ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΒΙΟΜΗΧΑΝΙΑΣ

Μεταπτυχιακή Εργασία

ΕΝΑΣ ΑΚΡΙΒΗΣ ΑΛΓΟΡΙΘΜΟΣ ΓΙΑ ΜΙΑ ΟΜΑΔΑ ΤΕΤΡΑΓΩΝΙΚΩΝ ΠΡΟΒΛΗΜΑΤΩΝ ΣΑΚΙΔΙΟΥ ΜΕ ΠΕΡΙΟΡΙΣΜΟΥΣ ΕΥΡΟΥΣ ΚΑΙ ΑΝΩ ΟΡΙΩΝ

υπό

ΑΝΔΡΕΑ ΓΑΒΡΑΝΗ

Μηχανικού Τηλεπικοινωνιών και Ηλεκτρονικών Σχολής Ικάρων, 2003

Υπεβλήθη για την εκπλήρωση μέρους των

απαιτήσεων για την απόκτηση του

Μεταπτυχιακού Διπλώματος Ειδίκευσης

2007

Εγκρίθηκε από τα Μέλη της Πενταμελούς Εξεταστικής Επιτροπής:

Πρώτος Εξεταστής (Επιβλέπων)	Δρ. Γεώργιος Κοζανίδης Λέκτορας, Τμήμα Μηχανολόγων Μηχανικών Βιομηχανίας, Πανεπιστήμιο Θεσσαλίας
Δεύτερος Εξεταστής	Δρ. Γεώργιος Λυμπερόπουλος Αναπληρωτής Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών Βιομηχανίας, Πανεπιστήμιο Θεσσαλίας
Τρίτος Εξεταστής	Δρ. Αθανάσιος Ζηλιασκόπουλος Αναπληρωτής Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών Βιομηχανίας, Πανεπιστήμιο Θεσσαλίας
Τέταρτος Εξεταστής	Δρ. Δημήτριος Παντελής Διδάσκων ΠΔ 407/80, Τμήμα Μηχανολόγων Μηχανικών Βιομηχανίας, Πανεπιστήμιο Θεσσαλίας
Πέμπτος Εξεταστής	Δρ. Παπαδημητρίου Κωνσταντίνος Καθηγητής, Τμήμα Μηχανολόγων Μηχανικών Βιομηχανίας, Πανεπιστήμιο Θεσσαλίας

Ευχαριστίες

Πρώτα απ’ όλα, θέλω να ευχαριστήσω τον επιβλέποντα της μεταπτυχιακής εργασίας μου, Λέκτορα Γεώργιο Κοζανίδη, για την πολύτιμη βοήθεια και καθοδήγησή του κατά τη διάρκεια της μελέτης μου. Επίσης, είμαι ευγνώμων στα υπόλοιπα μέλη της εξεταστικής επιτροπής της μεταπτυχιακής εργασίας μου, Καθηγητές κκ Γεώργιο Λυμπερόπουλο, Αθανάσιο Ζηλιασκόπουλο, Δημήτριο Παντελή και Κωνσταντίνο Παπαδημητρίου για την προσεκτική ανάγνωση της εργασίας μου και για τις πολύτιμες γνώσεις που μου προσέφεραν κατά τη διάρκεια των σπουδών μου. Οφείλω ευχαριστίες σε όλους τους κατά καιρούς καθηγητές μου, οι οποίοι συνετέλεσαν καθοριστικά στην απόκτηση του θεωρητικού μου υποβάθρου, το οποίο αποδείχτηκε απαραίτητο για την ολοκλήρωση μιας ιδιαίτερα απαιτητικής εργασίας. Ευχαριστώ τη Μαρία Ζαβού για την κατανόησή της, ιδιαίτερα κατά τη διάρκεια των τελευταίων μηνών της προσπάθειάς μου και όλους τους φίλους μου για την ηθική τους συμπαράσταση. Πάνω απ’ όλα, είμαι ευγνώμων στους γονείς μου, Θανάση και Μένη, και στην αδερφή μου Ασπελίνα για την ολόψυχη αγάπη και διαρκή υποστήριξή τους, στους οποίους και αφιερώνω την μεταπτυχιακή εργασία.

Ανδρέας Γαβράνης

ΕΝΑΣ ΑΚΡΙΒΗΣ ΑΛΓΟΡΙΘΜΟΣ ΓΙΑ ΜΙΑ ΟΜΑΔΑ ΤΕΤΡΑΓΩΝΙΚΩΝ ΠΡΟΒΛΗΜΑΤΩΝ ΣΑΚΙΔΙΟΥ ΜΕ ΠΕΡΙΟΡΙΣΜΟΥΣ ΕΥΡΟΥΣ ΚΑΙ ΑΝΩ ΟΡΙΩΝ

ΑΝΔΡΕΑΣ ΓΑΒΡΑΝΗΣ

Πανεπιστήμιο Θεσσαλίας, Τμήμα Μηχανολόγων Μηχανικών Βιομηχανίας, 2007

Επιβλέπων Καθηγητής: Δρ. Γεώργιος Κοζανίδης, Λέκτορας Βελτιστοποίησης Συστημάτων Παραγωγής / Υπηρεσιών

Περίληψη

Αυτή η μελέτη αφορά συνεχή προβλήματα Quadratic Knapsack με περιορισμούς άνω ορίων. Τα προβλήματα αυτά αποτελούν ειδικές περιπτώσεις Quadratic Programming και γενικότερα Μη Γραμμικού Προγραμματισμού. Η επιβολή περιορισμών τύπου Knapsack σε τέτοιου είδους προβλήματα έχει αρκετές ενδιαφέρουσες θεωρητικές εφαρμογές.

Επιπρόσθετα παρουσιάζονται εφαρμογές σε σημαντικούς τομείς που χρησιμοποιούν τη μορφοποίηση αυτή ως βάση για την επίλυση προβλημάτων, όπως για παράδειγμα η Βέλτιστη Επιλογή και Αναπροσαρμογή Χαρτοφυλακίου στον Οικονομικό Κλάδο, Προβλήματα Μεταφοράς και Ροών σε Δίκτυα στην Επιχειρησιακή Έρευνα, ο Ισοσκελισμός Πινάκων στο Μαθηματικό Κλάδο καθώς και εφαρμογές στον Τομέα της Συντήρησης Αεροσκαφών.

Οι μελέτες που έχουν γίνει μέχρι τώρα αφορούν την κλασσική μορφοποίηση όπου ο περιορισμός τύπου Knapsack ικανοποιείται σαν ισότητα. Σε αυτή την μεταπτυχιακή εργασία ερευνάται η περίπτωση όπου επιτρέπονται αποκλίσεις γύρω από μια κεντρική τιμή. Προτείνουμε και αναλύουμε έναν καινούριο αλγόριθμο και τροποποιούμε έναν ήδη υπάρχοντα για να καλύψουμε αυτή τη διαφοροποίηση. Τέλος παρουσιάζουμε αποτελέσματα που προκύπτουν από την υλοποίηση και εφαρμογή του αλγορίθμου σε διάφορα αριθμητικά προβλήματα.

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF MECHANICAL & INDUSTRIAL ENGINEERING

Postgraduate Work

AN EXACT ALGORITHM FOR A BOX CONSTRAINED CLASS OF QUADRATIC PROGRAMS SUBJECT TO UPPER BOUNDS

by

ANDREAS GAVRANIS

Electronics and Telecommunications Engineer of Hellenic Air Force Academy, 2003

Submitted in partial fulfillment

requirements for

Postgraduate Specialization Diploma

2007

Approved by:

First Examiner (Supervisor)	Dr. George Kozanidis Lecturer, Department of Mechanical & Industrial Engineering, University of Thessaly
Second Examiner	Dr. George Liberopoulos Associate Professor, Department of Mechanical & Industrial Engineering, University of Thessaly
Third Examiner	Dr. Athanasios Ziliaskopoulos Associate Professor, Department of Mechanical & Industrial Engineering, University of Thessaly
Fourth Examiner	Dr. Dimitrios Pandelis Instructor (P.D. 407/80), Department of Mechanical & Industrial Engineering, University of Thessaly
Fifth Examiner	Dr. Constantinos Papadimitriou Professor, Department of Mechanical & Industrial Engineering, University of Thessaly

Acknowledgements

First and foremost, I want to thank my postgraduate work supervisor, Lecturer Dr. George Kozanidis, for his valuable help and guidance throughout this work. I am also grateful to the other members of the examining committee of my postgraduate work Dr. Athanasios Ziliaskopoulos, Dr George Liberopoulos, Dr. Dimitrios Pandelis and Dr. Costas Papadimitriou for the close examination of my work and for the valuable knowledge I was given during my post-graduate studies. I owe grateful thanks to all the teachers I learned from, so as to obtain the essential theoretical background for the completion of a demanding study. I thank Maria Zavou for her understanding, especially through the last months of my effort as well as all my friends for their ethical support. Above all, I am grateful to my parents, Thanasis and Meni, and my sister Aspelina for their wholehearted love and constant support, to whom I dedicate this postgraduate work.

Andreas Gavranis

AN EXACT ALGORITHM FOR A BOX CONSTRAINED CLASS OF QUADRATIC PROGRAMS SUBJECT TO UPPER BOUNDS

ANDREAS GAVRANIS

University of Thessaly, Department of Mechanical & Industrial Engineering, 2007

Supervising Professor: Dr. George Kozanidis, Lecturer in Optimization Methods of Production/Service Systems

Abstract

This thesis considers continuous Quadratic Knapsack problems with bound constraints. These problems belong to the family of Quadratic Programming which is a major subdivision of Nonlinear Optimization. The addition of Knapsack constraints on Quadratic Programming problems is shown to have numerous applications, including Quadratic Programming defined on the convex hull of a set of points and the maximum clique problem.

Moreover important fields of study that use Quadratic Knapsack as core formulation are being presented. These include the Optimal Portfolio Selection, Quadratic Transportation, Multi-commodity Network Flows, Matrix Balancing problems and Aircraft Maintenance.

Traditional approaches for accommodating such Quadratic Knapsack constraints have been proposed and analyzed for the case of a single tight-bounded Knapsack constraint. Instead we introduce the case where deviation from the target value of the Knapsack constraint is allowed. In order to deal with our problem needs we modify an existing algorithm, and we propose and analyze a new one. Computational results on a variety of test problems are presented.

Contents

List of Tables

List of Figures

Chapter 1 Introduction

Before progressing into the mathematical content of this thesis, it is important to first provide some context and motivation. This thesis grew out of research in the area of optimization. As the name suggests, optimization deals with the application or development of mathematical programming techniques for decision-making. Evidence of vigorous research activity within this field is easy to document.

This introduction gives an overview of the optimization concept showing major subfields and describing the way mathematical programming techniques are applied to problems so as to render solutions. The introduction is closed with an overview of the complete work, placing it in relation to the rest of the literature.

1.1 Motivation and Background

This thesis presents an application of mathematical programming, specifically Quadratic Programming, which provides useful information to aid decision-makers. It is the work of subsequent sections and chapters to show exactly how a topic such as Quadratic Programming can be productively applied to many types of decisions. In detail, we focus on a specific application in Aircraft Maintenance which introduces a new formulation of the original Continuous Quadratic Knapsack formulation.

The main contribution of the research reported in this work is that we develop a new exact algorithm for a special class of Continuous Quadratic Knapsack Problems having reasonable solution times for nearly all instances encountered in practice, despite having Quadratic time bounds for a number of highly contrived problem instances.We give proof of the optimality of the algorithm, implement it in C programming language and give numerical results. We also describe a Quadratic Knapsack framework for the formulation, analysis and computation of solutions to a specific problem of military-aircraft maintenance. Last we present a modification of an exact algorithm presented by Pardalos and Kovoor **[13]**, in order to cope with the specific formulation of the problem in case the large number of variables poses long execution time problem.

1.2 Quadratic Knapsack

The Quadratic Knapsack problem (QKP) is one of the simplest Quadratic Programming problems defined as follows:

$$Min\ z(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x} - \mathbf{c}\mathbf{x} = \sum_{i=1}^{I}\left(\frac{1}{2}q_i x_i^2 - c_i x_i\right)$$

subject to $\sum_{i=1}^{n} d_i x_i = d_0$

$$x_i \geq 0$$

The continuous bounded Quadratic Knapsack problem is defined as follows

$$Min\ z(\mathbf{x}) = \sum_{i=1}^{I}\left(\frac{1}{2}q_i x_i^2 - c_i x_i\right)$$

subject to

$$\sum_{i=1}^{n} d_i x_i = d_0$$

$$a_i \le x_i \le b_i, \quad i = 1, 2, \dots, n$$

where $\mathbf{x} \in \Re^n$ is a variable vector, $\mathbf{Q} \in \Re^{n \times n}$, $\mathbf{c} \in \Re^n$ and d_0 is a scalar.

1.3 Literature Review

The Quadratic Knapsack problems are mainly classified by the nature of matrix $\mathbf{Q}$. When the matrix $\mathbf{Q}$ is positive semidefinite, i.e., the objective function $z(\mathbf{x})$ is convex, problem can be solved in polynomial time by the ellipsoid algorithm **[8]**, and several kinds of interior point algorithms (e.g. **[7]**, **[11]**, **[5]**, which solve general convex Quadratic problems including (QKP) as a special case). Also, P.M. Pardalos, Y. Ye and C.G. Han **[15]** show a potential reduction algorithm for a special case of (QKP) defined below.

$$\min f(x) = \mathbf{x}^T \mathbf{Q} \mathbf{x}$$

$$\text{s.t} \quad \sum_{i=1}^{n} x_i = 1, x \ge 0$$

where $\mathbf{Q}$ is a $n \times n$ symmetric matrix.

In particular, when (QKP) has a diagonal matrix $\mathbf{Q}$ with positive elements, an $O(n)$ algorithm has been proposed by P. Brucker **[3]**. The algorithm generates the corresponding KKT condition using binary search. Pardalos and N. Kovoor **[13]** also propose an $O(n)$ randomized method.

The convex case is important because of its frequent appearance as a subproblem in many application areas. Among those are general convex Quadratic Programming **[9]**, multicommodity network flow problems **[1]**, resource management **[2]**, and portfolio selection problems **[10]**.

The problem becomes extremely difficult if $z(\mathbf{x})$ is not convex. S. Sahni **[16]** shows that problems with negative diagonal matrix $\mathbf{Q}$ are $Np-hard$, which implies that the general indefinite case is also $Np-hard$.

Let $\upsilon_1,...,\upsilon_n$ be points in $\Re^m$ whose convex hull is P. The least distance problem is that of finding the point of P having the smallest Euclidean norm. This problem can be stated as

$$\min\ \mathbf{x}^T\mathbf{x}$$

$$\text{s.t} \qquad x=\sum_{i=1}^{n} z_i \upsilon_i$$

$$\sum_{i=1}^{n} z_i = 1\ , z_i \geq 0 , i=1,...,n$$

The above problem can be formulated as in **[15]** with $\mathbf{Q}=\mathbf{V}^T\mathbf{V}$ and $\mathbf{V}=(\upsilon_1,...,\upsilon_n)$.

As we see in the above, the indefinite case arises in several combinatorial optimization problems. For example, given a graph $G(V,E)$ where $V=\{1,...,n\}$ is a set of vertices and $E \subseteq V^2$ is a set of edges, a clique is a complete subgraph of G. The *maximum clique* problem is the problem of finding the maximum complete subgraph of G. For each vertex υ_i, introduce a variable $x_i, i=1,...,n$. This problem can be formulated in the following way:

$$Max\, f(x) = \sum_{(i,j)\in E} x_i x_j$$

$$\text{s.t.} \qquad \sum_{i=1}^{n} x_i = 1\,, x_i \geq 0\,, i = 1,\ldots,n$$

If G has a maximum clique of size k, then the global maximum is $f(x^*) = \frac{1}{2}\left(1 - \frac{1}{k}\right)$. We can also formulate the *maximum independent set problem* and the *node covering problem* in a similar fashion.

One can also formulate any Quadratic minimization problem over a convex hull by the Quadratic Knapsack problem. Consider the problem of the form:

$$\min_{z\in P} \ \text{q(z)= z}^T Mz \qquad \textbf{(1)}$$

where $z, r \in \Re^m, M \in \Re^{m\times m}$ and $P \subseteq \Re^m$ is the polytope described as the convex hull of a given set of points $\{v_1,\ldots,v_n\}$. It can be verified easily that the above general Quadratic problem has the following equivalent formulation

$$\text{global } \min \ \text{f}(\text{x}) = x^T Qx \qquad \textbf{(2)}$$

$$\text{s.t } x \in D = \left\{x : \sum_{i=1}^{n} x_i = 1, x \geq 0\right\},$$

with $Q = V^T MV$ and $V = \{v_1,\ldots,v_n\}$.

Let z^* and x^* be optimum solutions of (1) and (2), respectively. Then we have

$q(z^*) = \min_{z \in P}$ q(z)= $f(x^*) = \min_{x \in D} f(x)$ and moreover $z^* = Vx^*$.

There exist only a few algorithms for obtaining a global optimum solution for the case of the general indefinite **Q**. See **[15]** for a partitioning approach as well as an interior point method, while **[4]** surveys algorithms for general nonconvex Quadratic problems.

The case when the objective function is separable has also been well investigated by several authors. Some practical algorithms to obtain an exact solution are reported in **[14]**, **[6]**. S.A. Vavasis **[18]** shows an $O\left(n(\log n)^2\right)$algorithm for finding a local minimum of the problem, while K.G. Murty and S.N. Kabadi **[12]** show that verifying a local minimum for an indefinite Quadratic problem with general constraints is $Np-hard$. Also, Vavasis **[17]** gives an ε-approximation algorithm which is weakly polynomial in the problem size if the number of negative diagonal elements is fixed.

1.4 Structure of Postgraduate Work

The rest of this postgraduate work is divided into five chapters. More specifically:

In Chapter 2, we consider first a family of combinatorial problems known under the name Knapsack Problems and we present some important applications. Next, we study the foundations of nonlinear programming and focus on one of its major subsectors, namely Quadratic Programming .We introduce the Karush-Kuhn-Tucker (KKT) conditions for optimality and then use these conditions to provide a linear transformation of the Quadratic Programming Problem which can be dealt with the modified-SIMPLEX method.

In Chapter 3, we present some major applications of the Quadratic Knapsack formulation, which denote its major role in real-life applications. More precisely we begin with the Portfolio Selection and the Portfolio Rebalancing Problems. We continue with the Quadratic Transportation Problem and present the advantages of a Quadratic Programming formulation for Spatial Interaction Patterns over a linear one. We then specialize on Multi-commodity Network Flow Problems. Matrix balancing problems are being introduced next and we finish our application reference by introducing a special military-aircraft maintenance problem with a slight differentiation from the classical formulation of the Quadratic Knapsack Problem, in that a box Knapsack constraint is imposed

In Chapter 4, we focus on this special case of Box Constrained Quadratic Knapsack Problem with upper Bounds. After formulating the Problem we use the KKT Conditions of optimality applied on our specific problem in order to characterize the optimal solution. Global optimality is proven next. We then present an algorithm for the solution of the problem also dealing with algorithm optimality and complexity. In the last part of the chapter, we present the modification of an existing algorithm in order to make it applicable to our specific case.

In Chapter 5, we consider some theoretical and numerical aspects of the algorithms implementation comparing results with other known solution tools (i.e. AMPL) and present results of some numerical experiments so as to make clear the way the algorithm works.

In Chapter 6, this postgraduate work is summarized and directions for further research are given.

Appendix A contains the C Programming Language code of the algorithms implementation. Appendix B contains the AMPL modeling file.

Chapter 2 Knapsack Problems and Nonlinear Programming

2.1 Introduction

We begin this chapter by giving an overview of the family of Knapsack Problems, and by showing several applications of theoretical as well as of practical interest. We then introduce the basic concepts of Nonlinear Programming, and describe some basic application examples which make the difference with Linear Programming clear. Following this we state the Karush-Kuhn-Tucker conditions for Constrained Optimization. Quadratic programming is analyzed next and a transformation to a linear formulation is given by applying the KKT Conditions. One solution technique that can be used on the transformed Quadratic Problem, namely modified-SIMPLEX, is analyzed last.

2.2 The Family of Knapsack Problems

2.2.1 Knapsack Problem Formulation

This section considers several problems from the family of Knapsack Problems. In all variants of the problem we have a collection of items, each with a profit p_j and weight w_j, which are packed into one or more Knapsacks of capacity c. We will assume that all coefficients p_j, w_j, c are positive numbers although weaker assumptions sometimes may be handled in the individual problems.

The 0-1 Knapsack Problem is the problem of choosing a subset of the n items such that the corresponding profit sum is maximized but the knapsack capacity is not exceeded. This may be formulated as the following maximization problem:

Maximize $\sum_{j=1}^{n} p_j x_j$

Subject to $\sum_{j=1}^{n} w_j x_j \le c$

$x_j \in \{0,1\}, j = 1,\dots,n$

where x_j is a binary variable equal to 1 if item j is included in the Knapsack and 0 otherwise.

If we have a maximum quantity m_j for each item type j, then the Bounded Knapsack Problem arises, formulated as:

Maximize $\sum_{j=1}^{n} p_j x_j$

Subject to $\sum_{j=1}^{n} w_j x_j \le c$

$x_j \in \{0,1,\dots,m_j\}, j = 1,\dots,n$

Here, x_j is the number of items of each type to be included in the Knapsack, in order to obtain the largest objective value.

The Unbounded Knapsack Problem is a generalization of the Bounded Knapsack Problem, where an unlimited number of items for each type is available:

Maximize $$\sum_{j=1}^{n} p_j x_j$$

Subject to $$\sum_{j=1}^{n} w_j x_j \le c$$

$$x_j \ge 0 \text{ integer}, j = 1,\dots,n$$

In general there is no advantage by transforming an Unbounded Knapsack Problem to the bounded version.

Another generalization of the 0-1 Knapsack problem is to choose exactly one item j from each of k classes N_i, $i = 1,\dots,k$ such that the profit sum is maximized. This gives the Multiple-choice Knapsack Problem which is defined as

Maximize $$\sum_{i=1}^{k} \sum_{j \in N_i} p_{ij} x_{ij}$$

Subject to $$\sum_{i=1}^{k} \sum_{j \in N_i} w_{ij} x_{ij} \le c$$

$$\sum_{j \in N_i} x_{ij} = 1 ,\ i = 1,\dots,k$$

$$x_{ij} \in \{0,1\} ,\ i = 1,\dots,k ,\ j \in N_i$$

Here the binary variable $x_{ij} = 1$ states that item j was chosen from class i. The constraint $\sum_{j \in N_i} x_{ij} = 1$, $i = 1,\dots,k$ ensures that exactly one item is chosen from each class.

If the profit p_j equals the weight w_j for each item in a 0-1 Knapsack Problem we obtain the Subset-sum Problem, which may be formulated as:

Maximize $\sum_{j=1}^{n} w_j x_j$

Subject to $\sum_{j=1}^{n} w_j x_j \le c$

$x_j \in \{0,1\}$, $j = 1,\dots,n$

The name indicates that it can also be seen as the problem of choosing a subset of the values $w_1,\dots,w_n$ such that the sum is as large as possible without exceeding c.

Now, imagine a cashier who has to give back an amount of money c by using the smallest possible amount of the coins $w_1,\dots,w_n$. The Change-making Problem is then defined as:

Minimize $\sum_{j=1}^{n} x_j$

Subject to $\sum_{j=1}^{n} w_j x_j = c$

$x_j \ge 0$ integer , $j = 1,\dots,n$

where w_j is the face value of coin j, and we assume that an unlimited amount of each coin is available. The optimal number of each coin j that should be used is then expressed by x_j.

This problem may be considered as a minimization variant of the Unbounded Knapsack Problem, where $p_j = 1$ for $j = 1,\dots,n$ and where equality must hold in the capacity constraint.

If we have to choose n items to pack in m Knapsacks of possibly different capacities c_i such that the total profit is maximized we obtain the Multiple Knapsack Problem

Maximize $\sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij}$

Subject to $\sum_{j=1}^{n} w_j x_{ij} \le c_i \quad i = 1,\dots,m$

$\sum_{i=1}^{m} x_{ij} \le 1, \quad j = 1,\dots,n$

$x_{ij} \in \{0,1\}, \quad i = 1,\dots,m,\ j = 1,\dots,n$

Here $x_{ij} = 1$ indicates that item j should be packed into Knapsack i, while the constraint $\sum_{j=1}^{n} w_j x_{ij} \le c_i$ ensures that the capacity constraint of Knapsack i is satisfied. The constraint $\sum_{i=1}^{m} x_{ij} \le 1$ ensures that each item is chosen at most once.

A very useful model is the Bin-packing Problem where all n items should be packed in a number of equally sized bins, such that the number of bins actually used is as small as possible. Thus we have

Minimize $\sum_{i=1}^{n} y_i$

Subject to $$\sum_{j=1}^{n} w_j x_{ij} \le c y_i \quad i = 1,\dots,n$$

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad j = 1,\dots,n$$

$$y_i \in \{0,1\}, \qquad i = 1,\dots,n$$

$$x_{ij} \in \{0,1\}, \qquad i = 1,\dots,m, \; j = 1,\dots,n$$

where y_i indicates whether bin i is used, and x_{ij} states that item j should be packed in bin i. The constraint $\sum_{i=1}^{m} x_{ij} = 1$ ensures that every item is packed exactly once, while inequality $\sum_{j=1}^{n} w_j x_{ij} \le c y_i$ ensures that the capacity constraint holds for all bins actually used.

The most general form of a Knapsack Problem is the Multi-constrained Knapsack Problem, which basically is a general Integer Programming Problem where all coefficients, p_j, w_{ij} and c_i are nonnegative integers. Thus it may be formulated as

Maximize $$\sum_{j=1}^{n} p_j x_j$$

Subject to $$\sum_{j=1}^{n} w_{ij} x_j \le c_i, \; i = 1,\dots,m$$

$$x_j \ge 0 \text{ integer}, \; j = 1,\dots,n$$

2.2.2 Knapsack Applications

Knapsack Problems have numerous applications in theory as well as in practice. From a theoretical point of view, the simple structure pleads for exploitation of numerous interesting properties, that can make the problems easier to solve. Knapsack Problems also arise as subproblems in several algorithms for more complex combinatorial optimization problems, and these algorithms will benefit from any improvement in the field of Knapsack Problems.

Despite its name, practical applications of Knapsack Problems are not limited to packing problems: Assume that n projects are available to an investor, and that the profit obtained from the jth project is $p_j, j = 1,...,n$. It costs w_j to invest in project j, and only c dollars are available. An optimal investment plan may be found by solving a 0-1 Knapsack Problem.

Another application appears in a restaurant, where a person has to choose k courses, without surpassing the amount of c calories, his diet prescribes. Assuming that there are N_i dishes to choose among for each course $i = 1,...,k$, and w_{ij} is the nutritive value while p_{ij} is a rating saying how well each dish tastes. Then an optimal meal may be found by solving the Multiple-choice Knapsack Problem **[37]**.

The Bin-packing Problem has been applied for cutting iron bars in a kibbutz **[39]**, in order to minimize the number of bars used each day. Here w_j is the length of each piece demanded, while c is the length of each bar, as delivered from the factory.

Apart from these simple illustrations we should mention the following major applications: Problems in cargo loading, cutting stock, budget control, and financial management may be formulated as Knapsack Problems, where the specific model depends on the side constraints present. Sinha and Zoltners **[37]** proposed to use Multiple-choice Knapsack Problems to select which components should be linked in series in order to maximize fault tolerance. Diffe and Hellman **[36]** designed a public cryptography scheme whose security relies on the difficulty of solving the Subset-sum Problem. Martello and Toth **[37]** mention that two-processor scheduling problems may be solved as a Subset-sum Problem. Finally the Bin-packing Problem may be used for packing envelopes with a fixed weight limit.

The more theoretical applications either appear where a general problem is transformed to a Knapsack Problem, or where the Knapsack Problem appears as subproblem, e.g. for deriving bounds in a branch-and-bound algorithm designed to solve more complex problems. In the first category G. B. Mathews back in 1897 [34] showed how several constraints may be aggregated to one single Knapsack constraint, making it possible to solve any IP Problem as a 0-1 Knapsack Problem. Moreover Nauss [35] proposed to transform nonlinear Knapsack Problems to Multiple-choice Knapsack Problems. In the second category we should mention that the 0-1 Knapsack Problem appears as a sub problem when solving the Generalized Assignment Problem, which again is heavily used when solving Vehicle Routing Problems **[32]**. Also Krarup and Illes **[31]** apply a Knapsack type relaxation in connection with finite projective planes.

2.3 Nonlinear Programming

A key assumption of linear programming is that all its functions (objective function and constraints) are linear. Although this assumption essentially holds for numerous practical problems, it frequently does not hold. In fact, many economists have found that some degree of nonlinearity is the rule and not the exception in economic planning problems. Therefore, it is often necessary to deal directly with nonlinear programming problems, so we turn our attention to this important area.

In one general form, the nonlinear programming problem is to find $\mathbf{x} = (x_1, x_2, ..., x_n)$ so as to

$$\begin{aligned} &\text{Maximize} && f(\mathbf{x}) \\ &\text{Subject to} && g_i(\mathbf{x}) \le b_i \qquad i = 1, 2, ..., m \\ &&& \mathbf{x} \ge 0 \end{aligned}$$

where $f(\mathbf{x})$ and the $g_i(\mathbf{x})$ are given functions of the n decision variables.

No general algorithm that will solve every specific problem fitting this format is available. However, substantial progress has been made for some important special cases of this problem by making various assumptions about these functions, and research is continuing very actively. This area is quite extensive, and there is not enough space to survey it completely. Besides, that is beyond the scope of this research.

2.4 Types of Nonlinear Programming Problems

Nonlinear programming problems come in many different shapes and forms. Unlike the simplex method for linear programming, no single algorithm can solve all these different types of problems. Instead, algorithms have been developed for various individual classes (special types) of nonlinear programming problems. The class which is of great interest in this research is introduced briefly in the following section.

2.4.1 Linearly Constrained Optimization

Linearly constrained optimization problems are characterized by constraints that completely fit linear programming, so that all the $g_i(\mathbf{x})$ constraint functions are linear, but the objective function $f(x)$ is nonlinear. The problem is considerably simplified by having just one nonlinear function to take into account, along with a linear programming feasible region. A number of special algorithms based upon extending the simplex method to consider the nonlinear objective function have been developed. One important special case, which we consider next, is Quadratic Programming.

2.4.2 Quadratic Programming

Quadratic programming problems again have linear constraints, but now the objective function $f(x)$ must be Quadratic. Thus, the only difference between such a problem and a linear programming problem is that some of the terms in the objective function involve the square of a variable or the product of two variables.

Quadratic programming is very important, partially because such formulations arise naturally in many applications. For example, the problem of portfolio selection with risky securities described fits into this format. However, another major reason for its importance is that a common approach to solving general linearly constrained optimization problems is to solve a sequence of Quadratic Programming approximations.

2.5 The Karush-Kuhn-Tucker (KKT) Conditions for Constrained Optimization

We now focus on the question of how to recognize an optimal solution for a nonlinear programming problem (with differentiable functions).More precisely we focus on the necessary and under certain requirements sufficient conditions for an optimal solution.

In the preceding sections we already noted these conditions for unconstrained optimization, as summarized in the first two rows of Table 2.1. In the third row of Table 2.1 the conditions for the slight extension of unconstrained optimization where the only constraints are nonnegativity constraints are shown. As indicated in the last row of the table, the conditions for the general case are called the **Karush-Kuhn-Tucker conditions (or KKT conditions),** because they were derived independently by Karush **[19]** and by Kuhn and Tucker **[20]**. Their basic result is embodied in the following theorem.

Theorem 2.1 Assume that $f(\mathbf{x})$, $g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_m(\mathbf{x})$ are differentiable functions satisfying certain regularity conditions. Then $\mathbf{x}^* = \left(x_1^*, x_2^*, \ldots, x_n^*\right)$ can be an optimal solution for the nonlinear problem only if there exist m numbers $u_1, u_2, \ldots, u_m$ such that all the following KKT conditions are satisfied:

$$\left.\begin{array}{ll} 1. & \dfrac{\partial f}{\partial x_j} - \sum_{i=1}^{m} u_i \dfrac{\partial g_i}{\partial x_j} \le 0 \\ 2. & x_j^* \left(\dfrac{\partial f}{\partial x_j} - \sum_{i=1}^{m} u_i \dfrac{\partial g_i}{\partial x_j} \right) = 0 \end{array}\right\} \text{at } \mathbf{x} = \mathbf{x}^* \text{ for } j = 1, 2, \ldots, n$$

$$\left.\begin{array}{l} 3. \quad g_i(\mathbf{x}^*) - b_i \le 0 \\ 4. \ u_i\left(g_i(\mathbf{x}^*) - b_i\right) = 0 \end{array}\right\} \qquad \text{for } i = 1, 2, \ldots, m$$

5. $x_j^* \geq 0$ for $j = 1, 2, \ldots, n$

6. $u_i \geq 0$ for $i = 1, 2, \ldots, m$

Problem	Necessary Conditions for Optimality	Also Sufficient if:
One-variable unconstrained	$\frac{df}{dx} = 0$	f(x) concave
Multivariable unconstrained	$\frac{\partial f}{\partial x_j} = 0 \ (j = 1, 2, \ldots, n)$	f(x) concave
Constrained, nonnegativity constraints only	$\frac{\partial f}{\partial x_j} = 0 \ (j = 1, 2, \ldots, n)$ (or ≤ 0 if $x_j = 0$)	f(x) concave
General constrained problem	Karush-Kuhn-Tucker conditions	f(x) concave and $g_i(\mathbf{x})$ convex $(i = 1, 2, \ldots, m)$

Table 2.1 Necessary and sufficient conditions for optimality

Note that both conditions 2 and 4 require that the product of two quantities be zero. Therefore, each of these conditions is really saying that at least one of the two quantities must be zero. Consequently, condition 4 can be combined with condition 3 to express them in another equivalent form as

$(3,4)\ g_i(\mathbf{x}^*)-b_i=0$

$$\left(\text{or } \le 0 \text{ if } u_i=0\right) \text{ for } i=1,2,\ldots,m$$

Similarly, condition 2 can be combined with condition 1 as

$$(1,2)\frac{\partial f}{\partial x_j}-\sum_{i=1}^{m}u_i\frac{\partial g_i}{\partial x_j}=0$$

$$\left(\text{or} \le 0 \text{ if } x_j^*=0\right) \text{ for } j=1,2,\ldots,n$$

When m = 0 (no functional constraints), this summation drops out and the combined condition (1, 2) reduces to the condition given in the third row of **Table 2.1**. Thus, for m > 0, each term in the summation modifies the m = 0 condition to incorporate the effect of the corresponding functional constraint.

In conditions 1, 2, 4, and 6, the u_i, correspond to the dual variables of linear programming (we expand on this correspondence at the end of the section), and they have a comparable economic interpretation. However, the u_i, actually arose in the mathematical derivation as Lagrange multipliers. Conditions 3 and 5 do nothing more than ensure the feasibility of the solution. The other conditions eliminate most of the feasible solutions as possible candidates for an optimal solution.

However, note that satisfying these conditions does not guarantee that the solution is optimal. As summarized in the rightmost column of Table 13.3, certain additional convexity assumptions are needed to obtain this guarantee. These assumptions are spelled out in the following extension of the **Theorem 2.1**.

Corollary. Assume that $f(\mathbf{x})$ is a concave function and that $g_1(\mathbf{x}), g_2(\mathbf{x}), \dots, g_m(\mathbf{x})$ are convex functions (i.e., this problem is a convex programming problem), where all these functions satisfy the regularity conditions. Then $\mathbf{x}^* = \left(x_1^*, x_2^*, \dots, x_n^*\right)$ is an optimal solution, if and only if all the conditions of the theorem are satisfied.

For many complicated problems, it may be difficult, if not essentially impossible, to derive an optimal solution directly from the KKT conditions. Nevertheless, these conditions still provide valuable clues as to the identity of an optimal solution, and they also permit us to check whether a proposed solution may be optimal.

There also are many valuable indirect applications of the KKT conditions. One of these applications arises in the duality theory that has been developed for nonlinear programming to parallel the duality theory for linear programming. In particular, for any given constrained maximization problem (call it the primal problem), the KKT conditions can be used to define a closely associated dual problem that is a constrained minimization problem. The variables in the dual problem consist of both the Lagrange multipliers u_i $(i = 1, 2, \dots, m)$ and the primal variables x_j $(j = 1, 2, \dots, n)$. (For details on this formulation, see Chapter 8 of **[21]**. For a unified survey of various approaches to duality in nonlinear programming, see **[22]**.)In the special case where the primal problem is a linear programming problem, the x_j variables drop out of the dual problem and it becomes the familiar dual problem of linear programming. When the primal problem is a convex programming problem, it is possible to establish relationships between the primal problem and the dual problem that are similar to those for linear programming. For example, the strong duality property, which states that the optimal objective function values of the two problems are equal, also holds here. Furthermore, the values of the u_i variables in an optimal solution for the dual problem can again be interpreted as shadow prices ;i.e., they give the rate at which the optimal objective function value for the primal problem could be increased by (slightly) increasing the right-hand side of the corresponding constraint. We will see another indirect application of the KKT conditions in the next section.

2.6 Quadratic Programming

As already indicated in previous section, the Quadratic Programming problem differs from the linear programming problem only in that the objective function also includes x_j^2 and $x_i x_j, i \neq j$ terms. Thus, if we use matrix notation like that introduced at the beginning of Sec. 5.2, the problem is to find $\mathbf{x}$ so as to

Maximize $f(x) = \mathbf{cx} - \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Qx}$,

Subject to $\mathbf{Ax} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$

where $\mathbf{c}$ is a row vector, $\mathbf{x}$ and $\mathbf{b}$ are column vectors, $\mathbf{Q}$ and $\mathbf{A}$ are matrices, and the superscript T denotes the transpose of a matrix. The q_{ij} (elements of $\mathbf{Q}$) are given constants such that $q_{ij} = q_{ji}$ (which is the reason for the factor of $\frac{1}{2}$ in the objective function). By performing the indicated vector and matrix multiplications, the objective function then is expressed in terms of these q_{ij}, the c_j (elements of $\mathbf{c}$), and the variables as follows:

$$f(x) = \mathbf{cx} - \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Qx} = \sum_{j=1}^{n} c_j x_j - \frac{1}{2} \cdot \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij} x_i x_j$$

For each term where $i = j$ in this double summation, $x_i x_j = x_j^2$, so $-\frac{1}{2}q_{ij}$ is the coefficient of x_j^2. When $i \neq j$, then $-\frac{1}{2}\left(q_{ij}x_i x_j + q_{jj}x_j x_i\right) = q_{ij}x_i x_j$, so $-q_{ij}$ is the total coefficient for the product of x_i and x_j.

To illustrate this notation, consider the following example of a Quadratic Programming problem.

Maximize $f(x_{1,}x_2) = 15x_{1,} + 30x_2 + 4x_1x_2 - 2x_1^2 - 4x_2^2$,

subject to

$x_1 + 2x_2 \leq 30$ and

$x_1 \geq 0, x_2 \geq 0$.

In this case

$$\mathbf{c} = \begin{bmatrix} 15 & 30 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \mathbf{Q} = \begin{bmatrix} 4 & -4 \\ -4 & 8 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} 30 \end{bmatrix}$$

Note that Maximize $f(x) = \mathbf{cx} - \frac{1}{2}\mathbf{x}^{\mathbf{T}}\mathbf{Qx}$,

$$\mathbf{x}^{\mathbf{T}}\mathbf{Qx} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 4 & -4 \\ -4 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$=\begin{bmatrix}(4x_1-4x_2) & (-4x_1+8x_2)\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}$$

$$=4x_1^2-4x_2x_1-4x_1x_2+8x_2^2$$

$$=q_{11}x_1^2+q_{21}x_2x_1+q_{12}x_1x_2+q_{22}x_2^2$$

Multiplying through by $-\frac{1}{2}$ gives

$$-\frac{1}{2}\mathbf{x}^{\mathbf{T}}\mathbf{Qx}=-2x_1^2+4x_1x_2-4x_2^2$$

which is the nonlinear portion of the objective function for this example. Since $q_{11}=4$ and $q_{22}=8$, the example illustrates that $-\frac{1}{2}q_{jj}$ is the coefficient of x_j^2 in the objective function. The fact that $q_{12}=q_{12}=-4$ illustrates that both $-q_{ij}$ and $-q_{ji}$ give the total coefficient of the product of x_i and x_j.

Several algorithms have been developed for the special case of the Quadratic Programming problem where the objective function is a concave function. (A way to verify that the objective function is concave is to verify the equivalent condition that

$$\mathbf{x}^{\mathbf{T}}\mathbf{Qx}\geq 0$$

for all $\mathbf{x}$, that is, $\mathbf{Q}$ is a positive semidefinite matrix.) One of these algorithms **[23]**, the modified simplex method, has been quite popular because it requires using only the simplex method with a slight modification. The key to this approach is to construct the KKT conditions from the preceding section and then to re-express these conditions in a convenient form that closely resembles linear programming. Therefore, before describing the algorithm, we shall develop this convenient form.

2.6.1 The KKT Conditions for Quadratic Programming

For any Quadratic Programming problem, its KKT conditions can be reduced to a convenient form containing just linear programming constraints plus one complementarity constraint. In matrix notation again, this general form is

$$\mathbf{Qx} + \mathbf{A}^T\mathbf{u} - \mathbf{y} = -\mathbf{c}^T$$

$$\mathbf{Ax} + \mathbf{v} = \mathbf{b}$$

$$\mathbf{x} \geq 0 \quad \mathbf{u} \geq 0 \quad \mathbf{y} \geq 0 \quad \mathbf{v} \geq 0$$

$$\mathbf{x}^T\mathbf{y} + \mathbf{u}^T\mathbf{v} = 0$$

where the elements of the column vector $\mathbf{u}$ are the u_i, of the preceding section and the elements of the column vectors $\mathbf{y}$ and $\mathbf{v}$ are slack variables.

Because the objective function of the original problem is assumed to be concave and because the constraint functions are linear and therefore convex, the corollary to the **Theorem 2.1** applies. Thus, **x** is optimal if and only if there exist values of **y**, **u**, and **v** such that all four vectors together satisfy all these conditions. The original problem is thereby reduced to the equivalent problem of finding a feasible solution to these constraints.

It is of interest to note that this equivalent problem is one example of the linear complementarity problem, and that a key constraint for the linear complementarity problem is its complementarity constraint.

2.6.2 The Modified Simplex Method

The modified simplex method exploits the key fact that, with the exception of the complementarity constraint, the KKT conditions in the convenient form obtained above are nothing more than linear programming constraints. Furthermore, the complementarity constraint simply implies that it is not permissible for both complementary variables of any pair to be (nondegenerate) basic variables (the only variables > 0) when (nondegenerate) BF solutions are considered. Therefore, the problem reduces to finding an initial BF solution to any linear programming problem that has these constraints, subject to this additional restriction on the identity of the basic variables. (This initial BF solution may be the only feasible solution in this case.)

As we discussed in Sec. 4.6, finding such an initial BF solution is relatively straightforward. In the simple case where $\mathbf{c}^T \leq 0$ (unlikely) and $\mathbf{b} \geq 0$, the initial basic variables are the elements of $\mathbf{y}$ and $\mathbf{v}$ (multiply through the first set of equations by -1), so that the desired solution is $\mathbf{x} = 0$, $\mathbf{u} = 0$, $\mathbf{y} = -\mathbf{c}^T$, $\mathbf{v} = \mathbf{b}$. Otherwise, you need to revise the problem by introducing an artificial variable into each of the equations where $c_j > 0$ (add the variable on the left) or $b_i < 0$ (subtract the variable on the left and then multiply through by -1) in order to use these artificial variables (call them z_1, z_2, and so on) as initial basic variables for the revised problem. (Note that this choice of initial basic variables satisfies the complementarity constraint, because as nonbasic variables $\mathbf{x} = 0$ and $\mathbf{u} = 0$ automatically.)

Next, use phase 1 of the two-phase method to find a BF solution for the real problem; i.e., apply the simplex method (with one modification) to the following linear programming problem

Minimize $\quad Z = \sum_j z_j$,

subject to the linear programming constraints obtained from the KKT conditions, but with these artificial variables included.

The one modification in the simplex method is the following change in the procedure for selecting an entering basic variable.

Restricted-Entry Rule: When you are choosing an entering basic variable, exclude from consideration any nonbasic variable whose complementary variable already is a basic variable; the choice should be made from the other nonbasic variables according to the usual criterion for the simplex method.

This rule keeps the complementarity constraint satisfied throughout the course of the algorithm. When an optimal solution **x***, **u***, **y***, **v***, $z_1 = 0,\ldots,z_n = 0$ is obtained for the phase 1 problem, **x*** is the desired optimal solution for the original Quadratic Programming problem. Phase 2 of the two-phase method is not needed.

Example. We shall now illustrate this approach on the example given at the beginning of the section.

$\mathbf{Q} = \begin{bmatrix} 4 & -4 \\ -4 & 8 \end{bmatrix}$ is positive definite, so the algorithm can be applied.

The starting point for solving this example is its KKT conditions in the convenient form obtained earlier in the section. After the needed artificial variables are introduced, the linear programming problem to be addressed explicitly by the modified simplex method then is

Minimize $\quad Z = z_1 + z_2$,

subject to

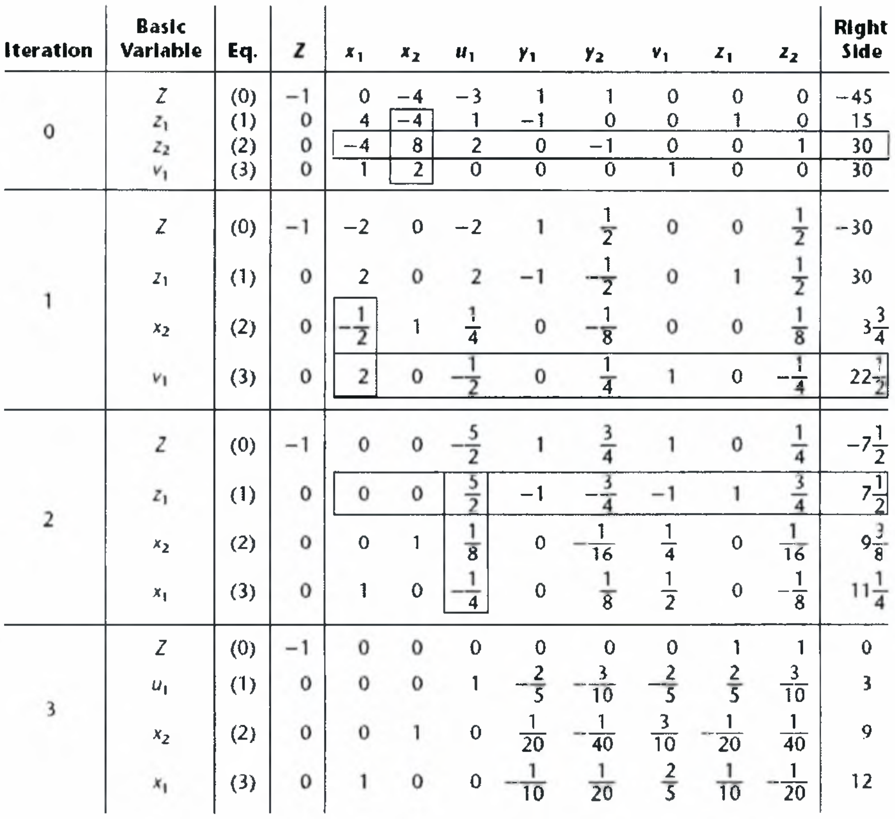
$$\begin{array}{llllllllll} 4x_1 & -4x_2 & +u_1 & -y_1 & & & +z_1 & & = 15 \\ -4x_1 & +8x_2 & +2u_1 & & -y_2 & & & +z_2 & = 30 \\ x_1 & +2x_2 & & & & +v_1 & & & = 30 \end{array}$$

and $x_1 \geq 0 \quad x_2 \geq 0 \;\; u_1 \geq 0 \;\; y_1 \geq 0 \;\; y_2 \geq 0 \;\; v_1 \geq 0$

$z_1 \geq 0 \;\; z_2 \geq 0$

The additional complementarity constraint $x_1y_1 + x_2y_2 + u_1v_1 = 0$ is not included explicitly, because the algorithm automatically enforces this constraint because of the restricted-entry rule. In particular, for each of the three pairs of complementary variables $(x_1, y_1), (x_2, y_2), (u_1, v_1)$ whenever one of the two variables already is a basic variable, the other variable is excluded as a candidate for the entering basic variable. Remember that the only nonzero variables are basic variables. Because the initial set of basic variables for the linear programming problem z_1, z_2, v_1, gives an initial BF solution that satisfies the complementarity constraint, there is no way that this constraint can be violated by any subsequent BF solution.

Table 2.2 shows the results of applying the modified simplex method to this problem. The first simplex tableau exhibits the initial system of equations after converting from minimizing Z to maximizing -Z and algebraically eliminating the initial basic variables from Eq. (0). The three iterations proceed just as for the regular simplex method, except for eliminating certain candidates for the entering basic variable because of the restricted-entry rule. In the first tableau, u_1 is eliminated as a candidate because it's complementary variable (v_1) already is a basic variable (but x_2 would have been chosen anyway because $-4 < -3$).



Iteration	Basic Variable	Eq.	Z	x_1	x_2	u_1	y_1	y_2	v_1	z_1	z_2	Right Side
0	Z	(0)	-1	0	-4	-3	1	1	0	0	0	-45
	z_1	(1)	0	4	$\boxed{-4}$	1	-1	0	0	1	0	15
	z_2	(2)	0	-4	$\boxed{8}$	2	0	-1	0	0	1	30
	v_1	(3)	0	1	$\boxed{2}$	0	0	0	1	0	0	30
1	Z	(0)	-1	-2	0	-2	1	$\frac{1}{2}$	0	0	$\frac{1}{2}$	-30
	z_1	(1)	0	2	0	2	-1	$-\frac{1}{2}$	0	1	$\frac{1}{2}$	30
	x_2	(2)	0	$\boxed{-\frac{1}{2}}$	1	$\frac{1}{4}$	0	$-\frac{1}{8}$	0	0	$\frac{1}{8}$	$3\frac{3}{4}$
	v_1	(3)	0	$\boxed{2}$	0	$-\frac{1}{2}$	0	$\frac{1}{4}$	1	0	$-\frac{1}{4}$	$22\frac{1}{2}$
2	Z	(0)	-1	0	0	$-\frac{5}{2}$	1	$\frac{3}{4}$	1	0	$\frac{1}{4}$	$-7\frac{1}{2}$
	z_1	(1)	0	0	0	$\boxed{\frac{5}{2}}$	-1	$-\frac{3}{4}$	-1	1	$\frac{3}{4}$	$7\frac{1}{2}$
	x_2	(2)	0	0	1	$\boxed{\frac{1}{8}}$	0	$-\frac{1}{16}$	$\frac{1}{4}$	0	$\frac{1}{16}$	$9\frac{3}{8}$
	x_1	(3)	0	1	0	$\boxed{-\frac{1}{4}}$	0	$\frac{1}{8}$	$\frac{1}{2}$	0	$-\frac{1}{8}$	$11\frac{1}{4}$
3	Z	(0)	-1	0	0	0	0	0	0	1	1	0
	u_1	(1)	0	0	0	1	$-\frac{2}{5}$	$-\frac{3}{10}$	$-\frac{2}{5}$	$\frac{2}{5}$	$\frac{3}{10}$	3
	x_2	(2)	0	0	1	0	$\frac{1}{20}$	$-\frac{1}{40}$	$\frac{3}{10}$	$-\frac{1}{20}$	$\frac{1}{40}$	9
	x_1	(3)	0	1	0	0	$-\frac{1}{10}$	$\frac{1}{20}$	$\frac{2}{5}$	$\frac{1}{10}$	$-\frac{1}{20}$	12

Table 2.2 Application of the modified Simplex to the Quadratic Programming Example

In the second tableau, both u_1 and y_2 are eliminated as candidates (because v_1 and x_2 are basic variables), so x_1 automatically is chosen as the only candidate with a negative coefficient in row 0 (whereas the regular simplex method would have permitted choosing either x_1 or u_1 because they are tied for having the largest negative coefficient). In the third tableau, both y_1 and y_2 are eliminated (because x_1 and x_2 are basic variables). However, u_1 is not eliminated because v_1 no longer is a basic variable, so u_1 is chosen as the entering basic variable in the usual way.

The resulting optimal solution for this phase 1 problem is $x_1 = 12$, $x_2 = 9$ and $u_1 = 3$, with the rest of the variables zero. Therefore, the optimal solution for the Quadratic Programming problem (which includes only the x_1 and x_2 variables) is $(x_1, x_2) = (12, 9)$.

2.7 Conclusions

In this chapter, we have considered a family of combinatorial problems known under the name of Knapsack Problems and. We have also studied the foundations of nonlinear programming and focused on one of its major subsectors namely Quadratic Programming .We have introduce the Karush-Kuhn-Tucker (KKT) conditions and showed an indirect application of these conditions to provide a linear transformation of the Quadratic Programming Problem which can be dealt with the modified SIMPLEX method.

Applications for both Knapsack Problems and Quadratic Programming have been presented, denoting why both are of great research interest. As we can easily see the combination of these two major subjects of interest, leads to the Quadratic Knapsack Problem which is the key concept of this thesis.

Chapter 3 Quadratic Knapsack Applications

3.1 Introduction

This chapter consists of the analysis of Quadratic Knapsack Applications. In detail we will study Problems of Portfolio Selection, Quadratic Transportation, Multi-Commodity Network Flow and Matrix Balancing. We supply brief introduction of the problem concepts as well as references of past work. We also consider their Quadratic Knapsack formulations. In the last section of the chapter we present a new formulation, where deviations from the target value are allowed for the Knapsack Constraint applied to an Aircraft-Maintenance Problem.

3.2 Portfolio Selection Problem

Constructing a portfolio of investments is one of the most significant financial decisions facing individuals and institutions. A decision-making process must be developed which identifies the appropriate weight each investment should have within the portfolio. The portfolio must strike what the investor believes to be an acceptable balance between risk and reward. In addition, the costs incurred when setting up a new portfolio or rebalancing an existing portfolio must be included in any realistic analysis. Convex transaction costs, including linear (proportional) transaction costs, piecewise linear transaction costs, and Quadratic transaction costs can be considered. In order to properly reflect the effect of transaction costs, we suggest rescaling the risk term by the funds available after paying the transaction costs.

Essentially, the standard portfolio optimization problem is to identify the optimal allocation of limited resources among a limited set of investments. Optimality is measured using a trade-off between perceived risk and expected return. Expected future returns are based on historical data. Risk is measured by the variance of those historical returns.

When more then one investment is involved, the covariance among individual investments becomes important. In fact, any deviation from perfect positive correlation allows a beneficial diversified portfolio to be constructed. Efficient portfolios are allocations that achieve the highest possible return for a given level of risk. Alternatively, efficient portfolios can be said to minimize the risk for a given level of return. These ideas earned their inventor a Nobel Prize and have gained such wide acceptance that countless references could be cited. The model of portfolio selection is originally presented in Markowitz **[10]** and is as follows. Assume:

(a) n securities

(b) an initial sum of money to be invested

(c) the beginning of a holding period

(d) the end of the holding period

Let $x_1,...,x_n$ be *investment proportion* weights. The x_i are the proportions of the initial sum invested in the n securities to form a portfolio at the beginning of the holding period. Unless restricted to the contrary, an x_i can take on any value. Nevertheless, all x_i must sum to one. An $x_i < 0$ means that security i is sold short with the cash generated then providing additional money to be invested in the other securities. An $x_i > 1$ is possible.

Assume a two-stock portfolio and an initial sum of \$100. If security 1 is sold short to the extent of \$40, then because all weights must sum to one, the \$40 plus the initial sum are invested in security 2 in which case $x_1 = -0.4$ and $x_1 = 1.4$.

Let r_i be the random variable for the percent return realized on security i between the beginning of the holding period and the end of the holding period. Let r_p be the random variable for the percent return realized on a portfolio between the beginning of the holding period and the end of the holding period, where

$$r_p = \sum_{i=1}^{n} r_i x_i$$

In this way, r_p is a function of both the r_i and the x_i. Since the r_i are not known until the end of the holding period, but the x_i must be chosen at the beginning of the period, attempting to maximize r_p via the above equation is a stochastic optimization problem. With solutions of a stochastic optimization problems not well defined, a decision is required on how to proceed.

Since an investor can never know at the beginning of the holding period the value of r_p to be realized at the end of the holding period, the investor is in a quandary. Ideally, an investor would like to position his initial sum to maximize his chances of reaping a high value of r_p while at the same time minimizing his exposure to disconcertingly low values of r_p. Assuming that all r_i are from distributions whose means μ_i, variances σ_{ii} and covariances σ_{ij} are known, Markowitz's mean-variance solution procedure, which has come to form the foundation of what we know of today as "modern portfolio analysis", is to proceed with the bi-criterion program

$$\min\left\{\sum_{i=1}^{n}\sum_{j=1}^{n}x_i\sigma_{ij}x_j=\sigma_p^2\right\}$$

$$\max\left\{\sum_{i=1}^{n}\mu_jx_j=\mu_p\right\}$$

$$\text{s.t.}\qquad \sum_{i=1}^{n}x_i=1$$

$l_i \le x_i \le u_i$ where σ_p^2 is the variance of r_p and μ_p is expected value. Let E_0 be the minimum expected portfolio return. The problem can take the following form

$$\min\left\{\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{Q}\mathbf{x}=\sum_{i=1}^{n}\sum_{j=1}^{n}x_i\sigma_{ij}x_j=\sigma_p^2\right\}$$

$$\boldsymbol{\mu}^T\mathbf{x}=\sum_{i=1}^{n}\mu_jx_j\ge E_0$$

$$\text{s.t.}\qquad \sum_{i=1}^{n}x_i=1$$

$$l_i \le x_i \le u_i$$

By varying the parameter E_0 and solving multiple instances of the problem, the set of efficient portfolios can be generated. This set, visualized in a risk/return plot, is called the efficient frontier. An investor may decide where along the efficient frontier (s)he finds an acceptable balance between risk and reward.

3.3 Quadratic Transportation Problems

The **Linear Transportation Problem (L.T.P.)** can be described as a minimum-cost flow problem over a network depicted in **Fig 3.2** .This network includes I supply and J demand nodes connected by direct links. Hitchcock's formulation of the transportation problem is

$$Min\ z(\mathbf{x}) = \sum_{i=1}^{I}\sum_{j=1}^{J} c_{ij}x_{ij}$$

subject to $$\sum_{j=1}^{J} x_{ij} = O_i \qquad \forall i = 1,2,\dots,I$$

$$\sum_{i=1}^{I} x_{ij} = D_j \qquad \forall j = 1,2,\dots,J$$

$$x_{ij} \geq 0 \quad , \quad \forall i,j$$

Assume further that the total supply equals the total demand, that is

$$\sum_i O_i = \sum_j D_j$$

where x_{ij} is the amount of movement from place i to j, c_{ij} is the given transport cost and O and D are the supplies and demands.

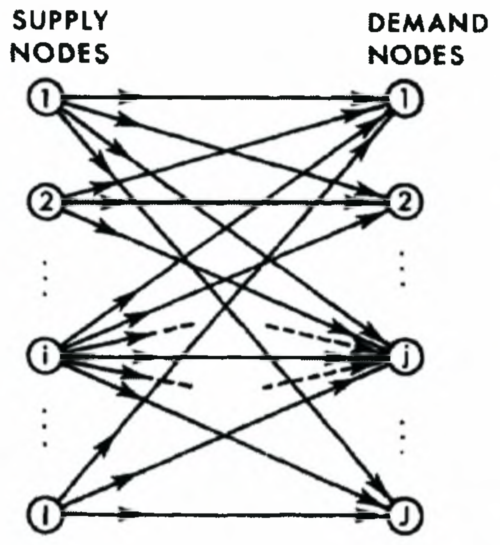


Figure 3.1 Linear Transportation Problem

In addition to the known marginal totals O_i and D_j, the transport cost quantities c_{ij} are also given. The **Quadratic Transportation Problem (Q.T.P.)** is an optimization problem defined as follows :

$$Min\ z(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{I}\sum_{j=1}^{J} c_{ij}\left(x_{ij}\right)^2 + \sum_{i=1}^{I}\sum_{j=1}^{J} d_{ij}\left(x_{ij}\right)$$

subject to $\quad \sum_{j=1}^{J} x_{ij} = O_i \qquad \forall i = 1,2,\ldots,I$

$$\sum_{i=1}^{I} x_{ij} = D_j \qquad \forall j = 1,2,\ldots,J$$

$$x_{ij} \geq 0 \qquad \forall i,j$$

$$\sum_i O_i = \sum_j D_j$$

where

x_{ij} is the amount of movement from place i to j

c_{ij} is the per unit transport cost

d_{ij} is the per unit depreciation cost (wear and tear or damage cost)

and O and D are the supplies and demands.

Properties that distinguish the solution to the Q.T.P. from that of the L.T.P. are that

(a) The x_{ij} are on average smaller numbers. This is forced by the Quadratic term in the objective function.

(b) The number of non-zero x_{ij} will exceed $I + D + 1$ and will approach $I \cdot D$.

(c) The x_{ij} are generally not integers.

Properties (a) and b) are more in accord with empirical spatial interaction tables than are the solutions to the L.T.P. This is expected because commodity flows are rendered more reliable by a diversity of sources, urban traffic is diverted to avoid congestion, and migration patterns are rendered diffuse due to information inadequacies. Spatial allocation models that use the L.T.P. thus yield results that are less realistic than can be obtained through the use of the Q.T.P. solution. Property c) of the L.T.P. is desirable, however, and suggests investigation of an Integer Q.T.P.

3.4 Multi-commodity Network Flows

In the classical transportation problem the cost of transportation is directly proportional to the number of units of the commodity transported. But in real world situations when a commodity is transported, a fixed cost is incurred in the objective function. The fixed cost may represent the cost of renting a vehicle, landing fees in an airport, set up costs for machines in a manufacturing environment etc.

The three dimensional fixed charge bi-criterion indefinite Quadratic transportation problem, can be used to formulate the real-world problem.

Suppose $i = 1, 2, \ldots, m$ are the origins

$j = 1, 2, \ldots, n$ are the destinations and

$k = 1, 2, \ldots, p$ are the various types of commodities to be transported in a three dimensional transportation problem. Let

$x_{ijk} =$ the amount of kth type of commodity transported form the ith origin to the jth destination

$c_{ijk} =$ the variable cost per unit amount of kth type of commodity transported form the ith origin to the jth destination, which is independent of the amount of the commodity transported, so long as $x_{ijk} > 0$.

$d_{ijk} =$ the per unit depreciation cost (wear and tear or damaged cost) of kth type of commodity transported form the ith origin to the jth destination, which is independent of the amount of the commodity transported, so long as $x_{ijk} > 0$

$A_{jk}=$ the total quantity of kth type of the commodity received by jth destination from all the sources.

$B_{ki}=$ the total quantity of kth type of the commodity available at the ith origin to be supplied to all destinations.

$E_{ij}=$ the total quantity of all types of commodities to be supplied from the ith origin to the jth destination.

$F_{ik}=$ the fixed cost associated with origin i and the kth type of commodity. We define F_{ik} according to the amount supplied as

$$F_{ik}=\sum_{j=1}^{n}F_{ijk}d_{ijk},\ i=1,2,...,m \quad k=1,2,...,p$$

where $d_{ijk}=\begin{cases}1 & if\ x_{ijk}>0\\ 0 & if\ x_{ijk}=0\end{cases},\ i=1,2,...,m,\ j=1,2,...,n,k=1,2,...,p$

Then the three dimensional indefinite Quadratic transportation problem is defined as

$$Min\ z(\mathbf{x})=\left(\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{p}c_{ijk}x_{ijk}\right)\left(\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{p}d_{ijk}x_{ijk}\right)+\sum_{i=1}^{m}\sum_{k=1}^{p}F_{ik}$$

subject to

$$\sum_{i=1}^{m}x_{ijk}=A_{jk} \quad ,j=1,2,...,n,k=1,2,...,p$$

$$\sum_{j=1}^{n} x_{ijk} = B_{ki} \qquad ,k = 1,2,\dots,p\,,i = 1,2,\dots,m\,,$$

$$\sum_{k=1}^{p} x_{ijk} = E_{ij} \qquad ,i = 1,2,\dots,m\,,\; j = 1,2,\dots,n$$

$$x_{ijk} \geq 0 \qquad ,i = 1,2,\dots,m\,,\; j = 1,2,\dots,n\,,k = 1,2,\dots,p$$

Also

$$\sum_{j=1}^{n} A_{jk} = \sum_{k=1}^{p} B_{ki}\,,k = 1,2,\dots,p \qquad \text{(i)}$$

$$\sum_{k=1}^{p} B_{ki} = \sum_{j=1}^{n} E_{ij}\,,i = 1,2,\dots,m \qquad \text{(ii)}$$

$$\sum_{i=1}^{m} E_{ij} = \sum_{k=1}^{p} A_{jk}\,,\; j = 1,2,\dots,n \qquad \text{(iii)}$$

$$\sum_{j=1}^{n}\sum_{k=1}^{p} A_{jk} = \sum_{k=1}^{p}\sum_{i=1}^{m} B_{ki} = \sum_{i=1}^{m}\sum_{j=1}^{n} E_{ij} \qquad \text{(iv)}$$

Here, there are m origins, n destinations and p types of commodities to be transported.

(i) implies kth type of commodity received by all destinations = kth type of commodity supplied from all origins,

(ii) implies different types of commodities supplied by the ith source = amount of commodities received by all destinations from the ith source

(iii) implies amount of commodities supplied from all sources to j th destination = different types of commodities received by the j th destination,

(iv) implies amount of commodities received by all destinations of different types of commodities = amount of commodities supplied from all origins to all destinations = amount of different types of commodities supplied from all origins.

Note: (i) to (iv) indicates that the transportation problem considered is a balanced transportation problem.

3.5 Matrix Balancing

The problem of adjusting the elements of a matrix so that they satisfy certain consistency requirements but still remain 'close' to the original matrix is generically referred to as matrix balancing. Matrix balancing problems arise in a wide range of practical contexts that include accounting, transportation, and demographics. These and several other applications are reviewed in an excellent overview by Schneider and Zenios **[28]**.

In a typical matrix balancing problem, we have a matrix that estimates certain quantities of interest, but these estimates do not satisfy consistency requirements that the actual values are known to satisfy. An example might be estimating the elements of a transition probability matrix which we know to be doubly stochastic. Consistency with the doubly stochastic property requires that the rows and columns sum to one. The doubly stochastic matrix is an example of one of two types of matrix balancing problems discussed by Schneider and Zenios **[28]**. They are adjusting the elements of a matrix so that the row and column sums equal certain prescribed values; adjusting the elements of a square matrix so that its row and column sums are equal to each other, but not necessarily to any prescribed values.

The conditions imposed on the row and column sums are called balance conditions, and a matrix that satisfies the balance conditions is said to be balanced. In the applications considered by Schneider and Zenios **[28]**, the balance conditions relate only to row and column sums. More generally, the balance conditions can be restrictions on the sums of various combinations of matrix elements. (See, for example, Censor and Zenios **[29]**.) The fair representation problem considered by Balinski and Demange **[30]** is one example. In the most general case, the balance conditions can be any set of linear restrictions on the matrix entries.

For a particular set of balance conditions there may be a large number of balanced matrices, but in matrix balancing, we seek a balanced matrix that is close to the original matrix.

3.5.1 A brief review of matrix balancing

We present a standard matrix balancing formulation for producing matrices with prescribed row and column sums. This formulation appears in **[28]**. Suppose that we are given an $n \times n$ nonnegative matrix **M** and positive vectors **s** and **d**, both in $\Re^n$, that provide target row and column sums. The associated matrix balancing problem can be written

$$Min\ z(\mathbf{x}) = \sum_{i=1}^{I}\sum_{j=1}^{J} f_{ij}x_{ij}$$

subject to $\displaystyle\sum_{j=1}^{J} x_{ij} = s_j \qquad \forall i = 1,2,\ldots,I$

$$\sum_{i=1}^{I} x_{ij} = d_j \qquad \forall j = 1,2,\dots,J$$

$$x_{ij} \geq 0 \qquad \forall i,j$$

$$x_{ij} > 0 \text{ only if } m_{ij} > 0$$

The constraints in this model can be viewed as the flow-balance equations in an associated transportation problem.

$$\begin{array}{cc} & r: \\ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 0 & 6 & 7 \\ 6 & 1 & 0 & 7 \\ 1 & 1 & 2 & 0 \end{bmatrix} & \begin{array}{c} 6 \\ 17 \\ 18 \\ 4 \end{array} \\ s: \quad 6 \quad 17 \quad 18 \quad 4 & \end{array}$$

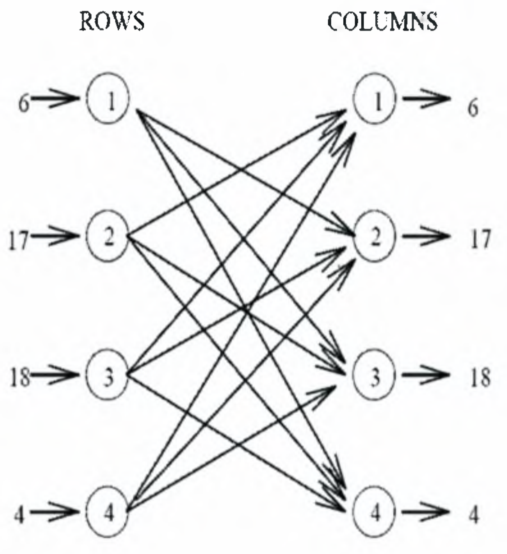


Figure 3.2 A matrix and it's associated transportation network

Figure 3.3 provides an example of a small matrix and its representation as a transportation network. There is one left-hand node associated with each row of M and one right-hand node associated with each column of M. There is a link from left-hand node i to right-hand node j whenever the corresponding matrix element m_{ij} of M is nonzero. Left-hand node i has supply s_i, while right-hand node i has demand d_i. To complete the network flow description of the problem, we associate a cost f_{ij} of sending x_{ij} units of flow on the link from i to j and try to minimize the total cost of satisfying demands. The optimal flows x_{ij} provide the new balanced matrix.

The objective function employed in matrix balancing is typically separable, nonlinear, and convex. The role of the objective is clearly to penalize deviations from the original matrix. Nonlinear objectives are attractive because they promote balance among the deviations by penalizing large deviations disproportionately more. Schneider and Zenios **[28]** note that Quadratic and entropy penalty functions are the ones that are typically used in practice.

Quadratic objective functions that minimize the (weighted) sum of squared deviations from the target matrix have been more widely studied. For problem [MB], we obtain the Quadratic penalty objective by letting

$$f_{ij}\left(x_{ij}\right) = w_{ij}\left(x_{ij} - m_{ij}\right)^2$$

where the w_{ij} 's are nonnegative weights. The resulting problem has a separable Quadratic objective and transportation constraints, whence we can see that the problem simplifies to the Quadratic Transportation Problem already referenced.

3.6 Aircraft Maintenance

The problem that we present here arises as part of an operations management problem in a typical Combat Wing of Military Aircrafts.

At the beginning of each planning horizon, the wing command issues the flight requirements for each period. These requirements determine the total time that all the aircraft should fly during each time period. Separate requirements are issued for each type of aircraft, because different aircraft types have different flight capabilities and maintenance needs. For this reason, the model introduced in this section can be applied repeatedly until all plans have been issued, if more than one aircraft types are involved. The requirements issued by the wing command contain target values from which only small deviations are permitted.

For each specific aircraft, we define its residual flight time as the total remaining time that the aircraft can fly until it has to undergo a maintenance check. The residual flight time of an aircraft is positive if and only if this aircraft is available to fly. The total residual flight time of the wing is equal to the sum of the residual flight times of all squadrons. Clearly, there exist many possible combinations of individual aircraft residual flight times that can result in the same total squadron or wing residual flight time. Similarly, we define the residual maintenance time of an aircraft as the total remaining time that the aircraft needs in order to complete its maintenance check before it can be available to fly again.

For the maintenance needs of the wing, there exists a maintenance station that is responsible for providing maintenance services to the aircraft of the wing. This station has certain space and time capacity capabilities. Given the flight requirements for each squadron and the physical constraints that stem from the capacity of the maintenance station, the objective is to issue a flight and maintenance plan for each individual aircraft so that some appropriate measure of effectiveness is optimized.

Consider the 2-dimensional graph shown in **Fig 3.4**. The vertical axis represents residual flight time measured in some appropriate units, and the horizontal axis represents the indices of the aircraft in increasing order of their residual flight times, 1 being the index of the aircraft with the smallest and N being the index of the aircraft with the largest residual time (N is the total number of aircraft).

Consider also the line segment connecting the origin and the residual flight times point with coordinates $(N, y_{\max})$, where $y_{\max}$ is the maximum time that an aircraft can fly between two consecutive maintenance checks. By mapping each aircraft on this graph, we can have a picture of the total availability of the squadron or the wing, whichever of the two the graph refers to.

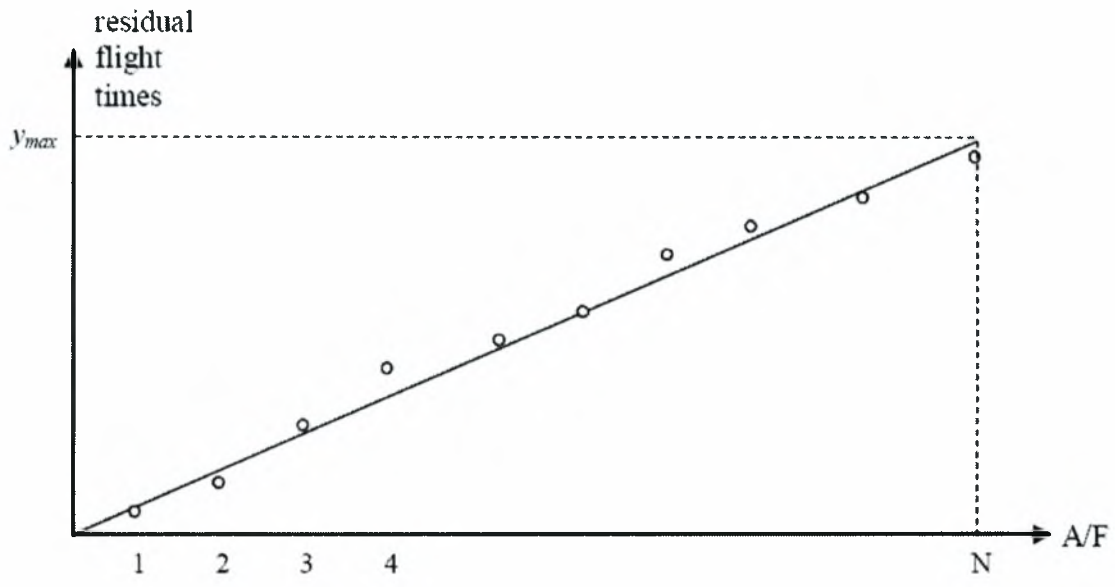


Figure 3.3 Visual representation of aircraft residual flight times

To describe the smoothness of the distribution of the total residual flight time of all aircraft we use a "total deviation index". This index is equal to the sum of the vertical distances (deviations) of each point mapping a single aircraft from the line segment that connects the origin with point $(N, y_{\max})$. The smaller this sum is, the smoother the distribution of the total residual flight time. Ideally, the total deviation index is equal to zero, in which case all points lie on the line segment. When issuing the individual aircraft plans, the intention is to keep each point as close to the line segment as possible, so that these deviations remain small.

The intuition behind the utilization of the graph described above is straightforward. By providing a wide range of different residual aircraft flight times, we can establish a smooth sequence that determines the order in which the aircraft should visit the maintenance station. This in turn prevents bottlenecks in the maintenance station and ensures a smooth utilization of the maintenance station. More importantly, it ensures a fairly constant level of aircraft availability.

As already noted, the flight time availability is equal to the sum of all aircraft residual flight times, but there are many residual flight time combinations that can result in the same total availability. To comprehend this better, consider a problem with four aircraft, each of which can fly a maximum of 120 hours between two consecutive maintenance checks. For a total flight time availability of 300 hours, a possible combination of the residual flight times is 30-30-120-120. Another one is 30-60-90-120. For the technique described above, the second distribution is preferable, because it ensures a smooth rotation of the aircraft, i.e. a smooth utilization of the maintenance center and a fairly constant level of aircraft availability. From the maintenance point of view smoothing the rotation of the aircraft is the most appropriate measure of effectiveness.

3.6.1 Problem Formulation

In this section, we present the mathematical model that we developed for the problem described above. We use the following notation:

Decision Variables:

x_i : flight time of aircraft i during planning horizon,

Parameters:

S	: required total flight time during planning horizon
y_i	: residual flight time of aircraft i at the beginning of planning horizon,
$X_{\max}$	: maximum time an aircraft can fly during planning horizon ,
$Y_{\min}$	: minimum residual flight time of an available aircraft,

L , U : real numbers denoting the maximum deviation from the value of S that can be tolerated $(U > L)$,

$s = \frac{y_{\max}}{N}$: the slope of the deviation line where

$y_{\max}$	: maximum residual flight time of an available aircraft,
N	: total number of aircrafts available for flight

Then, the referenced problem can be formulated as follows:

$$Min\ z=\sum_i \left(\left(y_i - i \cdot s\right) - x_i\right)^2$$

$$s.t. \qquad L \cdot S \le \sum_i x_i \le U \cdot S$$

$$0 \le x_i \le X_{\max}$$

$$y_i - x_i = y_i' \ge Y_{\min} \Leftrightarrow x_i \le -Y_{\min} + y_i \quad i \in N = \{1,2,...,N\}$$

and can take the following form

$$Min\ z = \sum_i \left((y_i - i \cdot s) - x_i \right)^2$$

$$s.t. \quad L \cdot S \le \sum_i x_i \le U \cdot S$$

$$0 \le x_i \le X_{left}(i) = \min(X_{\max}\ , y_i - Y_{\min})$$

$$U > L, i \in N = \{1,2,...,N\}$$

The objective function minimizes the sum of squares of all deviations from the line.

The first constraint set (Knapsack constraint) ensures that the flight requirements are met. Variables L and U define an interval $[L \cdot S, U \cdot S]$, in which the actual flight time for the planning horizon should lie. For example when $L = 0.95$ and $U = 1.05$ a 5% deviation from the flight requirements is permitted.

The second constraint set (box constraints) ensures that the residual flight time of an aircraft cannot exceed the maximum value neither the upper bound of the maximum time it can fly during the planning horizon.

3.6.2 General Form transformation

Let $(y_i - i \cdot s) - x_i = x_i' \Leftrightarrow x_i = (y_i - i \cdot s) - x_i'$

Then $\sum_i x_i = \sum_i (y_i - i\cdot s) - \sum_i x_i'$

We make the necessary transformations.

$$z=\sum_i \left((y_i - i\cdot s) - x_i\right)^2 = \sum_i (x_i')^2$$

Then $\left.\begin{array}{l}\sum_i x_i - U\cdot S \le 0\\ L\cdot S - \sum_i x_i \le 0\end{array}\right|\left.\begin{array}{l}\sum_i (y_i - i\cdot s) - \sum_i x_i' - U\cdot S \le 0\\ L\cdot S - \sum_i (y_i - i\cdot s) + \sum_i x_i' \le 0\end{array}\right|$

$$\Leftrightarrow \left.\begin{array}{l}\sum_i x_i' \ge \sum_i (y_i - i\cdot s) - U\cdot S\\ \sum_i x_i' \le \sum_i (y_i - i\cdot s) - L\cdot S\end{array}\right|\left.\begin{array}{l}\sum_i x_i' \ge L'\\ \sum_i x_i' \le U'\end{array}\right\} U' > L' \text{ where } \begin{array}{l}L' = \sum_i (y_i - i\cdot s) - U\cdot S\\ U' = \sum_i (y_i - i\cdot s) - L\cdot S\end{array}$$

Last $\quad 0 \le x_i \le X_{left}(i) \Leftrightarrow 0 \le (y_i - i\cdot s) - x_i' \le X_{left}(i)$

$$\Leftrightarrow (y_i - i\cdot s) - X_{left}(i) \le x_i' \le (y_i - i\cdot s) \Leftrightarrow a_i \le x_i' \le b_i \text{ where } \begin{array}{l}a_i = (y_i - i\cdot s) - X_{left}(i)\\ b_i = (y_i - i\cdot s)\end{array}$$

The problem takes the following form

$$Min\ z=\sum_i (x_i)^2$$

$$s.t. \quad \left.\begin{array}{l}\sum_i x_i \ge L\\ \sum_i x_i \le U\end{array}\right\} U > L$$

$$a_i \leq x_i \leq b_i$$

Which is the general case of the **box constrained Quadratic Knapsack with upper bounds**.

3.7 Conclusions

In this chapter, we studied Quadratic Knapsack Applications including Problems of Portfolio Selection and Rebalancing, Quadratic Transportation, Multi-Commodity Network Flow and Matrix Balancing. All these problems can be formulated in accordance with the standard Quadratic Knapsack Formulation. All formulations have in common that there exists one or more tight Knapsack constraint.

We find interest in investigating the case where deviations from the target value are allowed for the Knapsack Constraint, which is exactly what we have seen in the Aircraft-Maintenance Application.

Chapter 4 Box Constrained Quadratic Knapsack Problem with Upper Bounds

4.1 Introduction

In this chapter we focus on the special case of Quadratic Knapsack Problems, where deviations from the target value are allowed for the Knapsack constraint. Because of this deviation allowance algorithms and known techniques for the standard Quadratic Knapsack formulation cannot be directly applied. That is why we use the KKT Conditions in order to characterize the optimal solution to the problem. After that we prove global optimality of the solution. We introduce an algorithm for the solution of the problem and then focus on optimality and complexity issues.

The formulation used in this chapter is fitted for the need of the Aircraft maintenance application, but in the end of the chapter we generalize the concept.

4.2 Applying the KKT Conditions.

As we can see from **Section 3.6** the Aircraft-Maintenance problem can be formulated as

$$Min\ z = \sum_i \left((y_i - i \cdot s) - x_i \right)^2$$

$$s.t. \quad \left. \begin{array}{l} \sum_i x_i - U \cdot S \leq 0 \\ L \cdot S - \sum_i x_i \leq 0 \end{array} \right\} U > L$$

$$0 \leq x_i \leq X_{left}(i) = \min(X_{\max}\ , y_i - Y_{\min})$$

The Lagrangian for the problem is

$$L(x,\lambda_1,\lambda_2,u,v)=\sum_i\left((y_i-i\cdot s)-x_i\right)^2+\lambda_1\cdot\left(\sum_i x_i-U\cdot S\right)+\lambda_2\cdot\left(L\cdot S-\sum_i x_i\right)+\sum v_i\cdot\left(x_i-X_{left}(i)\right)$$

The necessary and sufficient Karush-Kuhn Tucker (KKT) conditions are:

$$-2\left((y_i-i\cdot s)-x_i\right)+\lambda_1-\lambda_2+v_i\ \geq 0$$

$$v_i\cdot\left(x_i-X_{left}(i)\right)=0 \qquad v_i\in\Re^+$$

$$\lambda_1\cdot\left(\sum_i x_i-U\cdot S\right)=0\quad \lambda_1\in\Re^+$$

$$\lambda_2\cdot\left(L\cdot S-\sum_i x_i\right)=0\quad \lambda_2\in\Re^+$$

$$\sum_i x_i-U\cdot S\leq 0$$

$$L\cdot S-\sum_i x_i\leq 0$$

$$0\leq x_i\leq X_{left}(i)$$

Theorem 4.1 (Characterization of the optimal solution of the Problem).

A feasible solution $\mathbf{x}^* = (x_i^*)$ is optimal solution for the problem if and only if there exist $\lambda_1, \lambda_2 \in \Re^+ : \lambda_1 \cdot \lambda_2 = 0$ such that for $\lambda = \lambda_1 - \lambda_2$ the following hold:

$$x_i^* = 0 \qquad , i \in N_0^\lambda = \{ i \in N : \lambda \geq 2(y_i - i \cdot s) \}$$

$$x_i^* = (y_i - i \cdot s) - \frac{\lambda}{2} \qquad , i \in N^\lambda = \{ i \in N : 2((y_i - i \cdot s) - X_{left}(i)) < \lambda < 2(y_i - i \cdot s) \}$$

$$x_i^* = X_{left}(i) \qquad , i \in N_{X_{left}}^\lambda = \{ i \in N : \lambda \leq 2((y_i - i \cdot s) - X_{left}(i)) \}$$

<u>**Proof**</u>

Necessity

Let $\mathbf{x}^* = (x_i^*)$ be the optimal solution.

Let $\lambda_1 > 0$ and $\lambda_2 > 0$

From the KKT conditions we have

$$\left. \begin{array}{r} \sum_i x_i - U \cdot S = 0 \\ L \cdot S - \sum_i x_i = 0 \end{array} \right\} \Leftrightarrow U = L \text{ Not feasible because } U > L$$

We conclude that an optimal solution of the problem exists if and only if $\lambda_1 \cdot \lambda_2 = 0$

We have the following cases:

Case 1. $\sum_i x_i = L \cdot S$

so $\lambda_1 = 0$ and $\lambda_2 \geq 0$ that is $\lambda = -\lambda_2$

The KKT conditions take the following form

$$-2\left(\left(y_i - i \cdot s\right) - x_i\right) - \lambda_2 + v_i \ \geq 0$$

$$v_i \cdot \left(x_i - b_i\right) = 0 \qquad\qquad v_i \in \Re^+$$

$$\sum_i x_i < U \cdot S$$

$$\sum_i x_i = L \cdot S$$

$$0 \leq x_i \leq X_{left}(i)$$

a) For $x_i^* = 0$ we have that $v_i = 0$

So $-2\left(y_i - i \cdot s\right) \geq \lambda_2 = -\lambda$

$\Leftrightarrow \lambda \geq 2\left(y_i - i \cdot s\right)$

b) For $x_i^* = X_{left}(i)$ we have that $v_i \geq 0$

So $-2\left(\left(y_i - i \cdot s\right) - X_{left}(i)\right) = \lambda_2 - v_i \leq \lambda_2 = -\lambda$

$$\Leftrightarrow \lambda \le 2\left(\left(y_i - i\cdot s\right) - X_{left}(i)\right)$$

c) For $0 < x_i^* < X_{left}(i)$ we have that $v_i = 0$

So $-2\left(\left(y_i - i\cdot s\right) - x_i^*\right) = \lambda_2 = -\lambda \Leftrightarrow y_i - i\cdot s - x_i^* = \dfrac{\lambda}{2}$

$$\Leftrightarrow x_i^* = y_i - i\cdot s - \frac{\lambda}{2}$$

However $0 < x_i^* < X_{left}(i)$

$$\Leftrightarrow 0 < y_i - i\cdot s - \frac{\lambda}{2} < X_{left}(i)$$

$$\Leftrightarrow y_i - i\cdot s - X_{left}(i) < \frac{\lambda}{2} < y_i - i\cdot s$$

$$\Leftrightarrow 2\left(y_i - i\cdot s - X_{left}(i)\right) < \lambda < 2\left(y_i - i\cdot s\right)$$

Case 2. $\sum_i x_i = U\cdot S$

So $\lambda_1 > 0$ and $\lambda_2 = 0$ that is $\lambda = \lambda_1$

The KKT conditions take the following form

$$-2\left(\left(y_i - i\cdot s\right) - x_i\right) + \lambda_1 + v_i \ge 0$$

$$v_i \cdot \left(x_i - b_i\right) = 0 \qquad v_i \in \Re^+$$

$$\sum_i x_i = U \cdot s$$

$$\sum_i x_i > L \cdot s$$

$$0 \le x_i \le X_{left}(i)$$

a) For $x_i^* = 0$ we have that $v_i = 0$

So $-2(y_i - i \cdot s) \ge -\lambda_1 = -\lambda$

$\Leftrightarrow \lambda \ge 2(y_i - i \cdot s)$

b) For $x_i^* = X_{left}(i)$ we have that $v_i \ge 0$

So $-2((y_i - i \cdot s) - X_{left}(i)) = -\lambda_1 - v_i \le -\lambda_1 = -\lambda$

$\Leftrightarrow \lambda \le 2((y_i - i \cdot s) - X_{left}(i))$

c) For $0 < x_i^* < X_{left}(i)$ we have that $v_i = 0$

So $-2((y_i - i \cdot s) - x_i^*) = -\lambda_1 = -\lambda \Leftrightarrow y_i - i \cdot s - x_i^* = \dfrac{\lambda}{2}$

$\Leftrightarrow x_i^* = y_i - i \cdot s - \dfrac{\lambda}{2}$

However $0 < x_i^* < X_{left}(i)$

$$\Leftrightarrow 0 < y_i - i \cdot s - \frac{\lambda}{2} < X_{left}(i)$$

$$\Leftrightarrow y_i - i \cdot s - X_{left}(i) < \frac{\lambda}{2} < y_i - i \cdot s$$

$$\Leftrightarrow 2\left(y_i - i \cdot s - X_{left}(i)\right) < \lambda < 2\left(y_i - i \cdot s\right)$$

Case 3. $L \cdot S < \sum_i x_i < U \cdot S$

So $\lambda_1 = 0$ and $\lambda_2 = 0$ that is $\lambda = 0$

The KKT conditions take the following form

$-2\left(\left(y_i - i \cdot s\right) - x_i\right) + v_i \ \geq 0$

$v_i \cdot \left(x_i - b_i\right) = 0 \qquad\qquad v_i \in \Re^+$

$\sum_i x_i \leq U \cdot s$

$\sum_i x_i \geq L \cdot s$

$0 \leq x_i \leq X_{left}(i)$

a) For $x_i^* = 0$ we have that $v_i = 0$

So $-2\left(y_i - i \cdot s\right) \geq 0 = -\lambda$

$\Leftrightarrow \lambda \geq 2(y_i - i \cdot s)$

b) For $x_i^* = X_{left}(i)$ we have that $v_i \geq 0$

So $-2\left((y_i - i \cdot s) - X_{left}(i)\right) = -v_i \leq 0 = -\lambda$

$\Leftrightarrow \lambda \leq 2\left((y_i - i \cdot s) - X_{left}(i)\right)$

c) For $0 < x_i^* < X_{left}(i)$ we have that $v_i = 0$

So $-2\left((y_i - i \cdot s) - x_i^*\right) = 0 = -\lambda \Leftrightarrow y_i - i \cdot s - x_i^* = \dfrac{\lambda}{2}$

$\Leftrightarrow x_i^* = y_i - i \cdot s - \dfrac{\lambda}{2}$

However $0 < x_i^* < X_{left}(i)$

$\Leftrightarrow 0 < y_i - i \cdot s - \dfrac{\lambda}{2} < X_{left}(i)$

$\Leftrightarrow y_i - i \cdot s - X_{left}(i) < \dfrac{\lambda}{2} < y_i - i \cdot s$

$\Leftrightarrow 2\left(y_i - i \cdot s - X_{left}(i)\right) < \lambda < 2(y_i - i \cdot s)$

Sufficiency

On the contrary let x_i^* satisfy the following

$$x_i^* = 0 \qquad , i \in N_0^\lambda = \{ i \in N : \lambda \geq 2(y_i - i \cdot s) \}$$

$$x_i^* = (y_i - i \cdot s) - \frac{\lambda}{2} \qquad , i \in N^\lambda = \{ i \in N : 2((y_i - i \cdot s) - X_{left}(i)) < \lambda < 2(y_i - i \cdot s) \}$$

$$x_i^* = X_{left}(i) \qquad , i \in N_{X_{left}}^\lambda = \{ i \in N : \lambda \leq 2((y_i - i \cdot s) - X_{left}(i)) \}$$

where $\lambda = \lambda_1 - \lambda_2$ with $\lambda_1, \lambda_2 \in \Re^+ : \lambda_1 \cdot \lambda_2 = 0$

We have the following cases

Case 1. $\lambda_1 = 0$ and $\lambda_2 > 0$ Then $\lambda = -\lambda_2 < 0$

Moreover for $i \in N^\lambda$ $2(y_i - i \cdot s) - x_i^* = \lambda < 0$

Let

(a) $\lambda = 2(y_i - i \cdot s) - x_i^*$ taken from the solution of the equation

$$\sum_{i \in N_0^\lambda} 0 + \sum_{i \in N^\lambda} X_{left}(i) + \sum_{i \in N_{X_{left}}^\lambda} \left((y_i - i \cdot s) - \frac{\lambda}{2} \right) = L \cdot S$$

(b) $u_i = v_i = 0$ για $i \in N^{\lambda}$

(c) $u_i = 2(y_i - i \cdot s) - \lambda \geq 0$ $v_i = 0$ για $i \in N_0^{\lambda}$

(d) $u_i = 0$ $v_i = 2((y_i - i \cdot s) - X_{left}(i)) - \lambda \geq 0$ για $i \in N_{X_{left}}^{\lambda}$

We can see that the KKT conditions are satisfied.

Case 2. $\lambda_1 > 0$ and $\lambda_2 = 0$ Then $\lambda = \lambda_1 > 0$

Moreover for $i \in N^{\lambda}$ $2(y_i - i \cdot s) - x_i^* = \lambda > 0$

Let

(a) $\lambda = 2(y_i - i \cdot s) - x_i^*$ taken from the solution of the equation

$$\sum_{i \in N_0^{\lambda}} 0 + \sum_{i \in N^{\lambda}} X_{left}(i) + \sum_{i \in N_{X_{left}}^{\lambda}} \left((y_i - i \cdot s) - \frac{\lambda}{2}\right) = U \cdot S$$

(b) $u_i = v_i = 0$ για $i \in N^{\lambda}$

(c) $u_i = 2(y_i - i \cdot s) - \lambda \geq 0$ $v_i = 0$ για $i \in N_0^{\lambda}$

(d) $u_i = 0$ $v_i = 2((y_i - i \cdot s) - X_{left}(i)) - \lambda \geq 0$ για $i \in N_{X_{left}}^{\lambda}$

We can see that the KKT conditions are satisfied.

Case 3. $\lambda_1 = 0$ and $\lambda_2 = 0$ Then $\lambda = 0$

Moreover for $i \in N^{\lambda=0}$ $2(y_i - i \cdot s) - x_i^* = \lambda = 0$

Let

(a) $\lambda = 2(y_i - i \cdot s) - x_i^* = 0$

(b) $u_i = v_i = 0$ για $i \in N^{\lambda=0}$

(c) $u_i = 2(y_i - i \cdot s) - \lambda \geq 0$ $v_i = 0$ για $i \in N_0^{\lambda=0}$

(d) $u_i = 0$ $v_i = 2((y_i - i \cdot s) - X_{left}(i)) - \lambda \geq 0$ για $i \in N_{X_{left}}^{\lambda=0}$

The KKT conditions are satisfied once again.

Remarks

Global optimality is proven by convexity. In detail we can see that the Hessian $\nabla^2(f(x))$ of our objective function is positive semidefinite, thus our objective function is convex and by **[43]** our optimal solution is global minimum..

We can see that the Lagrange multipliers λ_1 and λ_2 take zero value only if the Knapsack constraint $\sum_i x_i$ equals the value $U \cdot S$ or $L \cdot S$ respectively. In all other cases we shall have $\lambda_1 = \lambda_2 = 0$ and as a result $\lambda = 0$. This remark leads to the conclusion that if an optimal solution exists between the limits $\sum_i x_i = L \cdot S$ and $\sum_i x_i = U \cdot S$ then $\lambda = 0$.

$$x_i^* = 0 \qquad , i \in N_0^{\lambda=0} = \{i \in N : 0 \geq 2(y_i - i \cdot s)\}$$

$$x_i^* = (y_i - i \cdot s) \qquad , i \in N^{\lambda=0} = \{i \in N : 2((y_i - i \cdot s) - X_{left}(i)) < 0 < 2(y_i - i \cdot s)\}$$

$$x_i^* = X_{left}(i) \qquad , i \in N_{X_{left}}^{\lambda=0} = \{i \in N : 0 \leq 2((y_i - i \cdot s) - X_{left}(i))\}$$

By noting that $(y_i - i \cdot s)$ is the original deviation from the line, it follows that for $\forall i \in N$ having original deviation $(y_i - i \cdot s) \geq X_{left}(i)$ we shall have an optimal value $x_i^* = X_{left}(i)$ For the rest we shall have an optimal value $x_i^* = (y_i - i \cdot s)$ equal to the original deviation from the line or optimal value $x_i^* = 0$ for $\forall i \in N$ having negative original deviation from the line. As a result, if the solution lies between $L \cdot S$ and $U \cdot S$ then it must either be on the line or at the point where all $x_i^* = X_{left}(i)$, if this happens before reaching the line.

4.3 Solution Algorithm

We present the following algorithm for the solution of the problem.

```
1   $Sum = \sum_i x_i = 0$        $x_i = 0 \forall i \in N$
2   decision = not final
3   While decision = not final{
```

```
4    If not all $x_i = X_{left}(i)$ {
5        Find $diff_1 = Max\{(y_i - i\cdot s - x_i)\ where\ i \in N : x_i - X_{left}(i) \le 0\}$
         and $diff_2 = Max\left\{(y_i - i\cdot s - x_i)\ \begin{matrix} where\ i \in N : x_i - X_{left}(i) \le 0 \\ and\ (y_i - i\cdot s - x_i) \ne diff_1 \end{matrix}\right\}$
         and all $x_i$ at $diff_1$
6        If $diff_1 > 0$ {
7            If not US{
8                find $diff = \min\left\{ diff_1 - diff_2, diff_1, X_{left} - x_i, \frac{US - Sum}{\sum_i I_{diff}(i)} \right\}$
                 where $I_{diff}(i) = \begin{cases} 1 & ((y_i - i\cdot s) - x_i) = diff_1 \text{ and } x_i - X_{left}(i) \le 0 \\ 0 & otherwise \end{cases}$
9                update $Sum = Sum + diff \cdot \sum_i I_{diff}(i)$
                     $x_i = x_i + diff$ for all $x_i$ at $diff_1$
10           }
11           else if US $\rightarrow$ decision=optimal
12       }
13       else if $diff_1 \le 0$ {
14           If not LS{
15               find $diff = \min\left\{ diff_1 - diff_2, X_{left} - x_i, \frac{LS - Sum}{\sum_i I_{diff}(i)} \right\}$
         where $I_{diff}(i) = \begin{cases} 1 & ((y_i - i\cdot s) - x_i) = diff_1 \text{ and } x_i - X_{left}(i) \le 0 \\ 0 & otherwise \end{cases}$
```

```
                        update $Sum = Sum + diff \cdot \sum_i I_{diff}(i)$
                              $x_i = x_i + diff$ for all $x_i$ at $diff_1$
                  }
                  else if LS $\rightarrow$decision=optimal
            }
      }
      else if all $x_i = X_{left}(i)$ {
            if $LS \leq Sum \leq US$ $\rightarrow$decision=optimal
            else $\rightarrow$decision=infeasible
}
```

The way this algorithm takes action is similar to the way we use the sweep. In detail the algorithm acts on those points only that have not yet met the limit $x_i \leq X_{left}(i)$.

We begin from the point with the maximum deviation from the line and keep going down like sweeping until we reach the one having the second maximum deviation from the line, concurrently increasing the value of $\sum_i x_i$ by the amount of the difference of those two deviations. At this point we must note that in case two or more points have the same maximum deviation then the algorithm acts on these points as a group by altering the correspondent x_i's by the same amount equal to the difference between the two largest deviations and by increasing the value of $\sum_i x_i$ by the amount of this multiplied by the number of points on which the algorithm takes action. At the end of each iteration we update both the x_i 's and the $\sum_i x_i$ and calculate from scratch the difference between the two largest deviations, always ensuring that no basic restriction of the problem is being violated.

As a result the algorithm sets zero value to the points it has not yet reached when terminated because of some constraint violation, value $X_{left}(i)$ to the points that have already reached their limit value and value $x_i^* = (y_i - i \cdot s) - d$ to the rest points where d is the final distance from the line of the algorithm terminating point. This distance may be either positive when the algorithm stops above the line or negative when the algorithm stops below the line. The algorithm terminates either at the optimum solution if one exists or with no feasible solution.

We use the following notation

Zero-Line	The $f(i) = i \cdot s$ Line where $x_i = (y_i - i \cdot s)$
L-Line	The parallel to the Zero-Line where $\sum_i x_i = L \cdot S$
U-Line	The parallel to the Zero-Line where $\sum_i x_i = U \cdot S$
Xleft-Line	The parallel to the **Zero-Line** where $\sum_i x_i = \sum_i X_{left}(i)$ and $x_i^* = X_{left}(i)$

4.3.1 Proof of optimality

Corollary: The algorithm terminates at the optimum solution if one exists.

Proof

Given that a feasible solution exists we have that $L \cdot S \leq \sum_i X_{left}$

If $\sum_i X_{left} \leq U \cdot S$ then $L \cdot S \leq \sum_i X_{left} \leq U \cdot S$ and we examine all possible orderings of the constraints.

Case1 **Zero →L→ Xleft →U** **Optimal Solution L**

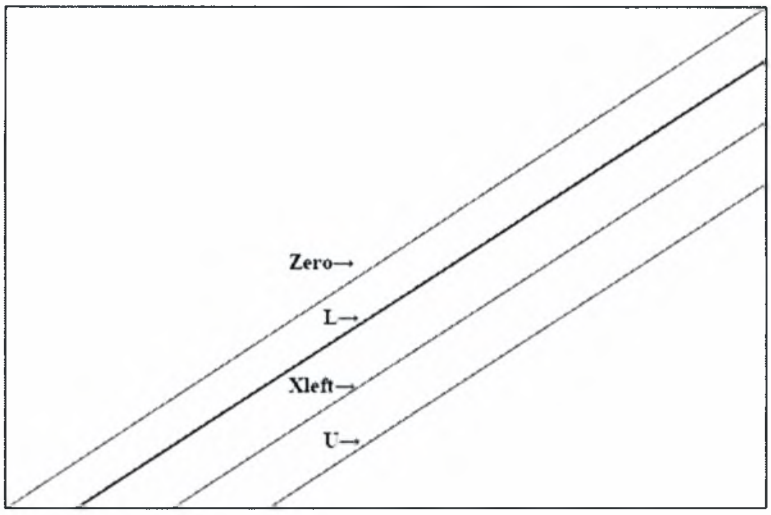


Figure 4.1 Constraints ordering in Case 1

Let $\frac{\lambda}{2}$ = Distance between L and the line. This distance is negative because L-Line is below the Zero-Line .We also have that $\lambda_1 = 0$ and $\lambda_2 > 0$ because the algorithm has stopped on L-Line so $\lambda = \lambda_1 - \lambda_2 = -\lambda_2 < 0$

We have variables of three kinds

(a) $x_i^* = 0$ For the points that have not been reached

$\frac{\lambda}{2} \geq (y_i - i \cdot s)$ is valid, since the points have not been reached that means their initial deviation is lower than the algorithm terminating point.

(b) $x_i^* = (y_i - i \cdot s) - \frac{\lambda}{2}$ For the points that lie on L.

$((y_i - i \cdot s) - X_{left}(i)) < \frac{\lambda}{2} < (y_i - i \cdot s)$ is valid since the points lie on L that is between Xleft and their initial deviation.

(c) $x_i^* = X_{left}(i)$ For the points we left behind because they have reached their upper bound.

$\frac{\lambda}{2} \leq ((y_i - i \cdot s) - X_{left}(i))$ is valid because the algorithm has continued downwards and has terminated at L.

Since $\lambda_1 \cdot \lambda_2 = 0$ our solution fulfils the requirements of the **Theorem 4.1** thus is optimal.

Case2 **L→ Zero →Xleft →U** **Optimal Solution Zero-Line**

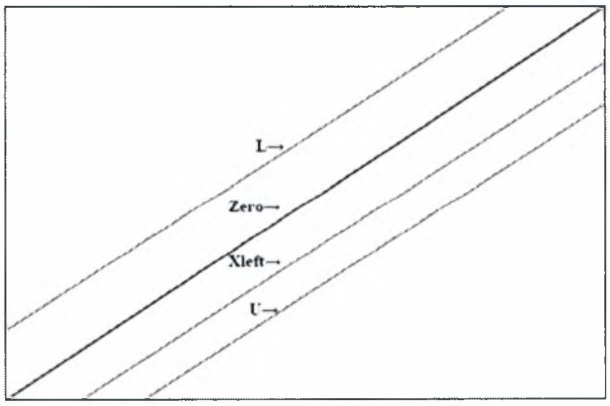


Figure 4.2 Constraints ordering in Case 2

We have that $\lambda_1 = 0$ and $\lambda_2 = 0$ because the algorithm terminates between L-Line and U-Line so $\lambda = 0$

We have variables of three kinds

(a) $x_i^* = 0$ For the points that have not been reached

$0 \geq (y_i - i \cdot s)$ is valid, since the points have not been reached that means their initial deviation is lower than the Zero-Line.

(b) $x_i^* = (y_i - i \cdot s)$ For the points that lie on the Zero-Line.

$((y_i - i \cdot s) - X_{left}(i)) < 0 < (y_i - i \cdot s)$ is valid since the points lie on the Zero-Line that is between Xleft and their initial deviation.

(c) $x_i^* = X_{left}(i)$ For the points we left behind because they have reached their upper bound.

$0 \leq ((y_i - i \cdot s) - X_{left}(i))$ is valid because the algorithm has continued downwards and has terminated at the Zero-Line.

Since $\lambda_1 \cdot \lambda_2 = 0$ our solution fulfills the requirements of the **Theorem 4.1** thus is optimal.

Case3 **L→ Xleft → Zero →U** **Optimal Solution Xleft**

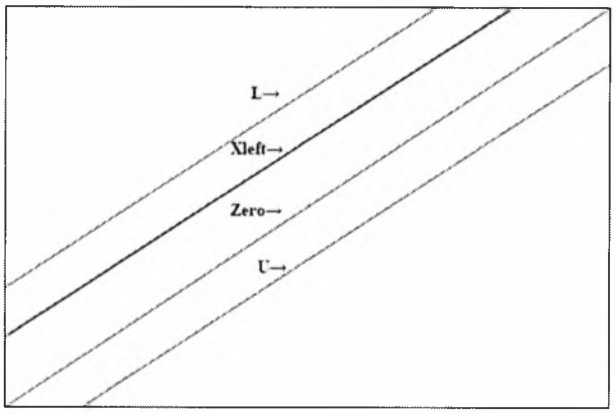


Figure 4.3 Constraints ordering in Case 3

We have that $\lambda_1 = 0$ and $\lambda_2 = 0$ because the algorithm terminates between L-Line and U-Line so $\lambda = 0$

We have only one kind of variables

$x_i^* = X_{left}(i)$ Because all points have reached their upper bound since the Algorithm has terminated on Xleft.

$0 \le \left(\left(y_i - i \cdot s\right) - X_{left}(i)\right)$ is valid since the Algorithm has terminated on Xleft before reaching the Zero-Line.

Since $\lambda_1 \cdot \lambda_2 = 0$ our solution fulfills the requirements of the **Theorem 4.1** thus is optimal.

Case4 **L→ Xleft →U→ Zero** **Optimal Solution Xleft**

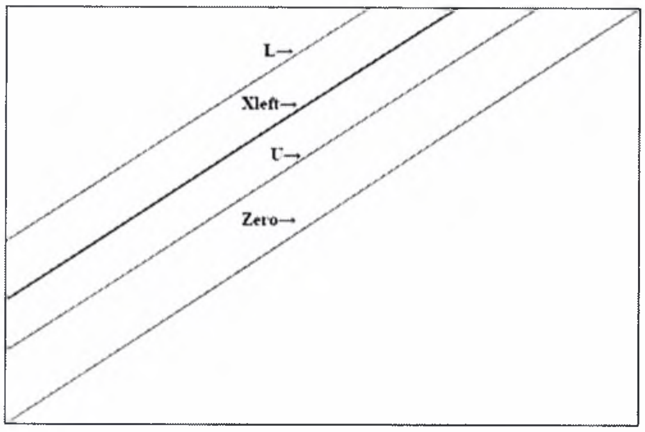


Figure 4.4 Constraints ordering in Case 4

We have that $\lambda_1 = 0$ and $\lambda_2 = 0$ because the algorithm terminates between L-Line and U-Line so $\lambda = 0$

We have only one kind of variables

$x_i^* = X_{left}(i)$ Because all points have reached their upper bound since the Algorithm has terminated on Xleft.

$0 \leq ((y_i - i \cdot s) - X_{left}(i))$ is valid since the Algorithm has terminated on Xleft before reaching the Zero-Line.

Since $\lambda_1 \cdot \lambda_2 = 0$ our solution fulfills the requirements of the **Theorem 4.1** thus is optimal.

Let $\frac{\lambda}{2}$ = Distance between U-Line and the Zero-Line. This distance is positive because U-Line is above the Zero-Line. We also have that $\lambda_1 > 0$ and $\lambda_2 = 0$ because the algorithm has stopped on U-Line so $\lambda = \lambda_1 - \lambda_2 = \lambda_1 > 0$

We have variables of three kinds

(a) $x_i^* = 0$ For the points that have not been reached

$\frac{\lambda}{2} \geq (y_i - i \cdot s)$ is valid, since the points have not been reached that means their initial deviation is lower than the algorithm terminating point.

(b) $x_i^* = (y_i - i \cdot s) - \frac{\lambda}{2}$ For the points that lie on U.

$((y_i - i \cdot s) - X_{left}(i)) < \frac{\lambda}{2} < (y_i - i \cdot s)$ is valid since the points lie on U, that is between Xleft and their initial deviation.

(c) $x_i^* = X_{left}(i)$ For the points we left behind because they have reached their upper bound.

$\frac{\lambda}{2} \leq ((y_i - i \cdot s) - X_{left}(i))$ is valid because the algorithm has continued downwards and has terminated at U.

Since $\lambda_1 \cdot \lambda_2 = 0$ our solution fulfills the requirements of the **Theorem 4.1** thus is optimal.

2) If $\sum_i X_{left} \geq U \cdot S$ then $L \cdot S \leq U \cdot S \leq \sum_i X_{left}$ and we examine all possible orderings of the constraints.

Zero →L →U→ Xleft	**Optimal Solution L**
L→ Zero →U→Xleft	**Optimal Solution Zero**
L →U →Zero → Xleft	**Optimal Solution U**
L →U→ Xleft→ Zero	**Optimal Solution U**

The proof can be done with the same reasoning.

4.3.2 Algorithm Complexity

```
1   $Sum = \sum_i x_i = 0 \qquad x_i = 0 \forall i \in N$
2   decision = not final
3   While decision = not final{                                              $O(1)$
4       If not all $x_i = X_{left}(i)$ {                                     $O(n)$
5           Find $diff_1 = Max\{(y_i - i \cdot s - x_i) \text{ where } i \in N : x_i - X_{left}(i) \le 0\}$
            and $diff_2 = Max\left\{(y_i - i \cdot s - x_i) \begin{matrix} \text{where } i \in N : x_i - X_{left}(i) \le 0 \\ \text{and } (y_i - i \cdot s - x_i) \ne diff_1 \end{matrix}\right\}$   $O(n)$
            and all $x_i$ at $diff_1$
6           If $diff_1 > 0$ {                                                $O(1)$
7               If not US{                                                   $O(1)$
8                   find $diff = \min\left\{diff_1 - diff_2, diff_1, X_{left} - x_i, \frac{US - Sum}{\sum_i I_{diff}(i)}\right\}$   $O(n)$
                    where $I_{diff}(i) = \begin{cases} 1 & ((y_i - i \cdot s) - x_i) = diff_1 \text{ and } x_i - X_{left}(i) \le 0 \\ 0 & otherwise \end{cases}$
9                   update $Sum = Sum + diff \cdot \sum_i I_{diff}(i)$           $O(n)$
                    $x_i = x_i + diff$ for all $x_i$ at $diff_1$
10              }
11              else if US $\rightarrow$ decision=optimal                    $O(1)$
12          }
13          else if $diff_1 \le 0$ {                                         $O(1)$
14              If not LS{                                                   $O(1)$
15                      find $diff = \min\left\{ diff_1 - diff_2, X_{left} - x_i, \frac{LS - Sum}{\sum_i I_{diff}(i)} \right\}$          $O(n)$
              where $I_{diff}(i) = \begin{cases} 1 & ((y_i - i \cdot s) - x_i) = diff_1 \text{ and } x_i - X_{left}(i) \le 0 \\ 0 & otherwise \end{cases}$
16                      update $Sum = Sum + diff \cdot \sum_i I_{diff}(i)$          $O(n)$
                        $x_i = x_i + diff$ for all $x_i$ at $diff_1$
17                  }
18                  else if LS $\rightarrow$decision=optimal          $O(1)$
19              }
20          }
21          else if all $x_i = X_{left}(i)$ {          $O(n)$
22              if $LS \le Sum \le US$ $\rightarrow$decision=optimal          $O(1)$
23              else $\rightarrow$decision=infeasible          $O(1)$
24  }
```

Each while loop is $O(n)$ since each command is worst case $O(n)$ and all commands are executed in serial.

In Worst Case line 8 will be executed n+3 times

$$\left.\begin{array}{ll} 1 & for\ diff_1 - diff_2 \\ 1 & \quad for \quad diff_1 \\ n & \quad for\ X_{left} - x_i \\ 1 & \quad for\ \dfrac{Us - Sum}{\sum_{i \in N^{diff}} i} \end{array}\right\} (n+3) = O(n)$$

In Worst Case line 15 will be executed n+2 times

$$\left.\begin{array}{ll} 1 & \text{for } diff_1 - diff_2 \\ n & \text{for } X_{left} - x_i \\ 1 & \text{for } \dfrac{Ls - Sum}{\sum_{i \in N^{diff}} i} \end{array}\right\} (n+2) = O(n)$$

Finally lines 11, 18, 22 and 23 will be executed no more than once (since after their execution causes the termination of the algorithm).Thus the while loop is $O(n)$, and the algorithm worst case complexity is $O(n^2)$.

Note: No sorting is needed for computing the maximum values, but we need to calculate $diff_1$ and $diff_2$ at each iteration.

4.4 General Form

4.4.1. General Form Formulation

Table 4.1 presents the correspondence of the general form and the special case introduced in **Section 3.6.2** and **Table 4.2** utilizes the transformations needed to transfer between these forms.

Special Case	General Form
$Min\ z = \sum_i ((y_i - i \cdot s) - x_i)^2$	$Min\ z = \sum_i (x_i')^2$
$s.t.\quad \sum_i x_i \le U \cdot S$	$s.t.\quad \sum_i x_i' \ge L$
$\sum_i x_i \ge L \cdot S \quad U > L$	$\sum_i x_i' \le U \quad U > L$

Table 4.1 Transformation between general form and special case

Correspondence	
$0 \le x_i \le X_{left}(i)$	$a_i \le x_i' \le b_i$
x_i	$x_i' = (y_i - i \cdot s) - x_i$
$\sum_i x_i$	$\sum_i x_i' = \sum_i (y_i - i \cdot s) - \sum_i x_i$
$X_{left}(i)$	$a_i = (y_i - i \cdot s) - X_{left}(i)$
0	$b_i = (y_i - i \cdot s)$
$L \cdot S$	$U' = \sum_i (y_i - i \cdot s) - L \cdot S$
$U \cdot S$	$L' = \sum_i (y_i - i \cdot s) - U \cdot S$

Table 4.2 Term correspondence between general form and special case

It is obvious that the solution x_i's obtained by solving the general form problem can be used to render the solution to our specialized problem thus

$$y_i - x_i = x_i' + i \cdot s$$

4.4.2 Optimality Theorem

Theorem 4.2 (Characterization of the optimal solution of the Problem).

A feasible solution $\mathbf{x}^* = (x_i^*)$ is optimal solution of the problem if and only if there exist $\lambda_1, \lambda_2 \in \Re^+ : \lambda_1 \cdot \lambda_2 = 0$ such as for $\lambda = \lambda_1 - \lambda_2$ the following hold:

$$x_i^* = b_i \qquad , i \in N_0^\lambda = \{i \in N : \lambda \geq 2b_i\}$$

$$x_i^* = \frac{\lambda}{2} \qquad , i \in N^\lambda = \{i \in N : 2a_i < \lambda < 2b_i\}$$

$$x_i^* = a_i \qquad , i \in N_{X_{left}}^\lambda = \{i \in N : \lambda \leq 2b_i\}$$

4.4.3 Solution Algorithm

We present the form of the algorithm in the general case

```
1    $Sum = \sum_i x_i = \sum_i b_i \quad x_i = b_i \forall i \in N$
2    decision = not final
3    While decision = not final{
4            If not all $x_i = a_i$ {
5                    Find $diff_1 = Max\{x_i \text{ where } i \in N : x_i - a_i \geq 0\}$
                     and $diff_2 = Max\left\{x_i \; \begin{matrix} x_i \text{ where } i \in N : x_i - a_i \geq 0 \\ \text{and } x_i \neq diff_1 \end{matrix}\right\}$
                     and all $x_i$ at $diff_1$
6                    If $diff_1 > 0$ {
7                            If not L{
```

```
8                       find $diff = \min\left\{ diff_1 - diff_2, diff_1, x_i - a_i, \frac{Sum - L}{\sum_i I_{diff}(i)} \right\}$
                            where $I_{diff}(i) = \begin{Bmatrix} 1 & x_i = diff_1 \\ 0 & otherwise \end{Bmatrix}$
9                       update $Sum = Sum - diff \cdot \sum_i I_{diff}(i)$
                            $x_i = x_i - diff$ for all $x_i$ at $diff_1$
10                  }
11                  else if L $\rightarrow$decision=optimal
12              }
13              else if $diff_1 \le 0$ {
14                  If not U{
15                      find $diff = \min\left\{ diff_1 - diff_2, x_i - a_i, \frac{Sum - U}{\sum_i I_{diff}(i)} \right\}$
                            where $I_{diff}(i) = \begin{Bmatrix} 1 & x_i = diff_1 \\ 0 & otherwise \end{Bmatrix}$
16                      update $Sum = Sum - diff \cdot \sum_i I_{diff}(i)$
                            $x_i = x_i - diff$ for all $x_i$ at $diff_1$
17                  }
18                  else if U $\rightarrow$decision=optimal
19              }
20          }
21          else if all $x_i = a_i$ {
```

```
            if $L \le Sum \le U$ $\rightarrow$decision=optimal
            else $\rightarrow$decision=infeasible
}
```

4.5 Modifying Existing Solution Techniques to fit our needs

Looking into the characterization theorem in more detail one can see that in every case the optimal solution satisfies one or more of the following Knapsack constraints.

$\sum_i x_i = U$ representing the U-Line or

$\sum_i x_i = L$ representing the L-line or

$\sum_i x_i = \sum_i a_i$ representing the Xleft-line or

$\sum_i x_i = \sum_{i:a_i>0} a_i$ representing the Zero-line.

It is obvious that our problem can be separated in four different Quadratic Knapsack Problems of the traditional formulation. Solving these problems separately using $O(n)$ algorithm already proposed by P. Brucker **[3]** or Pardalos and N. Kovoor **[13]** and then choosing the best optimal solution of the four yields the desirable optimal solution to the problem we study.

Examining all possible orderings of the constraints we can summarize on the following results.

Case1	Zero→	**L→**	Xleft→	U	**Optimal Solution L**
Case2	L→	**Zero→**	Xleft→	U	**Optimal Solution Zero-Line**
Case3	L→	**Xleft→**	Zero→	U	**Optimal Solution Xleft**
Case4	L→	**Xleft→**	U→	Zero	**Optimal Solution Xleft**
Case5	Zero→	**L→**	U→	Xleft	**Optimal Solution L**
Case6	L→	**Zero→**	U→	Xleft	**Optimal Solution Zero**
Case7	L→	**U→**	Zero→	Xleft	**Optimal Solution U**
Case8	L→	**U→**	Xleft→	Zero	**Optimal Solution U**

Table 4.3 All possible orderings of the constraints

We can easily see that the optimal solution always lies on the second-ordered constraint met that is on the constraint with the second largest value. Thus there is no need to solve four different Quadratic problems but only one after doing a simple ordering of the constraints' target values. (We hereby note that in the special case the second-ordered constraint is met on the constraint with the second smallest value). Thus by applying Pardalos and Kovoor **[13]** algorithm we get a $O(n)$ solution algorithm. In fact the algorithm uses binary search combined with a $O(n)$ median search implementation which yields $O(n \log n)$. However in terms of computational time we have $O(n)$. Since $\Omega(n)$ is an obvious lower bound for the problem complexity our proposed algorithm is $\Theta(n)$ and thus is optimal.

Although our first method has worst-case performance of $O(n^2)$ it is of interest because of its efficiency in practice and its simplicity and ease of implementation. One feature of our algorithm is that it requires no sorting to be performed, nor does it require either randomized or exact ordinal statistics to be computed. Cases that consist of many variables can be met with the modified $O(n)$ algorithm which gives optimal results in terms of computational time.

4.6 Conclusions

In this chapter we have focused on the special case of Quadratic Knapsack Problems, where deviations from the target value are allowed for the Knapsack constraint. We have used the KKT Conditions in order to characterize the optimal solution to the problem. After that we have proven global optimality of the solution. We then introduced a new algorithm for the solution of the problem and focused on optimality and complexity issues.

Throughout the chapter we followed the formulation of the Aircraft-Maintenance Problem presented in **Section 3.6** which gave rise to investigating this special kind of Quadratic Knapsack Problem. Later on we gave the more general form of this special type of Problem and supplied both the optimal solution characterization theorem and the solution algorithm suited for the general case. In the end of the chapter we showed how known techniques for the standard Quadratic Knapsack formulation can be directly applied in case the number of variables grows too large.

Chapter 5 Computational experience and numerical examples.

5.1 Introduction

In this chapter, we present results of some numerical experiments obtained by applying the algorithms suggested in the previous chapter to problems under consideration. The algorithms have been implemented in the C programming language and one can find this implementation in Appendix A. AMPL can also be used for solving the problem and thus computational times are supplied for comparison. AMPL modeling code can be found in Appendix B We also present some numerical examples so that the approach of the previous chapter can be made clear.

5.2 Computational Results

The computations were performed on an Intel Celeron 335 Prescott Processor 2.8Ghz/1GB DDR SDRAM IBM PC compatible. Each Problem was run 30 times. Parameters were randomly generated between regions that have physical meaning. Notation used is taken from **Section 3.6**.Parameters L and U were randomly chosen in each running so that the Algorithm randomly chooses the binding constraint. We first comment on the results of the $O\left(N^2\right)$ algorithm.

When $N < 400$ the run time of the algorithm is so small, that the timer does not recognize the corresponding value from its computer zero. In such cases the timer displays 0 seconds. As we can observe the (average number of iterations) is nearly equal to the number of variables for large *N*. Computational time is proportional to N^2 besides the algorithm's complexity is $O\left(N^2\right)$.

$X_{\max} = 300$	$S = 0,75 \cdot X_{\max}$				
Number of variables	400	2000	4000	20000	40000
Average Number of Iterations	415	2106	4176	20899	41876
Average run time (seconds)	0.015	0.125	0.468	11.5	45.75
$X_{\max} = 3000$	$S = 0,75 \cdot X_{\max}$				
Number of variables	400	2000	4000	20000	40000
Average Number of Iterations	512	2556	5160	25517	51151
Average run time (seconds)	0.015	0.092	0.381	9.294	37.186
$X_{\max} = 30000$	$S = 0,75 \cdot X_{\max}$				
Number of variables	400	2000	4000	20000	40000
Average Number of Iterations	537	2753	5559	27669	55196
Average run time (seconds)	0	0.074	0.279	6.991	27.6

Table 5.1 Computational Results for different values of $X_{\max}$

Different values of $\frac{S}{X_{\max}}$ do not seem to alter results thus we can conclude that the algorithms efficiency is independent of $\frac{S}{X_{\max}}$. L,U were on purpose randomly selected because we wanted the selection of the tight constraint to be made randomly.

$X_{\max} = 300$	$S = 0,75 \cdot X_{\max}$				
Number of variables	400	2000	4000	20000	40000
Average Number of Iterations	415	2106	4176	20899	41876
Average run time (seconds)	0.015	0.343	1.359	34.265	137.75
$X_{\max} = 300$	$S = X_{\max}$				
Number of variables	400	2000	4000	20000	40000
Average Number of Iterations	415	2106	4176	20899	41875
Average run time (seconds)	0.015	0.343	1.359	34.265	137.765
$X_{\max} = 300$	$S = 1,25 X_{\max}$				
Number of variables	400	2000	4000	20000	40000
Average Number of Iterations	415	2106	4176	20899	41876
Average run time (seconds)	0.015	0.343	1.359	34.265	137.703

Table 5.2 Computational Results for different values of $\frac{S}{X_{\max}}$

We now compare the $O(n^2)$sweep algorithm versus $O(n)$Pardalos and Kovoor modified algorithm and AMPL package.

	Average run time (seconds)					
Number of variables	400	2000	4000	20000	40000	80000
SWEEP	0.016	0.112	0.397	8.728	37.186	145.744
PARDALOS	0	0	0	0.015	0.032	0.047
AMPL	0.24	0.52	1.07	2.14	4,29	8.51
	Average Number of Iterations					
Number of variables	400	2000	4000	20000	40000	80000
SWEEP	415	2106	4176	20899	41875	81982
PARDALOS	11	13	14	17	17	17
AMPL	10	11	12	14	15	17

Table 5.3 Algorithms Comparison Results

The $O(n)$ Pardalos and Kovoor modified algorithm is completely dominant in terms of execution time. AMPL also produces excellent results. The $O(n^2)$sweep algorithm has the worst of the three execution times, especially when the number of the variables grows large. However all running times are fairly sensible meaning that in spite of being the slowest solution technique it can still be used in many real-life problems with satisfactory results.

5.3 Numerical Examples

We provide below the solution of two simple particular problems obtained by using the solution approach suggested in this thesis. We revise the model formulation of the Aircraft maintenance application.

5.3.1 Numerical Example 1

Parameter values:

$S = 200$: required total flight time during planning horizon

y_i : residual flight time of aircraft i at the beginning of planning horizon,

$X_{\max} = 300$: maximum time an aircraft can fly during planning horizon ,

$Y_{\min} = 0.1$: minimum residual flight time of an available aircraft,

$L = 0,95$, $U = 1,05$: real numbers denoting the maximum deviation from the value of S that can be tolerated $(U > L)$,

$s = \frac{y_{\max}}{N} = \frac{300}{8} = 37,5$: the slope of the deviation line where

$y_{\max} = 300$: maximum residual flight time of an available aircraft,

$N = 8$: total numbers of aircrafts available for flight

Results

$y_1 = 90$	$s = 37.5$
$y_2 = 100$	$2s = 75$

$y_3 = 133$	$3s = 112.5$
$y_4 = 150$	$4s = 150$
$y_5 = 218$	$5s = 187.5$
$y_6 = 250$	$6s = 225$
$y_7 = 263$	$7s = 262.5$
$y_8 = 300$	$8s = 300$

Iteration 1:
diff1 – diff2 = 22
diff1 – line = 52.5
Xmax – xi = 50
Ls – Σxi = 210

$x_1 = 22$	$y_1 = 68$
$x_2 = 0$	$y_2 = 100$
$x_3 = 0$	$y_3 = 133$
$x_4 = 0$	$y_4 = 150$
$x_5 = 0$	$y_5 = 218$
$x_6 = 0$	$y_6 = 250$
$x_7 = 0$	$y_7 = 263$
$x_8 = 0$	$y_8 = 300$

Σxi = 22

Iteration 2:
diff1 – diff2 = 5.5
diff1 – line = 30.5
Xmax – xi = 28
(Ls – Σxi)/2 = (210 – 22)/2 = 188/2 = 94

$x_1 = 27.5$	$y_1 = 62.5$
$x_2 = 0$	$y_2 = 100$
$x_3 = 0$	$y_3 = 133$
$x_4 = 0$	$y_4 = 150$
$x_5 = 5.5$	$y_5 = 212.5$
$x_6 = 0$	$y_6 = 250$
$x_7 = 0$	$y_7 = 263$
$x_8 = 0$	$y_8 = 300$

Σxi = 33

Iteration 3:
diff1 – diff2 = 4.5
diff1 – line = 25
Xmax – xi = 22.5
(Ls – Σxi)/4 = (210 – 33)/4 = 177/4 = 44.25

$x_1 = 32$	$y_1 = 58$
$x_2 = 4.5$	$y_2 = 95.5$
$x_3 = 0$	$y_3 = 133$

$x_4 = 0$	$y_4 = 150$
$x_5 = 10$	$y_5 = 208$
$x_6 = 4.5$	$y_6 = 245.5$
$x_7 = 0$	$y_7 = 263$
$x_8 = 0$	$y_8 = 300$
$\Sigma xi = 51$	

Iteration 4:
diff1 – diff2 = 20
diff1 – line = 20.5
Xmax – xi = 18
$(Ls - \Sigma xi)/5 = (210 - 51)/5 = 159/5 = 31.8$

$x_1 = 50$	$y_1 = 40$
$x_2 = 22.5$	$y_2 = 77.5$
$x_3 = 18$	$y_3 = 115$
$x_4 = 0$	$y_4 = 150$
$x_5 = 28$	$y_5 = 190$
$x_6 = 22.5$	$y_6 = 227.5$
$x_7 = 0$	$y_7 = 263$
$x_8 = 0$	$y_8 = 300$
$\Sigma xi = 141$	

Iteration 5:
diff1 – diff2 = 2
diff1 – line = 2.5
Xmax – xi = 22
$(Ls - \Sigma xi)/4 = (210 - 141)/4 = 69/4 = 17.25$

$x_1 = 50$	$y_1 = 40$
$x_2 = 24.5$	$y_2 = 75.5$
$x_3 = 20$	$y_3 = 113$
$x_4 = 0$	$y_4 = 150$
$x_5 = 30$	$y_5 = 188$
$x_6 = 24.5$	$y_6 = 225.5$
$x_7 = 0$	$y_7 = 263$
$x_8 = 0$	$y_8 = 300$
$\Sigma xi = 149$	

Iteration 6:
diff1 – diff2 = 0.5
diff1 – line = 0.5
Xmax – xi = 20
$(Ls - \Sigma xi)/5 = (210 - 149)/5 = 61/5 = 12.2$

$x_1 = 50$	$y_1 = 40$
$x_2 = 25$	$y_2 = 75$
$x_3 = 20.5$	$y_3 = 112.5$

$x_4 = 0$	$y_4 = 150$
$x_5 = 30.5$	$y_5 = 187.5$
$x_6 = 25$	$y_6 = 225$
$x_7 = 0.5$	$y_7 = 262.5$
$x_8 = 0$	$y_8 = 300$
$\Sigma xi = 151.5$	

<u>Iteration 7:</u>
diff1 – diff2 = 0 - (-oo) = +oo
Xmax – xi = 19.5
(Us – Σxi)/7 = (190 – 151.5)/7 = 38.5/7 = 5.5

$x_1 = 50$	$y_1 = 40$
$x_2 = 30.5$	$y_2 = 69.5$
$x_3 = 26$	$y_3 = 107$
$x_4 = 5.5$	$y_4 = 144.5$
$x_5 = 36$	$y_5 = 182$
$x_6 = 30.5$	$y_6 = 219.5$
$x_7 = 6$	$y_7 = 257$
$x_8 = 5.5$	$y_8 = 294.5$
$\Sigma xi = 190$	

optimal

5.3.2 Numerical Example 2

Parameter values:

$S = 200$: required total flight time during planning horizon

y_i : residual flight time of aircraft i at the beginning of planning horizon,

$X_{\max} = 300$: maximum time an aircraft can fly during planning horizon ,

$Y_{\min} = 0.1$: minimum residual flight time of an available aircraft,

$L = 0{,}45$, $U = 1{,}05$: real numbers denoting the maximum deviation from the value of S that can be tolerated $(U > L)$,

$s = \frac{y_{\max}}{N} = \frac{300}{8} = 37{,}5$: the slope of the deviation line where

$y_{\max} = 300$: maximum residual flight time of an available aircraft,

$N = 8$: total numbers of aircrafts available for flight

Results

$y_1 = 90$	$s = 37.5$
$y_2 = 100$	$2s = 75$
$y_3 = 133$	$3s = 112.5$
$y_4 = 150$	$4s = 150$
$y_5 = 218$	$5s = 187.5$
$y_6 = 250$	$6s = 225$
$y_7 = 263$	$7s = 262.5$
$y_8 = 300$	$8s = 300$

<u>Iteration 1:</u>
diff1 – diff2 = 22
diff1 – line = 52.5
Xmax – xi = 50
Ls – Σxi = 90

$x_1 = 22$	$y_1 = 68$
$x_2 = 0$	$y_2 = 100$
$x_3 = 0$	$y_3 = 133$
$x_4 = 0$	$y_4 = 150$
$x_5 = 0$	$y_5 = 218$
$x_6 = 0$	$y_6 = 250$
$x_7 = 0$	$y_7 = 263$
$x_8 = 0$	$y_8 = 300$
$\Sigma xi = 22$	

<u>Iteration 2:</u>
diff1 – diff2 = 5.5
diff1 – line = 30.5
Xmax – xi = 28
(Ls – Σxi)/2 = (210 – 22)/2 = 188/2 = 34

$x_1 = 27.5$	$y_1 = 62.5$
$x_2 = 0$	$y_2 = 100$
$x_3 = 0$	$y_3 = 133$
$x_4 = 0$	$y_4 = 150$
$x_5 = 5.5$	$y_5 = 212.5$

$x_6 = 0$	$y_6 = 250$
$x_7 = 0$	$y_7 = 263$
$x_8 = 0$	$y_8 = 300$
$\Sigma xi = 33$	

Iteration 3:

diff1 – diff2 = 4.5
diff1 – line = 25
Xmax – xi = 22.5
(Ls – Σxi)/4 = (210 – 33)/4 = 177/4 = 14.25

$x_1 = 32$	$y_1 = 58$
$x_2 = 4.5$	$y_2= 95.5$
$x_3 = 0$	$y_3 = 133$
$x_4 = 0$	$y_4 = 150$
$x_5 = 10$	$y_5 = 208$
$x_6 = 4.5$	$y_6 = 245.5$
$x_7 = 0$	$y_7 = 263$
$x_8 = 0$	$y_8 = 300$
$\Sigma xi = 51$	

Iteration 4:

diff1 – diff2 = 20
diff1 – line = 20.5
Xmax – xi = 18
(Ls – Σxi)/5 = (210 – 51)/5 = 159/5 = 7.8

$x_1 = 39.8$	$y_1 = 50.2$
$x_2 = 12.3$	$y_2= 87.7$
$x_3 = 7.8$	$y_3 = 125.2$
$x_4 = 0$	$y_4 = 150$
$x_5 = 17.8$	$y_5 = 200.2$
$x_6 = 12.3$	$y_6 = 237.7$
$x_7 = 0$	$y_7 = 263$
$x_8 = 0$	$y_8 = 300$
$\Sigma xi = 90$	

Iteration 5:

diff1 – diff2 = 12
diff1 – line = 12.7
Xmax – xi = 10.2
(US – Σxi)/5 = (210 – 90)/5 = 120/5 = 24

$x_1 = 50$	$y_1 = 40$
$x_2 = 22.5$	$y_2= 77.5$
$x_3 = 18$	$y_3 = 115$
$x_4 = 0$	$y_4 = 150$
$x_5 = 28$	$y_5 = 190$

$x_6 = 22.5$	$y_6 = 227.5$
$x_7 = 0$	$y_7 = 263$
$x_8 = 0$	$y_8 = 300$
$\Sigma xi = 141$	

Iteration 6:
diffl – diff2 = 2
diffl – line = 2.5
Xmax – xi = 22
$(Ls - \Sigma xi)/5 = (210 - 141)/4 = 69/5 = 17.25$

$x_1 = 50$	$y_1 = 40$
$x_2 = 24.5$	$y_2 = 75.5$
$x_3 = 20$	$y_3 = 113$
$x_4 = 0$	$y_4 = 150$
$x_5 = 30$	$y_5 = 188$
$x_6 = 24.5$	$y_6 = 225.5$
$x_7 = 0$	$y_7 = 263$
$x_8 = 0$	$y_8 = 300$
$\Sigma xi = 149$	

Iteration 7:
diffl – diff2 = 0.5
diffl – line = 0.5
Xmax – xi = 20
$(Us - \Sigma xi)/7 = (210 - 149)/5 = 61/5 = 12.2$

$x_1 = 50$	$y_1 = 40$
$x_2 = 25$	$y_2 = 75$
$x_3 = 20.5$	$y_3 = 112.5$
$x_4 = 0$	$y_4 = 150$
$x_5 = 30.5$	$y_5 = 187.5$
$x_6 = 25$	$y_6 = 225$
$x_7 = 0.5$	$y_7 = 262.5$
$x_8 = 0$	$y_8 = 300$
$\Sigma xi = 151.5$	

optimal

5.4 Conclusions

In this chapter, we presented results of some numerical experiments obtained by applying the algorithm suggested in the previous chapter to problems under consideration. Although the algorithm has $O(n^2)$ complexity instances of 40000 variables have been solved in a relatively sensible computational time. Its easy implementation makes it suitable for relatively small number of variables. Useful remarks have been made regarding sensitivity analysis. The C programming language implementation can be found in Appendix A. The AMPL modeling file can be found in Appendix B. The numerical examples presented at the end of the Chapter help so as the approach of the previous chapter can be made clear.

Chapter 6 Concluding Remarks

In this thesis, we studied Quadratic Knapsack problems where bound constraints are directly imposed on the continuous decision variables. These problems belong to the family of Quadratic Programming which is a major subsection of Nonlinear Optimization. The addition of Knapsack constraint on Quadratic Programming problems is shown to have numerous applications, including the least distance problem, Quadratic Programming defined on the convex hull of a set of points, and the maximum clique problem.

Moreover important fields of study that use Quadratic Knapsack as core formulation have been being presented. These include the Optimal Portfolio Selection, Quadratic Transportation, Multi-commodity Network Flows, Matrix Balancing problems and Aircraft Maintenance

Traditional approaches for accommodating such Quadratic Knapsack constraints have been proposed and analyzed for the case of a single tight-bounded Knapsack constraint. We have introduced the case where deviation from the target value of the Knapsack constraint is allowed.

The main contribution of the research reported in this work is that we have developed a new exact algorithm for a special class of Continuous Quadratic Knapsack Problems having reasonable solution times for nearly all instances encountered in practice, despite having Quadratic time bounds for a number of highly contrived problem instances.We have given proof of the optimality of the algorithm, implemented it in C programming language and gave numerical results. We also described a Quadratic Knapsack framework for the formulation, analysis and computation of solutions to a specific problem of military-aircraft maintenance.

We have also proposed modifications of existing algorithms so as they can deal with our specialized problem. Computational results on a variety of test problems have been presented showing that in spite of being $O(n^2)$ the algorithm remains appealing for problems with a reasonable number of variables.

Since $\Omega(n)$ is the lower bound for the complexity of every optimal solution algorithm and we have already presented an $O(n \cdot \log n)$ implementation that takes $O(n)$ computational time, it is of great interest whether a straightforward approach can be used to render an $O(n)$ algorithm. Besides, application of the results of this research on integer programming can also be a topic of further research. In detail the exact algorithms thoroughly presented in this thesis could be used to solve continuous relaxations of the integer programming problem, and then use rounding schemes or branch-and-bound techniques to find the optimal integral solution.

Appendix A C Implementation of the Algorithms.

```
/*------------------------------------------------------------------------
  Comparison of Quadratic Knapsack Exact Solution algorithms
  Gavranis Andreas <agavranis@gmail.com> March 2007
  This code is part of an implementation for the purposes
  of a postgraduate research.
 ------------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define N 30000
#define LIMIT 1e-6
#define MAXFLOW 1e38
#define SWAP(a,b) { float temp=(a);(a)=(b);(b)=temp; }
#defme MAXLIMIT 1e32

struct node
{
        int index;
        int data1;
        int data2;
        struct node *link;
};

/*------------------ Functions: ANSI C prototypes ------------------------*/

float qmedian(float a[], int n) ;


void swap(float *x,float *y);

void bsort1(float list[], int n);

void pardalos(float x[], float a[], float b[],float d,int n);

float choose_bound(float x[],float b[], float L, float U,int n);

struct node *insert(struct node *p, int value1,int value2, int count);

void erase (struct node *p);

void fprintnode_list ( struct node *p );
void printnode_list ( struct node *p );

float minimum(float x1 , float x2);

void bsort2(float list[][2], int n);

void readlist(float list[],int n);

float positive_minlist(float list[],struct node *front,int count,int n);

float positive_mindist(float list[][2], struct node *front,int count, int n);

void printlist(float list[],int n);

void printlist2(float list[][2],int n1,int n2);

int xleft_empty(float list[],int n);

int dist_empty(float dist[][2],float list[],int n);

struct node * next_dist(float xleft[],float dist[][2],int n, struct node *p, int *count , float *diff);

void update (float x[],float xleft[],float dist[][2] ,int n,struct node *front,int count,float xopt);

void print_compare(char *name, int j, float diff, float dist[][2],float Xleft[],struct node
*front,int count,int n, float L,char* lim);

void print_updated(char *name, float X[],float Xleft[],float Y[],float sum,int n);

void print_final(char *name,int n,float Y[],float s);

void print_iterations(char *name,int j);

void solve(float Y[],float Xleft[],int n,float s,float L,float U);

void print_ampl(char *name,int n,float Y[],float Xleft[],float s,float L,float U);

void print_original(char *name,int n,float Y[],float Xleft[],float s,float L,float U,float Xmax);

void print_duration(char *name,long double duration);

float choose_bound(float x[],float a[], float L, float U,int n);

void bsort(float list[], int n);

void pardalos(float x[], float a[], float b[],float d,int n);

float qmedian(float a[], int n);
```

```
/*------------------------- main() --------------------------------------*/


main( )
{
        long double start,stop,duration;
        int i;
        float s,Ymax,Xmax,target,alpha;
        float L,U;
        float *a;
        float *b;
        float *x;
        float Xleft[N],Xleft_COPY[N];
   float Y[N],Y_COPY[N];


        /*float Xleft[]={50,50,50,50,50,50,50,50};
   float Y[]={90,100,133,150,218,250,263,300};*/


        Xmax=300;
        Ymax=(float)0.5*Xmax;
        alpha=0.25;
        s=Xmax/N;
        L=(1-alpha)*Ymax;
        U=(1+alpha)*Ymax;

        for (i=0;i<N;i++) Y[i]=300*((float)rand())/(float) RAND_MAX;

        for (i=0;i<N;i++) Y_COPY[i]=Y[i];

        for (i=0;i<N;i++) Xleft[i]=s*((float)rand())/(float) RAND_MAX;

        for (i=0;i<N;i++) Xleft_COPY[i]=Xleft[i];


        /************** PRINT DATA FOR AMPL ****************/

        print_ampl("sweep.dat",N,Y,Xleft,s,L,U);

        /***************************************************/


        /************** PRINT ORIGINAL DATA ****************/

        print_original("num_sweep.txt",N,Y,Xleft,s,L,U,Xmax);
```

```
/****************************************************/

/************* COUNT EXECUTION TIME *****************/

start=clock();

solve(Y,Xleft,N,s,L,U);

stop=clock();

duration = (long double) (stop-start)/CLOCKS_PER_SEC;

/****************************************************/


/************** PRINT FINAL DATA *****************/

print_duration("num_sweep.txt",duration);

/****************************************************/

printf ("\nSweep Total execution time=%f \n",duration);

for (i=0;i<N;i++) Y[i]=Y_COPY[i];
for (i=0;i<N;i++) Xleft[i]=Xleft_COPY[i];

/************** PRINT ORIGINAL DATA *****************/

print_original("num_pardalos.txt",N,Y,Xleft,s,L,U,Xmax);

/****************************************************/

/******CONVERTING TO GENERAL FORM**********/

b=(float *) malloc (N*sizeof(float));
for (i=0;i<N;i++) b[i]=Y[i]-(i+1)*s;

a=(float *) malloc (N*sizeof(float));
for (i=0;i<N;i++) a[i]=Y[i]-(i+1)*s-Xleft[i];

x=(float *) malloc (N*sizeof(float));
for (i=0;i<N;i++) x[i]=0;


SWAP(L,U);
L=-L;
```

```
        U=-U;
        for (i=0;i<N;i++) {
                                        L+=(Y[i]-(i+1)*s);
                                        U+=(Y[i]-(i+1)*s);
        }


        /************* COUNT EXECUTION TIME *****************/

        target=choose_bound(x,a,L,U,N);

        start=clock();

        pardalos(x,a,b,target,N);

        stop=clock();

        duration = (long double) (stop-start)/CLOCKS_PER_SEC;

        /***************************************************/


        /******CONVERTING SOLUTION TO PROPER FORM**********/

        for (i=0;i<N;i++) Y[i]=x[i]+(i+1)*s;


        /***************************************************/


        /************** PRINT FINAL DATA *****************/

        print_final("num_pardalos.txt",N,Y,s);

        print_duration("num_pardalos.txt",duration);

        printf ("\nPardalos Total execution time=%f \n",duration);

        /***************************************************/

        return 0;
}

struct node *insert(struct node *p, int value1,int value2, int count)
  {
    if(p==NULL)
    {
      p=(struct node *)malloc(sizeof(struct node));
      if(p==NULL)
      {
    printf("Error\n");
          exit(0);
    }
    p-> data1 = value1;
    p-> data2 = value2;
    p-> index = count;
    p-> link = NULL;
  }
  else
      p->link = insert(p->link,value1,value2,count);/* the while loop replaced by
recursive call */
    return (p);
  }


void erase (struct node *p)
{
  struct node *temp=p;
  while(p != NULL)
  {
        temp = p;
        p = p->link;
        free(temp);
  }
p=NULL;

}


void fprintnode_list ( struct node *p )
{ FILE * fp;
   fp=fopen("num_sweep.txt","a");
         fprintf(fp,"\nActing on variables with indexes:");
    while (p!= NULL)
     {
  fprintf(fp,"\n%5d",(p-> data1)+1);

      p = p-> link;
    }

fprintf(fp,"\n");
fclose(fp);
}


void printnode_list ( struct node *p )
```

```
{
    printf("The data values in the list are");
    while (p!= NULL)
    {
                printf("\n%d\t",(p-> data1)+1);
      p = p-> link;
    }
}

float minimum(float x1 , float x2)
{float value;
value= (x1<x2) ? x1 : x2;
return value;
}

void bsort2(float list[][2], int n)
{
  int i,j;
  for(i=0;i<(n-1);i++)
    for(j=0;j<(n-(i+1));j++)
                if(list[j][0] < list[j+1][0])
                                    {
            swap(&list[j][0],&list[j+1][0]);
                                    swap(&list[j][1],&list[j+1][1]);
                                    }
}

void readlist(float list[],int n)
{
  int i;
  printf("Enter the elements\n");
  for(i=0;i<n;i++)
     scanf("%f",&list[i]);
}

float positive_minlist(float list[],struct node *front,int count,int n)
{
        float minl=MAXFLOW;

        struct node *temp2=front;

        while (temp2!= NULL)
                         {
                         if (list[temp2->data1]>0) minl = min(minl,list[temp2->data1]);
                         temp2 = temp2-> link;
                              }
  return minl;
}

float positive_mindist(float list[][2], struct node *front,int count, int n)
{
  float mind=MAXFLOW;
  struct node *temp2=front;

        while (temp2!= NULL)
                  {
                   if (list[temp2->data2][0]>0) mind = min(mind,list[temp2->data2][0]);
                         temp2 = temp2-> link;
             }
  return mind;
}

void printlist(float list[],int n)
{
  int i;
  for(i=0;i<n;i++)
    printf("%3d  %f\n",i+1,list[i]);
printf("\n");
}

void printlist2(float list[][2],int n1,int n2)
{
  int i,j;
  for(i=0;i<n1;i++)
                 {
                 printf("%3d   ",i+1);
                 for(j=0;j<n2;j++) printf("%f\t",list[i][j]);
                 printf("\n");
  }
}

int xleft_empty(float list[],int n)
{int i,result=0;
for (i=0;(i<n)&&(list[i]==0);i++);
result = (i==n) ? 1 : 0 ;
return result;
}

int dist_empty(float dist[][2],float list[],int n)
{int i,result=0;
for (i=0;(i<n)&&((dist[i][0]<=0)||(list[(int)(dist[i][1]-1)]==0));i++);
result = (i==n) ? 1 : 0 ;
return result;
}
```

```
struct node * next_dist(float xleft[],float dist[][2],int n, struct node *p, int *count , float *diff)
{
        int i,counter=0;
        float val;
        p=NULL;

        for (i=0;(xleft[(int)(dist[i][1]-1)]==0)&& (i<n);i++);
        p=insert(p,((int)dist[i][1]-1),i,counter+1);
        val=dist[i][0];
        counter=1;

        for (i=i+1; ((i<n) && ( ( dist[i][0]==val ) || ( xleft[(int)(dist[i][1]-1)]==0 ) )) ;i++)
                                if (( dist[i][0]==val )&&( xleft[(int)(dist[i][1]-1)]>0 ))
                                {

                                p=insert(p,((int)dist[i][1]-1),i,counter+1);

                                counter++;
                                };

*diff=(i==n)? MAXFLOW : (val-dist[i][0]);

*count=counter;
return p;
}

void update (float x[],float xleft[],float dist[][2] ,int n,struct node *front,int count,float xopt)

{
struct node *temp2=front;

        while (temp2!= NULL)
                {
                        x[temp2->data1]=x[temp2->data1]+xopt;

                        xleft[temp2->data1]=xleft[temp2->data1]-xopt;

                        dist[temp2->data2][0]=dist[temp2->data2][0]-xopt;
                        temp2 = temp2-> link;
                        };
}

void print_compare(char *name, int j, float diff, float dist[][2],float Xleft[],struct node
*front,int count,int n, float L,char* lim)
{
        FILE *fp;
        fp=fopen(name,"a");
        fprintf(fp,"Iteration %d\n",j);

        if (diff>MAXLIMIT) fprintf(fp,"diff1-diff2=+oo\n");
                                else fprintf(fp,"diff1-diff2=%10.5f;\n",diff);


        if (positive_mindist(dist,front,count,n)>MAXLIMIT) fprintf(fp,"diff1-line=+oo\n");
                                else fprintf(fp,"diff1-
line=%10.5f;\n",positive_mindist(dist,front,count,n));
        fprintf(fp,"Xleft(i)-Xi=%10.5f;\n",positive_minlist(Xleft,front,count,n));
        fprintf(fp,"(");
        fprintf(fp,lim);
        fprintf(fp,"-Sxi)/count=%10.5f;\n",L/count);
        fprintf(fp,"\n");

        fclose(fp);

        fprintnode_list(front);

}

void print_updated(char *name, float X[],float Xleft[],float Y[],float sum,int n)
{

        FILE *fp;
        int i;

        fp=fopen(name,"a");

        fprintf(fp,"\n");

        for (i=0;i<N;i++)
        fprintf(fp,"x%5d=%10.2f\txleft%5d=%10.2f\ty%5d=%10.2f\n",i+1,X[i],i+1,Xleft[i],i
+1,Y[i]-X[i]);

        fprintf(fp,"\n");


        fprintf(fp,"Sxi=%f;\n",sum);

        fprintf(fp,"\n");

        fclose(fp);
}

void print_final(char *name,int n,float Y[],float s)
{
        FILE *fp;
        int i;
        float z=0;

        fp=fopen(name,"a");

        fprintf(fp,"Optimal solution:\n");

        for (i=0;i<n;i++)      fprintf(fp,"y%5d=%10.5f\n",i+1,Y[i]);

        fprintf(fp,"\n");


        for (i=0;i<n;i++) z=z+(Y[i]-(i+1)*s)*(Y[i]-(i+1)*s);

        fprintf(fp,"N=%5d\nObjective optimum = %10.5f \n",N,z);


        fclose(fp);

}

void print_iterations(char *name,int j)
{
        FILE *fp;

        fp=fopen(name,"a");

        fprintf(fp,"Total iterations=%5d\n",j);

        fclose(fp);

}

void solve(float Y[],float Xleft[],int n,float s,float L,float U)
{
long double z=0;

int i,j=1,count=0;
float xopt,diff,sum=0;
float X[N];
float dist[N][2];
struct node *front=NULL,*rear=NULL;

for ( i = 0; i < n; i++ ) X[i]=0;

for ( i = 0; i < n; i++ )
        {
              dist[i][0]=Y[i]-(i+1)*s;
              dist[i][1]=i+1;
              }

        bsort2 (dist,n);

        for (;!((xleft_empty(Xleft,n))||(dist_empty(dist,Xleft,n))||(L<LIMIT));)
        {

        front = next_dist(Xleft, dist,n,front, &count ,&diff);


        xopt =
min(min((diff),positive_minlist(Xleft,front,count,n)),min((positive_mindist(dist,front,count,n)
),(L/count)));

        update (X,Xleft,dist,n,front,count,xopt);

        erase(front);

        U=U-count*xopt;
        L=L-count*xopt;

        sum+=count*xopt;

        j++;

        }

        if (dist_empty(dist,Xleft,n)) for (;!((xleft_empty(Xleft,n))||(L<LIMIT));)
        {
        {
        front=NULL;

        front = next_dist(Xleft, dist,n,front, &count ,&diff);

        xopt =
min(min((diff),positive_minlist(Xleft,front,count,n)),min((positive_mindist(dist,front,count,n)
),(L/count)));


        update (X,Xleft,dist,n,front,count,xopt);

        erase(front);

        U=U-count*xopt;
        L=L-count*xopt;
        sum+=count*xopt;
```

```
        j++;
        }
        if (xleft_empty(Xleft,n)) printf("\nNo feasible solution.\n");
        }

        else if (L<LIMIT) for
(;!((xleft_empty(Xleft,n))||(U<LIMIT)||(dist_empty(dist,Xleft,n)));)
        {

        front=NULL;


        front = next_dist(Xleft, dist,n,front, &count ,&diff);

        xopt =
min(min((diff),positive_minlist(Xleft,front,count,n)),min((positive_mindist(dist,front,count,n)
),(U/count)));


        update (X,Xleft,dist,n,front,count,xopt);

        erase(front);


        U=U-count*xopt;
        L=L-count*xopt;
        sum+=count*xopt;

        j++;
        }

        else printf("\nNo feasible solution.\n");

        for (i=0;i<N;i++) Y[i]=Y[i]-X[i];

        /**************FINAL****************/

        print_final("num_sweep.txt",N,Y,s);
        print_iterations("num_sweep.txt",j);


        /*************************************/

}

void print_ampl(char *name,int n,float Y[],float Xleft[],float s,float L,float U)
{
	FILE *fp;
	int i;

	fp=fopen(name,"w");

	fprintf(fp,"param N:=%d;\n\n",N);

	fprintf(fp,"param y1:=\n");

	for (i=0;i<N;i++)	fprintf(fp,"%d  %f\n",i+1,Y[i]);

	fprintf(fp,";\n");

	fprintf(fp,"param xleft:=\n");

	for (i=0;i<N;i++)	fprintf(fp,"%d  %f\n",i+1,Xleft[i]);


	fprintf(fp,";\n");

	fprintf(fp,"param s:=%f;\n",s);

	fprintf(fp,"param L:=%f;\n",L);

	fprintf(fp,"param U:=%f;\n",U);

	fclose(fp);
}
void print_original(char *name,int n,float Y[],float Xleft[],float s,float L,float U,float Xmax)
{
	FILE *fp;
	int i;

	fp=fopen(name,"w");

	fprintf(fp,"Xmax=%f;\n",Xmax);
	fprintf(fp,"LS=%f;\n",L);
	fprintf(fp,"US=%f;\n",U);
	fprintf(fp,"\n");

	for (i=0;i<n;i++)
	fprintf(fp,"y%5d=%10.5f\txleft%5d=%10.5f\t%5ds=%10.5f\n",i+1,Y[i],i+1,Xleft[i],i+
1,(i+1)*s);

	fprintf(fp,"\n");

	fclose(fp);
```

```
}

void print_duration(char *name,long double duration)
{
        FILE *fp;

        fp=fopen(name,"a");

        fprintf(fp,"Total Execution Time=%10.5f\n",duration);

        fclose(fp);

}


float choose_bound(float x[],float a[], float L, float U,int n)
{int i;
float bound[4];
bound[0]=L;
bound[1]=U;
bound[2]=0;
for (i=0;i<n;i++) bound[2]+=a[i];
bound[3]=0;
for (i=0;i<n;i++) bound[3]+=max(a[i],0);
bsort1(bound,4);
return bound[2];

}

void swap(float *x,float *y)
{
  float temp;
  temp = *x;
  *x = *y;
  *y = temp;
}

void bsort1(float list[], int n)
{
  int i,j;
  for(i=0;i<(n-1);i++)
    for(j=0;j<(n-(i+1));j++)
         if(list[j] > list[j+1])
              swap(&list[j],&list[j+1]);
}

void pardalos(float x[], float a[], float b[],float d,int n)
{
```

```
int *unsetv;
float *intervalpts;
float *temp1;
int *temp2;
float min=-(float)MAXFLOW,
          max=(float)MAXFLOW;
float tightsum=0,
          slackweight=0,
          testsum=0;
int i,j=1,counter;
float mid;
int pts_size;
int unsetv_size;

unsetv= (int *) malloc(n*sizeof(int));

intervalpts= (float *) malloc((2*n+2)*sizeof(float));

pts_size=2*n+2;
unsetv_size=n;


for (i=0;i<n;i++) unsetv[i]=(i+1);

for (i=0;i<n;i++) intervalpts[i]=a[i];
for (i=n;i<2*n;i++) intervalpts[i]=b[i-n];
intervalpts[2*n]=-(float)MAXFLOW;
intervalpts[2*n+1]=(float)MAXFLOW;

for (;(unsetv_size!=0);){

temp1 = (float *)malloc(pts_size*sizeof(float));
memcpy(temp1,intervalpts,pts_size*sizeof(float));

mid=qmedian(temp1,pts_size);

free (temp1);

testsum=0;
for (i=0;i<unsetv_size;i++) if (b[unsetv[i]-1]<mid) testsum+=b[unsetv[i]-1];
                                                            else if (a[unsetv[i]-1]>mid)
testsum+=a[unsetv[i]-1];
                                                            else testsum+=mid;


testsum=testsum+tightsum+slackweight*mid;
/*********UPDATE******************/

if (testsum<=d) min=mid;
if (testsum>=d) max=mid;

temp1 = (float *)malloc(pts_size*sizeof(float));
counter=0;

for (i=0;i<pts_size;i++) if ((intervalpts[i]>=min) && (intervalpts[i]<=max)) {

                              temp1[counter]=intervalpts[i];

                              counter++;

                                                            };
pts_size=counter;
free (intervalpts);
intervalpts = (float *)malloc(pts_size*sizeof(float));
memcpy(intervalpts,temp1,pts_size*sizeof(float));
free (temp1);

temp2 = (int *)malloc(unsetv_size*sizeof(float));
counter=0;

for (i=0;i<unsetv_size;i++) if (b[unsetv[i]-1]<=min) tightsum+=b[unsetv[i]-1];
                                                            else if (a[unsetv[i]-1]>=max)
tightsum+=a[unsetv[i]-1];
                                                            else if ((a[unsetv[i]-
1]<=min)&&(b[unsetv[i]-1]>=max)) slackweight++;
                                                            else {
                                                                          temp2[counter]=unsetv[i];
                                                                          counter++;
                                                                          };


                                                            unsetv_size=counter;


free(unsetv);
unsetv= (int *)malloc(unsetv_size*sizeof(float));
memcpy(unsetv,temp2,unsetv_size*sizeof(float));
free(temp2);


j++;

}

for (i=0;i<N;i++) if  (b[i]<=min)       x[i]=b[i];
```

```
                                                    else if (a[i]>=max) x[i]=a[i];
                                                    else if ((a[i]<=min)&&(b[i]>=max))
x[i]=(d-tightsum)/slackweight;


      /********************FINAL********************/


      print_iterations("num_pardalos.txt",j);

      /**************************************************/


}


float qmedian(float a[], int n)
{
  int low, high ;
  int median;
  int middle, ll, hh;

  low = 0 ; high = n-1 ; median = (low + high) / 2;
  for (;;) {
    if (high <= low) /* One element only */
      return a[median] ;

    if (high == low + 1) {  /* Two elements only */
      if (a[low] > a[high])
        SWAP(a[low], a[high]) ;
      return a[median] ;
    }

  /* Find median of low, middle and high items; swap into position low */
    middle = (low + high) / 2;
    if (a[middle] > a[high])    SWAP(a[middle], a[high]) ;
    if (a[low] > a[high])       SWAP(a[low], a[high]) ;
    if (a[middle] > a[low])     SWAP(a[middle], a[low]) ;

  /* Swap low item (now in position middle) into position (low+1) */
    SWAP(a[middle], a[low+1]) ;

  /* Nibble from each end towards middle, swapping items when stuck */
    ll = low + 1;
    hh = high;
    for (;;) {
      do ll++; while (a[low] > a[ll]) ;
      do hh--; while (a[hh]  > a[low]) ;
```

```
            if (hh < ll)
            break;

            SWAP(a[ll], a[hh]) ;
        }

        /* Swap middle item (in position low) back into correct position */
        SWAP(a[low], a[hh]) ;

        /* Re-set active partition */
        if (hh <= median)
            low = ll;
        if (hh >= median)
            high = hh - 1;
    }
}
#undef SWAP
```

Appendix B AMPL Modeling

```
# Parameters

param N;

param y1 {1..N};       # initial values of y

param xleft {1..N};    # initial values of Xleft

param L;                # lower limit on plan
param U;                # upper limit on plan

param s                # slope

# Decision variables


var x {1..N} >=0;       # flight time of aircraft
var y {1..N} ;          # residual flight time of aircraft

minimize convex1  : sum {n in 1..N}  (y[n]-n*s)^2;

subject to flight_hrs {n in 1..N}:
y[n] = y1[n] - x[n] ;    # residual flight time

subject to progr_hrs :
L <= sum {n in 1..N} x[n] <= U; # observe program

subject to upper_x {n in 1..N}:
x[n] <= xleft[n];  # maximum flight time
```

References

[1]. Ali, A., Helgason, R., Kennington, J., and Lall, H.: 'Computational comparison among three multicommodity network flow algorithms', Oper. Res.**28**, no. 4 (1980), 995–1000.

[2]. Bitran, G.R., and Hax, A.C.: 'Disaggregation and resource allocation using convex Knapsack problems with bounded variables', Managem. Sci.**27**, no. 4 (1981), 431–441.

[3]. Brucker, P.: 'An O(n) algorithm for Quadratic Knapsack problems', Oper. Res. Lett.**3**, no. 3 (1984), 163–166.

[4]. Floudas, C.A., and Visweswaran, V.: 'Quadratic optimization', Handbook Global Optim.: Nonconvex Optim. Appl., 2 Kluwer Acad. Publ. 1995, pp. 217–269.

[5]. Goldfarb, D., and Liu, S.: 'An O(n3L) primal interior point algorithm for convex Quadratic Programming', Math. Program. A**49**, no. 3 (1990/1), 325–340.

[6]. Horst, R., and Tuy, H.: Global optimization: Deterministic approaches, second Springer 1993.

[7]. Kojima, M., Mizuno, S., and Yoshise, A.: 'An $(O\left(\sqrt{n}L\right)$ iteration potential reduction algorithm for linear complementarity problems', Math. Program. A**50**, no. 3 (1991), 331–342.

[8]. Kozlov, M.K., Tarasov, S.P., and Khachiyan, LG.: 'Polynomial solvability of convex Quadratic Programming', Dokl. Akad. Nauk SSSR**248**, no. 5 (1979), 1049–1051.

[9]. Lin, Y.Y., and Pang, J.-S: 'Iterative methods for large convex Quadratic programs: A survey', SIAM J. Control Optim.25, no. 2 (1987), 383–411.

[10]. Markowitz, H.M.: 'Portfolio selection', Finance**7** (1952), 77–91.

[11]. Monteiro, R.D.C., Adler, I., Resende, M.G.C.: 'A polynomial-time primal-dual affine scaling algorithm for linear and convex Quadratic Programming and its power series extension', Math. Oper. Res.**15**, no. 2 (1990), 191–214.

[12]. Murty, K.G., and Kabadi, S.N.: 'Some NP-complete problems in Quadratic and nonlinear programming', Math. Program.**39**, no. 2 (1987), 117–129.

[13].PARDALOS, P.M., AND KOVOOR, N.: 'An algorithm for a singly constrained class of Quadratic programs subject to upper and lower bounds', Math. Program. A**46**, no. 3 (1990), 321–328.

[14].PARDALOS, P.M., AND ROSEN, J.B.: 'Constrained global optimization: Algorithms and applications', Vol. 268 of Lecture Notes Computer Sci., Springer 1987.

[15].PARDALOS, P.M., YE, Y., AND HAN, CHI-GEUN: 'Algorithms for the solution of Quadratic Knapsack problems', Linear Alg. & Its Appl.**152** (1991), 69–91.

[16].SAHNI, S.: 'Computationally related problems', SIAM J. Comput.**3** (1974), 262–279.

[17].VAVASIS, S.A.: 'Approximation algorithms for indefinite Quadratic Programming', Math. Program. B**57**, no. 2 (1992), 279–311.

[18].VAVASIS, S.A.: 'Local minima for indefinite Quadratic Knapsack problems', Math. Program. A**54**, no. 2 (1992), 127–153.

[19] 'W. KARUSH, 'Minima of Functions of Several Variables with Inequalities as Side Conditions,' M.S. thesis, Department of Mathematics, University of Chicago, 1939.

[20] H. W. KUHN AND A. W. TUCKER, 'Nonlinear Programming,' in Jerzy Neyman (ed.), Proceedings of the Second Berkeley Symposium, University of California Press, Berkeley, 1951, pp. 481-492.

[21] O. T. MANGASARIAN, 'Nonlinear Programming', McGraw-Hill, New York, 1969

[22] A. M. GEOFFRION, 'Duality in Nonlinear Programming: A Simplified Applications-Oriented Development,' SIAM Review, 13: 1-37, 1971.

[23] P. WOLFE, 'The Simplex Method for Quadratic Programming,' Econometrics, 27: 382-398, 1959.

[24] D. BERTSIMAS, C. DARNELL, AND R. SOUCY. 'Portfolio construction through mixed-integer programming' at Grantham, Mayo, Van Otterloo and Company. Interfaces, 29(f):49-66, 1999.

[25] A. CADENILLAS AND S. R. PLISKA. 'Optimal trading of a security when there are taxes and transaction costs'. Finance and Stochastics, 3:137-165, 1999.

[26] H. MARKOWITZ. 'Portfolio selection: efficient diversification of investments.' Blackwell, New York, 2nd edition, 1991.

[27] J. NIEHANS. 'Arbitrage equilibrium with transaction costs'. Journal of Money, Credit, and Banking, 26(2):249-270, 1994.

[28] M. H. SCHNEIDER AND S. A. ZENIOS. 'A comparative study of algorithms for matrix balancing', Operations Research, 38: 439-455, 1990.

[29] Y. CENSOR AND S. A. ZENIOS. 'Interval-constrained matrix balancing', Linear Algebra and its Applications, 150, 393-421, 1991.

[30] M. L. BALINSKI AND G. DEMANGE. 'Algorithms for proportional matrices in reals and integers', Mathematical Programming, 45, 193-210, 1989.

[31] J.KRARUP AND T.ILLES (1993), 'Maximum Q-free bipartite graphs and Knapsack-type programs', DIKU, University of Copenhagen, Denmark, Report 93/28.

[32] G.LAPORTE (1992), 'The Vehicle Routing Problem: An overview of exact and approximate algorithms', European Journal of Operational Research, **59,** 345 358.

[33] S.MARTELLO AND P.TOTH (1984), 'A mixture of dynamic programming and branch-and-bound for the subset-sum problem', Management Science, **30,** 765 771.

[34] G.B.MATHEWS (1897), 'On the Partition of Numbers', Proceedings of the London Mathematical Society, 28, 486-490.

[35] R. M. NAUSS (1978),'The 0-1 Knapsack problem with multiple choice constraint', European Journal of Operational Research, 2, 125-131.

[36] W.DIFFE AND M.E.HELLMAN (1976), 'New directions in cryptography', IEEE Trans. Inf. Theory, IT-36, 644-654.

[37] A. SINHA AND A. A. ZOLTNERS (1979), 'The multiple-choice Knapsack problem', Operations Research, 27, 503-515.

[38] R. K. AHUJA, T. L. MAGNANTI, AND J. B. ORLIN. Network Flows:'Theory, Algorithms, and Applications'. Prentice Hall, Englewood Cliffs, NJ, 1993.

[39] Z. SINUANY-STERN AND I.WINER (1994), 'The one dimensional cutting stock problem using two objectives', Journal of the Operational Research Society, **45,** 231-236.

[40] HILLIER ,LIEBERMANN, 'Introduction to Operations Research', McGraw Hill 7th Ed..

[41] CHRISTODOULOS A. FLOUDAS AND PANOS M. PARDALOS, 'Encyclopedia of Optimization.' Kluwer Academic Publishers 2001.

[42] D.PISINGER, 'Algorithms for Knapsack Problems'. Ph.D. Thesis Feb 1995 , Dept.of Comp.Science, University of Copenhagen.

[43] A.NEUMAIER, 'An Optimality Criterion for Global Quadratic Optimization'. Journal of Global Optimization 2:201-208,1992, Kluwer Academic Publishers.