. \tag{2.11}$$

The first of these constraints is convex while the second is non-convex. Relaxing the feasible region of problem (2.7) to $\hat{\mathcal{L}} \cap PSD$ is equivalent to dropping the non-convex constraint (2.11) from the problem. The procedure developed in [136] avoids dropping (2.11) completely. Instead a dynamic procedure based on axis rotations and disjunctive programming is developed to approximate (2.11).

If it is not possible to represent $\text{clconv}(\hat{\Omega})$ sufficiently accurately in the space (x, X) the problem can be lifted to even higher spaces. This allows the linearisation of constraints of degree greater than two [41]. A sequential convexification theorem is proven in [136] using this idea. The sequential convexification theorem allows us to decompose the non-convexity of problem (1.2) into a number of atomic non-convex problems. The ith atomic problem involves the minimization of a linear function over a non-convex feasible region of the form $\hat{\mathcal{L}} \cap PSD \cap \{(x, X) : X_{ii} \leq x_i^2\}$. A disjunctive approach to approximate this region is described in [136]. We note that this method only applies to problems with binary variables rather than general integer variables.

At this point all of the relaxations that have been given for $\text{clconv}(\hat{\Omega})$ have been defined in the space (x, X). The use of the auxiliary variables X can greatly increase the computational expense of solving a problem. An obvious question to ask is whether the strength of these relaxations can be expressed using inequalities defined only in terms of x. This problem is considered in [137]. Consider the relaxation of $\text{clconv}(\hat{\Omega})$ given by $\hat{\mathcal{L}} \cap RLT \cap PSD$, where RLT denotes the region described by the RLT inequalities described above. Let $\mathcal{Q}$ denote the coordinate projection of $\hat{\mathcal{L}} \cap RLT \cap PSD$ onto the space

of x. $\mathcal{Q}$ is a relaxation of clconv(Ω) defined only in terms of the original variables. An algorithmic framework is developed in [137] which, given a point $\bar{x}$, either shows that $\bar{x} \in \mathcal{Q}$ or provides a valid inequality which is violated by $\bar{x}$. This allows the relaxations derived in the lifted space to be used in the original space.

A possible alternative to the lifting procedures described above is to obtain a convex relaxation of problem (1.2) by replacing h with its convex envelope [41]. A convex envelope of a function h is the convex function which most closely underestimates h. Another approach, proposed in [101], is to use a combination of lifting and convex envelopes. However, finding the convex envelope of an arbitrary quadratic function over a polytope is itself an NP-hard problem. One approach used to reduce the cost of formulating convex underestimators, which was mentioned in section 2.2, is to underestimate each term in h separately. Specifically, the convex envelopes of bound constrained terms of the form $x_i x_j$ and $-x_i^2$ can be derived analytically and can be used to construct convex underestimators of quadratic objective functions [135].

Another relaxation approach which does not require lifting is to formulate a convex relaxation of problem (1.2) as a second order cone program (SOCP) [81, 137]. This approach can only be applied to problems with linear objective functions and quadratic constraints. Problem (1.2) can easily be transformed to this form by moving the objective into the constraints. The Hessian is then written as the difference of two carefully chosen positive semidefinite matrices, H^- and H^+. The quadratic constraint can now be written in the following form

$$\frac{1}{2}x^T H^+ x + g^T x \leq \frac{1}{2}x^T H^- x.$$

An auxiliary variable z is introduced to represent $\frac{1}{2}x^T H^- x$. The constraint on the auxiliary variable is relaxed to give

$$\frac{1}{2}x^T H^- x \leq z.$$

The quadratic constraint now takes the form

$$\frac{1}{2}x^T H^+ x + g^T x \leq z,$$
$$\frac{1}{2}x^T H^- x \leq z.$$

Since H^- and H^+ are positive semidefinite these constraints describe a convex feasible region. This relaxation is only effective if z is bounded above by some $\mu \in \mathbb{R}$. The value of μ will depend on the problem and the choices

of H^+ and H^-. Guidelines for choosing μ are given in [81]. The relaxed problem generated using this approach has a linear objective and linear and convex quadratic constraints. This problem can then be represented as an SOCP which can be solved using the primal-dual interior-point method [81].

Finally we discuss a class of valid inequalities developed in [61]. They are known as gap inequalities and are an extension of the gap inequalities derived in [87] for the max-cut problem. They are developed for the lifted space using the fact that the matrix in (2.9) is positive semidefinite. While these inequalities are tight they are NP-hard to compute. To reduce the computational expense a number of valid inequalities which are less tight but easier to compute are reviewed in [87]. Alternatively the gap inequalities could be calculated using heuristics or a relaxation of the gap inequalities could be used.

Solution methods

In this section we review the work that has been done on developing algorithms for solving non-convex MIQPs. Besides the modification of SCIP mentioned in section 2.2 the work that has been done in this area considers special cases of problem (1.2) rather than the general problem.

The first method we consider was developed in [38] and makes use of a semidefinite relaxation of the problem embedded in a Branch and Bound framework. The specific problem considered in [38] has the following form:

$$\begin{aligned} \min_{x} \quad & h(x) = \frac{1}{2}x^T Hx + g^T x \\ \text{s.t.} \quad & x \in D_1 \times \cdots \times D_n, \end{aligned} \tag{2.12}$$

where D_i is an arbitrary closed subset of $\mathbb{R}$. Some examples of possible subsets are $\{-1, 1\}$, $[-1, 3]$, $\mathbb{Z}$ and $\{0\} \cup [1, \infty)$. While the domains of the individual variables are more complicated then those that can be represented by problem (1.2), linear constraints cannot be represented by problem (2.12). Accordingly, the method developed in [38] can only solve a subset of the problems represented by problem (1.2), namely those with the form:

$$\begin{aligned} \min_{x} \quad & h(x) = \frac{1}{2}x^T Hx + g^T x \\ \text{s.t.} \quad & l \leq x \leq u, \\ & x = \left(x_c^T, x_d^T\right)^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}. \end{aligned} \tag{2.13}$$

Using the lifting procedure detailed in the previous section the following

convex relaxation of problem (2.12) can be derived:

$$\begin{aligned} \min_{x} \quad & h(x) = \frac{1}{2}\langle H, X\rangle + g^T x \\ \text{s.t.} \quad & (x_i, X_{ii}) \in \text{clconv}(P_i), \quad \forall i = 1, \ldots, n, \\ & X - xx^T \succcurlyeq 0, \end{aligned} \tag{2.14}$$

where $P_i = \{(x_i, x_i^2) : x_i \in D_i\}$. To use this relaxation linear constraints that can be used to model clconv(P_i) are needed. In general this requires an infinite number of constraints. However a separation algorithm is given in [38] which, given a pair of values $(\bar{x}_i, \bar{X}_{ii})$, either returns a valid inequality for P_i which is violated by $(\bar{x}_i, \bar{X}_{ii})$ or shows that $(\bar{x}_i, \bar{X}_{ii}) \in P_i$. Using this separation algorithm problem (2.14) can be solved using an interior point algorithm.

Problem (2.12) can now be solved using a Branch and Bound algorithm. The lower bounds for each node are obtained by solving problem (2.14) with an interior point algorithm. Upper bounds are obtained by evaluating the objective function of problem (2.12) at feasible values of x obtained from the solution of problem (2.14). The branching variable is chosen to be the variable minimizing $\tilde{X}_{ii} - \tilde{x}_i^2$ where $(\tilde{x}, \tilde{X})$ is the solution of problem (2.14). The branching point is chosen to be $\tilde{x}_i$. These choices of branching variable and branching point improve the convergence of the algorithm.

The second method considered in this section is a convex reformulation scheme developed in [33]. The reformulation results in an equivalent convex MIQP. Unlike the previous method the convex reformulation can handle linear constraints. However, it is restricted in the type of Hessian matrix it can handle. Specifically, only problems for which the n_cth principal leading submatrix of H is positive semidefinite can be solved using this procedure. The convex reformulation is achieved by adding a number of auxiliary variables to the objective function. The best coefficients for the auxiliary variables are determined by solving a semidefinite program [33]. Since the MIQP resulting from the reformulation is convex it can be solved using the methods outlined in section 2.3.1. Further details on this procedure are given in chapter 4.

In [40] it is shown that any non-convex MIQP with binary rather than general integer variables can be reformulated as a LP over the convex cone of copositive matrices. Problems with this form are known as copositive programs. While this reformulation does not remove the difficulty of the problem it does allow it to be transferred to the constraints representing the convex cone of copositive matrices. This allows methods developed for copositive programs to be applied to non-convex MIQPs. This reformulation is less effective than [33] since the algorithms developed for solving convex MIQPs

are more mature than existing algorithms for solving copositive programs. We note that the method developed in [40] can only be applied to problems with equality constraints and lower bound on the variables. However, any constrained MIQP can be transformed to a problem with the required form using slack variables.

We now discuss the modification of SCIP developed in Berthold et al. [30] which allows MIQPs to be solved. The LP relaxations used to obtained lower bounds are constructed by moving the objective into the constraints and using an outer approximation constructed by linearising the convex terms and using linear underestimators for the non-convex terms. The linear underestimators used are the convex envelopes for bilinear and concave univariate functions. The convex envelope of a concave univariate term is merely its secant. The concave envelope of a bilinear term ax_ix_j is obtained by introducing and additional variable κ_{ij} to the problem with the following constraints

$$\begin{cases} \kappa_{ij} \geq l_j x_i + l_i x_j - l_i l_j, \\ \kappa_{ij} \geq u_j x_i + u_i x_j - u_i u_j, \end{cases} \quad \text{if } a > 0, \tag{2.15}$$

$$\begin{cases} \kappa_{ij} \leq l_j x_i + u_i x_j - u_i l_j, \\ \kappa_{ij} \leq u_j x_i + l_i x_j - l_i u_j, \end{cases} \quad \text{if } a < 0. \tag{2.16}$$

We see that the underestimation of each bilinear term requires the addition of a variable and two inequality constraints to the problem. Upper bounds on the solution are obtained using heuristics, details on the heuristics used are give in Berthold et al. [30] and Achterberg [7]. Cutting planes are also made use of by the modification of SCIP. At each node of the Branch and Bound tree where the solution of the LP relaxation is infeasible for problem (1.2) an attempt is made to make the solution infeasible for the LP relaxation by adding cutting planes to the outer approximation. Additionally, at each node in the tree a domain propagation procedure is applied in attempt to reduce the feasible solution of the node. The modification of SCIP also includes a presolving phase in which various reformulations and simplifications are made in an attempt to simplify the problem, these procedures include dual bound reduction, domain propagation and reformulation of constraints containing binary variables. Further details are given in [30].

The branching variable x_i is chosen using a pseudocost branching rule. The pseudocost of a variable is an estimate of the change in the objective function value of the LP relaxation that will result from branching on the variable. Details on calculating the pseudocost are given in [7, 30]. The branching variable is chosen to be the variable with the highest pseudocost. The branching point $\hat{x}_i$ is chosen to be the value of the branching variable

at the solution of the LP relaxation $\tilde{x}$, unless this point would be too close to the bounds in which case it is shifted inwards. Specifically the branching point is given by

$$\hat{x}_i = \min\left\{\max\left\{\tilde{x}_i,\, 0.2l_i + (1-0.2)u_i\right\},\, 0.2u_j + (1-0.2)l_j\right\}.$$

This completes our discussion of the modification of SCIP.

Finally we note that a number of approaches have been developed for problems which use binary rather than integer variables and are linear in the binary variables. This problem is considered too distant from problem (1.2) to be reviewed in detail here. The interested reader is referred to [113, 114] and the references contained therein.

2.4 Conclusion

In this chapter we have presented a review of the traditional deterministic methods used for solving MINLPs. As explained in section 2.2.6, none of these approaches is suitable for the derivative free case. The methods which have been developed for solving derivative free MINLPs are reviewed in chapter 3. We have also given a review of the current state of research on MIQPs. Mature commercial software is available for solving these problems in the convex case and a list of packages that can be used to solve convex MIQPs is given. In this thesis we use CPLEX for solving convex MIQPs. A brief review of the literature related to convex MIQPs is also given. A more detailed review is given of the literature on non-convex MIQPs. We note that, besides the modification of SCIP, the solution methods that consider this problem are all concerned with special cases of problem (1.2). Procedures for solving the general version of problem (1.2) are developed in chapter 4.

Chapter 3

Review of Derivative Free Programming

This chapter contains a review of derivative free optimization for both continuous and mixed integer problems. In section 3.1 we examine the conditions under which derivative free methods are required and we give examples of applications where these conditions occur. Section 3.2 contains a review of derivative free optimization of continuous problems; the focus of the review is on methods relevant to mixed integer derivative free optimization. In section 3.3 we give a detailed review of the BOBYQA algorithm. In section 3.4 we review the current state of mixed integer derivative free optimization.

3.1 Introduction to derivative free optimization

A large proportion of currently existing optimization methods make extensive use of the derivatives of the objective function and the constraints. This is only natural since the derivatives contain a large amount of information that is extremely useful when trying to find the optimal value of a function, indeed the necessary conditions for local optima are normally stated in terms of the gradient of the objective function. The gradients are either supplied by the user or are estimated using finite differences. However, there are a number of situations in which the derivatives are unavailable or unreliable. Obviously in these situations traditional gradient based methods cannot be applied. This is the motivation behind the development of derivative free optimization methods [46].

Problems requiring derivative free solution approaches arise in a wide number of applications. However there are a fairly small number of general

problem characteristics which result in unavailable derivatives [46];

i) When the objective function is noisy derivatives are unreliable.

ii) When the objective function is expensive derivatives generally take a prohibitively long time to calculate.

iii) When the objective function values are obtained via black box simulation derivatives are generally unavailable.

Problems with these characteristics arise in engineering, finance, industrial design, dynamic pricing, medical imaging, groundwater supply problems and operations research, among others [46, 68, 71, 72]. These problem characteristics occur most commonly in combination with mixed variables in engineering applications [17, 71].

3.2 Derivative free optimization of continuous problems

A large amount of work has been done on developing various derivative free solution methods for continuous problems. These methods can be divided into three classes; metaheuristics, surrogate based optimization and local sampling [71]. Metaheuristics use stochastic methods to find an approximation to the global solution. Surrogate based optimization techniques make use of models of the objective function and are often used to solve problems with expensive objective functions. Local sampling methods are deterministic procedures designed to find local optima. A survey comparing 22 different algorithms taken from each of the three categories can be found in [130]. The local sampling methods are most relevant to this thesis since our solution technique can be thought of as a generalisation of this approach to the mixed integer case. Accordingly we give a fairly detailed overview of deterministic sampling in section 3.2.3. For the sake of completeness brief overviews of the metaheuristic and surrogate approaches are given in sections 3.2.1 and 3.2.2 respectively.

3.2.1 Metaheuristics

Metaheuristics are procedures designed to attempt to find global minima by methodically searching the solution space [3]. This is done by using stochastic methods to generate new sample points from previously evaluated points. One of the main characteristics of a number of metaheuristics is that they can

accept points which increase the objective function value, this is a mechanism to allow them to escape local minima [71]. In addition metaheuristics have very few restrictions on the types of problems to which they can be applied [3]. However, the overwhelming majority of metaheuristics lack deterministic proofs of convergence [3]. Almost all of the convergence results that do exist show that the metaheuristics converge asymptotically in probability [3]; if the algorithm is run for enough iterations the probability that the global minimum has been located can be made as close to one as desired [133]. These proofs do not provide any practical guarantee of convergence in real problems [133]. In the rare cases when deterministic proofs exist they generally rely on assumptions that slow the convergence of the algorithm considerably, an example from [112] is discussed below. We see that while metaheuristics are useful in improving the objective function value they should not be applied in situations were a deterministic solution is desired [3]. Another disadvantage of the use of metaheuristics in derivative free optimization is that many metaheuristics require a large number of function evaluations which makes them unsuitable for problems with expensive objective functions [71]. The literature on metaheuristics is large and we will merely give a brief overview of three of the most common derivative free methods; simulated annealing, particle swarm optimization and evolutionary algorithms. Interested readers are referred to [109, 145] and the references contained therein for a more extensive survey.

The simulated annealing algorithm is based on the physical annealing process, in which a substance is heated up followed by slow cooling while attempting to remain as close as possible to thermal equilibrium [151]. Simulated annealing algorithms treat the objective function as the free energy of a system and the variables as particles [71]. The temperature of this simulated system is gradually decreased following some user specified cooling schedule. The algorithm is stopped once some minimum temperature is reached [151]. A number of papers prove convergence in probability for specific instances of simulated annealing algorithms, for example see [63, 70, 103, 144]. However, numerical studies show that convergence can be very slow [76]. Indeed in [32, 133] problems are presented for which it is suggested that simulated annealing algorithms will not converge in a finite time. A good review of simulated annealing algorithms and their convergence properties is given in [151].

Particle swarm optimization algorithms use a finite set of particles, whose positions represent values of x, to explore the objective function. Each particle moves in the search space with a velocity that depends on the best position of that particle as well as the overall best position of the particles [26]. A proof of convergence in probability for an instance of a particle swarm

algorithm is given in [77]. Proofs of deterministic convergence are given in [150, 154]. However these proofs are given for a version of the algorithm which has had its stochastic components removed. This greatly reduces the effectiveness of the algorithm and the results in [150, 154] are mainly used to set parameters in the usual stochastic version of the particle swarm algorithm. Reviews of particle swarm optimization can be found in [26, 27, 121].

Evolutionary algorithms are algorithms which model biological evolution. Three algorithms of this type have been applied to optimization problems; genetic algorithms, evolutionary programming and evolution strategies [24]. Evolutionary algorithms deal with a finite set of sample points simultaneously. At each iteration a subset of the sample points is chosen to use to generate new sample points using various search operators. These search operators are chosen such that the algorithm models biological evolution [24]. Further details on the differences between the three types of evolutionary algorithms and the details of their implementation can be found in [10, 24, 164]. Convergence in probability has been proven for instances of genetic algorithms and evolution strategies, for example see [25, 83, 112, 132]. One of the rare deterministic convergence results mentioned in the first paragraph of this section is derived in [112] for a certain class of genetic algorithms under certain conditions. However, this method often displays very slow convergence in practice since the proof requires an improvement in the sample sets average function value at each iteration. If an iteration produces a sample set without this property it is discarded and more iterations are performed until an appropriate sample set is found. However, the further the algorithm has progressed the less likely the generation of such a sample set becomes. This slows the convergence of the algorithm considerably [3].

3.2.2 Surrogate based optimization

Surrogate based optimization involves replacing the objective function with a surrogate model which is easier to optimize. Surrogate optimization generally focuses on reducing the number of function evaluations and is most frequently used in engineering. Good reviews can be found in [46, 59, 71, 128, 140, 157]; the remainder of the material in this section is taken from these reviews. It is important to note that the surrogate model is not designed to fit f as well as possible, rather it should capture the main features of the objective function to allow the minimum to be reached as quickly as possible. The search for an optimum using surrogate models usually involves an iterative procedure in which a series of surrogate models are generated. This is done using a limited number of evaluations of f to aid in model correction and validation.

Surrogate models can be broadly divided into two classes, physical models

and functional models. Physical surrogate models result from a simplification of the physical or numerical process used to obtain the objective function value. This could be done, for example, by linearising terms in the equations describing a system or by using a coarser grid when solving a system of PDEs numerically. For some problems this simplified process can be used as the surrogate model. However, in a number of problems the simplified process does not approximate f sufficiently well to allow its use as a surrogate. A number of techniques have been developed to incorporate the simplified model into a surrogate optimization framework; the most commonly used are response correction, multi-point methods and space mapping. For further detail on these procedures readers are referred to the reviews listed above. Physical models are generally based on the physical characteristics of the system being modelled. Accordingly, they cannot typically be applied to problems other than the one for which they were developed.

Functional surrogate models are constructed without utilizing any particular knowledge of the process used to evaluate the objective function. Instead functional models are algebraic representations of the objective function constructed from data obtained by sampling the objective function. The generation of a functional model generally requires the following components; a set of basis functions, a sampling procedure, a fitting criterion and a procedure to combine the first three components into a coherent whole. As before interested readers are referred to the referenced reviews for specifics.

Convergence results have been proven for some surrogate optimization techniques, for example see [12, 35, 140]. However, a large percentage of existing surrogate optimization techniques lack convergence proofs and a number of approaches for which convergence has been proven actually prove convergence on the surrogate model rather than f [140].

3.2.3 Local sampling methods

Local sampling methods are iterative procedures which choose a new sampling point in the kth iteration from the region around the best point available at the kth iteration. In a large proportion of local sampling methods the new points are chosen in such a way that convergence to either first or second order stationery points can be proven [46]. The deterministic nature of many local sampling techniques is the reason that they are the main focus of the derivative free portion of this thesis. Local sampling procedures can further be divided into three categories; direct search, methods incorporating line-search based on simplex derivatives and trust region methods [46].

A large number of local sampling methods have been developed for either bound constrained or unconstrained optimization [46]. However a number

of methods have been developed to allow these approaches to be used in the solution of problems with more general constraints. Some of these approaches include augmented Lagrangians [53, 96], inexact restoration [39], extreme barrier approaches [20] and filter methods [18].

Direct search

Direct search methods are derivative free methods that do not make any derivative approximations or build any models. Instead direct search procedures proceed by sampling f at a finite number of points in each iteration and deciding which actions to take based solely on those objective function values [46]. Direct search methods can be divided into two categories; directional direct search and simplicial direct search. The difference between these categories lies in the procedure used to determine which points to sample. Directional direct search methods sample along a set of directions forming a positive basis while simplicial methods use simplices and operations over simplices to determine the sample points [46].

In general directional direct search procedures proceed in the following manner [46]. At each iteration a step size α_k is available. During each iteration of the procedure the objective function is evaluated at the points in the set

$$P_k = \{x_k + \alpha_k d : d \in \mathcal{D}\}, \tag{3.1}$$

where $\mathcal{D}$ is the set of search directions and x_k is the point with the lowest function value that has been found thus far. P_k can be thought of as a subset of the points contained in the following mesh

$$M_k = \{x_k + \alpha_k \mathcal{D} b : b \in \mathbb{Z}_+^n\}.$$

This mesh is merely conceptual; it is never generated in its entirety, rather points in the mesh are generated as need. P_k is given a specific order and is called the set of poll points while $\mathcal{D}$ is known as the set of poll directions. In line with this notation the procedure of evaluating the objective function at the poll points is called polling. Polling aims to find a point with a lower objective function value than x_k. If it is successful x_{k+1} is set to be the point obtained by polling and the step length is set such that $\alpha_{k+1} \geq \alpha_k$. If polling fails to find a point with a lower objective function value than x_k the quantities used in the $(k+1)$th iteration are $x_{k+1} = x_k$ and $\alpha_{k+1} \leq \alpha_k$. The polling procedure can either be opportunistic or complete. In opportunistic polling the poll step is stopped as soon as a point with a lower objective function value than x_k is found. Complete polling examines all of the poll points and then uses the point with the lowest objective function value. In

addition to the poll step a number of algorithms also contain a search step. During the search step the objective function is evaluated at a finite number of points in M_k with the aim of finding a point with a lower objective function value than x_k. This could be done, for example, using heuristics or surrogate models.

A number of direct search algorithms have been developed following this general procedure but using a variety of techniques to perform the individual steps. In the following paragraphs we give a brief discussion of some of the most well known of these algorithms. In addition we give a slightly more detailed discussion of simplicial direct search procedures.

Coordinate search Coordinate search is perhaps the simplest of the directional direct search procedures. Coordinate search uses the following positive basis to define its set of poll points

$$\mathcal{D} = [I_n, -I_n],$$

where I_n is the $n \times n$ identity matrix. Coordinate search generally uses opportunistic polling and a simple procedure to calculate α_{k+1}, for example we could have $\alpha_{k+1} = 2\alpha_k$ when polling succeeds and $\alpha_{k+1} = \frac{1}{2}\alpha_k$ if polling fails [46].

Generalised pattern search Torczon [149] introduced the class of generalised pattern search (GPS) algorithms. GPS algorithms systematically polls points that lie on a mesh M_k centered on x_k and defined by a finite set of positive spanning directions. Importantly it was shown that if all x_k are contained in a compact set and f is continuously differentiable in the neighbourhood of the level set $\{x : f(x) \leq f(x_0)\}$ where x_0 is the initial point then a sequence of GPS iterations converge to a point x^* satisfying $\nabla f(x^*) = 0$. In addition it is shown that coordinate search, evolutionary operation using factorial designs, the original pattern search algorithm developed in [73] and the multidirectional search method are all special cases of the GPS algorithm [149]. Search steps may be included in GPS algorithms without affecting the convergence results. The GPS algorithm was generalised to bound and linearly constrained problems in [94] and [95] respectively. In [18] non-linear constraints are incorporated into GPS using a filter method. Filter methods solve a bi-objective optimization problem where one of the objectives is f and the other is a measure of the constraint violation. However the filter method does not guaranty convergence to a stationery point [16].

Mesh adaptive direct search The mesh adaptive direct search (MADS) algorithm was developed by Audet and Dennis [19] to overcome the limitations on the convergence of the filter GPS algorithm [18]. MADS modifies the poll step of the algorithm by considering a variable set of poll directions. The set of possible poll directions is asymptotically dense in $\mathbb{R}^n$. In addition MADS uses two parameters two generate the poll points rather than the single parameter α_k in (3.1). The first parameter is the poll size parameter; the poll size parameter determines the region from which points can be selected. The second parameter is the mesh size parameter; the mesh size parameter defines a mesh within the region determined by the poll size parameter [130]. This change in the structure of the algorithm allows proof of convergence to either first [19] or second order [5] stationery points in the Clarke sense [43]. Whether the convergence is proven to first or second order depends on the assumptions made about the smoothness of the objective function [5, 19].

Incorporating line search Lucidi and Sciandrone [106] develop a directional direct search method which incorporates a line search along the poll directions. The algorithm uses the same set $\mathcal{D}$ as is used in coordinate search and can be applied to bound constrained problems. The algorithm proceeds by polling points defined using $\mathcal{D}$ until it finds a direction which gives a sufficient decrease in f. A derivative free line search is then performed along this direction and x_{k+1} is set equal to the resulting point. Once all of the poll directions have been considered for a specific x_k without x_{k+1} being set the algorithm sets x_{k+1} to any point satisfying $f(x_{k+1}) < f(x_k)$, i.e. the sufficient decrease condition is removed. It should be noted that the algorithm only allows the use of a search step once all of the poll directions have been considered. A convergence proof is given showing that the algorithm converges to a stationery point.

Simplicial direct search The simplest simplicial direct search algorithm is the well known Nelder-Mead algorithm [118]. Every iteration of the Nelder-Mead algorithm is based on a simplex of $n+1$ vertices ordered by increasing objective function value. The next sample point is determined by performing a reflection, an expansion, a contraction or a shrink on the simplex. Despite its popularity there is no guaranty that the Nelder-Mead algorithm will converge to a stationery point [111]. A number of modifications to have been proposed to to the original algorithm which allow convergence to be proven. In [152] an algorithm is proposed which controls the geometry of the simplex at each iteration as well as imposing a sufficient decrease condition. Another possible approach is to let the Nelder-Mead algorithm run as normal as long

as a sufficient decrease condition is satisfied. When the sufficient decrease condition is violated the geometry of the sample points is modified in such a way that convergence can be proven [78, 127].

Line-search using simplex derivatives

The second class of local sampling procedures we shall consider locate new points by using simplex gradients to perform derivative free line searches. The point x_{k+1} is set to the point found using the line search. The most well known of these approaches is the implicit filtering algorithm developed by Gilmore and Kelley [65]. The implicit filtering algorithm creates a new simplex at each iteration; in serial applications this can have a detrimental effect on the solution time. In [46] an alternative approach is proposed in which only one point in the simplex is changed in each iteration rather than discarding the entire simplex.

Trust region methods

Trust region methods for derivative free optimization maintain a quadratic or linear model of the objective function based solely on samples of the objective function [46]. Generally quadratic models are used as they allow the curvature of the objective function to be taken into account. The model is constructed in such a way that it can be trusted within some neighbourhood of x_k. The neighbourhood in which the model can be trusted is called the trust region. The models can be constructed using interpolation, regression or any other approximation technique. The most common choice for the trust region is an n-sphere centred on x_k. Given a model and a trust region in the kth iteration a trust region algorithm attempts to find x_{k+1} by finding the minimum of the quadratic model in the trust region. The aim of this type of iteration is to reduce the value of the objective function. In derivative free methods a second type of iteration is often required; one which improves the geometry of the interpolation points [46].

Another important part of a trust region algorithm is trust region management; this is the procedure used to adjust the size of the trust region. The basic idea is to compare the reduction in the objective function value with the reduction predicted by the model. If the comparison is good we either increase the size of the trust region or leave it unchanged. If the comparison is bad the size of the trust region may be reduced [46]. However when considering a reduction in the size of the trust region it is necessary to ensure that the poor comparison is due to the size of the trust region rather than a poor model. This must be considered since derivative free models do not

necessarily become more accurate as the size of the trust region is reduced. Often a geometry improvement iteration is used to increase the quality of the model [46].

The most successful trust region methods are those that use interpolation to build their quadratic models. We now give a brief discussion of the most common of these methods. The first algorithm we consider is simply called Derivative Free Optimization (DFO) and was developed in [45]. DFO requires that its interpolation points always satisfy a condition on their geometry; it uses as its interpolation points the largest subset of the evaluated sample points that satisfy said condition. However, if the number of points in the subset is greater than $\frac{1}{2}(n+1)(n+2)$ the size of the subset is reduced since a quadratic model can generally only interpolate $\frac{1}{2}(n+1)(n+2)$ points. Any freedom remaining in the quadratic model is taken up by minimising the Frobenius norm of the Hessian of the model. The Frobenius norm of a matrix is defined as follows

$$\|\Theta\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(\Theta)_{i,j}^2}, \qquad \Theta \in \mathbb{R}^{(n,n)}.$$

DFO also makes use of geometry improvement iterations to find new interpolation points if there are not a sufficient number of previously evaluated points satisfying the geometry condition.

The second algorithm we consider is BOBYQA, which was developed by Powell [126]. BOBYQA also uses Frobenius norm quadratic models to interpolate a set of sample points and makes use of geometry improvement iterations to ensure the quality of the model does not deteriorate. The derivative free mixed integer methods developed in chapter 5 are based on an extension of BOBYQA to the mixed integer case. Accordingly, a more extensive review of BOBYQA is appropriate; this review is given in section 3.3.

A different approach is taken in the wedge method [108]. Rather than using separate iterations to reduce the objective function and improve the geometry, the wedge method uses only one type of iteration which attempts to reduce the objective function value while simultaneously maintaining an acceptable geometry for the interpolation points. This done by adding an additional constraint to the trust region which prevents the new point from lying on a manifold which would result in an unacceptable geometry should an interpolation point be placed on it. The method derives its name from the fact that the additional constraint is wedge shaped. Unlike the previous approaches the wedge method does not use Frobenius models. Rather it uses a fully determined quadratic model and accordingly it always requires $\frac{1}{2}(n+1)(n+2)$ interpolation points.

3.3 The BOBYQA algorithm

In this section we give a review of the BOBYQA algorithm. The details of this description are all taken from [126] and the references contained therein. The rest of this section is organised as follows. Section 3.3.1 contains a brief overview of BOBYQA. In section 3.3.2 we describe the preliminary calculations and the derivation of the initial quadratic model. In section 3.3.3 we discuss the method used to update the quadratic model. The procedure used to select new interpolation points is discussed in section 3.3.4. In section 3.3.5 we describe a procedure used to deal with computational rounding errors. The remaining details of the algorithm are given in section 3.3.6.

3.3.1 Overview of BOBYQA

BOBYQA is an iterative, derivative free procedure which solves the following optimization problem:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & l \leq x \leq u, \\ & x \in \mathbb{R}^n. \end{aligned} \tag{3.2}$$

This is achieved by constructing a series of quadratic approximations Q_k to the objective function. To fully specify a quadratic model of a function usually requires $\frac{1}{2}(n+1)(n+2)$ interpolation points. BOBYQA allows the user to specify the number of interpolation points m as some integer such that $m \in \left[n+2, \frac{1}{2}(n+1)(n+2)\right]$. This reduction in the number of interpolation points allows BOBYQA to have a computational complexity of $\mathcal{O}(n^2)$. We now give a brief outline of BOBYQA, details on how each step is performed will be given later in this section. At the beginning of the kth iteration a quadratic model Q_k which satisfies the following conditions is available

$$Q_k(y_i) = f(y_i), \quad i \in K, \tag{3.3}$$

where the points y_i are the interpolation points and $K = \{1, 2, \ldots, m\}$. At the start of the kth iteration we also have a trust region radius Δ_k and a point x_k which has the property

$$f(x_k) = \min \{f(y_i) : i \in K\}.$$

Now if certain conditions are satisfied the algorithm stops on the kth iteration and returns x_k and $f(x_k)$. Otherwise, a new interpolation point is selected and replaces one of the current interpolation points. The new point is selected

by finding a step d_k such that $x_k + d_k$ is feasible and $\|d_k\| \leq \Delta_k$. This step is constructed by performing either a trust region iteration or an alternative iteration. A trust region iteration selects d_k by solving the following problem:

$$\begin{aligned} \min_{d_k} \quad & Q_k(x_k + d_k) \\ \text{s.t.} \quad & l \leq x_k + d_k \leq u, \\ & \|d_k\| \leq \Delta_k, \\ & d_k \in \mathbb{R}^n. \end{aligned} \tag{3.4}$$

An alternative iteration selects a d_k which will improve the geometry of the interpolation points. The method used to decide between trust region and alternative iterations is described in section 3.3.4. One of the interpolation points, y_t say, is now replaced by $x_k + d_k$ to form the new set of interpolation points $\widehat{y}_i$. The interpolation points now have the form

$$\widehat{y}_i = \begin{cases} y_i, & i \neq t, \\ x_k + d_k, & i = t. \end{cases} \tag{3.5}$$

Before the $(k+1)$th iteration x_{k+1}, Q_{k+1} and Δ_{k+1} are calculated. The following formula is used to calculate x_{k+1}

$$x_{k+1} = \begin{cases} x_k, & f(x_k) \leq f(x_k + d_k), \\ x_k + d_k, & f(x_k) > f(x_k + d_k). \end{cases}$$

The new quadratic model Q_{k+1} is generated by minimising the Frobenius norm of $\nabla^2 Q_{k+1} - \nabla^2 Q_k$ subject to the constraints

$$Q_{k+1}(\widehat{y}_i) = f(\widehat{y}_i), \quad i \in K. \tag{3.6}$$

This is done by solving a system of linear equations.

Another important feature of BOBYQA is that instead of using one trust region, as described in section 3.2.3, BOBYQA employs inner and outer trust regions. The outer trust region is specified by the radius Δ_k which has already been mentioned. The inner trust region is specified by a radius ρ_k which, like Δ_k is updated at each iteration. The inner trust region radius is used to restrict the placement of new interpolation points and in the termination conditions of BOBYQA.

3.3.2 The initial quadratic model

The user has to supply an initial point x_0, an initial inner trust region radius ρ_1, a final inner trust region radius ρ_{end} and the bounds l and u. The initial

value of the outer trust region radius is given by $\Delta_1 = \rho_1$. From here on we consider the case where $m = 2n + 1$, the reader interested in the more general case $m \in \left[n + 2, \frac{1}{2}(n + 1)(n + 2)\right]$ is referred to [126]. We focus on $m = 2n+1$ since numerical results show it to be the most effective value of m and fixing the value of m allows us to simplify the explanation of BOBYQA. Now, the initial set of interpolation points is constructed by placing points on the boundary of a trust region with radius Δ_1. Since all of the interpolation points are required to be feasible an error is returned if $u_i - l_i \leq 2\Delta_1$. The initial point x_0 may need to be moved to ensure that the interpolation points are feasible, this is done automatically. Let $(x_0)_i$ denote the ith element of x_0. The elements of x_0 are reset using the following formula

$$(x_0)_i = \begin{cases} l_i, & (x_0)_i < l_i, \\ u_i, & (x_0)_i > u_i, \\ l_i + \Delta_1, & l_i < (x_0)_i < l_i + \Delta_1, \\ u_i - \Delta_1, & u_i > (x_0)_i > u_i - \Delta_1, \\ (x_0)_i, & \text{otherwise.} \end{cases}$$

After x_0 is adjusted the value of the first interpolation point is set to $y_1 = x_0$. As shall be seen in the following sections, the point x_0 is used a number of times in BOBYQA. To prevent numerical errors it may occasionally be necessary to shift x_0 during the execution of BOBYQA. The procedure used to shift x_0 and the conditions under which the shift is deemed necessary are given in section 3.3.6. Now, for $i = 1, 2, \ldots, n$, the remaining interpolation points are defined as follows

$$\begin{cases} y_{i+1} = x_0 + \Delta_1 e_i & y_{n+i+1} = x_0 - \Delta_1 e_i, & \text{if } l_i < (x_0)_i < u_i, \\ y_{i+1} = x_0 + \Delta_1 e_i & y_{n+i+1} = x_0 + 2\Delta_1 e_i, & \text{if } (x_0)_i = l_i, \\ y_{i+1} = x_0 - \Delta_1 e_i & y_{n+i+1} = x_0 - 2\Delta_1 e_i, & \text{if } (x_0)_i = u_i, \end{cases} \tag{3.7}$$

where e_i is the ith unit vector in $\mathbb{R}^n$. We note that the interpolation points given by (3.7) may be expressed in the more general form

$$y_{i+1} = x_0 + \alpha_i e_i, \quad y_{n+i+1} = x_0 + \beta_i e_i, \quad i = 1, \ldots, n,$$

where α_i and β_i are non-zero real numbers satisfying $\alpha_i \neq \beta_i$. The interpolation points are expressed in this more general as it will allow some of the formulae developed below to be used in section 3.3.5.

We now describe the procedure used to set up the initial quadratic model. The quadratic model Q_1 can be written in the form

$$Q_1(x) = \frac{1}{2} x^T \nabla^2 Q_1 x + \tau^T x + \zeta, \tag{3.8}$$

where $\tau \in \mathbb{R}^n$ and $\zeta \in \mathbb{R}$. The objective function is evaluated at each of the interpolation points, this allows the use of (3.3) to specify m elements of ζ, τ and $\nabla^2 Q_1$. Specifically we fix ζ, all of the elements of τ and the diagonal elements of $\nabla^2 Q_1$. The remaining elements of $\nabla^2 Q_1$ are set to zero. This is equivalent to using the freedom in the off-diagonal elements of $\nabla^2 Q_1$ to minimise the Frobenius norm of $\nabla^2 Q_1$.

Now as was mentioned in section 3.3.1 the quadratic model is updated by solving a system of linear equations which is itself updated during each iteration. More details about the origin and form of the linear system are given in section 3.3.3. Here we give a brief initial description of the system to facilitate the discussion of the procedure used to set it up. The linear system is square and has the following form

$$\left[\begin{array}{c|c} A & Y^T \\ \hline Y & 0 \end{array}\right] \left[\begin{array}{c} \lambda \\ \hline p \\ q \end{array}\right] = \left[\begin{array}{c} r \\ \hline 0 \end{array}\right],$$

where $\lambda \in \mathbb{R}^m$, $p \in \mathbb{R}$, $q \in \mathbb{R}^n$ and $r \in \mathbb{R}^m$. The definitions of λ, p, q and r will be made clearer in section 3.3.3. A is an $m \times m$ symmetric matrix with the elements

$$(A)_{i,j} = \frac{1}{2}\left[(y_i - x_0)^T (y_j - x_0)\right]^2, \quad i, j \in K. \tag{3.9}$$

Y is an $(n+1) \times m$ matrix with the form

$$Y = \begin{bmatrix} 1 & 1 & \dots & 1 \\ y_1 - x_0 & y_2 - x_0 & \dots & y_m - x_0 \end{bmatrix}. \tag{3.10}$$

Now since p, q and λ are the unknowns in the linear system, the matrix W that we actually work with has the form

$$W = \left[\begin{array}{c|c} A & Y^T \\ \hline Y & 0 \end{array}\right]^{-1} = \left[\begin{array}{c|c} \Omega & \Xi^T \\ \hline \Xi & 0 \end{array}\right].$$

The structure of the initial interpolation points allows explicit formulae for Ω and Ξ to be derived [124]. Ξ in an $(n+1) \times m$ matrix whose elements are assigned as follows. $(\Xi)_{1,1} = 1$ and the remaining elements of the first row of Ξ are zero. For $i = 1, \dots, n$ the $(i+1)$th row of Ξ has three non-zero elements which are given by

$$\begin{aligned} (\Xi)_{i+1,1} &= -\frac{1}{\alpha_i} - \frac{1}{\beta_1}, \\ (\Xi)_{i+1,i+1} &= \frac{\beta_i}{\alpha_i (\beta_i - \alpha_i)}, \\ (\Xi)_{i+1,n+i+1} &= \frac{\alpha_i}{\beta_i (\alpha_i - \beta_i)}. \end{aligned}$$

Now instead of specifying the elements of Ω explicitly, Ω is expressed, using an $m \times (m-n-1)$ matrix Z, in the form $\Omega = ZZ^T$. Ω is written in this form to prevent numerical errors in the updating procedures from increasing the rank of Ω. Now the initial Z matrix will have only three non-zero elements in each column. For $i = 1, \ldots, n$ these elements are given by

$$\begin{aligned}(Z)_{1,i} &= \frac{\sqrt{2}}{\alpha_i \beta_i}, \\ (Z)_{i+1,i} &= \frac{\sqrt{2}}{\alpha_i(\beta_i - \alpha_i)}, \\ (Z)_{n+i+1,i} &= \frac{\sqrt{2}}{\beta_i(\alpha_i - \beta_i)}.\end{aligned}$$

This completes our discussion of the initial calculations.

3.3.3 Updating the quadratic model

As was mentioned in section 3.3.1 the quadratic model is updated by minimising the Frobenius norm of $\nabla^2 Q_{k+1} - \nabla^2 Q_k$ subject to the constraints

$$Q_{k+1}(\widehat{y}_i) = f(\widehat{y}_i), \quad i \in K.$$

The Frobenius norm of a matrix is defined in section 3.2.3. Define $D(x)$ to be the following quadratic function

$$D(x) = Q_{k+1}(x) - Q_k(x), \tag{3.11}$$

$$D(x) = p + (x - x_0)^T q + \frac{1}{2}(x - x_0)^T F (x - x_0), \tag{3.12}$$

where $p = Q_{k+1}(x_0) - Q_k(x_0)$, $q = \nabla Q_{k+1}(x_0) - \nabla Q_k(x_0)$ and $F = \nabla^2 Q_{k+1} - \nabla^2 Q_k$. Using this notation it can be said that the model is updated by minimising $\frac{1}{4}\|\nabla^2 D\|_F^2 = \frac{1}{4}\|F\|_F^2$ subject to the interpolation constraints

$$D(\widehat{y}_i) = f(\widehat{y}_i) - Q_k(\widehat{y}_i), \quad i \in K. \tag{3.13}$$

The factor of $\frac{1}{4}$ is included for convenience and does not affect the solution of the problem. The problem of updating the quadratic model has been transformed into a convex quadratic program whose Lagrangian has the form

$$\mathcal{L} = \frac{1}{4}\sum_{i=1}^{n}\sum_{j=1}^{n}(F)_{i,j}^2 - \sum_{j=1}^{m}\lambda_j \left[D\left(\widehat{y}_j\right) - f\left(\widehat{y}_j\right) + Q_k\left(\widehat{y}_j\right)\right]. \tag{3.14}$$

The partial derivatives of $\mathcal{L}$ with respect to the parameters of $D(x)$ are all zero at the solution of the quadratic programming problem. This gives us the following set of relations for the Lagrangian multipliers

$$\sum_{j=1}^{m} \lambda_j = 0, \tag{3.15}$$

$$\sum_{j=1}^{m} \lambda_j \left(\widehat{y}_j - x_0\right) = 0, \tag{3.16}$$

$$F = \sum_{j=1}^{m} \lambda_j \left(\widehat{y}_j - x_0\right) \left(\widehat{y}_j - x_0\right)^T. \tag{3.17}$$

These relations arise from differentiating $\mathcal{L}$ with respect to p, the elements of q and the elements of F respectively. Now using (3.11) and (3.12) we can express $Q_{k+1}(x)$ as follows

$$Q_{k+1}(x) = Q_k(x) + p + (x - x_0)^T q + \frac{1}{2}(x - x_0)^T F (x - x_0), \tag{3.18}$$

where F takes the form in (3.17). We see that the calculation of Q_{k+1} has been reduced to the calculation of p, q and λ. After substituting (3.18) into (3.13) the constraints on the quadratic program take the form

$$p + \left(\widehat{y}_i - x_0\right)^T q + \frac{1}{2} \sum_{j=1}^{m} \lambda_j \left[\left(\widehat{y}_i - x_0\right)^T \left(\widehat{y}_j - x_0\right)\right]^2 = f(\widehat{y}_i) - Q_k(\widehat{y}_i). \tag{3.19}$$

Recall from section 3.3.2 that the linear system used to find p, q and λ has the form

$$\left[\begin{array}{c|c} A & Y^T \\ \hline Y & 0 \end{array}\right] \left[\begin{array}{c} \lambda \\ \hline p \\ q \end{array}\right] = \left[\begin{array}{c} r \\ \hline 0 \end{array}\right], \tag{3.20}$$

where $r_i = f(\widehat{y}_i) - Q_k(\widehat{y}_i)$, A is defined in (3.9) and Y is defined in (3.10). Obviously in the definitions of A and Y, y_i must be replaced by $\widehat{y}_i$. It is now clear that the first m rows of the system are given by (3.19), the $(m+1)$th row is given by (3.15) and the last n rows are given by (3.16). This completes our discussion of the origin of the linear system.

As noted in section 3.3.2 the inverse matrix W of the linear system in the previous paragraph is the matrix that is actually used by BOBYQA. From here we denote the W matrix used in the kth iteration by W_k. It is important to note that the interpolation constraints (3.13) use the updated set of interpolation points $\widehat{y}_i$ so W_{k+1} is used to obtain Q_{k+1} from Q_k. The

Lagrangian multipliers λ are then equal to the components of $\Omega_{k+1}r$, p is equal to the first component of $\Xi_{k+1}r$ and q is given by the last n components of $\Xi_{k+1}r$. These expressions can be simplified further by noting that $\{y_i : i \in K\}$ and $\{\widehat{y}_i : i \in K\}$ only differ in one point so

$$r_i = f(\widehat{y}_i) - Q_k(\widehat{y}_i) = 0, \quad i \in K \backslash \{t\}. \tag{3.21}$$

It is clear from this equation that $\Omega_{k+1}r$ and $\Xi_{k+1}r$ will just be equal to multiples of the tth column of Ω_{k+1} and Ξ_{k+1} respectively. The multiplying factor in both cases will be $r_t = f(\widehat{y}_t) - Q_k(\widehat{y}_t)$. This allows us to calculate λ, p and q. However for reasons that will become clear p is not required.

Now, using the representation of Q_{k+1} in (3.18), the explicit calculation of all of the elements of $\nabla^2 Q_{k+1}$ from $\nabla^2 Q_k$ requires $\mathcal{O}(mn^2)$ operations [123]. This is clear from the fact that the calculation of F using (3.17) requires two vectors of length n to be multiplied together for each of the m interpolation points. To keep the complexity of the algorithm within $\mathcal{O}(n^2)$ the Hessian matrices of the quadratic models are written in the following form [123]

$$\nabla^2 Q_k = G_k + \sum_{j=1}^{m} \mu_j^k \left(y_j - x_0\right)\left(y_j - x_0\right)^T, \tag{3.22}$$

where $G_k \in \mathcal{S}^n$ and $\mu_j^k \in \mathbb{R}$. A procedure is described below which, given G_1 and μ_j^1, allows G_k and μ_j^k to be calculated for any k. The required initial values are given by $\mu_j^1 = 0$ and $G_1 = \nabla^2 Q_1$ where $\nabla^2 Q_1$ is found using the procedure described in section 3.3.2. A formula for $\nabla^2 Q_{k+1}$ with a similar form to (3.22) is required. Towards this end we define $\mu_j^{k+1} \in \mathbb{R}$ such that the following equations are satisfied

$$G_{k+1} = G_k + \mu_t^k \left(y_t - x_0\right)\left(y_t - x_0\right)^T, \tag{3.23}$$

$$\nabla^2 Q_{k+1} = G_{k+1} + \sum_{j=1}^{m} \mu_j^{k+1} \left(\widehat{y}_j - x_0\right)\left(\widehat{y}_j - x_0\right)^T. \tag{3.24}$$

An explicit formula needs to be found for μ_j^{k+1}. Substituting (3.22) into (3.23) gives

$$G_{k+1} = \nabla^2 Q_k - \sum_{\substack{j=1 \\ j \neq t}}^{m} \mu_j^k \left(y_j - x_0\right)\left(y_j - x_0\right)^T, \tag{3.25}$$

but from (3.5) we have $y_j = \widehat{y}_j$, $j \neq t$ so

$$G_{k+1} = \nabla^2 Q_k - \sum_{\substack{j=1 \\ j \neq t}}^{m} \mu_j^k \left(\widehat{y}_j - x_0\right)\left(\widehat{y}_j - x_0\right)^T. \tag{3.26}$$

Substituting (3.26) into (3.24) and recalling that $F = \nabla^2 Q_{k+1} - \nabla^2 Q_k$ gives

$$F = -\sum_{\substack{j=1 \\ j \neq t}}^{m} \mu_j^k \left(\widehat{y}_j - x_0\right)\left(\widehat{y}_j - x_0\right)^T + \sum_{j=1}^{m} \mu_j^{k+1} \left(\widehat{y}_j - x_0\right)\left(\widehat{y}_j - x_0\right)^T. \quad (3.27)$$

Substituting (3.17) into (3.27) and simplifying gives

$$\begin{aligned} &\sum_{\substack{j=1 \\ j \neq t}}^{m} \left(\mu_j^k + \lambda_j\right)\left[\left(\widehat{y}_j - x_0\right)\left(\widehat{y}_j - x_0\right)^T\right] + \lambda_t \left(y_t - x_0\right)\left(y_t - x_0\right)^T \\ &\quad = \sum_{j=1}^{m} \mu_j^{k+1} \left(\widehat{y}_j - x_0\right)\left(\widehat{y}_j - x_0\right)^T. \end{aligned} \quad (3.28)$$

Now, considering (3.23), (3.24) and (3.28) it is clear that the Hessian of the quadratic model can be updated by updating G_k and μ_j^k as follows

$$\begin{aligned} G_{k+1} &= G_k + \mu_t^k \left(y_t - x_0\right)\left(y_t - x_0\right)^T, \\ \mu_j^{k+1} &= \begin{cases} \mu_j^k + \lambda_j, & j \neq t, \\ \lambda_j, & j = t. \end{cases} \end{aligned} \quad (3.29)$$

In addition to reducing the complexity of the updating process, using this formula for the Hessian allows $\nabla^2 Q_k$ to be multiplied by any vector in $\mathbb{R}^n$ in $\mathcal{O}(mn)$ operations [123]. Accordingly this representation of the Hessian allows the complexity of BOBYQA to be kept within $\mathcal{O}(n^2)$.

A gradient of Q_k is required in each iteration; the gradient $q_k = \nabla Q_k(x_k)$ is used. This gradient is chosen since it both allows a more accurate calculation of d_k in trust region iterations and helps reduce numerical errors when q_k becomes small. Therefore, $Q_k(x)$ can be expressed in the following form

$$Q_k(x) = Q_k(x_k) + (x - x_k)^T q_k + \frac{1}{2}(x - x_k)^T \nabla^2 Q_k (x - x_k). \quad (3.30)$$

Now the values of $Q_k(x_k) = f(x_k)$ and $Q_k(x_{k+1}) = f(x_{k+1})$ are known since x_k and x_{k+1} are interpolation points. Therefore, as noted previously, the value of p is not required in our representation of Q_k. The formula used to update q_k is found by differentiating (3.18), giving the following equation

$$\nabla Q_{k+1}(x_k) = q_k + q + \sum_{j=1}^{m} \lambda_j \left(\widehat{y}_j - x_0\right)^T \left(x_k - x_0\right)\left(\widehat{y}_j - x_0\right). \quad (3.31)$$

This formula gives q_{k+1} when $x_{k+1} = x_k$. When $x_{k+1} = x_k + d_k$ the Taylor series expansion of $Q_{k+1}(x_k + d_k)$ gives

$$Q_{k+1}(x_k + d_k) = Q_{k+1}(x_k) + d_k^T \nabla Q_{k+1}(x_k) + \frac{1}{2} d_k^T \nabla^2 Q_{k+1} d_k. \tag{3.32}$$

Differentiating both sides of this equation gives

$$\nabla Q_{k+1}(x_k + d_k) = \nabla Q_{k+1}(x_k) + \nabla^2 Q_{k+1} d_k. \tag{3.33}$$

Combining (3.31) and (3.33) the following expression for q_{k+1} is obtained

$$q_{k+1} = \begin{cases} \nabla Q_{k+1}(x_k), & x_{k+1} = x_k, \\ \nabla Q_{k+1}(x_k) + \nabla^2 Q_{k+1} d_k, & x_{k+1} = x_k + d_k, \end{cases} \tag{3.34}$$

where $\nabla Q_{k+1}(x_k)$ is given by (3.31).

We now discuss the updating procedure for W_k. At the beginning of the $(k+1)$th iteration the matrix W_k, which is derived using y_i, is available. However, from the above description of the updating of the quadratic model it is clear that W_{k+1} is required to update Q_k. Therefore W_{k+1} needs to be calculated before the procedure for updating Q_k is applied. Before proceeding with our discussion of the updating of W_k we need to define the following quantities. Let $z \in \mathbb{R}^{n+m+1}$ be the vector with the following components

$$\begin{cases} z_i = \frac{1}{2} \left[(y_i - x_0)^T (x_k + d_k - x_0) \right]^2, & i \in K, \\ z_{m+1} = 1, & \\ z_{i+m+1} = (x_k + d_k - x_0)_i, & i = 1, \ldots, n. \end{cases} \tag{3.35}$$

In addition define ϕ, ψ, τ, $\sigma \in \mathbb{R}$ as follows

$$\phi = e_t^T W_k e_t, \tag{3.36}$$

$$\psi = 0.5 \|x_k + d_k - x_0\|^4 - z^T W_k z, \tag{3.37}$$

$$\tau = e_t^T W_k z, \tag{3.38}$$

$$\sigma = \phi\psi + \tau^2. \tag{3.39}$$

Now it is shown in [123] that W_{k+1} can be calculated from W_k using the following formula

$$\begin{aligned} W_{k+1} = & W_k + \sigma^{-1} [\phi (e_t - W_k z)(e_t - W_k z)^T - \psi W_k e_t e_t^T W_k + \\ & \tau \{ W_k e_t (e_t - W_k z)^T + (e_t - W_k z) e_t^T W_k \}], \end{aligned} \tag{3.40}$$

where e_t is the tth coordinate vector in $\mathbb{R}^{n+m+1}$. However, it should be noted that the $(m+1)$th row and column of W_k are not required when updating the quadratic model. The motivation for this statement follows from two observations. Firstly, as noted in the paragraph following (3.21), the value of p is not required by our quadratic model. In addition the $(m+1)$th component of the right hand side of (3.20) is zero. Accordingly, when the right hand side of (3.20) is pre-multiplied by W_k the $(m+1)$th column will have no effect on the result. Clearly it is not necessary to retain the $(m+1)$th row and column of W_k. However, (3.40) needs to be modified to allow us to update W_k without these elements. This can be done as follows. Let s be the integer in K such that $y_s = x_k$. Now let v denote the sth column of W_k^{-1}, the components of v are given by

$$\begin{cases} v_i = \frac{1}{2}\left[(y_i - x_0)^T (x_k - x_0)\right]^2, & i \in K, \\ v_{m+1} = 1, & \\ v_{i+m+1} = (x_k - x_0)_i, & i = 1, \ldots, n. \end{cases} \tag{3.41}$$

Now from the definition of v it is clear that $W_k^{-1} e_s = v$ which gives $W_k v = e_s$. Using this relation the following identities can easily be obtained

$$W_k z = W_k(z - v) + e_s, \tag{3.42}$$

$$z^T W_k z = (z - v)^T W_k (z - v) + 2z_s - v_s, \tag{3.43}$$

where z_s and v_s are the sth components of z and v respectively. Now each occurrence of $W_k z$ and $z^T W_k z$ in (3.37)–(3.40) is replaced by (3.42) and (3.43) respectively. Since $(z - v)_{m+1} = 0$ this allows W_{k+1} to be calculated from W_k when the $(m+1)$th row and column of W_k are not available.

The modified version of (3.40) is used to obtain Ξ_{k+1} from Ξ_k and Ω_k. However, a procedure to update the matrix Z_k, used in the factorisation of Ω_k, is still required. Z_k needs to be updated in such a way that $\Omega_{k+1} = Z_{k+1} Z_{k+1}^T$ were Ω_{k+1} is given by (3.40). The following updating procedure is employed here. First construct an $n \times n$ orthogonal matrix Ψ such that only the first component of the tth row of $Z_k \Psi$ is non-zero. A matrix Ψ with the required form can be constructed using Algorithm 1. Algorithm 1 proceeds by constructing a series of $n-1$ orthogonal matrices Θ each of which sets one element of the tth row to zero. The required orthogonal matrix Ψ is then just the product of the Θ matrices. Ψ is orthogonal since the product of two orthogonal matrices is itself orthogonal [74]. Now, since Ψ is orthogonal, Z_k

can be replaced by $Z_k\Psi$ in the factorisation of Ω_k, as follows

$$\begin{aligned}\Omega_k &= Z_k Z_k^T, \\ &= Z_k I Z_k^T, \\ \Omega_k &= (Z_k\Psi)(Z_k\Psi)^T.\end{aligned}$$

Now set $Z_k := Z_k\Psi$. This gives a Z_k matrix in which the only non-zero element in the tth row of Z_k is in the first column. Accordingly, only the first column of Z_k has to be changed to obtain Z_{k+1}. Specifically, it can be shown that the first column of Z_{k+1} has the following components [125]

$$(Z_{k+1})_{i,1} = \frac{\tau\,(Z_k)_{i,1} + (e_t - e_s - W_k(z-v))_i\,(Z_k)_{t,1}}{\sqrt{\sigma}}, \quad i \in K. \tag{3.44}$$

This completes our discussion of the updating of Q_k and W_k.

Algorithm 1 Generation of the orthogonal matrix used to update Z_k

$\Psi := I_n$
$\Phi := Z_k$
if there is a non zero component of the tth row of $Z_k\Psi$ besides the first component **then**
  for $i = 2$ to n **do**
    $\Theta := I_n$
    if $\Phi_{t,i} \neq 0$ **then**
      $\mu := 0_{n\times 1}$, $\upsilon := 0_{n\times 1}$
      $\mu_i := 1$, $\upsilon_1 := 1$
      $\mu_1 := \dfrac{\Phi_{t,1}}{\Phi_{t,i}}$, $\upsilon_i := -\dfrac{\Phi_{t,1}}{\Phi_{t,i}}$
      $\mu := \dfrac{\mu}{\|\mu\|}$, $\upsilon := \dfrac{\upsilon}{\|\upsilon\|}$
      Set the first column of Θ to μ and the ith column to υ
    end if
    $\Phi := \Phi\Theta$
    $\Psi := \Psi\Theta$
  end for
end if
Return Ψ

3.3.4 Choosing the direction vector

As noted in section 3.3.1, at each iteration the step d_k is found using either a trust region iteration or an alternative iteration. During both types of

iteration we find a d_k which satisfies the constraints

$$\|d_k\| \leq \Delta_k,$$
$$l \leq x_k + d_k \leq u.$$

In this section we describe the procedures used during both trust region and alternative iterations. We also describe the method used to select t as well as the method used decide which type of iteration to use during the execution of BOBYQA.

Trust region iterations

As noted in section 3.3.1, during trust region iterations d_k is found by minimising the current quadratic model subject to trust region constraints. Specifically d_k is taken to be the solution of the following problem:

$$\begin{aligned} \min_{d_k} \quad & Q_k(x_k + d_k) \\ \text{s.t.} \quad & l \leq x_k + d_k \leq u, \\ & \|d_k\| \leq \Delta_k, \\ & d_k \in \mathbb{R}^n. \end{aligned} \tag{3.45}$$

This problem is solved using an active set version of the truncated conjugate gradient method [126]. The algorithm begins at the centre of the trust region and the procedure is restarted with an enlarged active set if d_k becomes restricted by the bound constraints $l \leq x \leq u$. No indices are removed from the active sets of the subproblems. Additionally, if d_k reaches the boundary of the trust region and the algorithm would terminate with $\|d_k\| < \Delta_k$ then an attempt is made to improve the solution by changing d_k while remaining on the trust region boundary. Further details on this procedure are not given here, the interested reader is referred to [126] and the more extensive description in [68].

Alternative iterations

As noted in section 3.3.1, in alternative iterations d_k is chosen to improve the geometry of the interpolation points. Specifically, d_k is chosen such that numerical errors in the updating of W_k are avoided. This is done by ensuring that the value of σ, in (3.39), is large. We concentrate our efforts on σ since it is the only denominator in (3.40) and (3.44). Now define $\Lambda_t(x)$ to be a quadratic function satisfying the Lagrange interpolation conditions

$$\Lambda_t(y_i) = \delta_{it}, \quad i \in K, \tag{3.46}$$

where δ_{it} is the Kronecker delta. These interpolation conditions do not fully specify Λ_t; the remaining freedom is taken up minimising the Frobenius norm of $\nabla^2\Lambda_t$. It will be shown that the value of σ is closely related to Λ_t. However before describing this relationship further the method used to obtain the coefficients of Λ_t must be given. The method used to construct Λ_t is closely related to the method used to update Q_k since both procedures involve minimising a Frobenius norm. Consider the derivation of W_k given at the beginning of section 3.3.3. Suppose that $D(x)$ is redefined as follows

$$D(x) = \Lambda_t(x),$$

and that the interpolation constraints (3.13) are replaced with

$$D(y_i) = \delta_{it}, \quad i = 1, 2, \ldots, m. \tag{3.47}$$

Now following the same reasoning used at the start of section 3.3.3 and noting that the interpolation conditions (3.47) involve the old interpolation points y_i it can be shown that Λ_t can be written in the form

$$\Lambda_t(x) = p + (x - x_0)^T q + \frac{1}{2}(x - x_0)^T \nabla^2\Lambda_t(x - x_0), \tag{3.48}$$

where

$$\nabla^2\Lambda_t = \sum_{i=1}^{m} \lambda_i \left(y_i - x_0\right)\left(y_i - x_0\right)^T. \tag{3.49}$$

In this case λ_i are the components of $\Omega_k e_t$, p is the first component of $\Xi_k e_t$ and q is given by the remaining components of $\Xi_k e_t$. However, as noted in section 3.3.3, the $(m+1)$th row and column of W_k are not available so instead of using (3.48) Λ_t is expressed in the following form

$$\Lambda_t(x) = (x - x_k)^T \nabla\Lambda_t(x_k) + \frac{1}{2}(x - x_k)^T \nabla^2\Lambda_t(x - x_k),$$

where $\nabla\Lambda_t(x_k) = q + \nabla^2\Lambda_t(x_k - x_0)$. Λ_t can be expressed in this form since $\Lambda_t(x_k) = 0$. This completes our discussion of the method used to construct Λ_t.

We now give a more detailed description of the relationship between σ and Λ_t. Recall that $\tau = e_t^T W_k z$ and that the coefficients λ_i, q and p are the elements of $W_k e_t$. In addition, it is clear from its definition that W_k is

symmetric. Using this information we can rewrite τ as follows

$$\begin{aligned}
\tau &= e_t^T W_k z, \\
&= (W_k^T e_t)^T z, \\
&= (W_k e_t)^T z, \\
&= \left[\begin{array}{c} \lambda \\ \hline p \\ q \end{array}\right]^T z, \\
\tau &= \sum_{i=1}^{m} \lambda_i z_i + p z_{m+1} + \sum_{j=1}^{n} q_j z_{j+m+1}. \qquad (3.50)
\end{aligned}$$

Substituting the elements of z, which are given by (3.35), into (3.50) the following formula for τ is obtained

$$\tau = \frac{1}{2}(x_k + d_k - x_0)^T \nabla^2 \Lambda_t (x_k + d_k - x_0) + (x_k + d_k - x_0)^T q + p.$$

Comparing this to (3.48) it is clear that

$$\tau = \Lambda_t(x_k + d_k).$$

Substituting into (3.39) the following expression for σ is obtained

$$\sigma = \phi\psi + \left\{\Lambda_t(x_k + d_k)\right\}^2.$$

This is the promised relationship between σ and Λ_t. Now it is shown in [123] that we have both $\phi \geq 0$ and $\psi \geq 0$. Using these bounds it is trivial to show that σ is bounded below by $\{\Lambda_t(x_k + d_k)\}^2$.

It was noted at the start of this section that the aim of the alternative iterations is to ensure that σ is substantial. This is done by making the lower bound on σ, given in the previous paragraph, as large as possible. Accordingly the step d_k is found by solving the following problem:

$$\begin{aligned}
\max_{d_k} \quad & |\Lambda_t(x_k + d_k)| \qquad (3.51) \\
\text{s.t.} \quad & l \leq x_k + d_k \leq u, \\
& \|d_k\| \leq \Delta_k, \\
& d_k \in \mathbb{R}^n.
\end{aligned}$$

This problem is solved approximately. In particular $x_k + d_k$ is usually selected to lie on one of the straight lines between x_k and the other interpolation points. If this procedure would prevent the new set of interpolation points from spanning $\mathbb{R}^n$ then d_k is replaced by a Cauchy step. Further details of this procedure are not given here, the interested reader is referred to [126].

Choosing t and the type of iteration

In first part of this section we discuss the methods used to select t during trust region and alternative iterations. Two objectives are considered when choosing t. Firstly, it is desirable to cluster the interpolation points around x_k. In addition, it is necessary to ensure that the value of σ is acceptably large. The reasons for making σ large have been explained previously. The clustering of the interpolation points around x_k improves the accuracy of the quadratic model around x_k. This is important when checking that x_k is a minima, indeed we shall see later that one of the termination conditions of the algorithm is that all of the interpolation points lie within a certain neighbourhood of x_k. It is important to note that the objectives described above must be considered simultaneously; clustering the interpolation points around x_k without ensuring an acceptable value of σ could result in numerical errors. Clustering the interpolation points could also lead to areas of the feasible region being poorly sampled; in BOBYQA this is prevented by the inner trust region radius.

During trust region iterations t is chosen to be the integer in $K\backslash\{s\}$ (s is defined above (3.41)) which maximises the following quantity

$$\max\left[1, \frac{\|y_t - x_k\|^2}{\Delta_k^2}\right]\sigma_t,$$

where σ_t denotes the value of σ if t is chosen. Clearly, when choosing t using this condition both the clustering of the interpolation points and the value of σ are taken into account. During alternative iterations it is deemed that the choice of d_k is a sufficient guaranty that the value of σ will be acceptable. Accordingly, during alternative iterations our main aim when choosing t is clustering the interpolation points. This being the case, during alternative iterations t is set to the integer which satisfies the following equation

$$\|y_t - x_k\| = \max\left\{\|y_i - x_k\| : i \in K\right\}.$$

This completes our discussion of the choice of t.

We now describe the method used to choose between trust region and alternative iterations during the execution of BOBYQA. The decision process is illustrated in Figure 3.1. Figure 3.1 also shows the process used to choose ρ_{k+1}, the inner trust region radius. In this section we only describe the process used to decide whether $\rho_{k+1} < \rho_k$ or $\rho_{k+1} = \rho_k$[1]. The process used to set ρ_{k+1} when $\rho_{k+1} < \rho_k$ will be discussed in section 3.3.6. Now when

[1]The inner trust region radius is never increased so there is no need to consider the case $\rho_{k+1} > \rho_k$.

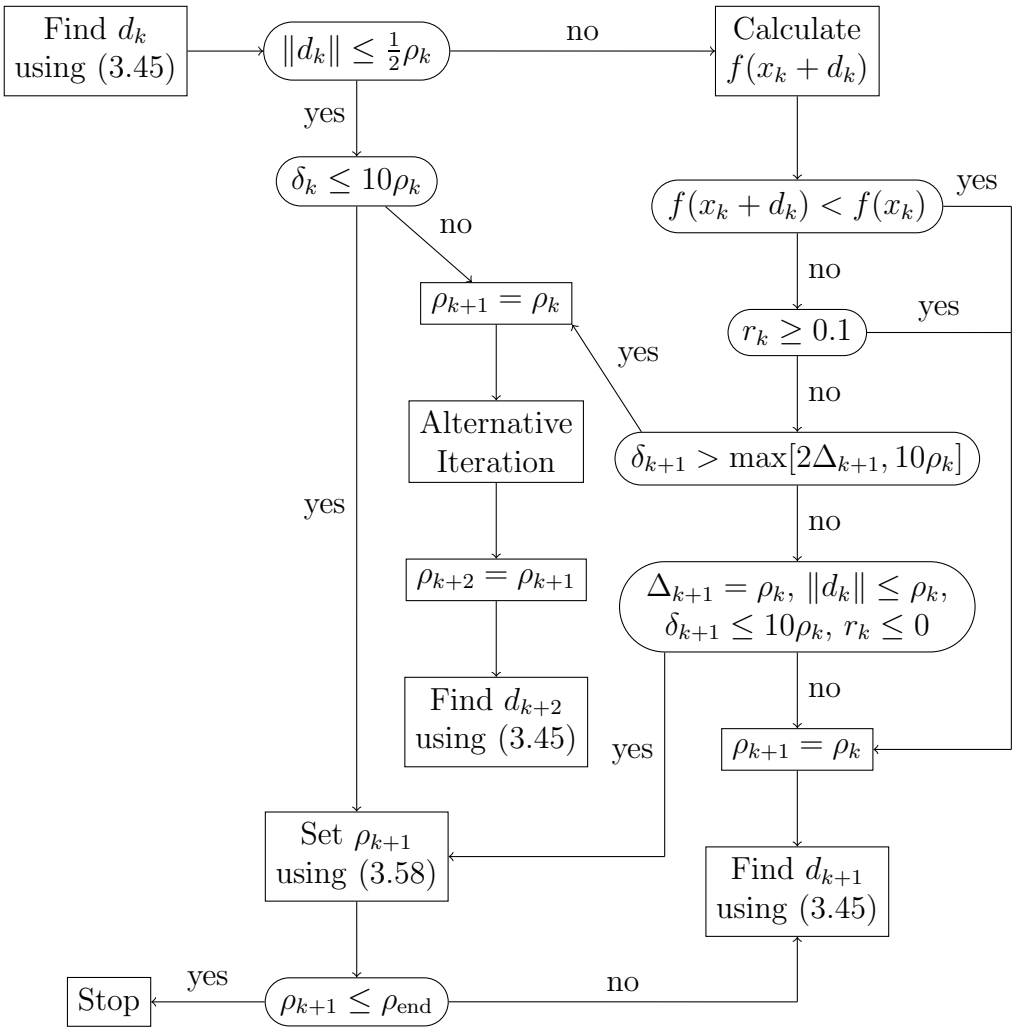


Figure 3.1: A flow chart showing the processes followed to choose between trust region and alternative iterations and to find ρ_{k+1}.

deciding between trust region and alternative iterations measures of both the clustering of the interpolation points and the quality of the quadratic model are required. Towards this end r_k is defined as

$$r_k = \frac{f(x_k) - f(x_k + d_k)}{Q_k(x_k) - Q_k(x_k + d_k)}, \tag{3.52}$$

and δ_k is defined as follows

$$\delta_k = \max \{\|y_i - x_k\| : i \in K\}.$$

Now it is undesirable to take steps that are small relative to the size of the trust region. Accordingly, once a trust region step is generated by solving problem (3.45) the inequality $\|d_k\| \leq 0.5\rho_k$ is evaluated. If this inequality holds d_k is discarded and either an alternative iteration is started or ρ_k is reduced and another trust region iteration is begun. Otherwise the trust region iteration proceeds as normal in which case the value of $f(x_k + d_k)$ will be calculated. Following [126] we refer to the situation where the trust region step is not discarded by saying that $f(x_k+d_k)$ is calculated. If the trust region step is rejected the clustering of the interpolation points is checked using the following condition, $\delta_k \leq 10\rho_k$. If this condition holds then ρ_{k+1} is chosen such that $\rho_{k+1} < \rho_k$ using a method discussed in section 3.3.6. If the new ρ_{k+1} satisfies $\rho_{k+1} \leq \rho_{\text{end}}$ then BOBYQA is stopped and x_k and $f(x_k)$ are returned. If $\rho_{k+1} > \rho_{\text{end}}$ a new trust region step is calculated and the decision process is repeated. On the other hand, if $\delta_k > 10\rho_k$ then $\rho_{k+1} = \rho_k$ and an alternative iteration is used to improve the geometry of the interpolation points. After the alternative iteration a trust region step is calculated with $\rho_{k+2} = \rho_{k+1} = \rho_k$. Now consider the case were $f(x_k + d_k)$ is calculated during a trust region iteration. If $f(x_k + d_k) < f(x_k)$ or $r_k \geq 0.1$ then it is assumed the algorithm is proceeding satisfactorily using the current trust region and set of interpolation points so another trust region step is calculated with $\rho_{k+1} = \rho_k$. If neither of these conditions hold the clustering of the interpolation points is checked using the following condition, $\delta_{k+1} > \max[2\Delta_{k+1}, 10\rho_k]$. If this condition does not hold the next iteration is an alternative iteration with $\rho_{k+1} = \rho_k$ to improve the interpolation points. If the condition does hold the inner trust region radius may need to be reduced for the algorithm to make further progress. This decision is made using the following conditions

$$\begin{aligned} \Delta_{k+1} &= \rho_k, \\ \|d_k\| &\leq \rho_k, \\ \delta_{k+1} &\leq 10\rho_k, \\ r_k &\leq 0. \end{aligned}$$

If these conditions hold ρ_{k+1} is chosen such that $\rho_{k+1} < \rho_k$. If the new ρ_{k+1} satisfies $\rho_{k+1} \leq \rho_{\text{end}}$ then BOBYQA is stopped and x_k and $f(x_k)$ are returned. If $\rho_{k+1} > \rho_{\text{end}}$ a new trust region step is calculated and the decision process is repeated. If any of the four conditions given above are not satisfied a new trust region step is calculated with $\rho_{k+1} = \rho_k$.

3.3.5 The RESCUE procedure

In this section a procedure named RESCUE is described. RESCUE is used to handle a specific type of numerical error that occasionally occurs during the calculation of σ. Numerical errors can cause large inaccuracies in the calculated value of σ. Large reductions in the value of σ are particularly troublesome since, as discussed in section 3.3.4, small values of σ can cause errors in the updating of W_k. Furthermore, it is obvious that negative values of σ would preclude the use of (3.44) to update Z_k. Now both the value of d_k chosen in alternative iterations and the value of t used during the trust region iterations are used to try and keep σ away from zero. However, despite these measures unacceptable losses in accuracy my still occur during the calculation of σ.

Recall that $\sigma = \phi\psi + \tau^2$ and that theoretically $\phi\psi$ is bounded below by zero. In this section we are mainly concerned with numerical errors that cause $\phi\psi$ to be negative. The factorisation of Ω_k ensures that ϕ is positive in practice since we have

$$\begin{aligned} \phi &= (W_k)_{t,t}\,, \\ &= e_t^T Z_k Z_k^T e_t, \\ &= \left\| Z_k^T e_t \right\|^2, \\ \phi &\geq 0. \end{aligned}$$

However, there is no guaranty that ψ will be positive in practice and computational experience shows that rounding errors can produce negative values of ψ. Since ϕ is always positive this makes $\phi\psi$ negative which reduces the value of σ. Occasionally these errors are tolerated as they do not always cause difficulty in the execution of BOBYQA. Quantitatively the current value of σ is rejected if it satisfies the relation

$$\sigma < \frac{1}{2}\tau^2. \tag{3.53}$$

If this criteria is satisfied RESCUE is used to try and obtain a better value of σ. RESCUE proceeds by constructing a new set of interpolation points

whose geometry is better than the set that resulted in the numerical error. The quadratic model is then updated to fit the new set of interpolation points. The procedures used to perform these tasks are described in the subsections below.

Generation of new interpolation points

In this section we discuss the procedure used by RESCUE to construct the new interpolation points. The first step of RESCUE is to discard the current W_k and Z_k matrices, replacements will be generated during the execution of RESCUE. The value of x_0 is changed to $x_0 = x_k$. Since W_k has been discarded this change in x_0 only results in a change in (3.22), the stored form of $\nabla^2 Q_k$. It is desirable to update (3.22) without changing μ_j^k, i.e. (3.22) should be updated by changing G_k. Let G_k^{Resc} denote the updated form of G_k. Noting that $\nabla^2 Q_k$ is invariant under changes of origin and that μ_j^k is unchanged, the following equation can be derived

$$G_k + \sum_{j=1}^{m} \mu_j^k \left(y_j - x_0\right)\left(y_j - x_0\right)^T = G_k^{\text{Resc}} + \sum_{j=1}^{m} \mu_j^k \left(y_j - x_k\right)\left(y_j - x_k\right)^T,$$

$$G_k^{\text{Resc}} = G_k + \sum_{j=1}^{m} \mu_j^k \left[\left(y_j - x_0\right)\left(y_j - x_0\right)^T - \left(y_j - x_k\right)\left(y_j - x_k\right)^T\right].$$

This equation allows us to update (3.22) when the origin is shifted to x_k. The next step of RESCUE is to construct a set of possible replacement interpolation points $\{\gamma_i : i \in K\}$. This is done using a similar procedure to that used to set up the initial interpolation points. Set $\gamma_1 = x_0$ and find non-zero multipliers α_i and β_i with $\alpha_i \neq \beta_i$ using a procedure described below. The remaining $m - 1$ points are set as follows

$$\begin{cases} \gamma_{i+1} = x_0 + \alpha_i e_i, & i = 1, \ldots, n, \\ \gamma_{i+n+1} = x_0 + \beta_i e_i, & i = 1, \ldots, n. \end{cases} \tag{3.54}$$

The values of the multipliers α_i and β_i are given by the following formulae

$$\alpha_i = \begin{cases} \Delta_k, & l \leq x_0 + \Delta_k e_i \leq u, \\ \Delta_k, & (x_k)_i + \Delta_k \leq u_i, \\ -\Delta_k, & (x_k)_i - \Delta_k \geq l_i, \end{cases}$$

$$\beta_i = \begin{cases} -\Delta_k, & l \leq x_0 - \Delta_k e_i \leq u, \\ l_i - (x_k)_i, & ((x_k)_i + \Delta_k \leq u_i) \wedge \left(|l_i - (x_k)_i| \geq 0.5\Delta_k\right), \\ u_i - (x_k)_i, & ((x_k)_i - \Delta_k \geq l_i,) \wedge \left(|u_i - (x_k)_i| \geq 0.5\Delta_k\right), \\ \frac{1}{2}\alpha_i, & \text{otherwise.} \end{cases}$$

New versions of W_k and Z_k are constructed using $\{\gamma_i : i \in K\}$ and the method described in section 3.3.2. RESCUE now proceeds by constructing a new set of interpolation points $\{\widehat{y}_i : i \in K\}$. This set is constructed by replacing as many points as possible in $\{\gamma_i : i \in K\}$ with points from $\{y_i : i \in K\}$ while maintaining an acceptable value of σ. It is desirable to use as many points as possible from $\{y_i : i \in K\}$ in the construction of $\{\widehat{y}_i : i \in K\}$ since the objective function has already been evaluated at each point in $\{y_i : i \in K\}$. More specifically the procedure for constructing $\{\widehat{y}_i : i \in K\}$ proceeds as follows. Initially the new interpolation points are given by $\{\widehat{y}_i = \gamma_i : i \in K\}$. Each point in $\{y_i : i \in K\}$ is assigned a score υ_i which is given by $\upsilon_i = \|y_i - x_k\|$. These scores are used to decide which point in $\{y_i : i \in K\}$ RESCUE should attempt to place in $\{\widehat{y}_i : i \in K\}$. Let υ^* denote the greatest of these scores. Now let ℓ be the integer giving the smallest positive value of υ_ℓ. RESCUE attempts to replace one of the points in $\{\widehat{y}_i : i \in K\}$ with y_ℓ. The point to be removed from $\{\widehat{y}_i : i \in K\}$ is denoted by $\widehat{y}_t$ where t is chosen to be the integer which maximises the following quantity

$$\begin{aligned}\sigma =& \phi\psi + \tau^2,\\ \sigma =& (W_k)_{t,t}\left(\frac{1}{2}\|y_\ell - x_k\|^4 - (z-v)^T W_k (z-v) + 2z_s - v_s\right)\\ &+ \left(e_t^T W_k (z-v)\right)^2. \end{aligned} \tag{3.55}$$

It is inexpensive to find this value of t since both ψ and $W_k(z-v)$ are independent of t. Once both t and ℓ have been selected RESCUE needs to decide whether to replace $\widehat{y}_t$ with y_ℓ. Now let σ_t be the value of σ given by (3.55) with the current t. Then the replacement is made if the following condition is satisfied

$$\sigma_t > 0.01\max\left[\left(e_i^T W_k (z-v)\right)^2 : i \in K\backslash\{s\}\right], \tag{3.56}$$

where the factor 0.01 was chosen through numerical experiment and we recall that s is the integer such that $y_s = x_k$. If (3.56) is satisfied RESCUE replaces $\widehat{y}_t$ with y_ℓ in $\{\widehat{y}_i : i \in K\}$. The matrices W_k and Z_k are then updated using the techniques outlined in section 3.3.3 and the score is set to $\upsilon_\ell = 0$. On the other hand if (3.56) is not satisfied the replacement is not made, the current Z_k and W_k are retained and υ^* is added to υ_ℓ. A new value of ℓ is then chosen since the manipulation of the scores will result in a new smallest positive υ_i. RESCUE then attempts to replace another point in $\{\widehat{y}_i : i \in K\}$ with the new y_ℓ by repeating the procedure described above. This section of the RESCUE procedure is terminated when either $m-1$ replacements have

occurred or all the values of ℓ with $v_\ell > 0$ have been considered without a successful replacement.

Updating the quadratic model

In this section we discuss the method used to update the quadratic model given the set of new points $\{\widehat{y}_i : i \in K\}$ and valid matrices W_k and Z_k. Let $\mathcal{T}$ be the set of integers such that $\widehat{y}_t$, $t \in \mathcal{T}$ is not in the original set $\{y_i : i = 1, \ldots, m\}$. If $\mathcal{T}$ is empty then obviously the current quadratic model already interpolates all of the points in $\{\widehat{y}_i : i \in K\}$. However, if $\mathcal{T} \neq \emptyset$ the quadratic model clearly needs to be updated. When $\mathcal{T} \neq \emptyset$ the objective function has not been evaluated at all of the points in $\{\widehat{y}_i : i \in K\}$ and the first step in updating Q_k is to obtain these objective function values. Now recall from section 3.3.4 that a function $\Lambda_t(x)$ satisfying the Lagrangian interpolation conditions (3.46) can be found using W_k. The fact that Λ_t satisfies (3.46) allows Q_k to be updated as follows. For each $t \in \mathcal{T}$ set

$$Q_k(x) := Q_k(x) + \{f(\widehat{y}_t) - Q_k(\widehat{y}_t)\}\Lambda_t(x). \tag{3.57}$$

For each t, (3.57) ensures that $Q_k(\widehat{y}_t) = f(\widehat{y}_t)$ while having no effect on the value of Q_k at the other interpolation points. That this is the case can easily be seen by considering (3.46). The updating of Q_k for each value of t is completed before moving to the next integer in $\mathcal{T}$. The matrix W_k remains fixed throughout the process of updating Q_k. We now discuss how to perform this updating procedure in practice given that Q_k is in the form of (3.30) with $\nabla^2 Q_k$ given by (3.22) and q_k given by (3.34). We first discuss the updating of $\nabla^2 Q_k$. Since the interpolation points y_j, $j \in \mathcal{T}$ are no longer made use of RESCUE adds $\sum_{j \in \mathcal{T}} \mu_j^k (y_j - x_0)(y_j - x_0)^T$ to G_k and sets $\mu_j^k = 0$, $j \in \mathcal{T}$. $\nabla^2 Q_k$ can now be expressed in the following form

$$\nabla^2 Q_k = G_k + \sum_{j=1}^{m} \mu_j^k \left(\widehat{y}_j - x_0\right)\left(\widehat{y}_j - x_0\right)^T.$$

Then for each $t \in \mathcal{T}$ set

$$\mu_j^k := \mu_j^k + \lambda_j, \quad j = 1, 2, \ldots, m,$$

where λ_j is the jth element of the tth column of Ω multiplied by $f(\widehat{y}_t) - Q_k(\widehat{y}_t)$. That this method correctly updates the Hessian can be seen by considering (3.57) and (3.49), the form of the Hessian of $\Lambda_t(x)$. The updating procedure for q_k makes use of (3.31). Since $x_k = x_0$ the sum over j on the right hand

side is equal to zero. Clearly then q_k is given by $q_k := q_k + q$, where q can be expressed as

$$q = \{f(\widehat{y}_t) - Q_k(\widehat{y}_t)\}\nabla\Lambda_t(x_k).$$

This completes our discussion of the updating of Q_k in RESCUE.

Incorporating RESCUE into BOBYQA

The incorporation of RESCUE into BOBYQA is the focus of this section. Firstly, it should be noted that one or more of the new interpolation points found by RESCUE may satisfy $f(x_k) > f(\widehat{y}_i)$. If this is the case x_k is changed and the change to q_k described in (3.34) is made before proceeding with the operation of BOBYQA.

If RESCUE is called during a trust region iteration then after the execution of RESCUE a new trust region step is generated using the new Q_k. It is possible that values of σ and τ generated after the call of RESCUE will also satisfy (3.53), in which case RESCUE is called again. However, once repeated calls to RESCUE occur it is always asked whether $\mathcal{T}$ was empty during the previous call to RESCUE. If $\mathcal{T}$ was empty then further calls to RESCUE will have no effect and an error is returned. Numerical experience shows that this error is very rare. RESCUE will only ever be called during trust region iterations when a trust region step satisfying $\|d_k\| \geq \frac{1}{2}\rho_k$ is generated. When $\|d_k\| \geq \frac{1}{2}\rho_k$ condition (3.53) is checked before $f(x_k + d_k)$ is calculated; since RESCUE will typically change Q_k, which in turn changes d_k, the value of $f(x_k + d_k)$ before RESCUE is called will not be required.

Two different branches can be taken if RESCUE is called during an alternative iteration. If $\mathcal{T} = \emptyset$ the alternative iteration is restarted using the new W_k. If RESCUE is called again during the restarted alternative iteration an error is returned, as before this error is very rare. If $\mathcal{T} \neq \emptyset$ a trust region iteration is begun with the calculation of a new trust region step using the Q_k returned by RESCUE. Similarly to the trust region case, condition (3.53) is checked before $f(x_k + d_k)$ is calculated during the alternative iteration. We note that the preceding description of the inclusion of RESCUE into BOBYQA is not included in Figure 3.1.

3.3.6 Further details of BOBYQA

This section contains the remaining information required to complete the description of BOBYQA. The first subsection contains the description of the trust region management procedure. The second subsection contains an outline of the method used to perform the occasional shift of origin. The

final section details two methods used to improve the convergence of the algorithm.

Trust region management

As was mentioned previously BOBYQA maintains two trust region radii; an outer radius Δ_k and an inner radius ρ_k. The inner trust region radius is used to keep the interpolation points apart by preventing steps that are too small relative to the size of the outer trust region. The inner trust region radius is also used in the stopping condition of the algorithm; the algorithm is stopped when $\rho_k \leq \rho_{\text{end}}$. In this section we outline the procedures used to find Δ_{k+1} and ρ_{k+1} from Δ_k and ρ_k respectively. The parameters used in the updating procedures were chosen through numerical experiment.

The situations in which $\rho_{k+1} = \rho_k$ along with those in which $\rho_{k+1} < \rho_k$ is required are shown in Figure 3.1 and discussed in section 3.3.4. The arguments will not be repeated here. However, a formula to calculate ρ_{k+1} when $\rho_{k+1} < \rho_k$ has not been given. The required reduction formula can be expressed as follows

$$\rho_{k+1} = \begin{cases} \rho_{\text{end}}, & \rho_k \leq 16\rho_{\text{end}}, \\ \sqrt{\rho_k \rho_{\text{end}}}, & 16\rho_{\text{end}} < \rho_k \leq 250\rho_{\text{end}}, \\ 0.1\rho_k, & \rho_k > 250\rho_{\text{end}}. \end{cases} \tag{3.58}$$

This formula is reduces ρ_k by a factor of ten unless there will only be one or two more reductions until $\rho_k = \rho_{\text{end}}$. Further information on the specific form of (3.58) can be found in [124]. This completes the discussion of the management of the inner trust region.

If the kth iteration is an alternative iteration $\Delta_{k+1} = \Delta_k$. If the kth iteration is a trust region iteration that calculates $f(x_k + d_k)$ then Δ_{k+1} is given by

$$\Delta_{k+1} = \begin{cases} \min\left[\frac{1}{2}\Delta_k, \|d_k\|\right], & r_k \leq 0.1, \\ \max\left[\frac{1}{2}\Delta_k, \|d_k\|\right], & 0.1 < r_k \leq 0.7, \\ \max\left[\frac{1}{2}\Delta_k, 2\|d_k\|\right], & r_k > 0.7. \end{cases} \tag{3.59}$$

This formula utilises the fact that the larger r_k is the better the quadratic model is working. Accordingly, when r_k is small the size of trust region is reduced in an attempt to improve the accuracy of the quadratic model. For larger r_k the size of the trust region is increased to allow a more efficient exploration of the solution space. Further details on this formula, as well as the other trust region management formulae in this section, are given in [124]. A refinement to (3.59), which also applies to the following formulae

for calculating Δ_{k+1}, is that Δ_{k+1} is set to ρ_k if $\Delta_{k+1} \leq 1.5\rho_k$. When a trust region step satisfying $\|d_k\| < \frac{1}{2}\rho_k$ is produced then Δ_{k+1} is given by

$$\Delta_{k+1} = \min\left[\frac{1}{10}\Delta_k, \frac{1}{2}\delta_k\right]. \tag{3.60}$$

When a situation arises in which $\rho_{k+1} < \rho_k$ then Δ_{k+1} is given by

$$\Delta_{k+1} = \max\left[\frac{1}{2}\rho_k, \rho_{k+1}\right]. \tag{3.61}$$

We note that the conditions which cause $\rho_{k+1} < \rho_k$ can occur with the conditions that result in the use of (3.59) or (3.60). In this case (3.61) is used instead of (3.59) or (3.60). A final refinement to the management of the outer trust region radius is that it may be temporarily reduced before alternative iterations. If $\delta_k < 10\Delta_k$ before an alternative iteration then Δ_k is reduced using the formula $\Delta_k = \max[0.1\delta_k, \rho_k]$. The previous value of Δ_k is restored after the alternative iteration. This completes our discussion of the management of the outer trust region.

Shifts of origin

It is important for numerical accuracy that the distance between x_0 and x_k does not become too large. For this reason the origin is shifted occasionally to $x_0 = x_k$. Specifically the origin is shifted to x_k when a step d_k is generated such that

$$\left(\|d_k\| \geq \frac{1}{2}\rho_k\right) \wedge \left(\|d_k\|^2 \leq 10^{-3}\|x_k - x_0\|^2\right).$$

The procedure used to shift the origin is derived in [123]; we give the details of the procedure here, the reader interested in the derivation is referred to [123]. Let Γ be an $n \times m$ matrix whose columns are given by

$$\Gamma e_j = \left\{s^T(y_i - x_{\text{av}})\right\}(y_i - x_{\text{av}}) + \frac{1}{4}\|s\|^2 s,$$

where $s = x_k - x_0$ is the magnitude of the shift and $x_{\text{av}} = \frac{1}{2}(x_0 + x_k)$. The updated W_k matrix is then given by

$$W_k := \left[\begin{array}{c|c} I & 0 \\ \hline \Gamma & I \end{array}\right] W_k \left[\begin{array}{c|c} I & \Gamma^T \\ \hline 0 & I \end{array}\right].$$

This will have no effect on Ω_k so the factorisation Z_k does not need to be changed. The quadratic model is updated by adding the following symmetric

matrix to G_k

$$\left\{\sum_{j=1}^{m}\mu_j y_j - x_{\rm av}\sum_{j=1}^{m}\mu_j\right\}s^T + s\left\{\sum_{j=1}^{m}\mu_j y_j - x_{\rm av}\sum_{j=1}^{m}\mu_j\right\}^T.$$

This completes our description of the origin shifting procedure.

Methods to increase the speed of convergence

The final details required for the description of BOBYQA are two refinements used to speed the convergence of the algorithm in specific situations. The first refinement is used when the quadratic model is very successful. The second refinement applies when the elements of the Hessian of the quadratic model are much too large. We note that these refinements are not included in Figure 3.1.

The first refinement is a procedure that can be used to give $\rho_{k+1} < \rho_k$ when $\|d_k\| < \frac{1}{2}\rho_k$ and $\delta_k > 10\rho_k$. This procedure is necessary as it can happen that a previous quadratic model has generated an x_k such that $\|x_k - x^*\|$ is much smaller than ρ_k, here x^* is the optimal solution. Then if the quadratic models remain accurate representations of the objective function the trust region steps generated by solving problem (3.45) may all be rejected by the condition $\|d_k\| \geq \frac{1}{2}\rho_k$, until ρ_k is reduced.

The technique used to overcome this problem uses the following estimate of the accuracy of the quadratic model

$$\varepsilon_{\max} = \max\left\{|f(x_l + d_l) - Q_l(x_l + d_l)| : l \in \{k-3, k-2, k-1\}\right\},$$

where $f(x_l+d_l)$, $l \in \{k-3, k-2, k-1\}$ are the three most recently calculated values of the objective function. The value of $\varepsilon_{\max}$ is said to be usable if RESCUE was not called in the previous three iterations and if $\|d_l\| \leq \rho_k$, $l \in \{k-3, k-2, k-1\}$. Obviously $\varepsilon_{\max}$ will also not be usable during the first three iterations. Now whenever a trust region step is generated which satisfies $\|d_k\| < \frac{1}{2}\rho_k$ and $\delta_k > 10\rho_k$ then the existence of a usable $\varepsilon_{\max}$ is checked. If $\varepsilon_{\max}$ is found to be usable we set $\rho_{k+1} < \rho_k$ if and only if the two tests outlined in the following paragraphs suggest that $x^* \in \{x : \|x - x_k\| \leq \rho_k\}$.

The first test checks whether a move to the trust region boundary is likely to reduce the objective function value. This is done using the information obtained when solving problem (3.45) using the truncated conjugate gradient procedure. Let $\mathcal{S}$ be the set of search directions such that the steps taken along the directions were not restricted by the bounds, $l \leq x \leq u$. Now it is necessary to examine the effect that a move from $x_k + d_k$ to $x_k + d_k + \theta s$, $s \in$

$\mathcal{S}$, $\theta \in \mathbb{R}$, where $\|d_k + \theta s\| = \rho_k$, has on the value of Q_k. A Taylor series expansion gives

$$Q_k(x_k + d_k + \theta s) = Q_k(x_k + d_k) + (\theta s)\, \nabla Q_k(x_k + d_k) + \frac{1}{2}\, (\theta s)^T\, \nabla^2 Q_k\, (\theta s)\,.$$

From the conjugacy property of the directions in $\mathcal{S}$ it is clear that $s\nabla Q_k(x_k + d_k) = 0$, $\forall s \in \mathcal{S}$. Therefore

$$Q_k(x_k + d_k + \theta s) = Q_k(x_k + d_k) + \frac{1}{2}\theta^2 s^T \nabla^2 Q_k s. \tag{3.62}$$

Since $\|d_k + \theta s\| = \rho_k$ and $\|d_k\| < \frac{1}{2}\rho_k$ the triangle inequality can be used to show that $\|\theta s\| > \frac{1}{2}\rho_k$. Substituting this into (3.62) gives

$$Q_k(x_k + d_k + \theta s) - Q_k(x_k + d_k) > \frac{1}{8}\rho_k^2 \|s\|^{-2} s^T \nabla^2 Q_k s.$$

Clearly, changes in Q_k provide an indication of the changes in the objective function. Accordingly, if the inequalities

$$\varepsilon_{\max} \leq \frac{1}{8}\rho_k^2 \|s\|^{-2} s^T \nabla^2 Q_k s, \quad \forall s \in \mathcal{S},$$

are satisfied it is deemed unlikely that a move from $x_k + d_k$ to the trust region boundary will reduce the value of the objective function.

The second test determines whether it is likely that a component of $x_k + d_k$ which is restricted by the bound constraints needs to be moved away from the bounds. A set $\mathcal{V}$ of multiples of the coordinate vectors is formed as follows. Include $\rho_k e_i$ or $-\rho_k e_i$ in $\mathcal{V}$ if the ith component of $x_k + d_k$ is equal to l_i or u_i respectively. It is deemed likely that ρ_k should be reduced if the differences

$$Q_k(x_k + d_k + v) - Q_k(x_k + d_k), \quad \forall v \in \mathcal{V},$$

are greater than $\varepsilon_{\max}$. In addition the second derivatives are ignored if the first order part of the difference is sufficiently large. The second test is then given by

$$\varepsilon_{\max} \leq \max\left[v^T \nabla Q_k(x_k + d_k),\; v^T \nabla Q_k(x_k + d_k) + \frac{1}{2} v^T \nabla^2 Q_k v\right], \quad \forall v \in \mathcal{V}.$$

If both of these tests hold then ρ_{k+1} is set using (3.58).

The second refinement used to speed the convergence of BOBYQA is designed to avoid the inefficiencies that occur when the elements of $\nabla^2 Q_k$ are too large. This can happen if the initial point is far from a local minimum

and in a region in which the value of f increases exponentially. The values of the elements of $\nabla^2 Q_1$ will be large in this case and an additional technique may be required to reduce them for later iterations since generally the change $\|\nabla^2 Q_{k+1} - \nabla^2 Q_k\|_F$ is made as small as possible subject to the interpolation conditions. To overcome this problem Q_{k+1} is compared to an alternative quadratic model Q_{k+1}^{alt} which is defined to be the quadratic model which satisfies the interpolation conditions and has its remaining degrees of freedom taken up by minimizing $\|\nabla^2 Q_{k+1}^{\text{alt}}\|_F$. Now define the operator $\mathcal{P}$ such that the value of $\mathcal{P}\nabla Q_{k+1}(x_{k+1})$ will be a vector in $\mathbb{R}^n$ whose ith component is given by

$$\left(\mathcal{P}\nabla Q_{k+1}(x_{k+1})\right)_i = \begin{cases} \min\left[0, \left(\nabla Q_{k+1}(x_{k+1})\right)_i\right], & (x_{k+1})_i = l_i, \\ \left(\nabla Q_{k+1}(x_{k+1})\right)_i, & l_i \leq (x_{k+1})_i \leq u_i, \\ \max\left[0, \left(\nabla Q_{k+1}(x_{k+1})\right)_i\right], & (x_{k+1})_i = u_i. \end{cases}$$

Now $\|\mathcal{P}\nabla Q_{k+1}(x_{k+1})\|$ is expected to be much smaller than $\|\mathcal{P}\nabla Q_{k+1}^{\text{alt}}(x_{k+1})\|$ when BOBYQA is making progress in reducing f. However, when the elements of $\nabla^2 Q_{k+1}$ are too large this relation tends to be reversed. Therefore, when the condition

$$\|\mathcal{P}\nabla Q_{k+1}^{\text{alt}}(x_{k+1})\| \leq 0.1\|\mathcal{P}\nabla Q_{k+1}(x_{k+1})\|, \tag{3.63}$$

holds for three consecutive trust region steps we replace Q_{k+1} by Q_{k+1}^{alt}. Condition (3.63) is checked during trust region iterations that calculate $f(x_k+d_k)$ after the value of $f(x_k+d_k)$ has been calculated and the updating procedure described in section 3.3.3 has been performed. This completes our description of the BOBYQA algorithm.

3.4 Derivative free optimization of mixed integer problems

Relatively little work has been done on developing derivative free solution methods for mixed integer programs. As with the methods developed for continuous problems, the work that has been done can be divided into metaheuristics, surrogate optimization and local sampling.

3.4.1 Metaheuristics and surrogate optimization

Most of the existing methods for solving derivative free MINLP problems are metaheuristics. These methods suffer from the same drawbacks mentioned

in section 3.2.1; lack of deterministic convergence proofs and a large number of function evaluations. Indeed, in the mixed integer case even proofs of convergence in probability are rare. As in the continuous case the literature is large and we only give some examples of the most common approaches. Algorithms based in simulated annealing are developed in [42, 131, 156, 166]. Proofs of convergence in probability are given in [131, 156]. Particle swarm algorithms for MINLP are developed in [84, 162]. Evolutionary approaches are developed in [47, 51, 100, 163]. It appears that no convergence results have been proven for particle swarm and evolutionary algorithms. The basic details of all three of these approaches are similar to those given in section 3.2.1; for more detailed descriptions of each algorithm the interested reader is referred to the relevant references.

Two surrogate methods have been developed for solving MINLPs. The first was developed by Hemker [71]. The proposed approach uses functional surrogate models, more specifically it uses a detrended kriging model with the form

$$s(x) = \beta + z(x), \tag{3.64}$$

where $s(x)$ is the surrogate model, $\beta \in \mathbb{R}$ and $z(x)$ is a stationery Gaussian random function with zero mean and a covariance of the form

$$\mathrm{Cov}[z(\tilde{x}), z(\bar{x})] = \sigma_z^2 \prod_{j=1}^{n} e^{-\theta_j(\tilde{x}_j - \bar{x}_j)^2}, \tag{3.65}$$

where $\tilde{x}$ and $\bar{x}$ are two of the points being used to construct the surrogate model. The parameters θ_j, β and σ_z are calculated using a maximum likelihood approach. Once the surrogate model is constructed it is minimised using a MINLP Branch and Bound algorithm to ensure that the solution returned is integer feasible. The next point at which the objective function is evaluated is the solution returned by the Branch and Bound algorithm, unless the point lies within an ε-ball of a previously evaluated point. In that case the new point is chosen by minimising the mean square error of the surrogate model. This problem is also solved using a Branch and Bound algorithm which ensures that the new point is integer feasible. Once the objective function has been evaluated at the new point the surrogate model is updated using all of the previous function evaluations. The algorithm is a heuristic with no convergence results being presented. This clearly makes it unsuitable for use in situations when deterministic convergence guaranties are desired.

A second surrogate optimization approach named SO-MI has been developed in Müller et al. [116]. SO-MI uses a radial basis function model with

the following form

$$s(x) = \sum_i \lambda_i \|x - \bar{x}_i\|^3 + \beta^T x + \alpha, \tag{3.66}$$

where $\beta \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$, $\bar{x}_i$ are points at which the objective function has been evaluated and the sum is over all of the points used to construct the surrogate model. The parameters λ_i, β and α are found by solving a linear system which insures that $s(x)$ interpolates the points used to construct the model. In this algorithm the surrogate model is not used directly to find the next point to be evaluated, rather four possible points to be evaluated are randomly generated. The objective function is only evaluated at one of the four points. This point is chosen to be the point which maximises a weighted sum of two scores; the first score is based on the distance between the random point and the previously evaluated points and the second score is based on the value of $s(x)$ at the random point. Once the objective function has been evaluated at the new point the surrogate model is updated. Convergence in probability can be proven for SO-MI.

3.4.2 Local sampling

All of the local sampling methods that have been developed for derivative free MINLP are of the directional direct search type. The methods that have been developed in this area at this point can be found in [1, 2, 3, 17, 102, 105]. Importantly deterministic proofs of convergence to local minima have been provided for each of the algorithms. All of the algorithms follow the same basic approach; the algorithms alternate between local minimization of the continuous variables, with the discrete variables held fixed and local minimization of the discrete variables, with the continuous variables held fixed. The differences between the algorithms mainly lie in the methods used to perform the continuous optimization. We now give a slightly more detailed description of each of the algorithms referenced above.

The first mixed integer direct search algorithm was proposed in [17]. The algorithm can solve problems with bound constraints on the continuous variables and can handle categorical variables. It uses a GPS algorithm to search the continuous variables and searches the discrete variables by evaluating all points in a user defined discrete neighbourhood. The polling step of the algorithm is divided into three stages; a continuous poll with the discrete variables held fixed, a discrete poll with the continuous variables held fixed and an extended poll. The extended poll step performs a continuous poll around each point found during the discrete poll which satisfies $f(x) < f(x_k) + \xi$, where $\xi > 0$ is a user defined threshold. The procedure to be used in the search

step is not specified. Paper [17] is theoretical; algorithms and convergence proofs are presented but only two illustrative examples are considered, one of which is of the form of problem (1.1).

The algorithm developed in [2] and [3] extends the method developed in [17] to handle general constraints on the continuous variables. This is done using a filter approach, as was discussed in section 3.2.3. The remaining details of the algorithm are essentially the same as [17]. In [3] a modification of the algorithm is proposed to allow any available derivative information to be used. In [141] the algorithm is extended to handle problems with stochastic objective functions. The algorithm from [2] and [3] has been implemented in the software package NOMAD (Nonsmooth Optimization by Mesh Adaptive Direct Search) [6, 90].

In [105] an algorithmic model for direct search methods is presented. The procedures used to perform the continuous and discrete local searches are not specified. However, restrictions on their form, which allow convergence to be proven, are given. These restrictions do not require the points sampled by the continuous search to lie on a mesh, rather a sufficient decrease condition must be imposed. The algorithmic model can handle general continuous constraints and categorical variables. As was the case in the previous two algorithms this method also makes use of continuous, discrete and extended poll steps. Paper [105] is theoretical; algorithms and convergence proofs are presented but only one example, which is of a form more general than problem (1.1), is considered.

The algorithm developed in [1] can solve problems with categorical variables and general continuous constraints. It uses MADS to search the continuous variables and searches the discrete variables by evaluating all points in a user defined discrete neighbourhood. The constraints are handled using an extreme barrier approach. This algorithm also uses continuous, discrete and extended poll steps. We note that [1] is purely theoretical; algorithms and convergence proofs are presented but there are no computational results.

Three similar direct search algorithms are developed in [102]. The algorithms solve problems with bound constraints and integer variables. Unlike the previous algorithms they cannot solve problems with categorical variables. The algorithm developed by Lucidi and Sciandrone [106] (see section 3.2.3 for details) is used to search the continuous variables. The method used to explore the discrete variables is the difference between the algorithms proposed in [102]. The first algorithm searches the discrete variables using a modification of the method in [106], this modification includes a sufficient decrease condition. The second algorithm removes the sufficient decrease condition from the discrete search used in the first algorithm. Neither of the first two algorithms makes use of an extended polling step. The third

algorithm uses a similar discrete search procedure to that used in the first algorithm. However, if a point with sufficient decrease is not found during the discrete polling step an extended polling step is used. As before the extended polling step is used to explore the neighbourhood of any points satisfying $f(x) < f(x_k) + \xi$ that were found during the discrete poll. Extensive computational results comparing the effectiveness of the three algorithms with NOMAD are presented in [102].

The main problem with these approaches is that the continuous and discrete variables are considered separately. This prevents them from taking into account the combined behaviour of the objective function. Accordingly information which might be helpful in reducing the objective function is not taken into account. We also note that no work has been done on developing a trust region based derivative free approach. The extension of the BOBYQA trust region method to the mixed integer case, in chapter 5 of this thesis, is intended to overcome these deficiencies.

3.5 Conclusion

In this chapter we have presented a review of derivative free optimization for continuous and mixed integer problems. The literature on continuous derivative free optimization is large and we mainly focused on the areas of continuous derivative free optimization that are used in the mixed integer case. In section 3.3 we gave a detailed review of the BOBYQA algorithm; this review is necessary since the method developed in chapter 5 is based on BOBYQA. From our review of the mixed integer derivative free literature it is clear that, in their current state, metaheuristics and surrogate optimization procedures are inadequate for problems where a deterministic guaranty of convergence to a minima is required. The various extensions of direct search procedures to the mixed integer case are more appropriate since they all have deterministic proofs of convergence. However, a weakness of these algorithms is that they treat the discrete and continuous variables separately. To overcome these limitations we develop a trust region based mixed integer method based on BOBYQA in chapter 5.

Chapter 4

Methods for Solving Non-Convex MIQPs

In this chapter we develop a number of methods for solving non-convex MIQPs. The methods developed involve preprocessing procedures and modifications of some of the solution methods described in chapter 2. Each of the methods discussed has been developed for problems whose Hessians have a specific structure. Combined these methods can solve any problem with the form of problem (1.2). Specifically we consider the following three classes of Hessians, listed here in order of increasing difficulty:

1. The n_cth principal leading submatrix is positive semidefinite. Solution methods for this class are developed in section 4.3.

2. The n_cth principal leading submatrix is invertible. Solution methods for this class are developed in section 4.2.

3. The n_cth principal leading submatrix is singular. Solution methods for this class are developed in section 4.4.

The methods developed for each class of Hessian can be used to solve problems whose classes are higher in the list but not those lower than itself. For example, the methods developed for class 2 can also be used to solve problems in class 1 but not those in class 3. However the methods developed become less efficient as you move further down the list. All of the solution approaches are based on a linear transformation whose basic form is developed in section 4.1. Specific transformations for problems in classes 2, 1 and 3 are developed in sections 4.2, 4.3 and 4.4 respectively. Concluding remarks are made in section 4.5.

4.1 The linear transformation

In this section we develop the general form of a linear transformation which can be applied to mixed integer problems. In the following sections this general form is used to derive transformations useful in solving problems whose Hessians have a specific structure. In deriving the transformation we make use of the fact that H can be expressed in the following form

$$H = \begin{bmatrix} H_{cc} & H_{cd} \\ H_{cd}^T & H_{dd} \end{bmatrix}, \tag{4.1}$$

where $H_{cc} \in \mathcal{S}^{n_c}$, $H_{dd} \in \mathcal{S}^{n_d}$ and $H_{cd} \in \mathbb{R}^{(n_c,n_d)}$. Now, consider a matrix V with the following form

$$V = \begin{bmatrix} U_{cc} & U_{cd} \\ 0 & U_{dd} \end{bmatrix}, \tag{4.2}$$

where $U_{cc} \in \mathbb{R}^{(n_c,n_c)}$ and $U_{dd} \in \mathbb{R}^{(n_d,n_d)}$ are arbitrary invertible matrices and $U_{cd} \in \mathbb{R}^{(n_c,n_d)}$ is an arbitrary matrix. Any matrix with this form is invertible [74]. Now under the linear transformation $x = Vy$ problem (1.2) is equivalent to the following problem:

$$\begin{aligned} \min_{y} \quad & h(Vy) = \frac{1}{2} y^T V^T H V y + g^T V y \\ \text{s.t.} \quad & AVy \leq b, \\ & DVy = e, \\ & l \leq Vy \leq u, \\ & y = \left[y_c^T,\, y_d^T\right]^T, \\ & U_{dd} y_d \in \mathbb{Z}^{n_d}, \\ & U_{cc} y_c + U_{cd} y_d \in \mathbb{R}^{n_c}. \end{aligned} \tag{4.3}$$

We can apply this linear transformation since V is always invertible. We need to simplify the integral constraint $U_{dd} y_d \in \mathbb{Z}^{n_d}$; we therefore restrict U_{dd} to be some unimodular matrix. A matrix is unimodular if it is integral and has a determinant of ± 1 [167]. Now since $|U_{dd}| = \pm 1$ both U_{dd} and U_{dd}^{-1} are integral and it is obvious that

$$U_{dd} y_d \in \mathbb{Z}^{n_d} \Leftrightarrow y_d \in \mathbb{Z}^{n_d}.$$

It is also obvious that

$$U_{cc} y_c + U_{cd} y_d \in \mathbb{R}^{n_c} \Leftrightarrow y_c \in \mathbb{R}^{n_c}.$$

Problem (4.3) now takes the following form:

$$\begin{aligned} \min_{x} \quad & h(Vy) = \frac{1}{2} y^T V^T H V y + g^T V y \\ \text{s.t.} \quad & AVy \leq b, \\ & DVy = e, \\ & l \leq Vy \leq u, \\ & y = \left[y_c^T,\, y_d^T \right]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}. \end{aligned} \tag{4.4}$$

We now consider the quadratic term, $y^T V^T H V y$, in problem (4.4). Substituting (4.1) and (4.2) into the quadratic term we obtain the following expression

$$\begin{aligned} y^T V^T H V y = & y_c^T U_{cc}^T H_{cc} U_{cc} y_c + 2 y_d^T \left(U_{cd}^T H_{cc} U_{cc} + U_{dd}^T H_{cd}^T U_{cc} \right) y_c \\ & + y_d^T \left(U_{cd}^T H_{cc} U_{cd} + U_{cd}^T H_{cd} U_{dd} + U_{dd}^T H_{cd}^T U_{cd} \right. \\ & \left. + U_{dd}^T H_{dd} U_{dd} \right) y_d. \end{aligned} \tag{4.5}$$

In the following sections we shall use the remaining freedom in the elements of V to simplify (4.5). The choice of elements will depend on the structure of H_{cc}.

4.2 Approach used when H_{cc} is invertible

In this section we consider the case when H_{cc} is invertible. The discussion is divided into two parts; in section 4.2.1 we describe the linear transformation used for these problems while in section 4.2.2 we discuss a Branch and Bound algorithm that can be used to solve the transformed problem. The transformation is chosen in such a way that the Branch and Bound algorithm can solve the transformed problem faster than the original problem. Computational results showing that the transformation has the desired effect are given in chapter 6.

4.2.1 Choosing the elements of V

When H_{cc} is invertible we use the freedom in our choice of the elements of V to remove as many of the bilinear terms in (4.5) as possible. This aim is chosen since the available solution approaches for problems of this type, such as SCIP and Baron, make use of Branch and Bound algorithms. When constructing the lower bounding problems in the Branch and Bound tree each bilinear term in the objective function is underestimated using the convex

envelopes (2.15) or (2.16). Therefore, each bilinear term adds one additional variable and two constraints to the lower bounding problem. Reducing the number of bilinear terms will decrease the size of the lower bounding problems which should improve the efficiency of the Branch and Bound algorithm. Additionally, the convex envelope of a sum of terms is not necessarily equal to the sum of the convex envelopes of the terms [75]. Reducing the number of terms that are underestimated may result in a tighter underestimating problem.

Before proceeding with our description of the method used to achieve this reduction in the number of bilinear terms we define the following sets of indices

$$J = \{1, 2, \ldots, n_c\},$$
$$I = \{n_c + 1, n_c + 2, \ldots, n\}.$$

Now, to remove the desired bilinear terms from (4.5) we need the following equation to be satisfied

$$U_{cd}^T H_{cc} U_{cc} + U_{dd}^T H_{cd}^T U_{cc} = 0. \tag{4.6}$$

We know that U_{cc} is invertible so (4.6) can be expressed as

$$H_{cc} U_{cd} = -H_{cd} U_{dd}. \tag{4.7}$$

In (4.7) H_{cc} and H_{cd} are known constant matrices, U_{cd} is unknown and U_{dd} must be unimodular. If we find some criteria to fix U_{dd} as a constant unimodular matrix we will have a system of $n_c \times n_d$ equations in $n_c \times n_d$ unknowns where the unknowns are the elements of U_{cd}. If H_{cc} is not invertible (4.7) will have no solution; this is the reason that we consider the cases when H_{cc} is and is not invertible separately.

There is still a fairly large amount of freedom in our choice of U_{dd} since at this point the only restriction on this portion of V is that it should be unimodular. For example U_{dd} can be any integral upper or lower triangular matrix with ± 1 on the diagonal. We fix the remaining freedom in U_{dd} by using it to reduce the bounds on the integer variables in the transformed problem. This allows us to construct tighter underestimators of the non-convex terms in $h(Vy)$ [30] and reduces the number of possible combinations of integer variables. The tighter underestimators are useful when solving problem (4.4) with a Branch and Bound method. The bounds on the variables in our transformed problem are given by

$$y_i^L \leq y_i \leq y_i^U,$$

where y_i^L and y_i^U can be found as follows [167]. Let

$$y_i^{L0} = \min_x \left\{ \left(V^{-1}\right)_i x : Ax \leq b, Dx = e, l \leq x \leq u \right\},$$
$$y_i^{U0} = \max_x \left\{ \left(V^{-1}\right)_i x : Ax \leq b, Dx = e, l \leq x \leq u \right\},$$

where $(V^{-1})_i$ is the ith row of V^{-1}. Denote the feasible region of these problems by Ω_q. This is also the feasible region of problem (1.2). The bounds on the variables in the transformed problem are given by

$$y_i^L = \begin{cases} y_i^{L0}, & i \in J, \\ \lceil y_i^{L0} \rceil, & i \in I, \end{cases} \tag{4.8}$$

$$y_i^U = \begin{cases} y_i^{U0}, & i \in J, \\ \lfloor y_i^{U0} \rfloor, & i \in I. \end{cases} \tag{4.9}$$

We now discuss the method proposed to fix the elements of U_{dd} to minimize the range of the bounds on the integer variables. Each row of U_{dd}^{-1} can be found by solving the following problem:

$$\underset{\left(U_{dd}^{-1}\right)_i}{\operatorname{argmin}} \left\{ \max_{x_d} \left[\left(U_{dd}^{-1}\right)_i x_d : x \in \Omega_q \right] - \min_{x_d} \left[\left(U_{dd}^{-1}\right)_i x_d : x \in \Omega_q \right] \right\} \tag{4.10}$$

$$\text{s.t.} \quad U_{dd} \text{ is unimodular,} \tag{4.11}$$

where $\left(U_{dd}^{-1}\right)_i$ is the ith row of U_{dd}^{-1}. We wish to solve problem (4.10) for $i = 1, \ldots, n_d$; the ith solution of problem (4.10) gives us the ith row of U_{dd}^{-1}. The constraint that U_{dd} must be unimodular makes problem (4.10) very difficult to solve. However, problem (4.10) can be solved analytically when only bound constraints are present in the original problem. In this case problem (4.10) takes the following form:

$$\underset{\left(U_{dd}^{-1}\right)_i}{\operatorname{argmin}} \left\{ \max_{x_d} \left[\left(U_{dd}^{-1}\right)_i x_d : l \leq x \leq u \right] - \min_{x_d} \left[\left(U_{dd}^{-1}\right)_i x_d : l \leq x \leq u \right] \right\}$$

$$\text{s.t.} \quad U_{dd} \text{ is unimodular.}$$

The linear programs in the objective function are now separable so we have:

$$\underset{\left(U_{dd}^{-1}\right)_i}{\operatorname{argmin}} \left\{ \sum_{j=1}^{n_d} \left(\max_{x_j} \left[\left(U_{dd}^{-1}\right)_{i,j} (x_d)_j : l_j \leq (x_d)_j \leq u_j \right] \right. \right.$$
$$\left. \left. - \min_{x_j} \left[\left(U_{dd}^{-1}\right)_{i,j} (x_d)_j : l_j \leq (x_d)_j \leq u_j \right] \right) \right\} \tag{4.12}$$

$$\text{s.t.} \quad U_{dd} \text{ is unimodular,}$$

where $\left(U_{dd}^{-1}\right)_{i,j}$ is the jth element of the ith row of U_{dd}^{-1}. Clearly the solution of jth maximization problem in the objective function of problem (4.12) is

$$(x_d)_j^* = \begin{cases} u_j, & \left(U_{dd}^{-1}\right)_{i,j} \geq 0, \\ l_j, & \left(U_{dd}^{-1}\right)_{i,j} < 0, \end{cases}$$

and the solution of the jth minimization problem is

$$(x_d)_j^* = \begin{cases} l_j, & \left(U_{dd}^{-1}\right)_{i,j} \geq 0, \\ u_j, & \left(U_{dd}^{-1}\right)_{i,j} < 0. \end{cases}$$

The objective function of problem (4.12) can now be written as follows

$$\begin{cases} \sum_{j=1}^{n_d} \left[\left(U_{dd}^{-1}\right)_{i,j} u_j - \left(U_{dd}^{-1}\right)_{i,j} l_j\right], & \left(U_{dd}^{-1}\right)_{i,j} \geq 0, \\ \sum_{j=1}^{n_d} \left[\left(U_{dd}^{-1}\right)_{i,j} l_j - \left(U_{dd}^{-1}\right)_{i,j} u_j\right], & \left(U_{dd}^{-1}\right)_{i,j} < 0. \end{cases} \tag{4.13}$$

Clearly, we can now express (4.12) as follows:

$$\begin{aligned} \underset{\left(U_{dd}^{-1}\right)_i}{\operatorname{argmin}} & \left\{ \sum_{j=1}^{n_d} \left|\left(U_{dd}^{-1}\right)_{i,j}\right| (u_j - l_j) \right\} \\ \text{s.t.} \quad & U_{dd} \text{ is unimodular.} \end{aligned} \tag{4.14}$$

Now consider the following relaxation of problem (4.14):

$$\begin{aligned} \underset{\left(U_{dd}^{-1}\right)_i}{\operatorname{argmin}} & \left\{ \sum_{j=1}^{n_d} \left|\left(U_{dd}^{-1}\right)_{i,j}\right| (u_j - l_j) \right\} \\ \text{s.t.} \quad & U_{dd}^{-1} \text{ is invertible and has integer elements.} \end{aligned} \tag{4.15}$$

Consider the ith problem with the form of problem (4.15), the problem gives us the ith row of U_{dd}^{-1}. Clearly every non-zero value of $\left(U_{dd}^{-1}\right)_{i,j}$ increases the objective function value so our solution should have as many zero elements as possible. Since U_{dd}^{-1} is constrained to be invertible $\left(U_{dd}^{-1}\right)_i$ must have at least one non-zero element. Combining this with the fact that the elements of U_{dd}^{-1} must be integral the solution of the ith problem can be expressed in the following form

$$\left(U_{dd}^{-1}\right)_{i,j} = \begin{cases} \pm 1, & j = p_i, \\ 0, & j \neq p_i, \end{cases}$$

where $p_i \in \mathbb{Z}$ is an index whose value must still be found. Since U_{dd}^{-1} must be invertible we must have $p_i \neq p_r \forall r \in \{1, 2, \ldots, n_d\} \backslash \{i\}$. Any set of p_i indices which satisfy this condition and any combination of choices of 1 or -1 for $\left(U_{dd}^{-1}\right)_{i,p_i}$ is a feasible solution of problem (4.15). However for our purposes all of these solutions can be thought of as being equivalent, for the following reasons. Since there is only one non-zero element in each row the ordering of the rows merely relabels the indices in problem (4.4), which has no effect on either the solution or the difficulty of problem (4.4). The choice of 1 or -1 also has no effect on the difficulty of problem (4.4) since it merely results in a rotation of one of the axes through 180°. Therefore, we shall only consider the solution $U_{dd}^{-1} = I_{n_d}$, which also gives $U_{dd} = I_{n_d}$. Clearly I_{n_d} is unimodular so it is also a feasible solution of problem (4.14). Therefore, since problem (4.15) is a relaxation of problem (4.14), $U_{dd}^{-1} = I_{n_d}$ is a global minimum of problem (4.14). Accordingly, when only bound constraints are present in the original problem, the global minimum of problem (4.10) is $U_{dd}^{-1} = I_{n_d}$ which gives $U_{dd} = I_{n_d}$.

Unfortunately, no analytic solution can be found when general linear constraints are present in the original problem. In this case we solve the problem approximately. We shall consider two different approximations here. The first approximation is constructed by simplifying the constraint that U_{dd} must be unimodular in two ways. Firstly we restrict ourselves to the set of upper triangular unimodular matrices since this makes it much easier to ensure that the determinant of U_{dd} is ± 1, as required. It also allows us to solve each problem separately since the elements in different rows are no longer coupled. With this simplification problem (4.10) takes the following form:

$$\underset{(U_{dd})_i}{\operatorname{argmin}} \left\{ \max_{x_d} \left[\left(U_{dd}^{-1}\right)_i x_d : x \in \Omega_q \right] - \min_{x_d} \left[\left(U_{dd}^{-1}\right)_i x_d : x \in \Omega_q \right] \right\} \quad (4.16)$$

$$\text{s.t.} \quad \left(U_{dd}^{-1}\right)_{i,i} = \pm 1, \quad (4.17)$$

$$\left(U_{dd}^{-1}\right)_{i,j} = 0, \quad j = 1, \ldots, i-1, \quad (4.18)$$

$$\left(U_{dd}^{-1}\right)_{i,j} \in \mathbb{Z}, \quad j = i+1, \ldots, n_d. \quad (4.19)$$

We now have a problem which can be solved via the solution of a number of integer linear programming problems. The second alteration we make is to

relax constraint (4.19) resulting in the following relaxation of problem (4.16):

$$\begin{aligned} &\underset{(U_{dd})_i}{\operatorname{argmin}} \left\{ \max_{x_d} \left[\left(U_{dd}^{-1}\right)_i x_d : x \in \Omega_q \right] - \min_{x_d} \left[\left(U_{dd}^{-1}\right)_i x_d : x \in \Omega_q \right] \right\} \quad (4.20) \\ \text{s.t.} \quad &\left(U_{dd}^{-1}\right)_{i,i} = \pm 1, \\ &\left(U_{dd}^{-1}\right)_{i,j} = 0, \quad j = 1, \ldots, i-1, \\ &\left(U_{dd}^{-1}\right)_{i,j} \in \mathbb{R}, \quad j = i+1, \ldots, n_d. \end{aligned}$$

It is obvious that problem (4.20) is much easier to solve than problem (4.10). Problem (4.20) was solved using the derivative free algorithm NOMAD [6, 90], which solves the problem by treating the objective function as a black box. NOMAD chooses fixed values for $(U_{dd})_i$ at each iteration and inputs them into the objective function of problem (4.20). The objective function is then evaluated by solving the linear programs for x_d using the values of $(U_{dd})_i$ input by NOMAD. Once we have solved problem (4.20) we need to restore the integrality of the off diagonal elements of U_{dd}^{-1} so that U_{dd} is unimodular. This was done by rounding the off diagonal elements of the solution of problem (4.20).

The second approximation considered is much simpler and is obtained by relaxing Ω_q. Define $\bar{l}$ and $\bar{u}$ as follows

$$\begin{aligned} \bar{l}_i &= \min_{x_i} \{x_i : Ax \leq b, Dx = e, l \leq x \leq u\}, \\ \bar{u}_i &= \min_{x_i} \{x_i : Ax \leq b, Dx = e, l \leq x \leq u\}. \end{aligned}$$

Now let $\bar{\Omega}_q$ denote the relaxed feasible region $\bar{\Omega}_q = \{x : \bar{l} \leq x \leq \bar{u}\}$. The second approximation of problem (4.10) is the following problem:

$$\begin{aligned} &\underset{\left(U_{dd}^{-1}\right)_i}{\operatorname{argmin}} \left\{ \max_{x_d} \left[\left(U_{dd}^{-1}\right)_i x_d : x \in \bar{\Omega}_q \right] - \min_{x_d} \left[\left(U_{dd}^{-1}\right)_i x_d : x \in \bar{\Omega}_q \right] \right\} \quad (4.21) \\ \text{s.t.} \quad &U_{dd} \text{ is unimodular.} \end{aligned}$$

From the discussion above we know that the solution of this problem gives $U_{dd} = I_{n_d}$. This may seem to be a fairly weak relaxation but we shall see from the computational results presented in chapter 6 that, for our purposes, its effectiveness is comparable to problem (4.20). The choice between these two approximations is discussed further in chapter 6.

We have now fixed U_{dd}. This allows us to fix U_{cd} as the matrix which satisfies (4.7). We have yet to impose any restriction on U_{cc}, besides demanding that it be invertible. Since H_{cc} is Hermitian it is diagonalisable

[15] and we let U_{cc} be the diagonalising matrix. Obviously the diagonalising matrix is not unique; to specify U_{cc} uniquely we make it the matrix with the normalised eigenvectors of H_{cc} as its columns[1]. Now consider the last term in (4.5); define Θ_{dd} as follows

$$\Theta_{dd} = U_{cd}^T H_{cc} U_{cd} + U_{cd}^T H_{cd} U_{dd} + U_{dd}^T H_{cd}^T U_{cd} + U_{dd}^T H_{dd} U_{dd}. \tag{4.22}$$

We can simplify (4.22) by expressing U_{dd} in terms of U_{cd} using (4.7), as follows

$$U_{cd} = -H_{cc}^{-1} H_{cd} U_{dd}. \tag{4.23}$$

Substituting (4.23) into (4.22) we obtain the following expression for Θ_{dd}

$$\Theta_{dd} = U_{dd}^T \left(H_{dd} - H_{cd}^T H_{cc}^{-1} H_{cd} \right) U_{dd}. \tag{4.24}$$

Now using the values of U_{dd}, U_{cd} and U_{cc} described above (4.5) takes the following form

$$y^T V^T H V y = y_c^T \Theta_{cc} y_c + y_d^T \Theta_{dd} y_d, \tag{4.25}$$

where we define Θ_{cc} as follows

$$\Theta_{cc} = U_{cc}^T H_{cc} U_{cc}. \tag{4.26}$$

Since we have fixed U_{cc} as the matrix which diagonlises H_{cc} we see that Θ_{cc} is diagonal. Problem (1.2) now takes the following form:

$$\begin{aligned} \min_{y} \quad & h(Vy) = \frac{1}{2} \left(y_c^T \Theta_{cc} y_c + y_d^T \Theta_{dd} y_d \right) + g^T V y \\ \text{s.t.} \quad & AVy \leq b, \\ & DVy = e, \\ & l \leq Vy \leq u, \\ & y^L \leq y \leq y^U, \\ & y = \left[y_c^T,\, y_d^T \right]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}. \end{aligned} \tag{4.27}$$

It is clear that the structure of the objective function has been simplified by the transformation. This has been achieved by reducing the number of bilinear terms in the objective function. The reduction is quantified in the following theorem.

Theorem 1. *$h(x)$ has at most $\frac{1}{2}(n_c^2 - n_c) + n_c n_d$ more bilinear terms than $h(Vy)$*

[1] Any symmetric $n \times n$ matrix has n linearly independent eigenvectors [15]

Proof. We consider the case with the maximum reduction in the number of bilinear terms, this occurs when H has no zero elements. The difference in the number of bilinear terms will then be equal to the number of zero elements in the upper triangular portion of the Hessian of problem (4.27). Now the Hessian of problem (4.27) has the following form

$$\Theta = \begin{bmatrix} \Theta_{cc} & 0_{n_c \times n_d} \\ 0_{n_d \times n_c} & \Theta_{dd} \end{bmatrix},$$

where $0_{n_c \times n_d}$ is a zero matrix with dimensions $n_c \times n_d$. Now Θ_{dd} is dense so it has no zero off diagonal elements, Θ_{cc} is diagonal so it has $\frac{1}{2}(n_c^2 - n_c)$ zero upper triangular elements and obviously $0_{n_c \times n_d}$ has $n_c n_d$ zero elements. Therefore $h(x)$ has at most $\frac{1}{2}(n_c^2 - n_c) + n_c n_d$ more bilinear terms than $h(Vy)$. □

We see that we have transformed problem (1.2) into an equivalent problem problem which has no terms containing both continuous and integer variables, has an objective function which is fully separable in the continuous variables and has at most $\frac{1}{2}(n_c^2 - n_c) + n_c n_d$ less bilinear terms in the objective function than problem (1.2). This has been achieved without adding any additional variables or constraints to the problem, the only detrimental effect is that the bound constraints have become linear inequality constraints. Clearly since problem (1.2) and problem (4.27) are equivalent we have

$$\min_x h(x) = \min_y h(Vy), \tag{4.28}$$

$$\operatorname*{argmin}_x h(x) = V \left\{ \operatorname*{argmin}_y h(Vy) \right\}. \tag{4.29}$$

These equations allow us to obtain the solution of problem (1.2) from that of problem (4.27).

4.2.2 The Branch and Bound Procedure

We now describe a Branch and Bound algorithm can be used to solve problem (4.27). This algorithm is based on BARON, which was described in chapter 2. Lower bounds are obtained using the convex envelopes of the non-convex terms in the objective functions. The convex envelopes for bilinear and concave univariate functions with known upper and lower bounds were given in chapter 2. These envelopes require knowledge of the upper and lower bounds of each of the variables. The bounds for our problem are given by (4.8) and (4.9).

We use the following notation in our description of the Branch and Bound algorithm; the overall lower bound OLB is the current lower bound on the solution, the overall upper bound OUB is the current upper bound on the solution, the stopping tolerance tol is set by the user and is used to stop the algorithm when $OUB - OLB < tol$. The Branch and Bound procedure is described in Algorithm 2, the details on how steps iii, iv, viii and xi were performed are given in more detail in the following paragraph.

We first discuss the node selection strategy used in step iii. Following [165] the node chosen for branching was the node with the smallest lower bound LB, i.e. the node of the current OLB. We now discuss step iv in which a branching variable y_b is chosen. The branching variable was chosen by finding the non-convex term with the greatest difference between the non-convex term and its convex underestimator at the solution of the convex underestimation of the problem. If the term was bilinear the branching variable was chosen to be the variable involved in the bilinear term with the largest domain. The branching point y_b^{val} was chosen as follows. Let

$$\varepsilon = \frac{y_b^U - y_b^L}{10},$$

where y_b^U and y_b^L are the upper and lower bounds of the branching variable at the current node. Let y_b^{up} be the value of y_b at the current OUB and let y_b^{low} be the value of y_b at the LB for the current node. Following [135] for every fifth node the value of y_b^{val} was given by

$$y_b^{\text{val}} = \frac{y_b^U - y_b^L}{2}.$$

Otherwise the value of y_b^{val} was given by [165]

$$y_b^{\text{val}} = \begin{cases} y_b^{\text{up}} & \text{if} \quad y_b^L + \varepsilon < y_b^{\text{up}} < y_b^U - \varepsilon, \\ y_b^{\text{low}} & \text{if} \quad \neg\left(y_b^L + \varepsilon < y_b^{\text{up}} < y_b^U - \varepsilon\right) \wedge \left(y_b^L + \varepsilon < y_b^{\text{low}} < y_b^U - \varepsilon\right), \\ \frac{y_b^U - y_b^L}{2} & \text{otherwise.} \end{cases}$$

Obviously if y_b is an integer variable the branching is performed using the new constraints $y_b \leq \lfloor y_b^{\text{val}} \rfloor$ and $y_b \geq \lceil y_b^{\text{val}} \rceil$. The valid upper bounds in step viii were obtained using `bnb20` [86]. `bnb20` is an open source convex MINLP solver which uses a Branch and Bound methodology. The optimality based range reduction technique used in step xi is taken from [135]. If the lower bound of y_i is active at the solution of the lower bounding problem and the Lagrangian multiplier λ_i of the constraint is greater than zero then the following inequality is valid

$$y_i < l_i + \frac{OUB - OLB}{\lambda_i}. \tag{4.30}$$

Algorithm 2 The Branch and Bound algorithm

i. Set the stopping tolerance tol, set $OLB = -\infty$, set $OUB = \infty$ and initialise the list of active nodes, ACTIVE, as an empty list.

ii. Add the problem to ACTIVE with the bounds given by (4.8) and (4.9). Form the convex underestimator of the problem and solve its convex relaxation to obtain an initial LB.

iii. Select a node for branching.

iv. Select the branching variable y_b. Select the value y_b^{val} at which the variable will be branched.

v. Add the two nodes formed by branching to ACTIVE.

vi. Form convex underestimators of the two nodes obtained from the branching using the convex envelopes given in [135].

vii. Solve the continuous relaxations of the convex undestimators. This was done using the MATLAB function `quadprog`. This gives valid lower bounds LB over the domain of the subproblems.

viii. If the two new nodes are on a multiple of the fourth level of the tree obtain an upper bound UB on the solution.

ix. Find OLB and OUB.

x. If $LB > OUB$ prune the node from ACTIVE.

xi. Apply an optimality based range reduction technique to each of the new nodes using (4.30) and (4.31). If either of the nodes becomes infeasible after the range reduction then prune that node.

xii. If at least one of the new nodes has not been pruned then remove the parent node from ACTIVE. If both of the new nodes have been pruned store LB of the parent node for comparison with OLB and remove the parent node from ACTIVE.

xiii. If $OUB - OLB < tol$ or if ACTIVE becomes empty then return OUB and stop otherwise go to step iii.

Similarly if the upper bound is active and $\lambda_i > 0$ then the following inequality is valid

$$y_i > u_i - \frac{OUB - OLB}{\lambda_i}. \tag{4.31}$$

This completes our description of the Branch and Bound algorithm.

4.3 Approach used when H_{cc} is positive definite

When H_{cc} is positive semidefinite problem (1.2) can be reformulated as a convex program. This allows us to apply convex mixed integer approaches to the solution of a non-convex problem. In this section we consider three different convex reformulation schemes for problem (1.2). The first reformulation, which was mentioned in chapter 2, was developed in [33] and is known as Mixed Integer Quadratic Convex Reformulation (MIQCR). MIQCR results in an equivalent convex MIQP and can be applied to any problem where H_{cc} is positive semidefinite. MIQCR is discussed in section 4.3.1. The reformulation discussed in section 4.3.2 combines the linear transformation in section 4.1 with the ideas used in [33]. We call this reformulation technique Mixed Integer Quadratic Transformation and Convex Reformulation (MIQTCR). MIQTCR also results in an equivalent convex MIQP and can be applied to any problem where H_{cc} positive definite. The third reformulation is discussed in section 4.3.3. This reformulation combines the linear transformation with a convexification scheme for bilinear integer programming developed in [122]. We call this reformulation Mixed Integer Quadratic Transformation and Bilinear Convexification (MIQTBC). MIQTBC results in an equivalent convex MINLP. MIQTBC can be applied to any problem where H_{cc} is positive definite and another fairly unrestrictive condition on the form of the Hessian holds.

4.3.1 MIQCR

In this section we assume without loss of generality that the origin has been shifted such that the lower bound on each variable is 0. The details of the method described in this section are all taken from [33]. We define the following sets of indices

$$P = \left\{(i,j) \in I^2 \cup (I \times J) \cup (J \times I)\right\},$$
$$E = \left\{(i,k) : i \in I,\, k = 0, \ldots, \lfloor \log_2 (u_i) \rfloor\right\},$$

where u_i is the upper bound on x_i. We convexify problem (1.2) by adding to the problem new variables w_{ij} and new linear constraints which ensure that $w_{ij} = x_i x_j$. If equality constraints are present in problem (1.2) we also add the following term to the objective function

$$\alpha \sum_{r=1}^{p} \left\{ \sum_{i=1}^{n} (D)_{r,i} \, x_i - e_r \right\}^2 , \tag{4.32}$$

where $(D)_{r,i}$ is the element in the rth row and ith column of D, and $\alpha \in \mathbb{R}$ is a parameter whose value is discussed later in this section. Clearly (4.32) is zero in the feasible region so its addition will have no effect on the objective function value.

More specifically we perform the convex reformulation by considering the following perturbed objective function

$$h_{\alpha,\beta}(x,w) = h(x) + \sum_{(i,j)\in P} \beta_{ij} \left(x_i x_j - w_{ij}\right) + \alpha \sum_{r=1}^{p} \left\{ \sum_{i=1}^{n} (D)_{r,i} \, x_i - e_r \right\}^2 . \tag{4.33}$$

Clearly $h_{\alpha,\beta}(x,w) = h(x)$ in the feasible region, if we enforce the equality constraint $w_{ij} = x_i x_j$. The following problem, which has (4.33) as its objective function, is equivalent to problem (1.2):

$$\begin{aligned}
\min_{x,w} \quad & h_{\alpha,\beta}(x,w) && && (4.34a)\\
\text{s.t.} \quad & Ax \le b, && && (4.34b)\\
& Dx = e, && && (4.34c)\\
& 0 \le x \le u, && && (4.34d)\\
& x_i = \sum_{k=0}^{\lfloor \log_2 u_i \rfloor} 2^k t_{ik}, && i \in I, && (4.34e)\\
& t_{ik} \in \{0,1\}, && (i,k) \in E, && (4.34f)\\
& z_{ijk} \le u_j t_{ik}, && (i,k) \in E,\ j \in I \cup J, && (4.34g)\\
& z_{ijk} \le x_j, && (i,k) \in E,\ j \in I \cup J, && (4.34h)\\
& z_{ijk} \ge x_j - u_j \left(1 - t_{ik}\right), && (i,k) \in E,\ j \in I \cup J, && (4.34i)\\
& z_{ijk} \ge 0, && (i,k) \in E,\ j \in I \cup J, && (4.34j)\\
& w_{ij} = \sum_{k=0}^{\lfloor \log_2 u_i \rfloor} 2^k z_{ijk}, && (i,j) \in I \times (I \cup J), && (4.34k)\\
& w_{ii} \ge x_i, && i \in I, && (4.34l)
\end{aligned}$$

$$w_{ij} = w_{ji}, \qquad (i,j) \in P, \tag{4.34m}$$
$$w_{ij} \geq x_i u_j + x_j u_i - u_i u_j, \qquad (i,j) \in P, \tag{4.34n}$$
$$w_{ij} \leq u_i x_j, \qquad (i,j) \in I \times J. \tag{4.34o}$$

We now show that problem (4.34) is equivalent to problem (1.2). It was noted above the $h_{\alpha,\beta}(x,w) = h(x)$ when $w_{ij} = x_i x_j$. Therefore it is sufficient to show that the constraints of problem (4.34) enforce the equality constraint $w_{ij} = x_i x_j$. Constraint (4.34e) is a unique binary decomposition of x_i. Using this decomposition we obtain the following equality

$$x_i x_j = \sum_{k=0}^{\lfloor \log_2 u_i \rfloor} 2^k t_{ik} x_j = w_{ij}. \tag{4.35}$$

We linearise (4.35) by introducing new variables z_{ijk} which are equal to $t_{ik}x_j$. Substituting z_{ijk} into (4.35) gives constraint (4.34k). Constraints (4.34g)–(4.34j) enforce the equality $z_{ijk} = t_{ik}x_j$. To show that this is the case we consider the two possible values of t_{ik} separately. First consider the case $t_{ik} = 0$. In this case we must have $z_{ijk} = 0$. Now consider the form of constraints (4.34g)–(4.34j) when $t_{ik} = 0$

$$\begin{aligned} z_{ijk} &\leq 0, \\ z_{ijk} &\leq x_j, \\ z_{ijk} &\geq x_j - u_j, \\ z_{ijk} &\geq 0. \end{aligned}$$

Clearly, in this case we have $z_{ijk} = 0$, as required. Now consider the case $t_{ik} = 1$. In this case we must have $z_{ijk} = x_j$. Consider the form of constraints (4.34g)-(4.34j) when $t_{ik} = 1$

$$\begin{aligned} z_{ijk} &\leq u_j, \\ z_{ijk} &\leq x_j, \\ z_{ijk} &\geq x_j, \\ z_{ijk} &\geq 0. \end{aligned}$$

Clearly, in this case we have $z_{ijk} = x_j$, as required. From the above discussion we see that, as required, the constraints in problem (4.34) enforce the constraint $w_{ij} = x_i x_j$.

We want to find the values of α and β_{ij} which give the maximum value for the minimum of the continuous relaxation of problem (4.34) while also ensuring that (4.33) is convex. Denote these values by α^* and β^*_{ij}. We choose

these values of α and β_{ij} since they will allow us to obtain a tight initial lower bound when solving the problem with a Branch and Bound algorithm. Now it can be shown that the continuous relaxation of problem (4.34) is equivalent to the following problem [33]:

$$\begin{aligned}
\min_{x,w} \quad & h_{\alpha,\beta}(x,w) && && (4.36a)\\
\text{s.t.} \quad & Ax \leq b, && && (4.36b)\\
& Dx = e, && && (4.36c)\\
& 0 \leq x_i \leq u_i, && i \in J, && (4.36d)\\
& w_{ii} \geq x_i, && i \in I, && (4.36e)\\
& w_{ij} = w_{ji}, && (i,j) \in P, && (4.36f)\\
& w_{ij} \geq x_i u_j + x_j u_i - u_i u_j, && (i,j) \in P, && (4.36g)\\
& w_{ij} \geq 0, && (i,j) \in P, && (4.36h)\\
& w_{ij} \leq u_i x_j, && (i,j) \in P, && (4.36i)\\
& w_{ij} \leq u_j x_i, && (i,j) \in P. && (4.36j)
\end{aligned}$$

This problem is derived by projecting the feasible region of continuous relaxation of problem (4.34) onto the x and w variables, see [33] for further details. The values of α^* and β^*_{ij} can now be found by solving the following problem:

$$\begin{aligned}
\max_{\alpha,\beta} \quad & \mathcal{V}\left\{\text{problem (4.36)}\right\} && (4.37a)\\
\text{s.t} \quad & H_{\alpha,\beta} \succcurlyeq 0, && (4.37b)
\end{aligned}$$

where $\mathcal{V}\left\{\text{problem (4.36)}\right\}$ is the optimal value of problem (4.36) and $H_{\alpha,\beta}$ is the Hessian of $h_{\alpha,\beta}$. Constraint (4.37b) ensures that the reformulated problem is convex. It is shown in [33] that the optimal value of problem (4.37) can be found using the following theorem.

Theorem 2. *α^* and β^*_{ij} can be deduced from the optimal values of the dual variables of the following semidefinite program:*

$$\begin{aligned}
\min_{x,X} \quad & \operatorname{Tr}(HX) + g^T x && (4.38a)\\
s.t. \quad & Ax \leq b, && (4.38b)\\
& Dx = e, && (4.38c)\\
& 0 \leq x_i \leq u_i, \quad i \in J, && (4.38d)\\
& \begin{bmatrix} 1 & x \\ x^T & X \end{bmatrix} \succcurlyeq 0, && (4.38e)
\end{aligned}$$

$$\sum_{r=1}^{a}\sum_{i=1}^{n}\left\{\sum_{j=1}^{n}(D)_{ri}(D)_{rj}X_{ij}-2(D)_{ri}e_r x_i\right\}=-\sum_{r=1}^{a}e_r^2, \tag{4.38f}$$

$$X_{ij}\leq u_j x_i, \quad (i,j)\in P, \tag{4.38g}$$

$$X_{ij}\leq u_i x_j, \quad (i,j)\in P, \tag{4.38h}$$

$$X_{ij}\geq u_j x_i + u_i x_j - u_i u_j, \quad (i,j)\in P, \tag{4.38i}$$

$$X_{ij}\geq 0, \quad (i,j)\in P, \tag{4.38j}$$

$$X_{ii}\geq x_i, \quad i\in I, \tag{4.38k}$$

$$x\in\mathbb{R}^n, \tag{4.38l}$$

$$X\in\mathbf{S}_n. \tag{4.38m}$$

The value of α^* is equal to the optimal value of the dual variable associated with constraint (4.38f)*. The coefficients β_{ij}^* are given by the following equation*

$$\beta_{ij}^*=\begin{cases}\beta_{ij}^{1*}+\beta_{ij}^{2*}-\beta_{ij}^{3*}-\beta_{ij}^{4*}, & i\neq j,\\ \beta_{ij}^{1*}+\beta_{ij}^{2*}-\beta_{ij}^{3*}-\beta_{ij}^{4*}-\beta_{ij}^{5*}, & i=j,\end{cases}$$

where β_{ij}^{1*}, β_{ij}^{2*}, β_{ij}^{3*}, β_{ij}^{4*} and β_{ii}^{5*} are the optimal values of the dual variables associated with constraints (4.38g)*,* (4.38h)*,* (4.38i)*,* (4.38j) *and* (4.38k) *respectively.*

Proof. See [33] □

Substituting the values of α^* and β_{ij}^*, obtained using Theorem 2, into problem (4.34) we obtain a convex MIQP which is equivalent to problem (1.2). The equivalent MIQP, problem (4.34), can be solved efficiently due to the tight lower bound given by the continuous relaxation of the problem. As noted previously MIQCR can be applied to any problem where the n_cth principal leading submatrix is positive semidefinite.

In the case when problem (1.2) has inequality constraints and no equality constraints it is desirable to reformulate the problem in such a way that we can use α to tighten the lower bound obtained for the continuous relaxation [33]. This can be done by using slack variables to convert the inequality constraints to equality constraints.

4.3.2 MIQTCR

In this section we combine the linear transformation from section 4.1 with the ideas developed in section 4.3.1. We choose the same elements of V as were chosen in section 4.2.1 resulting in a problem with the form of problem (4.27). We now assume, without loss of generality, that the origin of the

transformed problem has been shifted such that $y_i^L = 0 \ \forall i \in J \cup I$. This allows us to apply the methods developed in section 4.3.1.

The objective function of our transformed problem has no terms containing both continuous and integer variables. In addition the continuous terms in the objective function, $y_c^T \Theta_{cc} y_c$, are convex. Therefore, if we can perturb the discrete terms, $y_d^T \Theta_{dd} y_d$, in such a way that the discrete part of the objective function is convex, the entire resulting objective function will be convex. This leads us to consider a modification of the method described in section 4.3.1 in which we set $\beta_{ij} = 0, \forall (i,j) \notin I^2$. In developing this method we consider the following perturbed objective function

$$h_{\alpha,\beta}(y,w) = h(y) + \sum_{(i,j)\in I^2} \beta_{ij}\left(y_i y_j - w_{ij}\right) + \alpha \sum_{r=1}^{p} \left\{ \sum_{i=1}^{n} \left(DV\right)_{r,i} y_i - e_r \right\}^2 .$$

Now the following problem, which has this expression as its objective function, is equivalent to problem (1.2):

$$\begin{aligned}
\min_{y,w} \quad & h_{\alpha,\beta}(y,w) && && (4.39a)\\
\text{s.t.} \quad & AVy \le b, && && (4.39b)\\
& DVy = e, && && (4.39c)\\
& l \le Vy \le u, && &&\\
& 0 \le y \le y^U, && && (4.39d)\\
& y_i = \sum_{k=0}^{\lfloor \log_2 y_i^U \rfloor} 2^k t_{ik}, && i \in I, && (4.39e)\\
& t_{ik} \in \{0,1\}, && (i,k) \in E && (4.39f)\\
& z_{ijk} \le y_j^U t_{ik}, && (i,k) \in E,\ j \in I, && (4.39g)\\
& z_{ijk} \le y_j, && (i,k) \in E,\ j \in I, && (4.39h)\\
& z_{ijk} \ge y_j - y_j^U \left(1 - t_{ik}\right), && (i,k) \in E,\ j \in I, && (4.39i)\\
& z_{ijk} \ge 0, && (i,k) \in E,\ j \in I, && (4.39j)\\
& w_{ii} \ge y_i, && i \in I, && (4.39k)\\
& w_{ij} = \sum_{k=0}^{\lfloor \log_2 y_i^U \rfloor} 2^k z_{ijk}, && (i,j) \in I^2, && (4.39l)\\
& w_{ij} = w_{ji}, && (i,j) \in I^2, && (4.39m)\\
& w_{ij} \ge y_i y_j^U + y_j y_i^U - y_i^U y_j^U, && (i,j) \in I^2. && (4.39n)
\end{aligned}$$

We note that there is no constraint equivalent to (4.34o) in problem (4.39), this is because there are no w_{ij} variables with $(i,j) \in I \times J$ in $h_{\alpha,\beta}(y,w)$. The

equivalence between problems (1.2) and (4.39) can be shown by following the same reasoning used in the paragraph following problem (4.34). We see that problem (4.39) has fewer variables than problem (4.34). We now use the following trivial modification of Theorem 2 to find α^* and β_{ij}^*.

Theorem 3. *α^* and β_{ij}^* can be deduced from the optimal values of the dual variables of the following semidefinite program:*

$$\begin{aligned}
\min_{y,X} \quad & \mathrm{Tr}(HX) + g^T V y && \text{(4.40a)}\\
s.t. \quad & AVy \leq b, && \text{(4.40b)}\\
& DVy = e, && \text{(4.40c)}\\
& l \leq Vy \leq u,\\
& 0 \leq y_i \leq y_i^U, \quad i \in J, && \text{(4.40d)}\\
& \begin{bmatrix} 1 & y \\ y^T & X \end{bmatrix} \succcurlyeq 0, && \text{(4.40e)}\\
& \sum_{r=1}^{a}\sum_{i=1}^{n}\left\{\sum_{j=1}^{n}(DV)_{ri}(DV)_{rj}X_{ij} - 2(DV)_{ri}e_r y_i\right\} = -\sum_{r=1}^{a} e_r^2, && \text{(4.40f)}\\
& X_{ij} \leq y_j^U y_i, \quad (i,j) \in I^2, && \text{(4.40g)}\\
& X_{ij} \leq y_i^U y_j, \quad (i,j) \in I^2, && \text{(4.40h)}\\
& X_{ij} \geq y_j^U y_i + u_i y_j - y_i^U y_j^U, \quad (i,j) \in I^2, && \text{(4.40i)}\\
& X_{ij} \geq 0, \quad (i,j) \in I^2, && \text{(4.40j)}\\
& X_{ii} \geq y_i, \quad i \in I, && \text{(4.40k)}\\
& y \in \mathbb{R}^n, && \text{(4.40l)}\\
& X \in \mathbf{S}_n. && \text{(4.40m)}
\end{aligned}$$

The value of α^* is equal to the optimal value of the dual variable associated with constraint (4.40f)*. The coefficients β_{ij}^* are given by the following equation*

$$\beta_{ij}^* = \begin{cases} \beta_{ij}^{1*} + \beta_{ij}^{2*} - \beta_{ij}^{3*} - \beta_{ij}^{4*}, & i \neq j, \\ \beta_{ij}^{1*} + \beta_{ij}^{2*} - \beta_{ij}^{3*} - \beta_{ij}^{4*} - \beta_{ij}^{5*}, & i = j, \end{cases}$$

where β_{ij}^{1*}, β_{ij}^{2*}, β_{ij}^{3*}, β_{ij}^{4*} and β_{ii}^{5*} are the optimal values of the dual variables associated with constraints (4.40g)*,* (4.40h)*,* (4.40i)*,* (4.40j) *and* (4.40k) *respectively.*

Substituting the values of α^* and β_{ij}^*, obtained using Theorem 3, into problem (4.39) we obtain a convex MIQP which is equivalent to problem (1.2). The lower bound given by the continuous relaxation of problem (4.39)

is generally not as tight as the bound of problem (4.34). This is due to the fact that each β_{ij} term used to perturb the objective function is used to tighten the continuous relaxation of the objective function and MIQTCR uses fewer β_{ij} terms than MIQCR. However, due to this reduction in the number of β_{ij} terms, problem (4.39) has fewer variables than problem (4.34). This may allow problem (4.39) to be solved more efficiently than problem (4.34). In addition we note that the transformation might allow us to solve problem (4.40) faster than problem (4.38).

In the case when problem (1.2) has inequality constraints and no equality constraints we make use of α as described at the end of section 4.3.1. We note that the slack variable transformation must be applied to problem (4.27) as adding slack variables to the original problem will make H_{cc} positive semidefinite.

4.3.3 MIQTBC

MIQTBC uses a variation of the linear transformation with a convexification scheme developed for bilinear integer programming [122] to convexify problem (1.2). To use this scheme we require that the only non-convex terms in the objective function of the transformed problem are bilinear terms involving only the integer variables. It is shown below that a transformation resulting in an objective function with the required form exists and an algorithm is given to find the transformation.

The Linear Transformation for MIQTBC

In the following theorem we show that under a fairly unrestrictive condition we can choose V such that the only non-convex terms in the objective function are bilinear integer terms. When we discuss variable reordering in relation to the following theorem we mean interchanging two discrete variables or two continuous variables, we do not allow the interchanging of a continuous variable and a discrete variable.

Theorem 4. *If the elements of x can be reordered such that $H_{dd} - H_{cd}^T H_{cc}^{-1} H_{cd}$ has at least two principal leading submatrices which are not negative semidefinite then there exists a unimodular matrix U_{dd} such that the diagonal elements of Θ_{dd} are all positive.*

Proof. Let $\Lambda = H_{dd} - H_{cd}^T H_{cc}^{-1} H_{cd}$. Consider a transformation matrix V of the form

$$V = \begin{bmatrix} U_{cc} & U_{cd} \\ 0_{n_d,n_c} & \widetilde{U}_{dd} U_{dd} \end{bmatrix}.$$

Now the product of two unimodular matrices is unimodular so if both U_{dd} and $\widetilde{U}_{dd}$ are unimodular then so is $\widetilde{U}_{dd}U_{dd}$. Using this form of V we have $\Theta_{dd} = U_{dd}^T\widetilde{U}_{dd}^T\Lambda\widetilde{U}_{dd}U_{dd}$. Now let $\widetilde{\Lambda} = \widetilde{U}_{dd}^T\Lambda\widetilde{U}_{dd}$. We first show that there exists a unimodular matrix $\widetilde{U}_{dd}$ such that $\widetilde{\Lambda}$ has at least one positive diagonal element. Now let $A^{(k)}$ denote a $k \times k$ submatrix of A; the position of $A^{(k)}$ will be clear from the context. We can now write $\widetilde{U}_{dd}$ and Λ as follows

$$\widetilde{U}_{dd} = \begin{bmatrix} \pm 1 & 0_{1,n_d-1} \\ \widetilde{\mu} & \widetilde{U}_{dd}^{(n_d-1)} \end{bmatrix}, \Lambda = \begin{bmatrix} \alpha & \lambda^T \\ \lambda & \Lambda^{(n_d-1)} \end{bmatrix}, \tag{4.41}$$

where $\widetilde{U}_{dd}^{(n_d-1)}$ is a lower triangular matrix with ± 1 on its diagonal, $\widetilde{\mu} \in \mathbb{Z}^{n_d-1}$, $\lambda \in \mathbb{R}^{n_d-1}$ and $\alpha \in \mathbb{R}$. We can now express $\widetilde{\Lambda}$ as follows

$$\widetilde{\Lambda} = \begin{bmatrix} \widetilde{\mu}^T\Lambda^{(n_d-1)}\widetilde{\mu} \pm 2\lambda^T\widetilde{\mu} + \alpha & \widetilde{\mu}^T\Lambda^{(n_d-1)}\widetilde{U}_{dd}^{(n_d-1)} \pm \lambda^T\widetilde{U}_{dd}^{(n_d-1)} \\ \widetilde{U}_{dd}^{(n_d-1)\,T}\Lambda^{(n_d-1)}\widetilde{\mu} \pm \widetilde{U}_{dd}^{(n_d-1)\,T}\lambda & \widetilde{U}_{dd}^{(n_d-1)\,T}\Lambda^{(n_d-1)}\widetilde{U}_{dd}^{(n_d-1)} \end{bmatrix}.$$

Now since Λ must have at least two leading submatrices which are not negative semidefinite the variables in x can be reordered to make $\Lambda^{(n_d-1)}$ indefinite or positive semidefinite. Suppose that the variables have been ordered such that this is true. If this is the case, $\widetilde{\mu}^T\Lambda^{(n_d-1)}\widetilde{\mu} \pm 2\lambda^T\widetilde{\mu} + \alpha$ is unbounded above, regardless of the values of λ and α. Therefore $\exists\, \widetilde{\mu} \in \mathbb{Z}^{n_d-1}$ such that $(\widetilde{\Lambda})_{1,1}$ is greater than zero.

Suppose that we choose the elements of $\widetilde{U}_{dd}$ such that $(\widetilde{\Lambda})_{1,1} > 0$. Now Θ_{dd} can be written as $\Theta_{dd} = U_{dd}^T\widetilde{\Lambda}U_{dd}$. We now prove by induction that each leading submatrix of Θ_{dd} can be constructed to have only positive diagonal elements. We have

$$\begin{aligned} &\Theta_{dd}^{(1)} = U_{dd}^{(1)\,T}\widetilde{\Lambda}^{(1)}U_{dd}^{(1)} = (\pm 1)^2\widetilde{\Lambda}^{(1)} = \widetilde{\Lambda}^{(1)}, \\ &\Theta_{dd}^{(1)} > 0. \end{aligned}$$

Therefore the diagonal element of $\Theta_{dd}^{(1)}$ is positive. Now assume the diagonal elements of $\Theta_{dd}^{(k)}$ are positive. We need to show that there exists a unimodular matrix with $U_{dd}^{(k+1)}$ such that $\Theta_{dd}^{(k+1)}$ has positive diagonal elements. Now

$$\Theta_{dd}^{(k+1)} = U_{dd}^{(k+1)\,T}\widetilde{\Lambda}^{(k+1)}U_{dd}^{(k+1)},$$

where

$$U_{dd}^{(k+1)} = \begin{bmatrix} U_{dd}^{(k)} & \mu \\ 0_{1,k} & \pm 1 \end{bmatrix}, \widetilde{\Lambda}^{(k+1)} = \begin{bmatrix} \widetilde{\Lambda}^{(k)} & \widetilde{\lambda} \\ \widetilde{\lambda}^T & a \end{bmatrix},$$

where $U_{dd}^{(k)}$ is an upper triangular matrix with ± 1 on its diagonal, $\mu \in \mathbb{Z}^k$, $\widetilde{\lambda} \in \mathbb{R}^k$ and $a \in \mathbb{R}$. Therefore

$$\Theta_{dd}^{(k+1)} = \begin{bmatrix} U_{dd}^{(k)\,T}\widetilde{\Lambda}^{(k)}U_{dd}^{(k)} & U_{dd}^{(k)\,T}\widetilde{\Lambda}^{(k)}\mu \pm U_{dd}^{(k)\,T}\widetilde{\lambda} \\ \mu^T\widetilde{\Lambda}^{(k)}U_{dd}^{(k)} \pm \widetilde{\lambda}^T U_{dd}^{(k)} & \mu^T\widetilde{\Lambda}^{(k)}\mu \pm 2\widetilde{\lambda}^T\mu + a \end{bmatrix}.$$

We need to show that $\exists\, \mu \in \mathbb{Z}^k$ s.t. $\mu^T\widetilde{\Lambda}^{(k)}\mu \pm 2\widetilde{\lambda}^T\mu + a > 0$. Since this is a quadratic function of μ it is sufficient to show that one of the diagonal elements of $\widetilde{\Lambda}^{(k)}$ is greater than zero. Now $\widetilde{\Lambda}^{(k)}$ has at least one positive element, $(\widetilde{\Lambda}^{(k)})_{1,1}$. Therefore $\exists\, \mu \in \mathbb{Z}^k$ s.t that $(\Theta_{dd}^{(k+1)})_{k+1,k+1} > 0$ and by the assumption $(\Theta_{dd}^{(k)})_{i,i} > 0\ \forall i = 1, \ldots, k$. Therefore there exists a unimodular matrix U_{dd} such that the diagonal elements of Θ_{dd} are all positive. □

It should be noted that the assumption that the elements of x can be reordered such that $H_{dd} - H_{cd}^T H_{cc}^{-1} H_{cd}$ has at least two principal leading submatrices which are not negative semidefinite will not be satisfied for every H. This assumption is key to the proof of the theorem. However numerical experience suggests that the condition will be satisfied for a large number of Hessians, especially as the value of n increases. This is mainly due to the fact that we can rearrange the elements of Λ by relabelling the elements of x. For example if any of the diagonal elements of Λ are greater than zero the elements of x can be relabelled such that the assumption holds. However, the assumption can also hold when all of the diagonal elements of Λ are less than zero[2].

We now consider problem (4.4). We make the same choices for U_{cc} and U_{cd} as were made in section 4.2.1[3] and we use Algorithm 3, presented later in this section, to choose U_{dd} such that Theorem 4 is satisfied. We now have the following MIQP which is equivalent to problem (1.2):

$$\begin{aligned} \min_{y} \quad & h(Vy) = \frac{1}{2}\left(y_c^T\Theta_{cc}y_c + y_d^T\Theta_{dd}y_d\right) + g^TVy \\ \text{s.t.} \quad & AVy \leq b, \\ & DVy = b, \\ & l \leq Vy \leq u, \\ & y_i^L < y_i < y_i^U, \quad \forall i \in J \cup I, \\ & y = \left[y_c^T,\, y_d^T\right]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}, \end{aligned} \tag{4.42}$$

[2]It is possible for all of the diagonal elements of an indefinite matrix to be negative.

[3]These choices of U_{cc} and U_{cd} can be used since every positive definite matrix is invertible [74]

where $\Theta_{cc} = U_{cc}^T H_{cc} U_{cc}$ is diagonal and all the diagonal elements of Θ_{dd} are positive. We note that since we have used Algorithm 3 to choose U_{dd} the only non-convex terms in objective function of problem (4.42) are integer bilinear terms. In the following discussion we assume, without loss of generality, that the origin has been shifted such that $y_i^L = 1 \; \forall i \in J \cup I$.

The Algorithm Used to Generate V

While we have proven the existence of a transformation matrix V with the required properties we still need to provide a concrete method for finding matrices with this form. The required method is described by Algorithm 3. The algorithm finds a matrix with the required form using two basic stages. The first stage ensures that we have a $\widetilde{\Lambda}$ matrix with $(\widetilde{\Lambda})_{1,1} > 0$; steps ii to v are involved in this process. If Λ does not have a positive diagonal element in step ii then a matrix $\widetilde{U}_{dd}$, which will allow us to define $\widetilde{\Lambda}$ such that it has at least one positive diagonal element, is found in steps iii and iv. The second stage of the algorithm finds a matrix U_{dd} which ensures that the diagonal elements of $U_{dd}^T \widetilde{\Lambda} U_{dd}$ are all positive where $\widetilde{\Lambda}$ was found using the first part of the algorithm. Steps vi to ix are involved in the second stage of the algorithm. We restrict U_{dd} to be an upper triangular matrix with 1 or -1 on its diagonal. We set the upper triangular elements one column at a time. To ensure that the diagonal elements of $U_{dd}^T \widetilde{\Lambda} U_{dd}$ are positive we need to ensure that $\mu^T \widetilde{\Lambda}^{(k)} \mu \pm 2\widetilde{\lambda}^T \mu + a > 0$ is satisfied. To do this we first check, in step vi, whether this equation is satisfied by $\mu = 0$. If this is the case we set the upper triangular elements of the column to zero. Otherwise we use steps vii and viii to set the element of the column in the first row to the smallest number which allows this equation to be satisfied if all the other elements of μ are set to zero. We then set the diagonal element of U_{dd} to 1 or -1 if our choice of μ satisfies $\mu^T \widetilde{\Lambda}^{(k)} \mu + 2\widetilde{\lambda}^T \mu + a > 0$ or $\mu^T \widetilde{\Lambda}^{(k)} \mu - 2\widetilde{\lambda}^T \mu + a > 0$ respectively.

Convexification Scheme for Bilinear Integer Terms

We now describe the convex reformulation scheme used for the discrete bilinear terms in the objective function of problem (4.42). The method used involves a change of variables resulting in a linearly constrained convex MINLP. The details of the reformulation scheme are taken from [122]. As noted above we have shifted the origin of our problem such that the lower bound of each of the variables is one. We first consider bilinear terms of the form

$$a y_i y_j, \quad a > 0, \, i \neq j.$$

Algorithm 3 The U_{dd} selection algorithm for MIQTBC

i. Set rhs $= 1$, ind $= 2$ and $U_{dd} = 0_{n_d \times n_d}$.

ii. If Λ has at least one positive diagonal element reorder the variables such that $(\Lambda)_{1,1} > 0$, set $\widetilde{U}_{dd} = I_{n_d}$, set $\widetilde{\Lambda} = \Lambda$ and go to step vi. If all of the diagonal elements of Λ are negative go to step iii.

iii. Solve the following equation numerically for $\xi \in \mathbb{R}^{n_d - 1}$; $\xi^T \Lambda^{(n_d-1)} \xi + 2\lambda^T \xi + \alpha =$ rhs. The equation was solved using the MATLAB function `fsolve`.

iv. Let $\widetilde{\mu} = \text{round}(\xi)$. If $\widetilde{\mu}^T \Lambda^{(n_d-1)} \widetilde{\mu} + 2\lambda^T \widetilde{\mu} + \alpha > 0$ then let

$$\widetilde{U}_{dd} = \begin{bmatrix} 1 & 0_{1,n_d-1} \\ \widetilde{\mu} & I_{n_d-1} \end{bmatrix}$$

and go to step v. Otherwise set rhs $=$ rhs $+ 1$ and go to step iii.

v. Let $\widetilde{\Lambda} = \widetilde{U}_{dd}^T \Lambda \widetilde{U}_{dd}$ and reorder the variables such that $(\widetilde{\Lambda})_{1,1} > 0$.

vi. If $(\widetilde{\Lambda})_{\text{ind,ind}} > 0$ set $(U_{dd})_{1,\text{ind}} = 0$, $(U_{dd})_{\text{ind,ind}} = 1$ and go to step ix. Otherwise go to step vii.

vii. Set μ_1 and μ_2 as the solutions to the following problems

$$\begin{aligned} \min_{\mu_1} \quad & \mu_1 \\ \text{s.t} \quad & (\widetilde{\Lambda})_{1,1}\mu_1^2 + 2(\widetilde{\Lambda})_{\text{ind},1}\mu_1 + (\widetilde{\Lambda})_{\text{ind,ind}} > 0, \quad \mu_1 \geq 0, \quad \mu_1 \in \mathbb{Z}. \\ \min_{\mu_2} \quad & \mu_2 \\ \text{s.t} \quad & (\widetilde{\Lambda})_{1,1}\mu_2^2 - 2(\widetilde{\Lambda})_{\text{ind},1}\mu_2 + (\widetilde{\Lambda})_{\text{ind,ind}} > 0, \quad \mu_2 \geq 0, \quad \mu_2 \in \mathbb{Z}. \end{aligned}$$

viii. If $\mu_1 < \mu_2$ set $(U_{dd})_{1,\text{ind}} = \mu_1$ and $(U_{dd})_{\text{ind,ind}} = 1$ otherwise set $(U_{dd})_{1,\text{ind}} = \mu_2$ and $(U_{dd})_{\text{ind,ind}} = -1$.

ix. If ind $= n$ go to step x otherwise set ind $=$ ind $+ 1$ and go to step vi.

x. Return $\widetilde{U}_{dd}$, U_{dd} and the variable reorderings used during the algorithm.

For each variable involved in a bilinear term of this form we make the substitution

$$y_i = e^{Y_i}.$$

Since we only make the substitutions in the bilinear terms and not in the univariate terms or the constraints we add the following constraint to the problem

$$Y_i = \ln y_i. \tag{4.43}$$

To linearise this constraint we note that y_i can be expressed as a sum of binary variables as follows:

$$\begin{aligned} y_i &= 1 + \sum_{k=1}^{m_i} t_{ik} k, \\ \sum_{k=1}^{m_i} t_{ik} &\leq 1, \\ t_{ik} &\in \{0, 1\}, \end{aligned} \tag{4.44}$$

where

$$m_i = u_i - 1.$$

Substituting this into (4.43) and noting that at most one of the binary variables can be non-zero, we obtain the following

$$Y_i = \sum_{k=1}^{m_i} t_{ik} \ln (1 + k).$$

A bilinear term ay_iy_j will take the following form after the substitution:

$$\begin{aligned} \min \quad & a e^{Y_i + Y_j} \\ \text{s.t.} \quad & y_i = 1 + \sum_{k=1}^{m_i} t_{ik} k, \\ & \sum_{k=1}^{m_i} t_{ik} \leq 1, \\ & Y_i = \sum_{k=1}^{m_i} t_{ik} \ln (1 + k), \\ & y_j = 1 + \sum_{k=1}^{m_j} t_{jk} k, \end{aligned} \tag{4.45}$$

$$
\begin{aligned}
&\sum_{k=1}^{m_j} t_{jk} \leq 1, \\
&Y_j = \sum_{k=1}^{m_j} t_{jk} \ln(1+k), \\
&y_i, y_j, Y_i, Y_j \in \mathbb{R}, \\
&t_{ij} \in \{0,1\}.
\end{aligned}
$$

We now consider bilinear terms of the form

$$-a y_i y_j, \quad a > 0,\ i \neq j.$$

For each variable involved in a bilinear term of this form we make the substitution

$$y_i = \sqrt{Y_i}.$$

Again we only make this substitution in the bilinear terms and not in the univariate terms or the constraints. Accordingly, we add the following constraint to the problem

$$Y_i = y_i^2.$$

Substituting (4.44) into this constraint and again noting that at most one of the binary variables can be non-zero, we obtain the following

$$Y_i = 1 + \sum_{k=1}^{m_i} t_{ik} \left((k+1)^2 - 1 \right).$$

A bilinear term $-a y_i y_j$ will take the following form after the substitution:

$$
\begin{aligned}
\min \quad & -a\sqrt{Y_i Y_j} && (4.46)\\
\text{s.t.} \quad & y_i = 1 + \sum_{k=1}^{m_i} t_{ik} k, \\
& \sum_{k=1}^{m_i} t_{ik} \leq 1, \\
& Y_i = 1 + \sum_{k=1}^{m_i} t_{ik} \left((k+1)^2 - 1 \right), \\
& y_j = 1 + \sum_{k=1}^{m_j} t_{jk} k, \\
& \sum_{k=1}^{m_j} t_{jk} \leq 1,
\end{aligned}
$$

$$
\begin{aligned}
&Y_j = 1 + \sum_{k=1}^{m_j} t_{jk} \left((k+1)^2 - 1 \right), \\
&y_i, y_j, Y_i, Y_j \in \mathbb{R}, \\
&t_{ij} \in \{0, 1\}.
\end{aligned}
$$

We now substitute (4.45) and (4.46) into each of the bilinear terms in the objective function of problem (4.42). The resulting problem is a convex MINLP which is equivalent to problem (1.2) [122]. MIQTBC can be applied to any problem where the n_cth principal leading submatrix is positive definite and the elements of x can be reordered such that $H_{dd} - H_{cd}^T H_{cc}^{-1} H_{cd}$ has at least two principal leading submatrices which are not negative semidefinite.

4.4 Approach used when H_{cc} is singular

When H_{cc} is singular we again want to reduce the number of bilinear terms that must be underestimated in the objective function, for the same reasons that were given in section 4.2. Towards this end we choose the elements of V such that the Hessian Θ of the transformed problem, which is given in (4.5), can be written in the following form

$$
\begin{aligned}
&\Theta = \Theta^{(1)} + \Theta^{(2)}, \\
&\Theta = \begin{bmatrix} \Theta_{cc}^{(1)} & 0 \\ 0 & \Theta_{dd}^{(1)} \end{bmatrix} + \begin{bmatrix} \Theta_{cc}^{(2)} & \Theta_{cd}^{(2)} \\ \Theta_{cd}^{(2)T} & \Theta_{dd}^{(2)} \end{bmatrix},
\end{aligned} \tag{4.47}
$$

where $\Theta_{cc}^{(1)}$, $\Theta_{cc}^{(2)}$ and $\Theta_{dd}^{(2)}$ are diagonal and $\Theta^{(2)}$ is positive definite. The resulting problem will be solved using a Branch and Bound algorithm. Since $\Theta^{(2)}$ is positive definite none of the terms in $\Theta^{(2)}$ need to be underestimated when obtaining the lower bounds. Therefore, following the same reasoning as in Theorem 1, we will have a transformed problem with $\frac{1}{2}\left(n_c^2 - n_c\right) + n_c n_d$ fewer bilinear terms in the objective function that need to be underestimated.

We now show that there exists a matrix V which gives the Hessian of the transformed problem the form specified by (4.47). As before we require U_{cc} to be a matrix which diagonalises H_{cc}. Since H_{cc} is Hermitian this matrix must exist [15]. We now prove the existence of the required transformation.

Theorem 5. *$\exists\ U_{cc}$ such that U_{cc} diagonalises H_{cc} and Θ can be written in the following form*

$$
\Theta = \begin{bmatrix} \Theta_{cc}^{(1)} & 0 \\ 0 & \Theta_{dd}^{(1)} \end{bmatrix} + \Theta^{(2)}, \tag{4.48}
$$

where $\Theta^{(2)}$ is positive definite.

Proof. Let

$$\begin{aligned}
A &= U_{cc}^T H_{cc} U_{cc}, \\
B &= U_{cc}^T \left(H_{cc} U_{cd} + H_{cd} U_{dd} \right), \\
C &= U_{cd}^T H_{cc} U_{cd} + U_{cd}^T H_{cd} U_{dd} + U_{dd}^T H_{cd}^T U_{cd} + U_{dd}^T H_{dd} U_{dd}.
\end{aligned}$$

The Hessian in (4.5) now takes the following form

$$\begin{aligned}
\Theta &= \begin{bmatrix} A & B \\ B^T & C \end{bmatrix}, \\
\Theta &= \begin{bmatrix} A^f & 0 \\ 0 & C^f \end{bmatrix} + \begin{bmatrix} A^d & B \\ B^T & C^d \end{bmatrix},
\end{aligned} \tag{4.49}$$

where A^d and C^d are defined as diagonal matrices with positive values on the diagonal and A^f and C^f are defined such that

$$\begin{aligned}
A &= A^d + A^f, \\
C &= C^d + C^f.
\end{aligned}$$

Denote the second matrix in (4.49) as $\Theta^{(2)}$. Now the elements of U_{cd} and U_{dd} can be taken to be fixed, the methods used to fix these matrices will be discussed after the proof of the theorem. Now using the definition of B and the fact that U_{cd} and U_{dd} are fixed we see that the elements of B can be written in the following form

$$\left(B\right)_{j,k} = \sum_{m=1}^{n_c} \rho_m^{(k)} \left(U_{cc}\right)_{m,j}, \tag{4.50}$$

where $\rho_m^{(k)} \in \mathbb{R}$. Now we know that U_{cc} must be a matrix which diagonalises H_{cc} but this does not specify U_{cc} uniquely. If F is some matrix which diagonalises H_{cc} then the matrix U_{cc} obtained by multiplying each column of F by some real number will also diagonlise H_{cc} [15]. Therefore the magnitude of the elements of U_{cc} can be made arbitrarily small. It is then clear from (4.50) that the elements of U_{cc} can be chosen to make the magnitude of the elements of B arbitrarily small.

A matrix $F \in \mathbb{R}^{(n,n)}$ is said to be strictly diagonally dominant if [74]

$$\left|\left(F\right)_{i,i}\right| > \sum_{\substack{j=1 \\ j \neq i}}^{n} \left|\left(F\right)_{i,j}\right| \quad \forall i = 1, \ldots, n.$$

Now a strictly diagonally dominant matrix which is Hermitian and has positive diagonal elements is positive definite [74]. Now from the definitions of A^d

and C^d we know that the $\Theta^{(2)}$ has positive diagonal elements and it is obvious that $\Theta^{(2)}$ is Hermitian. We need to show that we can choose the elements of U_{cc} such that $\Theta^{(2)}$ is strictly diagonally dominant. We have shown above that the magnitude of the elements of B can be made arbitrarily small and we know that A^d and C^d are diagonal matrices. Therefore the magnitude of the off diagonal elements of $\Theta^{(2)}$ can be made arbitrarily small. It is then obvious that we can choose U_{cc} such that $\Theta^{(2)}$ is strictly diagonally dominant. Therefore $\exists\ U_{cc}$ such that U_{cc} diagonalises H_{cc} and Θ can be written in the following form

$$\Theta = \begin{bmatrix} \Theta_{cc}^{(1)} & 0 \\ 0 & \Theta_{dd}^{(1)} \end{bmatrix} + \Theta^{(2)},$$

where $\Theta^{(2)}$ is positive definite and $\Theta_{cc}^{(1)}$ is diagonal. □

We now discuss the methods used to fix the values of U_{cd} and U_{dd}. The method used to set U_{dd} is the same as that described in subsection 4.2. In fixing U_{cd} we note that although there will always exist some U_{cc} such that Theorem 5 is satisfied the elements of U_{cc} might be very small. This tends to increase the bounds on the transformed problem calculated using (4.8) and (4.9). We attempted to use our freedom in the choice of U_{cd} to minimise this effect. A number of methods were developed to try and achieve this, the most promising of which sets as many of the elements of $H_{cc}U_{cd} + H_{cd}U_{dd}$ to zero as possible. This was done because we need to use U_{cc} to make the elements of B small enough to make $\Theta^{(2)}$ positive definite. It was reasoned that since $B = U_{cc}^T(H_{cc}U_{cd} + H_{cd}U_{dd})$, if $H_{cc}U_{cd} + H_{cd}U_{dd}$ had a large number of zero elements then the elements of U_{cc} could be made larger while still satisfying (4.48). The efficiency of this choice of U_{cd} was tested. However, it was found through numerical experiment that the most efficient value for U_{cd} was the zero matrix. There may be more efficient choice of U_{cd} than that found in this thesis and the choice of this matrix warrants further investigation. Algorithm 4 was used to find U_{cc} satisfying (4.48). In Algorithm 4 μ, ν and ω are parameters set by the user which determine the size of the elements of U_{cc}, A^d and C^d respectively. The effectiveness of the transformation is dependent on the choice of μ, ν and ω. These parameters will need to be determined by numerical experiment for each type of problem solved. The values used in this work are given in chapter 6.

Using the specified values of U_{cd}, U_{dd} and U_{cc} we obtained the following

Algorithm 4 The U_{cc} selection algorithm for singular H_{cc}

i. Set $\mu > 0$, $\nu > 0$ and $\omega > 0$. Set $r = 0$.

ii. Let $\tilde{U}_{cc}$ be the matrix with the normalised eigenvectors of H_{cc} as its columns.

iii. Let $U_{cc} = \mu\tilde{U}_{cc}$, $\chi = \nu \max(\text{abs}(A))$ and $\zeta = \omega \max(\text{abs}(C))$, where A and C are defined prior to (4.49) and $\max(F)$ denotes the largest element of the matrix F and $\text{abs}(F)$ denotes the matrix obtained by taking the absolute values of the elements of F.

iv. Set $\tilde{\Theta}^{(2)}$ to the following matrix

$$\tilde{\Theta}^{(2)} = \begin{bmatrix} \chi I_{n_c} & B \\ B^T & \zeta I_{n_d} \end{bmatrix},$$

where B is defined prior to (4.49).

v. Test whether $\tilde{\Theta}^{(2)}$ is positive definite. If it is then go to step viii, otherwise go to step vi.

vi. Set $r = r + 1$ and $u^{(r)} = 0.9u^{(r)}$ where $u^{(r)}$ is the rth column of U_{cc}.

vii. If $r = n_c$ set $r = 0$. Go to step iv.

viii. Let $\Theta^{(2)} = \tilde{\Theta}^{(2)}$. Return $\Theta^{(2)}$ and U_{cc} and stop.

transformed problem:

$$\begin{aligned}
\min_{y} \quad & h(Vy) = \frac{1}{2}\left(y_c^T \Theta_{cc}^{(1)} y_c + y_d^T \Theta_{dd}^{(1)} y_d\right) + \frac{1}{2} y^T \Theta^{(2)} y + g^T V y \qquad (4.51)\\
\text{s.t.} \quad & AVy \leq b,\\
& DVy = e,\\
& l \leq Vy \leq u,\\
& y^L \leq y \leq y^U,\\
& y = \left[y_c^T,\, y_d^T\right]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d},
\end{aligned}$$

where $\Theta_{cc}^{(1)}$ is diagonal and $\Theta^{(2)}$ is positive definite. As noted at the start of this section, when obtaining the lower bounds we do not need to underestimate the bilinear terms in $y^T\Theta^{(2)}y$ since $\Theta^{(2)}$ is positive definite.

4.5 Conclusion

In this chapter we have developed a number of different approaches for use in solving non-convex MIQPs. Different approaches were developed for different classes of Hessians. All of the approaches developed in this chapter are based on carefully chosen linear transformations. When the n_cth principal leading submatrix is positive semidefinite the problem is solved by reformulating it as a convex MIQP. We use the linear transformation to reduce the number of terms required by the reformulation. The resulting problem can be solved using convex mixed integer algorithms. A different approach is taken when solving problems with invertible and singular n_cth principal leading submatrices. These problems are solved using Branch and Bound algorithms. The linear transformations in these cases are chosen in such a way that more efficient lower bounding problems can be obtained at the nodes of the Branch and Bound tree. Computational results illustrating the effectiveness the approaches developed in this chapter are given in chapter 6.

Chapter 5

Derivative Free Methods for MINLPs

In this chapter we present a trust region algorithm for finding local minima of problem (1.1) without using the derivative of the objective function. The algorithm makes use of a series of quadratic approximations to the objective function and is adapted from the BOBYQA algorithm for derivative free, bound constrained continuous optimization [126]. A detailed review of BOBYQA was given in chapter 3. Three different implementations of the algorithm are presented and deterministic proofs of convergence to local minima are provided for two of the implementations.

This chapter is organised as follows. In section 5.1 we discuss possible definitions of local minima for mixed integer programs. This is necessary since there are a number of possible definitions and it is important to have a clear definition of a local minimum when developing deterministic algorithms. Section 5.2 contains the details of the new derivative free algorithm. Details of the three different implementations of the algorithm are given as well as the convergence proofs for two of the implementations. Concluding remarks are made in section 5.3. The pseudocode for the derivative free algorithm is given in appendix A.

5.1 Local minima of mixed integer programs

Clearly, the first step in designing a deterministic local search procedure should be to develop a mathematical formulation of a local minimum. Any definition of a local minimum will have the form [119]

Definition 6. *A point x^* is a local minimum if,*

$$f(x^*) \leq f(x), \quad \forall x \in \mathcal{N}_{\min}(x^*),$$

where $\mathcal{N}_{\min}(x)$ is some neighbourhood of x.

As we shall see below the definitions of local minima in both the continuous and discrete cases take this form. The mixed integer nature of the variables in problem (1.1) makes it possible to envisage several different definitions of $\mathcal{N}_{\min}(x)$, each of which gives a different definition of a local minimum. We discuss several possible definitions in this section as well as suggesting some restrictions that could be used to decide which of these definitions to use. While these restrictions are applied to bound constrained MINLPs in this work they are developed for the general MINLPs with the form of problem (2.1).

Before proceeding with this discussion we need to define the following notation. Let Ω_m denote the feasible region of problem (2.1). In the special cases when $n_d = 0$ and $n_c = 0$ let Ω_c and Ω_d respectively denote the feasible region of problem (2.1) . For any x_c and $\varepsilon > 0$ let $B_\varepsilon(x_c)$ denote the open ball $\{y_c \in \mathbb{R}^{n_c} : \|y_c - x_c\| < \varepsilon\}$. Let $\mathcal{N}_m(x)$ denote a neighbourhood used in the definition of a local minimum of a mixed integer problem. We assume that $x \in \mathcal{N}_m(x)$. Let $\mathcal{N}_d(x_d)$ denote a discrete, user defined neighbourhood. Let $\mathcal{N}_r(x)$ denote a neighbourhood of x of the form

$$\mathcal{N}_r(x) = \{y \in \mathbb{R}^n : y_c = x_c,\, y_d \in \mathcal{N}_d(x_d)\}.$$

$\mathcal{N}_r(x)$ is just a commonly used subset of $\mathcal{N}_m(x)$. Now suppose we give x_d some feasible value in $f(x)$. The remaining freedom in the continuous variables will form a manifold. We refer to the feasible region of this manifold as a *feasible continuous manifold*. We also need to recall the continuous and discrete definitions of local minima as well as the definition of a global minimum before proceeding with our discussion. For continuous problems, $n_d = 0$, the following definition of a local minimum is used [119]

Definition 7. (Continuous local minimum) *A point $x^* \in \Omega_c$ is a local minimum if, for some $\epsilon > 0$,*

$$f(x^*) \leq f(x), \quad \forall x \in B_\varepsilon(x^*).$$

For discrete problems, $n_c = 0$, the following definition of a local minimum is used [160]

Definition 8. (Discrete local minimum) *A point $x^* \in \Omega_d$ is a local minimum if,*

$$f(x^*) \leq f(x), \quad \forall x \in \mathcal{N}_d(x^*).$$

The following definition of a global minimum is used for continuous [119], discrete [99] and mixed problems [120]

Definition 9. (Global minimum) *A point $x^* \in \Omega_m$ is a global minimum if,*

$$f(x^*) \leq f(x), \quad \forall x \in \Omega_m.$$

This completes the list of definitions required to allow us to proceed with our discussion of mixed integer local minima.

As was noted above a local minimum of a mixed problem can be defined in a number of ways. We suggest that some further restrictions need to be imposed on the definition to allow it to be chosen in a meaningful way. The existing definitions, which are discussed below, do not follow these restrictions and we shall provide an example illustrating the resulting deficiencies. The first two restrictions that we propose for the definition of a mixed local minimum are that it should reduce to Definition 7 when $n_d = 0$ and to Definition 8 when $n_c = 0$. The remaining restrictions ensure that desirable properties of Definitions 7 and 8 are retained in the mixed definition. Firstly, the fact that the neighbourhood in Definition 8 is user defined allows the user some control over the strength of the minimum obtained as well as the computational effort required to find the minimum. This is important since the problem is NP-hard; it may be prohibitively expensive to find local minima using a large neighbourhood, in which case a smaller neighbourhood can be used. Since the mixed problem is also NP-hard the mixed definition should allow the user some control over the size of the neighbourhood. In addition, we propose that the following are desirable characteristics of the discrete and continuous definitions and that the definition in the mixed case should have an analogous characteristic;

i) A point is a local minimum of a continuous, convex problem if and only if it is the global minimum.

ii) If $n_c = 0$ (discrete problem) and $\mathcal{N}_d(x) = \Omega_d$ then a point is a local minimum of the problem if and only if it is a global minimum.

The difference between these properties is that i) requires that the problem considered be convex while ii) requires that the neighbourhood include all feasible discrete points. Considering the mixed case it seems natural to combine these properties to give a restriction on the definition of a mixed minimum. The discussion above gives us the following restrictions on the definition of a local minimum

1) The definition of a mixed integer local minimum reduces to Definition 7 when $n_d = 0$.

2) The definition of a mixed integer local minimum reduces to Definition 8 when $n_c = 0$.

3) The definition of a mixed integer local minimum allows the user some control over the size of $\mathcal{N}_m$.

4) If $\mathcal{N}_m$ contains at least one point on each *feasible continuous manifold* and f and c_i are convex then a point is a local minimum of a mixed integer problem if and only if it is a global minimum.

It may be that more restrictions should be added to this list since these restrictions do not uniquely specify a definition of a local minimum, an example showing that this is the case is provided later in this section. However, it should be noted that the definition of a local minima is also not uniquely specified in the discrete case. The non-uniqueness of the definition might be an unavoidable effect of including discrete variables in the problem. While the restrictions listed above do not uniquely specify the definition of a local minima they do limit the possible choices enormously and allow us to define what we believe to be a practically meaningful definition of a local minimum.

We now discuss the two definitions of local minima that have been proposed in the literature. The following definition of a local minimum is used in [102] and [156],

Definition 10. (Separate local minimum) *A point $x^* \in \Omega_m$ is a local minimum of problem* (1.1) *if, for some $\epsilon > 0$,*

$$f(x^*) \leq f(x), \quad \forall x \in \{x : x_c \in B_\varepsilon(x_c^*),\, x_d = x_d^*\} \cap \Omega_m, \tag{5.1}$$
$$f(x^*) \leq f(x), \quad \forall x \in \mathcal{N}_r(x^*) \cap \Omega_m. \tag{5.2}$$

One of the implementations of our derivative free algorithm guarantees convergence to a separate local minimum. A stronger definition of a local minimum is used in [1, 2, 17, 105],

Definition 11. *A point $x^* \in \Omega_m$ is a local minimum of problem* (1.1) *if, for some $\epsilon > 0$,*

$$f(x^*) \leq f(x), \quad \forall x \in \left(\bigcup_{x \in \mathcal{N}_m(x^*)} B_\varepsilon(x_c) \times \{x_d\} \right) \bigcap \Omega_m,$$

where $\mathcal{N}_m(x)$ is some neighbourhood of x.

In all cases where $\mathcal{N}_m(x)$ is defined explicitly in this definition a neighbourhood with the form $\mathcal{N}_r(x)$ is used [17, 105]. We believe that this definition is too broad to be practically useful when $\mathcal{N}_m(x)$ is left in its most general form. Indeed, it is only slightly less general than Definition 6. Accordingly, from this point we shall only discuss the applied version of Definition 11.

The main problem with Definition 10 and the applied version of Definition 11 is that they do not satisfy restriction 4). We present an example below showing that this is the case. Another issue is that the definitions treat the continuous and discrete variables separately, ignoring the overall behaviour of the objective function. To overcome these problems we propose the following stronger definition of a local minimum.

Definition 12. (Combined local minimum) *A point $x^* \in \Omega_m$ is a local minimum of problem* (1.1) *if, for some $\epsilon > 0$,*

$$f(x^*) \leq f(x), \quad \forall x \in \{x : x_c \in B_\varepsilon(x_c^*),\, x_d = x_d^*\} \cap \Omega_m, \tag{5.3}$$

$$f(x^*) \leq f(x), \quad \forall x \in \mathcal{N}_{\text{comb}}(x^*) \cap \Omega_m. \tag{5.4}$$

Here $\mathcal{N}_{\text{comb}}(x^*)$ is the set of optimal points obtained when solving the following set of problems:

$$\left\{ \begin{array}{ll} \min\limits_{x} & f(x) \\ \text{s.t.} & l < x < u, \\ & x_d = \bar{x}_d, \\ & x = \left[x_c^T,\, x_d^T\right]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}, \end{array} \right| \; \bar{x} \in \mathcal{N}_r(x^*) \backslash \{x^*\} \Bigg\}. \tag{5.5}$$

These problems are solved using $\bar{x}$ as the initial point. The problems in (5.5) are continuous NLPs. Now clearly the points in $\mathcal{N}_{\text{comb}}(x)$ will depend on the method used to solve the problems in (5.5). Accordingly this definition is not fully specified unless the method used to solve these problems has been specified. The solver used should be deterministic with deterministic proof of convergence. The use of a stochastic method will result in different elements being contained in $\mathcal{N}_{\text{comb}}(x)$ each time the solver is run. From this point we shall call whichever solver is chosen by the user the neighbourhood solver. We also note that any point which is a combined local minimum will also be a separate local minimum. The converse is not true; separate local minima are not always combined local minima. We now prove that Definition 12 satisfies the restrictions we placed on the definition of a local minimum.

Theorem 13. *Definition 12 satisfies restrictions 1), 2), 3) and 4).*

Proof. Suppose that $n_d = 0$, in this case restriction 1) must be satisfied. From its definition $\mathcal{N}_r(x) = x$ when $n_d = 0$ which gives $\mathcal{N}_r(x) \backslash \{x\} = \emptyset$. Then from its definition it is clear that $\mathcal{N}_{\text{comb}}(x) = \emptyset$. Considering Definition 12 we see that it reduces to Definition 7 when $\mathcal{N}_{\text{comb}}(x) = \emptyset$ so restriction 1) is satisfied.

Now suppose that $n_c = 0$, in this case (5.3) reduces to

$$f(x^*) \leq f(x), \quad \forall x \in \{x : x = x^*\},$$

but this gives $f(x^*) \leq f(x^*)$ which is satisfied identically so it can be dropped from the definition. Now consider $\mathcal{N}_{\text{comb}}(x)$. The problems in (5.5) each contain the constraint $x = \bar{x}$ when $n_c = 0$ so the optimal point of each problem must be $\bar{x}$. Then, by its definition, $\mathcal{N}_{\text{comb}}(x)$ has the following form

$$\mathcal{N}_{\text{comb}}(x) = \{\bar{x} : \bar{x} \in \mathcal{N}_r(x) \backslash \{x\}\},$$
$$\mathcal{N}_{\text{comb}}(x) = \mathcal{N}_r(x) \backslash \{x\}.$$

Now consider Definition 8; clearly replacing $\mathcal{N}_r(x^*)$ with $\mathcal{N}_r(x^*) \backslash \{x^*\}$ will have no effect on which points satisfy the definition. We have shown that this is the form that Definition 12 takes when $n_c = 0$. Therefore Definition 12 is equivalent to Definition 8 when $n_c = 0$ and restriction 2) is satisfied.

Now, the number of points in $\mathcal{N}_{\text{comb}}(x)$ is determined by the number of points in the user defined neighbourhood $\mathcal{N}_r(x)$. Thus, Definition 12 satisfies restriction 3)

We now show that restriction 4) is satisfied. Consider a point x^* which satisfies Definition 12. Assume that f and c_i are convex and that the neighbourhood $\{x : x_c \in B_\varepsilon(x_c^*),\, x_d = x_d^*\} \cup \mathcal{N}_{\text{comb}}(x)$ contains at least one point on each *feasible continuous manifold*. From the definition of $\mathcal{N}_{\text{comb}}(x)$ we see that it will contain one point on each manifold besides that containing x^* and that each point will have the lowest objective function value on its manifold. Since x^* satisfies (5.3) it is the minimum on its continuous manifold. Therefore, there are no points with a lower objective function value than x^* on the manifold containing x^*. Now none of the points in $\mathcal{N}_{\text{comb}}(x)$ has a lower objective function value than x^* and each point in $\mathcal{N}_{\text{comb}}(x)$ is the minimum on one of the continuous manifolds. Therefore, there are no points on any of the other *feasible continuous manifolds* with a lower objective function value than x^*. Therefore x^* is a global minimum and restriction 4) is satisfied. □

It was stated previously that our restrictions do not uniquely specify a definition of a local minimum. That this statement is correct can be seen by using different neighbourhood solvers in the definition of $\mathcal{N}_{\text{comb}}(x)$ and considering non-convex problems. Clearly, different solvers are not guaranteed to return the same solutions for the problems in (5.5) when f or c_i are non-convex. It follows that Definition 12 is not unique. One of the implementations of our derivative free algorithm guarantees convergence to a combined local minimum.

We now show that Definition 10 and the applied version of Definition 11 do not satisfy restriction 4). This is done using a counterexample. The example is also used to illustrate the advantages of choosing a definition which satisfies our restrictions and the negative effects of ignoring said restrictions. Consider the following example:

$$\begin{aligned} \min_{[y,x]} \quad & \frac{5}{2}(x+y)^2 + \frac{1}{\sqrt{2}}(-x+y) \\ \text{s.t.} \quad & -2 \leq x, y \leq 2, \\ & y \in \mathbb{R},\, x \in \mathbb{Z}. \end{aligned} \tag{5.6}$$

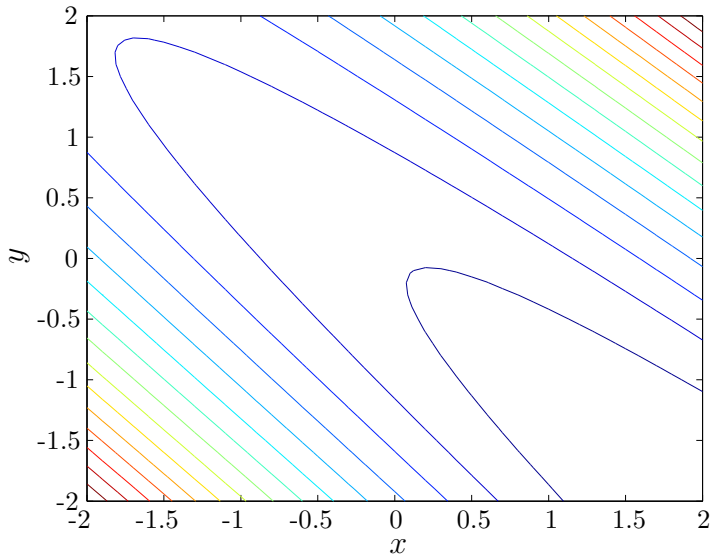
The constraints describing the feasible region of problem (5.6) are clearly convex and, since it is the sum of two convex terms, the objective function is also convex. The contours of the objective function are shown in Figure 5.1a. Figure 5.1b shows the feasible values of the objective function viewed along the x-axis. The green dots in Figure 5.1b are placed at the minimum of each parabola. Clearly each of the green points will satisfy (5.3), this is required by all three definitions. Now using Definition 10 or the applied version Definition 11 each green point will be a local minima, regardless of the choice of $\mathcal{N}_r([y,\, x])$. This is obvious from the fact that each green point gives the lowest possible objective function value for a specific value of y. Clearly then neither definition satisfies restriction 4). We also see here the dangers of choosing definitions without carefully considering their construction. Using the definitions in the literature fairly simple problems can be imagined for which the size $\mathcal{N}_d(x_d)$ will have no effect on whether a point is defined as a local minimum. This is clearly undesirable. Now consider Definition 12 with

$$\mathcal{N}_r\left([y,\, x]\right) = \{[\bar{y},\, \bar{x}] : \bar{y} = y,\, \bar{x} = -2, -1, \ldots, 2\}.$$

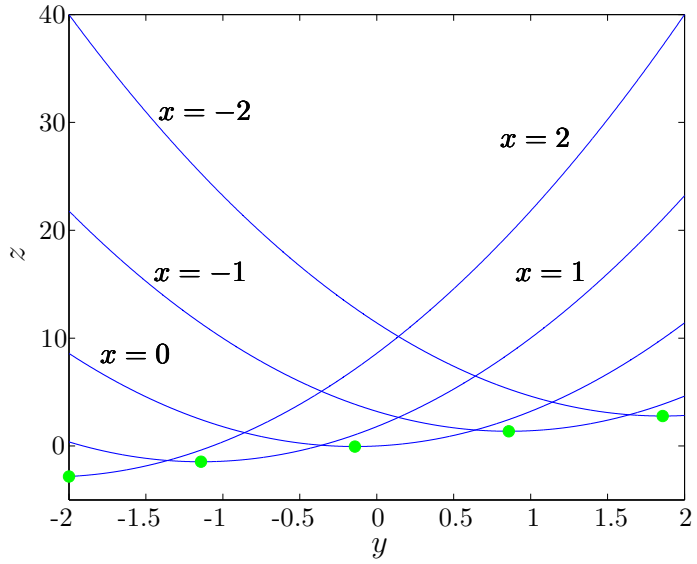
Using this definition of $\mathcal{N}_r$ there will be at least one point on each *feasible continuous manifold*. $\mathcal{N}_{\text{comb}}([y,\, x])$ will contain the four green points not lying on the same manifold as y. Clearly only the global minimum at $[2,\, -2]$ is a local minimum using this definition. We see that, as expected, Definition 12 satisfies restriction 4). In fact, for this problem the condition that the local minimum should be a global minimum does not require that a point in the neighbourhood lie on each manifold. This is clear from the fact that when applying Definition 12 with

$$\mathcal{N}_r\left([y,\, x]\right) = \{[\bar{y},\, \bar{x}] : \bar{y} = y,\, \bar{x} = x-1, x+1\},$$

only the global minimum at $[2,\, -2]$ is a local minimum.



(a) A contour plot of the objective function of problem (5.6).



(b) The feasible values of the objective function viewed along the x-axis. The green points are placed at the minimum of each parabola.

Figure 5.1: A figure illustrating important features of the objective function of problem (5.6).

5.2 Mixed integer algorithm

We now develop a trust region based, derivative free algorithm that can be used to solve problem (1.1). We present three different implementations of the algorithm; each implementation has different convergence results. The first implementation is a heuristic and is called HEMBOQA (HEuristic Mixed Bound constrained Optimization by Quadratic Approximation). The second implementation guarantees convergence to a separate local minimum, as defined in Definition 10, and is called SEMBOQA (SEparate Mixed Bound constrained Optimization by Quadratic Approximation). The third algorithm guarantees convergence to a combined local minimum, as defined in Definition 12, and is called COMBOQA (COmbined Mixed Bound constrained Optimization by Quadratic Approximation).

HEMBOQA is the adaption of BOBYQA to mixed integer programs. The differences between the two algorithms all stem from the fact that, as in BOBYQA, HEMBOQA requires all interpolation points to be feasible. Specifically the last n_d components of each interpolation point must be integral. The changes caused by the restriction that all of the interpolation points must be feasible will become clear in the following discussion. A description of the aspects of the HEMBOQA algorithm which differ from BOBYQA is given in sections 5.2.1 to 5.2.3. Pseudocode describing the execution of HEMBOQA is given in appendix A. SEMBOQA and COMBOQA are modifications of HEMBOQA. The details of the changes made to HEMBOQA to produce SEMBOQA and COMBOQA are given in section 5.2.4. Familiarity with the review of BOBYQA given in chapter 3 is required to completely understand the following discussion.

5.2.1 Initialisation

At the initialisation of the algorithm the user is required to specify the following quantities; an initial point x_0, an initial inner trust region vector $\rho_{\text{beg}} \in \mathbb{R}^n$, a final trust region vector $\rho_{\text{end}} \in \mathbb{R}^n$, the number of continuous variables n_c, the number of discrete variables n_d, the upper and lower bounds u and l, the maximum number of function evaluations $maxevals$ and the maximum time $maxtime$. The purposes of these quantities will be made clear in the following discussion. We note here that instead of the trust region radii used in BOBYQA, HEMBOQA uses trust region vectors. This is because HEMBOQA uses box shaped rather than spherical trust regions and each element of the trust region vector specifies the size of the box in one direction. The reasons for the use of the box shaped trust region are given in section 5.2.2. We also note that in the HEMBOQA algorithm we restrict

the number of interpolation points to have the value $m = 2n+1$ rather than allowing the user to choose m. Our description of BOBYQA in chapter 3 also used $m = 2n + 1$. However, in [126] provision is made for the user to choose the number of interpolation points.

We now discuss the procedure used to set up the initial set of interpolation points. Similarly to BOBYQA, the initial point x_0 may need to be moved to ensure that the interpolation points are feasible, this is done automatically. The elements of x_0 are reset using the following formula

$$(x_0)_i = \begin{cases} l_i, & (x_0)_i < l_i, \\ u_i, & (x_0)_i > u_i, \\ l_i + (\Delta_1)_i, & l_i < (x_0)_i < l_i + (\Delta_1)_i, \\ u_i - (\Delta_1)_i, & u_i > (x_0)_i > u_i - (\Delta_1)_i, \\ (x_0)_i, & \text{otherwise}, \end{cases} \tag{5.7}$$

where $(\Delta_1)_i$ is the ith component of Δ_1. When there are no binary variables in the problem the following procedure is used to set up the initial quadratic model and interpolation points. The initial inner and outer trust region vectors, ρ_1 and Δ_1, are both set to ρ_{beg}. The initial interpolation points are chosen as follows; set $y_1 = x_0$ and for $i = 1, \ldots, n$ define y_{i+1} and y_{n+i+1} as follows

$$\begin{cases} y_{i+1} = x_0 + (\Delta_1)_i e_i \quad y_{n+i+1} = x_0 - (\Delta_1)_i e_i, & \text{if } l_i < (x_0)_i < u_i, \\ y_{i+1} = x_0 + (\Delta_1)_i e_i \quad y_{n+i+1} = x_0 + 2(\Delta_1)_i e_i, & \text{if } (x_0)_i = l_i, \\ y_{i+1} = x_0 - (\Delta_1)_i e_i \quad y_{n+i+1} = x_0 - 2(\Delta_1)_i e_i, & \text{if } (x_0)_i = u_i. \end{cases} \tag{5.8}$$

This is just a modification of the formula used by BOBYQA which takes into account the fact that the trust regions are now box shaped. Since all of the interpolation points are required to be feasible an error is returned if $u_i - l_i < 2(\Delta_1)_i$. The initial quadratic model is set up by using the interpolation points to set p_1, q_1 and the diagonal elements of $\nabla^2 Q_1$. The remaining elements of $\nabla^2 Q_1$ are set equal to zero; this can be thought of as minimizing the Frobenius norm of $\nabla^2 Q_1$.

However, if binary variables are present in the problem we need to modify this procedure since using (5.8) when binary variable are present results in infeasible interpolation points. The procedure described below is used to overcome this problem. Suppose the ith variable is binary. In the following discussion we use i to index both interpolation points and specific coordinates of interpolation points, e.g. $(y_{i+1})_i$ is the ith coordinate of the $(i+1)$th interpolation point. Now, set $(x_0)_i = 0$ and $(\Delta_1)_i = 0.5$ and use (5.8) to set up the initial interpolation points. This will result in infeasible interpolation

points. These points are used to set up W_1. However, as we shall see the objective function is never evaluated at these points. Use the procedure in [126] to set up W_1. Then for each binary coordinate we create a new interpolation point y_{rep}, i.e. if the problem contains four binary variables then four new interpolation points will be generated. The new point is constructed by setting $(y_{\text{rep}})_i = 1$ and

$$(y_{\text{rep}})_j = \begin{cases} 1, & x_j \text{ is binary,} \\ (x_0)_j + (\Delta_1)_j, & (x_0)_j + (\Delta_1)_j < u_j \text{ and } x_j \text{ is not binary,} \\ (x_0)_j - (\Delta_1)_j, & \text{otherwise.} \end{cases} \quad (5.9)$$

Here j is an index given by

$$j = \begin{cases} 1, & i = n, \\ i + 1, & \text{otherwise.} \end{cases}$$

The remaining elements of y_{rep} are set equal to the corresponding elements of x_0. The point y_{rep} will always be feasible since x_0 is set using (5.7) and $u_i - l_i \geq 2(\Delta_1)_i$. We replace y_{i+1} with y_{rep} in the set of interpolation points and update W_1 using the procedure described in [126] using $x_0 = x_k$. Once this procedure has been performed for all of the binary variables we have a set of feasible interpolation points. We use this new set of interpolation points to set up Q_1. However, due to the new structure of the interpolation points, additional freedom is now required in order to satisfy the initial interpolation conditions, as explained in the following paragraph. Accordingly, we now use the interpolation points to set p_1, q_1, the diagonal elements of $\nabla^2 Q_1$ and the off diagonal element in the ith row and jth column of $\nabla^2 Q_1$ for each i corresponding to a binary variable.

We now explain why the off diagonal elements of $\nabla^2 Q_1$ have to be used in this case. Normally the coefficients of the initial quadratic model are found by solving a system of linear equations with the form $Fz = b$ where $F \in \mathbb{R}^{m,m}$ and $z, b \in \mathbb{R}^m$. The vector b contains the objective function values at the interpolation points, z is a vector of the coefficients of Q_1 and the vth row of F will have the following form

$$\left[(y_v)_1^2 \;\; (y_v)_2^2 \;\; \ldots \;\; (y_v)_n^2 \;\; (y_v)_1 \;\; (y_v)_2 \;\; \ldots \;\; (y_v)_n \;\; 1 \right].$$

Now the system $Fz = b$ will only have a solution if $\text{rank}(F) = m$ [15]. The rank of F is equal to the number of linearly independent rows of F. Normally the procedure used to set up the interpolation points ensures that they are linearly independent which in turn ensures that $\text{rank}(F) = m$. However,

considering (5.8) and the procedure used to set up y_{rep} we see that we have $y_{\text{rep}} = y_{i+n+1} + y_{j+1} - y_1$. Since one of the rows of F is a linear combination of three others $\text{rank}(F) = m-1$ so we will not be able to find the coefficients of Q_1. We now show that by using the off diagonal element in the ith row and jth column of $\nabla^2 Q_1$ we make it possible solve the system. The vth row of F now takes the form

$$\left[(y_v)_1^2 \;\; (y_v)_2^2 \;\; \ldots \;\; (y_v)_n^2 \;\; (y_v)_1 \;\; (y_v)_2 \;\; \ldots \;\; (y_v)_n \;\; 1 \;\; (y_v)_i \times (y_v)_j\right].$$

Adding this column to the matrix cannot make any of the previously independent rows dependent so if

$$(y_{\text{rep}})_i (y_{\text{rep}})_j \neq (y_{i+n+1})_i (y_{i+n+1})_j + (y_{j+1})_i (y_{j+1})_j - (y_1)_i (y_1)_j, \tag{5.10}$$

then we will have $\text{rank}(F) = m$. From the discussion above we see that we have $y_1 = x_0$, $(x_0)_i = 0$, $(y_{\text{rep}})_i = 1$, $(y_{j+1})_i = (x_0)_i$, $(y_{i+n+1})_j = (x_0)_j$ and $(y_{i+n+1})_i = 1$. Using this information we can rewrite (5.10) as follows

$$(y_{\text{rep}})_j \neq (x_0)_j. \tag{5.11}$$

From (5.9) we see that (5.11) will be satisfied provided that $(\Delta_1)_j \neq 0$. We must have $(\Delta_1)_j > 0$ since, as noted above, an error is returned if $u_i - l_i < 2(\Delta_1)_i$. Therefore (5.11) is satisfied so $\text{rank}(F) = m$ and the system $Fz = b$ has a solution, allowing the coefficients of Q_1 to be found.

5.2.2 Trust Region and Alternative Iterations

During trust region iterations the following problem is solved to find d_k:

$$\begin{aligned} \min_{d_k} \quad & Q_k(x_k + d_k) \\ \text{s.t.} \quad & l \leq x_k + d_k \leq u, \\ & -\Delta_k \leq d_k \leq \Delta_k, \\ & d_k^T = \left[d_k^{cT}, d_k^{dT}\right]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}, \end{aligned} \tag{5.12}$$

where $\Delta_k \in \mathbb{R}^n$. We note that we have replaced the spherical trust region used by BOBYQA with a box shaped trust region. This is done for two reasons. Firstly it simplifies the MIQP that must be solved to calculate d_k since the bound constraints in problem (5.12) are easier to handle than the quadratic constraints that would result from the use of a spherical trust region. The MIQPs are NP-hard and accordingly they are the most computationally expensive part of the algorithm. Clearly then, making the MIQPs

easier to solve should reduce the solution time. The second reason for the use of a box shaped trust region is that it allows us to have different trust region sizes for different coordinate directions. This is vital to the success of the algorithm as the trust region needs to become much smaller than one in the continuous coordinate directions to give a good solution. However if the size of the trust region becomes smaller then one in the integer coordinates we will no longer be able to change the discrete part of our solution. This will clearly inhibit the algorithm since the size of the trust region in the continuous coordinates generally becomes smaller than one fairly soon after the initialisation of the algorithm. This change in the form of the trust region forces us to modify a number of the formulae used in BOBYQA, these changes will be given in section 5.2.3. A similar modification is made to the problems solved during the alternative iterations. During alternative iterations the step d_k is found by solving the following problem:

$$\begin{aligned} \max_{d_k} \quad & |\Lambda_t(x_k + d_k)| \\ \text{s.t.} \quad & l \leq x_k + d_k \leq u, \\ & -\Delta_k \leq d_k \leq \Delta_k, \\ & d_k^T = \left[d_k^{cT},\, d_k^{dT} \right]^T \in \mathbb{R}^{n_c} \times \mathbb{Z}^{n_d}, \end{aligned} \tag{5.13}$$

where Λ_t is defined in chapter 3.

Both problem (5.12) and problem (5.13) are MIQPs. These problems were solved using a number of different methods. The method applied in a particular case depended on the Hessian of the problem. If the Hessian was positive definite the problems were solved using CPLEX. If the Hessian was indefinite but the n_cth principle leading submatrix of the Hessian was positive semidefinite the problem was solved using a convex reformulation scheme developed in [33]. If the n_cth principal leading submatrix was indefinite the problem was solved using the methods developed chapter 4.

5.2.3 Further details on the implementation of HEMBOQA

In this section we give the remaining details required to complete the description of HEMBOQA. First, a procedure is given which is used to ensure that a good solution is obtained in the continuous variables. We then describe the procedure used when patterns appear in the step lengths. Following that we outline the procedures used to choose t and to choose between trust region and alternative iterations. Finally, the trust region management procedure is given.

Before proceeding with the discussion outlined in the previous paragraph we give two further brief points on the implementation of HEMBOQA. Firstly, the procedure used by HEMBOQA to update Q_k and W_k is the same as the procedure used by BOBYQA. Secondly, if HEMBOQA exceeds the user specified run time or number of function evaluations the algorithm is stopped and the current x_k and $f(x_k)$ are returned.

Ensuring a good solution in the continuous variables

We now describe a feature of HEMBOQA used to ensure that the continuous part of the solution is accurate. Let $\mathcal{X}_i$ denote the *feasible continuous manifold* obtained by fixing the integer dimensions in the feasible region to have the values of y_i^d. Recall that s denotes the integer which satisfies $y_s = x_k$. Now it can happen that x_k is the only interpolation point on $\mathcal{X}_s$[1]. If this is the case it is unlikely that HEMBOQA will find an accurate estimate of the optimal point on $\mathcal{X}_s$; the location of this optimal point is required by all of our definitions of local minima. To allow HEMBOQA to obtain a more accurate estimate of the optimal point on $\mathcal{X}_s$ we require that there be a user specified number γ_{req} of interpolation points on $\mathcal{X}_s$ before the size of the trust region can be reduced and before the algorithm can be stopped. The value of γ_{req} chosen will depend on the accuracy required and the desired number of function evaluations; decreasing γ_{req} will result in a smaller number of function evaluations but a less accurate solution. Now let γ_{in} denote the number of interpolation points on $\mathcal{X}_s$. A number of steps are taken to try and ensure that $\gamma_{\text{in}} \geq \gamma_{\text{req}}$. Firstly, if $\gamma_{\text{in}} < \gamma_{\text{req}}$ none of the interpolation points on $\mathcal{X}_s$ can be replaced by $x_k + d_k$ when choosing the set of interpolation points for the $(k+1)$th iteration. Obviously, when trying to increase the number of points on $\mathcal{X}_s$ it is undesirable to remove any points already on $\mathcal{X}_s$. In addition if $\gamma_{\text{in}} < \gamma_{\text{req}}$ it is made more likely that a trust region iteration will be used rather than an alternative one. This done since numerical experiments show that trust region iterations are far more likely to produce points on $\mathcal{X}_s$ than alternative iterations. Despite these measures it is possible that an extremely large number of iterations will be completed before we obtain $\gamma_{\text{in}} \geq \gamma_{\text{req}}$. The following procedure is used to overcome this effect. If there have been more than 20 iterations during which x_k has remained constant and $x_k^d \neq x_k^d + d_k^d$ then any trust region step with $x_k^d = x_k^d + d_k^d$ will be accepted. In addition if there are 100 consecutive iterations during which x_k has remained constant

[1] An interpolation point lies on $\mathcal{X}_s$ if its integer components are equal to the integer components of x_k.

and $x_k^d \neq x_k^d + d_k^d$ we use the neighbourhood solver[2] to find a locally optimal point on $\mathcal{X}_s$ starting from x_k. We then set x_k to be the point returned by the neighbourhood solver and return x_k and $f(x_k)$ with a warning. The numbers 20 and 100 were chosen by numerical experiment.

Handling patterns in the step lengths

In this section we describe a procedure that is used to allow the algorithm to proceed if patterns appear in the step lengths. The following example will be used to illustrate what we mean when we speak of patterns in the step lengths. Consider a problem with $n = 2$ and suppose we find the following sequence of new interpolation points; $x_1 + d_1 = [1, 2]^T$, $x_2 + d_2 = [3, 4]^T$, $x_3 + d_3 = [5, 6]^T$, $x_4 + d_4 = [1, 2]^T$, $x_5 + d_5 = [3, 4]^T$ and $x_6 + d_6 = [5, 6]^T$. We see that there is a pattern in this sequence of new interpolation points. If a sequence such as the one described in our example occurs it will keep repeating, with no decrease in the objective function value, until the maximum number of function evaluations or the maximum time is reached. Clearly this is not a desirable situation.

A procedure has been developed which attempts to break patterns once they are detected. To allow patterns to be detected the previous interpolation points used by the algorithm must be stored. However, it will be impractical to store all of the new points found by HEMBOQA. Instead the number of previous points stored is specified by the user; this is a further quantity which must be specified by the user during the initialisation step. The number of points stored is denoted by *looplength*. If the size of the pattern is greater than half of *looplength* the pattern will not be detected. For instance, in the example above, which has a pattern length of three, if four interpolation points are stored HEMBOQA can only detect one duplicated point and it clearly needs to detect all three duplicated points to identify the pattern. When a pattern is detected the current value of d_k is discarded and a new step is chosen as follows. We select a random integer i from the set $\{1, \ldots, m\} \backslash \{s\}$[3]. We then use the neighbourhood solver to find a locally optimal point on $\mathcal{X}_i$ using y_i as a starting point, call this point x_{new}. If $x_{\text{new}} = x_k + d_k$ (here d_k is the step length which is discarded when the pattern is detected) or x_{new} is equal to one of the current interpolation points then choose a different interpolation point and repeat the process. If all of the interpolation points have been considered we use the neighbourhood

[2] As noted previously, the neighbourhood solver should be a deterministic solver which will return a locally optimal point.

[3] s is excluded since we require that HEMBOQA never be able to replace the best interpolation point.

solver to find a locally optimal point on $\mathcal{X}_s$ starting from x_k. Call this point $x_{k(new)}$ We then set $x_k := x_{k(new)}$ and return x_k and $f(x_k)$ with a warning. A warning is returned since the algorithm is only stopped because we have failed to break out of the pattern, it is therefore unlikely that the returned solution is optimal. Once we have found some x_{new} such that $x_{\text{new}} \neq x_k + d_k$ and x_{new} is not equal to one of the current interpolation points we choose a new d_k such that $x_k + d_k = x_{\text{new}}$.

Choosing t and the type of iteration

In first part of this section we discuss the methods used to select t during trust region and alternative iterations. Two objectives are considered when choosing t. Firstly, it is desirable to cluster the interpolation points around x_k. In addition, it is necessary to ensure that the value of σ is acceptably large; this helps prevent numerical errors in the updating process. The clustering of the interpolation points around x_k improves the accuracy of the quadratic model around x_k. This is important when checking that x_k is a minimum, indeed we shall see later that one of the termination conditions of the algorithm is that γ_{req} interpolation points on $\mathcal{X}_s$ lie within a certain neighbourhood of x_k.

During trust region iterations t is chosen to be the integer which maximises the following quantity

$$\max \left[1, \frac{\|y_t - x_k\|^2}{\prod_i (\Delta_k)_i} \right] \sigma_t. \tag{5.14}$$

where σ_t denotes the value of σ if t is chosen. Clearly, when choosing t using this condition both the clustering of the interpolation points and the value of σ are taken into account. However, as in BOBYQA, when $f(x_k + d_k) < f(x_k)$ it my be desirable to choose a different t. Accordingly when $f(x_k + d_k) < f(x_k)$ we calculate t_{rep} by replacing x_k with $x_k + d_k$ in (5.14). We then calculate ϕ_{rep}, ψ_{rep}, τ_{rep} and σ_{rep} by replacing t with t_{rep} in (3.36)-(3.39). If $\sigma_{\text{rep}} \geq 0.5\tau_{\text{rep}}^2$ then t, ϕ, ψ, τ and σ are replaced by t_{rep}, ϕ_{rep}, ψ_{rep}, τ_{rep} and σ_{rep} respectively. During alternative iterations t is set to the integer which satisfies the following equation

$$\|y_t - x_k\| = \max \{ \|y_i - x_k\| : i \in K \}. \tag{5.15}$$

Clearly (5.15) helps to cluster the interpolation points around x_k.

We now describe the decision process used to choose between trust region and alternative iterations during the execution of HEMBOQA. We also mention part of the trust region management procedure for the inner trust

region, whose size is controlled by $\rho_k \in \mathbb{R}^n$. Specifically we describe which situations require $\rho_{k+1} < \rho_k$ and which require $\rho_{k+1} = \rho_k$[4]. Now when deciding between trust region and alternative iterations measures of both the clustering of the interpolation points and the quality of the quadratic model are required. Towards this end r_k is defined as

$$r_k = \frac{f(x_k) - f(x_k + d_k)}{Q_k(x_k) - Q_k(x_k + d_k)}, \tag{5.16}$$

δ_k is defined as follows

$$\delta_k = \max \{\|y_i - x_k\| : i = 1, 2, \ldots, m\}, \tag{5.17}$$

and let y_δ denote the interpolation point which satisfies this equation. We define $\omega_k \in \mathbb{R}^n$ as follows. If $\gamma_{\text{in}} < \gamma_{\text{req}}$ then $(\omega_k)_i = \infty$ otherwise if $\gamma_{\text{in}} \geq \gamma_{\text{req}}$ then consider the γ_{in} points on $\mathcal{X}_s$ that are closest to x_k, call this set $\mathcal{Y}_s$. The ith element of ω_k is then set equal to the ith element of the point in $\mathcal{Y}_s$ with the greatest difference between its ith element and the ith element of x_k. In addition we introduce the following notation to aid in our description of the decision process. If a, $b \in \mathbb{R}^n$ are two vectors and $h(a, b)$ is any equation or inequality containing a and b then any$(h(a, b))$ is true if and only if $h(a_i, b_i)$ holds for any i. We also define all$(h(a, b))$ such that all$(h(a, b))$ is true if and only if $h(a_i, b_i)$ holds for all i.

Now it is undesirable to take steps that are small relative to the size of the trust region. Accordingly, once a trust region step is generated by solving problem (5.12) the following expression is evaluated

$$(\text{all}(d_k \leq 0.5\rho_k) \vee (\gamma_{\text{in}} \geq \gamma_{\text{req}})) \wedge (\text{all}(d_k \leq 0.1\rho_k) \vee (\gamma_{\text{in}} < \gamma_{\text{req}})). \tag{5.18}$$

If this expression is true d_k is discarded and either an alternative iteration is started or ρ_k is reduced and another trust region iteration is begun; the choice is made using (5.19), as described below. Otherwise the trust region iteration proceeds as normal, in which case the value of $f(x_k + d_k)$ will be calculated. Following [126] we refer to the situation where the trust region step is not discarded by saying that $f(x_k + d_k)$ is calculated. If the trust region step is rejected the clustering of the interpolation points is checked using the following expression

$$\text{all}(|\omega_k - x_k| \leq 2\rho_k) \vee (\gamma_{\text{in}} \geq \gamma_{\text{req}}). \tag{5.19}$$

If this expression is true ρ_{k+1} is chosen such that $\rho_{k+1} < \rho_k$ using a method discussed in the next subsection. If the new ρ_{k+1} satisfies all$(\rho_{k+1} \leq \rho_{\text{end}})$

[4]These relations are applied componentwise since ρ_k is a vector.

then HEMBOQA is stopped and x_k and $f(x_k)$ are returned. If any$(\rho_{k+1} > \rho_{\text{end}})$ a new trust region step is calculated and the decision process is repeated. On the other hand, if (5.19) is false then $\rho_{k+1} = \rho_k$ and an alternative iteration is started. After the alternative iteration a trust region step is calculated with $\rho_{k+2} = \rho_{k+1} = \rho_k$. Now consider the case where $f(x_k + d_k)$ is calculated during a trust region iteration. If $f(x_k + d_k) < f(x_k)$ or $r_k \geq 0.1$ then it is assumed that the algorithm is proceeding satisfactorily using the current trust region and set of interpolation points so another trust region step is calculated with $\rho_{k+1} = \rho_k$. If neither of these conditions hold the next iteration is an alternative iteration with $\rho_{k+1} = \rho_k$ if any interpolation points satisfy $|(y_i)_j - (x_k)_j| > 2$, $j = n_c+1, \ldots, n$ or if all$(|y_\delta - x_k| > \max(2\Delta_k, 10\rho_k)$. If neither of these conditions hold the inner trust region radius may need to be reduced for the algorithm to make further progress. This decision is made using the following conditions

$$\Delta_{k+1} = \rho_k, \tag{5.20}$$
$$\text{all}(d_k \leq \rho_k), \tag{5.21}$$
$$\text{all}(|\omega_k - x_k| \leq 2\rho_k), \tag{5.22}$$
$$f(x_k + d_k) \geq f(x_k), \tag{5.23}$$
$$r_k \leq 0. \tag{5.24}$$

If these conditions hold ρ_{k+1} is chosen such that $\rho_{k+1} < \rho_k$. If the new ρ_{k+1} satisfies all$(\rho_{k+1} \leq \rho_{\text{end}})$ then HEMBOQA is stopped and x_k and $f(x_k)$ are returned. If any$(\rho_{k+1} > \rho_{\text{end}})$ a new trust region step is calculated and the decision process is repeated. If any of the conditions given by (5.20)–(5.24) are not satisfied a new trust region step is calculated with $\rho_{k+1} = \rho_k$. This completes our discussion of the process used to decide between trust region and alternative iterations.

Trust region management

We now discuss the procedures used to manage the sizes of the inner and outer trust regions. Now, the situations in which $\rho_{k+1} = \rho_k$ along with those in which $\rho_{k+1} < \rho_k$ is required were discussed earlier in this section[5]. The arguments will not be repeated here. However, a formula to calculate ρ_{k+1} when $\rho_{k+1} < \rho_k$ has not been given. The required reduction formula can be

[5]As in BOBYQA we do not allow $\rho_{k+1} > \rho_k$.

expressed as follows

$$(\rho_{k+1})_i = \begin{cases} (\rho_{\text{end}})_i, & (\rho_k)_i \leq 16(\rho_{\text{end}})_i, \\ \sqrt{(\rho_k)_i(\rho_{\text{end}})_i}, & 16(\rho_{\text{end}})_i < (\rho_k)_i \leq 250(\rho_{\text{end}})_i, \\ 0.1(\rho_k)_i, & (\rho_k)_i > 250(\rho_{\text{end}})_i. \end{cases} \tag{5.25}$$

This is the same formula used in BOBYQA besides the fact that it is now applied to each element of the vector describing the box shaped trust region rather than to the single trust region radius of the spherical trust region. This completes the discussion of the management of the inner trust region.

If the kth iteration is an alternative iteration we set $\Delta_{k+1} = \Delta_k$. If the kth iteration is a trust region iteration that calculates $f(x_k + d_k)$ then Δ_{k+1} is given by

$$(\Delta_{k+1})_i = \begin{cases} \min\left[0.5(\Delta_k)_i, (d_k)_i\right], & r_k \leq 0.1, \\ \max\left[0.5(\Delta_k)_i, (d_k)_i\right], & 0.1 < r_k \leq 0.7, \\ \max\left[0.5(\Delta_k)_i, 2(d_k)_i\right], & r_k > 0.7, \end{cases}$$

which is a generalisation of the formula used in BOBYQA. After this formula has been applied we set $(\Delta_{k+1})_i = (\rho_k)_i$ if $(\Delta_{k+1})_i \leq 1.5(\rho_k)_i$. In all but two other situations, which are discussed below, we set $\Delta_{k+1} = \Delta_k$. The first situation in which $\Delta_{k+1} \neq \Delta_k$ occurs when we have $\rho_{k+1} < \rho_k$. In this case we set the elements of Δ_{k+1} using the following formula

$$(\Delta_{k+1})_i = \max[0.5(\rho_k)_i, (\rho_{k+1})_i], \tag{5.26}$$

which is a generalisation of the formula used in BOBYQA. The second case in which $\Delta_{k+1} \neq \Delta_k$ occurs when a trust region step is generated such that (5.18) is true. In this case Δ_{k+1} is given by $\Delta_{k+1} = 0.1\Delta_k$. After applying this reduction we set $(\Delta_{k+1})_i = (\rho_k)_i$ if $(\Delta_{k+1})_i \leq 1.5(\rho_k)_i$. In addition to these reductions we also temporarily reduce Δ_k at the start of an alternative iteration using the following formula

$$(\Delta_k)_i := \begin{cases} \max[0.1|(y_\delta - x_k)_i|, (\rho_k)_i], & |(y_\delta - x_k)_i| < 10(\Delta_k)_i, \\ (\Delta_k)_i, & \text{otherwise}, \end{cases} \tag{5.27}$$

which is a modification of the formula used in BOBYQA. The original value of Δ_k is restored after the alternative iteration. If, in any situation, we have $(\Delta_k)_i < 1$ for any $i \in \{n_c + 1, n\}$ then we set $(\Delta_k)_i = 1$. The reason for preventing the size of the trust region in the discrete coordinates from falling below one was given in section 5.2.2. This completes our description of the behaviour of Δ_k.

5.2.4 SEMBOQA and COMBOQA

No convergence results can be proven for HEMBOQA. To overcome this deficiency two new implementations of our derivative free algorithm are developed in this section. Before proceeding, we define manifold minima as continuous local minima of f on the *feasible continuous manifolds* of problem (1.1). For example, the solutions of the problems in (5.5) are manifold minima. In this section we assume that f has a finite number of manifold minima. We also define a slight modification of HEMBOQA which we denote HEMBOQA$_2$. HEMBOQA$_2$ requires as input a set of feasible interpolation points, an interpolating quadratic model, the W matrix used to update the quadratic model and a step length. Obviously, with this input, HEMBOQA$_2$ does not make use of the initialisation procedure described in 5.2.1. The rest of HEMBOQA$_2$ is the same as HEMBOQA. HEMBOQA$_2$ can be thought of as HEMBOQA running from $k = 2$ with the input to HEMBOQA$_2$ replacing the values obtained during the first iteration of HEMBOQA and the step length input referred to above replacing the step length that would be returned by the trust region or alternative procedures. The two implementations developed here use a number of calls to HEMBOQA and HEMBOQA$_2$. We still use *maxtime* and *maxevals* for the maximum time and number of function evaluations allowed in HEMBOQA and HEMBOQA$_2$. In this section we assume that *maxevals* is finite. We denote the maximum time and number of function evaluations in the new implementations by $t_{\max}$ and $n_{\max}$ respectively. Both of the implementations described in this section have the same basic form, which is described in Algorithm 5. The differences between the two implementations lie in how steps iii and v are performed. More details on the steps described in Algorithm 5 are given in the following paragraphs.

The first implementation, SEMBOQA, can be proven to converge to a separate local minima. In the following description we use the term major iteration to distinguish the iterations of SEMBOQA from the iterations of HEMBOQA and HEMBOQA$_2$. One major iteration of SEMBOQA is completed each time step iii is performed in Algorithm 5. As described in Algorithm 5, SEMBOQA uses major iterations with the form of HEMBOQA with a check at the end of each major iteration to determine whether x_k is a local minimum. If it is not a local minimum a new point with a lower objective function value than x_k is found and added to the interpolation points. We now give specific details on how some of the steps in Algorithm 5 are performed in SEMBOQA. The following procedure is used to perform the check for local minima in step iii. Whenever HEMBOQA or HEMBOQA$_2$ return x_k we use the neighbourhood solver to find a locally optimal point x_b

Algorithm 5 General structure of the deterministic implementations of the derivative free algorithm

i. Choose $t_{\max}$, $n_{\max}$ and the input required by HEMBOQA.

ii. Run HEMBOQA to obtain a new x_k and $f(x_k)$. Also store W_k, Q_k and the final set of interpolation points.

iii. If x_k is a local minimum go to step ix otherwise go to step iv.

iv. If # function evaluations $> n_{\max}$ or time $> t_{\max}$ go to step ix otherwise go to step v.

v. Generate a new feasible point x_n satisfying $f(x_n) < f(x_k)$ (see pg 117 for details).

vi. Run one iteration of HEMBOQA$_2$ with W_k, Q_k and the final set of interpolation points stored in step ii or step viii as inputs. The step length input into HEMBOQA$_2$ is given by $x_k - x_n$. If RESCUE is called during the first iteration of HEMBOQA$_2$ go to step vii otherwise go to step viii.

vii. Set $x_0 = x_n$ and go to step ii which restarts HEMBOQA with the new x_0.

viii. Run HEMBOQA$_2$ to obtain a new x_k and $f(x_k)$ with W_k, Q_k and the final set of interpolation points stored in step ii or the previous call to step viii as inputs. The step length input into HEMBOQA$_2$ is given by $x_k - x_n$. Store the new W_k, Q_k and final set of interpolation points. Go to step iii.

ix. Stop and return x_k and $f(x_k)$.

on $\mathcal{X}_s$ starting from x_k. We then set $x_k = x_b$, this is the x_k used in step iii. The objective function is then evaluated at every point in $\mathcal{N}_r(x_k)\backslash\{x_k\}\cap\Omega_m$. Denote the point in $\mathcal{N}_r(x_k)\cap\Omega_m$ with the smallest objective function value by x_n. If there is more than one point with the lowest objective function value choose the point that is closest to x_k. If $x_n = x_k$ then x_k is a local minimum and we stop and return x_k and $f(x_k)$. If $x_n \neq x_k$ we have not found a local minimum and $f(x_n) < f(x_k)$. The fact that $f(x_n) < f(x_k)$ when the test for optimality fails allows us to use x_n as the point that must be generated in step v. In step vi we see that HEMBOQA$_2$ is not used if RESCUE would be called on its first iteration. The call to RESCUE might result in the point x_n being discarded from the set of interpolation points. Since $f(x_n) < f(x_k)$, this would be discarding the best interpolation point, which we do not allow in our derivative free algorithm. The following theorem provides the proof of convergence for SEMBOQA.

Theorem 14. *SEMBOQA converges to a separate local minimum after a finite number of function evaluations.*

Proof. We assume that the neighbourhood solver converges to a manifold minimum after a finite number of evaluations. If this convergence has any conditions, such as requiring that f be continuous, then these conditions also apply to this theorem.

At the end of each major iteration either x_k is a local minimum or a point x_n such that $f(x_n) < f(x_k)$ has been found. Suppose we find x_n satisfying this inequality. Now we run the neighbourhood solver on $\mathcal{X}_s$ while checking for a local minimum and set x_k to the returned optimal point. Therefore, $f(x_n)$ is less than or equal to the value of a manifold minimum on $\mathcal{X}_s$. The point x_n is added to the set of interpolation points. Now, SEMBOQA is structured such that it will never discard the point with the current best objective function value. Therefore, the process of checking for a local minimum at the end of each major iteration either removes at least one manifold minimum from consideration or finds a separate minimum. From (5.1) we see that a separate local minimum must also be a manifold minimum. We also note that every problem must have at least one separate local minimum. This is due to the fact that the global minimum is clearly also a separate minimum. Considering these facts we see that, after some finite number of major iterations, either a separate local minimum will have been found or only one manifold minimum will still be under consideration. In the latter situation this manifold minimum must be a separate minimum since every problem must have at least one separate minimum. Therefore, SEMBOQA converges to a separate local minimum after a finite number of major iterations. Now, by assumption we have $maxevals < \infty$ so each

major iteration requires a finite number of function evaluations. Therefore, SEMBOQA converges to a separate local minimum after a finite number of function evaluations. □

The second implementation, COMBOQA, can be proven to converge to a combined local minima. COMBOQA also follows the structure outlined in Algorithm 5. The following procedure is used to perform the check for local minima in step iii. Whenever HEMBOQA or HEMBOQA$_2$ return x_k the neighbourhood solver is used to find a locally optimal point x_b on $\mathcal{X}_s$ starting from x_k. We then set $x_k = x_b$, this is the x_k used in step iii. $\mathcal{N}_{\text{comb}}(x_k)$ is then constructed using the neighbourhood solver. Denote the point in $\mathcal{N}_{\text{comb}}(x_k) \cap \Omega_m$ with the smallest objective function value by x_n. If there is more than one point with the lowest objective function value choose the point that is closest to x_k. If $x_n = x_k$ then x_k is a local minimum and we stop and return x_k and $f(x_k)$. If $x_n \neq x_k$ we have not found a local minimum. COMBOQA also makes use of a modified version of step v which has the following form

v. Generate a new feasible point x_n satisfying $f(x_n) < f(x_k)$. If $\|x_k - x_n\| > 10\|\rho_{\text{beg}}\|$ go to step vii.

This modifications is made since it is more efficient to generate a new set of interpolation points if the new best point x_n is far from the current set of interpolation points. The following theorem provides the proof of convergence for COMBOQA.

Theorem 15. *COMBOQA converges to a combined local minimum after a finite number of function evaluations.*

Proof. The proof of this theorem takes the same form as the proof of the convergence of SEMBOQA so we just give a brief outline here. As with SEMBOQA, at the end of each major iteration of COMBOQA either x_k is a local minimum or a point x_n such that $f(x_n) < f(x_k)$ has been found. Therefore each major iteration of COMBOQA either finds a local minimum or determines that at least one manifold minimum is not a local minimum. There are a finite number of manifold minima and a combined local minimum is also a manifold minimum. Therefore COMBOQA converges to a combined local minimum after a finite number of function evaluations. □

This concludes the development of the three implementations of our derivative free algorithm.

5.3 Conclusion

In this chapter we have developed a trust region, derivative free algorithm for solving problem (1.1). Three different implementations of the algorithm were developed. Two of the implementations are deterministic while the third is a heuristic. We note that convergence of the deterministic algorithms is only guaranteed for problems with a finite number of manifold minima. Computational results examining the effectiveness of these implementations are presented in chapter 6. We have also discussed possible definitions of local minima for mixed integer programs. There are a number of possible definitions and we propose four restrictions that can be used choose an effective definition. The definitions proposed in the literature do not satisfy these restrictions and an example has been presented to illustrate the resulting deficiencies. In addition, a new definition of a local minimum has been proposed. This definition does satisfy our restrictions and its effectiveness is illustrated using the same example.

Chapter 6

Computational Results and Discussion

In this chapter we present computational results which illustrate the effectiveness of the methods developed in chapters 4 and 5. The methods developed in chapter 4 for solving MIQPs are tested in section 6.1. Randomly generated problems with three different types of constraints are used as test problems. The problems are generated in such a way that we can specify that the Hessian should be either invertible, singular or have a positive definite n_cth principal leading submatrix. When the Hessian is invertible we can also specify the percentage of negative eigenvalues. This allows us to test all three of the methods developed in chapter 4. The methods developed in chapter 5 are tested in section 6.2. Thirty different test functions were used to generate test problems. Some of these functions are only defined for fixed n while others are defined for $n > 0$. These test functions were used to generate a total of 118 test problems for the derivative free algorithms in chapter 5. The test problems were also solved using the state of the art derivative free MINLP solver NOMAD. Some concluding remarks are made in section 6.3.

6.1 Mixed integer quadratic programming

In this section we examine the effectiveness of the methods for solving MIQPs developed in chapter 4. We recall that methods were developed for three different classes of Hessian. In section 6.1.1 we present the results obtained using the methods developed for invertible H_{cc}. The results obtained using the methods developed for positive definite H_{cc} are given in section 6.1.2. The results obtained when H_{cc} is singular are given in section 6.1.3. Test

problems whose Hessians have the required structure were generated using procedures given in appendix C. The tests in each section are made up of two parts; in the first part we examine general MIQPs while in the second we focus on the form of the MIQPs that have to be solved by our derivative free algorithms.

We consider only bound and inequality constraints in our test problems. Any problem with equality constraints can be transformed into this form. Three types of constraints are considered,

Type 1. Bound constraints: The only constraints present in the original problem were bound constraints.

Type 2. Sparse linear inequality constraints: The only constraints present in the original problem were linear inequality constraints and the matrix A had sparse block diagonal structure.

Type 3. Dense linear inequality constraints: The only constraints present in the original problem were linear inequality constraints and the matrix A was dense.

The linear inequality constraints were generated in such a way that the original problem was always feasible and bounded. The procedures used to generate the constraints are given in appendix C.

Unless noted otherwise all tests in this section were performed on a PC with an Intel Core i5 CPU at 3.2GHz with 4GB of RAM running 64-bit Windows 7. Whenever more than one algorithm was used to solve the same problem the algorithms were run consecutively with no time gap between the end of an algorithm and the start of the next. This was done to ensure a similar computational load. All solutions were checked for feasibility. If an algorithm ran for more than 10000 seconds on a problem it was stopped and declared unsuccessful for that problem. We also note that time is used as the performance measure when making the comparisons in this section.

6.1.1 Results obtained with H_{cc} invertible

In this section we consider the case where H_{cc} is invertible. In all of the sections in this chapter comparisons between algorithms are made using performance profiles, which were developed in Dolan and Moré [54]. Let $t_{p,s}$ be the performance measure for algorithm s to solve problem p. As noted above we use time as the performance measure in this section so $t_{p,s}$ is the time taken by algorithm s to solve problem p. The performance ratio $\rho_{p,s}$ is then

given by

$$\rho_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}} \tag{6.1}$$

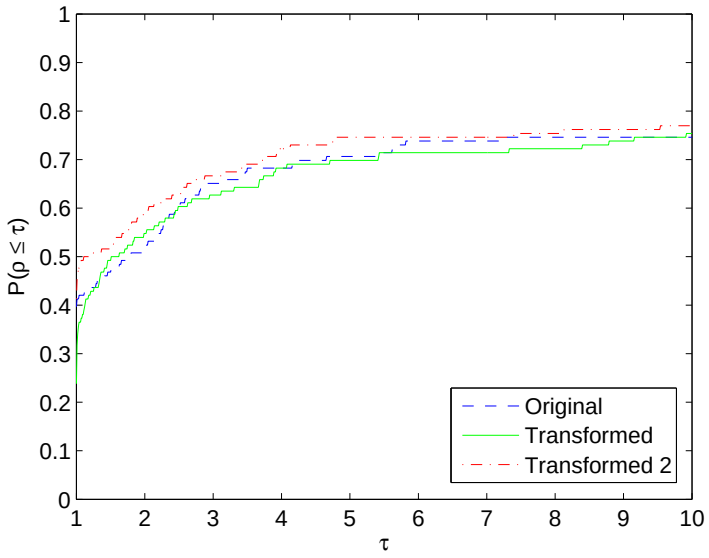
where $\mathcal{S}$ is the set of algorithms. Denote the fraction of performance ratios that are less than a factor $\tau \geq 1$ by $P(\rho_{p,s} \leq \tau : s \in \mathcal{S})$. The performance profile is a plot, for each algorithm, of $P(\rho_{p,s} \leq \tau : s \in \mathcal{S})$ vs τ. The same performance profile is often plotted multiple times with different ranges of τ to better display the behaviour of $P(\rho_{p,s} \leq \tau : s \in \mathcal{S})$.

We now present results examining the effectiveness of the transformation by solving problem (1.2) and problem (4.27) using SCIP 3.0, Baron 11.3 and Algorithm 2. Algorithm 2 and SCIP were both called from MATLAB 2010a; in the case of SCIP this was done using the mex interface provided in OPTI Toolbox 1.7 [48]. BARON was run on the NEOS server [49, 66] since no academic license is available for BARON. SCIP and BARON where both run with their default settings; the default absolute and relative convergence tolerances of SCIP are both 10^{-6}, the default absolute and relative convergence tolerances of BARON are 10^{-9} and 0.1 respectively. The absolute convergence tolerance *tol* in Algorithm 2 was set to 0.01. The fact that the convergence tolerances are different is irrelevant since we do not directly compare different algorithms in this section. We recall that the transformed problem has at most $\frac{1}{2}(n_c^2 - n_c) + n_c n_d$ fewer bilinear terms than the original problem. Clearly as n_c increases the reduction in the number of bilinear terms increases so we expect the transformation to become more effective as n_c increases. We therefore consider three different sets of test problems for each algorithm; the first containing problems with $n_c = n_d$, the second containing problems with $n_c > n_d$ and the third with $n_c < n_d$. We shall not consider problems with $n_c = 0$ or $n_d = 0$ since these are no longer mixed problems and specialised solution methods have been developed to solve these problems.

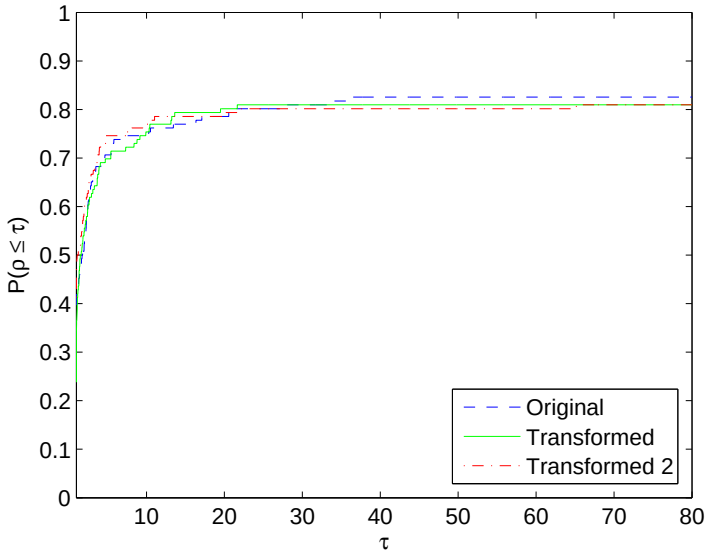
We first present the results obtained using SCIP. When $n_c = n_d$ the set of test problems was constructed using various values of n for each type of constraint; for type 1, 2 and 3 constraints we used $n = 4, 8, 16, 24, 32$, $n = 4, 8, 12, 16, 20$ and $n = 4, 6, 8, 10, 12$ respectively. Different values of n were used for different constraint types in an attempt to obtain problems with a similar level of difficulty for each type of constraint; for example problems with type 1 constraints are much easier to solve than problems with type 3 constraints so a larger range of n values was used for these constraints. The problem generation method used in this section allows the percentage of negative eigenvalues q of the Hessian to be specified. For each value of n problems with $q = 20, 40, 60, 80$ were generated. For each pair of values of n and q three test problems were randomly generated creating a test set of 180

problems. The results are presented in the form of a performance profile in Figure 6.1. In this figure, and the rest of the figures in this section, Original shows the results when solving problem (1.2), Transformed shows the results when solving the transformed problem with U_{dd} set using problem (4.20) and Transformed 2 shows the results when solving the transformed problem with $U_{dd} = I_{n_d}$. For reasons which are explained below, the time taken to solve problem (4.20) is not included in the results. The linear programs in the objective function of (4.20) were solved using the linear programming algorithm in CPLEX 12.1. When $n_c > n_d$ the set of test problems was constructed as follows. Similarly to $n_c = n_d$ for type 1, 2 and 3 constraints we used $n = 8, 16, 24, 32$, $n = 8, 12, 16, 20$ and $n = 8, 10, 12, 14$ respectively. For each value of n four values of n_c were used. The values were chosen to evenly divide the interval $[n/2 + 1, n - 1]$; for example for $n = 24$ we have $n_c = 13, 16, 20, 23$. For $n = 8$, $n_c = 4$ was also to construct the test set, i.e. $n_c = 4, 5, 6, 7$. For each pair of values of n and n_c problems with $q = 20, 40, 60, 80$ were generated. For each set of values of n, n_c and q two test problems were randomly generated creating a test set of 384 problems. The results are presented in the form of a performance profile in Figure 6.2. When $n_c < n_d$ the set of test problems was constructed as follows. Problems were generated using type 1, 2 and 3 constraints with $n = 8, 11, 14, 17$, $n = 8, 10, 12, 14$ and $n = 8, 9, 10, 11$ respectively. As before, four values of n_c were chosen for each n to evenly divide the interval $[1, n/2 - 1]$ with $n_c = 4$ also being used for $n = 8$. Smaller values of n were used than when testing $n_c > n_d$ as the difficulty of the problems increases as n_d increases. As before problems were generated with $q = 20, 40, 60, 80$ and for each set of values of n, n_c and q two test problems were randomly generated creating a test set of 384 problems. The results are presented in the form of a performance profile in Figure 6.3.

We see from Figure 6.1 that when $n_c = n_d$ the transformation generally allows the problems to be solved more quickly but solving the original problem allows more problems to be solved. An unexpected result is that it is in fact slightly more efficient to set $U_{dd} = I_{n_d}$ than it is to use problem (4.20). This result is obtained in all of the tests performed in this section. Since the solution time of problem (4.20) is not included in the results we know that this is due to the fact that $U_{dd} = I_{n_d}$ is a good choice of U_{dd} rather than the fact that problem (4.20) is difficult to solve. Including the solution time of problem (4.20) would not affect our conclusion that $U_{dd} = I_{n_d}$ is the superior choice. Observing the similarity of the results obtained for the two choices of U_{dd} it might seem that the choice of U_{dd} will not have much effect on the solution time of problem (4.27). To show that the choice of U_{dd} can have a large effect on efficiency we again consider the test set constructed when

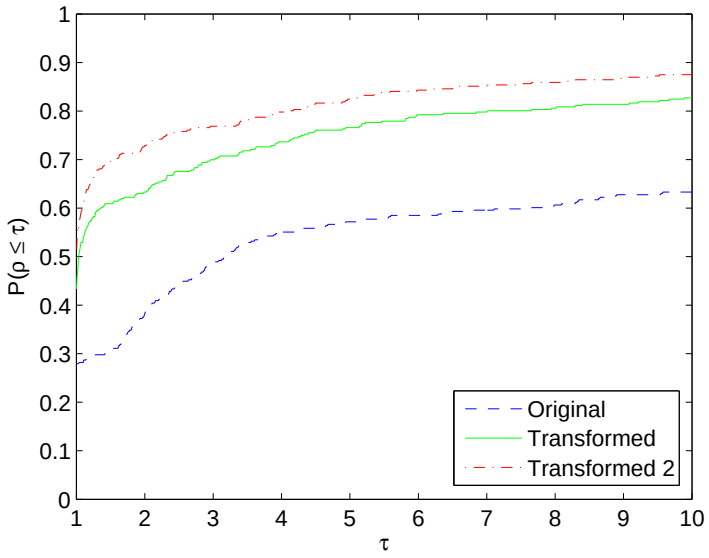


(a) Performance profile for $\tau \in [1, 10]$

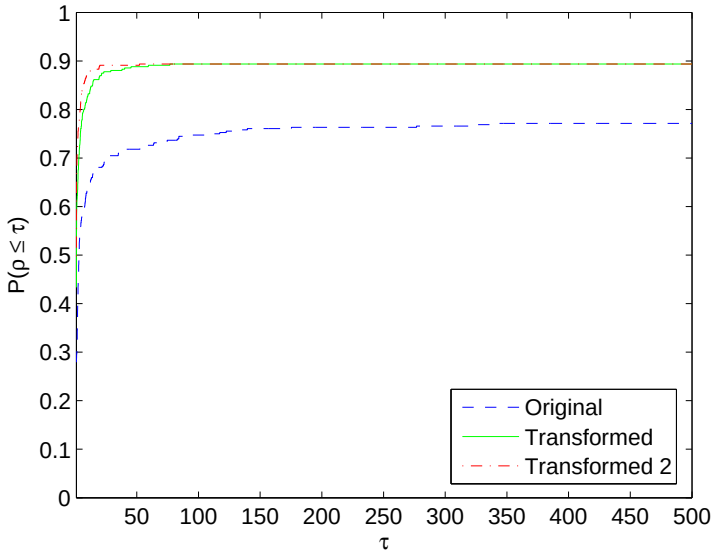


(b) Performance profile for $\tau \in [1, 80]$

Figure 6.1: Performance profile examining the effectiveness of the transformation when solving problems with $n_c = n_d$ using SCIP.

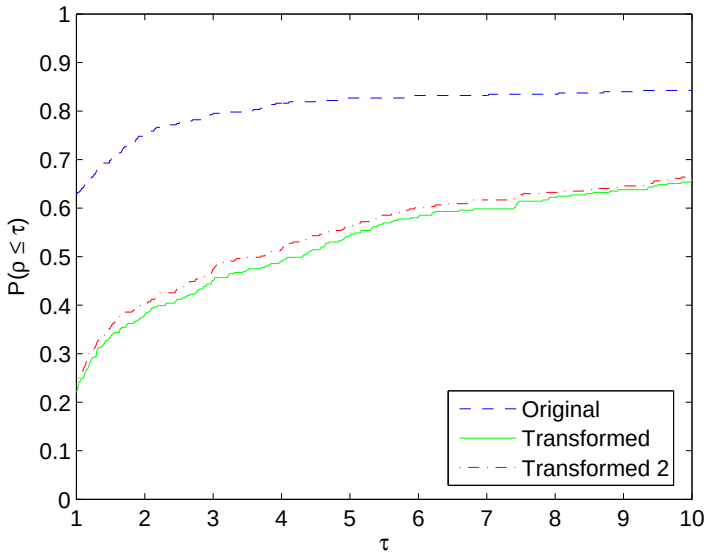


(a) Performance profile for $\tau \in [1, 10]$

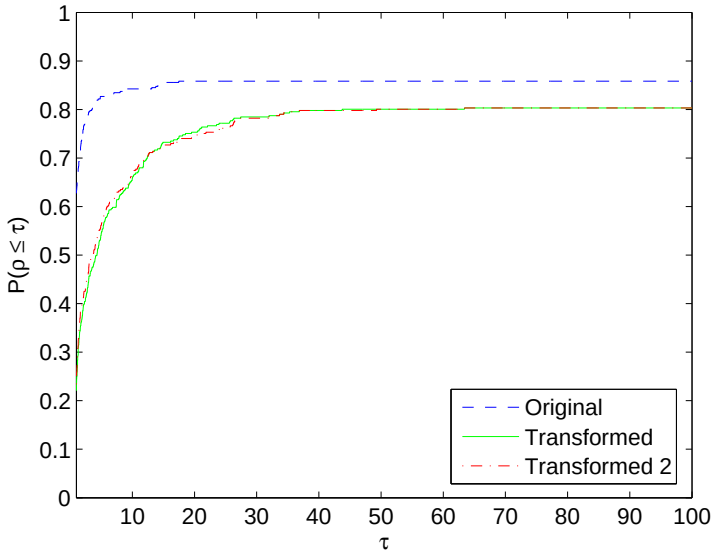


(b) Performance profile for $\tau \in [1, 500]$

Figure 6.2: Performance profile examining the effectiveness of the transformation when solving problems with $n_c > n_d$ using SCIP.



(a) Performance profile for $\tau \in [1, 10]$



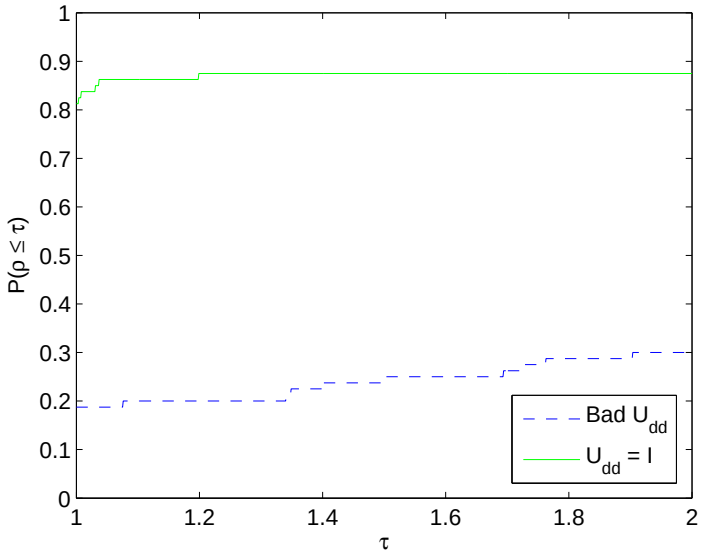
(b) Performance profile for $\tau \in [1, 100]$

Figure 6.3: Performance profile examining the effectiveness of the transformation when solving problems with $n_c < n_d$ using SCIP.

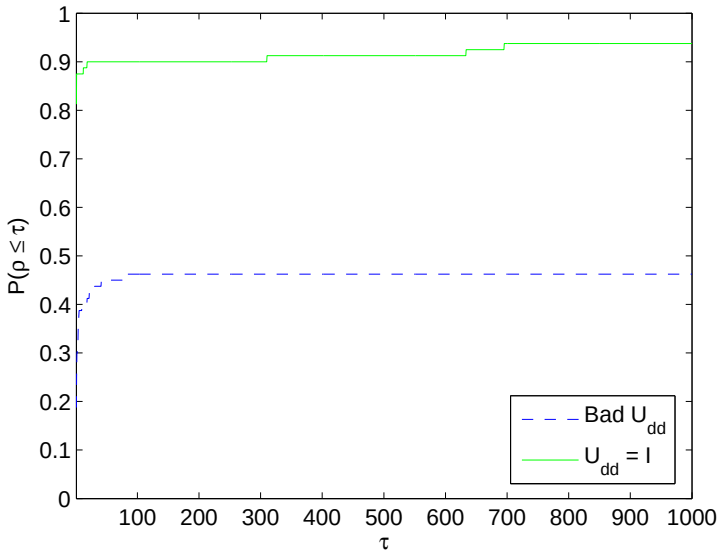
$n_c = n_d$. We remove the bound constrained problems from the set since we have an analytic solution of problem (4.20) for bound constrained problems. We use SCIP to solve two versions of problem (4.27) generated using two different U_{dd} matrices. In the first version we set $U_{dd} = I_{n_d}$ while in the other we set U_{dd} to be an upper triangular matrix with ones on the diagonal and all of the upper triangular elements set to 20. The results are presented in the form of a performance profile in Figure 6.4. It is clear that a poor choice of U_{dd} can have a drastic effect on the solution time; choosing U_{dd} carefully is important. Now, we see from Figures 6.2 and 6.3 that, as expected, the efficiency of our transformation increases as n_c increases. When $n_c > n_d$ it is clearly more efficient to solve problem (4.27) and when $n_c < n_d$ it is clearly more efficient to solve problem (1.2).

We now present the results obtained using BARON and Algorithm 2 which help to show that the results obtained using SCIP are characteristic of Branch and Bound algorithms in general rather than just being caused by the structure of SCIP. Smaller test sets were used to obtain the following results since they are just meant to confirm the results obtained using SCIP. We first present the results obtained using BARON. For $n_c = n_d$ the test set was constructed by generating problems with type 1, 2 and 3 constraints with $n = 8, 12, 16$, $n = 8, 10, 14$ and $n = 8, 10, 12$ respectively. For each value of n problems with $q = 20, 40, 60, 80$ were generated. For each pair of values of n and q two test problems were randomly generated creating a test set of 72 problems. The results for $n_c = n_d$ are presented in the form of a performance profile in Figure 6.5. For $n_c > n_d$ the same set of n and q values was used but n_c was given by $n_c = n - \lfloor n/4 \rfloor$. Clearly this test set is also made up of 72 problems. The results for $n_c > n_d$ are presented in Figure 6.6. For $n_c < n_d$ the same set of n and q values was used but n_c was given by $n_c = \lfloor n/4 \rfloor$. The results for $n_c < n_d$ are presented in Figure 6.7. Observing the relative behaviour of Transformed and Transformed 2 in the results obtained using SCIP and BARON we shall only consider $U_{dd} = I_{n_d}$ from this point on.

We now present the results obtained using Algorithm 2. For $n_c = n_d$ the test set was constructed by generating problems with type 1, 2 and 3 constraints with $n = 4, 8, 12, 16$, $n = 4, 8, 10, 14$ and $n = 4, 6, 8, 10$ respectively. For each value of n problems with $q = 20, 40, 60, 80$ were generated. For each pair of values of n and q three test problems were randomly generated creating a test set of 140 problems. The results for $n_c = n_d$ are presented in the form of a performance profile in Figure 6.8. For $n_c > n_d$ the test set was constructed by generating problems with type 1, 2 and 3 constraints with $n = 6, 12, 16$, $n = 6, 10, 12$ and $n = 6, 8, 10$ respectively. For each $n > 8$ four values of n_c were chosen for each n to evenly divide the interval $[1, n/2 - 1]$



(a) Performance profile for $\tau \in [1, 2]$



(b) Performance profile for $\tau \in [1, 1000]$

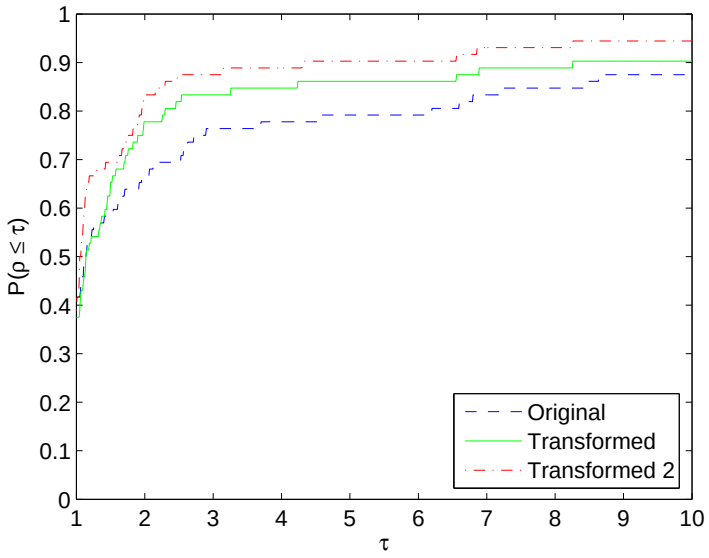
Figure 6.4: Performance profile examining the effect of the choice of U_{dd} when solving problems with $n_c = n_d$ using SCIP. Bad U_{dd} denotes the upper triangular choice of U_{dd}.

with $n_c = 4$ also being used for $n = 8$. For $n = 6$ we used $n_c = 4, 5$. Problems were generated with $q = 20, 40, 60, 80$ and for each set of values of n, n_c and q two test problems were randomly generated creating a test set of 240 problems. The results for $n_c > n_d$ are presented in the form of a performance profile in Figure 6.9. For $n_c < n_d$ the test set was constructed by generating problems with type 1, 2 and 3 constraints with $n = 4, 10, 14$, $n = 4, 8, 10$ and $n = 4, 6, 8$ respectively. For each $n > 8$ four values of n_c were chosen for each n to evenly divide the interval $[1, n/2 - 1]$ with $n_c = 4$ also being used for $n = 8$. For $n = 6$ we used $n_c = 1, 2$ and for $n = 4$ we used $n_c = 1$. Problems were generated with $q = 20, 40, 60, 80$ and for each set of values of n, n_c and q two test problems were randomly generated creating a test set of 192 problems. The results for $n_c < n_d$ are presented in the form of a performance profile in Figure 6.10.

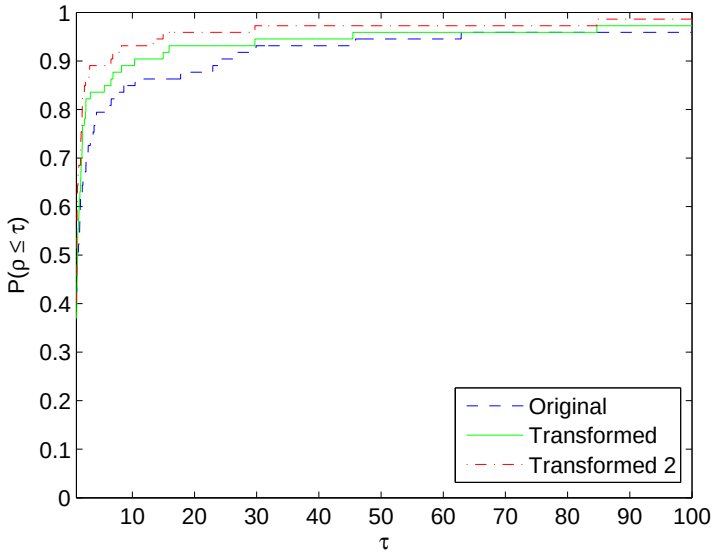
Considering Figures 6.5 to 6.10 we see that the behaviour observer with SCIP is replicated with BARON and Algorithm 2 when $n_c < n_d$ and $n_c > n_d$. However, for $n_c = n_d$ solving problem (4.27) is clearly now more efficient than solving problem (1.2). It should be noted that, especially when using BARON, the advantage, while noticeable, is not large. This differences in the behaviour of the solvers for $n_c = n_d$ is due to the different structure of the three solvers used.

In summary, the results presented thus far in this chapter allow the following conclusions to be drawn. Firstly, it is more efficient to use $U_{dd} = I_{n_d}$ rather than setting U_{dd} using problem (4.20). It is also clear that when solving MIQPs with more continuous variables than integer variables it is more efficient to solve the transformed problem than problem (1.2). When $n_c < n_d$ this result is reversed and it becomes more efficient to solve problem (1.2). When $n_c = n_d$ the results depend on the solver used; for two of the three solvers considered here it is more efficient to solve the transformed problem.

We now consider the specific case of solving the MIQP subproblems in HEMBOQA, SEMBOQA and COMBOQA. From this point on we shall refer to these algorithms as the MBOQA algorithms. We recall that these are bound constrained problems and note that they are solved by Algorithm 2 in this thesis. Since each algorithm requires a number of MIQPs to be solved we shall present the results in terms of average times rather than using performance profiles. Hopefully choosing the algorithm with the lowest average time will give the best performance for the MBOQA algorithms. The test set was constructed as follows. We consider problems with type 1 constraints and $n_c = n_d$. Problems with $n = 4, 6, 8, \ldots, 20$ were used. For each value of n problems with $q = 20, 40, 50, 60, 80$ were generated. For each pair of values of n and q two test problems were randomly generated creating a test set of 90 problems. We also note that for these tests the time limit

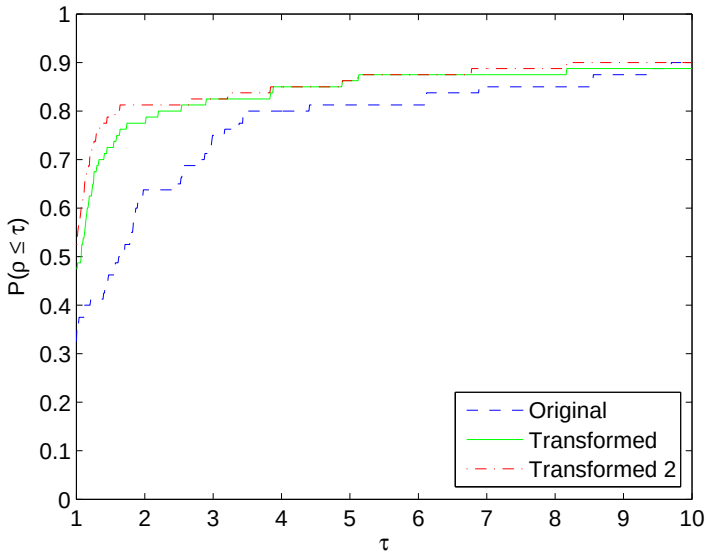


(a) Performance profile for $\tau \in [1, 10]$

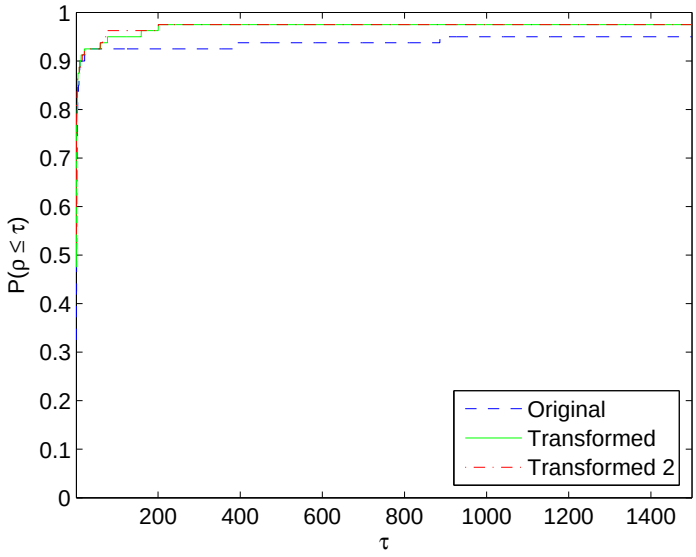


(b) Performance profile for $\tau \in [1, 100]$

Figure 6.5: Performance profile examining the effectiveness of the transformation when solving problems with $n_c = n_d$ using BARON.

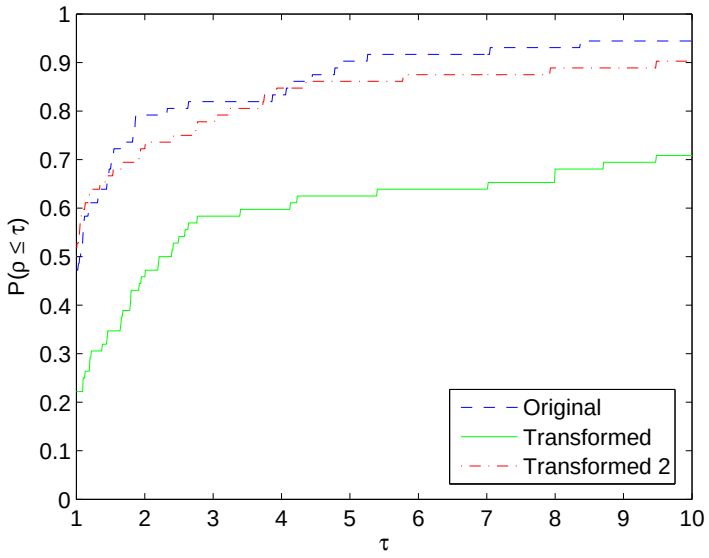


(a) Performance profile for $\tau \in [1, 10]$

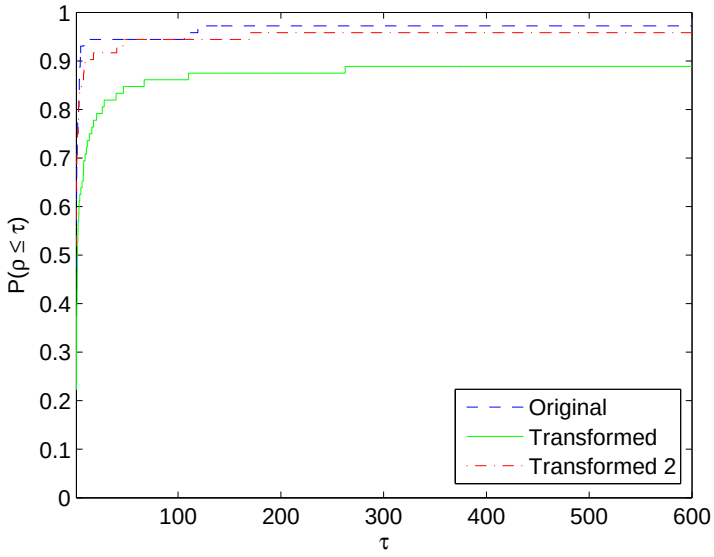


(b) Performance profile for $\tau \in [1, 1500]$

Figure 6.6: Performance profile examining the effectiveness of the transformation when solving problems with $n_c > n_d$ using BARON.

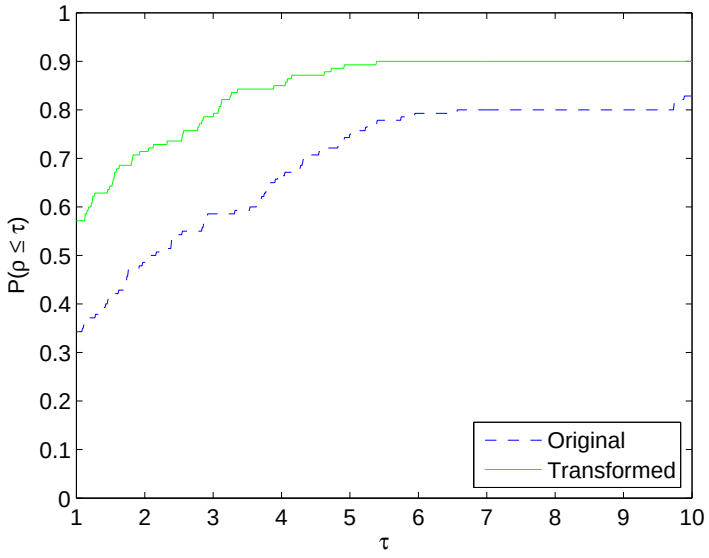


(a) Performance profile for $\tau \in [1, 10]$

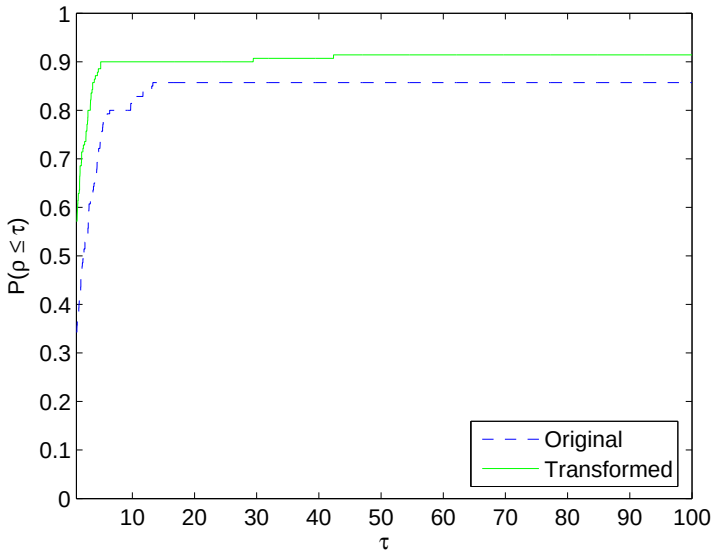


(b) Performance profile for $\tau \in [1, 600]$

Figure 6.7: Performance profile examining the effectiveness of the transformation when solving problems with $n_c < n_d$ using BARON.

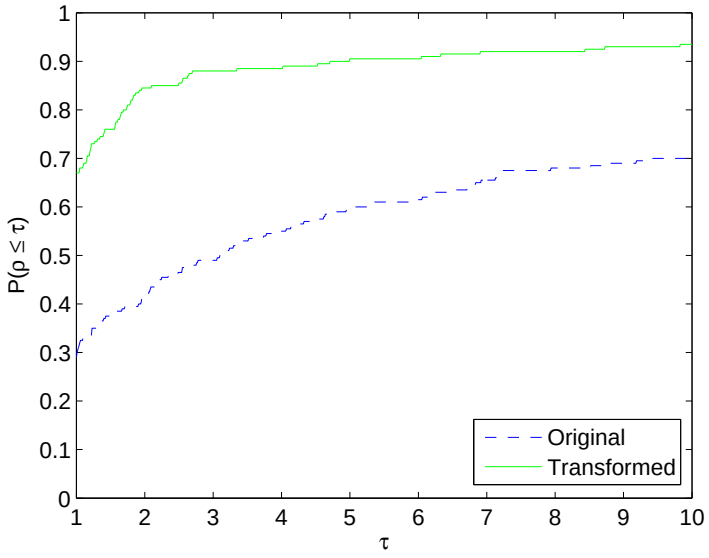


(a) Performance profile for $\tau \in [1, 10]$

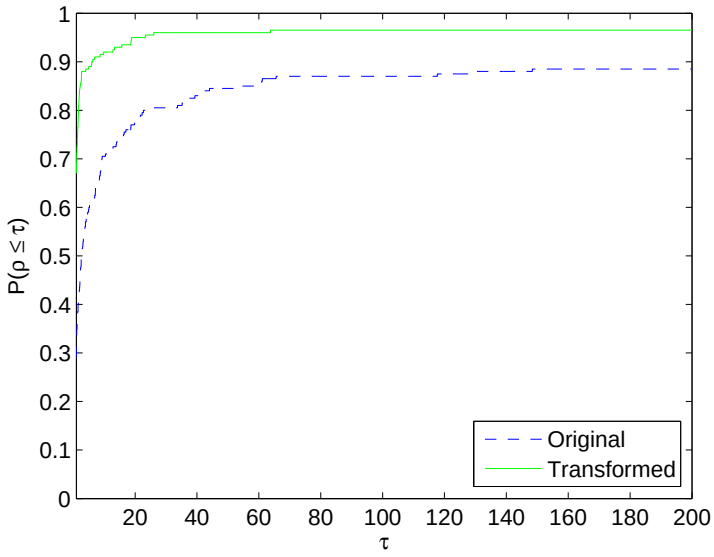


(b) Performance profile for $\tau \in [1, 100]$

Figure 6.8: Performance profile examining the effectiveness of the transformation when solving problems with $n_c = n_d$ and H_{cc} invertible using Algorithm 2.

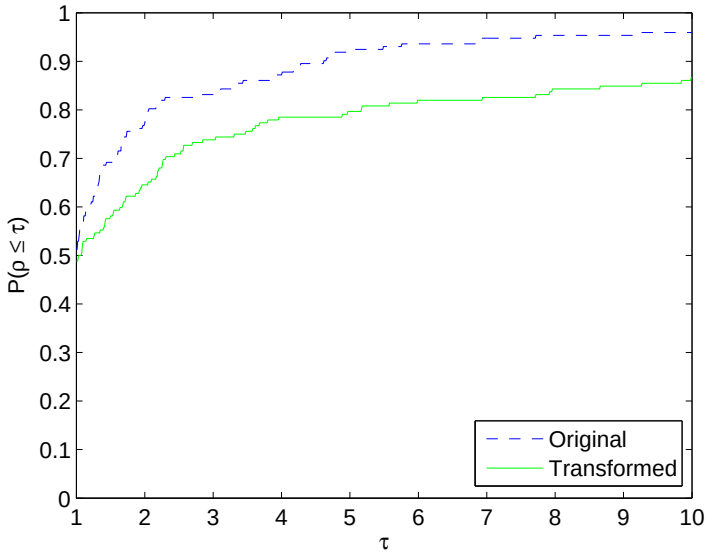


(a) Performance profile for $\tau \in [1, 10]$

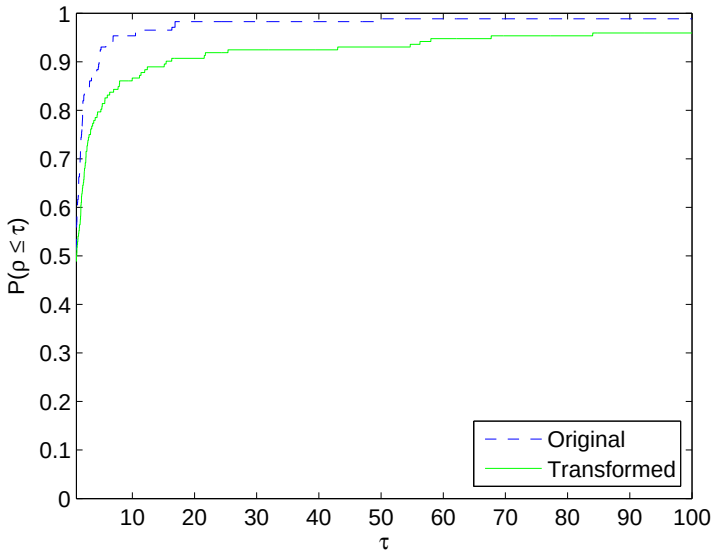


(b) Performance profile for $\tau \in [1, 200]$

Figure 6.9: Performance profile examining the effectiveness of the transformation when solving problems with $n_c > n_d$ and H_{cc} invertible using Algorithm 2.



(a) Performance profile for $\tau \in [1, 10]$



(b) Performance profile for $\tau \in [1, 100]$

Figure 6.10: Performance profile examining the effectiveness of the transformation when solving problems with $n_c < n_d$ and H_{cc} invertible using Algorithm 2.

n	Original problem			Transformed problem		
	time	# nodes	us	time	# nodes	us
4	0.1981	1.500	0	0.4850	13.40	0
6	0.4665	4.600	0	0.5650	10.90	0
8	4.0229	31.90	0	1.9911	34.10	0
10	89.419	314.9	0	11.755	152.9	0
12	781.11	1444	0	65.427	491.5	0
14	5228.8	4610	1	309.79	1474	0
16	11790	5539	4	2862.1	11693	0
18	18541	4983	8	8176.4	20182	1
20	20000	3711	10	14887	25549	4

Table 6.1: The results obtained when solving problem (1.2) and problem (4.27) with constraints of type 1, $n_c = n_d$ and H_{cc} invertible using Algorithm 2. The number of unsolved problems is denoted by us.

was extended to 20000 seconds. The results obtained using Algorithm 2 are presented in Table 6.1. The table gives the average time taken and number of nodes required to solve the original and transformed problem for each value of n. The values given are the average of the time and number of nodes for the ten test problems for each value of n.

Considering the results given in Table 6.1 we see that for $n \leq 6$ it is more efficient to solve problem (1.2) while for $n > 6$ it is more efficient to solve problem (4.27). This result is used when deciding whether or not to transform a MIQP subproblem in the MBOQA algorithms. This completes our discussion of the situation in which H_{cc} is invertible.

6.1.2 Results obtained with H_{cc} positive definite

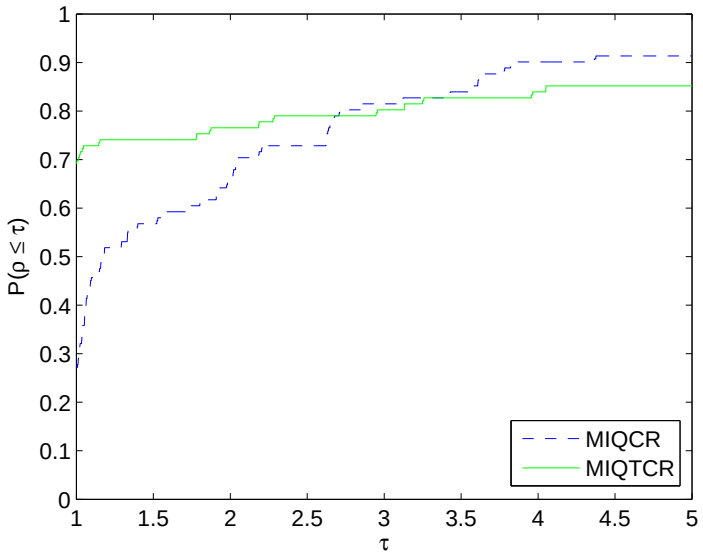
In this section we consider the case where H_{cc} is positive definite. The first set of results presented compares MIQCR and MIQTCR in the three cases $n_c = n_d$, $n_c > n_d$ and $n_c < n_d$. We also present results illustrating that for small problems it is more efficient to solve problem (1.2) directly rather than applying a convex reformulation. We then present the results used to determine which solver should be used in the MBOQA algorithms. Finally we present results comparing MIQCR, MIQTCR and MIQTBC when all three reformulated problems are solved using a MINLP solver. Since a MINLP solver must be used for MIQTBC the final set of results are not of practical

interest and are included for completeness.

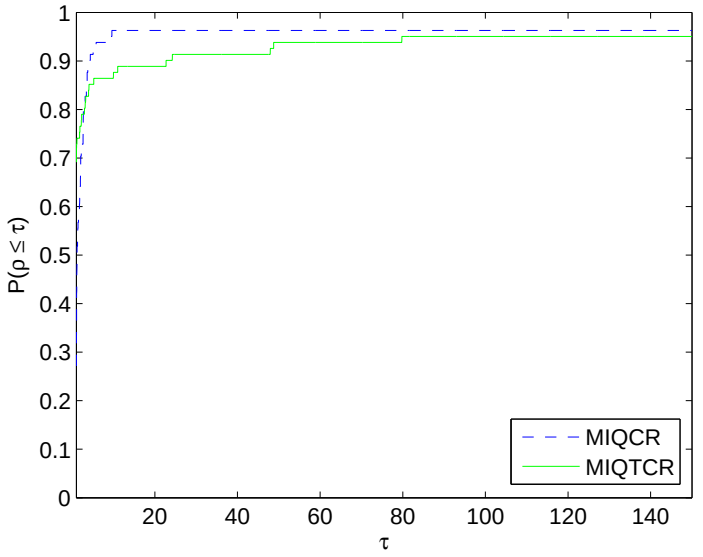
Results obtained using MIQP solvers

The first set of results presented in this section are used to compare MIQCR and MIQTCR. The MIQPs produced by the reformulation schemes were solved using CPLEX 12.1 and the SDPs in Theorems 2 and 3 were solved using SeDuMi 1.3 [143]. Both CPLEX and SeDuMi were called from MATLAB 2010a and were run with their default settings; the default convergence tolerance of SeDuMi is 10^{-9}, the default absolute and relative convergence tolerances of CPLEX are 10^{-6} and 10^{-4} respectively. When solving problems with constraints of types 2 and 3 the first inequality constraint is converted to an equality constraint using a slack variable. The allows α to be used in the reformulation. The first set of results presented compare MIQCR and MIQTCR when $n_c = n_d$. The set of test problems was constructed by generating problems with type 1, 2 and 3 constraints with $n = 4, 8, 16, 24, 32$, $n = 4, 8, 12, 16, 20$ and $n = 4, 6, 8, 10, 12$ respectively. For each value of n ten random problems were generated creating a test set of 150 problems. The results are presented in the form of a performance profile in Figure 6.11. The computation time includes the solution times of both CPLEX and SeDuMi. MIQCR and MIQTCR appear to be roughly equivalent for this set of test problems. We see from Figure 6.11 that while MIQTCR solves the greatest number of problems in the shortest time it takes much longer than MIQCR for a number of problems. When $n_c = n_d$ it appears that the reduction in the number of variables generated by MIQTCR is balanced by the tighter relaxation gap generated by MIQCR.

It should be clear from the discussion in section 4.3.2 that as n_c increases the difference in the number of variables required by MIQTCR and MIQCR should increase. This behaviour is illustrated in Table 6.2 which gives the average relaxation gap and number of variables in the reformulated problems when n_c is varied with $n = 8$ for problems with constraints of type 2. For each value of n_c five test problems were generated and the values given in the table are the averages of the results obtained for the five problems. We see that the percentage of the number of variables required by MIQTCR relative to MIQCR decreases from 82% to 27% as n_c increases from 1 to 7. For the same range of n_c the percentage of the relaxation gap in the problems generated by MIQCR relative to MIQTCR only varies between 68% and 43%. Similar behaviour is observed for all three types of constraints. Observing this behaviour it seems reasonable to compare MIQCR and MIQTCR for $n_c > n_d$. The set of test problems used to make this comparison was constructed as follows. For type 1, 2 and 3 constraints we used $n = 8, 16, 24, 32$, $n =$



(a) Performance profile for $\tau \in [1, 5]$



(b) Performance profile for $\tau \in [1, 150]$

Figure 6.11: Performance profile examining the effectiveness of the reformulations schemes MIQCR and MIQTCR when $n_c = n_d$.

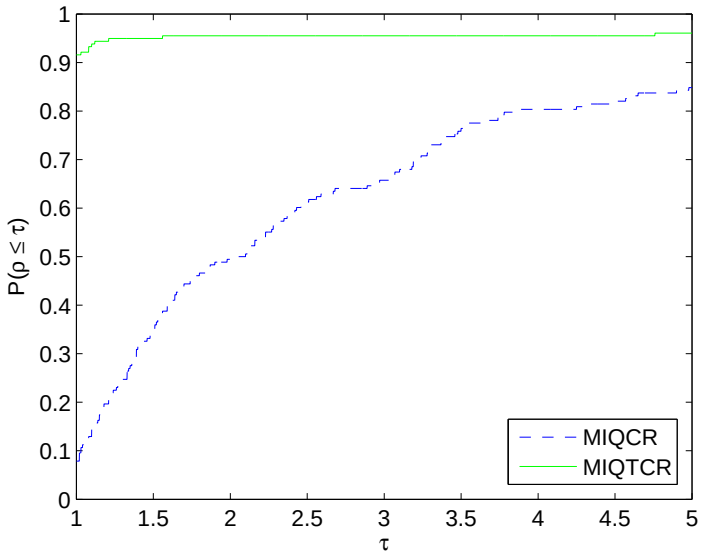
n_c	Relaxation gap			Number of variables		
	MIQCR	MIQTCR	Ratio	MIQCR	MIQTCR	Ratio
1	854.71	1399.4	0.611	414	340.4	0.822
2	21.987	46.252	0.475	297	196.2	0.661
3	257.09	588.40	0.437	310	175.6	0.566
4	26.621	59.940	0.444	221	103	0.466
5	88.929	191.43	0.465	182	69.2	0.380
6	27.858	41.272	0.675	153	46.6	0.305
7	18.973	27.846	0.681	66	18	0.273

Table 6.2: The average relaxation gap and number of variables for problems with constraints of type 2 when n_c is varied with $n = 8$. The ratio in the fourth column is the relaxation gap of MIQCR over that of MIQTCR. The ratio in the seventh column is the number of variables used by MIQTCR over the number used by MIQCR.

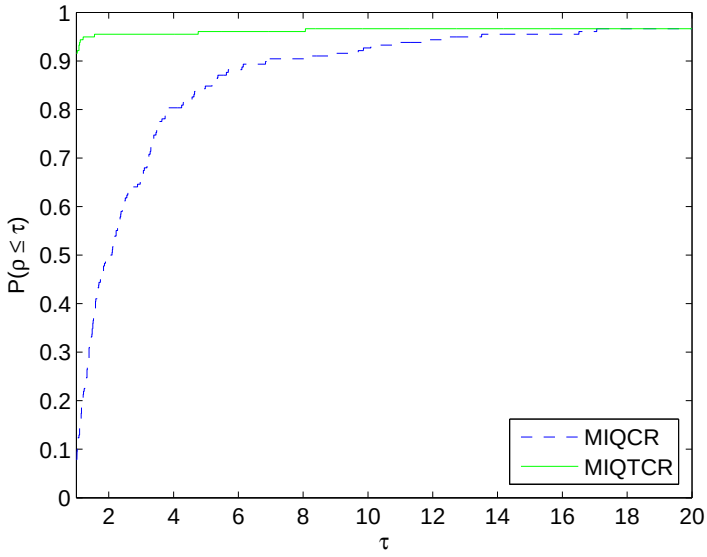
$8, 12, 16, 20$ and $n = 8, 10, 12, 14$ respectively. For each value of n four values of n_c were used. The values were chosen to evenly divide the interval $[n/2 + 1, n - 1]$. For $n = 8$, $n_c = 4$ was also to construct the test set. For each pair of values of n and n_c five test problems were generated creating a test set of 240 problems. The results obtained when solving the test problems using MIQCR and MIQTCR are presented in the form of a performance profile in Figure 6.12. Considering Figure 6.12 we see that MIQTCR has a clear advantage over MIQCR for problems of the type considered here with $n_c > n_d$. This advantage is due to the fact that for $n_c > n_d$ the reduction in the number of variables generated by MIQTCR has a greater effect than the tighter relaxation gap generated by MIQCR. As noted before, these results obviously only apply when $H_{cc} \succ 0$. When H_{cc} is positive semidefinite rather than positive definite MIQCR is the only applicable reformulation scheme.

We now consider the case $n_d > n_c$. The test set for problems of this form was constructed using type 1, 2 and 3 constraints with $n = 8, 11, 14, 17$, $n = 8, 10, 12, 14$ and $n = 8, 9, 10, 11$ respectively. As before, four values of n_c were chosen for each n to evenly divide the interval $[1, n/2 - 1]$ with $n_c = 4$ also being used for $n = 8$. Smaller values of n were used than when testing $n_c > n_d$ as the difficulty of the problems increases as n_d increases. The results for this set of problems are given in Figure 6.13. We see that, as expected, MIQCR is the superior reformulation scheme when $n_d > n_c$.

We now present results which allow us to determine whether there are

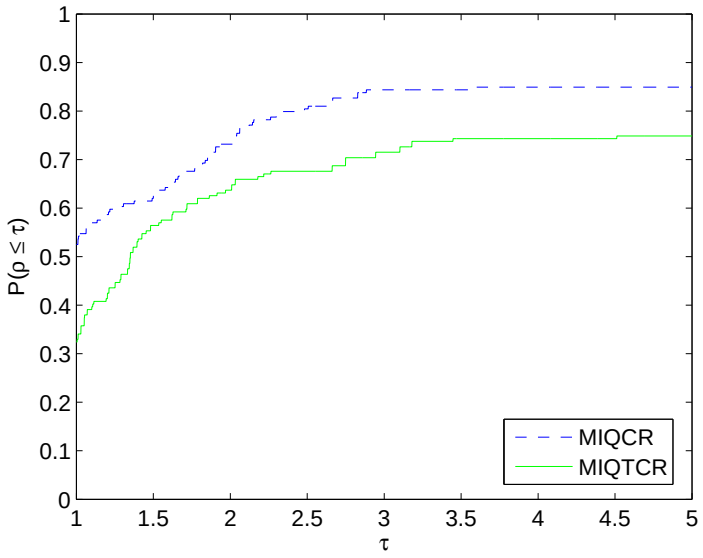


(a) Performance profile for $\tau \in [1, 5]$

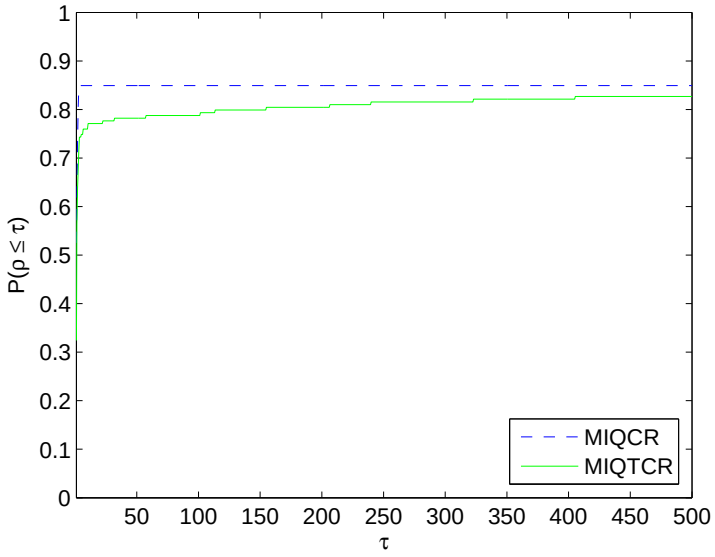


(b) Performance profile for $\tau \in [1, 20]$

Figure 6.12: Performance profile examining the effectiveness of the reformulations schemes MIQCR and MIQTCR when $n_c > n_d$.



(a) Performance profile for $\tau \in [1, 5]$



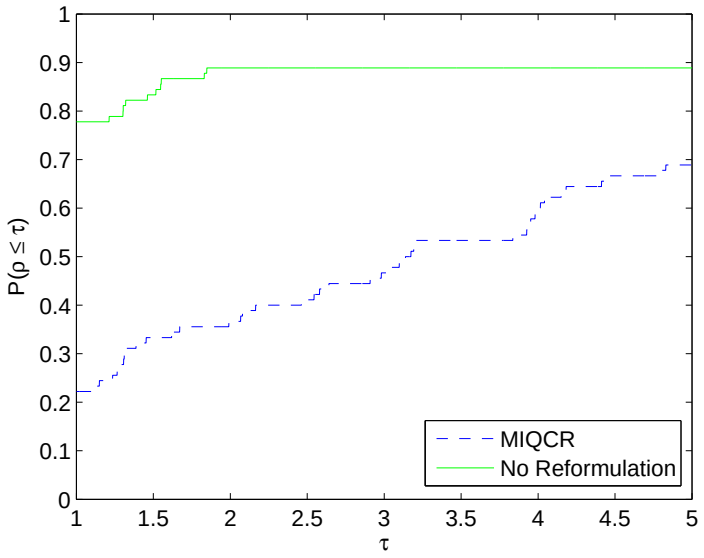
(b) Performance profile for $\tau \in [1, 500]$

Figure 6.13: Performance profile examining the effectiveness of the reformulations schemes MIQCR and MIQTCR when $n_c < n_d$.

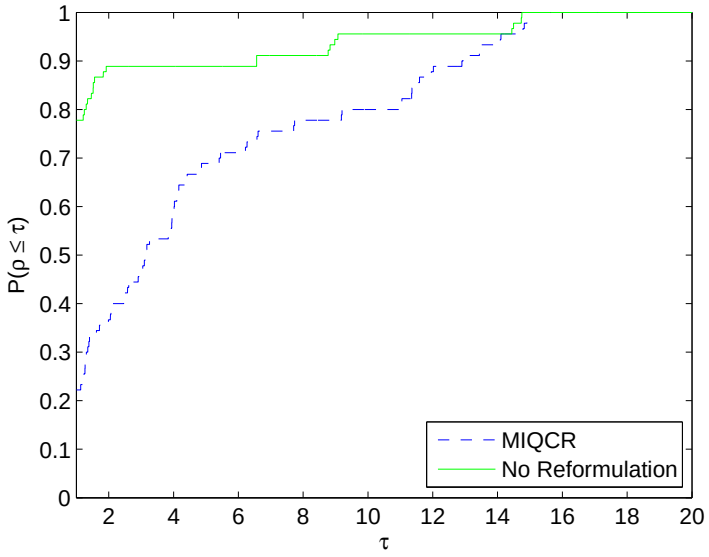
situations in which it may be more effective to solve problem (1.2) directly rather than applying MIQCR or MIQTCR. The test set was constructed with $n_c = n_d$ using type 1, 2 and 3 constraints with $n = 2, 4, 6, \ldots, 16$. For each value of n ten random problems were generated creating a test set of 240 problems. From Figure 6.11 we see that, since $n_c = n_d$, MIQCR and MIQTCR will produce similar results on this set of problems. Therefore, we shall not consider MIQTCR in this section. This helps make the comparison of the solution times of the original and reformulated problems clearer. The problems generated by MIQCR were solved using CPLEX 12.1 while problem (1.2) was solved using SCIP 3.0. Both CPLEX and SCIP are commercial solvers which should make the comparison fair. CPLEX and SCIP were both called from MATLAB. CPLEX and SCIP were run with their default options. The default absolute convergence tolerances are the same for CPLEX and SCIP. The relative convergence tolerances of CPLEX and SCIP cannot be directly compared since they are defined differently. The results are presented in two parts. The results obtained with $n \leq 6$ are given in Figure 6.14 while the results obtained with $n \geq 8$ are given in Figure 6.15. It is clear from the figures that for $n \leq 6$ it is more efficient to solve problem (1.2) directly rather than using a convex reformulation. For $n \geq 8$ this result is reversed and it becomes more efficient to apply a convex reformulation. This behaviour is caused by the contribution of the solution time of the SDP to the solution time of MIQCR. For small n the SDP makes a large contribution to the solution time. For large n the solution time of the SDP becomes negligible compared to the solution times of CPLEX and SCIP.

In summary, the results presented in this section allow the following conclusions to be drawn. When solving MIQPs of this type with more continuous variables than integer variables it is more efficient to use MIQTCR than MIQCR. When $n_c < n_d$ this result is reversed and it becomes more efficient to use MIQCR. When $n_c = n_d$ the two reformulation scheme appear to be roughly equivalent; while MIQTCR solves the greatest number of problems in the shortest time it takes much longer than MIQCR for a number of problems. Further for $n \leq 6$ it is more efficient to solve problem (1.2) directly rather than using a convex reformulation. For $n \geq 8$ this result is reversed and it becomes more efficient to apply a convex reformulation.

We now consider the specific case of solving the MIQP subproblems in the MBOQA algorithms. As before we consider the average results obtained for bound constrained problems. The test set was constructed as follows. We consider problems with type 1 constraints and $n_c = n_d$. Problems with $n = 4, 8, 12, \ldots, 32$ were used. For each value of n ten test problems were randomly generated creating a test set of 80 problems. The reformulated problems were solved using CPLEX. The results are presented in Table 6.3.

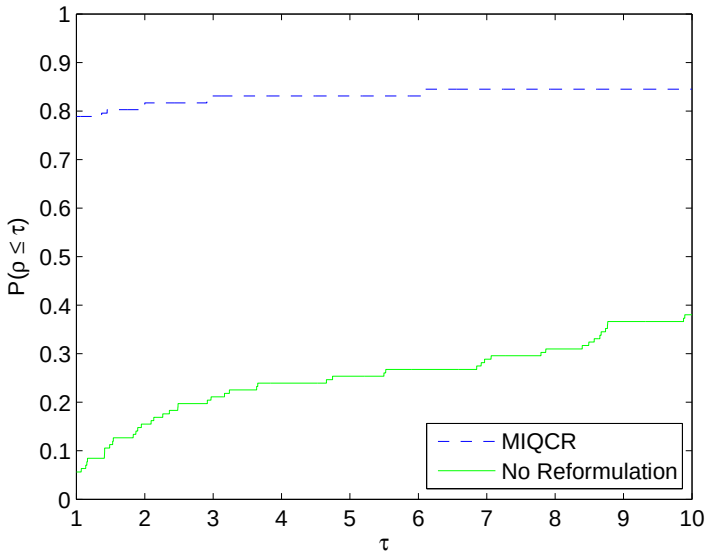


(a) Performance profile for $\tau \in [1, 5]$

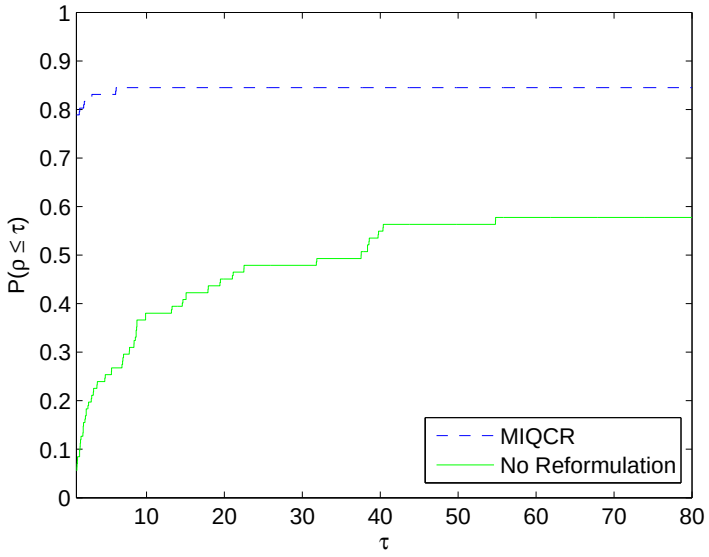


(b) Performance profile for $\tau \in [1, 20]$

Figure 6.14: Performance profile comparing the solution times of the reformulated problem and the original problem for $n \leq 6$.



(a) Performance profile for $\tau \in [1, 10]$



(b) Performance profile for $\tau \in [1, 80]$

Figure 6.15: Performance profile comparing the solution times of the reformulated problem and the original problem for $n \geq 8$.

n	MIQCR			MIQTCR		
	# nodes	time	us	# nodes	time	us
4	92	0.6848	0	77	0.6652	0
8	398	0.9113	0	980	0.8546	0
12	1248	1.5001	0	60351	1.9133	0
16	4914	4.3249	0	1.04×10^6	23.643	0
20	5729	9.2330	0	16.59×10^6	470.86	0
24	9556	31.084	0	0.118×10^9	2148.3	1
28	23366	84.736	0	0.270×10^9	8310.9	3
32	1.00×10^6	268.21	0	0.284×10^9	8909.5	6

Table 6.3: The results obtained when solving problems with constraints of type 1, $n_c = n_d$ and H_{cc} positive definite. The reformulated problems are solved using CPLEX. The number of unsolved problems is denoted by us.

The table gives the time taken and number of nodes required to solve the problems generated by MIQCR and MIQTCR for each value of n. The values given are the average of the time and number of nodes for the ten test problems with each value of n.

Considering the results given in Table 6.1 we see that for $n \leq 8$ it is more efficient to use MIQTCR while for $n > 8$ it is more efficient to solve the problem using MIQCR. This result is used to choose the convex reformulation scheme applied to the MIQP subproblems in the MBOQA algorithms when H_{cc} is positive definite. This completes our comparison of MIQCR and MIQTCR.

Results obtained using MINLP solver

We now examine the results obtained when solving the problems generated by MIQCR, MIQTCR and MIQTBC using MINLP solvers. As noted earlier, these results are not of great practical interest and are included for completeness. The reformulated problems were solved using Couenne 0.3.2 on the NEOS server [49, 66]. The reason that the NEOS server was used for these problems rather than making use of the Couenne binaries is the following. The Couenne binaries require .nl files as input, these files are generated by AMPL. However the author only had access to the student version of AMPL which only accepts problems with fewer than 300 variables and constraints. Since in this section we are using Couenne to solve the reformulated problems, rather than problem (1.2), the number of constraints

n	MIQCR	MIQTCR	MIQTBC
4	0.296	0.376	0.074
6	3.296	2.324	0.614
8	4.460	10.836	2.482
10	15.170	28.871	5.796
12	31.754	84.886	33.64
14	98.341	756.34	152.19
16	300.31	3084.4	1034.6

Table 6.4: The time taken to solve problems with constraints of type 1 with H_{cc} positive definite using Couenne.

n	MIQCR	MIQTCR	MIQTBC
4	5.412	4.313	1.330
6	42.082	20.522	6.456
8	47.235	49.611	19.410
10	110.43	192.12	151.96
12	301.37	451.29	475.54
14	1032.1	1688.3	2012.5

Table 6.5: The time taken to solve problems with constraints of type 2 with H_{cc} positive definite using Couenne.

is generally greater than 300 for $n \geq 6$. The required SDPs were again solved using SeDuMi 1.3 with its default options. We again solved problems with constraints of types 1, 2 and 3. For all three reformulation schemes we solved randomly generated MIQPs with $n_c = n_d$, n was varied between 4 and 16 for type 1 constraints, between 4 and 12 for type 2 constraints and between 4 and 10 for type 3 constraints. It was noted in section 4.3.3 that MIQTBC can only be applied if the elements of x can be reordered such that $H_{dd} - H_{cd}^T H_{cc}^{-1} H_{cd}$ has at least two principal leading submatrices which are not negative semidefinite. This property was guaranteed by testing each problem as it was generated and discarding it if the property did not hold. The time taken to solve a problem with a certain n was taken as the average time taken to solve 5 randomly generated problems with that n. The time taken to solve the test problems for constraints of types 1, 2 and 3 are given in Tables 6.4, 6.5 and 6.6 respectively.

n	MIQCR	MIQTCR	MIQTBC
4	3.094	1.714	0.657
6	15.83	10.15	12.45
8	99.32	255.03	68.34
10	5352.3	3687.3	1958.6

Table 6.6: The time taken to solve problems with constraints of type 3 with H_{cc} positive definite using Couenne.

It is clear from Tables 6.4, 6.5 and 6.6 that MIQTBC is the superior convexification scheme for constraints of type 1 when $n < 12$, for constraints of type 2 when $n < 10$ and for constraints of type 3 for all n examined. As explained in the previous section, the superiority of the various methods is due to a trade off between the number of variables in the reformulated problem and the size of the relaxation gap. We again note that if CPLEX had been used to solve problems (4.34) and (4.39) MIQCR and MIQTCR would have been more efficient that MIQTBC.

6.1.3 Results obtained with H_{cc} singular

We now consider the results obtained when H_{cc} is singular. As before we first examine the effectiveness of the transformation as n increases. We use the same method described in section 6.1.1 to test the effectiveness of the transformation but we now generate Hessians where H_{cc} is singular using the method described in appendix C. The values of μ, ν and ω in Algorithm 4 were set by numerical experiment. In order to simplify the selection of these parameters it was decided that each of the three parameters would be given the same value and this value would be denoted by σ. For problems with $n < 10$ we used $\sigma = 2$ and for $n \geq 10$ we used $\sigma = 2.5$.

Problems with the form considered in this section must be solved using Branch and Bound algorithms. Making use of the special structure of the objective function of problem (4.51) requires a change in the underestimating procedure used by the Branch and Bound algorithm. The algorithm needs to be told to split the Hessian into the two terms $\Theta^{(1)}$ and $\Theta^{(2)}$ and that none of the bilinear terms in the $\Theta^{(2)}$ need to be underestimated. Since this requires a change in the algorithm rather than the simple preprocessing that could be applied in section 6.1.1 we shall only consider Algorithm 2 in this section; we shall not consider BARON or SCIP.

The test sets in this section use the same values of n and n_c as those

used to test Algorithm 2 in section 6.1.1 and their description shall not be repeated here. Since q cannot be varied we instead generate 10 random test problems when $n_c = n_d$ and 7 random test problems for each pair of values of n and n_c when $n_c \neq n_d$. The number of problems in the test sets for $n_c = n_d$, $n_c > n_d$ and $n_c < n_d$ are 120, and 210 and 168 respectively. The results for $n_c = n_d$, $n_c > n_d$ and $n_c < n_d$ are presented in the form of a performance profiles in Figures 6.16, 6.17 and 6.18 respectively.

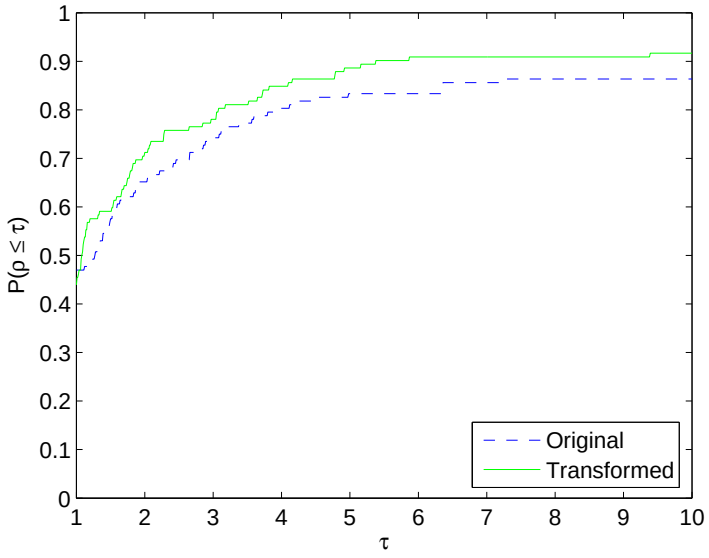
Considering Figures 6.16, 6.17 and 6.18 we see that the behaviour observed in section 6.1.1 is repeated here. The explanation for this behaviour is the same as that given in section 6.1.1. While the results presented here are positive a major drawback of this method is the sensitivity of the transformation to the choice of σ. A poor choice of σ can lead to a reversal of the results with problem (1.2) becoming easier to solve than problem (4.51) in all cases. We also note that for $n_c = 1$ the elements of U_{cc} become so small that numerical errors cause (4.8) and (4.9) to be unbounded for some variables. This gives us infinite original bounds on these variables so we cannot solve the transformed problem. This effect is independent of n; (4.8) and (4.9) are always unbounded for $n_c = 1$ but not for higher values of n_c.

We now consider the specific case of solving the MIQP subproblems in the MBOQA algorithms. As before we consider the average results obtained for bound constrained problems. The test set is constructed using the same values of n and n_c as those used in section 6.1.1. For each pair of values of n and n_c ten random test problems were generated. We also note that for these tests the time limit was extended to 20000 seconds. The results obtained using Algorithm 2 are presented in Table 6.7. The table gives the time taken and number of nodes required to solve the original and transformed problem for each value of n. The values given are the average of the time and number of nodes for the ten test problems for each value of n.

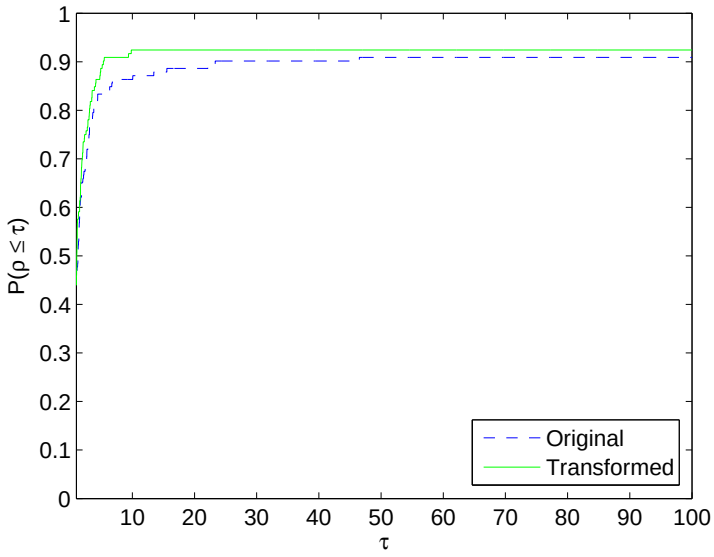
Considering the results given in Table 6.7 we see that it is more efficient to solve problem (4.51) than problem (1.2). This result is used when deciding whether or not to transform a MIQP subproblem in the MBOQA algorithms when H_{cc} is singular. This completes our discussion of the MIQP solution methods.

6.2 Mixed integer non-linear programming

In this section we examine the results obtained when using HEMBOQA, SEMBOQA and COMBOQA to solve a number of MINLP test problems. These results are compared to the results obtained using NOMAD. Thirty test functions, with a wide range of difficulty, were considered. Most of the

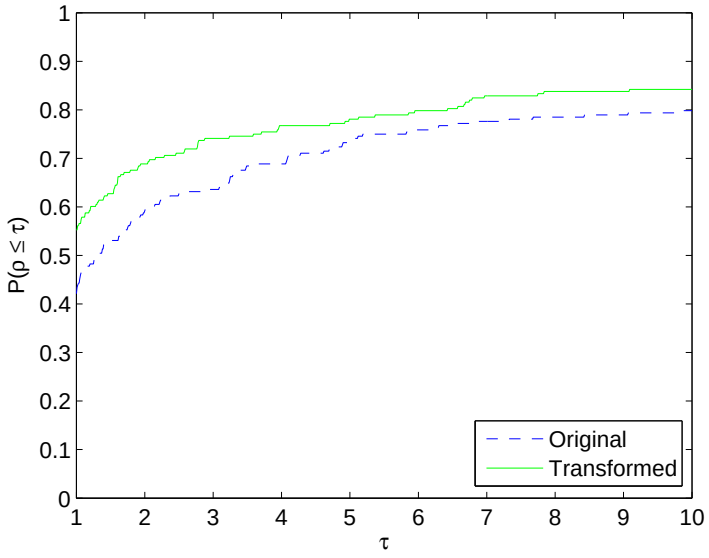


(a) Performance profile for $\tau \in [1, 10]$

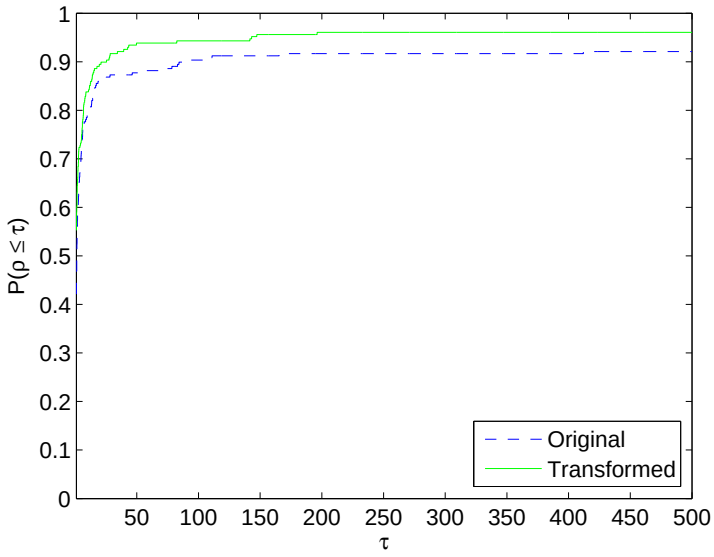


(b) Performance profile for $\tau \in [1, 100]$

Figure 6.16: Performance profile examining the effectiveness of the transformation when solving problems with $n_c = n_d$ and H_{cc} singular using Algorithm 2.

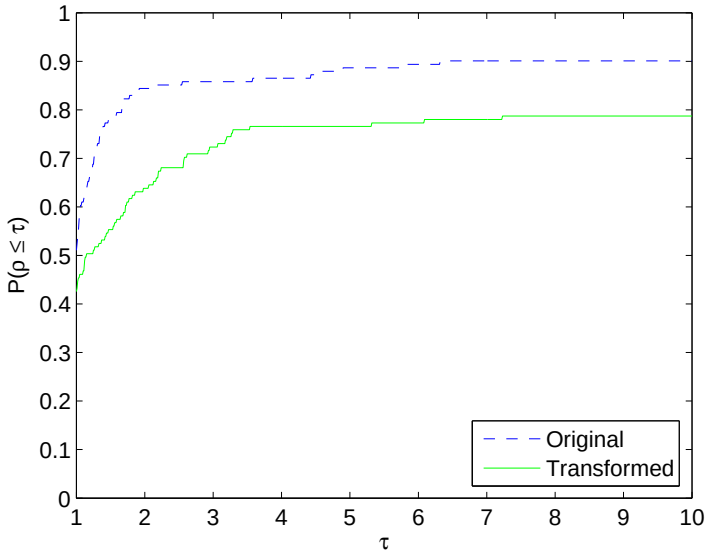


(a) Performance profile for $\tau \in [1, 10]$

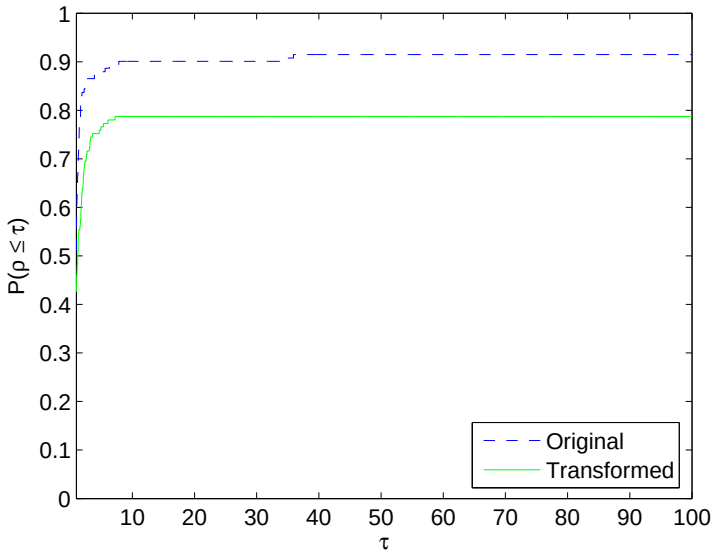


(b) Performance profile for $\tau \in [1, 500]$

Figure 6.17: Performance profile examining the effectiveness of the transformation when solving problems with $n_c > n_d$ and H_{cc} singular using Algorithm 2.



(a) Performance profile for $\tau \in [1, 10]$



(b) Performance profile for $\tau \in [1, 100]$

Figure 6.18: Performance profile examining the effectiveness of the transformation when solving problems with $n_c < n_d$ and H_{cc} singular using Algorithm 2.

n	Original problem			Transformed problem		
	time	# nodes	us	time	# nodes	us
4	0.4070	3.400	0	0.3182	7.600	0
6	27.134	285.4	0	0.7002	15.40	0
8	5302.2	26851	0	6.7017	149.7	0
10	838.76	2905	0	17.460	264.0	0
12	18983	30137	8	55.132	561.3	0
14	20000	16502	10	208.31	1556	0
16	20000	11409	10	2275.7	11773	2
18	20000	5430	10	20000	49470	10

Table 6.7: The results obtained when solving problem (1.2) and problem (4.27) with constraints of type 1, $n_c = n_d$ and H_{cc} singular using Algorithm 2. The number of unsolved problems is denoted by us.

functions considered are defined for n variables, for these functions problems were solved for $n = 2, 4, 6, 8, 10$. A list of the test functions used is given in Table 6.8. The table also gives the number of variables used for each of the problems. More detailed descriptions of the test functions are given in appendix C. The problems generated by these test functions are continuous, they were transformed into MINLPs by restricting x_i, $i = \frac{n}{2} + 1, \ldots, n$ to integral values. The problems listed in Table 6.8 were all solved using HEMBOQA, SEMBOQA, COMBOQA and NOMAD. Following [102], $\mathcal{N}_r(x)$ is defined as follows

$$\mathcal{N}_r(x) = \{y \in \mathbb{R}^n : y_c = x_c,\ \|y_d - x_d\| \leq 1\}.$$

We recall that $\mathcal{N}_r(x)$ is used in the definitions of local minima for SEMBOQA, COMBOQA and NOMAD. BOBYQA was used as the neighbourhood solver in HEMBOQA, SEMBOQA and COMBOQA.

All tests in this section were performed on a PC with an Intel Core 2 Duo CPU at 2.8GHz with 2GB of RAM running 32-bit Windows Vista. When the four algorithms were used to solve the same problem the algorithms were run consecutively with no time gap between the end of an algorithm and the start of the next. This was done to ensure a similar computational load. All solutions were checked for feasibility. The maximum time and maximum number of function evaluations for HEMBOQA, SEMBOQA, COMBOQA and NOMAD were set to infinity. The remaining parameters of the MBOQA algorithms were set as follows; $\gamma_{\text{req}} = n_c$, $looplength = 200$, $(\rho_{beg})_i = 1$,

Problem Name	n	Problem Name	n
Absolute value function	2,4,6,8,10	Powell's quadratic problem	4
Ackley's function	2,4,6,8,10		
Axis parallel hyper-ellipsoid function	2,4,6,8,10	Rastrigin's function	2,4,6,8,10
		Rosenbrock's valley	2,4,6,8,10
Becker and Lago Problem	2	Rotated hyper-ellipsoid function	2,4,6,8,10
Branin's function	2		
Dekkers and Aarts problem	2	Salomon problem	2,4,6,8,10
Easom's function (shifted)	2	Schaffer problem 1	2,4,6,8,10
Exponential problem	2,4,6,8,10	Schwefel's function (shifted)	2,4,6,8,10
Fourth function of De Jong	2,4,6,8,10		
Goldstein-Price function	2	Sinusoidal problem (shifted)	2,4,6,8,10
Griewangk's function	2,4,6,8,10		
Hosaki problem	2	Sphere function	2,4,6,8,10
Levy function	2,4,6,8,10	Step function	2,4,6,8,10
Linear function	2,4,6,8,10	Sum of different powers function	2,4,6,8,10
Michalemicz's function	2,4,6,8,10		
Neumaier problem 3	2,4,6,8,10	Wood's function	4
Perm function	2,4,6,8,10	Zakharov function	2,4,6,8,10

Table 6.8: The test functions used to examine the effectiveness of the derivative free MINLP algorithms.

$(\rho_{end})_i = 0.0001$, $i = 1, \ldots, n_c$ and $(\rho_{end})_i = 1$, $i = n_c + 1, \ldots, n_d$. The MATLAB implementation of NOMAD, NOMADm 4.7 [4], was used in these tests. NOMADm was run with all of its parameters at their default values, the default convergence tolerance of NOMADm is 0.0001. We note that since the maximum time and maximum number of function evaluations were set to infinity in our algorithms there were no failed tests.

Now, we see from Table 6.8 that each of the four algorithms was used to solve 118 test problems. The results obtained are given in detail in appendix B. However, as one would expect with this large number of test problems, the results in appendix B are rather lengthy and difficult to interpret. Accordingly we present a number of averaged quantities in this section which give a clearer idea of the behaviour of the algorithms.

Our first goal is to compare the effectiveness of the different algorithms. The properties of the solution which can be used to compare the algorithms are the value of $f(x^*)$, the time taken to solve the problem and the number of function evaluations required to solve the problem. The timing results considered here and in appendix B are the results obtained by the algorithms when function evaluations are inexpensive. We first note that NOMAD takes

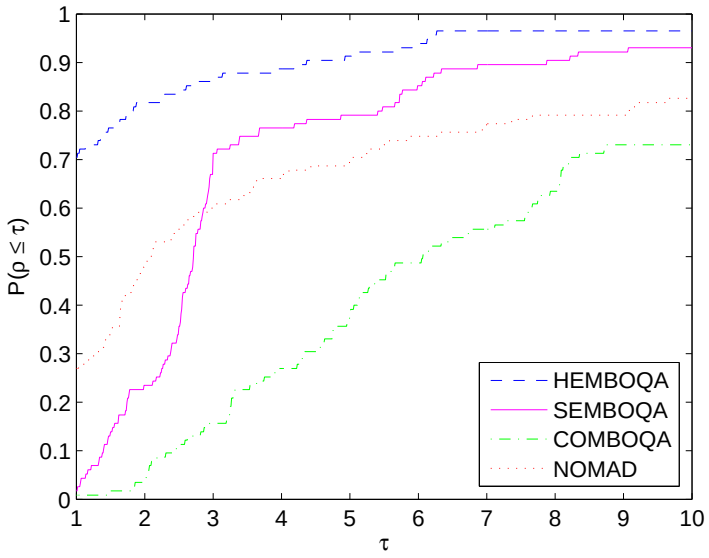
far less time to solve all of the test problems than the trust region based methods. However, in this thesis we are more concerned with problems involving expensive function evaluations. We therefore use $f(x^*)$ and the number of function evaluations as performance measures. We shall use two different approaches to compare our algorithms. The first approach is from [44, 117]. In this approach each algorithm is given a score of 1 to 4 for each test problem depending on its relative effectiveness for that problem. The best algorithm is given a score of 1. The average scores for each n as well as the average scores for the entire test set are given in Table 6.9. We also compare the algorithms using a performance profile with the number of function evaluations used as the performance measure. The performance profile is plotted in Figure 6.19. We cannot use performance profiles when using $f(x^*)$ as a performance measure. Instead we use the m-fold improvement $m_{p,s}$ defined in [13]

$$m_{p,s} = \frac{f_p(x_s^*) - f_p^b}{f_p^w - f_p^b}, \tag{6.2}$$

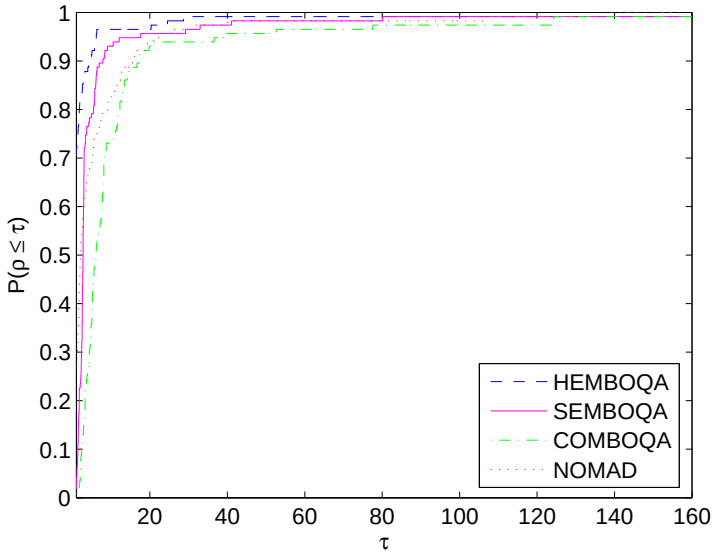
where f_p^b and f_p^w are, respectively, the best and worst objective function values returned by the derivative free algorithms when solving problem p and $f_p(x_s^*)$ is the solution returned by algorithm s for problem p. Similarly to performance profiles we plot the cumulative distribution function of the m-fold improvement in Figure 6.20.

From Table 6.9 and Figure 6.20 we see that COMBOQA is the superior algorithm when $f(x^*)$ is used as a performance measure. This is as expected since COMBOQA uses the strongest definition of a local minimum, Definition 12. HEMBOQA is the worst performing algorithm when using $f(x^*)$ as the measure of performance. Again, this is the expected result since HEMBOQA is a heuristic. It is also clear that SEMBOQA performs better than NOMAD even though they use the same definition of a local minimum, Definition 10. In summary, using $f(x^*)$ as the measure of performance the algorithms can be listed in order of effectiveness as follows; COMBOQA, SEMBOQA, NOMAD, HEMBOQA.

From Table 6.9 and Figure 6.20 we see that, as might be expected, the results are reversed when using the number of function evaluations as the measure of performance. HEMBOQA uses the lowest number of function evaluations most often since its heuristic nature means that it does not need to check that the point it returns is actually a local minima. The results obtained for SEMBOQA and NOMAD are much closer than when using $f(x^*)$ as the measure of performance, it appears that NOMAD has a slight advantage. COMBOQA consistently requires the greatest number of function evaluations. This is due to the fact that far more function evaluations are



(a) Performance profile for $\tau \in [1, 10]$



(b) Performance profile for $\tau \in [1, 160]$

Figure 6.19: Performance profile comparing the derivative free algorithms using number of function evaluations as a performance measure.

n	Criteria	Score			
		HEMBOQA	SEMBOQA	COMBOQA	NOMAD
2	$f(x^*)$	2.0741	1.5185	1.1481	1.5556
	# evals	1.1852	2.4444	3.6667	2.6270
4	$f(x^*)$	2.0000	1.2174	1.0870	1.8261
	# evals	1.3478	2.5652	3.6957	2.3913
6	$f(x^*)$	2.0000	1.4762	1.2857	1.8095
	# evals	1.4286	2.6190	3.5238	2.4286
8	$f(x^*)$	2.0000	1.2857	1.2381	1.9048
	# evals	1.6667	2.4286	3.5238	2.3810
10	$f(x^*)$	2.0952	1.6190	1.4762	1.8095
	# evals	1.6190	2.5238	3.4286	2.4286
Average	$f(x^*)$	2.0339	1.4234	1.2470	1.7811
	# evals	1.6272	2.3310	3.0640	2.2370

Table 6.9: Rankings of the four derivative free algorithms using the objective function value and the number of function evaluations as performance criteria.

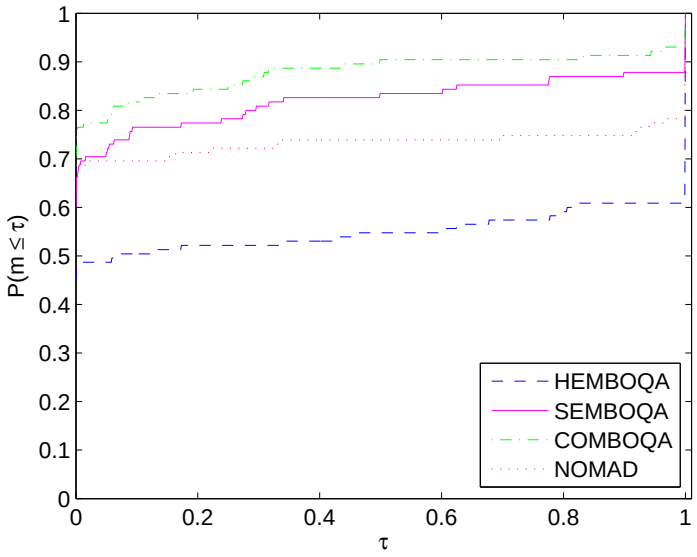


Figure 6.20: Cumulative probability distribution of the m-fold improvement of the derivative free algorithms.

required to construct $\mathcal{N}_{\text{comb}}(x)$ than are required to construct $\mathcal{N}_r(x)$. Using the number of function evaluations as the measure of performance the algorithms can be listed in order of effectiveness as follows; HEMBOQA, NOMAD, SEMBOQA, COMBOQA.

We can now suggest the following guidelines to be used when choosing between these four algorithms. If the objective function evaluations are inexpensive NOMAD should be the algorithm of choice. On the other hand, if the objective function is expensive a trade off must be made between the quality of the solution and the number of function evaluations required. If the objective function is very expensive HEMBOQA should be used as it generally requires the lowest number of function evaluations. If the quality of the solution is very important COMBOQA should be used. For problems where both the number of function evaluations and the quality of the solution are important we recommend the use of SEMBOQA or NOMAD. Though, as discussed in chapter 5, the use of these algorithms does involve the risk that increasing the size of $\mathcal{N}_r(x)$ will not improve the quality of the solution. If the size of $\mathcal{N}_r(x)$ is being increased to improve the quality of the solution it would be better to use COMBOQA.

We now wish to examine more closely the effect that increasing n has on the number of function evaluations required by our trust region methods. To this end the average number of function evaluations required for each n were considered. In obtaining these averages only those problems which are defined for any n were considered. Clearly considering a problem which is only defined for $n = 2$ will not help in determining behaviour as n changes. The average number of function evaluations are plotted in Figures 6.21, 6.22 and 6.23 for HEMBOQA, SEMBOQA and COMBOQA respectively.

We see from Figure 6.21 that the number of function evaluations required by HEMBOQA is approximately $\mathcal{O}(n^{2.05})$ on average. From Figures 6.22 and 6.23 we see that increasing n has a greater effect on the number of function evaluations for SEMBOQA and COMBOQA. We see that the number of function evaluations required by SEMBOQA is approximately $\mathcal{O}(n^{2.24})$, while the number of evaluations required by COMBOQA is approximately $\mathcal{O}(n^{2.51})$. As expected, n has the greatest effect on the complexity of COMBOQA and the smallest effect on HEMBOQA. However the difference between these complexities is surprisingly small. These results are poorer than the results obtained for BOBYQA in which the number of function evaluations appears to increase linearly with n. However, given the difficulty added to the problem by the integer variables, we feel that keeping the number of function evaluations required to approximately $\mathcal{O}(n^2)$ is a good result. These results are important as our methods are applicable to problems in which the objective function is expensive. Obviously then, it is desirable

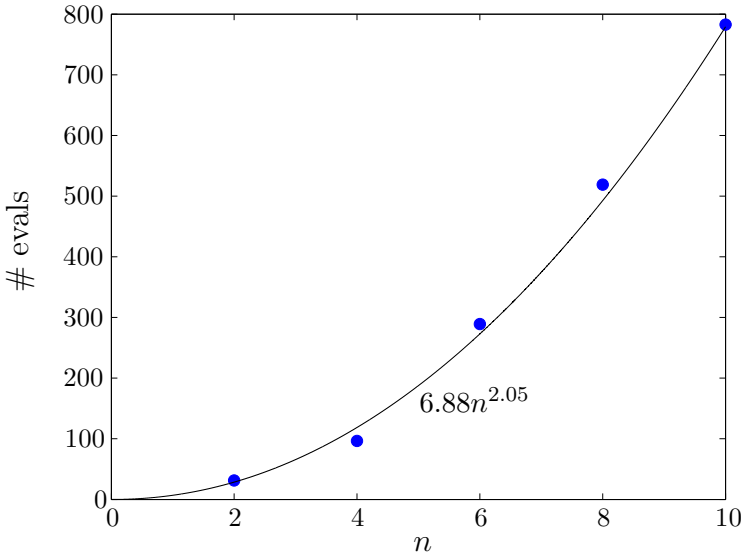


Figure 6.21: A plot showing the effect of increasing n on the average number of function evaluations required by HEMBOQA to solve the test problems.

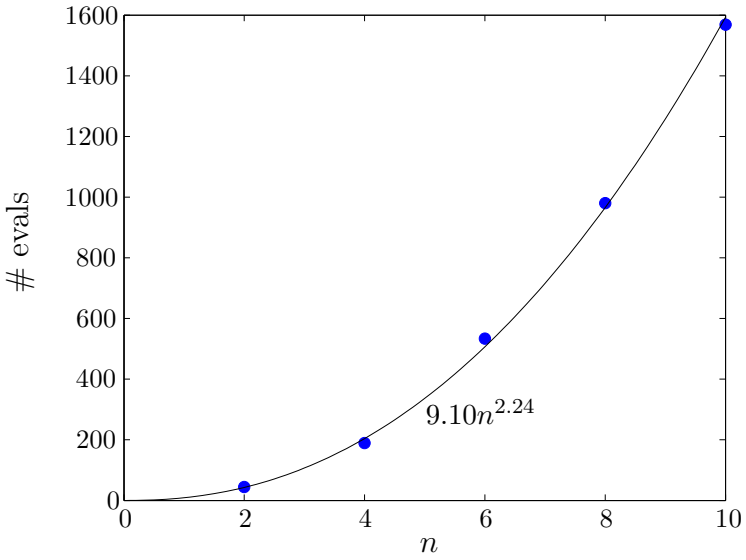


Figure 6.22: A plot showing the effect of increasing n on the average number of function evaluations required by SEMBOQA to solve the test problems.

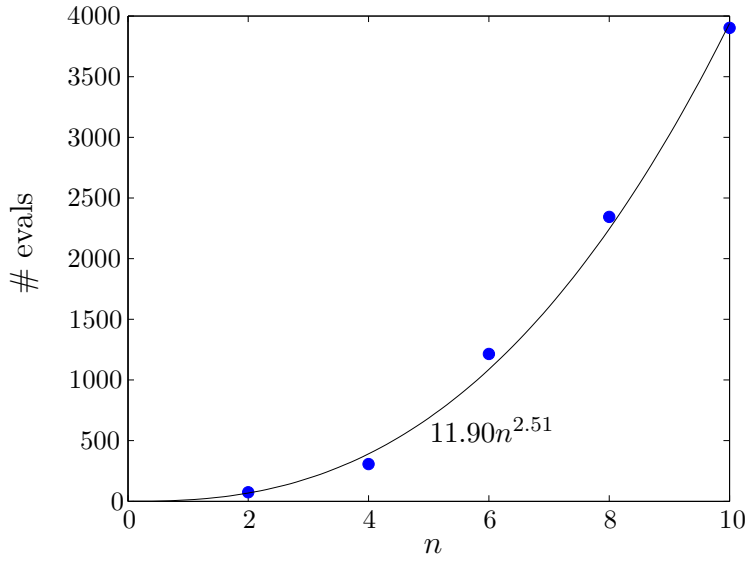


Figure 6.23: A plot showing the effect of increasing n on the average number of function evaluations required by COMBOQA to solve the test problems.

that the number of function evaluations not increase too drastically with n.

We shall not provide an analysis of the time taken by the trust region algorithms since, as noted above, NOMAD is far superior to the trust region methods when time taken is used as the measure of performance. This can clearly be seen by considering the results in appendix B. The large difference in the time is mainly due to the difficulty of solving the MIQP trust region subproblems, problem (5.12) and problem (5.13), when the n_cth principal leading submatrices of their Hessians are indefinite or singular.

Finally we note that the derivative free algorithms found the global minima of the test problems for a surprisingly large number of test problems considering that they are designed to find local minima. The percentage of test problems for which the algorithms located the global minimum are given in Table 6.10. As should be expected from the previous results, COMBOQA finds the greatest number of global minima while HEMBOQA finds the smallest number.

As was mentioned in chapter 3, computational results are also given in [102]. However, for two reasons, their results cannot easily be compared to those given here. Firstly no values for x_0 are given in [102]. Secondly they do not consider integer variables, rather they restrict some of the variables

HEMBOQA	SEMBOQA	COMBOQA	NOMAD
47.4%	54.2%	58.5%	52.5%

Table 6.10: Percentage of test problems for which the global minimum was found.

to the following values

$$x_i = l_i + k\frac{u_i - l_i}{10}, \quad k = 0, \ldots, 10,$$

This is only equivalent to our constraint that $x_i \in \mathbb{Z}$ when $u_i - l_i = 10$. However, Liuzzi et al. [102] compare their algorithms to NOMAD and the two methods are roughly equivalent when using $f(x^*)$ and the number of function evaluations as measures of performance, no timing results are given. It follows that the conclusions drawn here regarding the relative efficiency of NOMAD will likely also hold for the algorithms in [102].

6.3 Conclusion

Extensive computational results have been given illustrating the effectiveness of the methods developed in chapters 4 and 5. The results show that our methods give an improvement in efficiency over currently available methods in a number of situations. There are also situations in which our approaches are inferior to those currently existing in the literature. Since no method is superior in every situation these results should be used as a guide to aid in deciding which method should be applied to solve specific problems.

Chapter 7

Conclusion

A new type of deterministic derivative free mixed integer algorithm has been introduced in this thesis. This is the first algorithm which makes use of trust regions to solve derivative free MINLPs. The algorithm is an extension of BOBYQA [126] to the mixed integer case. Three implementations of the algorithm have been developed; deterministic convergence results have been proven for two of the implementations. The remaining contributions of this thesis are the development novel of approaches for solving non-convex MIQPs and a rigorous approach to the definition of local minima of mixed integer problems. These contributions were found to be required during the development of the derivative free algorithm. Extensive computational results have been given showing the effectiveness of the contributions made in this thesis.

7.1 Summary

Chapter 2 contains a review of derivative based mixed integer methods. The review is divided into two parts. In the first part derivative based methods for solving both convex and non-convex MINLPs are reviewed. Obviously these methods cannot be used to solve our problem since they use derivatives. The review of these methods is included for completeness and does not give many technical details. The second part of the chapter contains a review of methods developed specifically for solving MIQPs. We review both convex and non-convex methods. This material is more relevant to our work. Accordingly the review is more detailed than that of the general MINLP methods.

Chapter 3 contains a review of derivative free methods for solving both continuous and mixed integer problems. These methods are divided into three classes in the review; metaheuristics, surrogate optimization and local

sampling methods. The review of continuous methods is important since all of the mixed integer, derivative free methods that have been developed at this point are extensions of continuous methods to the mixed integer case. Chapter 3 also contains a detailed review of the BOBYQA algorithm [126], a continuous trust region method. The detail of the review is motivated by the fact our derivative free algorithm is an extension of BOBYQA to the mixed integer case. The review of BOBYQA contains a number of explanations of concepts considered obvious by the original author. Following the review of continuous derivative free methods is a review of the mixed integer approaches that have been developed to date. Emphasis is placed on the deterministic methods since they are the focus of this thesis. All of the deterministic methods reviewed are extensions of direct search methods to the mixed integer case.

In chapter 4 preprocessing procedures and solution methods are developed for solving non-convex MIQPs. Methods are developed for problems with three different types of Hessians; Hessians whose n_cth principal leading submatrices are positive semidefinite, invertible Hessians and singular Hessians. All the methods make use of a linear transformation with carefully chosen elements. The methods developed for Hessians whose n_cth principal leading submatrices are positive semidefinite are convex reformulation schemes based on convex reformulations developed in [33] and [122]. The methods developed for invertible and singular Hessians focus on using the linear transformation to reduce the number of terms in the objective function that need to be underestimated when solving the problem using a Branch and Bound algorithm.

In chapter 5 we develop our deterministic derivative free algorithm for solving mixed integer programs using trust regions. In the first section of the chapter we considered definitions of local minima for mixed integer programs. Due to the mixed nature of the variables a number of definitions can be envisaged, indeed a number of definitions have already been proposed in the literature. A number of deficiencies are exposed in the current definitions and a new definition is proposed that overcomes these problems. The new definition is developed by proposing four restrictions that the definition of a local minimum should satisfy. In the second part of chapter 5 three implementations of the derivative free algorithm are developed. The first implementation is a heuristic extension of BOBYQA to the mixed integer case. The differences between BOBYQA and our algorithm mainly stem from the fact that the trust region subproblems are now MIQPs rather than continuous quadratic programs. The second and third implementations are modifications of the heuristic which allow deterministic convergence results to be proven. The two implementations converge to two different types of

local minima.

Computational results are presented in chapter 6 showing the effectiveness of the methods developed in chapters 4 and 5. Each of the MIQP methods developed in chapter 4 was tested on a number of randomly generated problems with three different types of constraint. The methods developed in 5 were tested using thirty test functions which were used to generate a total of 118 test problems. The results of these tests allowed us to propose a number of guidelines which should be useful in determining which method should be applied in various situations.

7.2 Future Work

There are a number of areas in which it might be possible to extend or improve the work developed in this thesis.

Transformation for MIQPs The unimodular part of the transformation matrix used by the MIQP methods in chapter 4 was set to the identity matrix. When linear constraints are present in the problem the identity matrix is only an approximation of the problem we wish to solve to set unimodular matrix. It should be possible to find more efficient methods of solving the element assigning problems defined in chapter 4. Alternatively it may be possible to find a completely different approach to assigning to elements of the unimodular matrix.

Definition of local minima We have proposed a number of restrictions on the definition of a mixed integer local minimum. While these restrictions allow us to develop an improved definition they do not uniquely specify the definition, as discussed in chapter 5. It may be possible to develop more restrictions which allow the definition to be specified uniquely.

Time used by derivative free algorithm From the results presented in chapter 5 and appendix B it is clear that while our derivative free algorithm performs well in terms of the required number of function evaluations it performs poorly in terms of the time taken to solve the test problems. As was noted in chapter 5, this is mainly due to the time taken to solve the MIQP subproblems, especially when the subproblems are non-convex. Further work should be done on reducing the solution time. We can envisage a number of modifications to the algorithms which might allow this to be achieved. The MIQPs could be solved heuristically for large trust regions with deterministic methods only being applied near termination of the algorithm. Another approach could be to stop the major iterations of SEMBOQA and COMBOQA early; this could be thought of as a hybrid between pattern search and trust region methods. It might also be possible to reduce the solution

time by tuning the parameters of the algorithm. It should be noted that these modifications may negatively affect the number of function evaluations required by the algorithm.

Handling more general constraints The derivative free algorithm developed in this thesis can only solve problems with bound constraints. It would clearly be desirable to extend to method to more general constraints. It was mentioned in chapter 1 that a number of methods have been developed for extending bound constrained methods to solve problems with general constraints. These approaches include augmented Lagrangians [53, 96], inexact restoration [39], extreme barrier approaches [20], filter methods [18] and auxiliary functions [168]. A filter method is used in [3] to extend a mixed integer method to general constraints, it seems likely that a similar approach could be applied here.

Appendices

Appendix A

Pseudocode for the HEMBOQA Algorithm

In this appendix we present the HEMBOQA algorithm, developed in chapter 5, in pseudocode form. The notation used in the pseudocode is the same as that used in the description of HEMBOQA in chapter 5.

The HEMBOQA algorithm
input: x_0, ρ_{beg}, ρ_{end}, n_c, u, l, $maxevals$, $maxtime$, $looplength$, γ_{req}
$m := 2n + 1$
Set up the intial interpolation points, Q_1 and W_1 (see section 5.2.1)
Determine x_k (see page 34)
while The stopping conditions ((5.20)–(5.24)) are not satisfied **do**
  if (# function evaluations $>$ $maxevals$) $\vee$ (time $>$ $maxtime$) **then**
    stop and return x_k and $f(x_k)$
  end if
  Calculate γ_{in}
  if $\gamma_{\text{in}} < \gamma_{\text{req}}$ **then**
    Do not allow the replacement of any of the interpolation points on $\mathcal{X}_s$
  end if
  Find d_k using problem (5.12)
  if all$(d_k) < \epsilon$ **then**
    Stop and return x_k and $f(x_k)$
  end if
  if The integral components of $x_k + d_k$ have not been the same as the integral components of x_k for 20 consecutive iterations and all$(x_k + d_k = x_k)$ **then**
    $NoAlt := 1$

end if
if any$(d_k \geq 0.5\rho) \vee (noalt = 1) \vee ((\gamma_{\text{in}} < \gamma_{\text{req}}) \wedge (\text{any}(d_k \geq 0.1\rho))$ **then**
if there is a repeated pattern in the step lengths over the last *looplength* iterations (see section 5.2.3) **then**
Choose a random interpolation point y_i not equal to x_k
Find x_{new} by making the integer components equal to those of y_i and setting the continuous components to be the result of running the neighbourhood solver on $\mathcal{X}_{\text{i}}$
while all $(x_{\text{new}} - (x_k + d_k)) \leq 10^{-4}$ **do**
if All interpolation points have been considered **then**
Return x_k and $f(x_k)$ with a warning
end if
Choose a new random point which has not been considered before and find x_{new} as before
end while
$d_k := x_{\text{new}} - x_k$
end if
Calculate z and v using (3.35) and (3.41)
Select t using (5.14)
Calculate ϕ, ψ, τ and σ using (3.36)–(3.39)
while $(\sigma < 0.5\tau) \vee (\sigma > 10^{10})$ **do**
Use the RESCUE procedure (see section 3.3.5)
end while
if $f(x_k + d_k) < f(x_k)$ **then**
$x_{k+1} := x_k + d_k$
Select a new t using x_{k+1}, call it t_{rep} (see page 111)
Calculate α_{rep}, β_{rep}, σ_{rep} and τ_{rep} (see page 111)
if $\sigma_{\text{rep}} > 0.5\tau_{\text{rep}}$ (see page 111) **then**
$t := t_{\text{rep}}$ (see page 111)
$\alpha := \alpha_{\text{rep}}$ (see page 111)
$\beta := \beta_{\text{rep}}$ (see page 111)
$\sigma := \sigma_{\text{rep}}$ (see page 111)
$\tau := \tau_{\text{rep}}$ (see page 111)
end if
else
$x_{k+1} := x_k$
end if
Update the Q_k, W_k and the interpolation points (see section 3.3.5 and (3.5))
Calculate r_k using (5.16)
Calculate Δ_{k+1} (see section 5.2.3)

if $(r_k < 0.1) \wedge (f(x_k) > f(x_k + d_k)) \wedge (\exists i \text{ s.t. } |(y_i)_j - (x_k)_j| > 2,$
$j = n_c + 1, \ldots, n)$ **then**
$NextAlt := 1$
end if
if $(\Delta_{k+1} = \rho_k) \wedge (\text{all}(d_k \leq \rho_k)) \wedge (\text{all}(|\omega_k - x_k| \leq 2\rho_k)) \wedge$
$(f(x_k + d_k) \geq f(x_k)) \wedge (r_k < 0)$ **then**
if $\text{all}(\rho \leq \rho_{\text{end}})$ **then**
Stop and return x_k and $f(x_k)$
else
Reduce ρ using (5.25)
end if
end if
if $f(x_k + d_k) < f(x_k)$ **then**
$x_{k+1} := x_k + d_k$
end if
if $(r_k < 0.1) \wedge (NoAlt = 0) \wedge (f(x_k) > f(x_k + d_k)) \wedge$
$\text{all}(|y_\delta - x_k| > \max(2\Delta_k, 10\rho_k)$ **then**
$NextAlt := 1$
end if
end if
if (there was no trust region processing) $\wedge \text{all}(|\omega_k - x_k| \leq 2\rho_k)$ **then**
if $\text{all}(\rho \leq \rho_{\text{end}})$ **then**
Stop and return x_k and $f(x_k)$
else
Reduce ρ using (5.25)
end if
end if
if $(NextAlt = 1) \vee ((\gamma_{\text{req}} \geq \gamma_{\text{in}}) \wedge \text{all}(d_k < 0.5\rho)) \wedge (NoAlt = 0)) \vee$
$(\gamma_{\text{req}} < \gamma_{\text{in}}) \wedge \text{all}(d_k < 0.1\rho) \wedge (NoAlt = 0))$ **then**
Calculate δ_k using (5.17)
if $NextAlt = 1$ **then**
$\Delta_k := 0.1\Delta_k$
end if
Temporarily reduce Δ_k using (5.27)
Select t using (5.15)
Calculate the Lagrangian function Λ_t (see page 46)
Find d_k using problem (5.13)
Check for and deal with patterns in the step lengths using the procedure described in the trust region iteration
if $x_k + d_k$ is already an interpolation point **then**
$NoAlt := 1$

end if
if $NoAlt = 0$ **then**
Calculate z and v using (3.35) and (3.41)
Calculate ϕ, ψ, τ and σ using (3.36)–(3.39)
while $\sigma \leq 0.5\tau^2$ **do**
Use the RESCUE procedure (see section 3.3.5)
end while
Update the Q_k, W_k and the interpolation points (see section 3.3.5 and (3.5))
if $f(x_k + d_k) < f(x_k)$ **then**
$x_{k+1} = x_k + d_k$
end if
end if
end if
if the integral components of $x_k + d_k$ have not been the same as the integral components of x_k for 100 consecutive iterations **then**
Find $x_{k(\text{new})}$ by making the integer components equal to those of x_k and setting the continuous components to be the result of running the neighbourhood solver on $\mathcal{X}_s$
$x_k := x_{k(\text{new})}$
Stop and return x_k and $f(x_k)$
end if
end while

Appendix B

Results for the derivative free MINLP solvers

This appendix contains the detailed computational results obtained using HEMBOQA, SEMBOQA, COMBOQA and NOMAD. The results for $n = 2, 4, 6, 8, 10$ are given in Tables B.1, B.2, B.3, B.4 and B.5 respectively. For each problem the tables give the value of $f(x^*)$, the time taken to solve the problem, the number of objective function evaluations required to solve the problem and the distance between x^* and x^g. Here x^g denotes the global optimum. The norm used is the infinity norm. The value of $\|x^* - x^g\|_\infty$ is only given if the algorithm finds the global minimum of the test problem. In all of the tables the function names are shortened to a three letter key, the key is simply the first three letters of the function name, except for Schaffer problem 1 and Schwefel's function whose keys are SCA and SCW respectively. There are four rows in the tables for each problem key. The rows contain the results obtained using HEMBOQA, SEMBOQA, COMBOQA and NOMAD respectively. An analysis of the results contained in these tables is given in chapter 6.

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
ABS	4.31×10^{-9}	45.11	33	4.31×10^{-9}
	4.31×10^{-9}	46.21	44	4.31×10^{-9}
	4.31×10^{-9}	44.97	60	4.31×10^{-9}
	0	0.140	39	0
ACK	1.11×10^{-4}	37.70	33	3.94×10^{-5}
	3.05×10^{-5}	37.89	46	1.08×10^{-5}
	3.05×10^{-5}	37.95	68	1.08×10^{-5}
	8.88×10^{-16}	0.140	38	0

Continued on next page

Table B.1 – *Continued from previous page*

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
AXI	0	1.36	8	0
	0	1.36	20	0
	0	1.36	38	0
	0	0.140	42	0
BEC	5.04×10^{-8}	27.53	61	2.00×10^{-4}
	0	27.52	74	0
	0	27.12	94	0
BRA	0.857	6.95	21	-
	0.857	6.94	34	-
	0.398	11.27	105	-
	0.398	0.250	55	-
DEK	-24771	22.17	23	7.46×10^{-8}
	-24771	22.06	35	1.55×10^{-9}
	-24771	22.21	53	2.30×10^{-9}
	-24771	0.468	67	0
EAS	-6.86×10^{-5}	76.26	146	-
	-1	122.9	200	7.75×10^{-10}
	-6.86×10^{-5}	74.36	171	-
	-6.86×10^{-5}	0.172	37	-
EXP	0.368	1.23	5	0
	0.368	1.23	15	0
	0.368	1.24	23	0
	0.368	0.094	16	0
FOU	0.056	14.10	18	8.16×10^{-3}
	0.050	36.72	32	0.203
	0.053	10.70	70	0.0282
	0	0.109	33	0
GOL	92.0	6.19	19	-
	92.0	6.18	33	-
	3	10.39	145	1.17×10^{-10}
	264.5	0.156	40	-
GRI	0.055	8.76	20	-
	0.055	8.70	35	-
	0.055	8.68	59	-
	0.351	0.203	42	-
HOS	-1.942	5.35	18	-
	-2.346	8.60	51	0
	-2.346	8.63	91	0

Continued on next page

Table B.1 – *Continued from previous page*

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
	-2.346	0.125	30	0
LEV	0.125	9.28	19	-
	1.50×10^{-32}	11.83	54	-
	1.50×10^{-32}	11.88	100	-
	1.5×10^{-32}	0.125	35	-
LIN	-20	1.69	9	0
	-20	1.67	20	0
	-20	1.67	29	0
	-20	0.250	45	0
MIC	-0.801	13.58	25	-
	-0.801	22.03	63	-
	-0.801	22.13	91	-
	-0.801	0.140	41	-
NEU	-1.25	0.67	6	-
	-2	4.90	38	0
	-2	3.95	85	0
	-2	0.140	35	0
PER	1	7.85	8	-
	1	8.31	19	-
	1.11×10^{-18}	105.4	139	0
	1	0.109	17	-
RAS	0	1.04	7	0
	0	1.03	19	0
	0	1.01	37	0
	0	0.125	35	0
ROS	0.771	32.81	76	-
	0.771	30.64	93	-
	0.771	30.93	124	-
	0.771	0.172	44	-
ROT	0	1.71	9	0
	0	1.69	21	0
	0	1.69	39	0
	0	0.218	63	0
SAL	0.400	275.2	119	-
	0.500	58.06	59	-
	8.32×10^{-13}	65.99	149	8.32×10^{-12}
	0	0.140	38	0
SCA	0.037	6.35	21	-

Continued on next page

Table B.1 – *Continued from previous page*

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
	0.037	6.33	36	-
	0.037	6.32	67	-
	0.037	0.140	36	-
SCW	-7.88	16.54	17	-
	-67.6	15.78	39	-
	-67.6	15.85	77	-
	-126.8	0.406	47	-
SIN	-1.63	0.37	13	-
	-3.5	8.23	32	0
	-3.5	7.85	46	0
	-1.184	0.140	26	-
SPH	0	2.41	8	0
	0	1.64	20	0
	0	1.61	38	0
	0	0.156	42	0
STE	5	1.40	8	0
	5	1.40	18	0
	5	1.39	26	0
	5	0.156	32	0
SUM	0	0.68	6	0
	0	0.67	16	0
	0	0.67	30	0
	0	0.125	33	0
ZAK	6.89×10^{-5}	92.86	211	7.42×10^{-3}
	8.54×10^{-31}	90.03	224	8.27×10^{-16}
	8.54×10^{-31}	90.05	250	8.27×10^{-16}
	0	0.156	37	0

Table B.1: Results for $n = 2$

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
ABS	4.11×10^{-5}	188.0	69	3.34×10^{-5}
	2.02×10^{-6}	203.2	95	1.05×10^{-6}
	2.36×10^{-6}	69.27	174	1.24×10^{-5}
	0	0.281	97	0
ACK	5.83×10^{-6}	144.0	138	2.90×10^{-6}
	1.85×10^{-6}	170.1	152	8.24×10^{-7}
	1.85×10^{-6}	121.3	305	8.24×10^{-7}

Continued on next page

Table B.2 – *Continued from previous page*

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
	8.88×10^{-16}	0.312	94	0
AXI	0	1.376	12	0
	0	1.362	33	0
	0	1.367	97	0
	0	0.452	109	0
EXP	0.135	2.085	9	0
	0.135	2.110	27	0
	0.135	2.119	57	0
	0.135	0.140	31	0
FOU	0.275	1209	374	-
	0.024	4999	473	-
	0.085	2315	624	-
	0	0.250	76	0
GRI	7.40×10^{-3}	391.0	93	-
	7.40×10^{-3}	261.0	99	-
	7.40×10^{-3}	286.5	227	-
	0.140	0.577	126	-
LEV	0.125	68.89	57	-
	5.87×10^{-15}	128.9	155	-
	5.28×10^{-15}	89.07	306	-
	1.50×10^{-32}	0.250	81	-
LIN	-40	1.712	13	0
	-40	1.695	33	0
	-40	1.678	67	0
	-40	0.452	133	0
MIC	-2.144	487.0	80	-
	-2.158	557.9	461	-
	-2.158	535.8	411	-
	-2.158	0.359	132	-
NEU	-16	619.7	191	5.0×10^{-4}
	-16	182.9	75	0
	-16	437.7	215	0
	-15.33	0.593	153	-
PER	11.06	842.5	208	-
	0.049	4907	674	-
	0.049	2400	1340	-
	1.676	3.994	756	-
POW	2.64×10^{-12}	209.9	186	-

Continued on next page

Table B.2 – *Continued from previous page*

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
	7.81×10^{-13}	214.0	293	-
	3.28×10^{-4}	217.7	769	-
	0	0.265	99	0
RAS	0	0.993	11	0
	0	1.028	32	0
	0	0.999	96	0
	0	0.234	81	0
ROS	2.949	186.2	80	-
	2.100	1625	349	-
	2.100	185.5	265	-
	2.100	0.452	131	-
ROT	0	1.712	13	0
	0	1.911	35	0
	0	1.813	103	0
	0	0.827	196	0
SAL	0.600	283.0	92	-
	0.600	501.1	174	-
	0.400	472.9	376	-
	0.6	0.218	68	-
SCW	-15.75	1214	119	-
	-95.58	3231	173	-
	-372.0	11720	596	-
	-253.6	0.796	194	-
SIN	-1.634	0.433	36	-
	-3.5	46.31	206	0
	-3.5	61.41	272	0
	-0.746	0.218	65	-
SPH	0	1.972	12	0
	0	1.591	33	0
	0	1.631	97	0
	0	0.296	109	0
STE	10	21.71	14	0
	10	21.11	40	0
	10	13.50	86	0
	10	0.250	78	0
SUM	1.78×10^{-3}	21.94	56	-
	1.54×10^{-14}	21.95	97	2.43×10^{-5}
	1.54×10^{-14}	21.86	241	2.43×10^{-5}

Continued on next page

Table B.2 – *Continued from previous page*

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
	0	0.218	73	0
WOO	18.57	2128	267	-
	2.560	4559	796	-
	5.29×10^{-5}	4074	607	0
	41.08	0.749	235	-
ZAK	4.49×10^{-17}	1562	347	6.12×10^{-9}
	5.51×10^{-15}	1855	559	6.03×10^{-8}
	5.92×10^{-19}	780.8	501	6.77×10^{-10}
	8.322	0.437	152	-

Table B.2: Results for $n = 4$

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
ABS	0	2193	229	0
	8.05×10^{-6}	629.6	521	4.75×10^{-6}
	6.70×10^{-6}	998.7	365	2.47×10^{-6}
	0	0.499	174	0
ACK	1.502	1913	173	-
	1.502	3283	359	-
	1.502	1503	822	-
	8.88×10^{-16}	0.499	166	0
AXI	0	1.390	16	0
	0	1.364	45	0
	0	1.374	177	0
	0	0.499	202	0
EXP	0.050	2.965	13	0
	0.050	2.992	39	0
	0.050	2.975	105	0
	0.050	0.172	46	0
FOU	0.198	3190	70	-
	0.017	905.7	236	-
	0.038	2632	471	-
	0	0.343	120	0
GRI	7.40×10^{-3}	1125	115	-
	7.40×10^{-3}	3752	688	-
	7.40×10^{-3}	5476	650	-
	0.150	1.217	247	-

Continued on next page

Table B.3 – *Continued from previous page*

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
LEV	0.125	852.3	96	-
	1.54×10^{-13}	4180	659	-
	3.88×10^{-12}	790.5	1241	-
	1.50×10^{-32}	0.499	138	-
LIN	-60	1.748	17	0
	-60	1.705	46	0
	-60	1.663	121	0
	-60	1.045	264	0
MIC	-4.513	2071	302	-
	-4.513	1948	208	-
	-4.513	3962	729	-
	-4.513	0.499	171	-
NEU	-49.38	915.0	75	-
	-49.38	837.5	110	-
	-50	1835	422	0
	-46.38	1.435	520	-
PER	631.1	893.5	92	-
	2.410	1681	3033	-
	1.487	2664	11436	-
	2.09×10^{8}	0.936	242	-
RAS	0	1.077	15	0
	0	1.043	44	0
	0	1.068	176	0
	0	0.374	138	0
ROS	3.207	6728	424	-
	3.207	23729	622	-
	3.207	4472	618	-
	3.207	1.404	416	-
ROT	0	1.710	17	0
	0	1.707	49	0
	0	1.718	199	0
	0	1.279	397	0
SAL	0.900	26.09	41	-
	0.900	26.15	93	-
	0.400	5820	3181	-
	1.2	0.328	98	-
SCW	-23.63	10685	204	-
	-103.5	7711	232	-

Continued on next page

Table B.3 – *Continued from previous page*

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
	-379.9	32211	1044	-
	-380.4	2.153	386	-
SIN	-0.417	0.401	52	-
	-2.692	2375	159	-
	-2.692	841.9	403	-
	-0.596	0.406	133	-
SPH	0	2.537	16	0
	0	1.636	45	0
	0	1.618	177	0
	0	0.515	202	0
STE	15	34.57	18	0
	15	17.81	53	0
	15	18.88	146	0
	15	0.437	138	0
SUM	7.02×10^{-16}	9614	3444	1.61×10^{-4}
	3.56×10^{-17}	7659	3502	6.87×10^{-5}
	9.61×10^{-15}	2338	1543	3.07×10^{-4}
	0	0.343	120	0
ZAK	2.227	5075	642	-
	2.056	1760	460	-
	3.08×10^{-12}	11299	1484	8.68×10^{-7}
	37.85	1.747	454	-

Table B.3: Results for $n = 6$

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
ABS	3.30×10^{-3}	2059	430	2.41×10^{-3}
	1.52×10^{-5}	1925	668	8.12×10^{-6}
	7.83×10^{-6}	14872	1632	5.65×10^{-7}
	0	0.890	270	0
ACK	1.297	6763	1109	-
	6.92×10^{-6}	9042	449	2.71×10^{-6}
	7.70×10^{-6}	9030	950	4.19×10^{-6}
	8.88×10^{-6}	0.889	254	0
AXI	0	1.405	20	0
	0	1.412	53	0
	0	1.435	245	0

Continued on next page

Table B.4 – *Continued from previous page*

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
	0	0.811	321	0
EXP	0.0183	3.801	17	0
	0.0183	3.759	46	0
	0.0183	3.740	142	0
	0.0183	0.234	61	0
FOU	0.208	47922	589	-
	0.254	22063	466	-
	0.0221	46006	939	0.170
	0	0.546	195	0
GRI	1.63×10^{-13}	16615	1003	6.99×10^{-7}
	7.40×10^{-3}	2724	450	-
	7.40×10^{-3}	42950	2127	-
	0.143	2.636	385	-
LEV	3.48×10^{-7}	1568	125	-
	0.0895	4401	984	-
	3.70×10^{-13}	2011	771	-
	1.50×10^{-32}	0.811	206	-
LIN	-80	1.919	21	0
	-80	1.931	53	0
	-80	1.792	161	0
	-80	1.591	438	0
MIC	-5.075	5855	337	-
	-5.118	16617	3540	-
	-5.076	3440	1099	-
	-5.076	1.622	514	-
NEU	-112	63072	821	2.00×10^{-3}
	-112	19495	503	0
	-112	4980	1018	0
	-105	5.35	1198	-
PER	3.74×10^{10}	3398	173	-
	3.46×10^{9}	765.8	978	-
	466.8	1427	21429	-
	2.60×10^{10}	20.92	2233	-
RAS	0	1.225	19	0
	0	1.169	56	0
	0	1.079	280	0
	0	0.546	206	0
ROS	6.545	25207	501	-

Continued on next page

Table B.4 – *Continued from previous page*

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
	4.238	35914	1377	-
	4.238	11136	2116	-
	4.238	4.056	983	-
ROT	0	1.783	21	0
	0	1.767	57	0
	0	1.756	273	0
	0	2.3556	666	0
SAL	0.700	14192	697	-
	0.700	21006	1993	-
	0.800	45120	2091	-
	1.1	0.546	113	-
SCW	-31.50	5328	101	-
	-738.4	22551	553	-
	-644.2	35657	2056	-
	-507.23	3.9	639	-
SIN	-0.220	0.644	68	-
	-3.5	3195	567	0
	-3.5	4282	1133	0
	-0.49639	0.624	204	-
SPH	0	2.486	20	0
	0	1.763	53	0
	0	1.753	245	0
	0	0.84241	321	0
STE	20	22.72	22	0
	20	21.96	66	0
	20	95.59	222	0
	20	0.6708	212	0
SUM	1.19×10^{-10}	11073	3532	9.97×10^{-3}
	2.81×10^{-13}	25283	7141	2.04×10^{-3}
	0	7003	2030	0
	0	0.6084	174	0
ZAK	2.460	28640	1274	-
	3.741	11229	530	-
	2.029	39197	8268	-
	111.97	0.6708	208	-

Table B.4: Results for $n = 8$

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
ABS	7.80×10^{-5}	34653	1064	6.35×10^{-5}
	1.86×10^{-5}	42484	1612	6.91×10^{-6}
	4.83×10^{-6}	14570	771	1.27×10^{-6}
	0	1.170	385	0
ACK	1.155	20808	900	-
	1.155	32983	1058	-
	1.155	36233	2932	-
	8.88E-16	1.310	403	0
AXI	0	1.424	24	0
	0	1.408	61	0
	0	1.472	321	0
	0	1.373	466	0
EXP	6.74×10^{-3}	4.603	21	0
	6.74×10^{-3}	4.612	53	0
	6.74×10^{-3}	4.551	183	0
	6.74×10^{-3}	0.312	76	0
FOU	0.257	86048	1466	-
	0.072	37533	689	-
	0.075	71005	2000	-
	0	0.718	248	0
GRI	7.40×10^{-3}	1.74×10^{5}	2333	-
	7.40×10^{-3}	10679	1110	-
	7.40×10^{-3}	15487	1213	-
	0.155	4.571	558	-
LEV	8.05×10^{-9}	4536	188	-
	0.090	76594	2269	-
	0.090	9158	2265	-
	1.50×10^{-32}	0.936	285	-
LIN	-100	2.602	25	0
	-100	2.206	60	0
	-100	1.832	205	0
	-100	2.418	655	0
MIC	-4.353	39635	749	-
	-3.253	43289	989	-
	-4.336	45418	1393	-
	-5.175	2.590	774	-
NEU	-210	8562.2	538	2.00×10^{-3}
	-210	92908	757	0

Continued on next page

Table B.5 – *Continued from previous page*

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
	-209.4	7870	1108	-
	-200	18.44	2379	-
PER	2.25×10^{20}	194.7	23	-
	7.01×10^{15}	194.5	112	-
	1.92×10^{9}	23407	11306	-
	3.68×10^{18}	18.10	2446	-
RAS	0	1.255	23	0
	0	1.173	67	0
	0	1.082	397	0
	0	1.030	285	0
ROS	8.678	1.09×10^{5}	827	-
	5.246	42705	871	-
	5.246	36438	2169	-
	5.246	8.736	1599	0
ROT	0	1.789	25	0
	0	1.772	65	0
	0	1.840	355	0
	0	3.931	1003	0
SAL	1.100	33244	977	0
	0.400	27755	952	0
	0.500	1.21×10^{5}	5707	0
	1.3	0.624	156	-
SCW	-39.38	25658	232	-
	-39.38	2.36×10^{5}	1906	-
	-750.1	93995	1756	-
	-634.0	7.753	953	-
SIN	-0.220	0.935	100	-
	-3.5	54531	907	0
	-3.5	58579	1980	0
	-0.416	0.733	248	-
SPH	0	2.5689	24	0
	0	1.710	61	0
	0	1.757	321	0
	0	1.310	466	0
STE	25	49.09	26	0
	25	103.1	76	0
	25	62.78	296	0

Continued on next page

Table B.5 – *Continued from previous page*

problem	$f(x^*)$	time	# evals	$\|x^* - x^g\|_\infty$
	25	0.936	300	0
SUM	5.40×10^{-13}	37532	5764	7.42×10^{-3}
	5.36×10^{-13}	1.49×10^{5}	18834	6.81×10^{-3}
	8.90×10^{-13}	1.09×10^{5}	12366	6.79×10^{-3}
	0	0.718	235	0
ZAK	18.83	1.24×10^{5}	1106	-
	2.068	30405	430	-
	2.068	4.78×10^{5}	4597	-
	140.2	0.983	272	-

Table B.5: Results for $n = 10$

Appendix C

Test Problems

This appendix contains details of the test problems used in chapter 6. The algorithms used to generate random MIQP instances are given in section C.1. The details of the MINLP test problems are given in section C.2.

C.1 Method used to generate MIQPs

In this section we discuss the methods used to generate the non-convex MIQPs used in chapter 6. The generation of the MIQPs can be broken up into two stages; in the first stage the objective function is generated while in the second we generate the constraints. Obviously three objective function generation methods were required since methods were developed three different types of Hessians, as discussed in chapter 4. The required generation algorithms are given in section C.1.1. In section C.1.2 we give the algorithms used to generate the three types of constraints; bound, sparse and dense constraints. In the discussion in the following sections the random numbers were generated using the MATLAB function `rand(a,b)` which generates an $a \times b$ matrix containing random numbers drawn from the uniform distribution on the interval $(0, 1)$.

C.1.1 Generation of the objective function

The objective function of problem (1.2) is completely specified by giving H and g. For all three types of problems the elements of g were chosen uniformly at random from $(-1, 1)$. We now give the algorithms used to set H when H_{cc} is invertible, positive definite and singular. Hessians with H_{cc} invertible were generated using Algorithm 6. This algorithm is taken from [38] and generates symmetric matrices with a user defined percentage q of positive eigenvalues.

We note that it is very rare that the n_cth principal leading submatrix in step v is not invertible. The procedures used to generate H with H_{cc} positive definite and singular are given in Algorithm 7 and Algorithm 8 respectively. Both of these algorithms make use of Algorithm 6 to generate matrices with q positive eigenvalues. However, when we call Algorithm 6 in Algorithm 7 and Algorithm 8 we are in fact referring to Algorithm 6 without its final step.

Algorithm 6 Generation of Hessians with H_{cc} invertible

i. Set $q \in [0, 100]$. Let $\tilde{q} = \lfloor pn/100 \rfloor$.

ii. Choose n random numbers κ_i. Generate the first $\tilde{q}$ numbers uniformly at random from $(0, 1)$. Choose the remaining $(n - \tilde{q})$ numbers uniformly at random from $(-1, 0)$.

iii. Generate n random vectors with elements drawn uniformly at random from $(-1, 1)$. Orthonormalise these vectors to obtain the vectors l_i.

iv. The Hessian is given by the following equation

$$H = \sum_{i=1}^{n} \kappa_i l_i l_i^T. \tag{C.1}$$

v. Check whether the n_cth principal leading submatrix of H is invertible. If it is then return H otherwise go to step ii.

C.1.2 Generation of the constraints

In this section we give the algorithms used to generate the constraints on the MIQPs in chapter 6. Three types of constraints were used in chapter 6; bound constraints, sparse linear inequality constraints and dense linear inequality constraints. For bound constrained problems each variable was constrained to lie between minus two and two. The linear inequality constraints were generated in such a way that the original problem was always feasible and bounded. This was done by applying random linear transformations to bound constraints. The sparse linear inequality constraints were generated using Algorithm 9. The sparsity[1] of the constraint matrix gener-

[1]The sparsity of a matrix is a measure of the number of non-zero elements and is given by (number of non-zero elements)/(number of elements).

ated by Algorithm 9 decreases from 1 for $n = 2$ to 0.0667 for $n = 30$. The dense linear inequality constraints were generated using Algorithm 10. It is obvious that the constraints generated using these algorithms will always result in a feasible, bounded problem since they could be thought of as the result of transforming the constraints of a bound constrained problem with the variables constrained to lie between $-\beta$ and β.

C.2 MINLP test functions

In this section more details are given on the problems used to test the derivative free algorithm developed in chapter 5. The test functions were taken from [13, 52, 93, 52, 115, 161]. In the list below we give the name of the test function, the reference from which it was taken, the form of $f(x)$, the bounds used and x_0. Further details on these functions can be found in the relevant references. We note that when the function name is followed by the word 'shifted' the origin of the function has been moved such that the $x_g \in \mathbb{Z}^n$. This makes it possible to check whether the algorithms find the global minima.

- Absolute value function
  - $f(x) = \sum_{i=1}^{n} |x_i|$
  - $l_i = -10,\ u_i = 10, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 4, \quad \forall i = 1, \ldots, n$
- Ackley's function [13]
  - $f(x) = a \exp\left(b\sqrt{\sum_{i=1}^{n} \frac{x_i^2}{n}}\right) - \exp\left(\sum_{i=1}^{n} \frac{\cos(2\pi x_i)}{n}\right) + c,$
    where $a = -20$, $b = -0.2$ and $c = 20 + e^1$.
  - $l_i = -32,\ u_i = 32, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 3, \quad \forall i = 1, \ldots, n$
- Axis parallel hyper-ellipsoid function [161]
  - $f(x) = \sum_{i=1}^{n} i x_i^2$
  - $l_i = -5.12,\ u_i = 5.12, \quad \forall i = 1, \ldots, n$

- $(x_0)_i = 4, \quad \forall i = 1, \ldots, n$

- Becker and Lago problem [13]
  - $f(x) = (|x_1| - 5)^2 + (|x_2| - 5)^2$
  - $l_i = -10,\ u_i = 10, \quad i = 1, 2$
  - $(x_0)_i = 0, \quad i = 1, 2$

- Branin's function [13]
  - $f(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10$
  - $l = [-5,\ 0],\ u = [0,\ 15]$
  - $x_0 = [-3,\ 3]$

- Dekkers and Aarts problem [13]
  - $f(x) = 10^5 x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-5}(x_1^2 + x_2^2)^4$
  - $l_i = -20,\ u_i = 20, \quad i = 1, 2$
  - $(x_0)_i = 4, \quad i = 1, 2$

- Easom's function (shifted) [13]
  - $f(x) = -\cos(x_1)\cos(x_2)\, e^{-x_1^2 - x_2^2}$
  - $l_i = -100,\ u_i = 100, \quad i = 1, 2$
  - $(x_0)_i = 3, \quad i = 1, 2$

- Exponential problem [13]
  - $f(x) = \exp\left(-\frac{1}{2}\sum_{i=1}^{n} x_i^2\right)$
  - $l_i = -1,\ u_i = 1, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 1, \quad \forall i = 1, \ldots, n$

- Fourth function of De Jong [52]
  - $f(x) = \sum_{i=1}^{n} x_i^4 + \text{gaussian noise}$
  - $l_i =,\ u_i =, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 1, \quad \forall i = 1, \ldots, n$

- Goldstein-Price function [13]
  - $f(x) = \left[1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2\right)\right] \times \left[30 + (2x_1 - 3x_2)^2 \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right]$
  - $l_i = -2,\ u_i = 2, \quad i = 1, 2$
  - $(x_0)_i = 1, \quad i = 1, 2$
- Griewangk's function [13]
  - $f(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
  - $l_i = -600,\ u_i = 600, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 2, \quad \forall i = 1, \ldots, n$
- Hosaki problem [13]
  - $f(x) = \left(1 - 8x_1 + 7x_1^2 - \frac{7}{3}x_1^3 + \frac{1}{4}x_1^4\right) x_2^2 e^{-x_2}$
  - $l_i = [0\,0]\,,\ u_i = [5\,6]$
  - $x_0 = [3,\ 3]$
- Levy function [93]
  - $f(x) = \sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 \left[1 + 10\sin^2(\pi y_{i+1} + 1)\right] + (y_n - 1)^2 \left(1 + \sin(2\pi y_n)\right)^2$
    where $y_i = 1 + \frac{1}{4}(x_i - 1)$
  - $l_i = -10,\ u_i = 10, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 3, \quad \forall i = 1, \ldots, n$
- Linear function
  - $f(x) = \sum_{i=1}^{n} x_i$
  - $l_i = -10,\ u_i = 10, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 3, \quad \forall i = 1, \ldots, n$
- Michalemicz's function [161]

- $f(x) = -\sum_{i=1}^{n} \sin(x_i) \left[\sin\left(\frac{ix_i^2}{\pi}\right)\right]^{20}$
- $l_i = 0,\ u_i = \pi, \quad \forall i = 1, \ldots, n$
- $(x_0)_i = 1, \quad \forall i = 1, \ldots, n$

- Neumaier problem 3 [13]
  - $f(x) = \sum_{i=1}^{n} (x_i - 1)^2 - \sum_{i=2}^{n} x_i x_{i-1}$
  - $l_i = -n^2,\ u_i = n^2, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 1, \quad \forall i = 1, \ldots, n$

- Perm function [161]
  - $f(x) = \sum_{i=1}^{n} \left\{ \sum_{j=1}^{n} \left[j^{\,i} + \frac{1}{2} \right] \left[\left(\frac{x_j}{j} \right)^{i} - 1 \right] \right\}^2$
  - $l_i = -n,\ u_i = n, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 0, \quad \forall i = 1, \ldots, n$

- Powell's quadratic problem [13]
  - $f(x) = 121 * x_1^2 + 5\,(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10\,(x_1 - x_4)^4$
  - $l_i = -10,\ u_i = 10, \quad i = 1, 2, 3, 4$
  - $(x_0)_i = 1, \quad \forall i = 1, 2, 3, 4$

- Rastrigin's function [13]
  - $f(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) \right]$
  - $l_i = -5.12,\ u_i = 5.12, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 2, \quad \forall i = 1, \ldots, n$

- Rosenbrock's valley [13]
  - $f(x) = 100 \sum_{i=1}^{n-1} \left[\left(x_{i+1} - x_i^2 \right)^2 + (1 - x_i)^2 \right]$
  - $l_i = -2.048,\ u_i = 2.048, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 0, \quad \forall i = 1, \ldots, n$

- Rotated hyper ellipsoid function [115]
  - $f(x) = \sum_{i=1}^{n} \sum_{j=1}^{i} x_j^2$
  - $l_i = -65.53,\ u_i = 65.53, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 15, \quad \forall i = 1, \ldots, n$
- Salomon problem [13]
  - $f(x) = 1 - \cos\left(2\pi \|x\|\right) + 0.1\|x\|$
  - $l_i = -100,\ u_i = 100, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 4, \quad \forall i = 1, \ldots, n$
- Schaffer problem 1 [13]
  - $f(x) = \frac{1}{2} + \frac{2\left(\sin\sqrt{x_1^2 + x_2^2}\right)^2 - 1}{2\left(1 + 0.001\left(x_1^2 + x_2^2\right)\right)^2}$
  - $l_i = -100,\ u_i = 100, \quad i = 1, 2$
  - $(x_0)_i = 4, \quad i = 1, 2$
- Schwefel's function (shifted) [13]
  - $f(x) = -\sum_{i=1}^{n} \left(x_i - 0.0313\right) \sin\left(\sqrt{|x_i - 0.0313|}\right)$
  - $l_i = -500,\ u_i = 500, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 2, \quad \forall i = 1, \ldots, n$
- Sinusoidal problem (shifted) [13]
  - $f(x) = -2.5 \prod_{i=1}^{n} \sin\left(x_i - 2 + \frac{\pi}{2}\right) - \prod_{i=1}^{n} \sin\left(5\left(x_i - 2 + \frac{\pi}{2}\right)\right)$
  - $l_i = 0,\ u_i = \frac{\pi}{2}, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 0, \quad \forall i = 1, \ldots, n$
- Sphere function [161]
  - $f(x) = \sum_{i=1}^{n} x_i^2$

- $l_i = -5.12,\ u_i = 5.12, \quad \forall i = 1, \ldots, n$
- $(x_0)_i = 4, \quad \forall i = 1, \ldots, n$

- Step function [52]
  - $f(x) = 6n + \sum_{i=1}^{n} \lfloor x_i \rfloor$
  - $l_i = -3.12,\ u_i = 3.12, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 1, \quad \forall i = 1, \ldots, n$

- Sum of different powers function [115]
  - $f(x) = \sum_{i=1}^{n} |x_i|^{i+1}$
  - $l_i = -1,\ u_i = 1, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 1, \quad \forall i = 1, \ldots, n$

- Wood's function [13]
  - $f(x) = 100\left(x_1^2 - x_2\right)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90\left(x_3^2 - x_4^2\right)^2 + 10.1\left((x_2 - 1)^2 + (x_4 - 1)^2\right) + 19.8\,(x_2 - 1)\,(x_4 - 1)$
  - $l_i = -10,\ u_i = 10, \quad i = 1, 2, 3, 4$
  - $(x_0)_i = 3, \quad i = 1, 2, 3, 4$

- Zakharov function [161]
  - $f(x) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} \frac{x_i}{2i}\right)^2 + \left(\sum_{i=1}^{n} \frac{x_i}{2i}\right)^4$
  - $l_i = -5,\ u_i = 10, \quad \forall i = 1, \ldots, n$
  - $(x_0)_i = 3, \quad \forall i = 1, \ldots, n$

Algorithm 7 Generation of Hessians with H_{cc} positive definite

1. Generate an $n \times n$ matrix $\tilde{H}$ with $q = 50\%$ using Algorithm 6.
2. Generate an $n_c \times n_c$ matrix H_{cc} with $q = 100\%$ using Algorithm 6.
3. Generate H by replacing the n_cth principal leading submatrix of $\tilde{H}$ with H_{cc}.
4. If $H \succeq 0$ discard H and go to step 1, otherwise return H.

Algorithm 8 Generation of Hessians with H_{cc} singular

i. Generate an $n \times n$ matrix $\tilde{H}$ with $q = 50\%$ using Algorithm 6.

ii. Generate an $n_c \times n_c$ matrix H_{cc} using the procedure described above with $q = 50\%$ using Algorithm 6. However in step ii we replace r of the random numbers κ_i with 0. Here r is a randomly generated integer between 1 and n_c.

iii. H_{cc} generated in step ii is singular. Let H be the matrix obtained when n_cth principal leading submatrix of $\tilde{H}$ is replaced by H_{cc}.

Algorithm 9 Generation of sparse linear inequality constraints

i. Generate a random 2×2 matrix a with elements drawn uniformly at random from $(0, 2)$.

ii. If n is even generate an $n \times n$ matrix $A1$ by making each 2×2 block diagonal element a and setting all other elements to zero. If n is odd generate an $(n+1) \times (n+1)$ matrix $\widetilde{A1}$ by making each 2×2 block diagonal element a and setting all other elements to zero. Form $A1$ from $\widetilde{A1}$ by removing the $(n+1)$th row and $(n+1)$th column from $\widetilde{A1}$.

iii. Let A be the following $2n \times n$ matrix

$$A = \begin{bmatrix} A1 \\ -A1 \end{bmatrix}.$$

iv. Generate a random $n \times 1$ vector β with integer elements between 1 and 3 using the MATLAB function `randi(5,n,1)`.

v. Let b be the following $2n \times 1$ vector

$$b = \begin{bmatrix} \beta \\ \beta \end{bmatrix}.$$

Algorithm 10 Generation of dense linear inequality constraints

i. Generate a random $n \times n$ matrix $A1$ with elements drawn uniformly at random from $(0, 1)$.

ii. Let A be the following $2n \times n$ matrix

$$A = \begin{bmatrix} A1 \\ -A1 \end{bmatrix}.$$

iii. Generate a random $n \times 1$ vector β with integer elements between 1 and 3 using the MATLAB function `randi(5,n,1)`.

iv. Let b be the following $2n \times 1$ vector

$$b = \begin{bmatrix} \beta \\ \beta \end{bmatrix}.$$

Bibliography

[1] Abramson, M., Audet, C., Chrissis, J., and Walston, J. (2009). Mesh adaptive direct search algorithms for mixed variable optimization. *Optimization Letters*, 3(1):35–47.

[2] Abramson, M., Audet, C., and Dennis, Jr., J. E. (2007). Filter pattern search algorithms for mixed variable constrained optimization problems. *Pacific Journal of Optimization*, 3(3):477–500.

[3] Abramson, M. A. (2002). *Pattern Search Algorithms for Mixed Variable General Constrained Optimization Problems*. PhD thesis, Rice University.

[4] Abramson, M. A. (2007). NOMADm version 4.6 user's guide. Technical report, Department of Mathematics and Statistics, Air Force Institute of Technology.

[5] Abramson, M. A. and Audet, C. (2006). Second order convergence in mesh adaptive direct search. *SIAM Journal on Optimization*, 17(2):606–619.

[6] Abramson, M. A., Audet, C., Couture, G., Dennis, Jr., J. E., and Le Digabel, S. (2012). The NOMAD project. Software available at `http://www.gerad.ca/nomad`.

[7] Achterberg, T. (2007). *Constraint Integer Programming*. PhD thesis, Technischen Universität Berlin.

[8] Adjiman, C. S., Androulakis, I. P., and Floudas, C. A. (1997). Global optimization of MINLP problems in process synthesis and design. *Computers & Chemical Engineering*, 21:S445–S450.

[9] Agrawal, S. C. (1974). On mixed integer quadratic programs. *Naval Research Logistics Quarterly*, 21:289–297.

[10] Ahn, C. W. (2006). *Advances in Evolutionary Algorithms: Theory, Design and Practice*. Springer–Verlag, first edition.

[11] Al-Khayyal, F. A. and Larsen, C. (1990). Global optimization of a quadratic function subject to a bounded mixed integer constraint set. *Annals of Operations Reserach*, 25(1):169–180.

[12] Alexandrov, N., Dennis, Jr., J. E., Lewis, R. M., and Torczon, V. (1998). A trust region framework for managing the use of approximation models in optimization. *Structural Optimization*, 15(1):16–12.

[13] Ali, M. M., Khompatraporn, C., and Zabinsky, Z. B. (2005). A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization*, 31(4):635–672.

[14] Androulakis, I. P., Maranas, C. D., and Floudas, C. A. (1995). αBB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363.

[15] Anton, H. and Rorres, C. (2005). *Elementary Linear Algebra*. John Wiley & Sons, ninth edition.

[16] Audet, C. (2004). Convergence results for pattern search algorithms are tight. *Optimization and Engineering*, 5(2):101–122.

[17] Audet, C. and Dennis, Jr., J. E. (2001). Pattern search algorithms for mixed variable programming. *SIAM Journal on Optimization*, 11(3):573–594.

[18] Audet, C. and Dennis, Jr., J. E. (2004). A pattern search filter method for nonlinear programming without derivatives. *SIAM Journal on Optimization*, 14(4):980–1010.

[19] Audet, C. and Dennis, Jr., J. E. (2006). Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(2):188–217.

[20] Audet, C. and Dennis, Jr., J. E. (2009). A progressive barrier for derivative-free nonlinear programming. *SIAM Journal on Optimization*, 20(1):445–472.

[21] Axehill, D. (2005). *Applications of Integer Quadratic Programming in Control and Communication*. PhD thesis, Linköpings Universitet.

[22] Axehill, D. and Hansson, A. (2004). A preprocessing algorithm for MIQP solvers with applications to MPC. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 3, pages 2497–2502.

[23] Axehill, D., Vandenberghe, L., and Hansson, A. (2010). Convex relaxations for mixed integer predictive control. *Automatica*, 46(9):1540 – 1545.

[24] Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, first edition.

[25] Bäck, T., Rudolph, G., and Schwefel, H. P. (1993). Evolutionary programming and evolution strategies: Similarities and differences. In Fogel, D. B. and Atmar, J. W., editors, *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 11–22. Evolutionary Programming Society.

[26] Banks, A., Vincent, J., and Anyakoha, C. (2007a). A review of particle swarm optimization. Part I: Background and development. *Natural Computing*, 6(4):467–484.

[27] Banks, A., Vincent, J., and Anyakoha, C. (2007b). A review of particle swarm optimization. Part II: Hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing*, 7(1):109–124.

[28] Belotti, P., Lee, J., Liberti, L., Margot, F., and Wachter, A. (2009). Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4-5):597–634.

[29] Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252.

[30] Berthold, T., Heinz, S., and Vigerske, S. (2012). Extending a CIP framework to solve MIQCPs. In Lee, J. and Leyffer, S., editors, *Mixed Integer Nonlinear Programming*, pages 427–444. Springer.

[31] Bienstock, D. (1996). Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming*, 74(2):121–140.

[32] Bilbro, G. L. and Snyder, W. E. (1991). Optimization of functions with many minima. *IEEE Transactions on Systems, Man and Cybernetics*, 21(4):840–849.

[33] Billionnet, A., Elloumi, S., and Lambert, A. (2012). Extending the QCR method to general mixed-integer programs. *Mathematical Programming*, 131(1-2):381–401.

[34] Bonami, P., Kilinç, M., and Linderoth, J. (2009). Algorithms and software for convex mixed integer nonlinear programs. Technical Report 1664, Computer Sciences Department, University of Wisconsin-Madison.

[35] Booker, A. J., Dennis, Jr., J. E., Frank, P. D., Serafini, D. B., Torczon, V., and Trosset, M. W. (1999). A rigorous framework for optimization of expensive functions by surrogates. *Structural Optimization*, 17(1):1–13.

[36] Borchers, B. and Mitchell, J. E. (1994). An improved branch and bound algorithm for mixed integer nonlinear programming. *Computers and Operations Research*, 21(4):359–367.

[37] Buchheim, C., Caprara, A., and Lodi, A. (2010). An effective branch-and-bound algorithm for convex quadratic integer programming. In Eisenbrand, F. and Shepherd, F., editors, *Integer Programming and Combinatorial Optimization*, volume 6080 of *Lecture Notes in Computer Science*, pages 285–298.

[38] Buchheim, C. and Wiegele, A. (2010). Using semidefinite programming for solving non-convex mixed-integer quadratic problems. In *Proceedings of the European Workshop on Mixed Integer Nonlinear Programming*.

[39] Bueno, L. F., Friedlander, A., and Martínez, J. M. (2011). Inexact restoration method for derivative-free optimization with smooth constraints. Technical report, Optimization Online.

[40] Burer, S. (2009). On the copositive representation of binary and continuous nonconvex quadratic programs. *Mathematical Programming*, 120(2):479–495.

[41] Burer, S. and Saxena, A. (2012). The MILP road to MIQCP. In Lee, J. and Leyffer, S., editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 373–405.

[42] Cardoso, M. F., Salcedo, R. L., de Azevedo, S. F., and D., B. (1997). A simulated annealing approach to the solution of MINLP problems. *Computers & Chemical Engineering*, 21(12):1349–1364.

[43] Clarke, F. H. (1983). *Optimization and Nonsmooth Analysis*. John Wiley & Sons, first edition.

[44] Conn, A. R., Gould, N. I. M., and Toint, P. L. (1996). Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization. *Mathematical Programming*, 73(1):73–110.

[45] Conn, A. R., Scheinberg, K., and Toint, P. L. (1997). On the convergence of derivative-free methods for unconstrained optimization. In Iserles, A. and Buhmann, M., editors, *Approximation Theory and Optimization: Tributes to M. J. D. Powell*, pages 83–108. Cambridge University Press.

[46] Conn, A. R., Scheinberg, K., and Vincente, L. N. (2009). *Introduction to Derivative-Free Optimization*. MPS-SIAM, first edition.

[47] Costa, L. and Oliveira, P. (2001). Evolutionary algorithms approach to the solution of mixed integer non-linear programming problems. *Computers & Chemical Engineering*, 25(2-3):257–266.

[48] Currie, J. and Wilson, D. I. (2012). OPTI: Lowering the barrier between open source optimizers and the industrial MATLAB user. In *Proceedings of Foundations of Computer-Aided Process Operations*.

[49] Czyzyk, J., Mesnier, M. P., and More, J. J. (1998). The NEOS server. *IEEE Comput. Sci. Eng.*, 5(3):68–75.

[50] Dakin, R. J. (1965). A tree-search algorithm for mixed integer programming problems. *Computer Journal*, 8(3):250–255.

[51] Deep, K., Singh, K. P., Kansal, M. L., and Mohan, C. (2009). A real coded genetic algorithm for solving integer and mixed integer optimization problems. *Applied Mathematics and Computation*, 212(2):505–518.

[52] DeJong, K. A. (1975). *An Analysis of the Behavior of a class of Genetic Adaptive Systems*. PhD thesis, University of Michigan.

[53] Diniz-Ehrhardt, M. A., Martínez, J. M., and Pedroso, L. G. (2010). Derivative-free methods for nonlinear programming with general lower-level constraints. Technical report, Optimization Online.

[54] Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213.

[55] Duran, M. A. and Grossmann, I. E. (1986). An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36(3):307–339.

[56] Fletcher, R. and Leyffer, S. (1994). Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66(1-3):327–349.

[57] Fletcher, R. and Leyffer, S. (1998). Numerical experience with lower bounds for MIQP branch-and-bound. *SIAM Journal on Optimization*, 8(2):604–616.

[58] Flippo, O. E. and Rinnoy-Kan, A. H. G. (1993). Decomposition in general mathematical programming. *Mathematical Programming*, 60(1-3):361–382.

[59] Forrester, A. I. J. and Keane, A. J. (2009). Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45(1–3):50–79.

[60] Frangioni, A. and Gentile, C. (2007). SDP diagonalizations and perspective cuts for a class of nonseparable MIQP. *Operations Research Letters*, 35(2):181–185.

[61] Galli, L., Kaparis, K., and Letchford, A. N. (2011). Gap inequalities for non-convex mixed-integer quadratic programs. *Operations Research Letters*, 39(5):297 – 300.

[62] GAMS (2011). CPLEX 12. User manual.

[63] Gelfand, S. B. and Mitter, S. K. (1993). Metropolis-type annealing algorithms for global optimization. *SIAM Journal on Control and Optimization*, 31(1):111–131.

[64] Geoffrion, A. M. (1972). Generalized benders decompositions. *Journal of Optimization Theory and Applications*, 10(4):237–260.

[65] Gilmore, P. and Kelley, C. T. (1995). An implicit filtering algorithm for optimization of functions with many local minima. *SIAM Journal on Optimization*, 5(2):269–285.

[66] Gropp, W. and Moré, J. J. (1997). Optimization environments and the NEOS server. In Buhmann, M. D. and Iserles, A., editors, *Approximation Theory and Optimization*, pages 167–182. Cambridge University Press.

[67] Grossmann, I. E. (2002). Review of nonlinear mixed-integer and disjunctive programming techniques. *Optimization and Engineering*, 3(3):227–252.

[68] Gumma, E. A. E. (2010). *Derivative–Free Algorithm for Linearly Constrained Optimization Problems*. PhD thesis, University of Khartoum.

[69] Gupta, O. K. and Ravindran, A. (1985). Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31(12):1533–1546.

[70] Hajek, B. (1988). Cooling schedules for optimal annealing. *Mathematics of Operations Research*, 13(2):311–329.

[71] Hemker, T. (2008). *Derivative Free Surrogate Optimization for Mixed-Integer Nonlinear Black Box Problems in Engineering*. PhD thesis, Technischen Universität Darmstadt.

[72] Holmström, K., Quttineh, N.-H., and Edvall, M. (2008). An adaptive radial basis algorithm (ARBF) for expensive black-box mixed-integer constrained global optimization. *Optimization and Engineering*, 9(4):311–339.

[73] Hooke, R. and Jeeves, T. A. (1961). 'Direct search' solution of numerical and statistical problems. *Journal of the Association for Computing Machinery*, 8(2):212–229.

[74] Horn, R. A. and Johnson, C. R. (1985). *Matrix Analysis*. Cambridge University Press, first edition.

[75] Horst, R. and Tuy, H. (1993). *Global Optimization: Deterministic Approaches*. Springer-Verlag, second edition.

[76] Ingber, L. (1993). Simulated annealing: Practice versus theory. *Mathematical Com-puter Modelling*, 18(11):29–57.

[77] Jiang, M., Luo, Y. P., and Yang, S. Y. (2007). Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm. *Information Processing Letters*, 102(1):8–16.

[78] Kelley, C. T. (1999). Detection and remediation of stagnation in the Nelder-Mead algorithm using a sufficient decrease condition. *SIAM Journal on Optimization*, 10(1):43–55.

[79] Kelley, Jr., J. E. (1960). The cutting plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712.

[80] Kesavan, P. and Barton, P. I. (2000). Generalized branch-and-cut framework for mixed-integer nonlinear optimization. *Computers & Chemical Engineering*, 24(2-7):1361–1366.

[81] Kim, S. and Kojima, M. (2001). Second order cone programming relaxation of nonconvex quadratic optimization problems. *Optimization Methods & Software*, 15(3-4):201–224.

[82] Kim, S. and Kojima, M. (2002). Exact solutions of some nonconvex quadratic optimization problems via SDP and SOCP relaxations. *Computational Optimization and Applications*, 26(2):143154.

[83] Kingdon, J. (1992). Genetic algorithms: Deception, convergence and starting conditions. Technical Report RN/92/2, Department of Computer Science, University College London.

[84] Kitayama, S., Arakawa, M., and Yamazaki, K. (2006). Penalty function approach for the mixed discrete nonlinear problems by particle swarm optimization. *Structural and Multidisciplinary Optimization*, 32(3):191–202.

[85] Kojima, M. and Tunçel, L. (2000). Cones of matrices and successive convex relaxations of nonconvex sets. *SIAM Journal on Optimization*, 10(3):750–778.

[86] Kuipers, K. (2003). bnb20. `http://www.mathworks.com/matlabcentral/fileexchange/95`.

[87] Laurent, M. and Poljak, S. (1996). Gap inequalities for the cut polytope. *European Journal of Combinatorics*, 17(23):233–254.

[88] Lazimy, R. (1982). Mixed-integer quadratic programming. *Mathematical Programming*, 22(1):332–349.

[89] Lazimy, R. (1985). Improved algorithm for mixed-integer quadratic programs and a computational study. *Mathematical Programming*, 32(1):100–113.

[90] Le Digabel, S. (2011). Algorithm 909: NOMAD: Nonlinear optimization with the MADS algorithm. *ACM Transactions on Mathematical Software*, 37(4):1–15.

[91] Lee, S. and Grossmann, I. E. (2000). New algorithms for nonlinear generalized disjunctive programming. *Computers & Chemical Engineering*, 24(9-10):2125–2141.

[92] Lee, S. and Grossmann, I. E. (2001). A global optimization algorithm for nonconvex generalized disjunctive programming and applications to process systems. *Computers & Chemical Engineering*, 25(11-12):1675–1697.

[93] Levy, A. V. and Montalvo, A. (1985). The tunneling algorithm for the global optimization of functions. *SIAM Journal of Scientific and Statistical Computing*, 6(1):15–29.

[94] Lewis, R. M. and Torczon, V. (1999). Pattern search algorithms for bound constrained minimization. *SIAM Journal on Optimization*, 9(4):1082–1099.

[95] Lewis, R. M. and Torczon, V. (2000). Pattern search algorithms for linearly constrained minimization. *SIAM Journal on Optimization*, 10(3):917–941.

[96] Lewis, R. M. and Torczon, V. (2002). A globally convergent augmented lagrangian pattern search algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Optimization*, 12(4):1075–1089.

[97] Leyffer, S. (1993). *Deterministic Methods for Mixed Integer Nonlinear Programming*. PhD thesis, University of Dundee.

[98] Leyffer, S. (2001). Integrating SQP and branch-and-bound for mixed integer nonlinear programming. *Computational Optimization and Applications*, 18(3):295–309.

[99] Li, D. and Sun, X. (2006). *Nonlinear Integer Programming*. Springer–Verlag, first edition.

[100] Lin, Y.-C., Hwang, K.-S., and Wang, F.-S. (2004). A mixed-coding scheme of evolutionary algorithms to solve mixed-integer nonlinear programming problems. *Computers & Mathematics with Applications*, 47(8-9):1295–1307.

[101] Linderoth, J. (2005). A simplicial branch-and-bound algorithm for solving quadratically constrained quadratic programs. *Mathematical Programming*, 103(2):251–282.

[102] Liuzzi, G., Lucidi, S., and Rinaldi, F. (2011). Derivative-free methods for bound constrained mixed-integer optimization. *Computational Optimization and Applications*. DOI:10.1007/s10589-011-9405-3.

[103] Locatelli, M. (1996). Convergence properties of simulated annealing algorithms for continuous global optimization. *Journal of Applied Probabilty*, 33(4):1127–1140.

[104] Lovász, L. and Schrijver, A. (1991). Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1(2):166–190.

[105] Lucidi, S., Piccialli, V., and Sciandrone, M. (2005). An algorithm model for mixed variable programming. *SIAM Journal on Optimization*, 15(4):1057–1084.

[106] Lucidi, S. and Sciandrone, M. (2002). A derivative-free algorithm for bound constrained optimization. *Computational Optimization and Applications*, 21(2):119–142.

[107] Maranas, C. D. and Floudas, C. A. (1995). Finding all solutions of nonlinearly constrained systems of equations. *Journal of Global Optimization*, 7(2):143–182.

[108] Marazzi, M. and Nocedal, J. (2002). Wedge trust region methods for derivative free optimization. *Mathematical Programming*, 91(2):289–305.

[109] Marti, K. (2008). *Stochastic Optimtization Methods*. Springer–Verlag, second edition.

[110] McCormick, G. P. (1983). *Nonlinear Programming: Theory, Algorithms and Applications*. John Wiley & Sons, first edition.

[111] McKinnon, K. I. M. (1998). Convergence of the Nelder-Mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158.

[112] Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer–Verlag, third edition.

[113] Misener, R. and Floudas, C. (2012a). Global optimization of mixed-integer quadratically-constrained quadratic programs (MIQCQP) through piecewise-linear and edge-concave relaxations. *Mathematical Programming*, pages 1–28. 10.1007/s10107-012-0555-6.

[114] Misener, R. and Floudas, C. (2012b). GloMIQO: Global mixed-integer quadratic optimizer. *Journal of Global Optimization*, pages 1–48. 10.1007/s10898-012-9874-7.

[115] Molga, M. and Smutnicki, C. (2005). Test functions for optimization needs. Technical report, Zakład Systemów Dyskretnych.

[116] Müller, J., Shoemaker, C. A., and Piché, R. (2012). SO-MI: A surrogate model algorithm for computationally expensive nonlinear mixed-integer black-box global optimization problems. *Computers and Operations Research*, pages 1–18. http://dx.doi.org/10.1016/j.cor.2012.08.022.

[117] Nash, S. H. and Nocedal, J. (1991). A numerical study of the limited memory bfgs method and the truncated newton method for large scale optimization. *SIAM Journal on Oprimization*, 1(3):358–372.

[118] Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 7(4):308–313.

[119] Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer–Verlag, second edition.

[120] Piccialli, V. (2003). *Methods for solving Mixed Variable Programming Problems*. PhD thesis, University of Rome.

[121] Poli, R., Kennedy, J., and Blackwell, T. (2007). Particle swarm optimization: An overview. *Swarm Intelligence*, 1(1):33–57.

[122] Pörn, R., Harjunkoski, I., and Westerlund, T. (1999). Convexification of different classes of non-convex MINLP problems. *Computers & Chemical Engineering*, 23(3):439–448.

[123] Powell, M. J. D. (2002). Least Frobenius norm updating of quadratic models that satisfy interpolation conditions. Technical Report DAMTP 2002/NA08, Department of Applied Mathematics and Theoretical Physics, University of Cambridge.

[124] Powell, M. J. D. (2004a). The NEWUOA software for unconstrained optimization without derivatives. Technical Report DAMTP 2004/NA05, Department of Applied Mathematics and Theoretical Physics, University of Cambridge.

[125] Powell, M. J. D. (2004b). On updating the inverse of a KKT matrix. Technical Report DAMTP 2004/NA01, Department of Applied Mathematics and Theoretical Physics, University of Cambridge.

[126] Powell, M. J. D. (2009). The BOBYQA algorithm for bound constrained optimization without derivatives. Technical Report DAMTP 2009/NA06, Department of Applied Mathematics and Theoretical Physics, University of Cambridge.

[127] Price, C. J., Coope, I. D., and Byatt, D. (2002). A convergent variant of the Nelder-Mead algorithm. *Journal of Optimization Theory and Applications*, 113(1):5–19.

[128] Queipo, N., Haftka, R., Shyy, W., Goel, T., Vaidyanathan, R., and Tucker, P. K. (2005). Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41(1):1–28.

[129] Quesada, I. and Grossmann, I. E. (1992). An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. *Computers & Chemical Engineering*, 16(10-11):937–947.

[130] Rios, L. M. and Sahinidis, N. V. (2012). Derivative-free optimization: A review of algorithms and comparison of software implementations. *Journal of Global Optimization*, pages 1–47. 10.1007/s10898-012-9951-y.

[131] Romeijn, H. E., Zabinsky, Z. B., Graesser, D. L., and Neogi, S. (1999). New reflection generator for simulated annealing in mixed-integer/continuous global optimization. *Journal of Optimization Theory and Applications*, 101(2):403–427.

[132] Rudolph, G. (1994). Convergence analysis of canaonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101.

[133] Runtenbar, R. A. (1989). Simulated annealing algorithms: An overview. *IEEE Circuits and Devices Magazine*, 5(1):19–26.

[134] Ryoo, H. S. and Sahinidis, N. V. (1996). A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–139.

[135] Sahinidis, N. V. (1996). BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8(2):201–205.

[136] Saxena, A., Bonami, P., and Lee, J. (2010). Convex relaxations of non-convex mixed integer quadratically constrained programs: extended formulations. *Mathematical Programming*, 124(1):383–411.

[137] Saxena, A., Bonami, P., and Lee, J. (2011). Convex relaxations of non-convex mixed integer quadratically constrained programs: projected formulations. *Mathematical Programming*, 130(2):359–413.

[138] Sherali, H. D. and Adams, W. P. (1999). *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer Academic Publishers, first edition.

[139] Shimai, Y., Tani, J., Noguchi, H., Kawaguchi, H., and Yoshimoto, M. (2009). FPGA implementation of mixed integer quadratic programming solver for mobile robot control. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 447–450.

[140] Søndergaard, J. (2003). *Optimization Using Surrogate Models: by the Space Mapping Technique*. PhD thesis, Technical University of Denmark.

[141] Sriver, T. A., Chrissis, J. W., and Abramson, M. A. (2009). Pattern search ranking and selection algorithms for mixed variable simulation-based optimization. *European Journal of Operational Research*, 198(3):878–890.

[142] Stubbs, R. A. and Mehrotra, S. (1999). A branch-and-cut method for 0-1 mixed convex programming. *Mathematical Programming*, 86(3):515–532.

[143] Sturm, J. F. (2001). Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. Technical report, Department of Econometrics, Tilburg University.

[144] Szu, H. and Hartley, R. (1987). Fast simulated annealing. *Physics Letters A*, 122(3-4):157–162.

[145] Talbi, E. (2009). *Metaheuristics: From Design to Implementation*. John Wiley & Sons, first edition.

[146] Tawarmalani, M. (2001). *Mixed Integer Nonlinear Programs: Theory, Algorithms and Applications*. PhD thesis, University of Illinois.

[147] Tawarmalani, M. and Sahinidis, N. V. (2004). Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99(3):563–591.

[148] Tawarmalani, M. and Sahinidis, N. V. (2005). A polyhedral branch-and-cut approach to global optimization. *Mathematical programming*, 103(2):225–249.

[149] Torczon, V. (1997). On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7(1):1–25.

[150] Trelea, I. C. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317–325.

[151] Trosset, M. W. (2002). What is simulated annealing? *Optimization and Engineering*, 3(2):201–213.

[152] Tseng, P. (1999). Fortified-descent simplicial search method: A general approach. *SIAM Journal on Optimization*, 10(1):269–288.

[153] Türkay, M. and Grossmann, I. E. (1996). Logic-based MINLP algorithms for the optimal synthesis of process networks. *Computers & Chemical Engineering*, 20(8):959–978.

[154] van den Bergh, F. (2001). *An Analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria.

[155] Vecchietti, A. and Grossmann, I. E. (1997). LOGMIP: a disjunctive 0-1 nonlinear optimizer for process systems models. *Computers & Chemical Engineering*, 21:S427–S432.

[156] Wah, B., Chen, Y., and Wang, T. (2007). Simulated annealing with asymptotic convergence for nonlinear constrained optimization. *Journal of Global Optimization*, 39(1):1–37.

[157] Wang, G. G. and Shan, S. (2007). Review of metamodeling techniques in support of engineering design optimization. *Journal of Mechanical Design*, 129(4):370–380.

[158] Westerlund, T. and Pettersson, F. (1995). An extended cutting plane method for solving convex MINLP problems. *Computers & Chemical Engineering*, 19(S1):131–136.

[159] Westerlund, T. and Pörn, R. (2002). Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. *Optimization and Engineering*, 3(3):253–280.

[160] Wolsey, L. A. (1998). *Integer Programming*. John Wiley & Sons, first edition.

[161] Yang, C.-S. (2010). Test problems in optimization. In Yang, X.-S., editor, *Engineering Optimization: An Introduction with Metaheuristic Applications*. John Wiley & Sons.

[162] Yiqing, L., Xigang, Y., and Yongjian, L. (2007). An improved PSO algorithm for solving non-convex NLP/MINLP problems with equality constraints. *Computers & Chemical Engineering*, 31(3):153–162.

[163] Yokota, T., Gen, M., and Li, Y.-X. (1996). Genetic algorithm for nonlinear mixed integer programming problems and its applications. *Computers & Industrial Engineering*, 30(4):905–917.

[164] Yu, X. and Gen, M. (2010). *Introduction to Evolutionary Algorithms*. Springer–Verlag, first edition.

[165] Zamora, J. M. and Grossmann, I. E. (1999). A branch and contract algorithm for problems with concave univariate, bilinear and linear fractional terms. *Journal of Global Optimization*, 14(3):217–249.

[166] Zhang, C. and Wang, H.-P. (1993). Mixed-discrete nonlinear optimization with simulated annealing. *Engineering Optimization*, 21(4):277–291.

[167] Zheng, X. J., Sun, X. L., and Li, D. (2010). Separable relaxation for nonconvex quadratic integer programming: Integer diagonalization approach. *Journal of Optimization Theory and Applications*, 146(2):463–489.

[168] Zhu, W. and Ali, M. M. (2009). Solving nonlinearly constrained global optimization problem via an auxiliary function method. *Journal of Computational and Applied Mathematics*, 230(2):491–503.