Bond University

Personal Researcher Page of Stephen Sugden

January 1992

A Class of Direct Search Methods for Nonlinear Integer Programming

Contact Author

Start Your Own SelectedWorks

Notify Me of New Work

Available at: http://works.bepress.com/stephen_sugden/3

BOND UNIVERSITY
School of Information & Computing Sciences

A Class of Direct Search Methods for Nonlinear Integer Programming

by

Stephen John Sugden

This thesis is submitted to Bond University in fulfilment of the requirements for the degree of Doctor of Philosophy.

July, 1992

Declaration

This thesis represents my own work and contains no material which has been previously submitted for a degree or diploma at this University or any other institution, except where due acknowledgement is made.

Signature _____

Witness _____

Date _____

Acknowledgement

It was the great English mathematician Godfrey Harold Hardy who wrote:

> *... there is no permanent place in the world for ugly mathematics.*
>
> Hardy [35]

I consider myself fortunate indeed to have had the opportunity to pursue research work toward the PhD with Professor Bruce Murtagh. The mathematical foundations of Professor Murtagh's nonlinear programming work I have found both powerful and elegant. I am convinced that I could not have maintained interest in the work without being profoundly impressed by the inherent elegance and clarity of the underlying methods. I must also pay tribute to the high quality of supervision given by Bruce Murtagh. I can remember many times when I felt discouraged and unsure of the intrinsic worth of my own work. It was he who *never failed* to offer advice and encouragement.

I thank Bond University for the opportunity to pursue research in an environment, which although hectic, provided first-class facilities and supervision for the work. In particular I would like to thank my very good friend and colleague Dr Bernard Duszczyk for his support throughout the work, for his willingness to read the draft thesis, and for the many enjoyable games of tennis that were so welcome in order to restore some balance to the distribution of mental and physical activity for me. It is a great pity indeed that Bernard is no longer with Bond University.

I wish to record my deep gratitude to my friend and former colleague, Professor Richard Tweedie, Dean of the School of Information and Computing Sciences at Bond University from 1988 to 1991. Apart from his offer of support to undertake the PhD at Bond University, Richard was the person through whom I met my supervisor Professor Bruce Murtagh. I

thank Richard most sincerely for his initial confidence in my ability to pursue the PhD and also for his consistent encouragement throughout the all-too-short time of our association. He is sadly missed by all in the School.

Another colleague whose advice and support have been invaluable in the latter stages of the PhD research programme is Professor Les Berry, Director of the Centre for Telecommunication Network Research (CTNR) within the School of Information & Computing Sciences at Bond University. I thank him sincerely for his help in devising the network optimization model presented in this thesis, and also for his continual willingness to offer comments and suggestions despite the seemingly endless revisions to the model and his own very busy schedule. I am grateful also to CTNR Deputy Director Dr Richard Harris for his advice on the material of Chapter 14.

I would also like to thank my parents for their encouragement throughout my entire career, and in particular for their recent patience and encouragement when I visited them to complete some of the latter portions of this work.

Finally, but by no means of least significance, I beg forgiveness from my dear wife Iris and sons Benjamin and Stephen for the many lost hours of leisure time with them over the past few years. We plan to make up for lost time and I believe that the end result has been worth all the sacrifices made on both sides. I thank them for their tolerance and patience.

Dedication

I dedicate this thesis to my parents, John Ernest Sugden and Valma Lillian Mary Sugden, whose love, unselfish support and example over many years laid the foundations for the discipline and application necessary to complete this work.

Abstract

This work extends recent research in the development of a number of direct search methods in nonlinear integer programming. The various algorithms use an extension of the well-known FORTRAN MINOS code of Murtagh and Saunders [62] as a starting point. MINOS is capable of solving quite large problems in which the objective function is nonlinear and the constraints linear. The original MINOS code has been extended in various ways by Murtagh, Saunders and co-workers since the original 1978 landmark paper [62]. Extensions have dealt with methods to handle both nonlinear constraints, most notably MINOS/AUGMENTED [61] and integer requirements on a subset of the variables (MINTO) [58, 49]. The starting point for the present thesis is the MINTO code of Murtagh [58]. MINTO is a direct descendant of MINOS in that it extends the capabilities to general nonlinear constraints and integer restrictions. The overriding goal for the work described in this thesis is to obtain a good integer-feasible or near-integer-feasible solution to the general NLIP problem while trying to avoid or at least minimize the use of the ubiquitous *branch-and-bound* techniques. In general, we assume a small number of nonlinearities and a small number of integer variables.

Some initial ideas motivating the present work are summarised in an invited paper [59] presented by Murtagh at the 1989 CTAC (Computational Techniques and Applications) conference in Brisbane, Australia. The approach discussed there was to start a direct search procedure at the solution of the continuous relaxation of a nonlinear mixed-integer problem by first removing integer variables from the simplex basis, then adjusting integer-infeasible superbasic variables, and finally checking for local optimality by trial unit steps in the integers. This may be followed by a reoptimization with the latest point as the starting point, but integer variables held fixed.

We describe ideas for the further development of Murtagh's direct search method [59]. Both the old and new approaches aim to attain an integer-feasible solution from an initially relaxed (continuous) solution. Techniques such as branch-and-bound or Scarf's neighbourhood search [84] may then be used to obtain a locally optimal solution. The present range of direct search methods differs significantly to that described by Murtagh [59], both in heuristics used and major and minor steps of the procedures. Chapter 5 summarizes Murtagh's original

approach while Chapter 6 describes the new methods in detail. A feature of the new approach is that some degree of user-interaction (MINTO/INTERACTIVE) has been provided, so that a skilled user can "drive" the solution towards optimality if this is desired. Alternatively the code can still be run in "automatic" mode, where one of five available direct search methods may be specified in the customary SPECS file.

A selection of nonlinear integer programming problems taken from the literature has been solved and the results are presented here in the latter chapters. Further, a new communications network topology and allocation model devised by Berry and Sugden [2] has been successfully solved by the direct search methods presented herein. The results are discussed in Chapter 14, where the approach is compared with the branch-and-bound heuristic.

Table of Contents

Chapter 1 Introduction 1
1.1 Contributions of this work 2
1.2 Outline of thesis 3

Chapter 2 Classes of optimization problems 4
2.1 Local and global optima 4
2.2 Smoothness 5
2.3 Constraints 5
2.4 Convexity 6
2.5 Discrete optimization 6
2.6 Optimization examples 7
2.6.1 Example 1—linear objective 7
2.6.2 Example 2—quadratic objective; unconstrained 8
2.6.3 Example 3—quadratic objective; simple bound constraints 9
2.6.4 Nonlinear integer problems 12
2.7 Global optimization 16
2.8 Linear programming 17
2.8.1 The simplex solution method 18
2.9 Integer linear programming 19
2.9.1 Branch-and-bound for ILP 20
2.9.2 Alternative methods for IP 23
Special-purpose algorithms 23
Group-theoretic methods 24
Boolean-algebraic methods 24
Implicit enumeration 24
2.10 Approaches to unconstrained optimization 25
2.11 Approaches to constrained optimization—the Kuhn-Tucker conditions 26
2.11.1 Linear equality constraints 26
2.11.2 Linear inequality constraints 28
2.11.3 Nonlinear equality constraints 31
2.11.4 Nonlinear inequality constraints 34
2.12 Existing algorithms for continuous nonlinear optimization 37
2.12.1 1-Dimensional methods 37
2.12.2 A model descent algorithm schema (unconstrained) 38

Chapter 3 NLIP Literature 42
3.1 General 43
3.2 The outer approximation algorithm of Duran and Grossmann 43
3.3 The method of Mawengkang and Murtagh 45
3.4 Other approaches 45
3.5 Existing NLIP software—a necessarily brief survey 46

Chapter 4 MINOS and its descendants 48
4.1 Fundamental equations for MINOS 50
4.2 Steps of the MINOS algorithm 53
4.3 MINOS/AUGMENTED 56
4.4 MINTO 59

Chapter 5 Murtagh's direct search heuristic 60
5.1 Structure of the problem 60
5.2 CYCLE1—remove integer variables from the basis 64
5.3 CYCLE2 Pass 1—adjust integer-infeasible superbasics 69
5.4 CYCLE2 Pass 2—adjust integer feasible superbasics 72
5.5 Analysis and counterexamples for Murtagh's algorithm 73
5.5.1 Example 1 76
5.5.2 Example 2 78
5.5.3 Example 3 80
5.5.4 Summary of example results for CYCLE1 82
5.5.5 Conclusions 84

Chapter 6 Proposals for new direct search methods 85
6.1 Alternative approaches for CYCLE1 (remove integers from basis) 86
6.2 Alternative approaches for CYCLE2 (superbasic steps) 88
6.3 The new methods 89
6.3.1 Method 1 90
6.3.2 Method 2 90
6.3.3 Method 3 91
6.3.4 Method 4 93
6.3.5 Method 5 94
6.4 Some theoretical properties of the new methods 96

Chapter 7 Implementation of the new direct search methods 99
7.1 SPECS options 99
7.2 Some obstacles encountered 100
7.3 Interactive display program 101
7.4 Functions available in MINTO/INTERACTIVE 102
7.5 Sample screens from MINTO/INTERACTIVE 106
7.6 The FORTRAN workhorse routines 115
7.7 Utility programs 120
7.7.1 MPS generator 120
7.7.2 QIP generator 121
7.7.3 IOPB generator 121
7.8 Some FORTRAN traps 121
7.9 Ideas for future work 124

Chapter 8 Computational experience I—Results for counterexamples 126
8.1 Introduction 126
8.2 Example 1—general comments 126
8.2.1 Objective function/gradient routine CALCFG 127
8.2.2 MPS file 127
8.2.3 Continuous solution 128
8.2.4 Output for method 0 (branch-and-bound) 129
8.2.5 Output for method 1 129
8.2.6 Output for methods 2 and 3 129
8.2.7 Output for method 4 130
8.3 Results for example 2 130
8.4 Results for example 3 131

Chapter 9 Computational experience II—An example from Ravindran *et al* 132

Chapter 10 Computational experience III—A plant upgrade problem 134

Chapter 11 Computational experience IV—A heat exchange network optimization problem 138

Chapter 12 Computational experience V—Two examples from Myers 144
12.1 Myers problem #1 144
12.2 Myers problem #2 148

Chapter 13 Computational experience VI—A loading and dispatching problem in a random flexible manufacturing system 153

Chapter 14 Computational experience VII—A large-scale nonlinear integer programming model for joint optimization of communications network topology and capacitated traffic allocation 160
14.1 Complete graph 163
14.2 Definition of terms 167
14.3 Results and comments for the Berry–Sugden model 172

Chapter 15 Conclusions 177

References 179

Table of Figures

Unconstrained linear optimization 7
Constrained linear optimization 8
Unconstrained quadratic case 1 9
Unconstrained quadratic case 2 9
Constrained quadratic case 1 10
Constrained quadratic case 2 10
Constrained quadratic case 3 11
Constrained quadratic case 4 11
Independent versus combined steps 13
Example Q1 14
Example Q2 15
Linear programming 18
Integer linear programming 20
Descent schema 38
A projected Lagrangian method 57
Index sets for extended simplex partition 63
MINTO/INTERACTIVE displays continuous solution for small QIP 107
Integer-feasible solution for small QIP 108
Help screen #1 for MINTO/INTERACTIVE 109
Help screen #2 for MINTO/INTERACTIVE 110
Help screen #3 for MINTO/INTERACTIVE 111
Help screen #4 for MINTO/INTERACTIVE 112
Help screen #5 for MINTO/INTERACTIVE 113
Debug screen for MINTO/INTERACTIVE 114
Ravindran example optimal solution obtained using M4 133

Table of Tables

Nonbasic step 68
FORTRAN subroutines Part A 116
FORTRAN subroutines Part B 117
FORTRAN subroutines Part C 118
FORTRAN subroutines Part D 119
Plant upgrade model minor parameters 136
Plant upgrade model results summary 137
Heat exchange model minor parameters 141
Heat exchange model results summary 2 142
Myers problem 1 minor parameters 146
Myers problem 1 summary 2 147
Myers problem 2 minor parameters 150
Myers problem 2 summary 151
Shanker & Tzen model minor parameters 157
Shanker model results summary 2 158
Berry model results summary 1 175
Berry model results summary 2 176

Chapter 1

Introduction

Murtagh and Saunders give a concise definition of *optimization:*

> *Optimization is the process of obtaining the best possible result under the circumstances. The result is measured in terms of an* ***objective*** *which is minimized or maximized. The* ***circumstances*** *are defined by a set of equality and/or inequality* ***constraints****. (emphasis ours)*
>
> Murtagh and Saunders [63]

For our purposes, we can sharpen this definition slightly by stating that some of the quantities to be found are required to assume values which are whole numbers. Optimization problems containing restrictions such as these are known as **integer programs**. The present work is concerned with a class of algorithms for **nonlinear integer programs**, in which both the **objective** (quantity to be optimized) and possibly the **constraints** (explicit or implicit restrictions on the values that the variables may assume) are expressed in terms of *nonlinear* functions of the problem variables. If all variables are required to assume integer values, then we have a **pure-integer** program, else a **mixed-integer** program. It is customary to formulate nonlinear optimization problems as minimizations, and this is the approach adopted here.

In essence then, we are trying to find the *best* value of a function of one or many variables, usually under certain restrictions on the values of those variables. The overall "best" is termed the **global optimum**, however finding such a point is normally too computationally expensive and we must be content in most cases to settle for a point which is merely better than its neighbours—a so-called **local optimum**—see however later comments on recent methods for global minimization such as *simulated annealing*. Thus we are concerned here with methods for searching for local minima for the general nonlinear mixed-integer optimization problem.

Apart from being mathematically interesting, the study of optimization and development of algorithms for solving the associated problems has many practical benefits from the application standpoint. The literature of operations research abounds with examples of practical problems from areas as diverse as flexible manufacturing systems [59], process engineering [59], backboard wiring (Steinberg, [88]), electrical engineering (optimal power flow [63]), and optimal communications network design [23] to financial portfolio construction [63].

Many "real-world" problems tend to be large, non-linear, and require at least some of the variables to be integers. Problems in this category are some of the hardest to solve, for reasons which will be described in some detail in the chapters to follow. In particular, for problems with nonlinear constraints it is difficult to characterize a feasible step from the current solution vector to the next. For problems involving integer restrictions—whether linear or nonlinear—the essentially combinatorial nature of the problem gives rise to computation times which can be exponential in the problem size, eg the number of variables; problems of such **exponential complexity** are known to be intrinsically hard. For large-scale work, it is necessary to exploit the inherent *sparsity* of practical problems, and *state-of-the-art* linear programming (LP) software or nonlinear programming (NLP) software such as MINOS [62] takes advantage of sparsity techniques to allow users to tackle quite large problems, yet still retains the flexibility of a general-purpose code.

1.1 Contributions of this work

The present work has made original contribution to the field of **nonlinear integer programming** (NLIP) in the following ways:

1. It has sought to extend the set of available direct-search methods to solve large-scale NLIP using Murtagh and Saunders' concept of *superbasic variables* [62].

2. It is reported herein that such methods have successfully solved a new communications network optimization model. This model seeks to simultaneously optimize network topology and traffic allocation.

3. Improved optima and solution times have been obtained for several test problems in the nonlinear integer programming literature.

1.2 Outline of thesis

Chapter 2 considers in detail the basic classes of optimization problems and gives a brief summary of algorithms which been proposed for their solution. Comments are made as to the relative reliability and robustness of each of the methods.

Chapter 3 gives a survey of NLIP literature.

Chapter 4 sets forth the basic background material for the MINOS code of Murtagh and Saunders, including fundamental equations and the concept of *superbasic variables*.

Chapter 5 discusses Murtagh's direct search method based on superbasic variables, while Chapter 6 develops new modified and refined methods based on original ideas. This chapter presents discussion, proposals and analysis of five new direct search procedures.

Chapter 7 contains information on the implementation of the new approach, including some discussion and comments concerning the development methodology adopted, as well as remarks on the suitability and quality of the software tools both used and developed.

The material from Chapter 8–Chapter 14 presents computational experience with the new methods on a variety of problems from the NLIP literature, as well as a model proposed by Berry and the present author to solve a problem in communications network optimization.

Finally Chapter 15 gives a brief summary of what the work has achieved.

Chapter 2

Classes of optimization problems

There are many classes into which we may partition optimization problems, and also many competing algorithms which have been developed for their solution. We outline the areas of primary interest for the present work, with some brief contrasts to classes of problems that we do not consider here. In the following brief discussion, we assume that the objective function f maps *n-dimensional Euclidean space* R^n to R.

2.1 Local and global optima

An unconstrained local minimum is a point $x \in R^n$ such that there exists a neighbourhood in which the objective at each other point is no better. For a smooth function, it can be pictured geometrically as being at the bottom of a trough at which the gradient vector is zero and the Hessian matrix is necessarily positive semi-definite. Such points are normally not too hard to find using methods that make use of first and second derivative information, typically methods of the Newton class. In a constrained problem, a local minimum may occur at a point where the gradient is not zero, since a constraint boundary may have been reached. In general there may be many local minima, and it is also of interest to find which of the local minima is the "best". Such is *global minimization*, for which a number of alternative methods exist—see, for example Ratschek and Rokne [76]. In general, the task of finding a global minimum is a much harder problem than the task of finding a local minimum, primarily because it is much harder to *verify* that the claimed global minimum is actually that.

2.2 Smoothness

Functions for which continuous derivatives of sufficiently high order exist are referred to as *smooth*. For continuous optimization, we are usually interested in having continuous derivatives up to and including second order. Minimization problems in which the objective and constraint functions are of such type can make use of techniques of multivariable differential calculus which are unavailable for non-smooth functions. We refer primarily to the extensive set of methods which make use of gradient and curvature information to direct an iterative search process toward a local minimum. Methods in this very broad class include the Newton or quasi-Newton methods, good accounts of which may be found in the book by Gill, Murray and Wright [24]. For functions which are not smooth, only function value information can be used to direct the search process. Such techniques are referred to generally as *direct search.* One early approach is the *amœba* or *simplex* method of Nelder and Mead [67], which has recently found favour with researchers developing direct-search methods for *parallel* machines; the more powerful of the gradient-related methods such as *quasi-Newton* do not seem particularly suited to parallel implementations. Details of some work of this nature may be found in two papers by Dennis and Torczon, and Torczon [15, 90]. Brent [4] has written an entire monograph devoted to the topic of minimization without the use of derivatives, ie **direct search methods**. The conventional wisdom is to use reliable gradient information whenever it is available and to avoid "function-value only" methods unless there is no other choice. Gill, Murray and Wright give some useful advice on choice of methods in a *Questions and Answers* appendix to their well-known book [24].

2.3 Constraints

Unconstrained optimization of a function $f : R^n \to R$ of n variables may be thought of as a *search* (certainly from a computing point of view) in R^n for a locally optimizing vector $\mathbf{x}^*$ — constraints simply restrict the search space to some set F, where $F \subseteq R^n$. Thus for constrained optimization, we seek to:

$$\text{minimize} \quad f(\mathbf{x}), \quad \mathbf{x} \in F$$

An unconstrained problem is one in which any point in the domain of f (often the entire space R^n) is an eligible solution point, ie a *candidate* for a local minimum. F is the set of all candidate points and is called the *feasible* set. Any point $\mathbf{x} \in F$ is called a *feasible point*

or *feasible vector*. A constrained problem contains restrictions, in the form of equations or inequalities that must be satisfied at any proposed solution point. We can regard constraints as having the effect of *shrinking* or diminishing the available search space. A further type of restriction, mentioned above, is the requirement that some or all variables must assume *integer* values. Imposition of such conditions will also shrink the available search space, but in a radically different manner to that of, for example, linear constraints. If a set of m linearly-independent linear constraints is imposed on an originally unconstrained n-dimensional problem, then we effectively remove m degrees of freedom (dimensions) from the search space and then can operate in the *reduced* space of dimension n-m. On the other hand, if *integer* conditions are imposed, the search space reduces to a *lattice* of discrete points (in the case of a pure-integer problem). The search then becomes essentially a *combinatorial* one and is in many practical cases fruitless because of the sheer number of possibilities that must be separately examined.

2.4 Convexity

If the objective function is *convex*, equality constraints are *linear*, and inequality constraints are of the form $c_i(x) \geq 0$, ie *concave*, then it can be shown that a local optimum is also global—see, for example Gill, Murray and Wright [24, sec. 6.8.2.1] or Fletcher [18, chapter 9]. It is also worthy of note that many algorithms perform better on such problems, and it is normally possible to *tune* an algorithm to take advantage of convexity if it is known in advance that the problem in question has this desirable property (Gill, Murray and Wright [24, sec. 6.8.2.1]).

2.5 Discrete optimization

We discuss integer programming (both linear and nonlinear) later in this thesis, however the apparently more general problem of nonlinear optimization subject to *general discrete* restrictions has also received some recent attention. Such problems require a (generally nonlinear) objective to be minimized subject to nonlinear inequality constraints, with the added requirement that certain or all of the structural variables must take values from specified finite sets; the elements of these sets need not be integers. For a recent example in which the classical *penalty* function approach (Sequential Unconstrained Minimization

Technique (SUMT) of Fiacco and McCormick [20]) is applied in order to satisfy both nonlinear constraints and the discrete requirements, see the 1990 paper by Shin, Güerdal and Griffin [87], in which applications to engineering truss design are considered.

2.6 Optimization examples

To gain some appreciation of the issues involved in NLIP, we consider a sequence of simple, but progressively more difficult optimization problems, culminating in an admittedly small, but illustrative quadratic NLIP problem.

2.6.1 Example 1—linear objective

Here we consider an unconstrained linear univariate example: the simplest possible kind of problem. This is a trivial case, but illustrates some useful ideas which lead on to the next example. We seek the minimum of the function $f(x) = ax + b$.

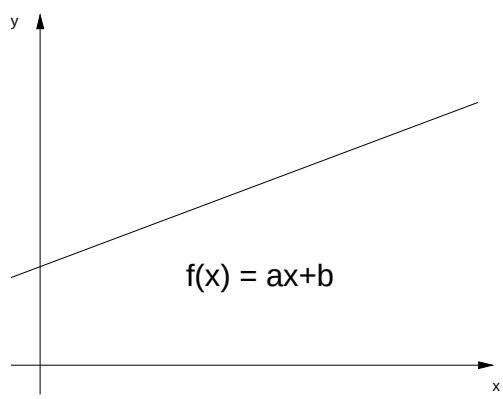

Figure 1 Unconstrained linear optimization

If we consider the unconstrained minimum problem illustrated figure 1, there is no solution since the objective function is unbounded, assuming that $a \neq 0$.

Imposition of a simple bound constraint $x \geq L$ on this problem, leads to a unique global minimum at the constraint boundary $x = L$, provided that $a > 0$ — see figure 2. This is typical of linear optimization (LP) in multidimensions.

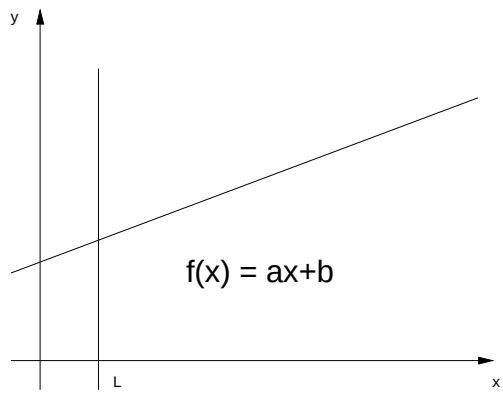

Figure 2 Constrained linear optimization

2.6.2 Example 2 — quadratic objective; unconstrained

We seek the unconstrained minimum of the function

$$f(x) \quad = \quad A(x-a)(x-b) \tag{1}$$

Case 1 $A > 0$

Local and global minimum is at $x^* = (a+b)/2$, and the value of the objective function is

$$f(x^*) \quad = \quad \frac{-A(b-a)^2}{4} \tag{2}$$

Case 2 $A < 0$

No local minimum; no global minimum; ie given any M; $\exists\, x \quad : \quad f(x) < M$.

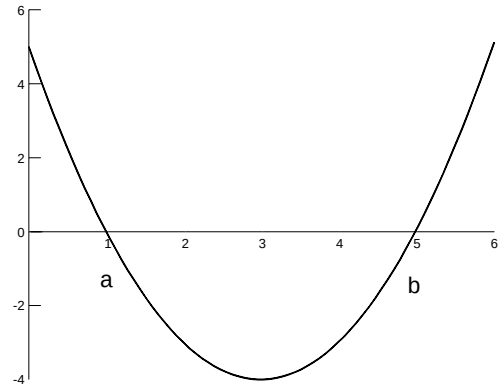

Figure 3 Unconstrained quadratic — case 1

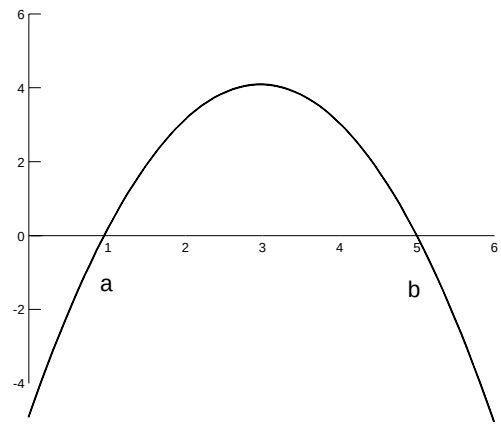

Figure 4 Unconstrained quadratic — case 2

2.6.3 Example 3—quadratic objective; simple bound constraints

Even in this simple case, many outcomes are possible. Once again our objective function is

$$f(x) \quad = \quad A(x-a)(x-b) \tag{3}$$

However we now impose the (simple bound) constraints

$$L \quad \leq \quad x \quad \leq \quad U \tag{4}$$

Case 1 $A > 0; \quad L < a < b < U$

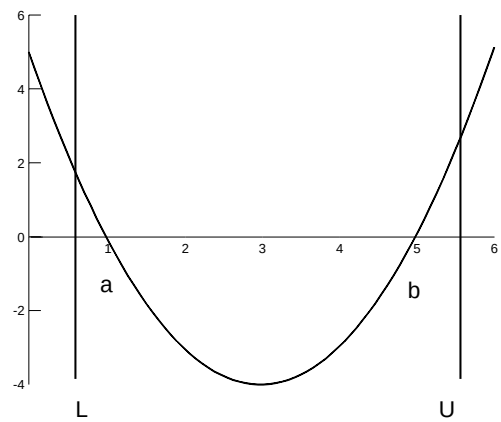

Figure 5 Constrained quadratic — case 1

$x^* \quad = \quad (a+b)/2$ is a (local) minimum on the interval $L \leq x \leq U$. Since f is convex, x^* is also a *global* minimum on the interval.

Case 2 $A > 0; \quad L < a < (a+b)/2 < U < b$

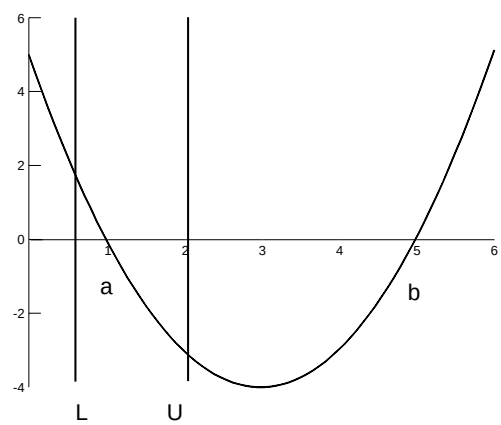

Figure 6 Constrained quadratic — case 2

$x^* \quad = \quad U$ is the global minimum.

Case 3 $A > 0; \quad L < U < (a+b)/2$

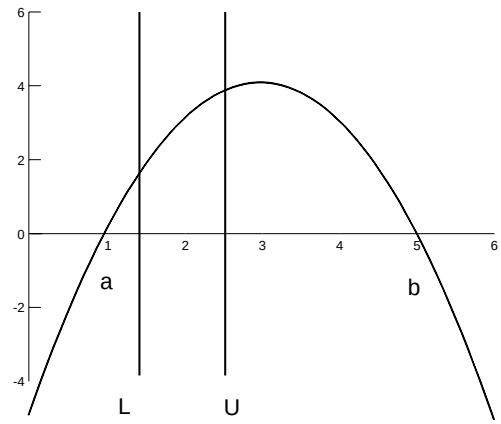

Figure 7 Constrained quadratic — case 3

The global minimum here is $x^* = L$, however this point is *not* a smooth local minimum. From elementary calculus, the necessary conditions for a smooth local minimum are that $f'(x) = 0$ and that $f''(x) \geq 0$.

Sufficient conditions for smooth local minimum are $f'(x) = 0$ and that $f''(x) > 0$.

Case 4 $A < 0$

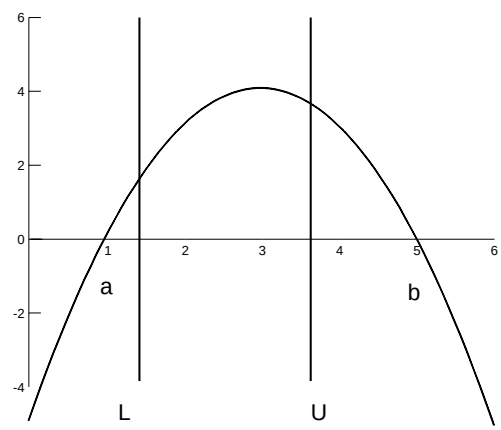

Figure 8 Constrained quadratic — case 4

We cannot get a local minimum since $f''(x) \equiv -2A$ everywhere. However it is always possible to get a global minimum, which then occurs at one of the endpoints, L or U.

We see that even for simple univariate quadratic functions, we have many cases to consider. We also note in passing that any reasonable computer algorithm to solve optimization problems must be able to cater for these fundamental cases and their generalizations to problems involving many variables.

2.6.4 Nonlinear integer problems

In order to illustrate some of the possible cases and difficulties for nonlinear integer problems we present some general observations on the limitations of simple-minded direct search, followed by two very small linearly-constrained **quadratic integer programs** (QIPs).

It will be seen that in some cases, simple rounding of the solution to the continuous relaxation produces acceptable results, while in other cases, this process may even lead to infeasibility, quite apart from any hope of local optimality.

The previous examples in this chapter have been concerned with univariate problems for which conventional graphs of the form $y = f(x)$ were used to illustrate constrained minima. By contrast, in the following general illustration and two simple quadratic integer programs (depicted in figures 9–11), we consider problems with two independent structural variables, and the cartesian plane is used to show the lattice of feasible points for each problem.

Notwithstanding the usual caveats regarding the use of diagrams to prove a point, it should be emphasized that, although this example is very simple, a very fundamental and important stumbling-block for direct search methods for integer programming is here illustrated. The diagram shows four lattice points: labelled 1, 2, 3, 4. Points 1 and 4 are infeasible as they lie outside the linear constraints. It should be clear from the diagram that independent steps in each of two orthogonal directions are seen to be infeasible with respect to linear constraints, however a combined oblique step gives feasibility, and this point may even be locally optimal. The search procedures presented in this work first spend a great deal of effort to rid the simplex basis of integer variables. After this is done, any integer-infeasibilities must of necessity be in the superbasic variables. The diagram under discussion here must be interpreted in the light of small increments in superbasic variables in some reduced search space. If independent steps fail in such a small example as this, then the situation can only become worse as the dimensionality of the problem increases. ***We cannot avoid the combinatorial barrier that is inherent in any kind of integer programming***—here it raises its ugly head in the form of the choice that must be made if we are going to try steps in more than one superbasic at a time. Figure 9 is but a simple example in which ’taking one at a

time' is sometimes insufficient to achieve integer-feasibility, and further examples can easily be constructed to show that taking two at a time can also fail. Here we have four lattice points, and a fifth point, which represents the solution of the continuous relaxation. Independent steps in either x or y will hit a constraint boundary, but a "diagonal" step succeeds.

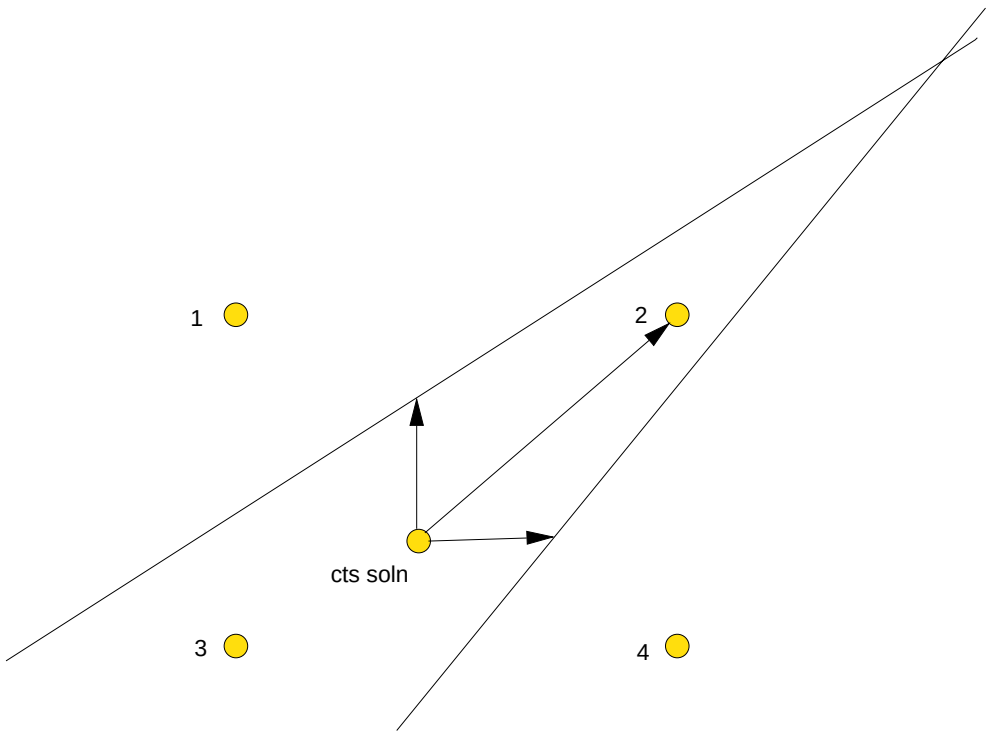

Figure 9 Independent versus combined steps

Quadratic integer example Q1

The problem considered here is

minimize

$$f(x_1, x_2) = (x_1 - 3.4)^2 + (x_2 - 1.6)^2 \tag{5}$$

subject to

$$x_1 - x_2 \geq 1 \tag{6}$$

$$4x_1 - x_2 \leq 16 \tag{7}$$

$$0 \leq x_1 \leq 5 \tag{8}$$

$$0 \quad \leq \quad x_2 \quad \leq \quad 4 \tag{9}$$

$$x_1, \quad x_2 \quad \text{integer} \tag{10}$$

In this example, we see that the continuous optimum is 0.0 at $\mathbf{x}_0^* \quad = \quad (3.4, 1.6)^T$, and the integer-feasible optimum is at $\mathbf{x}_1^* \quad = \quad (4, 2)^T$, with objective 0.52. The integer optimum can be obtained by heuristic rounding of the continuous solution. The feasible region is illustrated in figure 10, where the asterisk indicates the continuous solution, and filled circles indicate feasible lattice points. Note that there is also a second local optimum at $\mathbf{x}_2 \quad = \quad (3, 1)^T$ with the same objective value, 0.52.

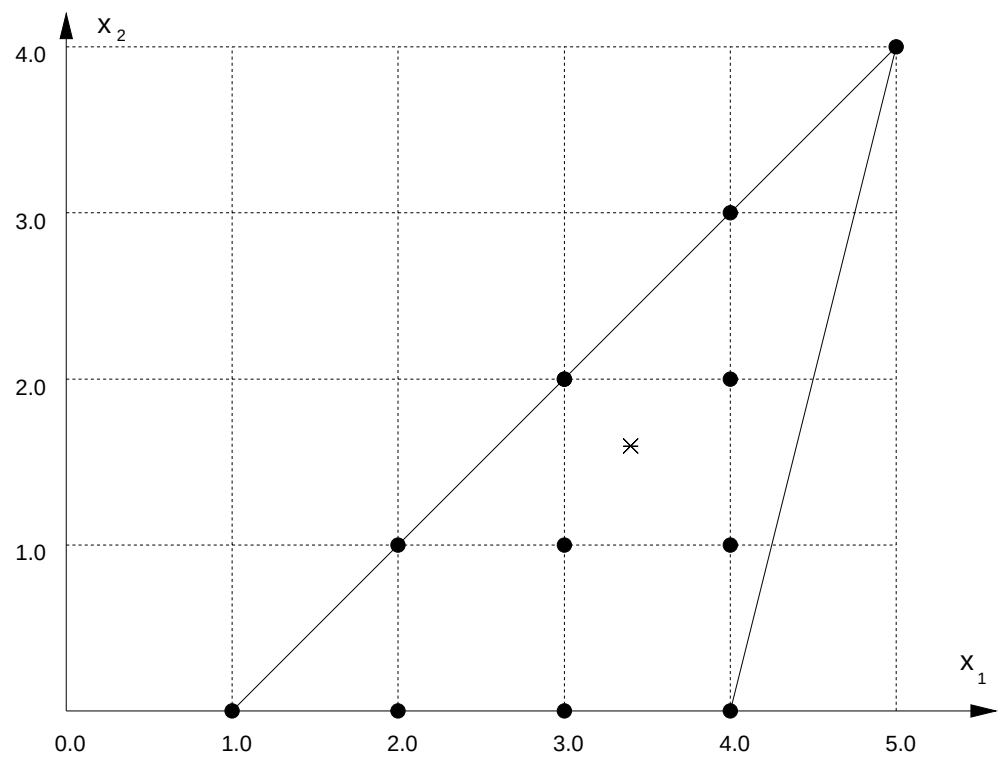

Figure 10 Example Q1

Quadratic integer example Q2

The problem considered here is

minimize

$$f(x_1, x_2) \quad = \quad (x_1 \quad - \quad 3.4)^2 \quad + \quad (x_2 \quad - \quad 1.6)^2 \tag{11}$$

subject to

$$4x_1 \quad - \quad 3x_2 \quad \geq \quad 8 \tag{12}$$

$$2x_1 \quad - \quad x_2 \quad \leq \quad 6 \tag{13}$$

$$0 \quad \leq \quad x_1 \quad \leq \quad 5 \tag{14}$$

$$0 \quad \leq \quad x_2 \quad \leq \quad 4 \tag{15}$$

$$x_1, \quad x_2 \quad \text{integer} \tag{16}$$

In this example, we see that the continuous optimum is once again 0.0 at $\mathbf{x}_0^* = (3.4, 1.6)^T$, and the integer-feasible optima are $\mathbf{x}_1^* = (4, 2)^T$ and $\mathbf{x}_2^* = (3, 1)^T$, as before (both have objective 0.52). This problem is illustrated in figure 11, and it is worthy of note that, if we wish to maintain feasibility with respect to the linear constraints, only the latter solution is obtainable by independent steps in the variables, and even then we must step x_2 before x_1.

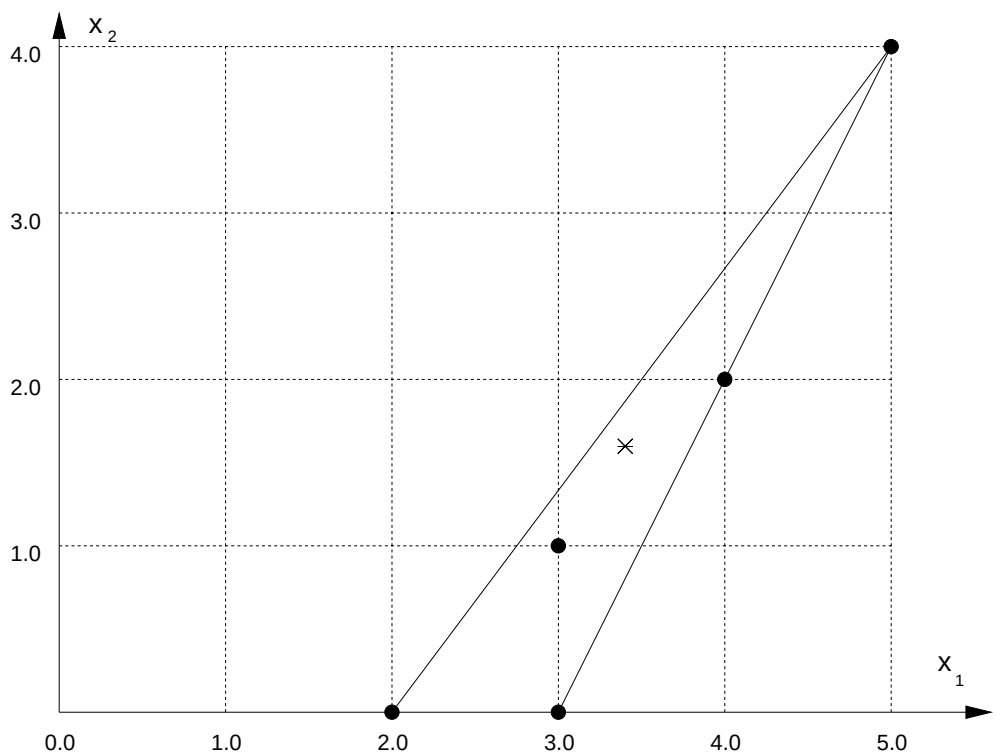

Figure 11 Example Q2

2.7 Global optimization

For a large class of practical problems, global minimization is, in general, an impossible task, although in a number of practical cases, such problems have been solved in a satisfactory manner—see, for example, the book by Ratschek and Rokne [76]. Normally, "real-world" optimization problems are global, constrained, mixture of discrete and continuous, nonlinear, multivariate and nonconvex, ie *the hardest possible mathematically!* It's not all bad news, since much useful progress has been made by taking advantage of simplifying models and special problem structure. In particular, the development of interval arithmetic and interval analysis by Moore, Mohd, Ratschek, Rokne and others [54, 55, 76] has led to significant algorithmic advances in the last few decades.

Interestingly, some of the more imaginative of recent attempts at optimization methods try to mimic perceived processes of nature. One such approach is that of *simulated annealing*; another is *evolution* via the class of so-called *genetic algorithms*.

The *simulated annealing* technique shows a lot of promise for global optimization problems. In its early stages, the method allows local deterioration of objective (but with gradually declining probability) in the hope that beyond local bumps may be deeper valleys. A good encapsulated description of simulated annealing is given in Press, Flannery, Teukolsky and Vetterling [75], where the old *travelling salesman* chestnut is discussed, and FORTRAN and Pascal code given for the method. Simulated annealing has become quite a popular line of research, and appears to be a particularly good heuristic for the *quadratic assignment problem* (QAP). For a definition of QAP the reader is referred to Connolly [8] or Mawengkang and Murtagh [49]. In particular, Connolly [8], Burkhard and Rendl [6], and Wilhelm and Ward [95] report some improved optima for several of the largest problems available in the literature. Although the annealing algorithm is "intrinsically of a sequential nature" [91, chapter 8], parallel implementations do exist, as reported by van Laarhoven and Aarts [91].

Other recent contenders for efficient solution of a very general class of optimization problems are members of the class of methods known loosely as *genetic algorithms*. The basic ideas are conventionally expressed in biological terms; thus a *gene pool* is maintained throughout the execution of a typical method, during which *mutations* are generated from potential solutions by crossbreeding (partial permutations). Resulting solutions at each generation are ranked according to some measure of fitness; in the case of minimization, an obvious class of criteria is that whose rules compute the objective corresponding to each contender, and then favours those with better objective value. It is hoped that via this process of *simulated evolution*, a *population* will gradually emerge which contains the *seeds* of the

final solution. Some useful tutorial articles on the subject of genetic algorithms are Morrow [56] and Wayner [94], while two recent monographs are Davis [14] and Goldberg [27]. A specific genetic method has been applied by Michalewicz, Krawczyk, Kazemi and Janikow [51] to optimal control problems; its performance in some cases comparing favourably with that of the GAMS/MINOS code of Murtagh and Saunders [26].

2.8 Linear programming

Linear programming (LP) problems are linear in both objective and constraints. The special nature of this class of problems makes possible a very elegant solution algorithm known as the *revised simplex method*—the classic reference is Dantzig [12], while a more modern treatment from a large-scale computational viewpoint is contained in the monograph of Murtagh [57].

The basic result of LP theory stems from the nature of the feasible set. The feasible set can be characterised geometrically as a *convex polytope* (or *simplex*), which can be imagined to be an n-dimensional polyhedron, and if an optimal solution exists, then there is at least one vertex of the feasible set that is optimal. Figure 12 illustrates a trivial LP in which the interior of the shaded quadrilateral OPQR represents the feasible set. The fundamental result tells us that if a finite optimal point exists, then (at least) one of the vertices O, P, Q, R (corresponding to so-called *basic feasible solutions*) is optimal.

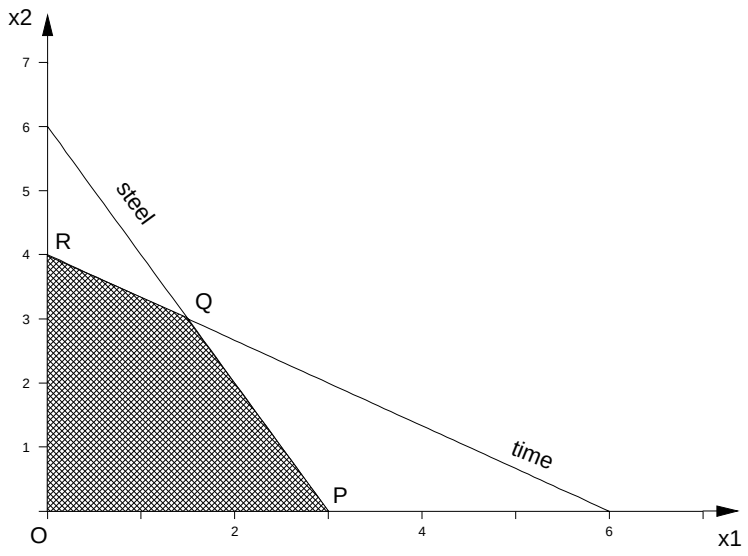

Figure 12 Linear programming

2.8.1 The simplex solution method

Diagrams such as figure 12 are used to illustrate some fundamental concepts of LP, however we cannot use graphical methods for real-world problems. An algebraic approach is required, suitable for implementation on a digital computer. The simplex algorithm systematically moves from vertex to vertex of *F*, the feasible set. Such vertices are termed *basic feasible solutions*. Each iteration improves the value of the objective until no further improvement is possible. It is a very elegant algorithm, which maintains feasibility of candidate solutions (basic solutions) at every step.

The great early success of *mathematical programming* was the development of the simplex method by George Dantzig and co-workers Orden and Wolfe [12, 13] for the solution of LP problems. The method was originally developed for hand calculation but was easily adapted for use on digital computers, for which the *revised simplex method* is normally used. Since then, many extensions and refinements have been developed for the method. One of the most important developments has been a class of techniques known in general terms as

sparse matrix techniques. These methods take advantage of the fact that almost all large LPs have very sparse constraint matrices, ie almost all zeros. Nowadays it is quite common to solve LPs containing tens of thousands of constraints. If such a problem were solved using only dense matrix methods, that is, by explicitly storing and processing all the zero elements, we would be dealing with the storage of hundreds of millions of coefficients—amounting to storage in excess of one gigabyte for the constraints alone. Not only is *space* wasted, but *time* also, since most of the processor's time would be occupied in doing multiplications by zero.

In recent years there have been alternative LP algorithms proposed, most notably that of Karmarkar [40], but the revised simplex method and its close variants are still the most popular. A good discussion of Karmarkar's method is also given in Strang [89]. It is interesting to observe that while the asymptotic complexity of the simplex method is exponential in the number of constraints, and Karmarkar's is only polynomial, the revised simplex method performs very well in practice. Artificial problems which elicit the worst-case exponential simplex behaviour have been constructed (see for example [69], chapter 8.6, p169) but these do not seem to occur in practice. Karmarkar's algorithm is an example of an **interior point** method, and some very recent work by Marsten, Subramanian, Saltzman, Lustig and Shanno [47] has interpreted interior point methods as a natural combination of the previously-known techniques due to Newton, Lagrange, and Fiacco and McCormick [20]. The claim is made that interior point methods are "the right way to solve large linear programs", and results reported by Lustig, Mulvey and Carpenter [45] are cited in which the OB1 code of Marsten *et al* [46] outperforms MINOS 5.3 [64] by a factor of ten on problems which have $m+n$ ranging from 7,391 to 29,424.

In spite of all this, the simplex method is well worth studying, not only for the insight it gives into the structure of LP problems, and also of course their solution, but for the useful extensions to the method which can be used to solve NLP problems. Lucid treatments of some of these extensions can be found in Gill, Murray and Wright [24], Murtagh and Saunders [62], and Murtagh [57].

2.9 Integer linear programming

Integer linear programming problems (ILPs) are LP problems in which extra constraints requiring some or all variables to be integer valued have been imposed. ILP is a very common problem class where variables representing indivisible units, eg men, machines do not admit

fractional solutions. Figure 13 shows the combinatorial nature of such problems by an enumeration of the (finite) feasible set of lattice points, rather grossly depicted by the filled squares.

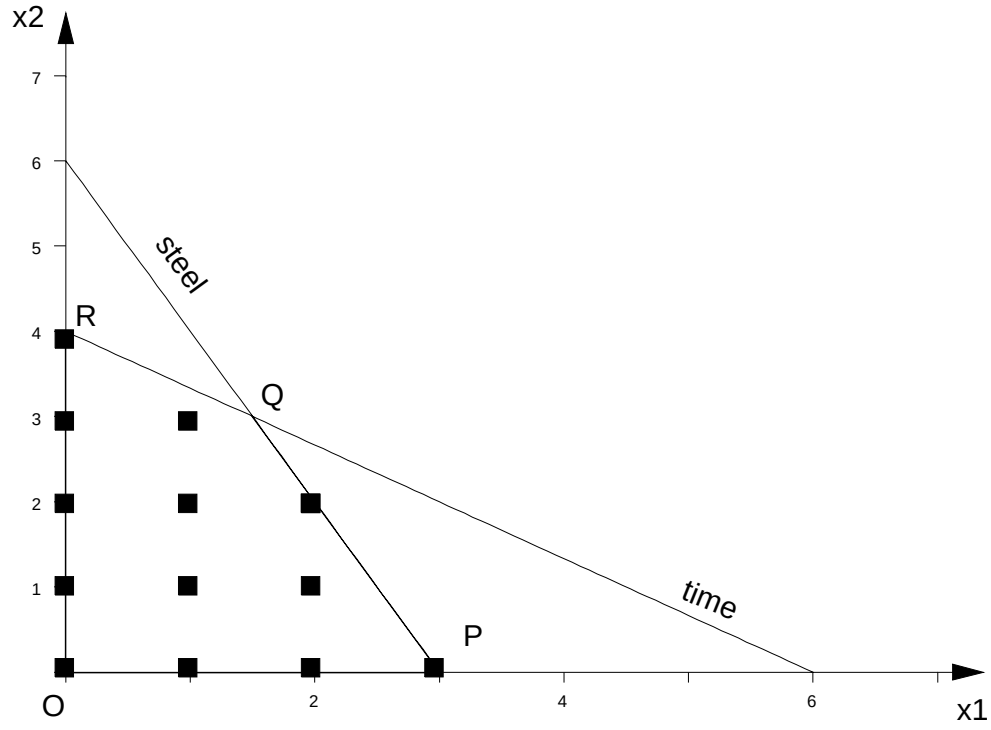

Figure 13 Integer linear programming

2.9.1 Branch-and-bound for ILP

ILPs are much harder to solve than continuous LPs, because of their fundamentally combinatorial nature. They are potentially exponential in computational complexity, even after the continuous LP relaxation has been solved by, say, the revised simplex method.

The standard algorithm for solving ILP is the very simple-in-concept *branch-and-bound.* In this approach, for which myriad variations exist, the LP relaxation is first solved. Then, if all required variables are within a specified tolerance of an integer value, the process halts. Otherwise the problem is split into two sub-problems at an infeasible integer variable. The subproblems are then put on a master list of subproblems to be solved. The next step is to select one subproblem from the list and repeat the above procedure. The whole process

terminates when there are no subproblems left. At all stages, we keep track of the best integer-feasible solution so far. Subproblems can be discarded entirely if it becomes clear that their solutions are no better than the current "best" integer feasible solution. The sequence of problems thus takes the form of a *tree*, from which it should be clear how the exponential complexity arises. The main aims are to keep the branching to as few nodes as possible, and the main drawback of the method is that unless special steps are taken, the number of subproblems (branches) can grow exponentially.

For a thorough description of the basic branch-and-bound approach, any of a large number of textbooks may be consulted; we cite Nemhauser and Wolsey [68] 355–367 (this work appears to be the most comprehensive general treatment of integer programming extant), Papadimitriou and Steiglitz [69] 433–448, Murtagh [57], 107–111, Minoux [52], 248–258 and Ravindran, Phillips and Solberg [77], 191–198.

It should be noted also that there exist ILPs which are intrinsically hard no matter what branch-and-bound heuristic is used for fathoming the search tree. In fact it can be shown that there are ILPs such that:

> *... regardless of the order for fixing variables, and regardless of the method for choosing nodes, all the enumerative algorithms will have to develop at least a number of nodes roughly equal to the square root of the number of nodes in the entire search tree, hence a number of nodes exponential in the number of variables.*
>
> Jeroslow [39]

In spite of these fundamental objections, nearly all commercial linear programming codes use the branch-and-bound method for solving linear integer programming problems. The approach can be easily adapted to nonlinear problems, and we give some details here.

Branch and bound for nonlinear problems

The original problem is solved as a continuous nonlinear program, ignoring the integrality requirements. Suppose the solution $x_j, \quad j \in J$ is not completely integer-feasible. We set

$$x_j \quad = \quad [x_j] \quad + \quad f_j, \qquad 0 \quad \leq \quad f_j \quad < \quad 1 \tag{17}$$

where $[x_j]$ is the smallest integer not exceeding x_j.

The approach is to generate two new subproblems, with additional bounds, respectively

$$l_j \quad \leq \quad x_j \quad \leq \quad [x_j] \tag{18}$$

and

$$[x_j]+1 \quad \leq \quad x_j \quad \leq \quad u_j \tag{19}$$

for a particular variable $j \in J$. This process of splitting the problem is called **branching**. As in the linear case, one of these new subproblems is now stored in a master list of problems remaining to be solved, and the other solved as a continuous problem. This represents the *depth-first* approach to branch-and-bound. Other strategies in which *both* incumbents are "stacked" and some heuristic used to select which of the previously-generated problems on the list to solve next. The process of branching and solving a sequence of continuous problems is repeated for different integer variables, $j \in J$, and different integers $[x_j]$. As stated, the logical structure of the method is often represented as a *tree*. Each node of the tree represents a subproblem solution. Branching at a given node will terminate if one of the following three criteria is satisfied:

Termination criteria for branch-and-bound

1. The subproblem has no feasible solution.
2. The solution of the subproblem is no better than the current best known integer feasible solution.
3. The solution is integer feasible (to within a pre-defined level of tolerance).

One benefit of the branch-and-bound approach is that both upper and lower bounds on the best possible integer solution are automatically available. Assuming the objective is to be *minimized*, the current best known integer feasible solution provides an upper bound, and the best of the remaining partially-integer solutions on the master list of problems to be solved provides a lower bound. It is usual to terminate the branch-and-bound procedure when the difference between these two bounds is within some pre-defined relative tolerance.

In general, the rate of convergence of the procedure is sensitive to the choice of variable $j \in J$, on which to branch. It is also dependent on the choice of the node to which backtracking is done, once the branching from a particular node is discontinued.

For nonlinear integer programs, it must be implicitly assumed that the problem is locally convex, at least in the neighbourhood of the original continuous solution which contains integer feasible solutions. Otherwise, the bounds discussed above are inadequate. It would not be valid to terminate the branching under the termination criterion 2 above, and it also would not be valid to terminate the procedure when the difference between the two bounds is sufficiently small.

2.9.2 Alternative methods for IP

In this section we discuss some alternatives to branch-and-bound which have been proposed for the solution of IP problems.

Special-purpose algorithms

Much of the literature on even *linear* IP is devoted to the description of special-purpose methods which have been developed to exploit the particular structure of the problem at hand. A very good example of a case in which special problem structure has been used to advantage is that of the airline crew scheduling problem reported by Ryan [80], where 200,000 0–1 (binary) variables are involved. The solution of an integer program of this magnitude would be out of the question if general-purpose methods such as pure branch-and-bound were used.

Nemhauser and Wolsey [68] devote a large amount of space to special-purpose algorithms for IP, which can be used to attack otherwise intractable problems. They cite three major reasons for motivating a search for special-purpose approaches:

(i) Prohibitive size of the problem formulation.
(ii) Weakness of bounds.
(iii) Slow speed of general-purpose methods, eg branch-and-bound.

Group-theoretic methods

Group-theoretic methods based on original ideas of Gomory in a sequence of papers [28, 29, 30] are used for pure integer problems and implementations can be found in some commercial codes. They are given only a cursory treatment in Nemhauser and Wolsey [68], however the book by Salkin and Mathur [81] contains an extensive account of these techniques, including much on computational experience. The reader seeking further details is referred to this book [81].

Boolean-algebraic methods

These apply to 0–1 programs, however any NLIP can be written as polynomial involving 0–1 variables. Methods for such transformations are described in the work by Hammer and Rudeanu [33]. A good summary of progress up to the late 1970s is given in the survey paper [34] by Hansen, where in particular, he notes that any nonlinear 0–1 program is equivalent to a 0–1 **knapsack problem** (IP with one constraint) in the same variables. From a computational point of view however, this result is not as useful as it may sound, since it may take at least as much time to find the single constraint as to solve the original problem by some other means. The paper [34] by Hansen is a good starting point for those interested in Boolean methods for IPs.

Implicit enumeration

Implicit enumeration is used on 0–1 problems, although it is in principle available for any pure-integer or mixed-integer problem since integer variables with finite upper and lower bounds may be converted to sets of 0–1 variables. By clever use of bounds, it may be arranged for certain problems that not all feasible lattice points need be explicitly considered, however it is with such methods that the exponential complexity of IP in general is laid bare, and it is difficult to imagine much future success for implicit enumeration even on practical problems of moderate size. This class of methods, related to branch-and-bound, is well-summarized in the dissertation of Mawengkang [48].

2.10 Approaches to unconstrained optimization

The general unconstrained nonlinear minimization problem can be stated in the form:

$$\text{minimize} \qquad F(\mathbf{x})$$

$$\mathbf{x} \in R^n$$

The quadratic objective function in n variables is used as a model and very simple test function for algorithms purporting to solve problems in this very broad class. Any algorithm for unconstrained optimization must perform well on quadratics, since all smooth functions are like quadratics in a sufficiently small neighbourhood of a local smooth optimum. Two fundamental theorems for development of unconstrained minimization methods are Taylor's theorem:

$$F(\mathbf{x}+\mathbf{p}) \quad = \quad F(\mathbf{x}) \quad + \quad \mathbf{g}(\mathbf{x})^T\mathbf{p} \quad + \quad \frac{1}{2}\mathbf{p}^T H(\mathbf{x}+\theta\mathbf{p}) \tag{20}$$

and the mean value theorem:

$$\mathbf{g}(\mathbf{x}+\mathbf{p}) \quad = \quad \mathbf{g}(\mathbf{x}) \quad + \quad H(\mathbf{x}+\theta\mathbf{p}) \tag{21}$$

in both of which we have

$$0 \quad < \quad \theta \quad < \quad 1 \tag{22}$$

It is here assumed that F is twice continuously differentiable with gradient vector $\mathbf{g}(\mathbf{x})$ and Hessian matrix $H(\mathbf{x})$, the respective elements of which are given by:

$$g_j(\mathbf{x}) \quad = \quad \frac{\partial F(\mathbf{x})}{\partial x_j} \tag{23}$$

$$h_{ij}(\mathbf{x}) \quad = \quad \frac{\partial^2 F(\mathbf{x})}{\partial x_i \partial x_j} \tag{24}$$

To have any hope of a local minimum, we need the Hessian matrix to possess a property called *positive definiteness*. In particular, for a quadratic function, the Hessian matrix H is *constant*.

For sufficiently smooth objective function, necessary conditions for a minimum for the unconstrained problem are $\mathbf{g}(\mathbf{x}^*) \quad = \quad \mathbf{0}$ and $H(\mathbf{x}^*) \quad \geq \quad \mathbf{0}$.

Sufficient conditions for a minimum are $\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$ and $H(\mathbf{x}^*) > \mathbf{0}$.

The shorthand notation $H(\mathbf{x}^*) > \mathbf{0}$ represents the condition *positive definiteness* for the Hessian matrix of partial derivatives. It can be shown that the following conditions for positive definiteness are all equivalent [24]:

Positive definiteness of Hessian

1. $\mathbf{x}^T H \mathbf{x} > 0; \quad \forall \mathbf{x} \neq \mathbf{0}.$

2. H has positive spectrum (set of eigenvalues).

3. LL^T (Cholesky) factors of H exist with the diagonal elements of L, $l_{ii} > 0$.

4. All multipliers in Gaussian elimination without pivoting (row or column interchanges) are positive.

5. All principal minors of H are positive.

2.11 Approaches to constrained optimization—the Kuhn-Tucker conditions

In this section we summarize some especially lucid material from Gill, Murray and Wright [24]. In a classic 1951 paper [43], Kuhn and Tucker derived conditions for constrained optimal points for a general nonlinear function f. The so-called *Kuhn-Tucker conditions* are presented for various classes of continuous constrained optimization problems.

The traditional technique of using so-called *Lagrange multipliers* to handle constraints is still a very powerful, theoretically elegant and the most widely used approach in practice, both for analytic and numerical methods.

2.11.1 Linear equality constraints

We define the linear equality-constrained problem LEP:

LEP

minimize $F(\mathbf{x})$

$\mathbf{x} \in R^n$

subject to $\hat{A}\mathbf{x} = \hat{\mathbf{b}}$

Optimality conditions

For each case to follow, the aim is to characterize a feasible step from the current iterate, and then to deduce some necessary conditions, and then some sufficient conditions for a local minimum. We begin with the simplest kind of constraints — linear equality constraints.

Since any feasible step $\mathbf{p}$ from $\mathbf{x}$ to $\mathbf{x}+\mathbf{p}$ must satisfy $\hat{A}\mathbf{p} = \mathbf{0}$, the step $\mathbf{p}$ must be an element of the *nullspace* (or *kernel*) of the matrix $\hat{A}$. Let a basis for the nullspace of $\hat{A}$ be formed by the columns of the matrix Z. Examination of the Taylor series about a proposed optimal point $\mathbf{x}^*$ reveals that we must have $Z^T\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$. The vector $Z^T\mathbf{g}(\mathbf{x}^*)$ is called the *projected gradient* at $\mathbf{x}^*$. Any point at which the projected gradient vanishes is termed a *constrained stationary point*. Likewise, we define the *projected Hessian* matrix $Z^T G(\mathbf{x}^*)Z$. At such a point it is easy to show that the gradient vector must be a linear combination of the rows of $\hat{A}$, ie there exists a vector λ^* such that

$$\mathbf{g}(\mathbf{x}^*) = \sum_{i=1}^{m} \hat{a}_i \lambda^* = \hat{A}^T \lambda^* \tag{25}$$

where λ^* is the vector of *Lagrange multipliers*. In a similar manner to the unconstrained case, we can derive second-order optimality conditions.

Note that the conditions are analogous to the unconstrained case, except that the projected gradient and projected Hessian are involved.

LEP — necessary conditions for minimum

$$\hat{A}\mathbf{x}^* = \hat{\mathbf{b}} \tag{26}$$

$$Z^T\mathbf{g}(\mathbf{x}^*) = \mathbf{0} \tag{27}$$

$$\mathbf{g}(\mathbf{x}^*) = \hat{A}^T\lambda^* \tag{28}$$

$$Z^T G(\mathbf{x}^*)Z \geq 0 \tag{29}$$

The second and third of these conditions are actually equivalent, and together the four become sufficient if we strengthen the last to a sharp inequality, thus:

$$Z^T G(\mathbf{x}^*)Z > 0 \tag{30}$$

2.11.2 Linear inequality constraints

We define the linear inequality-constrained problem LIP:

LIP

minimize $F(\mathbf{x})$

$\mathbf{x} \in R^n$

subject to $A\mathbf{x} \geq \mathbf{b}$

We need to distinguish between constraints which hold exactly and those which do not. Let us suppose that the point $\hat{\mathbf{x}}$ is feasible. The constraint $\mathbf{a}_i^T\hat{\mathbf{x}} \geq b_i$ is said to be *active* (or *binding*) if $\mathbf{a}_i^T\hat{\mathbf{x}} = b_i$, and *inactive* if $\mathbf{a}_i^T\hat{\mathbf{x}} > b_i$. The constraint is said to be *satisfied* if it is active or inactive. If $\mathbf{a}_i^T\bar{\mathbf{x}} < b_i$, the constraint is said to be *violated* at $\bar{\mathbf{x}}$.

Active constraints have a special significance in that they restrict feasible perturbations about a feasible point. We may define two categories of feasible perturbations with respect to an active inequality constraint. Firstly, if

$$\mathbf{a}_i^T\mathbf{p} \quad = \quad 0 \tag{31}$$

then the direction $\mathbf{p}$ is termed a *binding perturbation* with respect to the constraint, since this constraint remains active at all points $\hat{\mathbf{x}} + \alpha\mathbf{p}$. A move along a binding perturbation is said to remain *on* the constraint.

Secondly, if

$$\mathbf{a}_i^T\mathbf{p} \quad > \quad 0 \tag{32}$$

then $\mathbf{p}$ is termed a *non-binding perturbation* with respect to the constraint. This is because the constraint will become *inactive* at the perturbed point $\hat{\mathbf{x}} + \alpha\mathbf{p}$, assuming that $\alpha > 0$. Such a positive step along a non-binding perturbation is said to move *off* the constraint.

To determine if the feasible point $\mathbf{x}^*$ is also optimal for LIP, we must identify the active constraints. Let the t rows of the matrix $\hat{A}$ contain the coefficients of the constraints active at $\mathbf{x}^*$, with a similar convention for the vector $\hat{\mathbf{b}}$, so that $\hat{A}\mathbf{x}^* = \hat{\mathbf{b}}$. Once again let Z be a matrix whose columns form a basis for the set of vectors orthogonal to the rows of $\hat{A}$.

By considering the Taylor series expansion for f about $\mathbf{x}^*$ along a binding perturbation $\mathbf{p} = Z\mathbf{p}_Z$, we obtain

$$Z^T\mathbf{g}(\mathbf{x}^*) \quad = \quad \mathbf{0} \tag{33}$$

This is equivalent to

$$\mathbf{g}(\mathbf{x}^*) \quad = \quad \hat{A}^T\lambda^* \tag{34}$$

To ensure that non-binding perturbations do not allow a *descent direction* (a direction for the objective function decreases), we need to impose the condition that all Lagrange multipliers are nonnegative. Further, we obtain necessary second-order condition in a similar manner to LEP, in which the *projected Hessian* $Z^TG(\mathbf{x}^*)Z$ must be positive semi-definite. In summary, we have the necessary conditions:

LIP — necessary conditions for minimum

$$A\mathbf{x}^* \geq \mathbf{b} \text{ with } \hat{A}\mathbf{x}^* = \hat{\mathbf{b}} \tag{35}$$

$$Z^T\mathbf{g}(\mathbf{x}^*) = \mathbf{0} \tag{36}$$

$$\mathbf{g}(\mathbf{x}^*) = \hat{A}^T\lambda^* \tag{37}$$

$$\lambda_i^* \geq 0, \quad i = 1, \ldots, t \tag{38}$$

$$Z^T G(\mathbf{x}^*)Z \geq \mathbf{0} \tag{39}$$

As for the equality-constrained case, the second and third of these conditions are actually equivalent.

Algorithms for LIP are more complicated than those for LEP, since the set of constraints which are active at the solution (possibly the empty set) is generally unknown.

Sufficient conditions can also be given for LIP, but the complication of zero Lagrange multipliers means that we must explicitly formulate alternative sets of sufficient conditions.

LIP — sufficient conditions for minimum

$$A\mathbf{x}^* \geq \mathbf{b} \text{ with } \hat{A}\mathbf{x}^* = \hat{\mathbf{b}} \tag{40}$$

$$Z^T\mathbf{g}(\mathbf{x}^*) = \mathbf{0} \tag{41}$$

$$\mathbf{g}(\mathbf{x}^*) = \hat{A}^T\lambda^* \tag{42}$$

$$\lambda_i^* > 0, \quad i = 1, \ldots, t \tag{43}$$

$$Z^T G(\mathbf{x}^*)Z > \mathbf{0} \tag{44}$$

Once again, the second and third of these conditions are equivalent.

When zero Lagrange multipliers are present, the sufficient conditions include extra restrictions on the Hessian matrix to ensure that F displays positive curvature along any perturbation that is binding for all constraints with positive Lagrange multipliers, but may be binding or non-binding for constraints with zero Lagrange multipliers. Let $\hat{A}_+$ contain the coefficients of the active constraints with *positive* Lagrange multipliers and let Z_+ be a matrix whose columns span the nullspace of $\hat{A}_+$. In this case, sufficient conditions for $\mathbf{x}^*$ to be a strong local minimum of LIP are as follows.

LIP — alternative sufficient conditions for minimum

$$A\mathbf{x}^* \geq \mathbf{b} \text{ with } \hat{A}\mathbf{x}^* = \hat{\mathbf{b}} \tag{45}$$

$$Z^T\mathbf{g}(\mathbf{x}^*) = \mathbf{0} \tag{46}$$

$$\mathbf{g}(\mathbf{x}^*) = \hat{A}^T\lambda^* \tag{47}$$

$$\lambda_i^* \geq 0, \quad i = 1, \ldots, t \tag{48}$$

$$Z_+^T G(\mathbf{x}^*) Z_+ > \mathbf{0} \tag{49}$$

Once again, the second and third of these conditions are equivalent.

2.11.3 Nonlinear equality constraints

We define the nonlinear equality-constrained problem NEP:

NEP

minimize $F(\mathbf{x})$

$\mathbf{x} \in R^n$

subject to $\hat{c}_i(\mathbf{x}) = 0, \quad i = 1, \ldots, t.$

In contrast to the LEP case, in which all constraints are of course linear, there is in general *no feasible direction* **p** such that $\hat{c}_i(\mathbf{x}^* + \alpha\mathbf{p}) = 0$ for all sufficiently small α. To retain feasibility, we must move along an *arc*. Such an arc may be specified by the equation $\mathbf{x} = \alpha(\theta)$ with $\alpha(0) = \mathbf{x}^*$. Then, **p** is the tangent to this arc at $\mathbf{x}^*$. The basic necessary condition for optimality of $\mathbf{x}^*$ is that

$$\hat{A}_i(\mathbf{x}^*)^T\mathbf{p} = 0, \qquad \forall i \tag{50}$$

This is equivalent to

$$\hat{A}^T\mathbf{p} = \mathbf{0} \tag{51}$$

$\hat{A}$ is the *Jacobian matrix* of the constraints, defined by

$$a_{ij} = \frac{\partial c_i}{\partial x_j} \tag{52}$$

The vector **p** being orthogonal to the rows of the Jacobian at $\mathbf{x}^*$ is not a sufficient condition for **p** to be tangent to a feasible arc. To illustrate this idea, consider the two constraints

$$\hat{c}_1(x) = (x_1 - 1)^2 + x_2^2 - 1 \tag{53}$$

$$\hat{c}_2(x) = (x_1 + 1)^2 + x_2^2 - 1 \tag{54}$$

The origin is the only feasible point, so no feasible arc exists. But any vector of the form $\mathbf{p} = (0, \delta)^T$ satisfies the Jacobian orthogonality condition.

We need stronger conditions on the constraint functions to ensure that $\mathbf{p}$ is tangent to a feasible arc. Such further assumptions are termed *constraint qualifications*, and they can take many forms. One practical constraint qualification is that the constraint gradients at $\mathbf{x}^*$ are *linearly independent*. This is equivalent to the statement that the matrix $\hat{A}(\mathbf{x}^*)$ has full row rank.

For $\mathbf{x}^*$ to be optimal, F must be stationary along a feasible arc:

$$\nabla F(\alpha(\theta))\,|_{\theta=0} \quad = \quad \mathbf{0} \tag{55}$$

where

$$\hat{A}^T\mathbf{p} \quad = \quad \mathbf{0} \tag{56}$$

If $Z(\mathbf{x}^*)$ is a matrix whose columns form a basis for the nullspace of $\hat{A}$, ie the set of vectors orthogonal to the rows of $\hat{A}$, then we have

$$Z(\mathbf{x}^*)^T\mathbf{g}(\mathbf{x}^*) \quad = \quad \mathbf{0} \tag{57}$$

This condition is analogous to the condition in the linearly constrained case, except that the matrix Z is no longer constant. The vector $Z(\mathbf{x}^*)^T\mathbf{g}(\mathbf{x}^*)$ is termed the *projected gradient* of F at $\mathbf{x}^*$. As before the condition that the projected gradient is zero at $\mathbf{x}^*$, is equivalent to the condition that $\mathbf{g}(\mathbf{x}^*)$ must be a linear combination of the rows of $\hat{A}(\mathbf{x}^*)$.

$$\mathbf{g}(\mathbf{x}^*) \quad = \quad \hat{A}(\mathbf{x}^*)^T\lambda^* \tag{58}$$

for some t-vector of Lagrange multipliers.

Define the *Lagrangian* function as

$$L(\mathbf{x},\lambda) \quad = \quad F(\mathbf{x}) - \lambda^T\hat{\mathbf{c}}(\mathbf{x}) \tag{59}$$

Our necessary condition for optimality of $\mathbf{x}^*$ then can be stated as $\mathbf{x}^*$ *is a stationary point of the Lagrangian* when $\lambda = \lambda^*$.

For a second order necessary condition we define the Hessian of the Lagrangian

$$W(\mathbf{x},\lambda) \quad \equiv \quad G(\mathbf{x}) - \sum_{i=1}^{t} \lambda_i \hat{G}_i(\mathbf{x}) \tag{60}$$

We need

$$\mathbf{p}^T W(\mathbf{x}^*,\lambda^*)\mathbf{p} \quad \geq \quad 0 \tag{61}$$

which is equivalent to

$$Z(\mathbf{x}^*)^T W(\mathbf{x}^*, \lambda^*) Z(\mathbf{x}^*) \quad = \quad 0 \tag{62}$$

This is the *projected Hessian of the Lagrangian function*.

NEP — necessary conditions for minimum

$$\hat{\mathbf{c}}(\mathbf{x}^*) \quad = \quad \mathbf{0} \tag{63}$$

$$Z(\mathbf{x}^*)^T \mathbf{g}(\mathbf{x}^*) \quad = \quad \mathbf{0} \tag{64}$$

$$\mathbf{g}(\mathbf{x}^*) \quad = \quad \hat{A}(\mathbf{x}^*)^T \lambda^* \tag{65}$$

$$Z(\mathbf{x}^*)^T W(\mathbf{x}^*, \lambda^*) Z(\mathbf{x}^*) \quad \geq \quad 0 \tag{66}$$

Once again, the second and third of these conditions are equivalent, and sharpening the inequality on the projected Hessian of the Lagrangian in the last equation leads us to conditions which are sufficient for a constrained minimum:

$$Z(\mathbf{x}^*)^T W(\mathbf{x}^*, \lambda^*) Z(\mathbf{x}^*) \quad > \quad 0 \tag{67}$$

2.11.4 Nonlinear inequality constraints

We define the problem:

NIP

minimize $\quad F(\mathbf{x})$

$\mathbf{x} \in R^n$

subject to $\quad \hat{c}_i(\mathbf{x}) \quad \geq \quad 0, \qquad i = 1, \ldots, m.$

As in the linear case (LIP), we need to identify the active constraints. Only these constraints restrict feasible perturbations at $\mathbf{x}^*$. Again we assume constraint qualification holds. The conditions are given below.

NIP — necessary conditions for minimum

$$\mathbf{c}(\mathbf{x}) > 0 \text{ with } \hat{\mathbf{c}}(\mathbf{x}^*) \quad = \quad \mathbf{0} \tag{68}$$

$$Z(\mathbf{x}^*)^T \mathbf{g}(\mathbf{x}^*) \quad = \quad \mathbf{0} \tag{69}$$

$$\mathbf{g}(\mathbf{x}^*) \quad = \quad \hat{A}(\mathbf{x}^*)^T \lambda^* \tag{70}$$

$$\lambda_i^* \geq 0, \quad i = 1, \ldots, t \tag{71}$$

$$Z(\mathbf{x}^*)^T W(\mathbf{x}^*, \lambda^*) Z(\mathbf{x}^*) \quad \geq \quad 0 \tag{72}$$

Zero Lagrange multipliers cause problems in stating sufficient conditions for NIP, just as in the LIP case. We state first one set of sufficient conditions for NIP which avoids the problem by assuming all Lagrange multipliers are positive:

NIP — sufficient conditions for minimum

$$\mathbf{c}(\mathbf{x}) > 0 \text{ with } \hat{\mathbf{c}}(\mathbf{x}^*) = \mathbf{0} \tag{73}$$

$$Z(\mathbf{x}^*)^T \mathbf{g}(\mathbf{x}^*) = \mathbf{0} \tag{74}$$

$$\mathbf{g}(\mathbf{x}^*) = \hat{A}(\mathbf{x}^*)^T \lambda^* \tag{75}$$

$$\lambda_i^* > 0, \quad i = 1, \ldots, t \tag{76}$$

$$Z(\mathbf{x}^*)^T W(\mathbf{x}^*, \lambda^*) Z(\mathbf{x}^*) > 0 \tag{77}$$

When zero Lagrange multipliers are present, the sufficient conditions include extra restrictions on the Hessian matrix of the Lagrangian function to ensure that F displays positive curvature along any feasible arc that is binding for all constraints with positive Lagrange multipliers, but may be binding or non-binding for constraints with zero Lagrange multipliers. Let $\hat{A}_+(\mathbf{x}^*)$ contain the coefficients of the active constraints with *positive* Lagrange multipliers and let $Z_+(\mathbf{x}^*)$ be a matrix whose columns span the nullspace of $\hat{A}_+(\mathbf{x}^*)$. In this case, sufficient conditions for $\mathbf{x}^*$ to be a strong local minimum of NIP are as follows.

NIP — alternative sufficient conditions for minimum

$$\mathbf{c}(\mathbf{x}) > 0 \text{ with } \hat{\mathbf{c}}(\mathbf{x}^*) = \mathbf{0} \tag{78}$$

$$Z(\mathbf{x}^*)^T \mathbf{g}(\mathbf{x}^*) = \mathbf{0} \tag{79}$$

$$\mathbf{g}(\mathbf{x}^*) = \hat{A}(\mathbf{x}^*)^T \lambda^* \tag{80}$$

$$\lambda_i^* \geq 0, \quad i = 1, \ldots, t \tag{81}$$

$$Z_+(\mathbf{x}^*)^T W(\mathbf{x}^*, \lambda^*) Z_+(\mathbf{x}^*) > 0 \tag{82}$$

2.12 Existing algorithms for continuous nonlinear optimization

Generally speaking, function comparison methods are a poor choice when compared with methods making use of derivative information. They should be chosen only when derivatives are very difficult to compute, unreliable or not available at all. For some problems, the objective function is nonsmooth, and here function comparison methods may be the only ones available. For some very useful practical advice in this regard, the book by Gill, Murray and Wright [24] is recommended.

2.12.1 1-Dimensional methods

Some of the standard methods for minimization of a function of a single variable are bisection, Brent's method, Fibonacci search, golden section search, quadratic interpolation, and Newton's method [24, pp82–92]. Brent's original work is collected in his monograph [4] and source code for modern implementations in FORTRAN and Pascal can be found in Press, Flannery, Teukolsky and Vetterling [75]. First and second derivative information must be available to use Newton's method, whereas other techniques mentioned use only function values.

2.12.2 A model descent algorithm schema (unconstrained)

We consider now a very general specification for a *descent* method for linearly-constrained minimization which embodies the nullspace matrix Z and a line search subalgorithm. Figure 14 has the details.

```
k := 0;
x_0 := feasible initial estimate;
converged:= | f(x) |< ε
givingup:=false
while not (converged or givingup) do
        compute p_z                                   (*search direction*)
        p_k := Z p_z
        compute α_k such that  F(x_k + α_k p_k) < F(x_k)
        x_{k+1} := x_k + α_k p_k                      (*line search*)
        k := k + 1                                    (*update soln vector*)
        givingup:=(k > maxiterations)
endwhile
```

Figure 14 Descent schema

Notes

1. Feasibility is preserved by this class of algorithms.
2. $\mathbf{p}_k$ should be a descent direction, ie $\mathbf{g}_k^T Z \mathbf{p}_z < 0$.
3. The objective function must sustain a *sufficient decrease* at each iteration to give some hope of convergence in practice in a reasonable number of iterations. It is *not* sufficient to merely require that $F(x_{k+1}) < F(x_k)$. It is easy to devise examples where a decreasing sequence is generated but converges too slowly to be of any practical computational use. For further details, see eg Gill, Murray and Wright [24], pp100–102, 324–325.

The basic Newton method uses the quadratic model which is accurate for any smooth function sufficiently close to a local minimum. This method has remarkable convergence properties provided we are "sufficiently close". The best methods are variations on Newton's method, however to be effective the Hessian matrix needs to be positive definite.

Many alternative methods exist for choosing the descent direction **p**. Some well-known ones are steepest descent, Newton's method, the class of quasi-Newton methods, and the conjugate gradient method. We briefly discuss each of these in turn.

Steepest descent (1st derivative method)

The oldest method for minimization is that of *steepest descent*. At each iterate $\mathbf{x}_k$, we follow the negative gradient vector which is guaranteed to be a descent direction unless we are already at a stationary point. This method is much discussed in textbooks on multivariate minimization but is really a very poor method for machine implementation. The first published account is that of Cauchy, *circa* 1847 [7].

Conjugate gradient method (1st derivative method)

A generalization of the idea of an orthogonal basis for a vector space is used to generate a sequence of *non-interfering* search directions $\mathbf{p}_k$. This method was originally developed as a method for solving linear equations by Hestenes and Stiefel (1952) [36], and has the property that any quadratic function of n variables that has a minimum can be minimized in n steps, one in each of the conjugate directions, and the order in which the directions are applied does not matter. Its extension to nonlinear problems was the work of Fletcher and Reeves (1964) [19]. The conjugate gradient method is used commonly for large-scale problems when methods based on matrix factorizations are not possible because the matrix is too large or dense. This is the approach adopted for MINOS [64].

Newton's method (2nd derivative method)

Newton's method is based on the simple quadratic model, viz. that any smooth function looks like a quadratic with positive definite Hessian in the neighbourhood of a minimum. The simple quadratic function

$$F(\mathbf{x}+\mathbf{p}) \quad = \quad F(\mathbf{x}) \quad + \quad \mathbf{g}(\mathbf{x})^T\mathbf{p} \quad + \quad \frac{1}{2}\mathbf{p}^T G(\mathbf{x})\mathbf{p} \tag{83}$$

is used to model an arbitrary smooth function. It is easy to show that the quadratic is minimized when **p** satisfies

$$G_k \mathbf{p}_k = -\mathbf{g}_k \tag{84}$$

where $\mathbf{g}_k$ is the gradient vector at the current iterate $\mathbf{x}_k$ and G_k is the corresponding Hessian matrix.

A positive definite Hessian guarantees a unique solution $\mathbf{p}_k$, which is termed the *Newton direction*. If the quadratic form is *positive definite*, ie the (constant) Hessian $G > 0$, then exactly one iteration is required for the Newton direction to find the minimum.

If G is not positive definite then there exist various strategies for modifying the computed Hessian to find a direction to decrease F. Such *modified Newton* techniques rely on matrix factorizations to check the positive-definiteness of the Hessian.

The basic Newton method has very strong local convergence properties, provided we have positive definiteness, and this should be clear from its behaviour on a quadratic function, and the Taylor expression for f near a minimum. It is a second derivative method, which means that sufficient conditions for optimality can be checked. We need to be aware that it can also fail in a rather spectacular manner since the quadratic model may not be accurate away from the minimum.

A further variation of the basic Newton idea is to use finite differences of the gradient vector to arrive at an approximate Hessian. Such methods are termed *finite-difference Newton methods*. Proper adaptive implementations of such methods are generally as robust and rapidly-convergent as Newton-type methods which use the full second derivative information of the Hessian.

Quasi-Newton methods (2nd derivative methods)

These are based on the idea of building up curvature information as the iterations of a descent method proceed, using function value and gradient vector. Newton's method uses the exact Hessian and obtains the curvature at a single point. Quasi-Newton methods are based on the fact that an approximation to the curvature of a nonlinear function can be computed without explicitly forming the Hessian matrix.

The two well-known rank-two updates to the sequence of matrices approximating the Hessian are those of Davidon-Fletcher-Powell (DFP), and Broyden, Fletcher, Goldfarb, Shanno (BFGS). Good descriptions can be found in Gill, Murray and Wright [24]. The MINTO code of Murtagh, on which the present work is based, uses BFGS.

Active set methods for the linearly constrained problem

The approach used by Murtagh and Saunders in MINOS [62] is based on the idea of *superbasic variables*. At any given stage in the minimization, the number of superbasic variables n_S is a measure of dimensionality of the subspace formed by the intersection of the active constraints. Superbasic variables correspond to degrees of freedom for the search, ie to *free variables* which are currently neither basic (dependent) nor at their bounds (fixed). Superbasic variables are free to vary between their simple bounds, subject to maintaining feasibility of the current set of basic variables.

The search may be viewed as having two interwoven components: the problem of finding the right set of active constraints, and the problem of minimizing on that active set. Details are given in Chapter 4.

Chapter 3

NLIP Literature

NLIP is an intrinsically hard problem. When embarking on a research project for nonlinear integer programming, it is of little comfort to read statements such as the following from Scarf:

> *In the language of complexity theory, integer programming is what is known as an NP-complete problem: If there is a polynomial algorithm for integer programming, then virtually every problem we can think of is easy to solve—a quite unlikely possibility.*
>
> Scarf [82]

Note that the statement just cited from Scarf refers to *linear* integer programming! NLIP is certainly harder than integer LP.

Conventional methods of solving NLIP are based on various sequential linearizations of the problem and some variations on the basic branch and bound strategy, however in some cases, special advantage may be taken of structure in the problem under consideration. Indeed, there are reported instances where problems with hundreds of thousands of integer variables have been solved. One such example is that of Ryan [80] who reports the solution of an airline crew scheduling problem involving 200,000 binary variables on a microcomputer.

The general problem of nonlinear integer programming (NLIP), especially large-scale NLIP, is widely recognized as a "tough nut to crack". As with most domains of mathematics, nonlinear problems are often solved by generating a sequence of solutions to linear problems which in some sense approximate the original nonlinear problem. This is certainly the case with NLIP. In the following sections, we outline some of the most recent research into algorithms for the general NLIP problem.

3.1 General

A good starting point for NLIP is linear integer programming, and the best modern reference for integer and combinatorial optimization is the excellent monograph of Nemhauser and Wolsey [68]. This book is a comprehensive treatise of the subject, and while the authors admit that "it does not come close to covering all the literature" [*op cit (vii)*], it is nevertheless a weighty tome, and an indispensable reference for researchers in the field. In this work, the authors develop among other topics, the theory of **valid inequalities**, which are used to characterize the feasible set of an optimization problem; in particular those of most interest are the ones which are *active* at the final solution. The theory of valid inequalities was founded by Gomory in the late 1950s and early 1960s, and Nemhauser and Wolsey present in this book algorithms for generating all valid inequalities for a given problem. They also give a thorough discussion of **cutting-plane** algorithms (the systematic addition of linear constraints which "slice off" portions of the feasible set not containing integer-feasible points), also pioneered by Gomory. The book is replete with examples and useful exercises; in fact it is a goldmine of information on the state-of-the-art of integer and combinatorial optimization in the late 1980s, although notably absent is any treatment of NLIP.

3.2 The outer approximation algorithm of Duran and Grossmann

In 1986, Duran and Grossmann [17] published details of an outer approximation algorithm to solve MINLP. The approach involves the construction and solution of an alternating sequence of integer linear programming master problems, and inner nonlinear programming subproblems. The current subproblem is solved with the integer variables held fixed, and the master problem is formed by linearizing the functions at the solution of the subproblem.

The Duran and Grossmann method uses decomposition principles to exploit problem structure, which is assumed to be of the following form: linear in the integer variables and convex in the nonlinear portions of the objective and constraint functions (which only involve the so-called *nonlinear* variables). The general form of the class of problems addressed by this method is

minimize

$$c^T y \quad + \quad f(x)$$

subject to

$$g(x) \quad + \quad By \leq 0$$

$$x \in X \subseteq R^n$$

$$y \in U \subseteq R_+^m$$

The nonlinear function $f : R^n \to R$ and the vector functions $\mathbf{g} : R^n \to R^p$ are required to be continuously differentiable and convex on appropriate compact domains. As is conventional, the domain U of the integer variables is assumed to be some finite discrete set; most commonly the unit hypercube $Y = \{0, 1\}^m$.

The main ideas of the method as summarized in the original paper by Duran and Grossmann are as follows. The linearity of the discrete variables allows independent characterization of the continuous and discrete feasible search spaces of the problem. The continuous space may be expressed as an intersection of a finite collection of compact convex regions, each of which is parametrized by distinct values of the discrete variables. Outer-approximation of the convex sets by intersection of supporting half-spaces is used to define a master mixed-integer LP. The authors compare their method with the generalized Benders decomposition method and note that while both techniques make use of the mathematical tools of projection, outer-approximation and relaxation, their method tended to produce better lower bounds on the optimal objective value.

Early test results reported by Duran and Grossmann show promise for the method, which the authors indicate should particularly suit problems in which the NLP subproblems are expensive to solve. Fletcher, Leyffer and co-workers [44] at the University of Dundee are presently working on similar ideas to those presented by Duran and Grossmann [17].

3.3 The method of Mawengkang and Murtagh

Some recent work (1985/6) by Mawengkang and Murtagh [49] and Murtagh [59] has allowed progress to be made on a rather general and commonly-occurring class of NLIPs, namely those in which the proportions of integer variables and nonlinear variables are both small. In a 1986 paper [49], the authors describe experience in applying the MINOS (see [64, 62]) code to large nonlinear (both nonlinear objective and nonlinear constraints) integer programming problems. Their work was based on an extension of the constrained search approach used in MINOS, and the application areas considered were an instance of the quadratic assignment problem (QAP) and a natural gas pipeline network design problem. Since QAPs of order greater than about 10 or 15 are notoriously expensive to solve using approaches such as branch-and-bound, the authors adopted a direct search approach which treats a subset of the integer variables in a similar fashion to the superbasic variables of the MINOS algorithm. Just as the superbasic variables in MINOS allow the extra degrees of freedom needed for a nonlinear problem, certain of the integer variables were allowed to vary only in discrete steps during the search, thus maintaining integer feasibility. In fact, the present thesis involves an extension of the approach used by Mawengkang and Murtagh, and the reader is referred in particular to Chapter 5 and Chapter 6.

The paper [60] by Murtagh extends the work described in the previous paragraph, and also forms the starting point for the present dissertation. Murtagh's approach, as given in [60], is elaborated and analysed in the present Chapter 5.

3.4 Other approaches

Linearization techniques have been known for many years. Any IP can be reformulated as a $0-1$ problem (see for example Nemhauser and Wolsey [68]). If the objective and constraints are polynomials then the problem can be reformulated as a linear integer program (Garfinkel and Nemhauser (1972) [21]. Such techniques introduce many new $0-1$ variables and many new constraints. Extra constraints are not normally a problem, however each extra $0-1$ variable has, broadly speaking, the potential to double computation time.

Two alternative well-known methods for NLIP problem are Benders' decomposition method and Bellman's dynamic programming. These approaches are well-documented in the

literature and it serves little purpose to repeat them here. The interested reader may wish to consult for example Nemhauser and Wolsey [68] for modern treatments of either of these topics.

The thesis of Myers (1984) [66] also considers comparative branch-and-bound strategies, in conjunction with the design of Lagrangian relaxation and subgradient optimization strategies for linearly-constrained mixed-integer NLIPs. One of the major conclusions of this work was to favour a branching strategy which maximized the product of integer-infeasibility and component of the objective gradient vector, taken across all integer variables which are not currently integer-feasible.

Gupta & Ravindran (1985) [32] considered $3^3 = 27$ separate heuristics for branch and bound approaches to solving convex nonlinear integer programs. Their computational experience indicated that branching on the variable with greatest integer-infeasibility seemed to be best, but since their largest problem had only eight integer variables, one feels that such results are inconclusive at best.

In 1984, Balas and Mazzola [1] published an influential paper proposing a linearization approach involving the replacement of general nonlinear functions of binary 0–1 variables appearing in inequality constraints with a family of equivalent linear inequalities. This technique has the advantage of linearization without introducing additional variables.

Hansen's 1979 survey paper [34] gives a good summary of methods for 0–1 NLIP at that time, as well as examples of applications and an extensive bibliography. He concluded in particular that only a small proportion of the many algorithms that had been proposed up to that time had actually been implemented and tested; that network flow algorithms are useful for certain classes of quadratic 0–1 programs; a standard set of test problems would allow meaningful benchmark tests to be applied; and that Boolean formulations of 0–1 NLIPs can be useful for suggesting algorithms or proof techniques.

Several other approaches have been described in the literature, and good reviews of work prior to 1981 are found in Gupta and Ravindran [31] and Cooper [9].

3.5 Existing NLIP software—a necessarily brief survey

Many optimization packages have been written, but very few indeed for NLIP. The MINOS code of Murtagh and Saunders [64] provides nonlinear objective as well as linear constraints

capability, and MINOS/AUGMENTED adds nonlinear constraints capability, but, as noted above, the landscape is much more sparse when we examine available software for general-purpose *nonlinear integer* optimization—until recently the only general-purpose large-scale nonlinear integer package known to the present author is Murtagh's MINTO [58]. However, some recent work by Viswanathan and Grossmann [92] has resulted in the development of software for the solution of MINLP problems. The implementation of the new algorithm, which is an extension of the outer-approximation algorithm reported by Duran and Grossmann [17], has been done for several hardware platforms, and within the framework of the GAMS [5] system (the nonlinear step is done by MINOS 5.2 [65]). Also recently brought to the author's attention is the work of Paules and Floudas [72], which also uses GAMS as a base to allow expression and solution of MINLP problems. It works to "provide exact syntactic statement of algorithmic solution procedures" and caters for "completely general automated implementation of many well known algorithms including Generalized Benders Decomposition, the Outer Approximation / Equality Relaxation and Dantzig-Wolfe Decomposition". The APROS system specializes in catering for algorithms which involve some kind of decomposition technique and which require extensive communication of data between various subproblems which may be generated during the solution process.

In recent years, microcomputers have become so powerful that many mainframe software systems have been ported to personal computer platforms. The paper [93] by Wasil, Golden and Liu gives a reasonably up-to-date comparison of six PC-based packages which handle nonlinear optimization problems, although notably, none has integer capability.

The MINTO code of Murtagh [62] extends MINOS to add integer capability. MINTO is a powerful general-purpose mixed-integer optimizer which caters for nonlinear objective and/or constraints. It uses the MINOS or MINOS/AUGMENTED algorithm to arrive at a locally optimal point with respect to the continuous relaxation, and then switches to the branch-and-bound technique in order to ultimately satisfy the integer requirements. The MINOS engine is then reused to recursively solve the newly-generated subproblems arising from the branch-and-bound process. By contrast, the present work uses the MINTO algorithm as a starting point, and allows direct search for integer feasibility and local optimality once the continuous relaxation has been solved by the MINOS engine. Branch-and-bound may then be used as a last resort if one of the direct search mechanisms fails to achieve integer feasibility. MINOS, MINOS/AUGMENTED and MINTO are discussed in greater detail in Chapter 4, and the new direct-search methods in Chapter 6.

Chapter 4

MINOS[1] and its descendants

MINOS is a FORTRAN code designed to solve large optimization problems in which the objective may be nonlinear and the constraints linear, in addition to simple bounds.

In this chapter we present a summary of the fundamental ideas and equations underlying the steps of the MINOS algorithm of Murtagh and Saunders, as reported in their seminal 1978 paper *Large-Scale Linearly Constrained Optimization* [62]. This is the principal source for the reader requiring further details of both the theory and implementation of the original MINOS. As outlined in an earlier chapter, the algorithm has been extended by the original authors and co-workers to handle both nonlinear constraints (see chapter 4.3) and integer restrictions on the variables. The paper [61] by Murtagh and Saunders is the original published account of the nonlinear constraints development, while an excellent encapsulated summary of inner workings of the extended MINOS (with nonlinear constraints capability) can be found in Gill, Murray, Saunders and Wright [26].

Problem to be solved by MINOS

MINOS is a particular implementation of the *reduced-gradient* algorithm of Wolfe [96]. The class of problems solved by MINOS is the following:

minimize

$$F(\mathbf{x}) \quad = \quad f(\mathbf{x}^N) \quad + \quad \mathbf{c}^T\mathbf{x} \tag{85}$$

1 Modular In-core Nonlinear Optimization System

subject to

$$A\mathbf{x} = \mathbf{b} \tag{86}$$

$$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \tag{87}$$

The matrix A is $m \times n$ with $m \leq n$. It contains the columns of the $m \times m$ identity matrix I_m as a submatrix, corresponding to the problem formulation containing a full set of slack variables.

Such problems contain constraint matrices A which are typically highly-sparse and many also are only *slightly nonlinear*, in the sense that the variables occurring nonlinearly in the objective, denoted $\mathbf{x}^N$ above, form only a small percentage of the total variables.

The approach used by MINOS is an extension of the revised simplex method for linear programming (see for example Dantzig [12], Murtagh [57] or Gill, Murray and Wright [24]), and as such draws on a vast body of refinements that have been made over two or three decades since the original work of Dantzig [12]. Such advancements include upper and lower bounds on all variables and stable recurrences for update of a sparse factorization of the basis matrix.

The fundamental result on which the revised simplex method for linear programming (LP) is based is of course not true for nonlinear programming (NLP). For LP we know that at an optimal solution, at least $n - m$ variables are at a bound. Geometrically, we are at an extreme point, or boundary, of the feasible set. Even for constrained NLP, a locally-optimal point can easily be an interior point. An excellent discussion of necessary and sufficient conditions for constrained optimization (the so-called *Kuhn-Tucker* conditions) divided into the four categories of linear/nonlinear and equality/inequality is given in Gill, Murray and Wright [24, chapters 3.3 and 3.4], although here we shall discuss linear constraints only. The conditions have been summarized in the present section 2.11.

Nevertheless, for such a powerful and theoretically elegant technique as the revised simplex method, it is possible to extend its usefulness to nonlinear problems by redefining the simplex partition. Since the number of variables at bound (nonbasic) at a locally optimal point is not known, we introduce a third element of the partition, namely the *superbasic* variables (the terminology is that of Murtagh and Saunders [62]). In other words, in addition to the conventional revised simplex partition into columns corresponding to the so-called *basic* and *nonbasic* variables, MINOS employs a third component of the partition—that which

corresponds to the *superbasic* variables. Superbasics are essentially free variables and their number gives the dimension of the current search space. They are free to vary between simple bounds subject to maintaining feasibility of basics which are dependent on them.

As clearly pointed out by Murtagh and Saunders [62], an important advantage of the concept of basic solutions is the emphasis given to upper and lower bounds. The constraint matrix A is assumed to contain columns corresponding to the identity matrix, which in turn correspond to a full set of slack variables. Inequality constraints are easily accommodated in this manner, as is the problem of finding an initial feasible solution; this is just the conventional *phase one* of the revised simplex method. It should be noted that the so-called *artificial* variables used in phase one of the revised simplex method are simply slacks with upper and lower bounds of zero. They are of no special significance computationally, however it will be seen later that their presence in the basis causes problems for certain of the direct search techniques of Chapter 6.

4.1 Fundamental equations for MINOS

Assuming then we have the partition into basics, superbasics and nonbasic, the general linear constraints take the form:

$$A\mathbf{x} \quad = \quad [B \qquad S \qquad N] \begin{bmatrix} x_B \\ x_S \\ x_N \end{bmatrix} \quad = \quad \mathbf{b} \tag{88}$$

It is assumed that the nonlinear portion of the objective, ie $f(\mathbf{x}^N)$ is sufficiently smooth so that a Taylor series representation can be written:

$$f(\mathbf{x}+\Delta\mathbf{x}) \quad = \quad f(\mathbf{x}) \quad + \quad \mathbf{g}(\mathbf{x})^T\Delta\mathbf{x} \quad + \quad \frac{1}{2}\Delta\mathbf{x}^T G(\mathbf{x}+\gamma\Delta\mathbf{x})\Delta\mathbf{x} \tag{89}$$

where $0 \le \gamma \le 1$ and $G(\mathbf{x}+\gamma\Delta\mathbf{x})$ is the Hessian matrix of second partial derivatives evaluated at some point between $\mathbf{x}$ and $\mathbf{x}+\Delta\mathbf{x}$.

Given the partition into basic, superbasic and nonbasic variables, and assuming that $f(\mathbf{x})$ is a quadratic form, we have the following equations which must hold for a constrained stationary point.

$$\begin{bmatrix} B & S & N \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_B \\ \Delta\mathbf{x}_S \\ \Delta\mathbf{x}_N \end{bmatrix} = \mathbf{0} \tag{90}$$

ie the step remains on the surface given by the intersection of the active constraints

$$\begin{bmatrix} \mathbf{g}_B \\ \mathbf{g}_S \\ \mathbf{g}_N \end{bmatrix} + G \begin{bmatrix} \Delta\mathbf{x}_B \\ \Delta\mathbf{x}_S \\ \Delta\mathbf{x}_N \end{bmatrix} = \begin{bmatrix} B^T & 0 \\ S^T & 0 \\ N^T & I \end{bmatrix} \begin{bmatrix} \pi \\ \lambda \end{bmatrix} \tag{91}$$

From these we derive

$$\begin{aligned} \Delta\mathbf{x}_N &= 0 \\ \Delta\mathbf{x}_B &= -W\Delta\mathbf{x}_S \\ W &= B^{-1}S \end{aligned} \tag{92}$$

and

$$\Delta\mathbf{x} = \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta\mathbf{x}_S = Z\,\Delta\mathbf{x}_S \tag{93}$$

Gill and Murray have defined a class of algorithms in which the search direction along the surface of active constraints is characterized as being in the range space of a matrix Z which is orthogonal to the current matrix of linearized constraint normals. If $Ax = b$ is the current set of $n-s$ active constraints, then Z is an $n \times s$ matrix such that $AZ = 0$. The only further requirement on Z is that it have full column rank; thus several degrees of freedom are still available for the choice of Z. The form used by the MINOS procedure corresponds to the extended simplex partition in which the superbasic variables form a new component of the partition. This leads to the choice of Z given by (93). The reader interested in further details may consult the paper by Murtagh & Saunders [62].

Premultiplication of (93) by the matrix

$$\begin{bmatrix} I & 0 & 0 \\ -W^T & I & 0 \\ 0 & 0 & I \end{bmatrix} \tag{94}$$

leads to some useful relationships. We can estimate the Lagrange multipliers for the general constraints from the first row partition:

$$B^T\pi \quad = \quad \mathbf{g} \quad + \quad [I \quad 0 \quad 0] \quad G \quad \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta\mathbf{x}_S \tag{95}$$

When $\mathbf{x}$ is stationary, ie $\Delta\mathbf{x}_S \quad = \quad \mathbf{0}$, we obtain

$$B^T\pi \quad = \quad \mathbf{g}_B \tag{96}$$

Thus π is analogous to the pricing vector in the revised simplex algorithm.

The third row partition yields:

$$\lambda \quad = \quad \mathbf{g}_N \quad - \quad N^T\pi \quad + \quad [0 \quad 0 \quad I] \quad G \quad \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta\mathbf{x}_S \tag{97}$$

which when $\Delta\mathbf{x}_S \quad = \quad 0$ leads to:

$$\lambda \quad = \quad \mathbf{g}_N \quad - \quad N^T\pi \tag{98}$$

which is our vector of reduced costs in LP.

From the second row partition we get an expression for $\Delta\mathbf{x}_S$:

$$[-W^T \quad I \quad 0] \quad G \quad \begin{bmatrix} -W \\ I \\ 0 \end{bmatrix} \Delta\mathbf{x}_S \quad = \quad -\mathbf{h} \tag{99}$$

in which

$$\mathbf{h} \quad = \quad [-W^T \quad I \quad 0]\,\mathbf{g} \quad = \quad \mathbf{g}_S - W^T\mathbf{g}_B \quad = \quad \mathbf{g}_s - S^T\pi \tag{100}$$

4.2 Steps of the MINOS algorithm

The following gives a broad summary of the major computational steps of MINOS, and is summarized from Murtagh and Saunders' 1978 paper [63]. For step 0, no *activity* is actually stated, merely that certain quantities are precomputed, and enter the method as initial values.

Step 0: Initial conditions.

We assume that the following quantities are available:

(a) a feasible vector x satisfying $[B \quad S \quad N]\mathbf{x} = \mathbf{b}$ and $\mathbf{l} \le \mathbf{x} \le \mathbf{u}$

(b) the corresponding function value $f(\mathbf{x})$ and gradient vector $\mathbf{g}(\mathbf{x}) = [\mathbf{g}_B \quad \mathbf{g}_S \quad \mathbf{g}_N]^T$

(c) the number of superbasic variables, s, $(0 \le s \le n - m)$

(d) a factorization, LU of the $m \times m$ basis matrix B

(e) a factorization $R^T R$ of a quasi-Newton approximation to the $s \times s$ reduced Hessian matrix $Z^T G Z$.

(f) a pricing vector π, being the solution of $B^T \pi = \mathbf{g}_B$

(g) the reduced gradient vector $\mathbf{h} = \mathbf{g}_S - S^T \pi$

(h) convergence tolerances TOLRG and TOLDJ

Step 1: Test for convergence in the current subspace.

If $|h| > TOLRG$ then go to step 3.

Step 2: Price, ie estimate Lagrange multipliers, add one superbasic.

(a) calculate $\lambda = \mathbf{g}_N - N^T \pi$

(b) select $\lambda_{q_1} < -TOLDJ$, the largest elements of λ corresponding to variables at their lower (upper) bound. If none, STOP, the Kuhn-Tucker necessary conditions for an optimal solution are satisfied.

(c) otherwise

(i) choose $q = q_1$ or $q = q_2$ corresponding to $|\lambda_q| = \max(|\lambda_{q_1}|, |\lambda_{q_2}|)$

(ii) add $\mathbf{a}_q$ as a new column of S

(iii) add λ_q as a new element of $\mathbf{h}$

(iv) add a suitable new column to R

(d) increment s

Step 3: Compute the direction of search, $\mathbf{p} = Z\mathbf{p}_S$

(a) solve $R^T R\mathbf{p}_S = -\mathbf{h}$

(b) solve $LU\mathbf{p}_B = -S\mathbf{p}_S$

(c) set $\mathbf{p} = [\mathbf{p}_B \quad \mathbf{p}_S \quad \mathbf{p}_N]^T$

Step 4: ratio test (CHUZR)

(a) find $\alpha_{\max} \geq 0$, the greatest value α for which $\mathbf{x} + \alpha\mathbf{p}$ is feasible

(b) if $\alpha_{\max} = 0$ then go to step 7

Step 5

(a) find α, an approximation to α^*, where

$$f(\mathbf{x} + \alpha^*\mathbf{p}) = \min_{0 \leq \theta \leq \alpha_{\max}} f(\mathbf{x} + \theta\mathbf{p})$$

(b) change $\mathbf{x}$ to $\mathbf{x} + \alpha\mathbf{p}$ and set f and $\mathbf{g}$ to their values at the new α.

Step 6: Compute the reduced gradient vector $\bar{\mathbf{h}} = Z^T\mathbf{g}$

(a) solve $U^T L^T = \mathbf{g}_B$

(b) compute new reduced gradient $\bar{\mathbf{h}} = \mathbf{g}_S - S^T\pi$.

(c) modify R in accordance with a quasi-Newton recursion on $R^T R$ using $\alpha, \mathbf{p}_S$ and the change in the reduced gradient $\bar{\mathbf{h}} - \mathbf{h}$.

(d) set $\mathbf{h} = \bar{\mathbf{h}}$.

(e) if $\alpha < \alpha_{\max}$ then go to step 1. No new constraint was encountered so we remain in the current subspace.

Step 7: Change basis if necessary; delete one superbasic.

We assert that $\alpha = \alpha_{\max}$ and for some $p\,(0 \le p \le m+s)$, a variable corresponding to the pth column of $[B \quad S]$ has reached a simple bound.

(a) if a *basic* variable hit is bound $(0 < p \le m)$ then

(i) swap the pth and qth columns of B and S respectively, and correspondingly, the components of $\mathbf{x}_B$ and $\mathbf{x}_S$. The column of S with index q must be such that the new B is nonsingular.[1]

(ii) modify L, U, R, π to reflect this change in B.

(iii) compute the new reduced gradient, $\mathbf{h} = \mathbf{g}_S - S^T\pi$.

(iv) go to (c)

(b) otherwise, a *superbasic* variable hit its bound ($m < p \le m+s$). Define $q = p - m$.

(c) make the qth variable in S nonbasic at the appropriate bound, ie

(i) delete qth columns of S and R, and correspondingly, the qth components of $\mathbf{x}_S$ and $\mathbf{h}$.

---

1 This is also required for the direct search methods of chapter 6 in which a $B \leftrightarrow S$ pivot is done.

(ii) restore R to triangular form.

(d) decrement s and return to step 1.

4.3 MINOS/AUGMENTED

This section summarizes the major features and capabilities of the MINOS/AUGMENTED code of Murtagh and Saunders [61]. This system uses a projected Lagrangian algorithm to extend the MINOS capabilities to handle nonlinear constraints. The system is specifically designed for large-sparse constraint sets, which in most instances, contain a large subset of purely-linear constraints.

A projected Lagrangian method for the nonlinearly constrained problem

Consider the problem:

minimize

$$f^0(\mathbf{x}) \tag{101}$$

subject to

$$\mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{102}$$

$$\mathbf{l} \le \mathbf{x} \le \mathbf{u} \tag{103}$$

To solve this problem, we can solve a sequence of linearly constrained problems in which a linear approximation is used in place of the nonlinear constraints $\mathbf{f(x)} = \mathbf{0}$ and higher-order terms are adjoined to the objective function to form a *Lagrangian function*. The problem then becomes

minimize

$$L(\mathbf{x},\mathbf{x}_k,\lambda_k) \quad = \quad f(\mathbf{x}) \quad - \quad \lambda_k^T(\mathbf{f} - \tilde{\mathbf{f}}) \tag{104}$$

subject to

$$\tilde{\mathbf{f}} = \mathbf{0} \tag{105}$$

$$\mathbf{l} \le \mathbf{x} \le \mathbf{u} \tag{106}$$

where $\tilde{\mathbf{f}}$ is the linear approximation to $\mathbf{f}$ at $\mathbf{x}_k$, ie

$$\tilde{\mathbf{f}}(\mathbf{x},\mathbf{x}_k) \quad = \quad \mathbf{f}_k \quad + \quad J_k(\mathbf{x} - \mathbf{x}_k) \tag{107}$$

where $\mathbf{f}_k$ and J_k are respectively the constraint vector and *Jacobian* matrix of f, both evaluated at $\mathbf{x} = \mathbf{x}_k$.

The algorithmic details may be summarised as indicated in Figure 15.

$k := 0$
x:= initial estimate
$\lambda_0 :=$ initial estimate
$\rho :=$ positive penalty parameter
repeat
  solve the linearly constrained problem
  minimize

$$L(\mathbf{x},\mathbf{x}_k, \lambda_k, \rho) = f^0(\mathbf{x}) - \lambda_k^T(\mathbf{f} - \tilde{\mathbf{f}}) + \tfrac{1}{2}\rho(\mathbf{f} - \tilde{\mathbf{f}})^T(\mathbf{f} - \tilde{\mathbf{f}})$$

  subject to $\tilde{\mathbf{f}} = \mathbf{0}$ and $\mathbf{l} \le \mathbf{x} \le \mathbf{u}$
  to obtain x_{k+1} and λ_{k+1}
  if not converged **then**
    relinearize constraints at $\mathbf{x}_k$
    $k := k + 1$
  endif
until converged

Figure 15 A projected Lagrangian method

For the ensuing discussion, and following Murtagh and Saunders [61], we assume that the nonlinearly constrained problem as defined by equations (101)–(103) can be expressed in the following form, in which the linear components are explicitly shown:

$$\underset{[\mathbf{xy}]^T \in R^n}{\text{minimize}} \quad f^0(\mathbf{x}) + \mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{y} \tag{108}$$

$$\text{subject to} \quad \mathbf{f}(\mathbf{x}) + \mathbf{A}_1\mathbf{y} = \mathbf{b}_1 \qquad (m_1 \text{ rows})$$

$$\mathbf{A}_2\mathbf{x} + \mathbf{A}_3\mathbf{y} = \mathbf{b}_2 \qquad (m_2 \text{ rows})$$

$$\mathbf{l} \le \mathbf{x} \le \mathbf{u} \qquad (m = m_1 + m_2)$$

The solution method of MINOS/AUGMENTED consists of a sequence of *major iterations*, each of which involves a linearization of the nonlinear constraints at some point $\mathbf{x}_k$, corresponding to a first-order Taylor series approximation:

$$f^i(\mathbf{x}) \quad = \quad f^i(\mathbf{x}_k) \quad + \quad \mathbf{g}^i(\mathbf{x}_k)^T \quad + \quad \mathrm{O}(|\,\mathbf{x} - \mathbf{x}_k\,|^2) \tag{109}$$

Defining

$$\tilde{\mathbf{f}} \quad = \quad \mathbf{f}_k \quad + \quad J_k(\mathbf{x} - \mathbf{x}_k) \tag{110}$$

we then have

$$\mathbf{f} - \tilde{\mathbf{f}} \quad = \quad (\mathbf{f} - \mathbf{f}_k) \quad - \quad J_k(\mathbf{x} - \mathbf{x}_k) \tag{111}$$

At the *k*th major iteration of the algorithm, the following linearly constrained problem is solved:

minimize

$$\mathrm{L}(\mathbf{x},\mathbf{y},\mathbf{x}_k, \lambda_k, \rho) \quad = \quad f^0(\mathbf{x}) + \mathbf{c}^T\mathbf{x} + \mathbf{d}^T\mathbf{y} - \lambda_k^T(\mathbf{f} - \tilde{\mathbf{f}}) + \frac{1}{2}\rho(\mathbf{f} - \tilde{\mathbf{f}})^T(\mathbf{f} - \tilde{\mathbf{f}}) \tag{112}$$

subject to

$$[\mathbf{x} \quad \mathbf{y}]^T \quad \in \quad R^n \tag{113}$$

$$\tilde{\mathbf{f}} + \mathbf{A}_1\mathbf{y} = \mathbf{b}_1 \tag{114}$$

$$\mathbf{A}_2\mathbf{x} + \mathbf{A}_3\mathbf{y} = \mathbf{b}_2 \tag{115}$$

$$\mathbf{l} \le \mathbf{x} \le \mathbf{u} \qquad (m = m_1 + m_2) \tag{116}$$

4.4 MINTO

MINTO is an extension of MINOS/AUGMENTED to handle integer restrictions on some or all of the variables. The present work use the basic framework of MINTO as a starting point. Published details of MINTO are scant, however some information can be found in recent publications by Murtagh [58, 59] and the forthcoming monograph by Murtagh and Saunders [63].

Chapter 5

Murtagh's direct search heuristic

This chapter will describe in detail a direct-search algorithm for NLIP proposed by Murtagh [59]. In the sections to follow we analyse his method and propose an extension. It will be claimed that this modified approach will allow more flexibility and reliability in rapidly arriving at an integer-feasible solution which may then be used to start the branch-and-bound process, or in some cases to circumvent branch-and-bound entirely.

It should be noted also that after some extensive discussion with Murtagh and further research by the present author, a number of relatively minor points have been cleared up and some parts of the original algorithm refined slightly. In the interests of both maximum clarity and suitability for subsequent coding, the present notation is also slightly different to that of Murtagh [59], however no confusion should result. These minor changes are to be seen as quite distinct from the new direct search procedures given in Chapter 6.

5.1 Structure of the problem

The general form of the NLIP problem to be solved by the methods introduced in this thesis is given by the set of requirements labelled (117) below, and following Murtagh, we assume that a bounded feasible solution exists to the problem. The present formulation is that of Murtagh [57], p105, which is only slightly different to that given in Murtagh [59].

$$\underset{\mathbf{x} \in R^n}{\text{minimize}} \quad f^0(\mathbf{x}^N) + \mathbf{c}^T\mathbf{x}^L \tag{117}$$

$$\begin{aligned}
\text{subject to} \quad & \mathbf{f}(\mathbf{x}^N) + \mathbf{A}_1\mathbf{x}^L = \mathbf{b}_1 && (m_1 \text{ rows}) \\
& \mathbf{A}_2\mathbf{x}^N + \mathbf{A}_3\mathbf{x}^L = \mathbf{b}_2 && (m_2 \text{ rows}) \\
& \mathbf{l} \le \mathbf{x} \le \mathbf{u} && (m = m_1 + m_2) \\
& x_j \quad \text{integer}, \quad j \in J_I
\end{aligned}$$

There are n variables and m constraints, $m < n$.

Some (assumed small) proportion of the variables **x** are assumed to be nonlinear in either the objective function and/or the constraints, and some (also assumed small) proportion of the variables are required to be integer-valued. We refer to a variable as **nonlinear** if it appears nonlinearly in the problem formulation in either the objective function or the constraints.

The same structure without the integer requirements forms the basis of the MINOS large-scale nonlinear programming code (Murtagh and Saunders (1982, 1987)) [61, 64]. This involves a sequence of **major iterations**, in which the first-order Taylor series approximation terms replace the nonlinear constraint functions to form a set of linear constraints, and the higher order terms are adjoined to the objective function with Lagrange multiplier estimates.

The set of linear constraints (excluding bounds) is then written in the form:

$$A\mathbf{x} \quad = \quad [B \quad S \quad N]\begin{bmatrix} \mathbf{x}_B \\ \mathbf{x}_S \\ \mathbf{x}_N \end{bmatrix} \quad = \quad \mathbf{b} \tag{118}$$

B is $m \times m$ and non-singular, $\mathbf{x}_N$ are "non-basic" variables which are held at one or other of their bounds. $\mathbf{x}_B$ and $\mathbf{x}_S$ are referred to as *basic* and *superbasic* variables respectively, and in order to maintain feasibility during the next step they must satisfy the equation

$$B\Delta\mathbf{x}_B + S\Delta\mathbf{x}_S \quad = \quad \mathbf{0} \tag{119}$$

or, since the basis is non-singular, we may write

$$\Delta\mathbf{x}_B \quad = \quad -B^{-1}S\Delta\mathbf{x}_S \tag{120}$$

Apart from the choice involved in deciding which nonbasic to slide up or down toward its other bound in order to improve the objective function, we have freedom to alter the superbasics. A step to the interior of the feasible set is possible since the superbasics need not be at a bound and are normally between bounds.

Because of equation (120), the superbasics are seen as the driving force, since the step $\Delta\mathbf{x}_S$ determines the whole step $\Delta\mathbf{x}$. The key to the success of the algorithm in MINOS (Murtagh and Saunders [62]) is the assumption that the dimension of $\mathbf{x}_S$ remains small. According to computational experience reported by Murtagh [59], this can be assured if the proportion of nonlinear variables is small, but also in many instances in practice even when all the variables are nonlinear.

Similar assumptions will be made about the structure of nonlinear integer programs. It will be assumed that the proportion of integer variables in the problem is small.

Murtagh's approach to obtaining a (suboptimal) integer-feasible solution is via a direct search procedure using his concept of superbasic variables. Applications discussed in his CTAC '89 invited paper (Murtagh [59]) include optimal power flow (1200 constraints, 1500 variables—all nonlinear), manufacturing and process engineering. His work is an extension of ideas initially presented by Mawengkang and Murtagh (1986) [49], where the application considered was a quadratic assignment problem.

The first four sets of figure 16 partition the full index set $\{1,2,..,n\}$, ie $J_B \cup J_S \cup J_L \cup J_U = \{1,2,..,n\}$ and $J_\alpha \cap J_\beta = \varnothing$, $\alpha \neq \beta$. The set J_I of indices corresponding to integer variables is assumed to be of small cardinality, and $m + n_S + n_L + n_U = n$.

The approach assumes that the continuous problem is solved, and seeks an integer-feasible solution in the close neighbourhood of the continuous solution. The general philosophy is to leave non-basic integer variables at their respective bounds (and therefore integer valued) and conduct a search in the restricted space of basics, superbasics, and nonbasic continuous variables, $j \notin J_I$.

Murtagh's method may be broadly summarized as follows:

1. Obtain solution of the continuous relaxation (using the MINOS/MINTO code)

2. CYCLE1: remove integer variables from the basis by moving a suitable nonbasic away from its bound. The hope is to drive an infeasible integer basic variable to an integer value, and then to pivot it into the superbasic set; the previous nonbasic replacing it in the basis.

Some notation is first needed. We define the required index sets in figure 16.

Name	**Meaning**	**Cardinality**
J_B	set of indices for basic variables	$\lvert J_B \rvert = m$
J_S	set of indices for superbasic variables	$\lvert J_S \rvert = n_S$
J_L	set of indices for nonbasic variables at their lower bounds	$\lvert J_L \rvert = n_L$
J_U	set of indices for nonbasic variables at their upper bounds	$\lvert J_U \rvert = n_U$
J_I	set of indices for integer variables	$\lvert J_I \rvert = n_I$

Figure 16 Index sets for extended simplex partition

3. CYCLE2, pass1: adjust integer-infeasible superbasics by fractional steps to reach complete integer-feasibility.

4. CYCLE2, pass2: adjust integer feasible superbasics. This phase aims to conduct a highly-localized *neighbourhood search*—see Scarf [83]—to verify local optimality.

It should be noted that the designations CYCLE1, CYCLE2 etc do not appear in the CTAC ’89 paper (Murtagh [59]), however they were presented in the lecture (Murtagh [60]), so we shall use the terminology here.

We consider the detailed steps of CYCLE1, CYCLE2 Pass1, CYCLE2 Pass2 and then investigate the performance of Murtagh’s algorithm via some simple examples in the sections immediately following.

The method is imbedded in a branch-and-bound procedure in which branching to further subproblems will terminate if one of the following three criteria is satisfied:

1. The subproblem has no feasible solution.

2. The solution of the subproblem is no better than the current best known integer feasible solution.

3. The solution is integer feasible (to within a pre-defined level of tolerance).

Since the procedure of Murtagh determines a locally-optimal solution in the neighbourhood of the original continuous solution, there may be some merit in seeking the assurance of a branch-and-bound procedure for fathoming all possible integer solutions. There would be little cost in this, as the solution obtained by the above procedure should provide a tight bound which will serve to curtail the branching process very rapidly under criterion 2 above.

In the following chapters, we analyze the algorithm of Murtagh [60] presented here and then compare it with a modified version in which alternative direct search methods are used.

5.2 CYCLE1—remove integer variables from the basis

It is necessary to impose some preconditions or assumptions on the problem data before CYCLE1 can be expected to succeed.

We suppose that at the continuous solution an integer variable is basic at a non-integer value

$$x_{i'} \quad = \quad x_{i'} \quad + \quad f_{i'}, \qquad 0 < f_{i'} < 1 \tag{121}$$

Further, we suppose that a chosen non-basic non-integer variable x_{j^*} is being released from its lower bound.

The assumption that the proportion of integer variables is small becomes a key issue in ensuring that the interchange operations can take place; fortunately many practical problems have this characteristic. Note also that it is assumed there is a full set of slack variables present.

The work of Mawengkang and Murtagh [49] suggests a preferred choice for i' given by:

$$\min(f_{i'}, 1 - f_{i'}) \quad \leq \quad \min(f_i, 1 - f_i) \qquad i \in J_I \tag{122}$$

This choice of i' is motivated by the desire for minimal change in the objective function, and clearly corresponds to the integer basic with smallest integer-infeasibility. We observe however that this approach only makes sense if the components of the reduced gradient vector are comparable in magnitude.

Also, in choosing the non-basic (continuous) j^* for the steps of CYCLE1, Murtagh suggests the preferred criterion to be the value of j for which (in the present notation):

$$\min_{j \in (J_L \cup J_U) - J_I \mid \alpha_{i'j} \neq 0} \left| \frac{\lambda_j}{\alpha_{i'j}} \right| \tag{123}$$

occurs, where λ_j is the jth component of nonbasic partition of the reduced gradient vector or reduced costs vector λ_N, $\alpha_{i'j} = (B^{-1}\mathbf{a}_j)_{i'}$ and $\mathbf{a}_j$ is the column of A corresponding to the non-basic x_j.

In fact, since we have gone to the trouble of finding the integer basic with smallest integer-infeasibility, it won't make much sense to choose a nonbasic which forces our chosen basic *to go in the wrong direction* when the nonbasic j^* is moved. It is easy to create counterexamples which illustrate this problem. Thus we need to refine our heuristic for the choice of j^*. Such refinements are discussed at length in Chapter 6.

Comparison of the Murtagh & Saunders [62] MINOS paper with CTAC '89 paper [59] shows that instead of d_j (CTAC notation); in the notation of the MINOS paper, the numerator is λ_j, "analogous to reduced costs of LP"—see equation 15 of that paper.

In fact

$$\lambda = \mathbf{g}_N - N^T\pi \tag{124}$$

and

$$\pi = (B^T)^{-1}\mathbf{g}_B \tag{125}$$

—see Murtagh and Saunders [62], eqs 13, 15.

The reasoning behind this criterion is that it measures the deterioration of the objective function value per unit change in the basic variable $x_{i'}$.

The terms $\alpha_{i'j}$ are calculated by firstly producing a vector $\mathbf{z}^T = \mathbf{e}_{i'}^T B^{-1}$ (this is row i' of B^{-1}), and then calculating the inner product $\alpha_{i'j} = \mathbf{z}^T \mathbf{a}_j$. Once a particular j^* is chosen, the full vector $\alpha_{j^*} = B^{-1}\mathbf{a}_{j^*}$ is calculated for the ratio tests in equations (126)–(128).

As our chosen nonbasic x_{j^*} moves toward its other bound, four possible events may occur as follows.

Event 1. A basic variable x_{i_1}, $i_1 \neq i'$ hits its lower bound first.

Event 2. A basic variable x_{i_2}, $i_2 \neq i'$ hits its upper bound first.

Event 3. An integer basic variable x_{i_3}, $i_3 \in J_B \cap J_I$ becomes integer-feasible.

Event 4. The non-basic x_{j^*} hits its other bound first.

Notes

1. The possibility $i_1 = i'$ is excluded from event 1 and $i_2 = i'$ is excluded from event 2 above since the cases where $x_{i'}$ hits a bound are included in event 3, where the possibility $i_3 = i'$ arises. The desired outcome is clearly event 3.
2. An integer variable at a bound is necessarily integer-feasible.

Corresponding to each of the four possible events, we compute the following quantities:

$$\theta_1 = \min_{i \in J_B - \{i'\} \mid \alpha_{ij^*} > 0} \left(\frac{x_i - l_i}{\alpha_{ij^*}} \right) \tag{126}$$

$$\theta_2 = \min_{i \in J_B - \{i'\} \mid \alpha_{ij^*} < 0} \left(\frac{u_i - x_i}{-\alpha_{ij^*}} \right) \tag{127}$$

$$\theta_3 \quad = \quad \min \left(\min_{i \in J_I \cap J_B \mid \alpha_{ij^*} < 0} \frac{1 - f_i}{-\alpha_{ij^*}}, \quad \min_{i \in J_I \cap J_B \mid \alpha_{ij^*} > 0} \frac{f_i}{\alpha_{ij^*}} \right) \tag{128}$$

$$\theta_4 \quad = \quad u_{j^*} - l_{j^*} \tag{129}$$

where

$$\alpha_{ij} \quad = \quad (B^{-1}\mathbf{a}_j)_i \tag{130}$$

and $\mathbf{a}_j$ is the column of A corresponding to the non-basic x_j.

Therefore we have

$$\theta^* \quad = \quad \min(\theta_1, \theta_2, \theta_3, \theta_4) \tag{131}$$

If $\theta^* = \theta_1$ the basic variable x_{i_1} becomes non-basic at l_{i_1} and x_{j^*} replaces it in B. $x_{i'}$ stays basic with a new value (non-integer).

If $\theta^* = \theta_2$ then x_{i_2} becomes non-basic at u_{i_2} and x_{j^*} replaces it in B as above.

If $\theta^* = \theta_3$ then x_{i_3} is made superbasic at an integer value and x_{j^*} replaces it in B.

If $\theta^* = \theta_4$ then x_{j^*} remains non-basic, but now at its upper bound, and $x_{i'}$ stays basic with a new value (non-integer).

Similar ratios can be calculated for the case of x_{j^*} being released from its upper bound. In general, we can capture both possibilities (release from lower or upper bound) and avoid code duplication by defining the *direction indicator for CYCLE1*, σ_1, as follows. If x_{j^*} is released from its lower bound ($j \in J_L$) then $\sigma_1 = 1$. If released from upper bound ($j \in J_U$) then $\sigma_1 = -1$. Thus, $\sigma_1 \;=\; signum(\Delta x_{j^*})$, where Δx_{j^*} is the step in the nonbasic x_{j^*}, $j^* \in (J_L \cup J_U) - J_I$.

Both cases are summarized in the following table.

CYCLE 1—GENERALIZED

EVENT	LIMITING STEP	ACTION
1. A basic variable x_{i_1}, $i_1 \neq i'$ **hits its lower bound**	$\theta_1 = \min\limits_{i \in J_B \mid (i \neq i' \wedge \sigma_1 \alpha_{ij^*} > 0)} \left(\dfrac{x_i - l_i}{\sigma_1 \alpha_{ij^*}} \right)$	$x_{i_1} \rightarrow$ nonbasic at l_{i_1} (PIVOT) $x_{j^*} \rightarrow$ basic
2. A basic variable x_{i_2}, $i_2 \neq i'$ **hits its upper bound**	$\theta_2 = \min\limits_{i \in J_B \mid (i \neq i' \wedge \sigma_1 \alpha_{ij^*} < 0)} \left(\dfrac{u_i - x_i}{-\sigma_1 \alpha_{ij^*}} \right)$	$x_{i_2} \rightarrow$ nonbasic at u_{i_2} (PIVOT) $x_{j^*} \rightarrow$ basic
3. A basic variable x_{i_3}, $i_3 \in J_I \cap J_B$ **assumes an integer value**	$\theta_3 = \min \left(\min\limits_{i \in J_I \cap J_B \mid \sigma_1 \alpha_{ij^*} < 0} \dfrac{1 - f_i}{-\sigma_1 \alpha_{ij^*}}, \min\limits_{i \in J_I \cap J_B \mid \sigma_1 \alpha_{ij^*} > 0} \dfrac{f_i}{\sigma_1 \alpha_{ij^*}} \right)$	$x_{i_3} \rightarrow$ superbasic integer (PIVOT) $x_{j^*} \rightarrow$ basic **(see MINOS step 7a)**
4. The non-basic x_{j^*} **hits the other end of its range**	$\theta_4 = u_{j^*} - l_{j^*}$	$x_{j^*} \rightarrow$ nonbasic at other end of range

Table 1 Remove integer variables from the basis—a selected non-integer nonbasic x_{j^*}, $j^* \in (J_L \cup J_U) - J_I$ **is released from it bound.**

Notes on CYCLE 1

1. $\alpha_{ij} \quad = \quad (B^{-1}\mathbf{a}_j)_i$ and $\mathbf{a}_j$ is the column of A corresponding to the nonbasic $(\mathbf{x}_N)_j$

2. The maximum nonbasic step $\theta^* \quad = \quad \min\,(\theta_1, \theta_2, \theta_3, \theta_4)$

   Note that for CYCLE1, some θs may not exist, since the set of indices for which eg $\alpha_{ij^*} > 0$ may be empty. Note that θ_4 always exists and θ_3 always exists provided that the basis contains at least one integer variable. There is no doubt that the basis will always contain at least one integer variable throughout CYCLE1 since the principal termination condition for CYCLE1 is precisely that there be no integer variables left in the basis. A secondary guard is the customary iteration limit.

3. When coding the CYCLE1 algorithm, if two or more events occur simultaneously, we must always choose event 3 if possible, not just eg the first or last one to happen in some battery of *IF* statements. The ultimate aim of CYCLE1 is to force as many as possible of the infeasible integer variables to become non-basic or superbasic. Clearly then, event 3 is the most desirable outcome in each iteration of CYCLE1.

4. If we fail to achieve event 3 for our chosen i' and j^*, we may choose to explore the use of *multiple pricing* to try to choose alternative j^*s for which event 3 may happen. This is elaborated in one of the proposed new methods of Chapter 6.

5.3 CYCLE2 Pass 1—adjust integer-infeasible superbasics

Step 1.

Choose the superbasic with the smallest integer-infeasibility, i.e. we seek $j' \in J_S$ as the value of j for which

$$\zeta_0 \quad = \quad \min_{j \in J_S} \quad \min \quad (f_j, 1 - f_j) \tag{132}$$

occurs. J_S is the index set for the superbasics. A forward step will be taken if $1 - f_{j'} \le f_{j'}$ and a backward step is taken otherwise. We do this in the following way.

Define j_1 as the index of the superbasic variable with the smallest fractional component. Thus

$$f_{j_1} \quad = \quad \min_{j \in J_S} \quad f_j \tag{133}$$

Similarly, define j_2 as the index of the superbasic variable with the largest fractional component. Thus

$$f_{j_2} \quad = \quad \max_{j \in J_S} \quad f_j \tag{134}$$

Clearly the values of j_1 and j_2 may not be unique. At present we choose to resolve ties arbitrarily. Perhaps later the method could be refined—eg resolve ambiguity by selecting one from those corresponding to superbasic variables with least reduced cost.

The minimum integer-infeasibility is given by

$$\zeta_0 \quad = \quad \min\,(f_{j_1}, 1 - f_{j_2}) \tag{135}$$

and it will occur at $j = j_1$ if $f_{j_1} < 1 - f_{j_2}$ else at $j = j_2$ if $f_{j_1} > 1 - f_{j_2}$.

Also note that a full step to integer feasibility in $x_{j'}$ may not be possible since a simple bound on a basic may be hit. This is summarized in equations (136) and (137). The limiting backward step, $\Delta x_{j'} \quad < \quad 0$ is given by:

$$\Delta x_{j'} \quad = \quad -\min \quad \left(f_{j'}, \quad \min_{i \in J_B \,|\, \alpha_{ij'} < 0} \quad \left(\frac{(\mathbf{x}_B)_i - l_i}{-\alpha_{ij'}} \right), \quad \min_{i \in J_B \,|\, \alpha_{ij'} > 0} \quad \left(\frac{u_i - (\mathbf{x}_B)_i}{\alpha_{ij'}} \right) \right) \tag{136}$$

The limiting forward step, $\Delta x_{j'} \quad > \quad 0$ is given by

$$\Delta x_{j'} \quad = \quad \min \quad \left(1 - f_{j'}, \quad \min_{i \in J_B \,|\, \alpha_{ij'} > 0} \quad \left(\frac{(\mathbf{x}_B)_i - l_i}{\alpha_{ij'}} \right), \quad \min_{i \in J_B \,|\, \alpha_{ij'} < 0} \quad \left(\frac{u_i - (\mathbf{x}_B)_i}{-\alpha_{ij'}} \right) \right) \tag{137}$$

In the interests of avoiding code duplication, the computations implied by (136) and (137) may be described in a more compact form as follows.

We define the **direction indicator for CYCLE2**, σ_2:

$$\sigma_2 \quad = \quad \left\{ \begin{matrix} -1; & f_{j_1} & < & 1 - f_{j_2} \\ +1; & f_{j_1} & > & 1 - f_{j_2} \end{matrix} \right\} \tag{138}$$

σ_2 is just the sign of our desired step $\Delta x_{j'}$, and the limits to our step imposed by the simple bounds are given by

$$\zeta_l \quad = \quad \min_{i \in J_B \mid \sigma_2 \alpha_{ij'} > 0} \left(\frac{x_i - l_i}{\sigma_2 \alpha_{ij'}} \right) \tag{139}$$

$$\zeta_u \quad = \quad \min_{i \in J_B \mid \sigma_2 \alpha_{ij'} < 0} \left(\frac{u_i - x_i}{-\sigma_2 \alpha_{ij'}} \right) \tag{140}$$

Step 2.

Now see if we can make a fractional step (forward or backward) to make $x_{j'}$ integral. We must check to see that all basics remain feasible. In this part of the algorithm, the basis remains unchanged, ie there are no pivots, since we are making fractional adjustments to the superbasic integers which are presently integer-infeasible. The new values of the basics must of course follow because of their linear dependence on the superbasics and (fixed) nonbasics.

If a *full* step forward or backward ie $|\Delta x| \quad = \quad \zeta$ to the nearest integer is possible for superbasic j' then we take it, but if not then we step as far as possible without violating a simple bound on a basic.

Thus we define i_l as the value of i for which the minimum in (139) occurs; i_u as the value of i for which the minimum in (140) occurs; provided the respective index sets are not empty (in which case one or other of i_u, i_l may not exist). Our step 2 of CYCLE2 can now be recast as:

$$\Delta x_{j'} \quad = \quad \sigma_2 \min \quad (\zeta_0, \quad \zeta_l, \quad \zeta_u) \tag{141}$$

Is it possible for our step $\Delta x_{j'}$ to be zero? The answer to this question is no, since this would mean that a basic is already at a bound, ie that the current set of constraints is degenerate. This situation is ruled out by our assumption of non-degeneracy.

In determining σ_2 we must be very careful. For example, if integer-infeasibility = 0.5 then which way do we step? In example 3 of section 5.5.3 this is crucial since if we choose the wrong direction ($\sigma_2 = +1$) then our permissible step is 0.

5.4 CYCLE2 Pass 2—adjust integer feasible superbasics

The superbasics can be varied at will, subject to preserving the feasibility of the basic variables. Thus a search through the neighbourhood system, as defined by Scarf (1986) [83], will verify the (local) optimality of the integer-feasible solution obtained.

Step 1

This is basically a one-dimensional *steepest descent*.

Choose $j' \in J_I$. The criterion for selecting j' will be that of maximum reduced cost λ_j.

Step 2

Calculate $\alpha_{j'}$. Also determine direction of move—check sign of λ_j', and adjust the unit tests in step 3 in light of this.

Step 3

Check that a unit move is possible:

$$\frac{u_i - x_i}{+\alpha_{ij'}} \geq 1 \qquad \forall i \mid i \in J_B \mid \alpha_{ij'} > 0 \tag{142}$$

$$\frac{x_i - l_i}{-\alpha_{ij'}} \geq 1 \qquad \forall i \mid i \in J_B \mid \alpha_{ij'} < 0 \tag{143}$$

Step 4

Move by 1 unit; check that objective improves, ie search in neighbourhood system as defined by Scarf [83, 84].

5.5 Analysis and counterexamples for Murtagh's algorithm

Status information

The following information is required for CYCLE1 to commence:

1. The problem data, ie definition of objective function, the coefficients for the linear constraints, the lower and upper bounds, the set of integer variables.

2. Extended definition of the partition which differentiates between nonbasics at lower bound and nonbasics at upper bound; this can be defined by Murtagh's index vector *hb*, and status vector *hs* as follows:

Define hb_j to be the natural index of the *j*th variable of the partition, where $1 < j < m + n_S$.

Consider the *j*th *natural* variable. Then we define the **status indicator** hs_j as follows

$$hs_j = \left\{ \begin{array}{ll} 0; & \text{if nonbasic at lower bound} \\ 1; & \text{if nonbasic at upper bound} \\ 2; & \text{if superbasic} \\ 3; & \text{if basic} \end{array} \right\} \tag{144}$$

The index range for the basic variables is $1 \le j \le m$, and for the superbasic variables is $m + 1 \le j \le m + n_S$.

The partition may be illustrated as follows

B	S	NL	NU

$1 \le j \le m \qquad m + 1 \le j \le m + n_S$

Note: the nonbasics are not indexed directly since, being at one or other of their bounds, their values are implicitly known.

3. Current values of the superbasic variables

4. Some tolerances

CYCLE1 then has enough information to proceed.

We now consider the class of linearly constrained mixed-integer quadratic programs defined by the requirements (145)–(149).

Counterexample general form

minimize

$$f = \sum_{i=1}^{3} \gamma_i (x_i - \tau_i)^2 \tag{145}$$

subject to

$$1x_1 + 0x_2 + \omega_1 x_3 \leq b_1 \tag{146}$$

$$0x_1 + 1x_2 + \omega_2 x_3 \leq b_2 \tag{147}$$

$$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \tag{148}$$

$$x_2 \quad \text{integer} \tag{149}$$

The parameters $\gamma_i, \tau_i, \omega_1, \omega_2, b_1, b_2$ and the simple bounds **l, u** will be specified to illustrate some potential problems with CYCLE1 of the algorithm proposed by Murtagh [60].

Introducing slacks x_4, x_5 into the general form, and setting all $\gamma_i = 1.0$ and $\tau_1 = 1.2, \quad \tau_2 = 2.5, \quad \tau_3 = 0.0$, we have:

Counterexample general form with slacks

minimize

$$f = \sum_{i=1}^{3} \gamma_i (x_i - \tau_i)^2 \tag{150}$$

subject to

$$1x_1 + 0x_2 + \omega_1 x_3 + 1x_4 + 0x_5 \leq b_1 \tag{151}$$

$$0x_1 + 1x_2 + \omega_2 x_3 + 0x_4 + 1x_5 \leq b_2 \tag{152}$$

$$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \tag{153}$$

$$x_2 \quad \text{integer} \tag{154}$$

Relaxed and integer-feasible optima

The continuous unconstrained optimum for our chosen objective function is clearly $\mathbf{x} = \tau = (1.2, 2.5, 0, 0, 0)^T$.

It is also clear that the point $\mathbf{x} = (1.2, 2, 0, 0, 0.5)^T$ satisfies the general (linear) constraints, simple bounds, and is integer feasible. For the simple quadratic objective, it is easy to see that it is therefore a local optimum for the originally posed problem.

General tableau equations

If $\mathbf{x}_B$ is the vector of basic variables and $\alpha_j = (B^{-1}N)_j$, then the current tableau may be expressed as

$$\mathbf{x}_B = \beta - \sum_{j \in J_L} \alpha_j l_j - \sum_{j \in J_U} \alpha_j u_j - \sum_{j \in J_S} \alpha_j x_j \tag{155}$$

If $i \in J_B$ in the present notation then we may write

$$x_i = \beta_i - \sum_{j \in J_L} \alpha_{ij} l_j - \sum_{j \in J_U} \alpha_{ij} u_j - \sum_{j \in J_S} \alpha_{ij} x_j \tag{156}$$

Counterexamples—general

To examine the method proposed by Murtagh [59], we now consider a number of simple examples based on the preceding class of problems. For the first two examples, we suppose that the continuous optimum for this problem has x_1, x_2 basic, x_3 nonbasic, and x_4, x_5 superbasic. At this point the tableau equations will read

$$x_1 = b_1 - \omega_1 x_3 - x_4 \tag{157}$$

$$x_2 = b_2 - \omega_2 x_3 - x_5 \tag{158}$$

5.5.1 Example 1

Now it may happen that the interval $[l_3, u_3]$ is so narrow that x_3 hits its upper bound before events 1, 2 or 3 of CYCLE1 can occur. This can clearly be contrived by making $u_3 - l_3$ small enough. Since x_3 is the only nonbasic we see that CYCLE1 will not terminate (x_3 will oscillate between its bounds). To illustrate this we consider example 1, whose definition follows.

Definition of example 1

---

minimize

$$f = (x_1 - 1.2)^2 + (x_2 - 2.5)^2 + x_3^2 \tag{159}$$

subject to

$$1.0x_1 + 0.0x_2 - 1.0x_3 + 1.0x_4 + 0.0x_5 = 1.2 \tag{160}$$

$$0.0x_1 + 1.0x_2 + 0.1x_3 + 0.0x_4 + 1.0x_5 = 2.5 \tag{161}$$

$$(0,0,0,0,0)^T \leq \mathbf{x} \leq (5,5,1,100,100)^T \tag{162}$$

$$x_2 \text{ integer, ie } J_I = \{2\} \tag{163}$$

Continuous solution vector and partition:

$$\mathbf{x}^* = (1.2, 2.5, 0.0, 0.0, 0.0)^T \tag{164}$$

$$J_B = \{1,2\} \tag{165}$$

$$J_S = \{4,5\} \tag{166}$$

$$J_L = \{3\} \tag{167}$$

$$J_U = \varnothing \tag{168}$$

Results for example 1

We have the tableau equations:

$$x_1 = 1.2 + 1.0x_3 - 1.0x_4 \tag{169}$$

$$x_2 = 2.5 - 0.1x_3 - 1.0x_5 \tag{170}$$

Now increase x_3 from 0.0 toward its upper bound 1.0. Both basics remain feasible all the way and further, the only integer variable, x_2, does not become integer feasible.

Event 4 occurs ie $x_3 \rightarrow u_3$. Since it is the only nonbasic, the next iteration of CYCLE1 will see x_3 revert to its lower bound once again. It is clear that this sequence of events will be repeated *ad infinitum.*

5.5.2 Example 2

Another possible difficulty is that an initially basic variable may cycle in and out of the basis. To see this, suppose that x_1 hits its upper bound as x_3 is increased from l_3. Thus we must choose $\omega_1 < 0$. Select ω_2 small enough so that x_2 neither becomes integer-feasible nor hits its bound.

Assigning values to parameters to illustrate this phenomenon, we have example 2, which is identical to example 1, except that the upper bound on x_3 has been widened:

Definition of example 2

---

minimize

$$f = (x_1 - 1.2)^2 + (x_2 - 2.5)^2 + x_3^2 \tag{171}$$

subject to

$$1.0x_1 + 0.0x_2 - 1.0x_3 + 1.0x_4 + 0.0x_5 = 1.2 \tag{172}$$

$$0.0x_1 + 1.0x_2 + 0.1x_3 + 0.0x_4 + 1.0x_5 = 2.5 \tag{173}$$

$$(0,0,0,0,0)^T \leq \mathbf{x} \leq (5,5,5,100,100)^T \tag{174}$$

$$x_2 \text{ integer, ie } J_I = \{2\} \tag{175}$$

Continuous solution vector and partition:

$$\mathbf{x}^* = (1.2, 2.5, 0.0, 0.0, 0.0)^T \tag{176}$$

$$J_B = \{1,2\} \tag{177}$$

$$J_S = \{4,5\} \tag{178}$$

$$J_L = \{3\} \tag{179}$$

$$J_U = \varnothing \tag{180}$$

Results for example 2

We have the tableau equations:

$$x_1 = 1.2 + 1.0x_3 - 1.0x_4 \tag{181}$$

$$x_2 = 2.5 - 0.1x_3 - 1.0x_5 \tag{182}$$

x_3 is nonbasic at $l_3 = 0$. Now we allow x_3 to increase. It can be seen that x_2 would become integer-feasible at $x_2 = 2$ if x_3 reached 5, however x_1 hits its upper bound $u_1 = 5$ before this at $x_3 = 3.8$.

Thus

$x_1 \quad \rightarrow \quad$ nonbasic at $u_1 = 5$

$x_3 \quad \rightarrow \quad$ basic (at 3.8)

A pivot operation gives the new tableau equations:

$$x_3 = -1.2 + x_1 + x_4 \tag{183}$$

$$x_2 = 2.62 - 0.1x_1 - 0.1x_4 - x_5 \tag{184}$$

Since $x_1 = 5$, we have $x_2 = 2.12$ and $x_3 = 3.8$.

Now release x_1 from its upper bound 5, since it is the only (non-integer) nonbasic. It should be clear that we get *cycling* using this process since as x_1 decreases from 5, x_3 will hit $l_3 = 0$ when $x_1 = 1.2$ but x_2 will not become integer-feasible. We have no other choice since x_1 is the only nonbasic.

5.5.3 Example 3

Example 3 has the same structure as examples 1 and 2 respectively except that a different starting partition is used.

Definition of example 3

---

minimize

$$f = (x_1 - 1.2)^2 + (x_2 - 2.5)^2 + x_3^2 \tag{185}$$

subject to

$$1.0x_1 + 0.0x_2 - 1.0x_3 + 1.0x_4 + 0.0x_5 = 1.2 \tag{186}$$

$$0.0x_1 + 1.0x_2 + 0.1x_3 + 0.0x_4 + 1.0x_5 = 2.5 \tag{187}$$

$$(0,0,0,0,0)^T \leq \mathbf{x} \leq (5,5,1,100,100)^T \tag{188}$$

$$x_2 \text{ integer, ie } J_I = \{2\} \tag{189}$$

Continuous solution vector and partition:

$$\mathbf{x}^* = (1.2, 2.5, 0.0, 0.0, 0.0)^T \tag{190}$$

$$J_B = \{1,2\} \tag{191}$$

$$J_S = \{3\} \tag{192}$$

$$J_L = \{4,5\} \tag{193}$$

$$J_U = \varnothing \tag{194}$$

Results for example 3

Once again we have the tableau equations:

$$x_1 = 1.2 + 1.0x_3 - 1.0x_4 \tag{195}$$

$$x_2 = 2.5 - 0.1x_3 - 1.0x_5 \tag{196}$$

in which x_4 and x_5 are nonbasic at 0. We have $i' = 2$ since x_2 is the only basic integer variable and therefore $i = 2$ is the only candidate for i'. Similarly, applying the heuristic for determining j^*, we find that $j = 5$ is the only candidate for j^* ($j = 4$ is not eligible since $\alpha_{24} = 0$).

Due to x_5 being in NL, it was easy for Murtagh's CYCLE1 to remove the integer variable x_2 from the basis. Contrasting this with example 1, we see that termination of Murtagh's CYCLE1 depends crucially on the initial partition. Modifications to his method will be described in Chapter 6.

5.5.4 Summary of example results for CYCLE1

These are organized into table 2 overleaf.

EXAMPLE	PROBLEM	CAUSES	POSSIBLE RESOLUTIONS
1.	The only available nonbasic slides back and forth between bounds without making any progress toward removing integer x_2 from the basis.	1. Not enough nonbasics available. 2. Not being allowed to use superbasics. 3. Small α_{23}.	1. Consider superbasics to become nonbasic. 2. Pivot basic with suitable continuous superbasic.
2.	x_1 and x_3 alternate in and out of basis while our target integer variable x_2 does not go integer feasible.	1. Comparatively narrow bounds on x1, x3 coupled with only slight dependence ($\alpha_{21} = \alpha_{23} = 0.1$) on either x_1 or x_3 means not enough variation to make x_2 integral.	As above.
3.	No problem — cycle1 terminates in 1 iteration.	1. Fortuitous initial partition ie x_5 in NL allowed x_2 to achieve integer feasibility. Thus we need access to x_5 which was superbasic in examples 1 and 2 and therefore not accessible.	Not required.

Table 2 Resolution of problems with Murtagh's direct search

5.5.5 Conclusions

1. What implications do the counterexamples have for CYCLE1 of Murtagh's direct search algorithm? It is clear from the foregoing examples that at least 2 types of "cycling" are possible in CYCLE1 (unfortunate choice of words!).

   Firstly, we have seen that the extreme case of only one eligible continuous nonbasic x_j with very narrow bounds may result in the cycle $x_j \to u_j$, followed by $l_j \leftarrow x_j$ indefinitely. There is no change of basis in this type of cycling (example 1). Secondly, cyclic basis changes occur as a fixed sequence of nonbasics enter and leave the basis with period 2 or more. This phenomenon, which occurs in example 2, may also occur even in purely *linear* (MILP) problems, and is discussed further in the following chapter.

2. We need to ensure that CYCLE1 will always terminate. It has not yet been established that this is always possible. Certainly in the case where there are no integer variables in the basis to begin with, CYCLE1 must be skipped. In fact, we see from (1) above that CYCLE1 does *not* terminate in general.

3. For the present CYCLE1 of Murtagh, it is clear that at each iteration, one of events 1-4 must occur. However, there would seem to be no guarantee that event 3 will *ever* occur! This is obvious from examples 1 and 2 above. We need to consider ways in which CYCLE1 could be modified to ensure termination, ie no integer variables left in the basis. Until this occurs, no progress in the class of direct search methods based on Murtagh's concept of superbasic variables can be made. These matters are discussed at length in Chapter 6,where we consider modifications to Murtagh's CYCLE1.

4. It should be noted that this set of very simple examples was devised long before the interactive implementation of the methods of Chapter 6. It was therefore of some considerable interest to check the behaviour of the methods on the present small QIPs. This is discussed in Chapter 8.

5. Murtagh's original approach has been successful in solving a number of NLIPs. He assumed a small proportion of integer variables, and a small proportion of nonlinearities. We next seek alternative methods which extend his ideas, so that progress can be made on a somewhat wider class of problems.

Chapter 6

Proposals for new direct search methods

Problems with Murtagh's CYCLE1

In the previous chapter we considered Murtagh's heuristic for direct search. For CYCLE1 of Murtagh's algorithm (an attempt to remove all integer variables from the basis), it is supposed that a chosen non-basic non-integer variable x_{j^*} was being released from its lower bound. From the problems brought to light by the class of constrained integer quadratic examples of the previous section, it is clear that we need to modify Murtagh's method as presented in Chapter 5 and the paper [60].

Two cycling possibilities emerged: a nonbasic could oscillate between its bounds without change of basis and without event 3 (a basic integer variable becoming integer-feasible) occurring, and secondly, a sequence of variables cycling between the basic and nonbasic partitions without making progress toward emptying the basis of integer variables. In either case, Murtagh's CYCLE1 iterates indefinitely since the termination condition $J_B \cap J_I = \emptyset$ is never met. To have any hope of forcing a modified CYCLE1 to terminate, it is clear that we must have access to the ***superbasic*** variables. This is the case even for *linear* (MILP) problems, in which there are zero superbasics at the solution of the continuous relaxation. However, new degrees of freedom can be opened up for linear problems by the simple device of changing the status of a nonbasic variable to superbasic, but with no movement away from its bound.

6.1 Alternative approaches for CYCLE1 (remove integers from basis)

If all nonbasics are integer variables and CYCLE1 still hasn't terminated (eg basis is all-slack except for one integer variable and one slack is superbasic), then we may select a superbasic non-integer variable for j^* rather than a nonbasic. Since we want to remove *all* integer variables from the basis, it is reasonable to require that we have access to *all other* variables (ie superbasic as well as nonbasic) in order to do this. In essence, we are saying that we really should allow superbasics to vary without pivot just as nonbasics do. This extra degree of freedom may just be enough to get integer feasibility in a basic integer variable—it certainly is in the quadratic counterexamples considered earlier.

If non-termination of any proposed new CYCLE1 is detected, eg with iteration counter (difficulties getting our chosen basic integer variable to be coaxed to integer-feasibility), then we could just pivot an integer variable with a superbasic anyway even if it isn't integer feasible. Such a superbasic must correspond to a column of the current linearized general constraint matrix that will not cause the new basis to be singular (or near-singular).

Another approach for a modified CYCLE1 would be to start as in Murtagh's approach, ie try to get integers out of the basis by sliding selected continuous nonbasics to their other bounds. Do this for as long as possible—a condition for detection of termination of this process is required, and herein lies the difficulty. It was decided for the present research not to proceed with this approach, primarily because clean detection of non-termination in this case can be awkward and a simple iteration limit is crude and really avoids the structure of the problem.

If integers are still basic and no suitable nonbasic can be found to precipitate event 3, we may try a MINOS step 7a (see Murtagh and Saunders [62], p50). This involves a redefinition of the partition so that a continuous superbasic is interchanged with an integer basic, and it must be ensured that the basis remains non-singular. The Sherman-Morrison identity (see, for example Murtagh's *Advanced Linear Programming* [57]) can be used as a basis for checking a proposed new basis column, or simply that row i' of the current basis inverse must not be (nearly) orthogonal to the proposed new column. Now choose a nonbasic j^* to become superbasic (expand search space) and repeat the steps above. The nonbasic to be chosen must correspond to a continuous variable, and a suitable heuristic would be to go for one with a large $\alpha_{i'j^*}$ and wide bounds in the hope that subsequent movement of this variable when superbasics are altered in CYCLE2 will produce integer feasibility in $x_{i'}$ (in

fact, for the methods actually implemented, we just choose the first suitable nonbasic in natural order). Note that since x_{j^*} will be immediately selected for $B \leftrightarrow S$ pivot, the heuristic for j^* should be extended to require the nonsingular basis invariant.

Another idea is to pivot as many basic integers with suitable superbasic continuous variables as possible, since there will be no movement from continuous solution, just a redefinition of the partition. If, as mentioned in one paper by Mawengkang and Murtagh [49], there were only 2 or 3 superbasics at a continuous solution even though the problem has more than 1000 integer variables, then we may need to promote selected continuous nonbasics to be superbasic status in order to give us enough superbasics to pivot the integers out of the basis. Then the modified CYCLE1 can proceed as defined just above. This should be done in any case where there are *integer-feasible* basics present at the continuous solution, and is also the basis of method M2, to be discussed in section 6.3.2.

6.2 Alternative approaches for CYCLE2 (superbasic steps)

For Murtagh's CYCLE2, it may be worthwhile to put more emphasis on getting a *full step* to integer feasibility than just blindly looking for the superbasic with minimum integer-infeasibility and then trying a full step on that one, but being content with a partial step if we hit a bound on a basic. For example, superbasic #1 may have integer-infeasibility = 0.3 and superbasic #2 may have integer-infeasibility = 0.4; however a full step to integer feasibility may be possible with #2 but not with #1. It seems reasonable to prefer a full step with #2 instead of a partial step with #1—provided the objective does not worsen too much.

Once j' is chosen and the corresponding $\alpha_{j'}$ is calculated we can do as much work as we like with it—the real computational burden is in choosing j' and calculating $\alpha_{j'}$, so all possible steps may be calculated once j' is chosen. Therefore, it would not be unreasonable to choose a few (perhaps 3–5) "likely prospects" for j' (integer-infeasible superbasic to be stepped) and evaluate $\alpha_{j'}$ for all of them. This is somewhat akin to "multiple pricing" in ordinary LP, and is also in keeping with the idea of the number of integer variables being small. In this manner, if no step is possible with current j', then another could be tried. The linearized general constraints are automatically satisfied for any step away from the current feasible point, however the simple bounds on the basics will limit our step. Since all integer variables were removed from the basis in CYCLE 1, the only barrier to complete feasibility of the current solution is presented by the integer-infeasible superbasics (nonbasic integer variables are at a bound and are therefore integer-valued).

6.3 The new methods

After CYCLE1 terminates, we may either apply some form of Murtagh's CYCLE2 as defined in the CTAC 89 invited lecture [60], or we could fix the integer-feasible integer variables and solve the resulting continuous problem.

Based on the somewhat disjointed batch of ideas presented in the previous two sections, we wish to propose several new direct search methods for NLIP. In the presentation, some simplifications have been made in order to give pseudocode which is reasonably readable. For example, code which controls the abortion of loops in the case of detected errors has been omitted for reasons of simplicity. Calls to subroutines in the actual code have been replaced with short, descriptive statements of the basic functions of the routines concerned. It is hoped in this way to convey a fairly clear statement of the processes involved without undue and distracting detail.

Some abbreviations/identifiers used in the pseudocode for the methods are as follows:

```
numbasinf     number of basis integer-infeasibilities
maxits        maximum iterations (constant)
jstar         nonbasic selected to move away from bound
thetastar     largest permissible step for nonbasic jstar
ierrcode      error code
its           iteration counter
searching     Boolean flag to control search for nonbasic jstar
stilllooking  as above
jmin          initial value for of j for jstar search
n             number of columns (variables) in problem
event3count   number of event 3 occurrences, ie basic goes IF
jstarstar     superbasic for B <--> S pivot
jns           nonbasic to be promoted to superbasic
```

One further comment is in order: throughout the remaining chapters, reference is made to *Method 0*. This is simply branch-and-bound as implemented in Murtagh's MINTO code [58]. The method is well-described by several authors, and also briefly summarized in the present section 2.9.1.

6.3.1 Method 1

Method 1 is essentially CYCLE1 of the procedure described by Murtagh in [59], however the first significant operation on each traversal of the loop is the detection and pivoting of integer-feasible basics to the superbasic set. This operation involves the selection of a suitable superbasic variable to be pivoted into the basis. The corresponding column of the linearized constraint set must be such that the new basis is non-singular. Pseudocode for M1 is given in figure 17.

```
its := 0
do while (numbasinf>0) & (ierrcode=NOERROR) & (its<maxits)
     inc(its)
     pivot any integer-feasible basics with suitable superbasics
     compute jstar[1]
     compute minimum ratio thetastar
     do nonbasic step
     recompute number of basic integer-infeasibilities
enddo
```

Figure 17 Pseudocode for M1

6.3.2 Method 2

The idea for this method was to take immediate advantage of any superbasics available at the continuous solution to pivot out as many basic integers as possible, after each such pivot attempting a step to integer-feasibility before the next pivot. Initially, those basics *already feasible* would be pivoted with suitable superbasics, as in M1 above. The motivation behind M2 is that we have more control over an integer variable if it is superbasic than if it is basic. Also, a pivot operation does not alter the solution vector, merely the extended partition definition, so that the objective does not change either. From the standpoint of algorithmic taxonomy, it can be considered a *greedy* strategy (see, for example, McMahon [50]), since we try to postpone any deterioration in objective for as long as possible. Pseudocode for M2 is given in figure 18.

1 Nonbasic to move away from bound.

```
its := 0
pivot any integer-feasible basics with suitable superbasics
do while (numbasinf>0) & (ierrcode=NOERROR) & (its<maxits)
     inc(its)
     compute jstarstar[1]
     do B<-->S pivot[2]
     recompute basics, objective and gradients
     recompute number of basic integer-infeasibilities
enddo
do while ((numbasinf>0) & (ierrcode=NOERROR) & (its<maxits))
     inc(its)
     compute jstar[3]
     compute minimum ratio thetastar
     do nonbasic step
     recompute number of basic integer-infeasibilities
enddo
try to step each infeasible integer superbasic to nearest integer
```

Figure 18 Pseudocode for M2

6.3.3 Method 3

The aim in this method is to insist on event 3 (our chosen basic integer variable goes integer-feasible) for the nonbasic step if possible. This involves a multiple-pricing operation since we do not give up in our search for a suitable nonbasic to precipitate event 3 until no suitable ones are left. Clearly, artificials are not considered, since they cannot move. Note also that M3 invokes M4 at the very end in case further progress can be made. Pseudocode for M3 is given in figure 19.

---

1 Superbasic for $B \leftrightarrow S$ pivot.

2 Pivot basic i' with superbasic j^{**}.

3 Nonbasic to move away from bound.

```
pivot any integer-feasible basics with suitable superbasics
jmin := 0
searching := true
event3count := 0
its := 0
do while searching
     jmin := jmin + 1
     j := jmin
     jstar := 0
     stilllooking := (j <= n)
     do while stilllooking
        if column j is nonbasic & not artificial then
           check if column j corresponds to integer variable
           if column j does not correspond to integer variable then
              jstar := j
              jmin  := j + 1
           endif
        endif
        inc(j)
        stilllooking := (jstar = 0) & (j < n)
     enddo
     compute minimum ratio thetastar
     if event = 3 then
        do nonbasic step
        recompute number of basic integer-infeasibilities
        pivot any integer-feasible basics with suitable superbasics
        increment event3count
     endif
     searching := (numbasinf > 0) & (its <= maxits)
     if jmin >= n then
        jmin := 0
        if event3count = 0 then
           searching := false
        else
           event3count := 0
        endif
     endif
enddo
invoke M4
```

Figure 19 Pseudocode for M3

6.3.4 Method 4

This strategy was developed in an attempt to combine the best of the earlier methods which were tried and found wanting. It has been found to be generally the best, although, and this is typical for methods which have at least some heuristic component, it is certainly not the best on all problems. Support for this statement can be seen by inspection of the computational experience chapters of this work.

Method 4 is sometimes successful in removing all integer variables from the basis. It is identical to M5 except that access to fixed or artificial variables to be considered for j_{NS}[1] is denied. In general, it is advisable to avoid artificials if possible since their presence in the basis can mean that no movement of superbasic or nonbasics is possible. This definitely "cramps the style" of later steps involving movement of superbasic variables, since we have effectively "backed ourselves into a corner" and cannot move. Having said that, we must also note that there exist problems for which the termination of M4 *requires* use of artificials for $B \leftrightarrow S$ pivot. This is the precise reason for the existence of M5, described in the next section. Pseudocode for M4 is given in figure 20.

1 j_{NS} is the index of a nonbasic column suitable for change of status to superbasic, and subsequent pivot with the integer basic i'.

```
its := 0
pivot any integer-feasible basics with suitable superbasics
do while (numbasinf>0) & (ierrcode=NOERROR) & (its<maxits)
     increment iteration count
     try to step each infeasible superbasic feasibility[1]
     recompute number of basic integer-infeasibilities
     compute jstar[2]
     compute minimum ratio thetastar
     if event = 3 then
        do nonbasic step
     else
        seek jstarstar[3]
        if jstarstar can't be found then
           seek nonbasic, jns[4], suitable to go superbasic then basic
        endif
        if suitable nonbasic jns found then
           change status of column jns to superbasic
           label new superbasic column jstarstar
           do B<-->S pivot[5]
        else
           ierrcode := CANTFINDAJSTARSTAR
        endif
     endif
     recompute number of basic integer-infeasibilities
     pivot any integer-feasible basics with suitable superbasics
enddo
```

Figure 20 Pseudocode for M4

6.3.5 Method 5

Method 5 is one which guarantees to remove all integer variables from the basis since we have access to all superbasic and all nonbasic continuous variables—including artificials. As mentioned in the previous section, M5 is identical to M4 except that M5 is allowed to consider artificials as candidates for changing status to superbasic for subsequent pivot with

---

1 This tries, in turn, to step each of the superbasic integer variables which is currently infeasible with respect to the integer requirements to the nearest integer.

2 Nonbasic to move away from bound.

3 Superbasic for $B \leftrightarrow S$ pivot.

4 Must be linearly independent of all current basic columns except possibly iprime; fixed and artificial variables are *not* considered.

5 Pivot basic i' with superbasic j^{**}.

an integer basic, whereas M4 is denied such freedom. It has been found that problems exist, eg Berry–Sugden (see the present Chapter 14) or Shanker–Tzen (see the present Chapter 13), for which access to artificials is necessary for M5, to terminate. The termination of M5 is guaranteed and is the subject of a theorem in section 6.4. Pseudocode for M5 is given in figure 21.

```
its := 0
pivot any integer-feasible basics with suitable superbasics
do while (numbasinf>0) & (ierrcode=NOERROR) & (its<maxits)
     increment iteration count
     try to step each infeasible superbasic feasibility[1]
     recompute number of basic integer-infeasibilities
     compute jstar[2]
     compute minimum ratio thetastar
     if event = 3 then
        do nonbasic step
     else
        seek jstarstar[3]
        if jstarstar can't be found then
           seek nonbasic, jns[4], suitable to go superbasic then basic
        endif
        if suitable nonbasic jns found then
           change status of column jns to superbasic
           label new superbasic column jstarstar
           do B<-->S pivot[5]
        else
           ierrcode := CANTFINDAJSTARSTAR
        endif
     endif
     recompute number of basic integer-infeasibilities
     pivot any integer-feasible basics with suitable superbasics
enddo
```

Figure 21 Pseudocode for M5

1 This tries, in turn, to step each of the superbasic integer variables which is currently infeasible with respect to the integer requirements to the nearest integer.

2 Nonbasic to move away from bound.

3 Superbasic for $B \leftrightarrow S$ pivot.

4 Must be linearly independent of all current basic columns except possibly i'. Fixed and artificial variables are considered.

5 Pivot basic i' with superbasic j^{**}.

6.4 Some theoretical properties of the new methods

It was noted in Chapter 5 that there may be no non-integer nonbasics for Murtagh's CYCLE1. This means that j^* does not exist and CYCLE1 cannot proceed. We are led therefore to consider allowing superbasic variables to be considered for direct pivot with basics in the hope that the number of integer-infeasibilities in the basis may be decreased.

Before deriving sufficient conditions for direct-search method M5 to terminate with no integer variables in the basis, we examine briefly two negative results. Lemmas 1 and 2 are presented in order to show that certain conditions are *not* sufficient for the termination of M5. For each of the following results, we assume the standard form of MINOS; corresponding to major iteration of MINTO, in which relinearization of constraints is made at the commencement of each major iteration. A full set of slack variables is present, ie the constraint matrix A contains the identity I_m as a submatrix. We assume that the continuous relaxation has been solved with local optimum at $\mathbf{x}^*$.

Lemma 1

There exists a NLIP in which the linearized constraint matrix contains a full set of slacks (ie I_m is a submatrix of A) for which Murtagh's cycle 1 fails to terminate except in the event of an error condition or iteration limit being reached.

Proof

Consider either counterexample 1 or counterexample 2 of Chapter 5.

Lemma 2

Consider the method M4. Then there exists a NLIP for which M4 fails to terminate except in the event of an error condition or iteration limit being reached.

Proof

All that is required here is a single example, and the problem reported by Shanker & Tzen [86] and discussed in the present Chapter 13 fills the bill.

Remarks

1. We note in all test problems that M5 terminates with zero integer-infeasibilities in the basis. This is no coincidence, and we next prove that, because we have access to a full set of slacks, some of which may be artificials, and we are permitted to use all slacks if necessary for pivot operations, that M5 *always terminates*.

2. From the two preceding lemmas, we see that M4 is not sufficient for cycle 1 to terminate, ie to remove all integer variables from the basis. We now show that M5 is sufficient for all problems. It should be noted also that, for certain problems, it is not *necessary* to invoke M5 to remove all integers from the basis, since, in particular, counterexample 1 of Chapter 5 terminated using the MINTO starting partition and invoking M3.

Theorem 1

Consider the class of problems defined in section 5.1, with the further assumption that a bounded, feasible solution exists. Then for any such problem there exists a basis containing ***no*** integer variables and this basis is attainable starting from the continuous solution and applying method M5.

Proof

Define $m_{IB} \quad = \quad | J_B \cap J_I |$, the number of basic integer variables.

The first observation that needs to be made is that the number of basic integer-infeasibilities must decrease by at least one on each iteration of the main *do while* loop of M5. Stated another way, we need to establish the *loop invariant* that m_{IB} decreases by at least one on each traversal of the loop. The result will then follow by induction on m_{IB}.

The only steps which can alter the number of basic integer variables are

(i) do nonbasic step
(ii) do $B \leftrightarrow S$ pivot
(iii) pivot any integer-feasible basics with suitable superbasics

Each of these is designed to pivot out integer-feasible basic variables. The worst case is that event 3 never occurs. Then either j^{**} or j_{NS} must be found. Since the basis is already nonsingular, we show that we can *always* find one, provided that $m_{IB} \geq 1$, ie that there is at least one integer variable still basic. Then, on each iteration, m_{IB} must decrease by at least one, and thus M5 terminates.

Now to complete the proof, we note the preconditions for the loop:

(i) $m_{IB} \geq 1$ (this must be so, else the loop has already terminated).

(ii) basis is nonsingular

Let there be m_A slack/artificial basic columns, where $0 \leq m_A \leq m-1$. These are elementary vectors (columns of the identity I_m). Then we have $m - m_A$ remaining columns of I_m which are either superbasic or nonbasic, and $1 \leq m - m_A \leq m$. Suppose none of these is suitable to pivot with column i' of the current basis. But B is a basis for E^m, so that B with column i' removed consists of $m-1$ linearly independent vectors, and is thus a basis for E^{m-1}. The remaining $m-1$ columns of B along with $m - m_A$ slack/artificial superbasic or nonbasic columns span E^m, since they contain all columns of I_m. Thus we have a basis for E^m. This is a contradiction, and the proof is complete.

Note on Theorem 1

The result just proved holds even for a mixed-integer linear program (MILP), however there is no guarantee that the integer-free basis so obtained corresponds to an integer-feasible point. In general, there will still be integer-infeasibilities in the superbasic variables, which may now be present even in a linear problem. Superbasics were introduced by Murtagh and Saunders [62] in order to cater for nonlinearities; they are being used here to also help in the quest for integer-feasibility, and so are applicable even to MILPs.

Chapter 7

Implementation of the new direct search methods

Both Murtagh's direct search method [59] and the proposed new methods were implemented initially as Pascal prototypes due to the superior algorithmic expressive capabilities of Pascal over FORTRAN, and also due to the superior high-quality development and debugging tools available on PCs in products such as Borland's *Turbo Pascal*®. The development machines were 80386/80387-based IBM or compatibles running PC-DOS 3.3 and MS-DOS 5.0. The Pascal compilers used were versions 5.0, 5.5 and 6.0 of Borland's Turbo Pascal®, all of which have excellent program preparation and debugging facilities.

Once a working prototype algorithm was fully developed and debugged, it was naïvely believed that a fairly simple, mechanical process would be required to hand-translate to FORTRAN and imbed the resulting code in Murtagh's MINTO NLIP optimizer. For various reasons, some of which are described in section 7.8, this was in fact *not* the case.

7.1 SPECS options

Two options were added to Murtagh's SPECS file, namely DIRECT SEARCH METHOD and FIX INTEGERS. These operate as follows.

DIRECT SEARCH METHOD n

The user may select any of five direct search methods M1–M5 or branch-and-bound (M0—this is the default), or interactive (99).

n	Effect
0	Branch-and-bound
1	Method 1 (Murtagh's heuristic)
2	Method 2
3	Method 3
4	Method 4
5	Method 5
99	Interactive

FIX INTEGERS YES
FIX INTEGERS NO

This option controls whether the integer-feasible variables at the termination of any of the direct search methods are held fixed for subsequent continuous reoptimization, followed by branch-and-bound if required. Default is YES.

7.2 Some obstacles encountered

It must be admitted that early versions of the direct search methods were rather primitive. A variety of test problems gave ample opportunity to hone these methods to the point where it could be reasonably claimed that they work moderately well on a wide variety of MINLPs.

Some difficulties were encountered when running the Shanker-Tzen problem described in Chapter 13. In particular, since the Shanker-Tzen problem is linear, there are no superbasic variables present at the continuous solution. Also, it is a highly-degenerate 0-1 problem, and has 6 feasible basics at the continuous solution. When this problem was encountered, the first approach was to change the status of a suitable nonbasic variable to superbasic and then pivot feasible integer basics out ($B \leftrightarrow S$). This works up to a point, however we end up with a lot of artificial variables in the basis, and when we try to do a nonbasic step later, no movement is possible. For example, in the Shanker-Tzen problem, a point is reached

where 4 integers are basic, compared with 18 at the continuous solution. Unfortunately, artificials prevent any movement since they are fixed at zero. Originally, the search for j_{ns} (nonbasic to be promoted to superbasic) did not consider artificial nonbasics, but this did not get very far—the supply of nonbasics was exhausted after the first two $N \rightarrow S$ operations—and no nonbasic j_{ns} could be found to become superbasic since all rows of the current basis inverse were orthogonal to all continuous nonbasic columns. For a number of problems M4 is very successful, but the above-mentioned difficulties first observed with the Shanker-Tzen model made it clear that M4 would not be adequate for some problems. Thus M5 was born.

7.3 Interactive display program

A special program was written in Borland Turbo C® to allow the user to direct the progress of the direct search algorithm. Some experience with it enabled various proposed methods to be tested and subsequently "hard-wired" into the direct search FORTRAN subroutine. The display program follows a spreadsheet paradigm in that updated values of the solution vector, objective, reduced gradient vector and other problem parameters are redisplayed after each step, which typically takes of the order of a second or less on a 20 MHz 80386/80387 machine with problem size 168 rows and 316 columns, of which 84 are integer (this is the Berry–Sugden model of Chapter 14).

The user can select from a menu, one option of which is to instruct MINTO/INTERACTIVE to proceed in the conventional "automatic" mode to the solution, without further intervention from the user. All functions available within the MINTO/INTERACTIVE system are described in the next section.

The C program was written in order to provide some interactive control of the direct search methods by the user. The user receives immediate feedback on the success or failure of an operation. For example, one may wish to manually select a nonbasic j^* and see the result of computing each *minimum ratio* θ for that particular j^*. If suitable (normally one is seeking event 3), then the user would opt to execute the step, thus removing an integer from the basis while simultaneously achieving integer-feasibility in that variable (see Chapter 5 and Chapter 6). The C code does considerable local checking, such as simple bound violations on proposed steps in superbasic variables, but for complex checks, control is passed back to the MINTO engine which for example, will check a proposed superbasic move for possible

simple bound violations on the basic variables. If such a violation occurs, an appropriate error status is returned to the C program and the user is then informed of the error with a descriptive message.

Apart from simple, atomic operations such as computing the minimum ratio θs, *entire direct search methods may be invoked directly from the C program*. As the iterations of the search proceed, updated information such as number of integer-infeasibilities, number of superbasic variables, etc is displayed as progress information for the user. It is indeed instructive to observe the progress of the various search strategies in this manner.

7.4 Functions available in MINTO/INTERACTIVE

The following keystrokes are used to control the operation of the interactive display program. A brief description of the effect of each is given.

[→] The [→] (horizontal tabulation) key is used to toggle the display mode from E to F format. In particular, this facility is useful to inspect the integer variables and quickly see the level of integer-infeasibility. For example, if the representation 39.02195 is displayed rather than the representation 0.3902195E+02, the integer-infeasibility 0.02195 (approximately) is much more easily discerned. On the other hand, very small values such as 10^{-11} are usually more conveniently displayed in scientific or engineering form as 0.1E-10 or 1.0E-11 or perhaps 10.0E-12.

[Esc] The escape key is used to request recalculation of the current solution. Control is transferred back to the MINTO engine which then recomputes the solution vector, gradient vector and objective function.

[A] This control switches automatic recalculation mode on or off. It is similar in function to the corresponding feature in a conventional electronic spreadsheet such as Lotus 1-2-3®.

D — This keystroke produces a pop-up screen which allows the user to switch each of sixteen debugging flags on or off. This facility is useful for testing new portions of code and to a certain extent, overcame the poor debugging facilities available for development of the MINTO/INTERACTIVE system (not easy to debug because of mixed-language programming).

R — Refresh screen. This function allows the user to rewrite the display screen, perhaps after some operation has overwritten part of the display, because of poor formatting.

Q — Quit to MINTO; continue in automatic mode. This terminates the interactive display program and MINTO proper regains control. The problem will then solve to completion using branch-and-bound if required, ie if any integer-infeasibilities remain.

↑ — Move cursor up one row on the display—similar to spreadsheet.

↓ — Move cursor down one row on the display—similar to spreadsheet.

Home — Move cursor to top of arrays.

End — Move cursor to end of arrays.

PgUp — Scroll up one screenful.

PgDn — Scroll down one screenful.

> — Increase superbasic by the current value of Δx. The cursor must be positioned on a row corresponding to a superbasic variable, and the operation is disallowed if this is not the case. Feasibility with respect to the general linearized constraint set is automatically checked and the operation denied if the move would violate a constraint. If acceptable, the move is made and the recomputed objective and solution redisplayed.

< — Decrease superbasic by the current value of Δx. See comments above for increase superbasic.

F1 Entry to on-line help system. The initial help screen will pop up, and the user may page down to further help screens. See figures 24–28 for snapshots of the help screens.

F2 Changes the status of the current nonbasic to superbasic. The cursor must be positioned on a row corresponding to a nonbasic variable.

F3 Uses the refined CTAC ’89 heuristic [59] to search for a suitable j^* for a nonbasic move.

F4 Pivots the currently-selected superbasic j^{**} with the currently-selected integer basic i'.

F5 Calculates the minimum ratio θ^* for a nonbasic move.

F6 Execute nonbasic move after θ^* calculated.

F7 Set the value of Δx for a subsequent superbasic move.

F8 Calculate superbasic index j^{**} for basic/superbasic pivot.

F9 Automatically calculate nonbasic using heuristic to become superbasic and select as j^{**} for subsequent basic/superbasic pivot. This nonbasic column must be linearly independent of the current basis columns, with the exception of the basic column i'. Equivalently, the basis nonsingularity invariant must be preserved after the proposed pivot operation.

F10 Changes the status of the current superbasic to nonbasic. The cursor must be positioned on a row corresponding to a superbasic variable. This function is essentially the inverse of that described above for the keystroke F2, however it is more complex and has not yet been fully implemented.

Ins Selects/deselects the current basic or nonbasic as i', j^* respectively. When a row is selected, the background colour on the screen will change to reflect the new status. Likewise, when deselected, the background colour will revert to the default.

Shift-Ins	As above but for the j^{**} selection.
Del	Cancel all selections, ie deselect all rows.
Shift-F4	Try to automatically pivot all integer-feasible basics into the superbasic set. This operation is necessary before the direct various search methods may commence, and in fact, is built-in to all Mn.
Shift-F5	Tries to achieve a decrease in the number of superbasic integer-infeasibilities by selecting superbasics in pairs to be stepped to integer-feasibility.
Shift-F7	Force the current superbasic variable to the user-supplied value. The cursor must be positioned on a row corresponding to a superbasic variable, and the operation is disallowed if this is not the case. Any bound violations are displayed as flashing values after the operation, and an error message may be displayed.
Gray-+	Step current superbasic to next integer. The cursor must be positioned on a row corresponding to a superbasic variable, and the operation is disallowed if this is not the case. This operation is convenient to check for feasibility of a proposed superbasic step. It will be disallowed if any constraint violation occurs.
Gray--	Step current superbasic to previous integer. See comments above.
Gray-*	Toggle screen display 25/28/43 line modes. For VGA screens, this option allows many more data rows for the problem to be displayed as a single screenful (28 rows in 43-line mode).
C	Clear iteration count. If an attempted method has become stuck in a loop and used up all its iterations, this option allows the user to reset the counter and try another method without having to restart the program.
I	Fix current integer-feasible variable at its current value.

1	Automatically run method M1.
2	Automatically run method M2.
3	Automatically run method M3.
4	Automatically run method M4.
5	Automatically run method M5.

7.5 Sample screens from MINTO/INTERACTIVE

This section presents some screen snapshots of the MINTO/INTERACTIVE system.

	i	hbinv	hs	bl	x	bu	gradient	red g
♦	1	4	3	0	40.91594	80	-2.86e-07	1e-14
♦	2	2	3	0	39.65044	79	-1.82e-07	-2.62e-15
♦	3	3	3	0	40.55087	80	-1.13e-06	-2.13e-15
♦	4	1	3	0	39.02059	79	7.19e-07	-9.43e-15
♦	5	5	3	0	40.47456	80	-6.01e-07	-0.115
	6	7	2	0	40.00000	80	-2.93e-07	0
	7	6	2	0	40.00000	80	-1.46e-07	0
	8	8	2	0	40.00000	80	7.06e-08	0
	9		0	0	0.00000	40	3.82e-07	0
	10		0	0	0.00000	40	0	0

newx	0	#Binf	5	j*	0	i1	0	j**	0	m	5	imn	0
dx	0.1	#Bfeas	0	j*shrt	0	i2	0	σ1	0	n	31	icsr	0
obj	6.3318e-13	#Sinf	0	j*long	0	i3	0	jmin	1	ns	4	imx	9
θ1	0	#Sfeas	0			event	0	j’’	0	nl	22	pgsz	10
θ2	0	auto	OFF	art?	N	i’0	1	jsup	0	nu	0	csr	4
θ3	0	opcode	32			i’1	5	jns	0	ni	5		
θ4	0	errcode	0	its	0	i’	1	σ’	-1				

F1	-Help	F2	-N-->S	F3	-Calcj*	F4	-BSpiv	F5	-Calcθ	F6	-NStep
F7	-Setdx	F8	-Calcj**	F9	-AutoNS	F10	-S-->NL	F11	-FixInts	F12	-Quit!

Figure 22 MINTO/INTERACTIVE displays continuous solution for a small QIP

	i	hbinv	hs	bl	x	bu	gradient	red g
♦	1	8	2	0	41.00000	80	2.83	-10.8
♦	2	12	2	0	40.00000	79	0	5.25
♦	3	11	2	0	40.00000	80	-9.42	74
♦	4	10	2	0	39.00000	79	0.667	12
♦	5	13	2	0	40.00000	80	2.75	-7.27
	6	7	2	0	40.00000	80	-2.93e-07	-55.5
	7	6	2	0	40.00000	80	-1.46e-07	-11.4
	8	3	3	0	35.29167	80	0.168	-43.7
	9			0	0.00000	40	3.82e-07	0
	10			0	0.00000	40	-0.0412	0

newx	0	#Binf	0	j*	0	i1	5	j**	8	m	5	imn	0
dx	0.074409	#Bfeas	0	j*shrt	13	i2	2	σ1	1	n	31	icsr	4
obj	26.835	#Sinf	0	j*long	0	i3	3	jmin	1	ns	8	imx	9
θ1	10.069	#Sfeas	5			event	0	j’’	0	nl	18	pgsz	10
θ2	4.2462	auto	OFF	art?	N	i’0	0	jsup	13	nu	0	csr	8
θ3	0.33333	opcode	23			i’1	0	jns	0	ni	5		
θ4	40	errcode	0	its	5	i’	0	σ’	0				

F1 -Help F2 -N-->S F3 -Calcj* F4 -BSpiv F5 -Calcθ F6 -NStep
F7 -Setdx F8 -Calcj** F9 -AutoNS F10-S-->NL F11-FixInts F12-Quit!

Figure 23 MINTO/INTERACTIVE displays integer-feasible solution for a small QIP

```
   i  hbinv hs     bl            x             bu        gradient       red g
♦   1                HELP FOR MINTO/INTERACTIVE ──────── Page 1/5        -10.8
♦   2  1                                                                 5.25
♦   3  1  MINTO/INTERACTIVE is a nonlinear integer optimizer.              74
♦   4  1                                                                  12
♦   5  1  The interactive display program allows the user to direct     -7.27
    6      the progress of the direct search algorithm of Murtagh &     -55.5
    7      Sugden. The user may request elementary operations such as   -11.4
    8      autocalc of minimum ratio θ*, or simply ask that predefined  -43.7
    9      heuristics designed to achieve integer-feasibility be run.       0
   10      Local optimality with respect to the superbasic variables        0
           may also be checked automatically or manually.
newx       Variables are displayed in natural, not partition, order,        0
dx         and diamonds indicate integer variables.                r        0
obj                                                                         9
θ1         Elements of the partition are colour coded as follows:       z  10
θ2                                                                          4
θ3             lower       upper       super       basic
θ4                                                   PgDn

F1 -Help     F2 -N-->S    F3 -Calcj*   F4 -BSpiv    F5 -Calcθ    F6 -NStep
F7 -Setdx    F8 -Calcj**  F9 -AutoNS   F10-S-->NL   F11-FixInts  F12-Quit!
```

Figure 24 Help screen #1 for MINTO/INTERACTIVE

```
   i  hbinv hs     bl            x             bu        gradient       red g

♦   1                HELP FOR MINTO/INTERACTIVE          Page 2/5          -10.8
♦   2  1                                                                   5.25
♦   3  1  TAB     Toggle display mode (E or F format)                       74
♦   4  1  ESC     Recalculate current solution (back to MINTO)              12
♦   5  1  A       Toggle autocalc mode                                    -7.27
    6     D       Toggle debug switches                                   -55.5
    7     R       Refresh screen                                          -11.4
    8     Q       Quit to MINTO; continue in automatic mode               -43.7
    9     S       Toggle silent mode (error buzz ON/OFF)                      0
   10     ↑       Move cursor up                                              0
          ↓       Move cursor down
newx      HOME    Move cursor to top of arrays                                0
dx        END     Move cursor to end of arrays                          r     0
obj       PgUp    Scroll up one screenful                                     9
θ1        PgDn    Scroll down one screenful                             z    10
θ2         >      Increase superbasic by deltax                               4
θ3         <      Decrease superbasic by deltax
θ4                                                        PgUp/PgDn

F1 -Help     F2 -N-->S    F3 -Calcj*   F4 -BSpiv    F5 -Calcθ    F6 -NStep
F7 -Setdx    F8 -Calcj**  F9 -AutoNS   F10-S-->NL   F11-FixInts  F12-Quit!
```

Figure 25 Help screen #2 for MINTO/INTERACTIVE

```
   i  hbinv hs     bl            x             bu        gradient       red g
♦   1                HELP FOR MINTO/INTERACTIVE          Page 3/5        -10.8
♦   2  1                                                                 5.25
♦   3  1  F1        Entry to on-line help system                          74
♦   4  1  F2        Current nonbasic goes superbasic                      12
♦   5  1  F3        Calculate j* for nonbasic move                     -7.27
    6     F4        Basic <--> Superbasic pivot                        -55.5
    7     F5        Calculate θ* for nonbasic move                     -11.4
    8     F6        Do nonbasic move                                   -43.7
    9     F7        Define deltax for super move                           0
   10     F8        Calculate nonbasic index j** for move                  0
          F9        Autocalc Nonbasic --> Superbasic
newx      F10       Superbasic to nonbasic lower                           0
dx        F11       Fix integers at current values                r        0
obj       F12       QUIT! - Return to DIRSCH + shutdown                    9
θ1        Ins       Toggle basic/super/nonbasic selection         z       10
θ2        ShiftIns  Toggle j** selection                                   4
θ3        Del       Cancel all selections
θ4                                                   PgUp/PgDn

F1 -Help     F2 -N-->S    F3 -Calcj*   F4 -BSpiv    F5 -Calcθ    F6 -NStep
F7 -Setdx    F8 -Calcj**  F9 -AutoNS   F10-S-->NL   F11-FixInts  F12-Quit!
```

Figure 26 Help screen #3 for MINTO/INTERACTIVE

```
   i  hbinv hs     bl            x             bu        gradient       red g
♦   1                HELP FOR MINTO/INTERACTIVE          Page 4/5 6.47e-09
♦   2  1                                                          1.57e-08
♦   3     ShiftF4   - Auto pivot IF basics to superbasic          9.56e-09
♦   4  1  ShiftF5   - Step superbasics in pairs                   .48e-323
♦   5     ShiftF7   - Set value for current superbasic                   0
♦   6     GreyPlus  - Step current superbasic to next integer            0
♦   7     GreyMinus - Step current superbasic to previous integer        0
♦   8     Grey *    - Toggle screen display 25/28/43 line modes          0
♦   9     C         - Clear iteration count                              0
♦  10  1  F         - Toggle Fixed/Artificial variables for jns          0
          I         - Fix current integer-feasible variable
newx      1         - Invoke direct search method M1                     0
dx        2         - Invoke direct search method M2                r    0
obj       3         - Invoke direct search method M3                     9
θ1        4         - Invoke direct search method M4                z   10
θ2        5         - Invoke direct search method M5                     4
θ3
θ4                                                      PgUp/PgDn
F1 -Help     F2 -N-->S    F3 -Calcj*   F4 -BSpiv    F5 -Calcθ    F6 -NStep
F7 -Setdx    F8 -Calcj**  F9 -AutoNS   F10-S-->NL   F11-FixInts  F12-Quit!
```

Figure 27 Help screen #4 for MINTO/INTERACTIVE

```
   i  hbinv hs     bl            x             bu        gradient       red g

♦   1                HELP FOR MINTO/INTERACTIVE          Page 5/5        -10.8
♦   2  1                                                                 5.25
♦   3  1  Display Headings                                                 74
♦   4  1                                                                  12
♦   5  1  i        - natural index of current variable x[i]            -7.27
    6     hbinv    - partition index of current natural variable       -55.5
    7     hs       - partition indicator:  0=NL; 1=NU; 2=S; 3=B        -11.4
    8     bl       - lower bound of current natural variable           -43.7
    9     x        - value of current natural variable                     0
   10     bu       - upper bound of current natural variable               0
          gradient - component of gradient vector
newx      red g    - component of reduced gradient vector                 0
dx                                                                  r     0
obj                                                                       9
θ1                                                                  z    10
θ2                                                                        4
θ3
θ4                                                        PgUp

F1 -Help     F2 -N-->S    F3 -Calcj*   F4 -BSpiv    F5 -Calcθ    F6 -NStep
F7 -Setdx    F8 -Calcj**  F9 -AutoNS   F10-S-->NL   F11-FixInts  F12-Quit!
```

Figure 28 Help screen #5 for MINTO/INTERACTIVE

```
   i  hbinv hs     bl            x             bu        gradient       red g
                        TOGGLE DEBUG SWITCHES
♦   1                                                                    -10.8
♦   2  1  √ 0 Solution vector in natural order                            5.25
♦   3  1    1 Partition index vectors                                       74
♦   4  1    2 Soln vec, grad, red grad in ptn order                         12
♦   5  1  √ 3 IOPB (input/output parameter block                         -7.27
    6       4 Checkpoints in super basic move check                     -55.5
    7       5 Generated alpha columns                                   -11.4
    8       6 Computed basics for super move check                      -43.7
    9       7 Predicted basic for C1P2                                      0
   10     √ 8 Parameters from calc theta routine                           0
            9 Parameters from calc z tranpose routine
newx        A CALCG parameters                                              0
dx          B UPDATEXFGH major checkpoints                            r     0
obj       √ C CALCJS parameters                                             9
θ1          D Alpha sub jstar in CALCTHETA                            z    10
θ2        √ E Results of super move check                                   4
θ3          F Progress of IF basics to super pivot
θ4
            <del> - cancel all dumps      <ins> - select all dumps
F1 -Help                                                                Step
F7 -Setdx    F8 -Calcj**  F9 -AutoNS   F10-S-->NL   F11-FixInts  F12-Quit!
```

Figure 29 Debug screen for MINTO/INTERACTIVE

7.6 The FORTRAN workhorse routines

This section gives a very brief overview of the major FORTRAN subroutines which form the building-blocks for implementation of the various direct search methods.

Name	Purpose
ASSERT	A very short but effective routine which simply checks that a claimed predicate is in fact true. If so then control returns to the caller, otherwise execution is aborted with a caller-supplied diagnostic message. Sample call: `call assert(iprime .gt. 0,'CALCJNS:: iprime <= 0')`
C1P1STEP	Pivot the current basic integer variable i' with a chosen superbasic variable j^{**}.
C1P2STEP	The abbreviation is for CYCLE 1, PHASE 2 step. A nonbasic step causes one of four possible outcomes. We seek integer-feasibility for our selected basic (event #3). If a change of basis occurs (events 1–3) then MODLU is called to update the sparse basis factorization.
C2P1	The abbreviation is for CYCLE 2 PHASE 1. This involves a step for an integer-infeasible superbasic variable to achieve integer feasibility.
CALCINF	Compute many parameters relating to integer-infeasibilities: calculate min/max integer-infeasibilities: scan all basics and superbasics and compile indices of min/max integer-infeasibilities for both basics and superbasics separately. Also detect (super)basic integers which are in fact integer-feasible. Count the number of integer-feasible integer variables, both basic and superbasic.
CALCJNS	Find a nonbasic suitable for becoming superbasic and ultimately to be pivoted into the basis. The corresponding column of the linearized constraint matrix A must not be (nearly) orthogonal to row i' of the basis inverse, where i' is the integer basic variable to be pivoted.
CALCJS	Calculate j^*—a nonbasic which *prima facie* has the best chance of forcing our basic to integer-feasibility while not worsening the objective too much.

Table 3 The FORTRAN subroutines Part A

CALCJSS	Calculate j^{**}—a superbasic which, when pivoted into the basis, the basis remains nonsingular.
CALCTH	The "minimum ratio test". Computes the four thetas, corresponding to limits imposed on the move of a nonbasic by basics reaching bounds, basics going integer-feasible, or nonbasic reaching the other end of its range.
CALCZT	Compute row i' of the basis inverse, where i' is the current basic selected for step to integer-feasibility.
CHUZQI	Choose a column q corresponding to a continuous superbasic variable which will become j^{**} (see entry for CALCJSS above).
DIRSCH	This is the main control routine for the direct search methods and subfunctions which may be invoked from the interactive display program. It contains the command interpretation loop and calls to many of the routines listed in this section.
DUMPCOL4 DUMPCOL8 DUMPICOL	These are utility routines useful for debugging—they write floating-point or integer columns (vectors) to the output file.
DUMPIOPB	Writes the contents of the IOPB (input/output parameter block) common block to the output file. This block is used for communication with the interactive display program, which is written in C.
DUMPPTN	Writes the current extended simplex partition information to the output file.
DUMPXNAT	Writes the current solution vector in natural order to the output file.

Table 4 The FORTRAN subroutines Part B

ERRMSG	Definition of parametrized error messages common to FORTRAN and C routines.
ERRS	Definition of parametrized error codes common to FORTRAN and C routines.
FIXINTS	Redefine bounds on all integer-feasible integer variables so as to fix them at the current value. Used when exiting from the direct search routine with no integer-infeasibilities so that a continuous resolve will not alter the integer values obtained at so much effort.
ICHKNAT	See if *i*th NATURAL variable is an integer.
ICHKPTN	See if *i*th PARTITION variable is an integer.
IFBTOS	Pivot integer feasible basics to superbasic. Needed in particular for Berry model, in which 9 basics are integer-feasible at the continuous solution. Precondition: CALCINF has been called or i' is otherwise up-to-date (eg by manual selection in disp.c).
INITIOPB	Initialize the IOPB common block structure.
IOPB	Definition of the IOPB (include file). This file automatically generated by the OPGEN program (see section 7.7).
NATTOPAR	Returns the partition index of natural variable *i*, ie inverts Murtagh's hb index.
NTOS	Change status of nonbasic jns (optionally automatically selected by this routine to be suitable for subsequent basis pivot—see CHUZQNS/CALCJNS above) to superbasic. Also select as j^{**} for subsequent pivoting into basis with basic i'.
OPMSG	Definition of parametrized operation messages common to FORTRAN and C routines.
OPS	Definition of parametrized operation codes common to FORTRAN and C routines.

Table 5 The FORTRAN subroutines Part C

SMOV2	Assuming that all attempts to separately step integer superbasic variables have failed, this routine tries simultaneous moves (four possibilities) for two currently-chosen superbasic variables. Deltax1 and deltax2 are the POSITIVE forward steps separately required for integer feasibility in the superbasics jsuper1, jsuper2. Precondition: no integers in basis.
SPMOVCHK	Check if proposed step in superbasic variable x[iopb.jsuper] in PARTITION order is feasible.
SUPERADJ	Tries to step each integer-infeasible superbasic integer variable to feasibility, checking after each one if by chance any basics have become feasible—this actually happens in some highly-degenerate 0-1 problems such as Shanker, Berry. The present routine is called during method loops M4, M5 in particular. NB This routine is NOT the same in function as C2P1 or C2P2.
SUPERMOV	Moves superbasic jsuper by deltax (positive or negative) and updates x, f, g but first checks for constraint or simple bound violations by calling SUPERMOVECHK.
UPDATXFG	Returns latest function value gradient, and reduced gradient. Also updates pricing vector and reduced gradient norm.

Table 6 The FORTRAN subroutines Part D

7.7 Utility programs

During the development process, some simple but useful software tools were written. The fundamental computing principle of having a single defining occurrence for each object that a computer is going to eventually process, coupled with suitable tools for converting such objects to the various forms required by existing software, was the overriding objective. The present writer has a particular aversion to the reinvention of specific, well-known wheels.

Accordingly, many tools were written—the primary ones being a FORTRAN dump code generator, an MPS generator, a QIP generator and a generator for iopb common block include file, error codes and error messages strings. Since mixed language programming was used, it was very awkward to keep a single point of definition for parameters: operation codes, error codes, messages etc so as to have consistency between the FORTRAN and C routines. Pascal programs were written to automatically generate the C and FORTRAN source files from a single definition of these parameters. Each utility program is described in further detail below.

7.7.1 MPS generator

At an intermediate stage of the development and testing for the direct search techniques, it was realised that a utility which would accept an algebraic description of a MILP and then write a corresponding MPS file would be of great benefit. MPS files are not noted for their ease of comprehension by human readers, but are deliberately designed to be a suitable input format for MP problems. A simple linear expression parser and detection of simple bounds plus building of symbol table were the main requirements, so that this program was developed relatively quickly using Borland's *Turbo Pascal®, version 6.0.*

Once this tool was available, problems could be expressed succinctly in normal algebraic notation, and then the generator invoked to produce the MPS file. Since the original problem was available in algebraic form, it could then be included in the present document in the knowledge that no corruption of data had occurred, because of re-keying or other unnecessary and error-prone duplication of effort.

7.7.2 QIP generator

For initial testing of the direct search methods, some small linearly-constrained QIPs were very useful. These were provided by a special purpose program, GENQIP, written in *Turbo Pascal®*.

7.7.3 IOPB generator

Another program was written in *Turbo Pascal®* to read several definition files containing error codes, error message strings, operation codes, operation code descriptions, definition of scalar parameters which would ultimately become elements of a FORTRAN COMMON BLOCK iopb (input/output parameter block), and correspondingly, a C structure. This approach ensured that communication between the C and FORTRAN code was based on a consistent, single point of definition for all objects involved. Since only the starting address of the FORTRAN common block is passed to the C routines, much stack overhead could be avoided (only one pointer rather some 50 parameters was passed), to say nothing of increased code readability and maintainability. As with most efforts toward code improvement and generalization, this approach involved some considerable work initially, but paid handsome dividends as the development work proceeded.

Nevertheless, many frustrating hours were spent trying to debug very simple errors which, while not detected in FORTRAN until run-time or link-time, would not have even got past the compiler if a more modern language such as MODULA-2 had been used. Some further comments on the limitations of FORTRAN appear in section 7.8.

7.8 Some FORTRAN traps

> *The decline and fall of the Roman number system provides an interesting case study on the importance of codes and representations. It shows that a long time is needed to overthrow established systems, even when the alternatives are far superior. A modern example of this is the QWERTY keyboard, which is used on almost every typewriter and terminal. It is known that alternative layouts can improve productivity by more than 30 percent, but who will make the effort to*

change? What other examples of this phenomenon can you think of? What about computer character codes or ***programming languages****. (emphasis ours) — Hext [37].*

The following paragraphs contain some brief observations from an experienced FORTRAN programmer who prefers to avoid the language wherever possible. Despite the excellent quality of the Lahey FORTRAN77 compiler (with useful extensions), the fundamentally primitive nature of the FORTRAN language caused a very considerable amount of wasted time when developing and debugging the MINTO/INTERACTIVE system. It should be noted that the FORTRAN language only supports *independent* compilation as distinct from *separate* compilation, which is a feature of modern languages such as MODULA-2, Ada®, or even Turbo Pascal® (versions 4.0*ff*). For definitions of these terms, the reader may consult, for example the excellent book by Booch [3].

It is worthy of note that when the United States Department of Defense requested submissions for the design of a new language for all *embedded systems* software development (culminating in *Ada®*), *all five shortlisted designs were based on Pascal*, a true block-structured language unlike FORTRAN or C.

Problems experienced

1. It was necessary to be very careful when using array indexing. If INTEGER*2 instead of INTEGER*4 were used, the results were essentially unpredictable, certainly without delving into low-level details. It should *not* be necessary for an application programmer in the 1990s to have to resolve such issues — such menial tasks can be performed with ease by the programming language compiler if the language definition is sufficiently precise.

2. It is rather inconvenient to have to pass several dozen parameters, with resulting problems with line-length just to get variable array dimensioning. It is indeed ironic in a language which does not permit long identifiers that problems of this kind are encountered.

3. Debugging facilities were rather primitive, although any debugger is better than none. The loose type-checking and implicit typing inherent in FORTRAN are the root causes of so many *unnecessary* (from the point of view of programming-language design) debugging problems that they simply do not bear mentioning. One example only:

forgetting to include a required *include file* fails to elicit an error diagnostic from either compiler or linker when an undeclared object is referenced, because of the archaic FORTRAN system of implicit typing.

4. Poor subprogram interface checking — only at run-time can this be done in FORTRAN unless one has access to a language-sensitive linker which is aware of the independent compilation of FORTRAN, C and related languages — unlike languages such as Ada and MODULA-2.

5. FORTRAN is so woeful when it comes to I/O, especially screen I/O that it was decided to write the interactive display program in C (another detestable language, however the Lahey F77 compiler used had no link to the preferred languages Pascal or MODULA-2). The FORTRAN–C interface means that even less type-checking and interface checking than usual in FORTRAN can be done. As noted elsewhere, a special program was written in Pascal to read files containing definitions of operation codes, error codes, parameter block and generate FORTRAN and C code to be included at compile time.

6. IMPLICIT NONE could not be switched on (to get maximum possible compile-time checking) because of existing large volume of code.

7. Run-time errors such as non-existent subroutine simply hang the system, whereas with a modern language such as MODULA-2 it would not even get past the compiler or linker!

Include files

Many *include files* were used during development. The prime advantage with this approach when writing in disparate languages is to ensure a single point of definition for parameters and common-block variables. It would be possible of course to write automatic checkers or source-code generators (as outlined above), but surely this highlights the fundamental limitations of the hopelessly dated approach of FORTRAN — even FORTRAN77 — to basic issues such as type-checking and declaration of all objects before use.

7.9 Ideas for future work

As the algorithmic development work proceeded, the rate at which new ideas for search procedures were being created far outstripped the rate at which such ideas could be implemented and tested. This is certainly not a new phenomenon, and is to be expected, given the current poor collection of high-level languages available for the rapid development of easily-maintainable mathematical programming software. Thus, at the time of writing, we still have several strategies which are as yet untested. A brief summary of those which seem to hold promise is given in this section.

Much time was spent on development work, both FORTRAN "workhorse" routines, and also the C interactive code. Some further work simply exercising this code would be most valuable since the basic tools are there for skilled user (ie optimization researcher) to experiment with various approaches based on Murtagh and Saunders' fundamental ideas of superbasic variables and search in the reduced space.

An idea inspired by the Shanker & Tzen problem of Chapter 13 is that, for the j^* heuristic, we may also need to relax the requirement of "short step". In fact, the development code has been altered to compute two j^*'s, viz one corresponding to a "short" step to integer feasibility for the chosen basic variable, and one corresponding to a "long" step. We prefer the short step, since this is more likely to stay closer to the continuous solution, or alternatively, suffer a smaller deterioration in objective. However, if no suitable j^* can be found for the short step, we may choose to take the long step. In fact, this has actually been implemented in the latest version of the code.

Concerning CYCLE1, ie movement of selected nonbasics in order to achieve integer feasibility in a chosen basic variable. A pre-condition of this part of the search is that we start at the continuous solution, even if some $B \leftrightarrow S$ pivot operations have already been done. An implication of this fact is that on the first iteration, the reduced gradients are zero (we are at a constrained continuous optimum). Thus, on the first iteration, we cannot estimate the change in objective on the basis of first-order information, nor can we choose j^* on the basis of "steepest descent". The reduced Hessian will in general give information adequate to find a nonbasic giving smallest increase of objective.

On subsequent iterations we wish to choose a nonbasic x_{j^*} to give steepest descent for objective *and* integer feasibility on an independently-chosen integer basic variable $x_{i'}$. This will of course not be possible in general, but it would be a shame to miss it if the chance came up. Therefore it is proposed that we should detect it, but degrade to another choice for j^* (see later) when it doesn't happen (most of the time)—it should not be too expensive

to compute. We also must of course maintain feasibility of other basics. We may therefore speak of a "feasible j" to get integer feasibility for our chosen basic which is hopefully a descent, but in general is chosen to minimize any increase in objective.

Since $\Delta f \quad = \quad \lambda_{j^*} \Delta x_{j^*}$, a first order estimate of Δf is known immediately for each eligible nonbasic. Scan such nonbasics and keep track of the best feasible j, ie such that Δf is smallest (most negative). The best feasible j will often give $\Delta f > 0$ (in fact this *must* be so initially). Further, there may be *no feasible j* at all, ie "event 3" doesn't happen this time around.

If this is the case then we accept one of the other outcomes as defined in Murtagh's paper [59]. Having already chosen $x_{Bi'}$, on every iteration we need to ask the question, "is the current nonbasic x_j a *feasible descent candidate*, ie is

$$\lambda_j \, signum(\Delta x_j) \quad < \quad -tolrg \quad ?$$

where *tolrg* is the reduced gradient tolerance. Also, is it going to take us in the correct direction to get the basic $x_{i'}$ feasible? We could easily go the wrong way and waste any benefit of having computed the basic with least integer-infeasibility. It is clear that there is some room for refinement of these heuristics.

Chapter 8

Computational experience I—Results for counterexamples

8.1 Introduction

The observations made in Chapter 5 were confirmed by running the counterexamples and using old and new methods. We present details only for the first of the examples below, since the others give no further insight.

8.2 Example 1—general comments

Running example 1 of Chapter 5 using MINTO/INTERACTIVE confirmed the non-termination of CYCLE1 of Murtagh's direct-search approach (the present M1). As diagnosed in Chapter 5, the method fails because there is not enough "freedom of movement". In particular, the presence of only one nonbasic j^* at the continuous solution, for which α_{2j^*} is small does not allow our target integer variable x_2 to move to integer feasibility before our solitary nonbasic hits its other bound.

Invocation of the present M4 followed by a simple superbasic step gives the (globally) optimal solution to this problem.

It is interesting to note also that our contrived continuous partition for this problem is not so different from the actual partition at the continuous MINTO solution. Our partition has x_1, x_2 basic; x_4, x_5 superbasic and x_3 nonbasic at lower bound 0; whereas the continuous MINTO solution has the same basis, but *no superbasics* (x_3, x_4, x_5 are *all nonbasic* at 0).

Nevertheless, application of Murtagh's procedure (the present M1) to this latter MINTO solution yields $\mathbf{x} = (2.2, 2, 1.0, 0.0, 0.4)^T$ with objective value 1.25, which is locally optimal with respect to the integer restriction but clearly inferior to the global optimum $\mathbf{x} = (1.2, 2, 0.0, 0.0, 0.5)^T$ with objective value 0.25 obtained by the present M4.

It should be noted however that continuous reoptimization after Murtagh's heuristic with x_2 fixed then gives the same result as M4. The final partition for M1 has x_1, x_5 basic; x_2 superbasic; x_3 nonbasic at upper bound 1; x_4 nonbasic at lower bound 0, whereas that for M4, while having the same basis, has x_2, x_4 superbasic, and x_3 nonbasic at lower bound 0.

As stated in Chapter 5 (table 2), it is clear that we need access to the superbasics even for CYCLE1 and even for simple linearly-constrained QIPs to achieve global optimality, or even to achieve integer-feasibility. Thus we conclude that there exist problems for which M1 is unsuitable, and in particular, *does not terminate*, except if the customary iteration limit guard is imposed. This result confirms the observations made in Chapter 5.

8.2.1 Objective function/gradient routine CALCFG

```
      SUBROUTINE CALCFG( MODE,N,X,F,G,NSTATE,NPROB )
      IMPLICIT   REAL*8(A-H,O-Z)
      REAL*8     X(N),G(N)
      COMMON    /IOCOMM/ IREAD,IPRINT
**
** counterexample 1
**
      f = (x(1)-1.2)**2 + (x(2)-2.5)**2 + x(3)**2
      g(1) = 2.0*(x(1) - 1.2)
      g(2) = 2.0*(x(2) - 2.5)
      g(3) = 2.0*x(3)
      g(4) = 0.0
      g(5) = 0.0
      RETURN
      END
```

8.2.2 MPS file

```
BEGIN   SPECS FILE FOR counter1
   MINIMIZE
   ROWS                     200
   COLUMNS                  400
```

```
   ELEMENTS                2600
   INTEGER VARIABLES          5
   NONLINEAR VARIABLES       25
   PRIORITY                  NO
   LIST LIMIT               200
   DIRECT SEARCH METHOD   99
   FIX INTEGERS             YES
END     SPECS FILE FOR counter1
*
*
*
NAME          random
ROWS
 E  ROW1
 E  ROW2
COLUMNS
    x1        ROW1            1.0000
*
              MARKER                   INTORG
    x2        ROW2            1.0000
              MARKER                   INTEND
*
    x3        ROW1            -1.000
    x3        ROW2             0.100
*
    x4        ROW1            1.0000
*
    x5        ROW2            1.0000
*
RHS
    B         ROW1          1.2
    B         ROW2          2.5
BOUNDS
 LO BD        x1              0.0000
 UP BD        x1              5.0000
 LO BD        x2              0.0000
 UP BD        x2              5.0000
 LO BD        x3              0.0000
 UP BD        x3              1.0000
 LO BD        x4              0.0000
 UP BD        x4              100.00
 LO BD        x5              0.0000
 UP BD        x5              100.00
ENDATA
```

8.2.3 Continuous solution

```
PROBLEM NAME   counter1     OBJECTIVE VALUE    0.0000000000E+00
SECTION 2 - COLUMNS
```

```
 NUMBER  .COLUMN.  AT  ...ACTIVITY...
      1  x1        BS         1.20000
      2  x2        BS         2.50000
A     3  x3        LL         0.00000
A     4  x4        LL         0.00000
A     5  x5        LL         0.00000
A     6  B         EQ        -1.00000
```

8.2.4 Output for method 0 (branch-and-bound)

```
PROBLEM NAME   counter1      OBJECTIVE VALUE    2.5000000000E-01
SECTION 2 - COLUMNS
 NUMBER  .COLUMN.  AT  ...ACTIVITY...
      1  x1        BS         1.20000
      2  x2        IV         2.00000
A     3  x3        LL         0.00000
A     4  x4        LL         0.00000
      5  x5        BS         0.50000
A     6  B         EQ        -1.00000
```

8.2.5 Output for method 1

```
PROBLEM NAME   counter1               OBJECTIVE VALUE    2.5000000000E-01
SECTION 2 - COLUMNS
 NUMBER  .COLUMN.  AT  ...ACTIVITY...
      1  x1        BS         1.20000
      2  x2        IV         2.00000
A     3  x3        LL         0.00000
A     4  x4        LL         0.00000
      5  x5        BS         0.50000
A     6  B         EQ        -1.00000
```

8.2.6 Output for methods 2 and 3

```
PROBLEM NAME   counter1               OBJECTIVE VALUE    2.5000000000E-01
SECTION 2 - COLUMNS
 NUMBER  .COLUMN.  AT  ...ACTIVITY...
```

```
      1  x1        BS         1.20000
      2  x2        IV         2.00000
A     3  x3        LL         0.00000
A     4  x4        LL         0.00000
      5  x5        BS         0.50000
A     6  B         EQ        -1.00000
```

Notes

1. It is interesting to observe that both M2 and M3 arrive at the correct global optimum *on their own!* However M2 does this with our starting partition and M3 with the MINTO starting partition.

2. For M2, starting with our partition (2 superbasics), the number of superbasics remains at 2. For M3, starting with MINTO partition (0 superbasics), the number of superbasics at the integer solution is 1.

8.2.7 Output for method 4

```
PROBLEM NAME   counter1               OBJECTIVE VALUE    2.5000000000E-01
SECTION 2 - COLUMNS
 NUMBER  .COLUMN.  AT  ...ACTIVITY...
      1  x1        BS         1.20000
      2  x2        IV         2.00000
A     3  x3        LL         0.00000
D     4  x4       SBS         0.00000
      5  x5        BS         0.50000
A     6  B         EQ        -1.00000
```

8.3 Results for example 2

These were identical to those for example 1.

8.4 Results for example 3

For this example, M1 works as expected (no superbasic step after termination of the method was required. M2 and M3 both fail to remove integers from the basis, while M4 succeeds on its own (no superbasic step required).

Chapter 9

Computational experience II—An example from Ravindran *et al*[1]

This section reports computational experience with a very small quadratic integer problem from Ravindran [77], 472*ff*. It is recognized that this problem is rather trivial, however we include it because of a notable comparison between certain of the new methods and the branch-and-bound integerizing procedure of Murtagh's MINTO code.

Description of the problem

maximize

$$13x_1 - 5x_2^2 + 30.2x_2 - x_1^2 + 10x_3 - 2.5x_3^2 \tag{197}$$

subject to

$$2x_1 + 4x_2 + 5x_3 \leq 10 \tag{198}$$

$$x_1 + x_2 + x_3 \leq 5 \tag{199}$$

$$x_1, \quad x_2, \quad x_3 \geq 0 \tag{200}$$

All x_j integer.

1 Ravindran, Phillips and Solberg.

A screen snapshot of the solution parameters after invocation of M4 is given in figure 30.

	i	hbinv	hs	bl	x	bu	gradient	red g
♦	1	4	2	0	3.00000	2.5e+04	0	-20.2
♦	2	3	2	0	1.00000	2.5e+04	0	-7
♦	3		0	0	0.00000	2.5e+04	-20.2	0
	4		0	-1	-1.00000	-1	-7	0
	5	1	3	0	0.00000	1e+20	0	0
	6	2	3	0	1.00000	1e+20	0	0

newx	0	#Binf	0	j*	0	i1	0	j**	0	m	2	imn	0
dx	-0.34444	#Bfeas	0	j*shrt	5	i2	0	σ1	1	n	6	icsr	0
obj	-55.2	#Sinf	0	j*long	0	i3	1	jmin	1	ns	2	imx	5
θ1	1e+20	#Sfeas	2			event	0	j’’	0	nl	2	pgsz	10
θ2	1e+20	auto	OFF	art?	N	i’0	0	jsup	4	nu	0	csr	4
θ3	0	opcode	32			i’1	0	jns	0	ni	3		
θ4	1e+20	errcode	0	its	1	i’	0	σ’	0				

F1 -Help	F2 -N-->S	F3 -Calcj*	F4 -BSpiv	F5 -Calcθ	F6 -NStep
F7 -Setdx	F8 -Calcj**	F9 -AutoNS	F10-S-->NL	F11-FixInts	F12-Quit!

Figure 30 Ravindran example optimal solution obtained using M4

Notes

1. The published optimal solution in Ravindran *et al* [77] is $\mathbf{x}^* = (3,1,0)^T$, with objective value 55.2. The continuous solution objective is 56.268.

2. MINTO using branch-and-bound with default node selection and branching options arrives at a suboptimal solution $\mathbf{x}^* = (1,2,0)^T$, with objective 52.4.

3. Using any of the present methods M2, M4 or M5 yields the published optimal solution *without branch-and-bound being required at all*. Further, no superbasic steps are required after termination of the Mn—the solution is integer feasible and optimal.

Chapter 10

Computational experience III—A plant upgrade problem

This problem is taken from Daellenbach, George and McNickle [11] and is cited in Mawengkang [48] where he solves it using the earlier direct search method of Mawengkang and Murtagh [48, 49].

Introductory description of problem

An electrical manufacturer wishes to upgrade its machinery as running costs are increasing as machines begin to wear out or become obsolete. The present plant cannot meet a recent increased demand. Two types of wire are produced by the manufacturer: bare wire and insulated wire. The problem ultimately reduces to the determination of the configuration of the new plant such that production costs are minimized while ensuring that demand is met. A more detailed description may be found in Daellenbach, George and McNickle [11].

Mathematical statement of the problem

Let y_i be the number of machines of options i, and x_{ij} be the proportion of annual machine time on machines of options i to produce wire size j, ($i = 1, \ldots, 5$ and $j = 1, 2$). The linear objective function is the total running cost, represented by P, and the problem is:

minimize

$$P = 30y_1 + 33.6x_{11} + 32.88x_{12} + 50y_2 + 47.4x_{21} + 47.04x_{32} + 80y_3 + 58.8x_{31} \tag{201}$$
$$+57x_{32} + 100y_4 + 62.4x_{41} + 54.7x_{42} + 140y_5 + 86.4x_{51} + 82.8x_{52}$$

subject to

$$5880x_{11} + 8820x_{21} - 7200x_{31} - 9600x_{41} \geq 3000 \tag{202}$$

$$4704x_{12} + 8232x_{22} - 6000x_{32} - 7800x_{42} \geq 2000 \tag{203}$$

$$6984x_{31} + 9312x_{41} + 9312x_{51} \geq 14000 \tag{204}$$

$$5820x_{32} + 7566x_{42} + 6984x_{52} \geq 10000 \tag{205}$$

$$\tag{206}$$

$$y_1 \geq x_{11} + x_{12} \tag{207}$$

$$y_2 \geq x_{21} + x_{22} \tag{208}$$

$$y_3 \geq x_{31} + x_{32} \tag{209}$$

$$y_4 \geq x_{41} + x_{42} \tag{210}$$

$$y_5 \geq x_{51} + x_{52} \tag{211}$$

$$y_1 \leq 2 \tag{212}$$

$$y_3 + y_4 \leq 1 \tag{213}$$

$$x_{ij} \geq 0, \qquad i = 1, \ldots, 5; \quad j = 1, 2 \tag{214}$$

$$y_i \geq 0, \quad \text{and integer}, \qquad i = 1, \ldots, 5. \tag{215}$$

Results

Results are tabulated overleaf and agree with the published values from Daellenbach, George and McNickle. Solution vectors and objective were identical in all cases.

Quantity	At continuous solution	After M1	After M2	After M3	After M4	After M5
# rows (incl obj), *m*	13					
# columns (incl slacks), *n*	30					
# integer variables, *ni*	5					
# integer-infeasible basics	2	0	3	0	0	0
# integer-feasible basics	3	0	0	0	0	0
# integer-infeasible superbasics	0	0	0	0	1	1
# integer-feasible superbasics	0	4	1	4	3	3
# superbasics	0	4	1	4	4	4
Objective	715.418	729.576	729.576	729.576	729.576	729.576
# nonbasics at lower bound, *nl*	13	9	12	11	9	9
# nonbasics at upper bound, *nu*	4	4	4	2	4	4
# integer-infeasibilities[1]	5	0	3	0	1	1

Table 7 Plant upgrade model minor parameters

1 This value is the sum of the number of basic integer-infeasibilities and the number of superbasic integer-infeasibilities.

Run mode[1]	Objective	# branch-and-bound nodes	Run time (sec)[2]	# degeneracies	MINOS Iterations[3]
Continuous	715.418	N/A	1	2	8
M0[4]	729.576	4	3	2	18
M1	729.576	0	3	2	15
M2	729.576	3	3	2	20
M3	729.576	0	3	2	21
M4	729.576	1	3	1	18
M5	729.576	1	3	1	18

Table 8 Plant upgrade model results summary

1 Integer-feasible variables fixed after each Mn terminates.

2 Run on a 20MHz 80386/80387 system with 32-bit code generation switched ON.

3 This is measured as the number of line-searches required.

4 Method 0 is branch-and-bound.

Chapter 11

Computational experience IV—A heat exchange network optimization problem

This problem is taken from Papoulias and Grossmann (1983) [70] and is cited in Mawengkang [48] where it is solved using the earlier direct search method of Mawengkang and Murtagh [49]. The problem is a mixed-integer linear program. In a chemical processing system, the heat exchanger network must integrate the hot and cold streams in a process so that the amount of heating and cooling utilities required is minimized. A more detailed description of this example may be found in the original paper by Papoulias and Grossmann [70].

Mathematical statement of problem

minimize

YAA01+YAB01+YAC01+YAD01+YAE01+YBA01+YBB01+
YBC01+YBD01+YBE01+YCA01+YCB01+YCC01+YCD01+
YCE01+YDA01+YDB01+YDC01+YDE01

subject to

QAD01	=	120.0
QAD01 + RA01	=	600.0
QAD02 + QBD02	=	240.0
QAD02 - RA01 + RA02	=	300.0

QBD02 + RB02	=	60.0
QAC03 + QBC03 + QCC03	=	160.0
QAD03 + QBD03 + QCD03	=	600.0
QAC03 + QAD03 - RA02 + RA03	=	750.0
QBC03 + QBD03 - RB02 + RB03	=	120.0
QCC03 + QCD03 + RC03	=	320.0
QAA04 + QBA04 + QCA04 + QDA04	=	200.0
QAB04 + QBB04 + QCB04 + QDB04	=	300.0
QAC04 + QBC04 + QCC04 + QDC04	=	320.0
QAA04 + QAB04 + QAC04 - RA03 + RA04	=	600.0
QBA04 + QBB04 + QBC04 - RB03 + RB04	=	0.0
QCA04 + QCB04 + QCC04 - RC03 + RC04	=	0.0
QDA04 + QDB04 + QDC04 + RD04	=	360.0
QAA05 + QBA05 + QCA05 + QDA05	=	400.0
QAE05 + QBE05 + QCE05 + QDE05	=	1179.9999
QAA05 + QAE05 - RA04 + RA05	=	150.0
QBA05 + QBE05 - RB04 + RB05	=	0.0
QCA05 + QCE05 - RC04 + RC05	=	0.0
QDA05 + QDE05 - RD04 + RD05	=	360.0
QAA06 + QBA06 + QCA06 + QDA06	=	100.0
QAA06 - RA05	≤	0.0
QBA06 - RB05	=	0.0
QCA06 - RC05	=	0.0
QDA06 - RD05	=	0.0
QAA04 + QAA05 + QAA06 - 700YAA01	≤	0.0
QAB04 - 300YAB01	≤	0.0
QAC03 + QAC04 - 480YAC01	≤	0.0
QAD01 + QAD02 + QAD03 - 960YAD01	≤	0.0
QAE05 - 1179.9999YAE01	≤	0.0
QBA04 + QBA05 + QBA06 - 180YBA01	≤	0.0
QBB04 - 180YBB01	≤	0.0
QBC03 + QBC04 - 180YBC01	≤	0.0
QBD02 + QBD03 - 180YBD01	≤	0.0
QBE05 - 180YBE01	≤	0.0
QCA04 + QCA05 + QCA06 - 320YCA01	≤	0.0
QCB04 - 300YCB01	≤	0.0
QCC03 + QCC04 - 320YCC01	≤	0.0
QCD03 - 320YCD01	≤	0.0

$$
\begin{aligned}
QCE05 - 320YCE01 &\leq 0.0 \\
QDA04 + QDA05 + QDA06 - 700YDA01 &\leq 0.0 \\
QDB04 - 300YDB01 &\leq 0.0 \\
QDC04 - 480YDC01 &\leq 0.0 \\
QDE05 - 720YDE01 &\leq 0.0
\end{aligned}
$$

and also subject to

$$
YAA01, YAB01, YAC01, YAD01, YAE01, YBA01, YBB01, YBC01, YBD01, YBE01, YCA01, YCB01, YCC01, YCD01, YCE01, YDA01, YDB01, YDC01, YDE01 \in \{0,1\}
$$

Results

Results for this problem are summarized in the tables overleaf.

Quantity	At continuous solution	After M1[1]	After M2	After M3	After M4	After M5
# rows (incl obj), *m*	47					
# columns (incl slacks), *n*	114					
# integer variables, *ni*	19					
# integer-infeasible basics	6	4	6	1	0	0
# integer-feasible basics	13	0	0	0	0	0
# integer-infeasible superbasics	0	0	0	0	4	4
# integer-feasible superbasics	0	15	13	18	15	15
# superbasics	0	15	13	18	19	19
Objective	5.6083	5.6083	5.6083	5.6083	5.6083	5.6083
# nonbasics at lower bound, *nl*	66	66	53	48	47	47
# nonbasics at upper bound, *nu*	1	1	1	1	1	1
# integer-infeasibilities[2]	7	4	6	1	4	4

Table 9 Heat exchange model minor parameters

1 This procedure cycled on nonbasic step, and failed to terminate.

2 This value is the sum of the number of basic integer-infeasibilities and the number of superbasic integer-infeasibilities.

Run mode[1]	Objective	# branch-and-bound nodes	Run time (sec)[2]	# degeneracies	MINOS iterations[3]
Continuous	5.6083	N/A	4	15	22
M0[4]	8.00	245	227	16	3742
M2[5]	8.00	17	10	18	84
M3	8.00	16	12	19	111
M4	8.00	12	11	17	78
M5	8.00	12	10	16	74

Table 10 Heat exchange model results summary

1 Integer-feasible variables fixed after each Mn terminates.

2 Run on a 20MHz 80386/80387 system with 32-bit code generation switched ON.

3 This is measured as the number of line-searches required.

4 Method 0 is branch-and-bound.

5 M1 not tabulated since this procedure cycled on nonbasic step, and failed to terminate.

Notes

1. This problem was solved successfully using the new direct search methods which yielded a minimum of 8.0, in agreement with the previously published minimum of Papoulis and Grossmann [70], who use the LINDO [85] code to obtain it. The minimum obtained by Mawengkang [48] for the same problem was 9.0 (this seems to be a typographical error on the part of Mawengkang).

2. For this problem, the heuristic of Murtagh (essentially the present M1) has problems with cycling. One benefit of MINTO/INTERACTIVE is that such cycling usually is obvious from the on-screen behaviour as updated parameters are displayed on each traversal of the search loop.

3. We observe that the present direct search methods (followed by a short branch-and-bound) achieve integer feasibility and indeed local optimality for this problem, in a much shorter time than branch-and-bound alone.

Chapter 12

Computational experience V—Two examples from Myers[1]

A recent contribution to linearly-constrained NLIP by Myers (1984) [66] examines several strategies for improving the efficiency of branch-and-bound and the use of Lagrangian relaxation for linearly-constrained mixed-integer problems. The twenty test problems he presents are all purely integer, linearly-constrained, and with separable nonlinear objective.

We have solved several of the test problems listed in Myers' dissertation, in some cases obtaining much-improved optima. Due to limitations of space, computational experience is reported with only the first two of Myers' problems here.

12.1 Myers problem #1

The Myers problem #1 is defined as follows

1 PhD dissertation of D.C. Myers.

minimize

$$f = 4.0\exp(x_1) + 5.0\exp(-0.4x_2) - 2.0x_3 + x_4^3 + 3.0x_4^2 + 0.1x_5^6 + x_6^2 - \ln(2.0x_7 + 1.0) - \ln(x_8 + 3.0) + x_9^2 - 4.0x_9 + x_{10}^3 + \sqrt{x_2 + 4.0} \tag{216}$$

subject to

$$-3x_1 - 2x_2 + 2x_4 - 5x_5 + 3x_6 - x_7 - 2x_9 \geq -26.6 \tag{217}$$

$$-2x_2 - 5x_3 + 4x_5 - 2x_6 + 2x_9 + 2x_{10} \geq 6.6 \tag{218}$$

$$7x_1 + 3x_2 + 3x_3 - 2x_4 + 6x_7 + x_9 + 2x_{10} \geq 57.7 \tag{219}$$

$$-3x_1 - 2x_2 - 3x_3 + 3x_4 - 4x_5 - x_6 + 4x_7 - 3x_8 - 3x_9 + 2x_{10} \geq -5.8 \tag{220}$$

$$x_1 + 2x_2 + 2x_3 - 3x_4 - 3x_5 - 2x_6 - x_8 + 6x_9 - 3x_{10} \geq -10.5 \tag{221}$$

$$2x_1 + x_2 - 2x_3 + x_4 + 3x_5 + 2x_7 + 3x_9 - 4x_{10} \geq 7.5 \tag{222}$$

$$-2x_1 + 5x_3 - 3x_4 - x_5 + x_6 - x_7 - x_{10} \geq -20.5 \tag{223}$$

$$5x_1 + 2x_2 + x_3 - x_4 + 4x_5 - x_6 - 2x_7 + 3x_8 + 3x_{10} \geq 35.1 \tag{224}$$

All x_j integer and nonnegative.

Quantity	At continuous solution	After M1	After M2	After M3	After M4	After M5
# rows, (incl obj) *m*	9					
# columns (incl slacks), *n*	20					
# integer variables, *ni*	10					
# integer-infeasible basics	5	2	5	2	0	0
# integer-feasible basics	0	0	0	0	0	0
# integer-infeasible superbasics	3	3	0	3	2	2
# integer-feasible superbasics	0	3	3	3	6	6
# superbasics	3	6	3	6	8	8
Objective	-0.88502	-0.7315	-0.4370	-0.7315	-0.4324	-0.4324
# nonbasics at lower bound, *nl*	3	3	3	3	3	3
# nonbasics at upper bound, *nu*	5	2	5	2	0	0
# integer-infeasibilities[1]	8	5	5	5	2	2

Table 11 Myers problem 1 minor parameters

1 This value is the sum of the number of basic integer-infeasibilities and the number of superbasic integer-infeasibilities.

Run mode	Objective	# Branch-and-bound nodes	Run time (sec)[1]	Function and gradient calls	# Degeneracies	MINOS iterations[2]
Continuous	-0.885	N/A	3	47	0	24
M0[3]	2.0406	164	60	1233	0	1104
M1	7.7187	14	9	183	0	95
M2	103.91	145	45	681	1	790
M3	7.7187	14	6	136	0	95
M4	29.981	88	29	448	0	475
M5	29.981	88	29	448	0	475

Table 12 Myers problem 1 results summary

1 Run on a 20MHz 80386/80387 system with 32-bit code generation switched ON.

2 This is measured as the number of line-searches required.

3 Method 0 is branch-and-bound.

Notes

1. The results of running the various methods for the present problem from Myers' dissertation are given in tables 11 and 12. With respect to Myers' published result [66], some improved optima were obtained using the direct search methods M1–M4, although, notably, none was able to obtain the MINTO's branch-and-bound (M0) solution having objective 2.04 (M3 did come reasonably close in a much shorter time than M0). Of course, the price paid for this improved minimum is extra computation time.

2. The minimum claimed by Myers for this problem is 98.33 at $(2, 3, 0, 1, 1, 0, 5, 5, 1, 4)^T$, but on evaluation of the objective at this point, the result is 94.33. In view of the lexical similarity of the two values, it would seem that 98.33 is a simple typographical error, and the correct value is in fact 94.33.

3. Only M2 fails to improve on the minimum found by Myers.

12.2 Myers problem #2

The Myers problem #2 is defined as follows

minimize

$$\begin{aligned} f \;=\; & -2.0\ln(x_1+1.5) + 3.0x_2 + \exp(0.4x_3) \\ & -\ln(3x_4+2) - 4.0x_4 + x_5^3 + x_6^4 \\ & + 2.0x_6^3 - 5.0x_6 + 4.0x_7^2 + 3.0\exp(-x_8) \\ & + x_8 + 4.0x_9^2 - 3.0\sqrt{x_{10}} \end{aligned} \tag{225}$$

subject to

$$-3x_1 - 2x_2 + 2x_4 - 5x_5 + 3x_6 - x_7 - 2x_9 \;\geq\; -26.6 \tag{226}$$

$$-2x_2 - 5x_3 + 4x_5 - 2x_6 + 2x_9 + 2x_{10} \;\geq\; 6.6 \tag{227}$$

$$7x_1 + 3x_2 + 3x_3 - 2x_4 + 6x_7 + x_9 + 2x_{10} \;\geq\; 57.7 \tag{228}$$

$$-3x_1 - 2x_2 - 3x_3 + 3x_4 - 4x_5 - x_6 + 4x_7 - 3x_8 - 3x_9 + 2x_{10} \;\geq\; -5.8 \tag{229}$$

$$x_1 + 2x_2 + 2x_3 - 3x_4 - 3x_5 - 2x_6 - x_8 + 6x_9 - 3x_{10} \quad \geq \quad -10.5 \tag{230}$$

$$2x_1 + x_2 - 2x_3 + x_4 + 3x_5 + 2x_7 + 3x_9 - 4x_{10} \quad \geq \quad 7.5 \tag{231}$$

$$-2x_1 + 5x_3 - 3x_4 - x_5 + x_6 - x_7 - x_{10} \quad \geq \quad -20.5 \tag{232}$$

$$5x_1 + 2x_2 + x_3 - x_4 + 4x_5 - x_6 - 2x_7 + 3x_8 + 3x_{10} \quad \geq \quad 35.1 \tag{233}$$

All x_j integer and nonnegative.

Quantity	At continuous solution	After M1	After M2	After M3	After M4	After M5
# rows, (incl obj) *m*	9					
# columns (incl slacks), *n*	20					
# integer variables, *ni*	10					
# integer-infeasible basics	5	3	5	1	0	0
# integer-feasible basics	1	0	0	0	0	0
# integer-infeasible superbasics	2	2	0	2	0	0
# integer-feasible superbasics	0	3	3	5	8	8
# superbasics	3	5	3	7	8	8
Objective	-4.1551	-0.491	-2.576	2.875	26.184	26.184
# nonbasics at lower bound, *nl*	3	3	3	3	3	3
# nonbasics at upper bound, *nu*	5	3	5	1	0	0
# integer-infeasibilities[1]	7	5	5	3	0	0

Table 13 Myers problem 2 minor parameters

1 This value is the sum of the number of basic integer-infeasibilities and the number of superbasic integer-infeasibilities.

Run mode	Objective	# Branch-and-bound nodes	Run time (sec)[1]	# Function and gradient calls	# Degeneracies	MINOS iterations[2]
Continuous	-4.155	N/A	2	24	0	16
M0[3]	23.38	102	93	1717	0	1799
M1	26.19	19	9	170	0	93
M2	26.19	14	5	81	0	56
M3	140.3	29	8	125	0	122
M4	26.18	0	2	34	0	24
M5	26.18	0	2	34	0	24

Table 14 Myers problem 2 results summary

1 Run on a 20MHz 80386/80387 system with 32-bit code generation switched ON.

2 This is measured as the number of line-searches required.

3 Method 0 is branch-and-bound.

Notes

1. The results of running the various methods for the present problem from Myers' dissertation are given in tables 13 and 14. Once again, with respect to Myers' published results [66], some improved optima were obtained using the direct search methods M1–M4. In fact, all Mn but M3 improved on Myers' published value of 28.55.

2. Our M0, ie branch-and-bound, achieves an objective of 23.38 which is superior to Myers' published value of 28.55.

3. M4 and M5 achieve integer-feasibility with no need for branch-and-bound. This much-reduced computation time is of course at the expense of a suboptimal solution. However, the solution obtained in this way is still superior to Myers' published solution.

Chapter 13

Computational experience VI—A loading and dispatching problem in a random flexible manufacturing system

In this chapter we report computational experience in solving an example of a well-known class of problems being increasingly reported in the optimization literature. We refer to the class of loading and dispatching problems in a random flexible manufacturing system, and the example considered in this chapter was reported by Shanker and Tzen (1985) [86].

Detailed accounts of this problem are given in the original paper by Shanker and Tzen [86], and also in the dissertation of Mawengkang [48], who solved it using a similar approach to one of the present direct search direct techniques. Mawengkang also presents a useful summary of previous attacks on this class of problems, including variations on the ubiquitous branch-and-bound approach. Accordingly, only a brief description and algebraic formulation will be given here. In a nutshell, the problem is in fact a mixed-integer linear program (MILP) involving 41 binary variables, 8 continuous variables and 29 constraints. An original formulation contained one nonlinear constraint, but this is easily transformed out of the problem. Both formulations are given in Shanker and Tzen [86], and we consider here the linear formulation only.

Brief description of the problem.

The following gives only the essential variables involved in the example being considered. For full details of the general model, the reader may consult the paper by Shanker and Tzen [86]. The objective is designed for workload balancing and minimization of late jobs.

Decision variables

$$x_i = \left\{ \begin{array}{ll} 1; & \text{if job } i \text{ selected} \\ 0; & \text{otherwise} \end{array} \right\} \tag{234}$$

$$x_{ijk} = \left\{ \begin{array}{ll} 1; & \text{if operation } k \text{ of job } i \text{ is assigned on machine } j \\ 0; & \text{otherwise} \end{array} \right\} \tag{235}$$

$$O_j = \text{overload on machine } j \tag{236}$$

$$U_j = \text{underload on machine } j \tag{237}$$

where

$$i = 1, \ldots, m \tag{238}$$

$$k = 1, \ldots, y_i \tag{239}$$

$$j = 1, \ldots, n \tag{240}$$

Problem statement

Objective

minimize

$$O_1 + O_2 + O_3 + O_4 + U_1 + U_2 + U_3 + U_4 - 500x_1 - 0.0007x_2 \tag{241}$$
$$-0.0013x_3 - 500x_4 - 0.0015x_5 - 0.0014x_6 - 0.0006x_7 - 0.0005x_8$$

subject to the following:

Tool slot constraints

$$x_{211}+2x_{311}+x_{631}+x_{721}+x_{811}+3x_{831}+x_{821} \quad \leq \quad 5 \tag{242}$$

$$x_{113}+3x_{323}+x_{413}+2x_{513}+x_{623}+x_{713}+x_{723}+x_{813} \quad \leq \quad 5 \tag{243}$$

$$x_{224}+x_{214}+2x_{314}+x_{424}+x_{614}+x_{624}+3x_{734}+x_{714} \quad \leq \quad 5 \tag{244}$$

$$x_{232}+2x_{512}+x_{522}+x_{632}+x_{622}+x_{722}+x_{712}+x_{822}+x_{812}-x_{25} \quad \leq \quad 5 \tag{245}$$

$$x_{232}+x_{522}-x_{25} \quad \leq \quad 1 \tag{246}$$

$$-x_{232}-x_{522}+2x_{25} \quad \leq \quad 0 \tag{247}$$

Unique job routing constraints

$$x_{211}+x_{214} \quad \leq \quad 1 \tag{248}$$

$$x_{314}+x_{311} \quad \leq \quad 1 \tag{249}$$

$$x_{512}+x_{513} \quad \leq \quad 1 \tag{250}$$

$$x_{624}+x_{622}+x_{623} \quad \leq \quad 1 \tag{251}$$

$$x_{632}+x_{631} \quad \leq \quad 1 \tag{252}$$

$$x_{713}+x_{712}+x_{714} \quad \leq \quad 1 \tag{253}$$

$$x_{722}+x_{723}+x_{721} \quad \leq \quad 1 \tag{254}$$

$$x_{811}+x_{812}+x_{813} \quad \leq \quad 1 \tag{255}$$

$$x_{822}+x_{821} \quad \leq \quad 1 \tag{256}$$

Non-splitting of jobs constraints

$$x_{113}-x_{1} \quad = \quad 0 \tag{257}$$

$$x_{211}+x_{214}+x_{224}+x_{232}-3x_{2} \quad = \quad 0 \tag{258}$$

$$x_{314}+x_{311}+x_{323}-2x_{3} \quad = \quad 0 \tag{259}$$

$$x_{413}+x_{424}-2x_{4} \quad = \quad 0 \tag{260}$$

$$x_{512} + x_{513} + x_{522} - 2x_5 \quad = \quad 0 \tag{261}$$

$$x_{614} + x_{624} + x_{622} + x_{623} + x_{632} + x_{631} - 3x_6 \quad = \quad 0 \tag{262}$$

$$x_{713} + x_{712} + x_{714} + x_{722} + x_{723} + x_{721} + x_{734} - 3x_7 \quad = \quad 0 \tag{263}$$

$$x_{811} + x_{812} + x_{813} + x_{822} + x_{821} + x_{831} - 3x_8 \quad = \quad 0 \tag{264}$$

Machine capacity constraints

$$225x_{211} + 338x_{311} + 210x_{631} + 156x_{721} + 325x_{811} \tag{265}$$
$$+312x_{831} + 091x_{821} + U_1 - O_1 \quad = \quad 480$$

$$198x_{232} + 198x_{512} + 225x_{522} + 210x_{632} + 070x_{622} + 156x_{722} \tag{266}$$
$$+228x_{712} + 091x_{822} + 325x_{812} + U_2 - O_2 \quad = \quad 480$$

$$144x_{113} + 143x_{323} + 084x_{413} + 198x_{513} + 070x_{623} + 228x_{713} \tag{267}$$
$$+156x_{723} + 325x_{813} + U_3 - O_3 \quad = \quad 480$$

$$216x_{224} + 225x_{214} + 338x_{314} + 114x_{424} + 160x_{614} \tag{268}$$
$$+070x_{624} + 276x_{734} + 228x_{714} + U_4 - O_4 \quad = \quad 480$$

Integer requirements

All x_i and all $x_{ijk} \in \{0, 1\}$.

Quantity	At continuous solution	After M1	After M2	After M3	After M4	After M5
# rows, (incl obj) *m*	29					
# columns (incl slacks), *n*	80					
# integer variables, *ni*	41					
# integer-infeasible basics	10	10	10	5	10	0
# integer-feasible basics	6	4	4	4	4	0
# integer-infeasible superbasics	0	0	0	0	0	9
# integer-feasible superbasics	0	2	2	7	2	7
# superbasics	0	2	2	7	2	16
Objective	-1000	-1000	-1000	-1000	-1000	-1000
# nonbasics at lower bound, *nl*	45	43	43	38	43	30
# nonbasics at upper bound, *nu*	6	6	6	6	6	5
# integer-infeasibilities[1]	10	10	10	5	10	9

Table 15 Shanker & Tzen model minor parameters

1 This value is the sum of the number of basic integer-infeasibilities and the number of superbasic integer-infeasibilities.

Run mode	Objective	Total unbalance	Jobs selected	# Branch-and-bound nodes	Run time (sec)[1]	# Degener-acies	MINOS iterations[2]
Continuous	-1000	N/A	N/A	N/A	4	7	28
M0[3]	-878	122	1,2,4,5,6	361	306	13	5231
M1	-433	567	1,4,5,8	14	8	10	79
M2	-348	652	1,2,4,5,6,8	45	23	14	321
M3	473	973	1,3,7	246	131	11	2003
M4	-433	567	1,4,5,8	14	8	10	79
M5	-493	507	1,2,4,8	15	15	11	166

Table 16 Shanker & Tzen model results summary

1 Run on a 20MHz 80386/80387 system with 32-bit code generation switched ON.

2 This is measured as the number of line-searches required.

3 Method 0 is branch-and-bound.

Discussion of results

1. As stated by Mawengkang [48], the continuous solution to this problem contains 30 feasible binary integers out of 41. His process for achieving integer-feasibility was successful and the result of *total system unbalance = 370* is superior to that obtained by any of the present Mn, which is somewhat incongruous, since his direct search is essentially the present M1.

2. It can be seen from table 16 that, while suboptimal, the present method M5 achieves a respectable result in terms of *total system unbalance*. The unbalance (507) is not as good as that obtained by branch-and-bound (122), but superior to the Shanker and Tzen loading policy #5, which leads to a *total system unbalance = 761*. Note also that the run time on a 20Mhz 80386 PC with floating-point coprocessor for the present M5 is only 15 seconds, whereas branch-and-bound (M0) takes 306 seconds.

Chapter 14

Computational experience VII—A large-scale nonlinear integer programming model for joint optimization of communications network topology and capacitated traffic allocation

Introduction

This chapter is concerned with a NLIP model for the optimization of communications network topology and traffic allocation which has been proposed by Berry and Sugden [2].

Classification of network optimization problems

In Chapter 5.5 of his work *Queueing Systems, Volume 2: Computer Applications* [41], Kleinrock presents a taxonomy of network design optimization problems in which four broad classes are defined. In each case, the objective to be minimized is the *average message delay*, which will not be defined here. The reader seeking further details is referred to Kleinrock's book [*op cit*]. The four problem classes are as follows:

(i) The *capacity assignment* (CA) problem, in which the network flows and topology are given, and one seeks to determine the *channel capacities*.

(ii) The *flow assignment* (FA) problem, in which the network capacities and topology are given, and one seeks to determine the *channel flows*.

(iii) The *capacity and flow assignment* (CFA) problem, in which the only the network topology is given, and one seeks to determine the *channel capacities and flows*.

(iv) The *topology, capacity and flow assignment* (TCFA) problem, in which nothing is given, and one seeks to determine the *topology, channel capacities and flows*.

In an obvious extension of Kleinrock's taxonomy, we may classify the problem discussed in this chapter as a *topology and flow assignment* (TFA) problem, in which link capacities are specified and we seek to minimize total network cost with respect to topology and link flows.

Literature

The following paragraphs contain a very brief survey of literature in the general field of network optimization, but specialized to those papers concerned with simultaneous consideration of both topology and allocation (routing).

In a recent paper [23], Gersht and Weihmayer describe an algorithm to simultaneously generate communications network topologies and allocate line capacities while optimizing the total network cost. Most prior network optimization work has been concerned with the separate optimization of topology and capacity allocation.

In a very recent research report, Gavish [22] has summarized some of the most promising approaches to the problem of topological design and capacity expansion of telecommunication networks. Such problems are characterized by the intrinsic complexity of network topology design and enormous dimensionality of the corresponding mathematical programming models. In general these models are extremely large NLIPs—even for moderately-sized networks, eg 20 nodes or less.

In his survey [*op cit*], Gavish has considered the design of efficient exact and heuristic procedures for solving the topological design and capacity allocation/expansion planning problem for large-scale telecommunication networks. Apart from the usual creativity involved in taking advantage of special problem structure, the approaches have been based on Lagrangian and surrogate constraint decompositions, and various global search strategies.

To give some idea of the general state-of-the-art, Gavish notes that:

> *.... Despite the practical importance of the problem, only a handful of investigations have been reported so far.....*
>
> *.... A mathematical formulation of moderate size networks consists of tens of millions of variables and hundreds of thousands of constraints.*
>
> Gavish [22]

Even if there were no integer restrictions and all constraints and objective were linear, this would still be a formidable problem even with modern large-scale LP technology—see for example Murtagh [57]. When it is remembered that the class of network optimization problems being considered here almost invariably involves large numbers of ***integer*** variables with the accompanying insurmountable combinatorial barriers, then it may be at least dimly appreciated how fundamentally intractable such problems are.

Gavish gives a general statement of the quantities involved in the expansion of network topology and capacity, based on a multi-period dynamic expansion approach as follows:

Given

- *number of possible switches and their locations*
- *traffic requirement matrix for each period*
- *cost structures as a function of time*

Minimize

net present worth of total cost

with respect to

- *when and where to install conduits (network topology expansion)*
- *when and where to expand line capacities (network capacity expansion)*
- *how to route network traffic (routing decisions)*

subject to

- *reliability constraints*
- *grade of service constraints or loss or delay constraints*

- *flow conservation constraints*
- *capacity constraints*
- *other types of side constraints that are application-dependent*

Gavish [22]

The present chapter considers application of the direct-search techniques developed in this thesis to a very small example of a network optimization problem. Our formulation is a very simple one which is static with respect to time, however it serves to illustrate the benefits of the direct-search techniques developed herein. The model is proposed by Berry and Sugden [2] for network synthesis and capacitated traffic allocation and is described below. It is a linearly-constrained quadratic mixed-integer program. Before stating the problem, we recall some elementary definitions from graph theory in the section to follow.

For further background on the graph-theoretical concepts involved, written from a computational viewpoint, the reader may consult the excellent books by Nemhauser and Wolsey [68], Papadimitriou and Steiglitz [69], or Reingold, Nievergelt and Deo [78].

For the reader interested in extra details from the standpoint of application to large-scale telecommunication networks, a wealth of references is to be found in the Gavish survey paper cited [*op cit*], from which we select the following (pertaining to network topology and capacity expansion): Zadeh [97], Christofides and Brooker [10], Doulliez and Rao [16], Minoux [53], and Parrish, Cox, Kühner and Qiu [71].

14.1 Complete graph

Given a positive integer n_0, representing the number of vertices (also called *nodes*) in the network to be considered, we first construct (conceptually) the *complete graph* K_{n_0}. In this simple undirected graph, each vertex is connected to all other vertices by a single edge. We label the vertices of K_{n_0} simply 1, 2, ..., n_0. The vertex set is thus $V = \{1, 2, \ldots, n_0\}$. Since the graph is undirected, the edges of K_{n_0} correspond to the 2–element subsets of V (rather than ordered pairs), so that the edge set of K_{n_0} is given by

$$E = \{ \{p,q\} : p,q \in V\} \tag{269}$$

The cardinality of the edge-set of K_{n_0} is therefore given by

$$|E| = N = \binom{n_0}{2} = \frac{n_0(n_0-1)}{2} \tag{270}$$

Example

We consider K_7. Here we have $V = \{1,2,3,4,5,6,7\}$, $N = 21$, and

$$\begin{aligned} E = \{ & \{1,2\}, \{1,3\}, \{1,4\}, \{1,5\}, \{1,6\}, \{1,7\}, \\ & \{2,3\}, \{2,4\}, \{2,5\}, \{2,6\}, \{2,7\}, \\ & \{3,4\}, \{3,5\}, \{3,6\}, \{3,7\}, \\ & \{4,5\}, \{4,6\}, \{4,7\}, \\ & \{5,6\}, \{5,7\}, \\ & \{6,7\} \} \end{aligned} \tag{271}$$

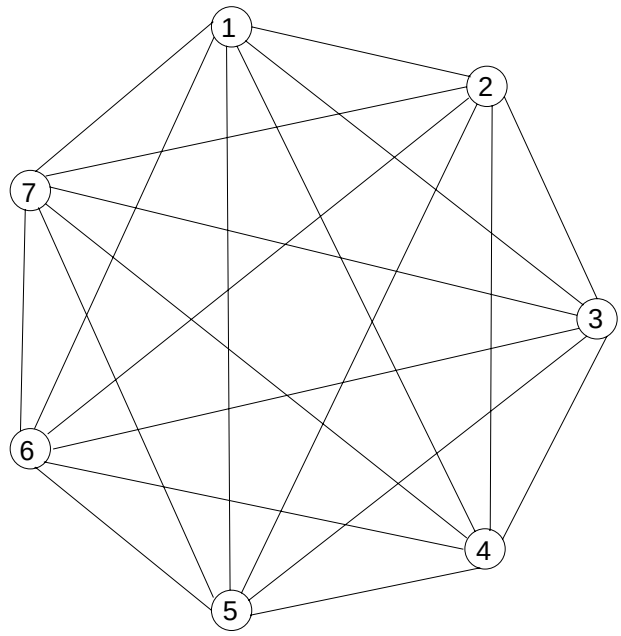

Figure 31 A representation of the complete graph K_7**.**

Paths in K_{n_0}

A *path of length l* from vertex p to vertex q is defined to be an *edge sequence* of the form $(\{v_0, v_1\}, \quad \{v_1, v_2\}, \quad \ldots, \quad \{v_{l-1}, v_l\})$, where $v_0 = p$ and $v_l = q$. Note that elements of the sequence are doubleton sets, each representing an edge in the graph.

Since K_{n_0} is *simple*, ie no multiple edges or self-loops, we can equally well define a path as a *vertex sequence.* Suppose once again that p and q are vertices. Then a *path of length l* from vertex p to vertex q is defined to be the *vertex sequence* $(v_0, v_1, v_2, \ldots, v_{l-1}, v_l)$, where $v_0 = p$ and $v_l = q$, the other v_i being vertices of the graph, with none repeated. Of course, paths may also be defined inductively, ie *recursively*. For such a definition, the reader may consult almost any elementary text on graph theory or discrete mathematics; for definiteness, we cite Ross and Wright [79]. The vertex $v_0 = p$ is referred to as the *origin vertex* and $v_l = q$ as the *destination vertex*; thus $(v_0, v_l) \quad = \quad (p, q)$ is an *origin-destination pair*, or OD−pair. It may be argued that an origin-destination pair (OD−pair) should be an *ordered pair* (k_1, k_2) of nodes specifying an edge in a digraph, however we take the view that, in spite of the "direction" implied by the term origin-destination, we need only consider undirected paths.

Enumerating the OD–pairs

For K_{n_0}, the number of OD–pairs is given by the expression

$$\binom{n_0}{2} = \frac{n_0(n_0-1)}{2} \tag{272}$$

Consider now the ith OD–pair in some enumeration (eg lexicographic order of vertex suffixes). We have $c_{i1}, c_{i2}, c_{i3}, c_{i4},$ being the costs of the four cheapest paths from p to q. If l is the length of such a path, then $1 \le l \le n_0 - 1$ (a vertex may not be revisited). Note: links (edges) may also be numbered using the same scheme since there is a 1–1 correspondence between OD–pairs and edges.

Consider also the set P of all paths of maximal length ($l = n_0 - 1$) from p to q. How many such paths are there? Since a vertex may not be revisited, there is a 1–1 correspondence between such paths and the permutations of the elements of the set $V \setminus \{p, q\}$. Thus, the number of paths of length $l = n_0 - 1$ is given by the expression $(n_0 - 2)!$, and all such paths may be enumerated by any algorithm which lists the permutations of the elements of a given set.

Consider now paths of length $n_0 - 2$. These may be enumerated by considering each subset of cardinality $n_0 - 3$ of $V \setminus \{p, q\}$, and for each one, computing all the permutations of vertices.

Proceeding in this manner, we find that for K_{n_0}, the number of paths of length l between any pair of vertices is given by the expression

$$\binom{n_0-2}{l-1}(l-1)! = (n_0-2)\,(n_0-3)\ldots(n_0-l-1) \tag{273}$$

This expression represents the number of permutations of $l - 1$ objects chosen from a set of $n_0 - 2$ objects.

14.2 Definition of terms

We define the variables and constant parameters involved in the Berry–Sugden NLIP network model. For further semantic details on the terms defined, and notation used (which is essentially standard), the reader is referred to the 1972 monograph of Potts and Oliver [74].

i = OD-pair number, $1 \le i \le N$

λ = link (edge) number, $1 \le \lambda \le N$

j = route (chain) number, $1 \le j \le 4$

c^0_{pq} = cost for link from vertex p to vertex q per unit traffic

t^0_{pq} = traffic between vertex p and vertex q

c_{ij} = cost for OD-pair i on route j per unit traffic

t_i = total traffic carried for OD-pair i

h_{ij} = chain flow, ie traffic carried for OD-pair i on route j

α_{ij} = proportion of total traffic on jth route for OD-pair i

f_λ = actual total traffic (flow) on link λ

$\aleph_\lambda$ = maximum total traffic (flow) on link λ

$$x_{ij} = \left\{ \begin{array}{ll} 1; & \text{if } j\text{th route is chosen for OD-pair } i \\ 0; & \text{otherwise} \end{array} \right\}$$

$$a^i_{\lambda j} = \left\{ \begin{array}{ll} 1; & \text{if } j\text{th route for OD-pair } i \text{ uses link } \lambda \\ 0; & \text{otherwise} \end{array} \right\}$$

The last-defined quantity $a^i_{\lambda j}$, is known as the *link-chain incidence matrix*.

For the present model we look at the four cheapest routes and choose precisely two routes for each OD–pair. This seemingly arbitrary requirement does in fact have a rational basis. The *robustness* of a network may be defined as the ability to carry traffic on more than one route for a given OD–pair. One advantage of this property is that a measure of *redundancy*

is obtained, so that in the event of failure of any link (edge) in a route (thus rendering that route unavailable), at least one other is available to carry at least some of the traffic. Since we are modelling a fixed capacity constraint for each link (edge), it is certainly possible that the entire volume of planned traffic for a given OD–pair cannot be supported if one of the allocated routes goes down because of a link failure. Teletraffic researchers also speak of network *link diversity* or *node diversity*, which are concepts related to our robustness.

MP formulation for the Berry–Sugden model

minimize

$$\sum_{i=1}^{N} t_i \sum_{j=1}^{4} c_{ij}\alpha_{ij}x_{ij} \tag{274}$$

subject to

$$(R) \qquad \sum_{j=1}^{4} x_{ij} = 2, \qquad i = 1, \ldots, N \tag{275}$$

$$(S) \qquad \sum_{j=1}^{4} \alpha_{ij} = 1, \qquad i = 1, \ldots, N \tag{276}$$

$$(T) \qquad \sum_{i=1}^{N} t_i \sum_{j=1}^{4} a^{i}_{\lambda j}\alpha_{ij} \leq \aleph_{\lambda}, \qquad \lambda = 1, \ldots, N \tag{277}$$

$$(U) \qquad x_{ij} - \alpha_{ij} \geq 0, \qquad i = 1, \ldots, N; \quad j = 1, \ldots, 4 \tag{278}$$

$$(V) \qquad \alpha_{ij} \geq 0, \qquad i = 1, \ldots, N; \quad j = 1, \ldots, 4 \tag{279}$$

$$(W) \qquad x_{ij} \in \{0, 1\}, \qquad i = 1, \ldots, N; \quad j = 1, \ldots, 4 \tag{280}$$

Notes

1. The objective function is the total network cost, requiring choice of exactly two routes for each OD–pair.

2. The R category constraints specify that precisely two routes must be allocated for each OD–pair. This is to ensure *robustness*, as defined above.

3. The S category constraints specify that the sum of fractional allocations for a given OD–pair is unity. This follows directly from the definition of α_{ij}.

4. From the foregoing definitions and constraints, we see that the traffic h_{ij} (also termed *chain flow*) on route j for OD–pair i is given by the expression $h_{ij} = \alpha_{ij} t_i$. Summing traffic over all four routes for a given OD–pair i yields the relationship:

$$\sum_{j=1}^{4} h_{ij} \quad = \quad \sum_{j=1}^{4} \alpha_{ij} t_i \quad = \quad t_i \sum_{j=1}^{4} \alpha_{ij} \quad = \quad t_i$$

The T constraints place upper bounds on the total traffic carried by each link (edge) of the network.

5. We wish to have $\alpha_{ij} > 0$ imply $x_{ij} = 1$. It is a logical contradiction to allocate nonzero traffic ($\alpha_{ij} > 0$) to an unselected route ($x_{ij} = 0$). Also, there is no point in selecting a route, but then allocating zero traffic, however this is a harmless situation. The U category of constraints will force the former condition at the cost of introducing a considerable number of (linear) constraints.

6. After the NLIP model is solved, the outcomes of interest are:

   (i) Which links (edges) are used from K_{n_0}?

   (ii) For a given OD–pair, what is the chain-flow pattern? ie how is the flow distributed among the four possible routes?

   (iii) For each link, what is the total flow? Actually, when this information is available, the answer to (i) is clear, since only those links from the complete graph which have been used will have nonzero flows. For link λ, the total flow f_λ is given by the expression

$$f_\lambda \quad = \quad \sum_{i=1}^{N} t_i \sum_{j=1}^{4} \alpha_{ij} a_{\lambda j}^{i} \tag{281}$$

Size of QIP

It may be noted that even a moderate number of nodes n_0 in the network gives rise to quite a large QIP. In general, if we count the numbers of rows and columns for the corresponding QIP, we obtain:

(a) $4N$ continuous variables

(b) $4N$ integer variables

(d) N category R constraints

(e) N category S constraints

(f) N category T constraints

(c) $4N$ category U constraints

Therefore, *in toto* we have $7N = 7n_0(n_0 - 1)/2$ rows and $8N = 4n_0(n_0 - 1)$ columns. When slacks and right-hand-side are included (as they will be in MINTO) we have $n = 15N + 1 = 15n_0(n_0 - 1)/2 + 1$ columns.

Example

We consider a problem in which $n_0 = 7$, thus leading to 168 structural variables and 147 constraints. This leads to an effective $n = 316$ (168 structural + 147 slack + 1 rhs) and of course $m = 147$. This is indeed a large QIP for such a small network ($n_0 = 7$).

Traffic and cost matrices

The traffic and cost matrices, T^0 and C^0 respectively, for this example are as follows:

$$T^0 = [t_{ij}^0] = \begin{bmatrix} 0 & 7 & 9 & 5 & 3 & 4 & 2 \\ 7 & 0 & 6 & 5 & 2 & 1 & 3 \\ 9 & 6 & 0 & 8 & 9 & 2 & 3 \\ 5 & 5 & 8 & 0 & 2 & 7 & 6 \\ 3 & 2 & 9 & 2 & 0 & 5 & 2 \\ 4 & 1 & 2 & 7 & 5 & 0 & 4 \\ 2 & 3 & 3 & 6 & 2 & 4 & 0 \end{bmatrix} \tag{282}$$

$$C^0 \quad = \quad [c_{ij}^0] \quad = \quad \begin{bmatrix} 0 & 5 & 12 & 8 & 6 & 4 & 1 \\ 5 & 0 & 6 & 8 & 15 & 6 & 4 \\ 12 & 6 & 0 & 5 & 4 & 7 & 7 \\ 8 & 8 & 5 & 0 & 2 & 7 & 6 \\ 6 & 15 & 4 & 2 & 0 & 5 & 2 \\ 4 & 6 & 7 & 7 & 5 & 0 & 4 \\ 1 & 4 & 7 & 6 & 2 & 4 & 0 \end{bmatrix} \tag{283}$$

A special-purpose program was written to generate the four best paths for each of the 21 OD−pairs, and then to generate the required MPS file and CALCFG FORTRAN subroutine. The generated **t** vector and C matrix follow.

$$\mathbf{t} \quad = \quad [t_i] \quad = \quad (7, 9, 5, 3, 4, 2, 6, 5, 2, 1, 3, 8, 9, 2, 3, 2, 7, 6, 5, 2, 4)^T \tag{284}$$

$$C \quad = \quad [c_{ij}] \quad = \quad \begin{bmatrix} 5 & 5 & 12 & 10 \\ 8 & 11 & 11 & 10 \\ 8 & 7 & 11 & 8 \\ 6 & 9 & 3 & 10 \\ 4 & 11 & 11 & 5 \\ 1 & 9 & 8 & 8 \\ 6 & 11 & 13 & 13 \\ 8 & 10 & 11 & 12 \\ 6 & 8 & 10 & 10 \\ 6 & 9 & 10 & 8 \\ 4 & 6 & 12 & 10 \\ 5 & 13 & 12 & 6 \\ 4 & 9 & 12 & 7 \\ 7 & 9 & 11 & 10 \\ 7 & 6 & 10 & 9 \\ 2 & 8 & 11 & 9 \\ 7 & 10 & 7 & 8 \\ 6 & 9 & 4 & 11 \\ 5 & 10 & 9 & 6 \\ 2 & 7 & 8 & 9 \\ 4 & 5 & 10 & 7 \end{bmatrix} \tag{285}$$

The traffic capacities for each link are given by the vector $\aleph$:

$$\aleph = [\aleph_i] = (15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15)^T$$

The elements t_i of the traffic vector **t** are obtained simply by listing the elements t_{ij}^0 in lexicographic order of vertex pairs ij, ie

$$t_{12}^0, \quad t_{13}^0, \quad t_{14}^0, \quad t_{15}^0, \quad t_{16}^0, \quad t_{17}^0, \quad t_{23}^0, \quad t_{24}^0, \quad t_{25}^0, \quad t_{26}^0, \quad t_{27}^0,$$

$$t_{34}^0, \quad t_{35}^0, \quad t_{36}^0, \quad t_{37}^0, \quad t_{45}^0, \quad t_{46}^0, \quad t_{47}^0, \quad t_{56}^0, \quad t_{57}^0, \quad t_{67}^0$$

The numbers are just the elements of the upper triangle of T^0 (excluding main diagonal), and enumerated row-wise.

As stated earlier, the elements c_{ij} were computed by a special program written for the task; this job being to find among all possible paths in K_{n_0} from each OD–pair $\{p, q\}$, the four cheapest ones. Thus, C^0 is a matrix of dimension $n_0(n_0 - 1)/2 \quad \times \quad 4$.

14.3 Results and comments for the Berry–Sugden model

1. Several hurdles had to be overcome while solving the Berry–Sugden model using the direct search methods. The size of the Berry–Sugden executable module (640kb) and the usual DOS 640kb memory limitations required a bare-bones DOS 5.0 configuration with no device drivers or other resident programs. This was most inconvenient for development work. Also, some patching of routines to update factorizations of basis and Hessian was needed in order to allocate enough memory for the generation of many new superbasic variables.

2. A relatively large number of integer-feasible basics at the continuous solution required a procedure to automatically pivot these variables from the basis to the superbasic set.

3. For post-processing, a special-purpose program READCOLS was written to read the MPS output from MINTO and then construct network quantities of interest. In particular the link-flow for each of the 21 links was computed by simple summation, and links having zero flow were then easily noted. Such links are then unused from

the complete graph K_7. The READCOLS program also flagged traffic allocation warnings if a route is selected but no traffic allocated, ie $x_{ij} \neq 0$ but $\alpha_{ij} = 0$. As noted above, this situation is not really a problem, so the program prints a simple warning message. The alternative condition for which constraints were formulated to explicitly exclude is $\alpha_{ij} > 0$ but $x_{ij} = 0$. A feasible solution was obtained so one expects that no such condition will occur, however it is very easy to check in the READCOLS program, and therefore serves as simple confirmation that the requirement has been modelled correctly.

4. The READCOLS program was extended to also implement a *greedy allocation* heuristic, so that some comparison could be made with the direct search results. The greedy strategy simply processes each OD–pair in lexicographic sequence and tries to allocate as much traffic as possible to the cheapest route, then if necessary to the remaining three routes in strict order of increasing cost. It should be noted that this strategy need not lead to a *feasible solution*, let alone an optimal solution. However, for the present example, the greedy allocation method just described arrived rapidly at a (necessarily feasible) suboptimal solution with objective 531.0. It was not required to satisfy allocation of exactly two routes for each OD–pair for this approach.

5. In summary, for the example presented, it was found that, for most of the methods Mn used, 4 links from a total of 21 from the complete graph K_7 were not used, ie no route eventually selected to carry at least part of the traffic for the 21 OD–pairs actually used those 4 links. For example, in the case of M3N, the unused links are those joining vertices (1,3), (1,5), (2,5) and (4,6). It was also found that of the 42 routes allocated, 19 had zero traffic.

6. Variation of the upper bound $\aleph_i$ on the link flows produced expected behaviour from the model. As the bounds were decreased, further spread (robustness) in allocation was in evidence.

7. Another interesting test case which could be considered is to force the model to select a minimal spanning tree for the topology + allocation problem by making certain routes prohibitively expensive. In fact, the model would need to be reformulated in order to achieve this, since at present we insist on the selection of exactly two routes for each OD–pair.

8. A 0–1 problem such as that described by the Berry-Sugden model must of necessity be highly-degenerate (many basics at a bound) if there are considerable numbers of integer variables present in the basis at the optimal solution. A 0–1 variable cannot but be at a bound if it is integer-feasible.

9. Some cycling problems were noted. Branch-and-bound had trouble with a subproblem and the iteration limit expired because of cycling. M2 also had cycling problems.

10. The remaining methods M1, M3, M4, M5 produced results close to the (possibly suboptimal) result achieved by the aborted branch-and-bound process. M1, M3, M4 actually produced the same objective as branch-and-bound when integers were *not fixed* after the respective direct search procedure.

11. M5 fared better on an earlier version of the present MINLP model, in that it terminated with only two of the 84 integer variables infeasible and superbasic (from approximately 65 at the continuous solution). Note that M5 *always* terminates with no integer variables basic. The present model has approximately 50 integer-infeasibilities at the termination of M5 (all necessarily superbasic), and this may be directly attributed to the imposition of 21 extra link capacity constraints (the T set of constraints). Intuitively, it may be imagined that M5 had much less "room to move" in its attempt to evict integers from the basis. Consequently, the $B \leftrightarrow S$ pivot operation (which does *not* decrease the total integer infeasibilities) would seem to have been invoked more often than the nonbasic step (which *always* decreases integer infeasibilities by at least one) than was previously the case.

12. Post-processing or even simple inspection of the MPS output solution indicated that the constraints have had the desired effect—the limited link traffic capacities and "encouragement" of diverse allocation have allowed some non-trivial assignments to be made. In this, we refer to assignments which perhaps might otherwise be made by a *greedy* allocation algorithm, which would be expected to allocate all traffic for a given OD–pair to the cheapest route if this were possible (see comment #4 above).

13. Since the numerical results for the Berry–Sugden model are quite voluminous, no output is included here, however post-processing of the MPS output file by the READCOLS program is included along with summary information in the tables overleaf.

Quantity[1]	**At continuous solution**	**After M1**	**After M3**	**After M4**	**After M5**
# rows, (incl obj) *m*	1478				
# columns (incl slacks), *n*	316				
# integer variables, *ni*	84				
# integer-infeasible basics	47	31	35	37	0
# integer-feasible basics	14	1	1	0	0
# integer-infeasible superbasics	16	16	16	12	26
# integer-feasible superbasics	0	29	25	28	51
# superbasics	39	68	63	57	83
Objective	243.06	404.20	380.60	349.37	545.33
# nonbasics at lower bound, *nl*	66	50	46	66	30
# nonbasics at upper bound, *nu*	64	51	60	64	56
# integer-infeasibilities[2]	63	47	51	49	26

Table 17 Berry–Sugden model solution minor parameters

1 M2 is excluded from the table since it did not terminate for this problem.

2 This value is the sum of the number of basic integer-infeasibilities and the number of superbasic integer-infeasibilities.

Run mode[1]	Objective	# Branch-and-bound nodes	Run time (sec)[2]	#Objective/ gradient calls	# Degeneracies	MINOS iterations[3]
Continuous	243.06	N/A	65	266	15	261
M0	510.00	233	1563	13171	69	9999[4]
M1F	511.00	27	189	1110	70	734
M1N	510.00	50	320	2242	70	1332
M3F[5]	510.73	1	194	1406	69	909
M3N	510.00	52	338	2595	73	1542
M4F	527.00	28	252	1194	74	804
M4N	510.00	55	440	2782	69	1631
M5N	518.00	52	396	2490	67	1472

Table 18 Berry–Sugden model results summary

1 M5F omitted because it does not terminate.

2 Run on a 20MHz 80386/80387 system with 32-bit code generation switched ON.

3 This is measured as the number of line-searches required.

4 Iteration limit exceeded

5 The suffix F refers to integer-feasible variables being fixed after direct search; N *not* fixed.

Chapter 15

Conclusions

This work has presented a number of direct search strategies for achieving integer-feasibility for a class of mixed-integer nonlinear programming problems in a relatively short time. It has improved on the previous direct search approaches of Murtagh [59], and Mawengkang and Murtagh [48], also based on the superbasic variables and active constraint method of Murtagh and Saunders [62]. It was found that the method described by Mawengkang sometimes had trouble terminating because of one or two *cycling* phenomena, at least on certain problems. Alternative direct-search methods have been proposed and tested. The results appear to be quite encouraging. The present methods have solved instances of a new network optimization model proposed by Berry and Sugden [2] and done so in very reasonable time on an 80386 PC.

The new direct search methods have been shown to be successful on a range of problems, while not always able to achieve global optimality, generally achieve integer-feasibility (perhaps with some aid from branch-and-bound) in a much shorter time than branch-and-bound alone. In in a significant number of cases the suboptimal point so obtained is acceptable, since the exponential complexity of integer programming in general precludes branch-and-bound except on small to medium problems unless one is very lucky and tight bounds are obtained early in the branching process.

The fifth of the new direct search methods is herein proven to always terminate with no integer variables in the simplex basis. Since such termination is a precondition for further development of the method along the lines of trial fractional, then integer steps in the superbasic integer variables, a foundation has been established for this further work.

McMahon [50] defines greedy algorithms as

... those non-backtracking algorithms in which irrevocable decisions of global significance are made on the basis of local information.

McMahon [50]

The direct search methods of the present work certainly fall into this category, and it is recognized that the use of such methods normally implies suboptimal solutions. Nevertheless, it must be remembered that published methods also fall into this category, and global optimality, or even guaranteed *local* optimality for the general MINLP problem is a goal which is in many practical instances, simply out of reach. A useful avenue of further research would be aimed at obtaining tight upper and lower bounds for the objective function at a nearby locally optimal point for a solution obtained by the proposed new direct search methods.

The new methods have been implemented in conjunction with an interactive module which allows a skilled user to "drive" the NLIP solver engine. Alternatively, if desired, the system may be run entirely in the conventional "batch" mode, in which any of the direct search strategies may be automatically invoked. The use may assume interactive control of the search process at the point where the solution of the continuous relaxation has just been found. From this point a variety of operations designed to give information about progress toward both integer-feasibility and improvement of objective may be selected from a menu. Full error-checking is provided so that the current solution vector remains feasible at all times. For example, a trial step in a superbasic variable (either discrete or continuous) may be attempted, but will not be allowed if it would violate any of the current set of linearized constraints; in fact, in this situation, the first basic variable to be violated is indicated in the on-screen error message to the user. In a significant number of such cases, it becomes clear that no further progress is possible because the current search strategy has led us to an impasse.

If small steps are too tedious, then the user may select to any of the five direct search procedures to be executed at any time and observe the progress toward integer-feasibility as the solution parameters are dynamically updated on the screen. The display follows a pseudo-spreadsheet paradigm in which rows may be selected for subsequent operation by simply moving the cursor. Valuable insight into the internal mechanisms and run-time behaviour of the direct search process has already been obtained by observing the progress on-screen. It is hoped that more experience with a larger class of MINLP problems will lead to further refinement of the search procedures described in this work.

References

1 Balas, E. and Mazzola, J.B. (1984). Nonlinear 0-1 Programming: I. Linearization Techniques II. Dominance Relations and Algorithms. *Mathematical Programming* **30**:1–45.

2 Berry, L.T.M. (1991). *Private communication.*

3 Booch, G. (1987). *Software engineering with Ada*. 2nd edition. Benjamin-Cummings.

4 Brent, R.P. (1973). *Algorithms for minimization without derivatives*, Prentice-Hall.

5 Brooke, A., Kendrick, D. and Meeraus. (1988). *A GAMS A Users' Guide*. Scientific Press, Palo Alto.

6 Burkhard, R.E. and Rendl, F. (1984). A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operations Research* **17**:169–174.

7 Cauchy, A. (1847). Méthode générale pour la résolution des systémes d'équations simultanées. *Comp. Rend. Acad. Sci. Paris* 378–383.

8 Connolly, D.T. (1990). An improved annealing scheme for the QAP. *European Journal of Operations Research* **46**:93–101.

9 Cooper, M.W. (1981). A Survey of Methods for Pure Nonlinear Integer Programming. *Management Science* **27**:353–361.

10 Christofides, N. and Brooker, P. (1974). Optimal expansion of an existing network. *Mathematical Programming* **6**:197–211.

11 Daellenbach, H.G., George, J.A. and McNickle, D.C. (1983). *Introduction to operations research techniques. 2nd edition*. Allyn & Bacon.

12 Dantzig, G.B. (1962). *Linear programming and extensions*. Princeton university press, Princeton, NJ.

13 Dantzig, G.B., Orden, A. and Wolfe, P. (1955). The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics* **5**:183–195.

14 Davis, L. (1987). *Genetic algorithms and simulated annealing*, Los Altos CA: Morgan Kaufmann.

15 Dennis, J. and Torczon, V. (1990). *Direct search methods on parallel machines*. Rice University, Dept of Mathematical Sciences. Technical Report TR90–19, Sept 1990.

16 Doulliez, P.J. and Rao, M.R. (1975). Optimal network capacity planning: a shortest path scheme. *Operations Research* **23**:811–818.

17 Duran, M.A. and Grossmann, I.E. (1986). An Outer-Approximation Algorithm for a Class of Mixed-Integer Nonlinear Programs. *Mathematical Programming* **36**:307–339.

18 Fletcher, R. (1987). *Practical methods of optimization*. Wiley.

19 Fletcher, R. and Reeves, C.M. (1964). Function minimization by conjugate gradients. *Computer Journal*, **7:**149–154.

20 Fiacco, A. and McCormick, G. (1968). *Nonlinear programming: sequential unconstrained minimization techniques*. Wiley.

21 Garfinkel, R.S. and Nemhauser, G.L. (1972). *Integer programming*. Wiley.

22 Gavish, B. (1991). *Topological design and capacity expansion of telecommunication networks—State of the art survey*. Centre for Telecommunication Network Research Report #11/91. Bond University, Australia.

23 Gersht, A. and Weihmayer, R. (1990). Joint optimization of data network design and facility selection. *IEEE Journal on Selected Areas in Communications* **8**:1667–1681.

24 Gill, P.E., Murray, W. and Wright, M.H. (1981). *Practical Optimization*. Academic Press. ISBN 0-12-283952-8.

25 Gill, P.E., Murray, W., Saunders, M.A. and Wright, M.H. (1984). Software and its relationship to methods. *SIAM Numerical Optimization 1984*. ISBN 0-89871-054-5.

26 Gill, P.E., Murray, W., Saunders, M.A. and Wright, M.H. (1988). *GAMS/MINOS User Manual, Appendix D*. Scientific Press.

27 Goldberg, D.E. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

28 Gomory, R. (1965). On the relation between integer and noninteger solutions to linear programs. *Proceedings of the National Academy of Science* **53**:2, 260–265.

29 Gomory, R. (1967). Faces of an integer polyhedron. *Proceedings of the National Academy of Science* **57**:1, 260–265.

30 Gomory, R. (1969). Some polyhedra related to combinatorial problems. *Journal of Linear Algebra and its Applications* **2**:4, 451–558.

31 Gupta, O.K. and Ravindran, A. (1983). Nonlinear integer programming algorithms: A Survey. *OPSEARCH* **20**:189–206.

32 Gupta, O.K. and Ravindran, A. (1985). Branch and bound experiments in convex nonlinear integer programming. *Management Science* **31**:1533–1546.

33 Hammer, P.L. and Rudeanu, S. (1968) *Boolean methods in operations research and related areas*. Springer-Verlag, Heidelberg.

34 Hansen, P. (1979). Methods of 0-1 nonlinear programming. *Annals of Discrete Mathematics* **5**:53–70.

35 Hardy, G.H. (1967). *A mathematician's apology*, 85. Cambridge University Press.

36 Hestenes, M.R. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *J. Res. N.B.S.* **49:**409–436.

37 Hext, J. (1990). *Programming structures. Volume I—Machines and programs*, 46. PHI.

38 Himmelblau, D.H. (1972). *Applied nonlinear programming*. McGraw-Hill.

39 Jeroslow, R.G. (1974). Trivial integer programs unsolvable by branch-and-bound. *Mathematical programming* **6:**105–109.

40 Karmarkar, N. (1984). *A new polynomial-time algorithm for linear programming.* Internal report, Mathematical Sciences Division, AT&T Bell Laboratories, Murray Hill, New Jersey.

41 Kleinrock, L. (1976). *Queueing Systems Volume 2: Computer Applications.* Wiley.

42 Kocis, G.R. and Grossmann, I.E. (1986). A Relaxation Strategy for the Structural Optimization of Process Synthesis. *Annual AIChE Meeting*, Miami, November 1986.

43 Kuhn, H.W. and Tucker, A.W. (1951). Nonlinear programming. *Proceedings of the second Berkeley symposium on mathematical statistics and probability*, 481–492, Berkeley, University of California Press.

44 Leyffer, S. (1991). *Private communication*.

45 Lustig, I., Mulvey, J. and Carpenter, T. (1989). Formulating stochastic programs for interior point methods. Technical Report SOR–89–16, Department of Civil Engineering and Operations Research, Princeton University, New Jersey.

46 Marsten, R., Saltzman, M., Shanno, D., Pierce, G. and Ballantijn, J. (1989). Implementation of a dual affine interior point algorithm for linear programming. *ORSA Journal on Computing* **1**:4, 287–297.

47 Marsten, R., Subramanian, R., Saltzman, M., Lustig, I. and Shanno, D. (1990). Interior point methods for linear programming: Just call Newton, Lagrange, and Fiacco and McCormick. *Interfaces* **20**:4, 105–116.

48 Mawengkang, H. (1988). *Nonlinear Integer Programming*. PhD dissertation, University of New South Wales.

49 Mawengkang, H. and Murtagh, B.A. (1986). Solving Nonlinear Integer Programs with Large-Scale Optimization Software. *Annals of Operations Research* **5**:425–437.

50 McMahon, G.B. (1989). *A structural taxonomy for algorithms*. Working Paper 1989–3–007, July 1989. School of Information and Computing Sciences, Bond University.

51 Michalewicz, Z., Krawczyk, J.B., Kazemi M. and Janikow, C.Z. (1990). Genetic algorithms and optimal control problems. *Proc. 29th IEEE Conference on Decision and Control*, 1664–1666.

52 Minoux, M. (1986). *Mathematical Programming*. Wiley.

53 Minoux, M. (1987). Network synthesis and dynamic network optimization. *Annals of Discrete Mathematics*. **31**:283–324.

54 Mohd, I.B. (1986). *Global optimization using interval arithmetic*. PhD dissertation, University of St Andrews.

55 Moore, R.E. (1966). *Interval analysis*, Prentice-Hall

56 Morrow, M. (1991). Genetic algorithms. *Australian Personal Computer* **12:**4, 85–93.

57 Murtagh, B.A. (1981). *Advanced Linear Programming: Computation and Practice*. McGraw-Hill. ISBN 0-07-044095-6.

58 Murtagh, B.A. (1988). *MINTO User Manual*.

59 Murtagh, B.A. (1989). Nonlinear Integer Programming with Applications in Manufacturing and Process Engineering. *Proceedings of the Computational Techniques and Applications Conference: CTAC–89*, 103–113.

60 Murtagh, B.A. (1989). Nonlinear Integer Programming with Applications in Manufacturing and Process Engineering. *Unpublished overhead transparencies for CTAC–89 invited paper*.

61 Murtagh, B.A. and Saunders, M.A. (1982). A Projected Lagrangian Algorithm and its Implementation for Sparse Nonlinear Constraints. *Mathematical Programming Study* **16**:84–117.

62 Murtagh, B.A. and Saunders, M.A. (1978). Large-Scale Linearly Constrained Optimization. *Mathematical Programming* **14**:41–72.

63 Murtagh, B.A. and Saunders, M.A., *Large-Scale Optimization* (unpublished manuscript).

64 Murtagh, B.A. and Saunders, M.A. (1987). *MINOS 5.1 User's Guide*. Report SOL 83-2OR, Stanford University, December 1983 (revised January 1987).

65 Murtagh, B.A. and Saunders, M.A. (1985). *MINOS 5.2 User's Guide.* Systems Optimization Laboratory, Department of Operations Research, Stanford University.

66 Myers, D.C. (1984). *The design of branch and bound, Lagrangian relaxation and subgradient strategies for mixed integer programming problems*. PhD dissertation. Virginia Polytechnic Institute and State University.

67 Nelder, J.A. and Mead, R. (1965). A simplex method for function minimization. *Computer Journal* **7**:308–313.

68 Nemhauser, G.L. and Wolsey, L.A. (1988). *Integer and Combinatorial Optimization*. Wiley. ISBN 0-471-82819-X.

69 Papadimitriou, C.H. and Steiglitz, K. (1982). *Combinatorial optimization. Algorithms and complexity*. PHI.

70 Papoulias, S.A. and Grossmann, I.E. (1983). A structural optimization approach in process synthesis—heat recovery networks. *Computers and Chemical Engineering*. **7**:707–721.

71 Parrish, S.H., Cox, T., Kühner, W. and Qiu, Y. (1990). *Planning for optimal expansion of leased communication networks*. Technical Report USwest Boulder, Colorado, (*Annals of Operations Research*, forthcoming).

72 Paules, G.E. and Floudas, C.A. (1989). APROS: Algorithmic development methodology for discrete-continuous optimization problems. *Operations Research* **37**:902–915.

73 Peressini, A., Sullivan, F. and Uhl, J. (1988). *The mathematics of nonlinear programming*. Springer-Verlag.

74 Potts, R.B. and Oliver, R.M. (1972). *Flows in transportation networks*. Academic Press, Mathematics in Science and Engineering Series, Volume **90**.

75 Press, W.H., Flannery, B.P., Teukolsky, S.A. and Vetterling, W.T. (1988). *Numerical Recipes*. Cambridge University Press.

76 Ratschek, H. and Rokne, J. (1988). *New computer methods for global optimization*. Halstead Press.

77 Ravindran, A., Phillips, D.T. and Solberg, J.J. (1987). *Operations Research, Principles and Practice*. 2nd edition. Wiley.

78 Reingold, E.M., Nievergelt, J. and Deo, N. (1984?). *Combinatorial Algorithms—Theory and Practice*. PHI.

79 Ross, K.A. and Wright, C.R.B. (1992). *Discrete Mathematics, 3rd edition, 193*. PHI.

80 Ryan, D.M. (1990). *Mathematics meets the real world—a guided tour of crew scheduling*. Abstract of paper presented at 26th Applied Mathematics Conference of the Australian Mathematical Society, Greenmount, February 1990.

81 Salkin, H.M. and Mathur, K. (1989). *Foundations of integer programming*, North-Holland.

82 Scarf, H.E. (1990). Mathematical programming and economic theory. *Operations Research*, **38:**377–385.

83 Scarf, H.E. (1986). Neighbourhood Systems for Production Sets with Indivisibilities. *Econometrica* **54**:507–532.

84 Scarf H.E. (1986). Testing for optimality in the absence of convexity. *Social Choice and Public Decision Making*. Chapter 6. Cambridge University Press 1986. Edited by Heller, W.P., Starr, R.P. and Starrett, D.A.

85 Schrage, L.E. (1981). *User Manual for LINDO*. The Scientific Press, Palo Alto.

86 Shanker, K. and Tzen, Y.J. (1985). A Loading and Dispatching Problem in a Random Flexible Manufacturing System. *International Journal of Production Research* **23**:579–595.

87 Shin, D., Güerdal Z. and Griffin, O. (1990). A penalty approach for nonlinear optimization with discrete design variables. *Engineering optimization* **16**:29–42.

88 Steinberg, L. (1961). The backboard wiring problem: A placement algorithm. *SIAM Review* **3**:37.

89 Strang, G. (1986). *Introduction to applied mathematics*. Wellesley-Cambridge Press.

90 Torczon, V. (1990). *Multi-directional search: a direct search algorithm for parallel machines*. PhD dissertation. Rice University, Dept of Mathematical Sciences. Technical Report TR90–7, May 1990.

91 Van Laarhoven, P.J.M. and Aarts, E.H.L. (1987). *Simulated Annealing: Theory and Applications*. Reidel.

92 Viswanathan, J. and Grossmann, I.E. (1990). A combined penalty function and outer-approximation method for MINLP optimization. *Computers and Chemical Engineering* **14**:769–782.

93 Wasil, E., Golden, B. and Liu, L. (1989). State-of-the-art in nonlinear optimization software for the microcomputer. *Computers and operations research* **16**:497–512.

94 Wayner, P. (1991). Genetic algorithms. *BYTE* **16**:1, 361-368.

95 Wilhelm, M.R. and Ward, T.L. (1987). Solving quadratic assignment problems by simulated annealing. *IIE Transactions* **19**:1, 107–119.

96 Wolfe, P. (1967). Methods of nonlinear programming. *Nonlinear programming*. North-Holland. Ed J. Abadie, 97–131.

97 Zadeh, N. (1974). On building minimum cost communication networks over time. *Networks* **4**:19–34.

Index

A

Aarts, E.H.L., 16
active set methods, 41
Ada, 122
airline crew scheduling, 42
algebraic form, 120
amœba method, 5
artificial variable, 100
ASSERT, 116

B

backboard wiring problem, 2
Balas, E., 46
basic feasible solution, 17, 18
basic variable, 61
Bellman, R., 45
Benders decomposition, 44
Berry, L.T.M., iii, 101, 160, 163, 168, 172
bisection method, 37
Booch, G., 122
branch-and-bound, v, vi, 20, 60, 63, 100
  depth-first, 22
branching, 22
Brent, R.P., 37
Brent's method, 37
Brooker, P., 163
Broyden, C.G., 40
Burkhard, R.E., 16

C

C (programming language), 122
C1P1STEP, 116
C1P2STEP, 116
C2P1, 116
CALCINF, 116
CALCJNS, 116
CALCJS, 116
CALCJSS, 117
CALCTH, 117
CALCZT, 117
Carpenter, T., 19
Cauchy, A., 39
chain flow, 169
Cholesky, 26
Christofides, N., 163
CHUZQI, 117
compilation
  independent, 122
  separate, 122
complete graph, 163
complexity, 42
concavity, 6
conjugate-gradient method, 39
Connolly, D.T., 16
constrained stationary point, 27
constraint
  active, 28, 29, 35, 41
  equality, 1
  inactive, 28
  inequality, 1
  integer, 2
  nonlinear, 2
  satisfaction, 28
  simple-bound, 8, 9
  violation, 28
constraint qualification, 33, 35
continuous relaxation, v
convexity, 6
  local, 23
Cooper, M.W., 46
Cox, T., 163
CTAC, v, 62, 63
curvature, 40
  positive, 31

D

Daellenbach, H.G., 134
Dantzig, G.B., 17, 18, 49
Davidon, W.C., 40
Davis, L., 17
degeneracy, 100
Deo, N., 163
descent direction, 29, 39
direct search, v, 3
DIRECT SEARCH HEURISTIC, 99
DIRSCH, 117
disp.c, 101
Doulliez, P.J., 163
DUMPCOL4, 117
DUMPCOL8, 117
DUMPICOL, 117
DUMPIOPB, 117
DUMPPTN, 117
DUMPXNAT, 117
Duran, M.A., 43
Duszczyk, B., ii

E

eigenvalue, 26
embedded system, 122
ERRMSG, 118
ERRS, 118
exponential complexity, 2, 19 - 21

F

feasibility
  preservation of, 38
feasible arc, 33
feasible direction, 32
feasible set, 5, 17
Fiacco, A.V., 7, 19
Fibonacci search, 37
FIX INTEGERS, 99
FIXINTS, 118
Flannery, B.P., 16, 37
Fletcher, R., 6, 40, 44
flexible manufacturing systems, 2
FORTRAN, 48, 99, 121, 122
  Lahey, 122

G

Garfinkel, R.S., 45
Gauss, K.F., 26
Gaussian elimination, 26
Gavish, B., 161, 163
gene pool, 16
genetic algorithms, 16
GENQIP, 121
George, J.A., 134
Gersht, A., 161
Gill, P.E., 5, 6, 19, 26, 37, 38, 40, 48, 49
Goldberg, D.E., 17
Golden, B., 47
golden section search, 37
Goldfarb, D., 40
Gomory, R., 24
gradient vector, 4, 39
Griffin, O., 7
Grossmann, I.E., 43, 138
Güerdal, Z., 7
Gupta, O.K., 46

H

help screens, 109
Hammer, P.L., 24
Hansen, P., 24, 46
Hardy, G.H., ii
Hessian matrix, 4, 25, 26, 31, 38, 40, 172
  of Lagrangian, 33

I

ICHKNAT, 118
ICHKPTN, 118
IFBTOS, 118
INITIOPB, 118
integer program, 1
  mixed, 1
  nonlinear, 1
  pure, 1, 6
interactive display program, 101
IOPB, 118
IOPB generator, 121

J

Jacobian matrix, 32, 57
Janikow, C.Z., 17
Jeroslow, R.G., 21

K

Karmarkar, N., 19
Kazemi, M., 17
kernel
  see nullspace
Krawczyk, J.B., 17
Kuhn, H.W., 26
Kuhn-Tucker conditions, 26
Kühner, W., 163

L

Lagrange, J., 19
Lagrange multiplier, 26, 27, 29, 33, 52, 61
  positive, 31, 35
  zero, 30, 31, 35, 36
Lagrangian function, 33, 56
lattice, 6, 20
Leyffer, S., 44
linear programming, 2, 17, 18
  integer, 19
  large-scale, 17
Liu, L., 47
local minimum
  strong, 31
Lustig, I., 19

M

manufacturing, 62
Marsten, R., 19

mathematical programming, 18
Mathur, K., 24
Mawengkang, H., 16, 24, 45, 64, 134, 138, 159
Mazzola, J.B., 46
McCormick, G.P., 7, 19
McNickle, D.C., 134
Mead, R.A., 5
mean value theorem (MVT), 25
Michalewicz, Z., 17
minimum
  global, 8, 10
  unconstrained, 4
minimum ratio, 101
minor, 26
MINOS, v, 2, 41, 45, 46, 48, 53, 61
MINOS/AUGMENTED, v, 47
Minoux, M, 163
MINTO, v, 40, 47, 99
  interactive, vi
MINTO/INTERACTIVE, 101, 122
MODULA-2, 121, 122
Mohd, I.B., 16
Moore, R.E., 16
Morrow, M., 17
MPS generator, 120
MS-DOS, 99
Mulvey, J., 19
Murray, W., 5, 6, 19, 26, 37, 38, 40, 48, 49
Murtagh, B.A., ii, v, 1, 16, 17, 19, 40, 41, 45, 46, 48, 49, 60 - 65, 73, 74, 76, 82, 84, 85, 96, 99, 134, 138, 162
mutation, 16
MVT
  see mean value theorem
Myers, D.C., 46, 144

N

NATTOPAR, 118
neighborhood, 4, 25
neighborhood search, v, 63
Nelder, J.A., 5
Nemhauser, G.L., 23, 43, 45, 163
network flow, 46
Newton, I., 19, 38
Newton's method, 37 - 39
  finite-difference, 40
Newton-like methods, 4
Newton direction, 40
Nievergelt, J., 163
NLIP
  boolean formulation of 0-1, 46
nonlinear integer programming, 2, 42
nonlinear programming, 2
NP-completeness, 42
NTOS, 118
nullspace, 27, 31, 33, 36

O

objective, 1, 4, 18
OD-pair
  see OD-pair, 165
operations research, 2
OPMSG, 118
OPS, 118
optimal network design, 2
optimal power flow, 2, 62
optimization
  combinatorial, 2, 6, 20
  constrained, 26
  global, 2, 4
  local, 2
  unconstrained, 5, 25
Orden, A., 18
origin-destination pair
  see OD-pair, 165
outer approximation, 43

P

Papadimitriou, C.H., 163
Papoulias, S.A., 138
Parrish, S.H., 163
Pascal, 99, 122
  Turbo, 99, 120 - 122
perturbation
  binding, 29, 31
  feasible, 29, 35
  non-binding, 29
pipeline network design, 45
polyhedron, 17
polynomial complexity, 19
polytope
  convex, 17
portfolio construction, 2
positive definiteness, 4, 25, 29, 38
Powell, M.J., 40
Press, W.H., 16, 37
pricing vector, 52
process engineering, 2, 62
projected gradient vector, 27, 33
projected Hessian matrix, 27, 29
  of Lagrangian, 34
projected Lagrangian method, 56

Q

QAP
  see quadratic assignment problem
QIP
  see quadratic integer program, 12
QIP generator, 121
Qiu, Y., 163
quadratic assignment problem, 16, 45, 62
quadratic function, 25
quadratic integer program, 12
quadratic interpolation, 37
quasi-Newton methods, 5, 39, 40
QWERTY keyboard, 121

R

rank, 33
rank-two update, 40
Rao, M.R., 163
Ratschek, H., 4, 16
Ravindran, A., 46, 132
READCOLS, 172
reduced costs vector, 52
reduced space, 6
Reingold, E.M., 163
Rendl, F., 16
revised simplex method, 17, 18, 52
Rokne, J., 4, 16
Rudeanu, S., 24
Ryan, D.M., 23, 42

S

Salkin, H.M., 24
Saltzman, M., 19
Saunders, M.A., v, 1, 17, 19, 41, 46, 48, 61
Scarf, H.E., v, 42, 63, 72
Shanker, K., 100, 153
Shanno, D., 19, 40
Sherman-Morrison identity, 86
Shin, D., 7
simplex, 17
  basis, v
simulated annealing, 2, 16
smooth function, 5
smoothness, 25, 37, 38
SMOV2, 119
sparsity, 2, 19
SPECS, 99
spectrum, 26
SPMOVCHK, 119
spreadsheet, 101
steepest descent, 39
Steiglitz, K., 163
Steinberg, L., 2
Strang, G., 19
Subramanian, R., 19
Sugden, B.E., iii
Sugden, I., iii
Sugden, J.E., iv
Sugden, S.J., 160, 163
Sugden, S.J., jr., iii
Sugden, V.L.M., iv
SUMT, 7
SUPERADJ, 119
superbasic variable, v, 3, 41, 50, 61, 62, 100
SUPERMOV, 119

T

Taylor’s theorem, 25
Taylor series, 27, 29, 40, 61
Teukolsky, S.A., 16, 37
Tucker, A.W., 26
Tweedie, R.L., ii
Tzen, Y.J., 100, 153

U

UPDATXFG, 119

V

Van Laarhoven, P.J.M., 16
Vetterling, W.T., 16, 37

W

Ward, T.L., 16
Wasil, E., 47
Wayner, P., 17
Weihmayer, R., 161
Wilhelm, M.R., 16
Wolfe, P., 18, 48
Wolsey, L.A., 23, 43, 45, 163
Wright, M.H., 5, 6, 19, 26, 37, 38, 40, 48, 49

Z

Zadeh, N., 163