Relaxation and Exact Algorithms for Solving Mixed Integer-Quadratic Optimization Problems

by

Constantine Nikolaos Tziligakis

Submitted to the Operations Research Center in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1999

© Massachusetts Institute of Technology 1999. All rights reserved.

Author ........................................................................................................

Operations Research Center

August 6, 1999

Certified by ............................................................... Dimitris Bertsimas

Boeing Professor of Operations Research, Sloan School of Management and Operations Research Center

Thesis Supervisor

Accepted by ............................................................... James Orlin

Co-Chairman, Operations Research Center
Relaxation and Exact Algorithms for Solving Mixed Integer-Quadratic Optimization Problems

by

Constantine Nikolaos Tziligakis

Submitted to the Operations Research Center on August 6, 1999, in partial fulfillment of the requirements for the degree of Master of Science

Abstract

We develop various algorithms for solving mixed integer-quadratic problems. These problems exhibit exponential complexity resulting from the presence of integer variables. Traditional approaches that apply in pure integer programming are not very helpful, since the existence of continuous variables in our problems complicates their use.

We develop relaxation and heuristic algorithms designed so as to provide tight lower and upper bounds to the optimal solution of the mixed combinatorial problem. In some cases the obtained range, in which the optimum lies, is small enough to be considered satisfactory by itself. This has been accomplished in problems with up to 150 variables.

Exact algorithms have also been developed and guarantee the optimal solution upon termination. The idea of Branch and Bound enhanced with the use of lower and upper bounds obtained with the aforementioned methods is implemented for that purpose. Problems with up to 70 variables have been solved.

Our ideas and algorithms are applied to the Problem of Index and Portfolio Replication with a limited number of assets. This problem arises in Finance, but, in its more general form, can find application in various areas ranging from Statistics to Optimal Control and Manufacturing.

Thesis Supervisor: Dimitris Bertsimas
Title: Boeing Professor of Operations Research, Sloan School of Management and Operations Research Center
Acknowledgments

I would like to thank my supervisor, Professor Dimitris Bertsimas, for the very interesting topic and his continuous support throughout the development of this thesis. His insightful comments and ideas are greatly appreciated.

This research was partially supported by a Grant from Merrill Lynch & Co. I would like to thank Mr. Joe Argilagos, Dr. Jim Gatheral and Mr. Tim Clorite, in the Equity Derivatives and the Global Portfolio Trading Group for providing useful comments and for giving me the opportunity to spend two summers at the firm's premises in New York. Many thanks also go to Ed Somekh, Dr. Peter Li, Jerry Weiss and Max Roberts in the Latin America Capital Markets and Credit Derivatives Desks.

Special thanks go to the Operations Research Center and the Sloan School of Management administrative and support personnel. In particular, Mrs. Paulette Mosley, Mrs. Laura Rose, Ms. Danielle Bonaventura and Mr. Ray Faith, for all their help and attention throughout my studies at the Operations Research Center.

I should not forget to thank Ramazan Demir, Geoffrey Lauprete, Eric Marandon, Dr. Ioana Popescu and Dr. Jay Sethuraman, all current or ex-students at the Operations Research Center, for their help, nice discussions and jokes.

Many special thanks go to my wonderful friends in Boston and in New York for their support and the unforgettable memories. I feel the need to mention, in particular, Isabelle Parfait and my roommate and friend Lefteris Iliopoulos.

This work is dedicated to my parents Nikolaos and Georgia and my brother Yannis to whom I am grateful for their support and love.
Contents

1 Introduction ................................. 14
  1.1 General Problem Description ................. 14
  1.2 Previous Work ........................... 15
  1.3 General Remarks on Algorithms ............... 16
  1.4 Portfolio Problem Formulation ............... 18

2 Relaxation Algorithms ..................... 21
  2.1 Positive Semidefinite Algorithms .......... 21
    2.1.1 Interior Point Methods ................. 26
    2.1.2 The Eigenvalue-Cut Method .............. 39
  2.2 Quadratic Positive Semidefinite Algorithms 44

3 Heuristic Algorithms ...................... 52
  3.1 Previous work ........................... 53
  3.2 Randomization Heuristics ................. 53
    3.2.1 A Simple Randomization Heuristic ....... 54
    3.2.2 M-Randomization Heuristic .............. 56
    3.2.3 Positive Semidefinite Randomization Heuristic 58
  3.3 Greedy Heuristics ....................... 62
    3.3.1 Simple Bottom-Up and Top-Down Greedy Heuristic 62
    3.3.2 Greedy-Square ($G^2$) .................. 64
    3.3.3 Greedy-Cube ($G^3$) .................... 66
  3.4 Local Search Heuristics ................... 67
3.5 Summary .................................................... 71

4 Exact Algorithms ........................................... 75
4.1 Branch and Bound ........................................ 75
4.2 The Use of Bounds in Branch and Bound ............... 81
4.3 The Portfolio Problem - Branch and Bound Results .... 84
   4.3.1 Branch and Bound with the Relaxed Version ...... 88
   4.3.2 Branch and Bound with Positive Semidefinite Relaxation ... 90

5 Evaluation and Comparison of Approximate and Exact Algorithms 93
5.1 Relaxation Algorithms - Lower Bounds ................ 93
5.2 Feasible Solution Algorithms - Heuristics and Branch and Bound ... 97
5.3 Summary ..................................................... 100

6 Conclusions and Future Work .............................. 104

Bibliography .................................................. 107
List of Figures

3-1 Algorithmic performance of the heuristics in the case of \( x'Qx \) problems. All problem sizes from 10 to 100 variables are shown. We include the Simple Randomization Heuristic (SRH), M-Randomization Heuristic (M-RH), Positive Semidefinite Heuristic (PSDH), Greedy-Square (G2), Greedy-Cube (G3) and the Simulated Annealing Heuristic (SAH). 73

3-2 Algorithmic performance of the heuristics in the case of \( z'Wz \) problems. All problem sizes from 10 to 100 variables are shown. We include the Simple Randomization Heuristic (SRH), M-Randomization Heuristic (M-RH), Positive Semidefinite Heuristic (PSDH), Greedy-Square (G2), Greedy-Cube (G3) and the Simulated Annealing Heuristic (SAH). 74

4-1 Schematic of the Branch and Bound method. All nonfixed variables are assumed to be relaxed. ................................................. 79
List of Tables

2.1 Fixed binary variables and their values for fixed-variable relaxation algorithm runs in Chapter 2. .......................................................... 30

2.2 Lower bounds and running times for the Relaxed Formulation when no binary variables are fixed. A star(*) denotes that for the corresponding problem $Z_{RQ}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{RQ}/Z_{OPT}$ is larger than or equal to the “starred” numbers shown in the Table. .......................... 31

2.3 Lower bounds and running times for the Relaxed Formulation when the binary variables of Table 2.1 are fixed at their {0,1} values. The relaxation is run for the remaining variables. A star(*) denotes that for the corresponding problem $Z_{RQ}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{RQ}/Z_{OPT}$ is larger than or equal to the “starred” numbers shown in the Table. .......................................................... 32
2.4 Lower bounds and running times for the (Linear) Positive Semidefinite Formulation when no binary variables are fixed. Solution by SDPA (interior-point method) with accuracy $10^{-6}$. A star(*) denotes that for the corresponding problem $Z_{PSD}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{PSD}/Z_{OPT}$ is larger than or equal to the "starred" numbers shown in the Table. .......................................................... 34

2.5 Lower bounds and running times for the (Linear) Positive Semidefinite Formulation when no binary variables are fixed and mixed constraints 2.26 have been replaced by $\alpha_i Z_{ii} \leq x_i \leq Z_{ii}$. Furthermore, constraints $Z_{ij} \geq 0$ have been replaced by $Z_{ii} \geq 0$. Solution by SDPA (interior-point method) with accuracy $10^{-6}$. A star(*) denotes that for the corresponding problem $Z_{PSD}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{PSD}/Z_{OPT}$ is larger than or equal to the "starred" numbers shown in the Table. .......................................................... 36

2.6 Lower bounds and running times for the (Linear) Positive Semidefinite Formulation when the binary variables of Table 2.1 are fixed to their {0,1} values. Relaxation is run for the remaining variables. A $Z_{PSD}/Z_{OPT}$ number greater than 100% means that pruning would occur at that node in branch and bound. Solution by SDPA (interior-point method) with accuracy $10^{-6}$. A star(*) denotes that for the corresponding problem $Z_{PSD}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{PSD}/Z_{OPT}$ is larger than or equal to the "starred" numbers shown in the Table. .......................................................... 38
2.7 Lower bounds and running times for the (Linear) Positive Semidefinite Formulation in \(x'Qx\) problems when no binary variables are fixed. Solution by Eigenvalue-Cut Method. ........................................... 42

2.8 Demonstration of the effect of setting a fixed number of cuts to be maintained in the Eigenvalue-Cut method in the case of the \(x'Qx\) problem with \(N = 10\) and \(M = 4\). Used tolerance \(10^{-3}\). .......................... 43

2.9 Lower bounds and running times for the Quadratic Positive Semidefinite Formulation in \(x'Qx + z'Wz\) problems when no binary variables are fixed. Algorithm stops after 200 iterations. ..................... 50

2.10 Lower bounds and running times for the Quadratic Positive Semidefinite Formulation when the binary variables of Table 2.1 are fixed to their \(\{0,1\}\) values. The relaxation is run for the remaining variables. A \(Z_{QPSD}/Z_{OPT}\) number greater than 100% means that pruning would occur at that node in branch and bound. The algorithm stops after 200 iterations. .................................................. 51

3.1 Sample Upper bound results from Simple Randomization Heuristic. Number of generated samples equals \(5N\), where \(N\) is the number of binary variables in the problem. A star(*) denotes that for the corresponding problem \(Z_{SRH}\) was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, \(Z_{SRH}/Z_{OPT} – 1\) is larger than or equal to the "starred" numbers shown in the Table. .................. 56
3.2 Sample Upper bound results from M-Randomization Heuristic. Number of generated samples equals $5N$, where $N$ is the number of binary variables in the problem. A star (*) denotes that for the corresponding problem $Z_{M-RH}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{M-RH}/Z_{OPT} - 1$ is larger than or equal to the “starred” numbers shown in the Table.

3.3 Sample Upper bound results from Positive Semidefinite Randomization Heuristic using the SDPA algorithm. The number of generated samples equals $5N$, where $N$ is the number of binary variables in the problem. The solution accuracy is $10^{-4}$. A star (*) denotes that for the corresponding problem $Z_{PSDH}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{PSDH}/Z_{OPT} - 1$ is larger than or equal to the “starred” numbers shown in the Table.

3.4 Sample Upper bound results from Positive Semidefinite Randomization Heuristic with the Eigenvalue-Cut method. A star (*) denotes that for the corresponding problem $Z_{PSDH}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{PSDH}/Z_{OPT} - 1$ is larger than or equal to the “starred” numbers shown in the Table.
3.5 Sample Upper bound results for the Simple Bottom-Up Greedy Heuristic. In parentheses in the last column we show whether the best bound has come from the relaxed (r) or the fixed (f) version of the heuristic. A star(*) denotes that for the corresponding problem $Z_{BUGH}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{BUGH}/Z_{OPT} - 1$ is larger than or equal to the "starred" numbers shown in the Table. 

3.6 Sample Upper bound results for Greedy-Square Heuristic (Relaxed Version). A star(*) denotes that for the corresponding problem $Z_{G2r}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{G2r}/Z_{OPT} - 1$ is larger than or equal to the "starred" numbers shown in the Table. 

3.7 Sample Upper bound results for Greedy-Square Heuristic (Fixed Version). A star(*) denotes that for the corresponding problem $Z_{G2f}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{G2f}/Z_{OPT} - 1$ is larger than or equal to the "starred" numbers shown in the Table. 

3.8 Sample Upper bound results for Greedy-Cube Heuristic (Relaxed Version). A star(*) denotes that for the corresponding problem $Z_{G3}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{G3}/Z_{OPT} - 1$ is larger than or equal to the "starred" numbers shown in the Table.
3.9 Sample Upper bound results for the Simulated Annealing Heuristic. The starting sample is randomly generated. The number of temperature reductions is set to 100, the number of repetitions at the same temperature to $2M$, where $M$ is the number of “1” binary variables in the problem and $\alpha = 0.95$. A star(*) denotes that for the corresponding problem $Z_{SAH}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{SAH}/Z_{OPT} - 1$ is larger than or equal to the “starred” numbers shown in the Table.

4.1 Total number of nodes and estimated running times in Branch and Bound assuming that no pruning takes place.

4.2 Total number of nodes and estimated running times in Branch and Bound assuming that no pruning takes place.

4.3 Branch and Bound with use of the simple relaxed formulation. Exact solution version.

4.4 Branch and Bound with use of the simple relaxed formulation. $\epsilon = 0.1$ solution version.

4.5 Branch and Bound with use of the positive semidefinite algorithm SDPA. Exact solution for “$z^T W z$” and 0.1-approximate solution for “$x^T Q x$” problems.

5.1 Results for solution accuracy and running time from relaxation algorithms in “$x^T Q x$” problems. By solution accuracy we refer to the ratio $LB/Optimum$ whenever the optimal solution is known, or $LB/UB$ whenever only an upper bound is available ($LB$ stands for the lower bound estimate and $UB$ for the best upper bound available). In the second case, a star(*) is added before the entries above. Both non-fixed and fixed variable cases considered.
5.2 Results for solution accuracy and running time from relaxation algorithms for $z'Wz$ problems. By solution accuracy we refer to the ratio $LB/\text{Optimum}$ (LB denotes the Lower Bound). Note that in all problems above the exact solution is known from Branch and Bound.

5.3 Results for solution accuracy and running time from heuristic algorithms and branch and bound for $x'Qx$ problems. Accuracy is defined as the ratio $(HUB/\text{optimum})-1$ ($HUB$ stands for Heuristic Upper Bound), wherever an optimal solution is known and $(HUB/UB)-1$, when the optimal solution is not known ($UB$ stands for the best available Upper Bound). In the second case, a star(*) is added in front of the entries above. The sign "~" means "less than $10^{-3}\%$".

5.4 Results for solution accuracy and running time from heuristic algorithms and branch and bound for $z'Wz$ problems. Accuracy is defined as the ratio $(HUB/\text{optimum})-1$ ($HUB$ stands for Heuristic Upper Bound). Note that for all problems in this Table the optimal solution is known from Branch and Bound. The sign "~" means "less than $10^{-3}\%$".

5.5 Comparison of relaxation algorithms of this thesis. A scale from 1 to 4 is used, where 1 means best performer.

5.6 Comparison of heuristic algorithms in this thesis. A scale from 1 to 5 is used, where 1 means best performer.
Chapter 1

Introduction

Mixed integer-quadratic optimization is a relatively untouched area of research in the operations research community. It involves problems where one seeks to optimize a quadratic cost function with respect to both continuous- and integer-valued variables. Both kinds of variables can be present in the cost function and at the same time there is a system of constraints that needs to be satisfied. In this work, we develop and evaluate various algorithms for solving mixed integer-quadratic programs both approximately and exactly.

1.1 General Problem Description

A mixed integer-quadratic optimization problem is formulated in general as follows:

\[
\begin{align*}
\text{Minimize} \quad & x'Qx + z'Wz + c'_x x + c'_z z \\
\text{subject to} \quad & Ax + Bz \leq d \\
& x \geq 0, z \in \{0, 1\}^N.
\end{align*}
\] (1.1)

In the above, \(x \in \mathbb{R}_+^N\) is a vector of continuous non-negative variables and \(z \in \{0, 1\}^N\) a vector of binary variables, \(Q\) and \(W\) are \(N \times N\) positive definite matrices, \(A\) and \(B\) are \(M \times N\) matrices and \(c_x, c_z\) are \(N\)-vectors. The presence of integer variables
makes Problem (1.1) very hard to solve.

1.2 Previous Work

While quite a lot of work has been devoted in the last decade on methods for solving mixed integer-quadratic programs, the main focus has been on problems with only binary variables in the objective and the constraints. In graph equipartition problems, for example, where one seeks to partition the nodes of a graph into two sets $S_1, S_2$ of equal cardinality so that the total weight of one of the two sets $S_1$ is maximized, there is a single constraint. Researchers have considered a series of problems including, among others, quadratic versions of the knapsack problem and various max-cut and graph partition problems. In all cases, the integer variables are binary in that they can assume only one of two values, $\{0,1\}$ or $\{-1, +1\}$. Capitalizing mainly on ideas from mixed integer programming with a linear objective function, researchers have developed various algorithms for handling purely integer quadratic problems. The prevalent method is branch and bound. Attempts to improve its performance include introducing cuts based on the underlying polyhedral structure of the problem and its constraints [$30, 12, 33, 25, 34, 2, 19$], as well as positive semidefinite relaxations ([25, 34, 2, 32, 21, 19] and references therein.) In the positive semidefinite approach, a considerable amount of effort has been devoted into coming up with approximate solutions which are within a certain percentage of the optimal one [$17, 14, 28, 37, 38$]. Various heuristics based predominantly on randomization schemes have also been considered and have led to close form bounds in certain cases. See [17, 6] for example.

To the best of our knowledge, the only attempt to solve a problem similar to (1.1), has been Bienstock [$7$]. Bienstock is addressing the problem of quadratic programming problems where there is an upper bound on the number of positive variables. The problem is similar to the one considered in this thesis, but the solution approach is different. Bienstock does not introduce $\{0, 1\}$ variables but develops a branch and cut algorithm to find the solution. Although the results are promising, the author asserts that there is still much work to be done to deal with larger problems and flat
objectives. We will refer again to Bienstock’s work in Chapter 4 when we present our results for a similar problem.

1.3 General Remarks on Algorithms

The introduction of continuous variables in our case adds considerably to the practical usefulness and applicability of mixed integer-quadratic programming. Various problems that arise in quite diverse areas, from Finance and Statistics to Optimal Control and Product Manufacturing, involve a quadratic objective with respect to both continuous and binary variables. These control variables can appear both in the objective and the constraints, and they usually correspond to a decision of “to do or not to do”, “to include or not to include”, etc. More specifically we will consider two different formulations that find application in Finance, Statistics and graph equipartition problems.

The first type of problems will include only continuous variables $x$ in the objective in the form of quadratic and linear terms. Binary variables are present in the constraints only. This is the index and portfolio tracking problem with a limited number of names and corresponds to a real situation in the Asset Management industry. It can also find applications in Statistics in a multifactor regression model, for example, when one wants to identify the optimal subset of regressors to use among a large number of them. The portfolio problem will be formulated in the following section and from now on we will denote this problem by $x'Qx$.

The second type of problem is more general and includes $z$ terms also in the objective. We construct problems with quadratic terms in $z$. Such problems can find application in graph equipartition problems, in optimal control and manufacturing when, for example, a binary decision implies a fixed cost or benefit, or even in an enhanced version of the Portfolio problem. We will denote this problem as $x'Qx + z'Wz$ or just $z'Wz$ from now on and throughout the thesis. In the next section we mention some ways in which the portfolio problem can include such terms in the objective.
The path we follow in this work for developing algorithms relies heavily on the realization that quadratic combinatorial problems are not only characterized by increased complexity. The possibility of a “flat” quadratic objective is a very real one and can render exact algorithmic approaches such as branch and bound quite inefficient. By “flat” objective we mean that there exist many suboptimal solutions with costs very close to the optimal one. Since branch and bound is by far the most trustworthy method to guarantee the optimal solution to combinatorial problems, the focus has to be in developing approximation algorithms that can provide tight lower bounds to the minimization problem. Some heuristics can be devised, that quickly come up with upper bounds. Such approximation algorithms have to run fast and provide bounds as tight as possible. Used in conjunction with branch and bound, these could be quite effective in leading to the optimal solution in a reasonable amount of time.

In Chapter 2 we outline some approximation algorithms based on positive semidefinite relaxations of the original problem. The presentation is rather general in the beginning and refers to Problem (1.1) providing comments on the bounds. Computational results follow for the two types of problems we are considering. The same pattern is followed in Chapter 3 where heuristic algorithms are presented. They are based mainly on greedy search and variations, local search and various randomization schemes. In Chapter 4 we describe how relaxation and heuristic algorithms from Chapters 2 and 3 can be utilized with a branch and bound scheme to find the exact solution to the problem. After a general discussion we focus on the problems of interest in this thesis. In Chapter 5, we follow with a general evaluation and comparison of the various approaches and wrap up our computational results. Finally, in the last chapter we conclude and provide some directions for future work in the subject.

We implement our models on a SUN Sparc 20 computer with 128MB of main memory. For linearly-constrained quadratic programs and linear programming we use CPLEX Version 4.0.9, a commercially available software optimization package available by CPLEX Optimization, Inc. The SDPA positive semidefinite program (Version 4.10) of Fujisawa, Kojima and Nakata at the Tokyo Institute of Technology [15] is used to solve certain relaxation formulations in Chapter 2. We have written
our programs in C and C++ for efficiency and compatibility with the CPLEX and SDPA libraries.

1.4 Portfolio Problem Formulation

The problem we consider consists of replicating a financial index or "target" portfolio by using a subset of the assets involved in its composition. The objective is to find the subset that minimizes tracking error and transaction costs. Various linear constraints can be added to the problem corresponding to specific sector, liquidity or other requirements. The tracking error term is by construction quadratic in the portfolio asset weights and the transactions cost term can be either linear or quadratic. The combinatorial (binary) "to include" or "not to include" structure of the problem is apparent. Also note that the variables to be solved for, namely, the tracking portfolio weights are continuous variables. The problem belongs to the class of mixed integer-quadratic problems (1.1).

Consider a financial index consisting of $N$ names with weights $w_i, i = 1, \ldots, N$ such that $\sum_{i=1}^{N} w_i = 1$. Assume that one wants to replicate the index as close as possible by forming a new portfolio of $M < N$ names. The objective is to identify which $M$ assets should one use and with what weights $x_i, i = 1, \ldots, M$, where $\sum_{i=1}^{N} x_i = 1$. Ignoring transaction costs for the moment we want to minimize the expected tracking error. This is formulated as follows.

Denote by $s$ the $N$ securities return vector. The return on the index is a random variable and is expressed as $r_I = \sum_{i=1}^{N} w_i s_i$. The replicating portfolio return is also the random variable $r_P = \sum_{i=1}^{M} x_i s_i$. One can extend vector $x$ with zeros for the assets not included in the tracking portfolio $P$. The expected tracking error $TE$ is then written in vector notation as:

$$TE = E\{(r_I - r_P)'(r_I - r_P)\} = E\{(w - x)'ss'(w - x)\} = (w - x)'Q(w - x).$$

$q = E\{ss'\}$ is the $N \times N$ asset covariance matrix and it can be calculated using historical data, single- or multi-factor models, or GARCH models [10]. It is assumed
given for our purposes. The tracking error is a quadratic function of the portfolio weights \( x \). One obvious constraint is the so-called "budget" constraint \( \sum x_i = 1 \). To account for the combinatorial nature of the problem, one introduces \( \{0,1\} \) variables \( z_i, i = 1, \ldots, N \). If \( z_i = 0 \), then asset \( i \) is not included in the replicating portfolio. The opposite is true when \( z_i = 1 \). An obvious constraint is then the maximum number of names constraint, which is written as \( \sum z_i = M \) (cardinality constraint). The problem is formulated as shown below:

\[
Z_{OPT} = \text{Minimize} \quad x'Qx + c'_x x \\
\text{subject to} \quad \sum x_i = 1 \quad \text{(1.3)} \\
\sum z_i = M \quad \text{(1.4)} \\
\alpha_i z_i \leq x_i \leq z_i, i = 1, \ldots, N \quad \text{(1.5)} \\
z \in \{0, 1\}^N.
\]

In the above we have expanded the objective in \( x \) and discarded the constant term \( w'Qw \). Note, that \( c_x = -2w'Q \). We have introduced the mixed constraints (1.5), that basically force \( x_i \) to be zero when the asset is not included, or lie in \([\alpha_i, 1]\) when the asset is included. The positive quantity \( \alpha_i < 1 \) has the meaning of a minimum weight requirement for asset \( i \). If asset \( i \) is to be included in the portfolio it has to contribute at least as high as \( \alpha_i \% \) in the composition. In practical terms, one does not want to incur the cost of adding a name in the portfolio that does not represent an appreciable portion of it. A similar restriction could be introduced for the maximum value \( x_i \) can take, but we do not feel this is necessary. Note, that the vector \( \alpha \) is specified by the user. \( Q \) is symmetric and positive definite so a solution to the above problem exists and is unique.

In problem (1.2) the binary variables do not appear in the objective explicitly. This can happen if one extends the problem to include fixed costs associated with the assets included and/or penalize including a larger number of names in the tracking portfolio. Quadratic and/or linear terms in \( z \) can then appear in the objective (parenthesis term in (1.2)). Further extensions would include a sector exposure term and a
return term. In the first case, one specifies different groups of assets (sectors) in the “target” portfolio and requires that the tracking portfolio has exposure to all different sectors that is close to the “target” portfolio’s exposure. In the second case, low expected return tracking portfolios are penalized. Additional terms in the objective can be introduced as piecewise-linear or quadratic transaction costs (as functions of $x$). Liquidity considerations can also be taken into account. Bertsimas et al. [4] consider the portfolio tracking problem through mixed integer linear programming. They model tracking error as the piecewise linear term $\sum_{i=1}^{N} |w_i - x_i|$. The reader is referred to their work for more information regarding various extensions to the problem. All of these extensions add linear or quadratic terms of $x$ and/or $z$ in the objective. Alternatively, some of them can be formulated as additional linear constraints.

We focus on the simple version (1.2), where we will only add a simple quadratic cost term in $x$. We will also examine problems where a quadratic term in $z$ is present in the objective. It turns out that in many cases, this additional “structure” is desirable, as for example in relaxation algorithms where stronger lower bounds may be obtained, or in Branch and Bound where smaller enumeration trees result. It is worth mentioning, that Problem (1.2) can find applications in many other fields as well. Whenever a decision has to be made, that concerns meeting certain objectives as closely as possible by using a limited number of resources, Problem (1.2) applies.
Chapter 2

Relaxation Algorithms

In this Chapter we present some algorithms designed to provide approximate solutions to Problem (1.1). These are mainly based on semidefinite relaxations, that aim to address the combinatorial part posed by the presence of the \{0,1\} variables. These formulations attempt to recast (1.1) as a linear programming problem by increasing the number of the variables and posing the positive semidefiniteness requirement on a symmetric matrix. If the formulation is strong enough, good lower bounds to the minimization problem can be calculated. Randomization heuristics can also be employed at the end to provide strictly feasible solutions close to the optimal one. In each section of this chapter the discussion is rather general in the beginning referring to Problem (1.1). Then it becomes more specific and addresses the specific Problem (1.2). Computational results for each method are also presented.

2.1 Positive Semidefinite Algorithms

We begin by restating Problem (1.1) in a slightly different form.

\[
Z_{OPT} = \text{Minimize} \quad x'Qx + z'Wz + c'x + c'z \\
\text{subject to} \quad a'_ix \leq h_i, i = 1, \ldots, m_x \\
b'_jz \leq r_j, j = 1, \ldots, m_z \\
c'_iz + g'_lz \leq d_l, l = 1, \ldots, m_{xz}
\]  

(2.1)  
(2.2)  
(2.3)  
(2.4)
\[ x \geq 0, z \in \{0, 1\}^N. \]

Matrices \( Q \) and \( W \) are symmetric positive semidefinite. We assume that \( a_i \geq 0, i = 1, \ldots, m_x \) and \( b_i \geq 0, i = 1, \ldots, m_z \). One can start by relaxing the \( \{0,1\} \) variables, so that they can take any value in the interval [0,1]. The problem is then a linearly constrained quadratic problem and can be easily solved by standard methods.

\[
Z_{RQ} = \text{Minimize} \quad x'Qx + z'Wz + c'_x x + c'_z z \\
\text{subject to} \quad a'_ix \leq h_i, i = 1, \ldots, m_x \\
\quad b'_jz \leq r_j, j = 1, \ldots, m_z \\
\quad c'_lx + g'_lz \leq d_l, l = 1, \ldots, m_{xz} \\
\quad x \geq 0, z \in [0, 1]^N. \tag{2.5}
\]

Denote by \( S \) the polyhedron defined by constraints (2.2) - (2.4) above. Let \( P = \{x \in R_x^N, z \in [0, 1]^N : x, z \in S\} \) and let \( P_0 = \text{conv}(P \cap \{0, 1\}^N) \) denote the convex hull of 0-1 solutions. Then clearly \( P_0 \subseteq P \) and therefore \( Z_{RQ} \leq Z_{OPT} \). Formulation (2.5) provides a lower bound to (2.1), but this bound is quite weak in general.

An alternative way to form a relaxed version of (1.1) is to linearize the quadratic cost function by switching to a higher-dimension "quadratic space." One introduces variables \( X_{ij} = x_ix_j \) and \( Z_{ij} = z_iz_j \) for \( i \leq j, i, j = 1, \ldots, N \). These variables are used in place of the quadratic products in the objective function. Note, that the \( z_i \in \{0, 1\}, i = 1, \ldots, N \) constraint is equivalent to \( z_iz_i = z_i \) and therefore one can replace \( z_i \) with \( Z_{ii} \) in the cost function and the constraints. Also note, that the matrix solutions \( X \) and \( Z \) are symmetric and positive semidefinite, since \( X = xx' \) and \( Z = zz' \). Although the \( \{0,1\} \) constraint effectively disappears, the positive semidefiniteness constraint on \( X \) and \( Z \) has to be imposed. We define the inner product \( \bullet \) between two matrices \( A \) and \( B \) as follows: \( A \bullet B = \text{trace}(A'B) = \sum_{i=1}^{N} \sum_{j=1}^{N} A_{ij}B_{ij} \). Then a positive semidefinite relaxation of Problem (2.5) is written as:

\[
Z_{PSD1} = \text{Minimize} \quad Q \bullet X + c'_x x + \hat{W} \bullet Z \tag{2.6}
\]
\[ s.t. \quad A_{ij} \cdot X \leq h_i h_j, \forall i, j = 1, ..., m_x, i \leq j \quad (2.7) \]
\[ a_i^l x \leq h_i, i = 1, ..., m_x \]
\[ B_{ij} \cdot Z \leq r_i r_j, \forall i, j = 1, ..., m_z, i \leq j \quad (2.8) \]
\[ c_i^l x + g_i^l \text{diag}(Z) \leq d_l, l = 1, ..., m_{xz} \quad (2.9) \]
\[ x \geq 0, X \succeq 0, Z \succeq 0. \]

In the above formulation \( \hat{W} = W + \text{Diag}(c_z) \), where \( \text{Diag}(c) \) is a diagonal matrix with the elements of \( c \) in the diagonal. \( A_{ij} \) and \( B_{ij} \) are square symmetrical matrices and equal \( a_i a_j^l \) and \( b_i b_j^l \), respectively. \( \text{diag}(Z) \) is a vector with elements the diagonal entries of \( Z \).

Note that (2.7), (and similarly (2.8)) hold by multiplication of constraints \( a_i^l x \leq h_i \) and \( a_j^l x \leq h_j \) (\( b_i^l z \leq r_i \) and \( b_j^l z \leq r_j \)). Constraint (2.7) (and similarly (2.8)) is valid since \( a_i, a_j \geq 0 \) (\( b_i, b_j \geq 0 \)). It can be proved [21], that multiplying constraints (2.2) and (2.3) with their transpose provides a stronger formulation than simply taking (2.3) and substitute \( z_i \) by \( Z_{ii} \), or even squaring the constraints.

Note, that we have kept the \( x \) vector as a separate variable (\( x_i \neq X_{ii} \)). Otherwise, we would have to deal with \( x_i z_j \) (nonlinear \( x_i Z_{ij} \)) cross-terms in the constraints. This somehow obligates us to write (2.4) as shown in (2.9) above. An undesirable feature of formulation (2.6) is that we have to keep \( x \) separately from \( X \) without even connecting them through some kind of constraints. In more specific problems one would normally be able to avoid having to keep \( x \), as for example in the case of a simplex constraint on \( x \). Multiplying (2.2) on both sides with \( x_i \) can provide a desirable intermixing of \( x \) and \( X \), but more on this follows shortly. Problem (2.6) provides a tighter bound to (2.1) than the relaxed version (2.5). To see this, consider \( M(P) \) to be the set of all symmetric matrices satisfying the inequalities above, and let \( N(P) = \{ x, z : X, Z \in M(P), z = \text{diag}(Z) \} \). \( N(P) \) is a projection of \( M(P) \) from the \( N \times (N + 1)/2 \)-dimensional "quadratic" space of \( Z \) to the \( N \)-dimensional space of \( z \). By construction, every element of \( N(P) \) belongs to \( P \), but the reverse is not true. Also, \( P_0 \) is a subset of \( N(P) \). Therefore, we have \( P_0 \subseteq N(P) \subseteq P \) and so
\[ Z_{RQ} \leq Z_{PSD1} \leq Z_{OPT} \]

In an effort to introduce more constraints in the positive semidefinite relaxation strengthening the formulation, an idea that comes naturally is to take each of the initial constraints and multiply them with either \( x_i \geq 0 \), \( z_i \geq 0 \), or \((1 - x_i)(\text{if it is } \geq 0)\), \((1 - z_i) \geq 0\). One can then linearize the quadratic terms by substituting \( X_{ij} = x_i x_j \) and \( Z_{ij} = z_i z_j \). This certainty increases the number of constraints, but it provides tighter bounds. Denoting \( \overline{M}(P) \) the set of all symmetric matrices satisfying the inequalities resulting from the above procedure, it is clear that \( \overline{M}(P) \subseteq M(P) \). Accordingly, if \( \overline{N}(P) = \{ x, z : X, Z \in \overline{M}(P), z = \text{diag}(Z) \} \) then \( \overline{N}(P) \subseteq N(P) \) and therefore the optimal solution to this formulation will be larger or equal to \( Z_{PSD1} \).

One can tighten the relaxation even more by observing that for any 0-1 solution \( z \), the matrix \( Z = zz' \), has to satisfy \( Z - \text{diag}(Z)\text{diag}(Z)' = 0 \). This is a quadratic equality, but it can be relaxed to \( Z - \text{diag}(Z)\text{diag}(Z)' \geq 0 \), which is equivalent to

\[
\begin{bmatrix}
1 & \text{diag}(Z)'

\text{diag}(Z) & Z
\end{bmatrix} \succeq 0.
\]

The proof that this additional semidefinite constraint provides a tighter formulation, follows the same lines with the proofs above. For a more formal presentation of the ideas mentioned above the reader is referred to [25, 34, 2, 21, 16]

Applying the procedure to problem (2.6) we derive a stronger relaxation formulation which reads as follows:

\[
Z_{PSD} = \text{Minimize} \quad Q \bullet X + c'_x x + \hat{W} \bullet Z \tag{2.10}
\]

\[
\text{s.t.} \quad \sum_{j=1}^{N} a'_{ij} X_{jk} \leq h_i x_k, \forall i = 1, ..., m_x, k = 1, ..., N \tag{2.11}
\]

\[
a'_{ix} \leq h_i, i = 1, ..., m_x
\]

\[
A_{ij} \bullet X \leq h_i h_j, \forall i, j = 1, ..., m_x, i \leq j \tag{2.12}
\]

\[
\sum_{j=1}^{N} b'_{ij} Z_{jk} \leq r_i Z_{kk}, \forall i = 1, ..., m_z, k = 1, ..., N \tag{2.13}
\]

\[
B_{ij} \bullet Z \leq r_i r_j, \forall i, j = 1, ..., m_z \tag{2.14}
\]

\[
c'_l x + g'_l \text{diag}(Z) \leq d_l, l = 1, ..., m_{xz} \tag{2.15}
\]
\[ x \geq 0, \quad X \geq 0, \quad \begin{bmatrix} 1 & \text{diag}(Z)' \\ \text{diag}(Z) & Z \end{bmatrix} \succeq 0. \]

Note, that in (2.10) there is a much larger number of constraints than in (2.6). Also, mixing between the \( x \) and \( X \) variables occurs naturally. The higher number of constraints will render solution approaches slower and this is the cost to pay for attaining a lower bound \( Z_{PSD} \) which generally lies closer to the optimal solution \( (Z_{PSD1} \leq Z_{PSD} \leq Z_{OPT}) \). Whether this cost is worth paying, depends on the quality of the bound improvement in comparison to the increase in running time.

More elaborate ways to strengthen positive semidefinite relaxations, build on the structure and properties of the so called boolean quadric polytope [3, 29, 35]. Consider, for simplicity, an unrestricted purely integer program of the form \( \min \{ c'z + z'Wz : z \in \{0,1\}^N \} \), and write \( Z_{ij} = z_iz_j \) for \( i \leq j \). The quadric polytope is defined as \( QP^N = \text{conv}\{ (z,Z) \in R^N \times R^{N(N+1)/2} : z \in \{0,1\}^N, Z \in \{0,1\}^{N(N+1)/2} \} \), where \( \text{conv} \) denotes the convex hull of a set. \( QP^N \) is a bounded polyhedron with \( 2^N \) vertices. Its structure is quite complicated in general, but one can derive a number of inequalities which define facets of \( QP^N \). These inequalities can then be used as additional constraints to sharpen positive semidefinite relaxations. A quite successful family of such constraints are the so called triangle inequalities. These come from \( QP^3 \) and are written as:

\[
\begin{align*}
Z_{ij} & \geq 0 \\
Z_{ij} & \leq Z_{ii} \\
Z_{ii} + Z_{jj} & \leq 1 + Z_{ij} \\
Z_{ik} + Z_{jk} & \leq Z_{kk} + Z_{ij} \\
Z_{ij} + Z_{ik} + Z_{jk} + 1 & \leq Z_{ii} + Z_{jj} + Z_{kk}
\end{align*}
\]  

(2.16)

for all possible \( i,j,k \). Note, that in the above we have assumed \( z_i = Z_{ii} \). There is a total number of \( 2 \binom{N}{3} + N(3N + 1)/2 \) inequalities (2.16). In many problems,
some of these inequalities will automatically hold. When the problem is constrained one can extend the polytope-inequality idea to the underlying constraint-defining integer polyhedron associated with both the linear and quadratic representation of the problem. A number of cuts can be introduced, that tighten the relaxation further (see [30, 12, 33, 2] for example.) How to devise such cuts depends on the particular structure of the problem considered. For an example in the case of the 0-1 quadratic knapsack problem see [21]. As a final note, we mention that it is not clear how one can extend the polytope ideas in a case such as ours, in which apart from the binary \{0,1\} variables, there is a group of continuous variables also to be dealt with. It can certainly be quite challenging to identify facet defining inequalities for the mixed x-z polyhedron. Restricting ourselves to the z-polytope only could be an approach, but it can be considerably complicated by the presence of mixed constraints (2.15). A careful look at the structure of the particular problem under consideration is required.

Excluding the positive semidefiniteness constraints on the matrices, Problem (2.10) is a linear program with linear constraints and can be solved fast using standard algorithms. Quite a few methods have been developed to solve positive semidefinite problems efficiently and we give a presentation in the next section. Our focus is on interior-point methods.

2.1.1 Interior Point Methods

Positive semidefinite relaxations have been studied extensively in the last few years for a wide range of problems. Among them we distinguish the MAXCUT problem, quadratically constrained quadratic problems, the smallest/largest inscribed ellipsoid problem, eigenvalue optimization as well as combinatorial optimization problems. For a survey we refer the reader to [36], [13] and http://www.zib.de/helmberg/semidef.html.

Positive Semidefinite problems are typically solved by interior point methods. See [36], [13] and references therein. Various packages have been developed and a list is available at http://www.zib.de/helmberg/semidef.html#Software. We used the SDPA package developed by K. Fujisawa, M. Kojima and K. Nakata at the Tokyo Institute of Technology in 1998 [15]. The package is implemented in C++ and is based on
the infeasible primal-dual interior point method. This involves both primal and dual matrices in the system of equations used to determine the Newton step. Evaluation of both the primal and dual costs is taking place at each step. A clearer picture on the algorithm’s convergence is possible this way. In the context of the combinatorial problems studied, if the primal and dual values coincide at the end one can be certain that this is a valid lower bound to the original mixed integer-quadratic program. In addition, the package features various search direction approaches for the Newton method. It comes in the form of a callable library of routines and allows for efficient input and handling of sparse matrix structures. The program solved by SDPA is of the following form:

\[
PRIMAL: \text{ Minimize } \sum_{i=1}^{m} c_i x_i \\
\text{subject to } X = \sum_{i=1}^{m} F_i x_i - F_0 \\
X \text{ symmetric, } X \succeq 0.
\]

\[
DUAL: \text{ Maximize } F_0 \cdot Y \\
\text{subject to } F_i \cdot X = c_i, i = 1, \ldots, m \\
Y \text{ symmetric, } Y \succeq 0.
\]

The \(X\) and \(Y\) matrices above can be block diagonal. This facilitates their use in our case where we have to deal with the continuous and binary variables matrices \(X = xx'\) and \(Z = zz'\), respectively. These can be represented as two separate blocks in the dual matrix \(Y\) above. Note that the problems we are studying in this work correspond to the dual in SDPA which corresponds to maximization of the objective. The only drawback is that the constraints handled by SDPA are equality constraints only. For that reason we model inequalities by introducing additional slack variables, one for each inequality, then placing them in the diagonal of \(Y\) as new variables. Since \(Y\) will be positive semidefinite at solution, these diagonal elements will have to be positive as well. This way inequalities in the problem can be modeled. Consider
for example the mixed inequality \( x_i \leq z_i, i = 1, \ldots, n \). One will have to introduce \( n \) additional variables \( \epsilon_i \) and put them as an additional block in \( Y \). Then the inequality is written as the equality constraint \( z_i - x_i - \epsilon_i = 0 \).

Coming back to the mixed integer-quadratic programs, assume that the problem we are looking at involves \( N \) variables, that is \( N \) continuous and \( N \) discrete variables, and \( m \) inequality constraints. The size of the positive semidefinite program to be solved by SDPA will then be of size \( N(N+2) + m \). As a consequence, a combinatorial problem of size 50, translates into a semidefinite problem that involves 2550 variables or more. This is quite large by positive semidefinite algorithm standards. Just to give the reader an idea, we mention that in the case of the portfolio tracking problem (see Chapter 5), a problem with \( N = 50 \) and 3,977 constraints (3,825 of which are inequality constraints) was solved in about 5 hours by SDPA. This is a quite good performance for a positive semidefinite problem that in effect has 6,425 variables, but the limitations inherent to such an approach become obvious should one attempt to tackle larger-scale combinatorial problems. Below we give more details on the implementation of SDPA along with results in the context of the portfolio tracking problem. We also comment on the quality of the lower bounds produced and how they compare to lower as well as upper bounds available by other methods when no or some of the binary variables are fixed to either 0 or 1. The last case is of particular interest for node pruning in branch and bound schemes that aim to solve the problem exactly (see Chapter 4).

The Portfolio Problem - Computational Results for the Relaxed and Positive Semidefinite Formulations

In this section we attempt to solve relaxation formulations of problem (1.2). We consider the simple relaxed version of the original problem and the (linear) positive semidefinite formulation solved by the interior point method. One problem instance was created for each dimension \( N \) by randomly generating a vector \( s \in [-1, 1] \). Matrix \( Q \) was formed as \( Q = ss' \). A random positive (but less than 0.01) cost vector was also generated. Its elements were added on the diagonal elements of \( Q \). The same
procedure was followed to add the $W$ matrix in problems with $z$ variables in the objective. These problem instances are the ones studied computationally throughout this thesis. In all problems the $\alpha$'s have been taken equal to 0.01.

**RELAXED PROBLEM FORMULATION**

This is simply the original problem (1.2) where one allows the $\{0,1\}$-integer variables to take any value between 0 and 1. This way the problem is transformed to a continuous variables problem with a quadratic objective and linear constraints. It can be solved fast using quadratic solvers, like CPLEX, but the lower bound estimate is expected to be quite weak in general. The formulation is shown below.

\[
Z_{RQ} = \text{Minimize} \quad x'Qx + c'_x x + (z'Wz) \quad (2.17)
\]

\[
\text{subject to} \quad \sum x_i = 1 \quad (2.18)
\]

\[
\sum z_i = M \quad (2.19)
\]

\[
\alpha_i z_i \leq x_i \leq z_i, i = 1, \ldots, N \quad (2.20)
\]

\[
0 \leq z_i \leq 1, i = 1, \ldots, N.
\]

We solve the relaxed version of different size problems using the barrier optimization solver in CPLEX. Lower bounds and running times are shown in Table 2.2. We examine both problems with and without a quadratic $z$ part in the objective.

We find useful to consider the performance of our relaxation algorithms when some binary variables have been fixed to either 0 or 1. This gives a better idea on how the generated lower bounds would behave in a branch and bound scheme. We randomly pick some variables and their values (shown in Table 2.1) and run the relaxation algorithms for the nonfixed-variable set. This is done for all algorithms in this chapter. In Table 2.3 we present some results for the simple relaxed version (2.17) of the portfolio problem.

We need to mention, that for \text{"}x'Qx\text{"} problems with $N \geq 30$ and \text{"}x'Qx + z'Wz\text{"} problems with $N \geq 100$ we divided the lower bound with the best upper bound
available from heuristics (Chapter 3), since the optimal solution was not available from Branch and Bound. This is true for all results in Chapter 2.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Problem Type</th>
<th>Fixed Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>all</td>
<td>[2 3 6 8]={1001}</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>all</td>
<td>[2 4 7 12 15]={01010}</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>all</td>
<td>[5 9 13 16 20 22 27]={0101100}</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>all</td>
<td>[4 8 15 19 21 29 34 40 45]={100010010}</td>
</tr>
</tbody>
</table>

Table 2.1: Fixed binary variables and their values for fixed-variable relaxation algorithm runs in Chapter 2.

Note the improvement in the bounds as some variables get fixed. This is especially pronounced in the case of the \( z'Wz \) problems. Despite the improvement, however, the lower bounds are still very small and pruning in an exact Branch and Bound algorithm is not likely to occur for most nodes (pruning would occur if the \( Z_{RQ}/Z_{OPT} \) number was > 100%).

This method is fast, but quite simple. The lower bound estimates cannot be tighter than those obtained by more sophisticated algorithms, like, for example, positive semidefinite relaxations, which build upon problem (2.17) to produce more strict formulations.

*(LINEAR) POSITIVE SEMIDEFINITE RELAXATION*

In this subsection we consider the (linear) positive semidefinite relaxation of problem (1.2). We call this “linear” positive semidefinite as opposed to the quadratic positive semidefinite formulation which follows in the next subsection. In the linear case, both the \( x \) and \( z \) parts of the problem are approximated by a linear objective in a higher dimensional space. We formulate problem (1.2) according to the more strict positive semidefinite formulation (2.10) as follows:

\[
Z_{PSD} = \text{Minimize} \quad Q \cdot X + c'_x x + (W \cdot Z) \\
\text{subject to} \quad \sum x_i = 1
\]
Table 2.2: Lower bounds and running times for the Relaxed Formulation when no binary variables are fixed. A star(*) denotes that for the corresponding problem $Z_{RQ}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{RQ}/Z_{OPT}$ is larger than or equal to the "starred" numbers shown in the Table.

\[
\begin{align*}
\sum X_{ij} &= x_i, \ i = 1, \ldots, N \\
\sum Z_{ii} &= M \\
\sum Z_{ij} &= MZ_{ii}, \ i = 1, \ldots, N \\
\alpha_i\alpha_jZ_{ij} &\leq X_{ij} \leq Z_{ij}, \ i, \ j = 1, \ldots, N, \ i \geq j \\
0 &\leq Z_{ij}, \ i, \ j = 1, \ldots, N, \ i \geq j \\
X &\succeq 0, \ \left[ \begin{array}{c}
1 \\
diag(Z)' \\
diag(Z)
\end{array} \right] \succeq 0.
\end{align*}
\]

Constraints (2.23) and (2.25) have resulted by multiplying both sides of (2.22) and (2.24) with $x_i$ and $z_i$ respectively taking into account that $X_{ij} = x_ix_j$, $Z_{ij} = z_iz_j$ and $Z_{ii} = z_i$. Constraint (2.26) has resulted by multiplying inequality (1.5) for $i$ and
\begin{table}[h]
\begin{center}
\begin{tabular}{ccc}
\hline
$x'Qx$ - SOME FIXED VARIABLES \\
N & M & Time (sec) & $Z_{RQ}/Z_{OPT}$ (\%) \\
\hline
10 & 4 & 0.06 & 50.276 \\
20 & 5 & 0.13 & 31.814 \\
30 & 6 & 0.35 & *47.188 \\
50 & 8 & 0.84 & *26.491 \\
\hline
\end{tabular}
\end{center}
\end{table}

\begin{table}[h]
\begin{center}
\begin{tabular}{ccc}
\hline
$x'Qx + z'Wz$ - SOME FIXED VARIABLES \\
N & M & Time (sec) & $Z_{RQ}/Z_{OPT}$ (\%) \\
\hline
10 & 4 & 0.09 & 64.760 \\
20 & 5 & 0.24 & 47.620 \\
30 & 6 & 0.72 & 55.730 \\
50 & 8 & 1.52 & 60.103 \\
\hline
\end{tabular}
\end{center}
\end{table}

Table 2.3: Lower bounds and running times for the Relaxed Formulation when the binary variables of Table 2.1 are fixed at their \{0,1\} values. The relaxation is run for the remaining variables. A star (*) denotes that for the corresponding problem $Z_{RQ}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{RQ}/Z_{OPT}$ is larger than or equal to the “starred” numbers shown in the Table.

\[ j. \] We solve (2.21) using SDPA ([15]).

\emph{Interior – Point Method Results}

The big advantage of the interior-point method is that one is able to monitor both primal and dual costs in the course of the algorithm at no significant additional cost. If the two costs coincide at the end (no duality gap), the objective value obtained is the true and best lower bound the positive semidefinite formulation can give. The algorithm starts with an arbitrary positive definite diagonal matrix and moves iteratively using Newton steps that direct the primal and dual costs towards convergence ensuring that the corresponding matrices stay within the positive semidefinite cone.

A number of parameters are important for good algorithmic performance. First is the search direction parameter. This has to do with the way the Newton step is taken [36, 15] and SDPA offers three choices [15]. We have found that the HRVW/KSH/M
Another important parameter is the solution accuracy parameter (called \( \epsilon \) in SDPA). This determines the accuracy of an approximate optimal solution of the semidefinite program. It relates to the duality gap and serves as the stopping criterion for the method. In fact, \( \epsilon \) is equal to \(( \text{primal cost} - \text{dual cost})/(|\text{primal cost}|+1)\). The smaller it is, the more it will take the method to terminate but the more accurate the solution will be. A very small \( \epsilon \) can cause numerical instability. Typical values for \( \epsilon \) are \(10^{-8}\) (too strict), \(10^{-6}\), or \(10^{-4}\). Choosing \(10^{-4}\) versus \(10^{-8}\) can speed up the calculation without a big sacrifice in the quality of the lower bound obtained. In fact, we have observed that using a looser \( \epsilon \) is preferable when the positive semidefinite solution for \( Z \) is then used to generate random samples (see Chapter 3) and is definitely preferable in Branch and Bound schemes where the relaxation solution accuracy is of little importance and what matters the most is fast execution.

Another important parameter relates to the initial point (starting matrix) used in the method. In SDPA this is a diagonal matrix with all its elements set equal to a constant. This constant has to be positive and Fujisawa et al. suggest a sufficiently large value. We use the default value of 100.

Before moving on to the results it is very important to mention, that in positive semidefinite problems strong duality does not hold. In other words, \textit{optimality does NOT imply zero duality gap} (weak duality). The only case that strong duality holds is when the problem is strictly feasible, that is, when the solution matrix is positive definite (no zero eigenvalues). In our problems this is not the case (especially when some variables are fixed). The duality gap is then measured by the “mu” parameter in SDPA. A problem that can appear in these cases is that the algorithm does not terminate until it reaches the maximum number of iterations. One can detect such a behavior by looking at the primal and/or dual cost improvement, which can stay stagnant for a few iterations, and terminate the program early. The presence of a

\footnote{Stands for the relevant papers on the subject by Helmberg, Rendl, Vanderbei, Wolkowicz [20], Kojima, Shindoh, Hara [23] and Mehrotra [26].}
duality gap does not imply that the solution is not optimal, unless all matrix solution eigenvalues are nonzero.

In Table 2.4 we show results available for the same group of problems examined for the relaxed version in the previous section, when NO variables are fixed.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Eff. No. of vars</th>
<th>Time (sec)</th>
<th>(Z_{PSD}/Z_{OPT}(%))</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>285</td>
<td>2.6</td>
<td>40.083</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>1,070</td>
<td>60.6</td>
<td>25.976</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>2,355</td>
<td>571.0</td>
<td>*20.570</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>6,425</td>
<td>15,930.0</td>
<td>*23.344</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Eff. No. of vars</th>
<th>Time (sec)</th>
<th>(Z_{PSD}/Z_{OPT}(%))</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>285</td>
<td>3.1</td>
<td>91.529</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>1,070</td>
<td>66.9</td>
<td>94.185</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>2,355</td>
<td>691.0</td>
<td>95.809</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>6,425</td>
<td>18,374.0</td>
<td>97.448</td>
</tr>
</tbody>
</table>

Table 2.4: Lower bounds and running times for the (Linear) Positive Semidefinite Formulation when no binary variables are fixed. Solution by SDPA (interior-point method) with accuracy \(10^{-6}\). A star(*) denotes that for the corresponding problem \(Z_{PSD}\) was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, \(Z_{PSD}/Z_{OPT}\) is larger than or equal to the "starred" numbers shown in the Table.

Notice the dramatic improvement over the relaxed version bounds in the "\(z'Wz\)" case. In the "\(x'Qx\)" problems, however, the lower bound improvement is small. A possible explanation of this will be provided shortly. Note that the effective number of variables in the problems shown above is \(N(N+2)\). The \(N = 50\) problem is therefore a positive semidefinite program with 2600 variables. Because in fact SDPA handles only equality constraints, one has to introduce slack variables for the inequality constraints in the problem. As we mentioned earlier, this introduces an additional number of variables, which in problem (2.21) is \(1.5N(N + 1)\). So the actual size of a \(N = 50\) problem is 6425 variables! This is a large problem by positive semidefinite standards. It is clear from Table 2.4 that the use of this method in obtaining good lower bounds in a Branch and Bound scheme is almost prohibitive for mixed integer-quadratic
problems of size larger than $N = 50$.

We observed that solution $Z$ does not satisfy equality $Z_{ij} = Z_{ii}Z_{jj}$ in general. This is to be expected, since this equality is not supposed to hold when the corresponding elements of $Z$ are not strictly 0 or 1. Imposing such a constraint explicitly would lead to a stronger bound, but the constraint would be nonlinear.

Note also that in the "$x'Qx$" case, there is no significant improvement in $Z_{PSD}$ compared to $Z_{RQ}$. We have checked that from triangle inequalities (2.16), the $Z_{ij} + Z_{ik} + Z_{jk} + 1 \leq Z_{ii} + Z_{jj} + Z_{kk}$ inequality was usually violated. This was true for both types of problems. Inequalities like this one can be introduced to further improve the bound. Given that the total number of such inequalities is $\binom{N}{3}$ we did not attempt to add them into the formulation, because the size of the problem would become much larger and this would limit our ability to solve larger problems.

In general we have observed that in the no fixed variable case, the mixed constraints (2.26) are satisfied with strict inequality. We tried to run the SDPA algorithm again without explicitly imposing these constraints with the hope that the result will not be affected, but the algorithm will run much faster. The results we have obtained show a dramatic improvement in running time (by at least an order of magnitude) with a very small (in larger problems even indistinguishable) change in the lower bound. This has allowed us to tackle even larger problems with up to 70 binary variables. Note that by removing these constraints ($N(N + 1)$ in total) the total number of positive semidefinite variables in the algorithm will be reduced considerably, since these are inequality constraints in the problem. Alternatively, one could just introduce mixed constraints of the form $\alpha_i Z_{ii} \leq x_i \leq Z_{ii}$, $i = 1, \ldots, N$ for safety, in more general problems. This represents $N$ instead of $N(N + 1)$ constraints and slack variables. A further simplification can arise if one also removes the $N(N + 1)/2$ constraints $Z_{ij} \geq 0$, $\forall i \geq j$. These can be replaced by $Z_{ii} \geq 0$ for safety in more general problems. This simplification is supported by computational evidence in our problems that indicates that the $Z$ solution matrix elements turn out positive all the time. Replacing $Z_{ij} \geq 0$ with $Z_{ii} \geq 0$ further reduces the number of
slack variables and constraints (by $N(N + 1)/2 - N$) and allows us to consider even larger problems with up to 200 binary variables. The results are shown in Table 2.5 and are to be compared with those in Table 2.4.

<table>
<thead>
<tr>
<th>$x^TQx$ - NO FIXED VARIABLES</th>
<th>N</th>
<th>M</th>
<th>No. of Vars</th>
<th>Time (sec)</th>
<th>$Z_{PSD}/Z_{OPT}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>185</td>
<td></td>
<td>0.09</td>
<td>40.082</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>670</td>
<td></td>
<td>0.44</td>
<td>25.978</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>1,485</td>
<td></td>
<td>1.26</td>
<td>*20.571</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>3,925</td>
<td></td>
<td>7.33</td>
<td>*23.343</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>7,595</td>
<td></td>
<td>25.56</td>
<td>*14.331</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
<td>10,400</td>
<td></td>
<td>98.10</td>
<td>*15.064</td>
</tr>
<tr>
<td>150</td>
<td>25</td>
<td>23,100</td>
<td></td>
<td>458.80</td>
<td>*16.750</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x^TQx + z^TWz$ - NO FIXED VARIABLES</th>
<th>N</th>
<th>M</th>
<th>No. of Vars</th>
<th>Time (sec)</th>
<th>$Z_{PSD}/Z_{OPT}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>185</td>
<td></td>
<td>0.14</td>
<td>91.529</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>670</td>
<td></td>
<td>0.44</td>
<td>94.185</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>1,485</td>
<td></td>
<td>1.31</td>
<td>95.809</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>3,925</td>
<td></td>
<td>8.18</td>
<td>97.448</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>7,595</td>
<td></td>
<td>35.57</td>
<td>98.089</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
<td>10,400</td>
<td></td>
<td>131.60</td>
<td>*95.958</td>
</tr>
<tr>
<td>150</td>
<td>25</td>
<td>23,100</td>
<td></td>
<td>680.70</td>
<td>*99.998</td>
</tr>
<tr>
<td>200</td>
<td>40</td>
<td>40,800</td>
<td></td>
<td>2,134.00</td>
<td>*34.223</td>
</tr>
</tbody>
</table>

Table 2.5: Lower bounds and running times for the (Linear) Positive Semidefinite Formulation when no binary variables are fixed and mixed constraints 2.26 have been replaced by $\alpha_i z_{ii} \leq x_i \leq Z_{ii}$. Furthermore, constraints $Z_{ij} \geq 0$ have been replaced by $Z_{ii} \geq 0$. Solution by SDPA (interior-point method) with accuracy $10^{-6}$. A star(*) denotes that for the corresponding problem $Z_{PSD}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{PSD}/Z_{OPT}$ is larger than or equal to the "starred" numbers shown in the Table.

The deterioration of the lower bound in the $N = 200$ case may have to do with the fact that there is only one upper bound available from heuristics (see Chapter 3), which is probably weak.

A drawback of the SDPA Solver is that it does not handle inequality constraints explicitly. One has to introduce additional variables in the solution matrices to handle inequality constraints. Another drawback is, that in SDPA Version 4.10 there is no routine in the SDPA library that reads an input file. As a result one has to
create the input data structure every time the program is called. This can take considerable computational time in a Branch and Bound scheme. We should mention, however, that as far as we know the developers of SDPA are working on releasing a new enhanced version of SDPA, which is going to run faster than Version 4.10 and will include additional library modules to handle data input and output more efficiently.

*Results with Fixed Variables*

The performance of relaxation algorithms when some variables are fixed to 0 or 1 is important in Branch and Bound for early tree pruning. In Table 2.6 we show lower bounds obtained from formulation (2.21) when some variables are fixed. In particular, the variables shown in Table 2.1 are set to their respective 0-1 values and the relaxation is run for the remaining variables. It is possible now that the lower bound be larger than the optimal solution (or the best available upper bound). This corresponds to a number greater than 100% in the rightmost column of Table 2.6. In such a case, pruning of that particular node in a branch and bound tree would result.

Notice, that in Table 2.6 we have again used SDPA without the redundant constraints (2.26). We only add constraints $X_{ij} = 0, Z_{ij} = 0, \forall j$ (2N total) if the corresponding binary variable $z_i$ has been fixed to 0, or $Z_{ii} = 1, Z_{ij} = Z_{jj}, \forall j \neq i$ (N total) if the corresponding binary variable $z_i$ has been fixed to 1. This results in serious improvement in running time, since the number of semidefinite variables involved is much smaller. For example, compare a running time of 1190 sec (if constraints (2.26) were included) with 147.6 sec when these constraints are not included for the “$x'Qx$” problem with $N = 30, M = 6$.

The running times in Table 2.6 are a little bit longer than in the no fixed variable case, since there are additional constraints in the problem. Note that the lower bound estimates are much stronger now and in some cases pruning can occur. We checked that the equality $Z_{ij} = Z_{ii}Z_{jj}$ strictly holds when the corresponding variables $z_i$ and $z_j$ have been fixed to either 1 or 0. In a few cases, however, we observe abnormal termination of the SDPA algorithm. The algorithm terminates after the
| \(x'Qx\) - SOME FIXED VARIABLES |
|---|---|---|
| N | M | Time (sec) | \(Z_{PSD}/Z_{OPT} (%)\) |
| 10 | 4 | 1.9 | 50.330 |
| 20 | 5 | 23.2 | 32.292 |
| 30 | 6 | 147.6 | *50.154 |
| 50 | 8 | 172.0 | *35.171 |

| \(x'Qx + z'Wz\) - SOME FIXED VARIABLES |
|---|---|---|
| N | M | Time (sec) | \(Z_{PSD}/Z_{OPT} \) |
| 10 | 4 | 2.2 | 122.285 |
| 20 | 5 | 26.8 | 94.991 |
| 30 | 6 | 148.0 | 99.304 |
| 50 | 8 | 296.0 | 98.673 |

Table 2.6: Lower bounds and running times for the (Linear) Positive Semidefinite Formulation when the binary variables of Table 2.1 are fixed to their \{0,1\} values. Relaxation is run for the remaining variables. A \(Z_{PSD}/Z_{OPT}\) number greater than 100% means that pruning would occur at that node in branch and bound. Solution by SDPA (interior-point method) with accuracy \(10^{-6}\). A star(*) denotes that for the corresponding problem \(Z_{PSD}\) was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, \(Z_{PSD}/Z_{OPT}\) is larger than or equal to the "starred" numbers shown in the Table.

maximum number of iterations with a duality gap. This is to be expected since the solution is not strictly positive definite and the duality gap cannot diminish to zero (termination criterion for the algorithm). The appropriate lower bound in this case is the corresponding dual cost of the method \(^2\). Since SDPA takes at the most 20-30 iterations (depending on the problem size) to arrive at the final solution, a way to avoid unnecessary waste of computational time (in a Branch and Bound scheme for example) is to set the maximum number of iterations parameter in SDPA equal to, say, 35. Unfortunately, more elaborate ways involving checking for duality gap improvement and stop when there is no improvement in a number of iterations, cannot be implemented since we have no access to the SDPA source code.

Notice the clear difference in the bounds with the relaxed version results in Table 2.3. The effect is especially pronounced in the \(x'Qx + z'Wz\) problems. Observe

\(^2\)Recall that our primal problem corresponds to SDPA’s dual.
also the larger values that the bounds now have with respect to the no fixed variables case in Table 2.4, while there is no difference in running time.

Regarding the triangle inequalities (2.16), we observe that the first four are satisfied in general. The last one, however, is violated in quite a few instances, in particular when the variables involved are not among the fixed ones. Introducing some of these constraints may strengthen the formulation even more in both problem classes.

2.1.2 The Eigenvalue-Cut Method

In this section we divert from traditional ways to address positive semidefinite programs and we present a simple yet intuitive way to solve such problems. We outline the basic steps and ideas and we briefly comment on some potential drawbacks. It should be noted, that this is a novel idea, to the best of our knowledge, and as such there is a lot of room for improvement. For simplicity consider the following semidefinite program.

\[
\text{Minimize} \quad C \cdot X \\
\text{subject to} \quad A_i \cdot X = b_i, \ i = 1, \ldots, m \\
X \succeq 0.
\]  

\[ (2.29) \]

The unknown is the symmetric matrix \( X \) and the above problem can be thought of as a linear program where the variables are the elements of \( X \) and \( X \) must lie in the (closed convex) cone of positive semidefinite symmetric matrices \( S^n_+ \). \( A_i \) and \( b_i \) are given and define linear constraints that the elements of \( X \) have to satisfy. \( C \) is also given.

Assume that problem (2.29) is solved without the \( X \succeq 0 \) constraint, but with additional constraints imposing symmetry for \( X \), that is, \( X_{ij} = X_{ji}, i, j = 1, \ldots, N, i \neq j \). Suppose the solution is some matrix \( X^{(1)} \). This is not guaranteed to be positive semidefinite. One can calculate its eigenvalues and eigenvectors. The eigenvalue-cut algorithm consists of isolating the eigenvectors corresponding to the negative eigenvalues and introducing new constraints to the problem (“cuts”) that force positive
semidefiniteness in the space defined by these eigenvectors. So let us denote the set of these eigenvectors as \( V^{(1)} = \{v_1, v_2, ..., v_l\} \), where \( l \) is the number of the negative eigenvalues \( \Lambda^{(1)} = \{\lambda_1, \lambda_2, ..., \lambda_l\} \) of \( X^{(1)} \). Then we require that \( v_i'Xv_i \geq 0, i = 1, ..., l \). We introduce these new constraints back into the problem and solve again for \( X \) obtaining a new solution \( X^{(2)} \). We continue following exactly the same procedure as above until we reach at a solution \( X^* \) with nonnegative eigenvalues. The outline of the algorithm is as follows:

- **Initial step**: Impose symmetry constraints for matrix \( X \) and solve (2.29) without the \( X \succeq 0 \) constraint.

- **Step 1**: Determine the negative eigenvalues of \( X^{(k)} \) (kth iteration) and their corresponding eigenvectors \( \{v_1, v_2, ..., v_l\} \). If no negative eigenvalues exist, terminate, otherwise go to Step 2.

- **Step 2**: Add new constraints \( v_i'Xv_i \geq 0, i = 1, ..., l \) to the problem.

- **Step 3**: Solve new problem for matrix \( X^{(k+1)} \). Go to Step 1.

In the implementation of the algorithm we do not require the eigenvalues of \( X^{(k)} \) to be strictly positive. In order for the algorithm to terminate at the \( k^{th} \) iteration, we consider it enough to have \( \forall \lambda_i \in \Lambda^{(k)}, |\lambda_i| \leq \varepsilon \), where \( \varepsilon \) is a tolerance parameter. So, if \( \varepsilon = 10^{-3} \) this means that when \( \min_{\lambda_i \in \Lambda^{(k)}} \{\lambda_i\} \geq -\varepsilon \) the algorithm will terminate at the \( k^{th} \) iteration.

Note that we do not remove any previously added constraints during the process. We expect that as new cuts are added the eigenvalues are forced towards positive values, which they eventually reach. One potential practical problem with the method is very slow convergence towards the end of the algorithm and extensive memory requirements especially for large problems. As new constraints are added, the programs to be solved become larger and require more time and space for their solution. A variation to the algorithm would be to impose a maximum number of cuts that can be maintained, beyond which older constraints are removed from the stack to be
replaced by the new ones. The issues discussed here are addressed in more detail in the following pages in the context of the Portfolio problem.

**The Portfolio Problem - Computational Results for the Eigenvalue-Cut Method**

In this section we provide results for the (linear) positive semidefinite relaxation (2.21) when it is solved by the eigenvalue-cut method.

The algorithm starts by solving problem (2.21) without the positive semidefinite constraints (2.28). The resulting $X$ and $Z$ matrices will have negative eigenvalues, which will then be pushed towards positivity by introduction of the eigenvalue cuts as previously explained. The algorithm terminates when all eigenvalues become positive. The fact that the starting matrices for the method are not positive semidefinite is quite important. The positive semidefinite cone will be approached from the opposite direction than with an interior point algorithm. We have observed that although convergence is fast in the beginning, it slows considerably as the cone boundary is approached. Convergence can become hopelessly slow at the later stages of the algorithm. Because, in fact, negative eigenvalues (even of very small magnitude) will always exist, one needs to impose “nonnegativity” criteria for the eigenvalues. The so called tolerance parameter $\epsilon$ is introduced and an eigenvalue $\lambda_i$ is considered close enough to zero when $|\lambda_i| \leq \epsilon$. The smaller $\epsilon$ is, the faster the algorithm will terminate but the further from the positive semidefinite cone the solution will be. Another issue that can arise is computer memory limitation. As more cuts are added, the resulting linear programs become more time consuming and memory gets drained. We have observed that the inability of the method to solve large problems with sufficient accuracy has to do with how much memory is available in the computer.

In Table 2.7 we present some results when no variables are fixed for “$x'Qx$” problems. We denote the lower bound provided by the method as $Z_{E-C}$.

Note the bound improvement as the tolerance decreases, but observe the failure of the method in problems larger than 20 binary variables. This is due to the fact that with tolerances higher than the ones specified, the method fails to terminate.
Table 2.7: Lower bounds and running times for the (Linear) Positive Semidefinite Formulation in \( x'Qx \) problems when no binary variables are fixed. Solution by Eigenvalue-Cut Method.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>No. of vars</th>
<th>Tolerance</th>
<th>Time (sec)</th>
<th>( \frac{Z_{E-C}}{Z_{OPT}} )(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>120</td>
<td>10^{-2}</td>
<td>6.5</td>
<td>( \ll 1 )</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>120</td>
<td>10^{-3}</td>
<td>213</td>
<td>27.98</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>120</td>
<td>10^{-4}</td>
<td>389</td>
<td>37.98</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>440</td>
<td>10^{-2}</td>
<td>988</td>
<td>( \ll 1 )</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>960</td>
<td>10^{-1}</td>
<td>423</td>
<td>( \ll 1 )</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>960</td>
<td>5 \times 10^{-2}</td>
<td>1,196</td>
<td>( \ll 1 )</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>960</td>
<td>2 \times 10^{-2}</td>
<td>3,528</td>
<td>( \ll 1 )</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>2,600</td>
<td>5 \times 10^{-1}</td>
<td>2,938</td>
<td>( \ll 1 )</td>
</tr>
</tbody>
</table>

because of memory shortage. As a result, the lower bound estimates are much worse than those obtained with other methods since tighter eigenvalue tolerances cannot be handled.

A solution to the memory problem would be to keep the number of added cuts fixed to a certain number. This way, when the limit has been reached each new cut is added at the expense of an old one. We have experimented with this idea and have found that it can reduce the running time of the method in half, assuming tolerance is fixed. With no consideration about time, one could also use an even tighter tolerance this way. In Table 2.8 we present results on this for the \( x'Qx \) problem with \( N = 10, M = 4 \) where four variables have been fixed to 0 or 1. The tolerance used is 10^{-3}. Note that when there is no restriction on the number of cuts, the number of cuts upon termination is 329.

As one decreases the maximum number of constraints that the program can keep, running time is decreasing initially, but after a certain point (in this case 225) it starts to increase quite rapidly. First note that as new cuts are added in the eigenvalue-cut method, the solution of the underlying LPs becomes progressively slower and slower. When one restricts the maximum number of cuts the program can keep, then after some threshold point the program will have to drop some initial constraints from the stack in order to be able to add new ones. Because some of the "older" cuts are
<table>
<thead>
<tr>
<th>Max # Constraints</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>350</td>
<td>13.1</td>
</tr>
<tr>
<td>300</td>
<td>10.6</td>
</tr>
<tr>
<td>250</td>
<td>9.4</td>
</tr>
<tr>
<td>225</td>
<td>7.7</td>
</tr>
<tr>
<td>200</td>
<td>25.9</td>
</tr>
<tr>
<td>180</td>
<td>29.5</td>
</tr>
<tr>
<td>150</td>
<td>No Convergence</td>
</tr>
</tbody>
</table>

Table 2.8: Demonstration of the effect of setting a fixed number of cuts to be maintained in the Eigenvalue-Cut method in the case of the \( x'Qx \) problem with \( N = 10 \) and \( M = 4 \). Used tolerance \( 10^{-3} \).

lifted, strict improvement in the “positivity” of the eigenvalues is not guaranteed. The overall running time is expected to increase because the convergence rate decreases. On the other hand, these additional LP problems after the threshold point are going to run faster than if there was no restriction, because they involve smaller number of constraints. This trade-off is what makes desirable to set such limits in the eigenvalue-cut algorithm sometimes. The other reason is of course memory restrictions. We should warn, however, that this approach can not always be successful. Too small number of cuts allowed can lead to convergence failure, even increased running time as was demonstrated in the example above. One needs to experiment carefully in each problem considered to find the right combination of parameters. In larger problems we were not really able to replicate the behavior above with such success. The number of cuts could not be reduced by a lot, because pretty soon this led to divergence.

As we will see in Chapter 3, despite its poor performance, the eigenvalue-cut method can lead to excellent results when it comes to random sample generation from the solution matrix \( Z \). A possible explanation of this will be given in Section 3.2.3.
2.2 Quadratic Positive Semidefinite Algorithms

In Section 2.1 we focused on positive semidefinite algorithms that serve as approximation algorithms to combinatorial problems providing lower bounds to the optimal solution. Among other approximations involved, linearization of the quadratic objective is an important one and makes the approach quite appealing for efficient algorithm development. Note, that linearization occurred in both the continuous and binary variables. In this section we present a method aimed at providing lower bounds that are at least as tight as those obtained by the positive semidefinite relaxations of Section 2.1.

As a first step we attempt to keep the quadratic part of the objective that involves the continuous variables only. Instead of the symmetric matrix $X \ (N(N + 1)/2$ variables) one has to deal with the N-dimensional vector $x$. Of course, the objective is not linear in $x$. The binary variable part of the problem is formulated as a positive semidefinite program involving a symmetric matrix $Z$ exactly as illustrated in Section 2.1. If there was no intermixing of the $x$ and $z$ variables in the objective and/or the constraints, solving the resulting problem would be simple. The $x$-part would be solved fast as a linearly constrained quadratic program and the $z$-part would be solved as a positive semidefinite program with much smaller size than before. The lower bound would then be obtained by just adding the two optimal costs. It turns out that this desirable "separability" can still be achieved when there is intermixing of the $x$ and $z$ variables. One just introduces these constraints in the objective through some Lagrange multipliers. The remaining constraints are separable and the same can be said for the objective. There is still some coupling, however, through the Lagrange multipliers present in both the $x$ and $Z$ parts of the objective. The method is an extension of the Lagrangean dual methods in linear integer programming. See for example [5].

Consider the relaxed version of problem (2.5). We introduce the symmetric matrix $Z = zz'$ and convert the $z$-part in the objective and constraints in terms of $Z$. We also introduce a Lagrange multiplier vector $\mu$ with size equal to the number of mixed


\[ Z_{QPSD} = \max_{\mu \geq 0} \min_{x,Z} x'Qx + c'_x x + \hat{W} \cdot Z + \sum_{l=1}^{m_z} \mu_l (c'_i x + g'_l \text{diag}(Z) - d_i) \]

subject to

\[
\begin{align*}
    a'_i x &\leq h_i, \forall i = 1, \ldots, m_x \\
    \sum_{j=1}^{N} b'_{ij} Z_{jk} &\leq r_i Z_{kk}, \forall i = 1, \ldots, m_z, k = 1, \ldots, N \\
    B_{ij} \cdot Z &\leq r_i r_j, \forall i, j = 1, \ldots, m_z, i \leq j \\
    x &\geq 0, Z \succeq 0.
\end{align*}
\]

The problem decomposes as follows:

\[ Z_{QPSD} = \max_{\mu \geq 0} (V_z(\mu) + V_Z(\mu)), \]

where,

\[ V_z(\mu) = \min x'Qx + c'_x x + \sum_{l=1}^{m_z} \mu_l (c'_i x - d_i) \]

\[ s.t. \quad a'_i x \leq h_i, i = 1, \ldots, m_x \]

\[ x \geq 0, \]

\[ V_Z(\mu) = \min \hat{W} \cdot Z + \sum_{l=1}^{m_z} \mu_l g'_l \text{diag}(Z) \]

\[ s.t. \quad \sum_{j=1}^{N} b'_{ij} Z_{jk} \leq r_i Z_{kk}, \forall i = 1, \ldots, m_z, k = 1, \ldots, N \]

\[ B_{ij} \cdot Z \leq r_i r_j, \forall i, j = 1, \ldots, m_z \]

\[ \begin{bmatrix} 1 & \text{diag}(Z)' \\ \text{diag}(Z) & Z \end{bmatrix} \succeq 0. \]

Problem (2.31) is a linearly constrained quadratic program and (2.32) a positive semidefinite relaxation program on matrix Z. We are now in a position to present the Quadratic Positive Semidefinite Algorithm.

1. Choose a starting point \( \mu^0 \); set \( k = 0 \).
2. Given \( \mu^k \), solve (2.32). This will provide an optimal \( Z_k^* \) and objective value \( V_{Z_k^*}(\mu^k) \). Then solve (2.31) and obtain the optimal \( x_k^* \) and objective value \( V_{x_k^*}(\mu^k) \).

3. Update \( \mu \) by letting \( k = k + 1 \) and \( \mu_{l+1} = \max\{\mu_l + \theta_k(c_l'x_k^* + g_l'diag(Z_k^*) - d_l), 0\} \), for \( l = 1, \ldots, m_{xx} \). \( \theta_k \) is a positive stepsize parameter. If an appropriately defined stopping criterion is met terminate (the final solution is then \( V_{Z_k^*}(\mu^k) + V_{x_k^*}(\mu^k) \)), otherwise go to Step 2.

Important in the implementation of the above algorithm is the specification of the termination criterion and the stepsize parameter. Regarding the first, a valid choice would be either that \( \mu^{k+1} - \mu^k = 0 \), or that there is no significant improvement in the objective value (lower bound). As an alternative, the number of iterations can be fixed to a certain value. Regarding the stepping parameter, we use \( \theta_k = \theta_0 \alpha^k \), \( k = 0, 1, 2, \ldots \) where \( \alpha \) is a scalar satisfying \( 0 < \alpha < 1 \). Other more sophisticated choices can be used [5] but the one above is sufficient for our purposes. The choice of the starting vector \( \mu^0 \), \( \theta_0 \) and \( \alpha \) is quite important for fast convergence. When a fixed number of iterations stopping criterion is adopted this should be done with care. Along the course of the algorithm the objective value will proceed from lower to larger values. In the context of the mixed integer-quadratic problems we consider, this means that the lower bound estimate will progressively become tighter and tighter. Stopping the algorithm prematurely may yield a poor bound. On the other hand, allowing the algorithm to run beyond the point that no significant progress in the objective value is realized, is a waste of time. We believe that a relatively loose stopping criterion on the percentage progress of the objective value is a quite appropriate one for the problems we are considering. This also takes into account the fact that in some cases the \( \mu^{k+1} - \mu^k = 0 \) criterion is just impossible to meet. In the next section we provide details on the implementation and lower bound results for this algorithm in the Portfolio problem.
The Portfolio Problem - Quadratic Positive Semidefinite Relaxation Results

We start with a detailed description of the algorithm for problem (1.2). Consider keeping the quadratic objective for $x$ and writing the $z$ part in positive semidefinite form. Also consider introducing the mixed constraints (1.5) into the objective through some positive Lagrange multiplier vectors $p$ and $q$ as shown in (2.33). Note that because there is no $X$ matrix now, constraints (1.5) are written as $\alpha_i Z_{ii} \leq x_i \leq Z_{ii}$, $i = 1, \ldots, N$.

\[
Z_{QPSD} = \max \min \quad x'Qx + c'_x x + (W \bullet Z) + \sum_{i=1}^{N} p_i (x_i - Z_{ii}) + \sum_{i=1}^{N} q_i (\alpha_i Z_{ii} - x_i)
\{p,q\}\{x,Z\} \tag{2.33}
\]

subject to
\[
\sum x_i = 1 \quad \tag{2.34}
\]
\[
\sum Z_{ii} = M \quad \tag{2.35}
\]
\[
\sum Z_{ij} = MZ_{ii}, i = 1, \ldots, N \quad \tag{2.36}
\]
\[
p, q \geq 0, x \geq 0, \begin{bmatrix} 1 & \text{diag}(Z)' \\ \text{diag}(Z) & Z \end{bmatrix} \succeq 0.
\]

Considering fixed $p$ and $q$ the above problem is separable. To see this denote by $L_x = \{ x \in \mathbb{R}_+^N : \sum x_i = 1 \}$ and by $L_Z = \{ Z \in S^N : \begin{bmatrix} 1 & \text{diag}(Z)' \\ \text{diag}(Z) & Z \end{bmatrix} \succeq 0 \}$, $\sum Z_{ii} = M$, $\sum Z_{ij} = MZ_{ii}$, $i = 1, \ldots, N$, where $S^N$ is the set of all $N \times N$ symmetric matrices.

Now fix $p$ and $q$ to some value $\tilde{p} \geq 0$ and $\tilde{q} \geq 0$ and write the objective in (2.33) in two separate parts. The problem then becomes:

\[
H(\tilde{p}, \tilde{q}) = \min_{x \in L_x} F(x) + \min_{Z \in L_Z} G(Z)
\]

where $F(x) = x'Qx + c'_x x + \sum_{i=1}^{N} (\tilde{p}_i - \tilde{q}_i)x_i$ and $G(Z) = (W \bullet Z) + \sum_{i=1}^{N} (\alpha_i \tilde{q}_i - \tilde{p}_i)Z_{ii}$.

Fixing $p$ and $q$ the optimal value of the mixed $x - Z$ problem is just the sum of
the optimal values of the separate $x$ and $Z$ problems shown below:

\[
G_\ast(p, q) = \text{Min} \quad W \bullet Z + \sum_{i=1}^{N} (\alpha_i q_i - p_i)Z_{ii} \\
\text{s.t.} \quad \sum Z_{ii} = M \\
\sum Z_{ij} = MZ_{ii}, i = 1, \ldots, N \\
\begin{bmatrix}
1 & \text{diag}(Z)'
\text{diag}(Z) & Z
\end{bmatrix} \succeq 0,
\]

\[
F_\ast(p, q) = \text{Min} \quad x'Qx + c'_x x + \sum_{i=1}^{N} (p_i - q_i) x_i \\
\text{s.t.} \quad \sum x_i = 1 \\
x \geq 0.
\]

The problem finally reduces to:

\[
Z_{QPSD} = \text{Maximize} \quad H(p, q) \equiv F_\ast(p, q) + G_\ast(p, q)
\text{subject to} \quad p \geq 0, q \geq 0.
\]

Maximization over $p$ and $q$ is carried out by the steepest ascent algorithm. The gradients $\nabla p$ and $\nabla q$ are taken from the derivative of the cost function in (2.33) with respect to $p$ and $q$ scaled by a positive parameter $\theta$. The new iterates are calculated from $p^{k+1} = p^k + \theta \nabla p^k$ and $q^{k+1} = q^k + \theta \nabla q^k$. It is easy to see that $(\nabla p)_i = x_i - Z_{ii}$ and $(\nabla q)_i = \alpha_i Z_{ii} - x_i$. The stepsize parameter $\theta$ has to be chosen in such a way that the algorithm moves smoothly and steadily towards larger values of the cost $H(p, q)$. Finally, one has to ensure that the new iterates stay in the feasible region $\mathbb{R}_+^N \times \mathbb{R}_+^N$.

The algorithm is as follows.

1. Choose a starting point $\{p^0, q^0\}$; set $k = 0$.

2. Given $\{p^k, q^k\}$, solve (2.37). This will provide an optimal $Z^k_\ast$ and objective value $G_\ast(p^k, q^k)$. Then solve (2.38) and obtain the optimal $x^k_\ast$ and objective
value $F_*(p^k, q^k)$.

3. Update $p$ and $q$ by letting $k = k + 1$ and $p_i^{k+1} = \max\{p_i^k + \theta_k(x_{*,i}^k - Z_{*,ii}^k), 0\}$, and $q_i^{k+1} = \max\{q_i^k + \theta_k(\alpha_i Z_{*,ii}^k - x_{*,i}^k), 0\}$ for $i = 1, \ldots, m_{xx}$. $\theta_k$ is a positive stepsize parameter where $\theta_k = \theta_0 \alpha^k$ and $0 < \alpha < 1$. If an appropriately defined stopping criterion is met terminate (the final solution is then $G_*(p^k, q^k) + F_*(p^k, q^k)$), otherwise go to Step 2.

Successful execution of the QPSD algorithm depends crucially on a few parameters. These are the starting vectors $p^0$ and $q^0$, $\theta_0$ and $\alpha$. We have observed that when a binary variable $z_i$ is not fixed or fixed to 1, then $p_i^0$ and $q_i^0$ should be set at a value < 1. This is because the mixed constraint $\alpha_i Z_{ii} \leq x_i \leq Z_{ii}$ is not expected to be satisfied with equality and therefore the corresponding Lagrange multipliers are expected to be zero at the optimal solution. On the other hand, if $z_i$ is set to 0, then $p_i^0$ and $q_i^0$ should be set at a slightly higher value, since they are expected to be positive at termination. We have observed that the larger the problems size, the smaller these values have to be set at. Regarding $\theta_0$ and $\alpha$, a good empirical rule seems to be to set them close to the values of $p_i^0$ and $q_i^0$.

Another important implementation issue has to do with the termination criterion used. We have seen that with the parameters set as described above, convergence occurs within 200 iterations at the most. In addition to the “maximum 200 iterations” criterion, one can order the algorithm to stop when no significant improvement in the objective value has been observed after, say, 10 consecutive iterations. That could be implemented, for example, as $|\frac{H(p^{k+1}, q^{k+1}) - H(p^k, q^k)}{H(p^k, q^k)}| \leq 10^{-8}$.

A big advantage of the quadratic positive semidefinite relaxation is that it removes constraints (2.26) from the corresponding positive semidefinite subproblem (2.37) by construction. This makes execution of the SDPA algorithm much faster as we mentioned in Section 2.3. One can furthermore remove constraints of the form $Z_{ij} \geq 0$, which prove to be redundant and reduce the SDPA problem’s size even more. In Table 2.9 we show some results when no variables are fixed for the “$x'Qx + z'Wz$” problems.
<p>| &quot;$x'Qx + z'Wz$&quot; - NO FIXED VARIABLES |
|-------------------------|------------------|----------------------|</p>
<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Time (sec)</th>
<th>$Z_{QPSD}/Z_{OPT}(%)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>14.9</td>
<td>91.529</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>84.0</td>
<td>94.185</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>237.0</td>
<td>95.809</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>1,356.0</td>
<td>97.448</td>
</tr>
</tbody>
</table>

Table 2.9: Lower bounds and running times for the Quadratic Positive Semidefinite Formulation in "$x'Qx + z'Wz$" problems when no binary variables are fixed. Algorithm stops after 200 iterations.

Observe that the lower bounds $Z_{QPSD}$ in Table 2.9 are almost the same as those obtained by SDPA ($Z_{PSD}$ in Table 2.5). When no variables are fixed, constraints $\alpha_iZ_{ii} \leq x_i \leq Z_{ii}$ are strict inequalities. This means that the corresponding Lagrange multipliers $p$ and $q$ will have to be zero at solution, which results in the solution of QPSD to be very close to the PSD one. The slight difference that might be expected due to the $x'Qx$ versus the $Q \cdot X + c'x$ objective formulation for the $x$-part is very small in our problems. On the other hand, when some binary variables are fixed to 0, then things change because the mixed constraints hold with equality and the multipliers will be strictly positive, resulting an improvement in the bound estimates between the QPSD and the PSD algorithms. This is clearly observed in Table 2.10, where we show results when some binary variables are fixed.

It is worth noting, that the running time of the Quadratic Positive Semidefinite method, although fast for what it does, it is longer than SDPA. This is to be attributed to the iterative nature of the method. Although the $Z$ subproblem is much faster to solve than the corresponding PSD semidefinite formulation, this has to be done here at every iteration. Also, in some of these problems convergence at the optimal value was achieved in a fewer number of iterations than 200. We chose to present all results assuming 200 iterations for uniformity.

The lower bound improvement in the fixed variable case is strong and it would result pruning of almost all nodes considered in Table 2.10 in a Branch and Bound

---

3In very flat problems, for example, we expect more or less $x_i \sim 1/N$ and $Z_{ii} \sim M/N$ to hold.
<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Time (sec)</th>
<th>$Z_{QPSD}/Z_{OPT}$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>48.7</td>
<td>128.94</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>237.6</td>
<td>99.18</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>1,418.0</td>
<td>122.48</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>10,532.0</td>
<td>114.07</td>
</tr>
</tbody>
</table>

Table 2.10: Lower bounds and running times for the Quadratic Positive Semidefinite Formulation when the binary variables of Table 2.1 are fixed to their \{0,1\} values. The relaxation is run for the remaining variables. A $Z_{QPSD}/Z_{OPT}$ number greater than 100% means that pruning would occur at that node in branch and bound. The algorithm stops after 200 iterations.

algorithm. Nevertheless, the QPSD algorithm cannot be applied in such an exact scheme without a lot of experimentation first to determine the appropriate values that have to be used for its parameters.
Chapter 3

Heuristic Algorithms

In this chapter we present various heuristic approaches for finding good approximate solutions to the mixed integer-quadratic optimization problem. These heuristics are based on simple search ideas. They are constructed so that they run fast and provide a feasible solution that is relatively close to the optimal one. Unfortunately, no a priori guarantee for the goodness of the approximate solution exists for almost all the heuristics presented in this chapter. The heuristic algorithms can be used either as stand-alone modules providing good approximate solutions in case the problem's size is prohibitive for exact solving, or as part of a general branch and bound scheme, in which case, they can assist in cutting nodes early due to their ability to provide tight upper bounds to the optimal solution.

We consider three main approaches. The first relies on randomization schemes that are either stand-alone or they start based on specific information available, for example, after solving a relaxation. The second approach is based on greedy search and variations and the third relies on local search algorithms like, for example, simulated annealing. We present the above methods in the sections that follow, along with notes regarding their performance in practical problems. We note that our focus is on problems where constraint \( \sum_{i=1}^{N} z_i = M \) exists. First we begin with some comments on previous work on heuristic approaches applied to mixed integer quadratic combinatorial optimization problems similar to the Portfolio replication problem of this thesis.
3.1 Previous work

Little work has been done on developing heuristics for constrained mixed integer quadratic optimization problems. Chang et al. [11] apply approaches based on genetic algorithms, tabu search and simulated annealing to the cardinality constrained Portfolio problem. Their goal is to calculate an efficient frontier when the maximum number of assets constraint is imposed. Their approach is particularly interesting, because the heuristics they develop not only aim at identifying a good sample; they also construct the solution of the underlying linearly constrained quadratic program. The authors argue that avoiding using a quadratic solver every time a binary sample is generated, saves much computational time. They consider problems from 30 to 225 variables using actual financial data for indices around the world. A number of 1000N samples, where N is the problem's size, is generated at each heuristic run.

3.2 Randomization Heuristics

Randomization heuristics have been quite popular for combinatorial optimization problems. Their main attraction is that they are simple in their implementation and run relatively fast. For some problems, it has been proved that randomization schemes can come up with quite close bounds to the true optimal solution. As an example, we mention the graph bisection and Max-Cut problems, where a randomization heuristic guarantees a solution within 12.8% of the optimal one [17].

In this section, we examine various randomization heuristics for our problem. We start with a simple implementation and move towards more involved approaches. In each section, we present complete results and we comment on some technical issues regarding implementation and use of each heuristic in Branch and Bound. All the results presented here assume that no binary variables have already been fixed when the heuristic method is applied. The problem instances studied are one “x'Qx” and one “x'Qx + z'Wz” for each N. They are the same as in Chapter 2.

We need to mention again, that for “x'Qx” problems with N ≥ 30 and “x'Qx +
$z'Wz$" problems with $N \geq 50$ we divided the upper bound with the best upper bound available from the heuristics of this Section, since the optimal solution was not available from Branch and Bound. This is true for all results in this section. In particular, our performance measure is $(\text{Upper Bound/Optimal Sol}) - 1$. This number has to be $\geq 0$ and the closer to 0 it is, the better the heuristic. For the $N \geq 150$ problems there are only a couple of heuristic runs. This is because most heuristics could not be run for these problem sizes in a reasonable amount of time. The results, therefore, are not really representative in these cases.

As one would expect, the real challenge for upper-bound generating algorithms is in problems with steep rather than flat objectives. It is therefore interesting to compare their performance in the "steeper" $x'Qx + z'Wz$" problems in the results shown in this section.

### 3.2.1 A Simple Randomization Heuristic

This heuristic consists of randomly generating binary samples for $z$, that do not render the problem infeasible, and then solve the resulting linearly constrained quadratic problem for $x$ for each one of the samples. Generating random samples is fast, as is solving the resulting quadratic program. This way an upper bound can be estimated quite easily. This could be used as a starting upper bound in the branch and bound algorithm. Depending on the nature of the problem, the samples can be drawn from a binomial, uniform, multivariate normal or some other distribution and could be subject to certain restrictions as dictated by the constraints on the binary vector $z$. Certain parameters of the underlying distribution can be calibrated to reflect the relative likelihood of binary variables to be 0 or 1. For example, the resulting $[0, 1]$ values of the relaxed version of the problem at the current node can be used as some kind of "weight" parameters for the underlying statistical distribution.

In our case, we implement a rather simple version. Taking into account that in our problems the $\sum_{i=1}^{N} z_i = M$ constraint with $M < N$ is present, we build a uniform distribution random number generator. This provides a random number in $[0, N]$. Let $i$ be the integer part of this number. We set the corresponding element $z_i$ equal to
1. We continue until M different binary variables have been set to 1. We check if the rest of the constraints in z are satisfied. If yes, this is a valid sample and we solve the underlying quadratic program in x to estimate a cost. The procedure is repeated for a fixed number of iterations and the lowest cost value is kept along with the sample. For a minimization problem this is an upper bound to the optimal solution.

A big advantage of the algorithm is that it runs quite fast. One can test quite a big number of samples this way. The drawback is that there is absolutely no guarantee for the quality of the upper bound. We have observed that the method tends to provide very good results for most problems; it can fail in certain other problems especially if the quadratic objective is quite steep. For illustration and comparison purposes we present below a few results for the simple randomization heuristic approach in the portfolio problem for both \(x'Qx\) and \(z'Wz\) cases. Note that the second type of problem has steeper quadratic objective and therefore it represents more of a challenge to heuristic algorithms.

We denote by \(Z_{SRH}\) the upper bound provided by the method. Results for different problems are shown in Table 3.1 when a number of 5N random samples is generated. This choice is arbitrary and since the method is fast, it can be increased as desired.

In general, the SRH works quite well and is quite fast compared to other heuristics. Even in less “flat” problems of moderate sizes it generates descent bounds. Note, however, the disappointing performance in large “\(z'Wz\)” problems (size 70 and above). This may be attributed to the fact that the specified number of samples (5N), increases in a much smaller rate than the number of feasible solutions to the problem. The fact that these problems are not as “flat” as the “\(x'Qx\)” ones, makes this even worse. One would have to increase the number of samples a lot more to probably obtain better estimates. Similar behavior is observed in the M-Randomization Heuristic results below. Nevertheless, the bounds of this simple method are comparable to the ones obtained by other more “expensive” heuristic approaches.
Table 3.1: Sample Upper bound results from Simple Randomization Heuristic. Number of generated samples equals $5N$, where $N$ is the number of binary variables in the problem. A star(*) denotes that for the corresponding problem $Z_{SRH}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{SRH}/Z_{OPT} - 1$ is larger than or equal to the “starred” numbers shown in the Table.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Time (sec)</th>
<th>$(Z_{SRH}/Z_{OPT} - 1)$(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>0.8</td>
<td>0.0077</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>5.0</td>
<td>0.0099</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>13.0</td>
<td>*0.1179</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>55.0</td>
<td>*0.0029</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>152.3</td>
<td>*3.913 × 10⁻⁴</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
<td>547.9</td>
<td>*0.0000</td>
</tr>
<tr>
<td>150</td>
<td>25</td>
<td>2,603.0</td>
<td>*6.8 × 10⁻⁴</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“$x'Qz + z'Wz$” Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>70</td>
</tr>
</tbody>
</table>

3.2.2 M-Randomization Heuristic

This is an extension of the previous approach. The idea behind this heuristic draws partly on a paper on Portfolio Optimization by Patel and Subrahmanyan [31]. In this paper, the authors consider the problem of index tracking with a limited number of names when the correlation matrix has all of its off-diagonal elements equal to some constant. They suggest that one first solves the problem when all the assets are included in the portfolio and arrange them in order of ascending portfolio weights (probable negative weights go first in stack). Then one takes all possible combinations of the $k$ smallest- with the $M - k$ , $(k = 0, \ldots, M)$ largest-weight assets together and sees what cost each one of them results in. The combination with the minimum cost is the exact optimal solution. We should emphasize that the above result is made possible by the simple fact that the off-diagonal correlation coefficients are all equal and by the presence of both positive and negative portfolio weights when one solves
the problem with all the assets included. Assets with opposite-sign weights act as a hedge for each other and should be part of a good portfolio solution. This convenience is not present in our case where the quadratic matrices do not have equal elements and solving the relaxed version of the problem for the continuous variables is not expected to provide negative values in general. Indeed, trying this idea in some of our problems gave very disappointing results.

Instead, we developed a randomization heuristic idea, which we call “M-Randomization Heuristic”, that draws from this paper. The algorithm, is presented below.

- **Initial step**: Solve the problem as if all binary variables \( z \) equal 1 and \( M = N \). Order variables in ascending order of \( x \). Set upper bound \( upbd = \infty \). Set current sample equal to \( M \) largest-x variables. Set \( k = 1 \).

- **Step 1**: For current sample solve the problem and obtain cost. If cost < \( upbd \), set optimal sample equal to current sample, else go to Step 2.

- **Step 2**: Set \( k = k + 1 \). If \( k \) larger than maximum allowed number of iterations, then terminate. Otherwise, exchange a randomly selected variable from current sample with one randomly selected variable from outside the sample to form new sample. Go to Step 1.

Like the Simple Randomization Heuristic (SRH), the M-Randomization Heuristic (M-RH) is very fast. It performs better than SRH in general. Because it has more structure than SRH, it is expected to be more robust and yield good results across a wider range of problems. There is no formal proof of that however. In Table 3.2 we present some results for the same kind of problems considered for SRH above for comparison. We denote by \( Z_{M,RH} \) the upper bound provided by the method. The number of generated samples is again \( 5N \).

The disappointing performance in the \( z'Wz \) problem with \( N = 70 \) may be attributed to the fact that the specified number of samples \( (5N) \), increases in a much smaller rate than the number of feasible solutions to the problem. In the \( N = 150 \) case the method has provided the best upper bound performance. That is why a 0.0
<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Time (sec)</th>
<th>((Z_{M-RH}/Z_{OPT} - 1)) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>0.9</td>
<td>0.0161</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>4.8</td>
<td>0.0106</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>13.2</td>
<td>*0.1276</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>79.6</td>
<td>*0.0000</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>151</td>
<td>*44.64 \times 10^{-4}</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
<td>506</td>
<td>*0.4740</td>
</tr>
<tr>
<td>150</td>
<td>25</td>
<td>2,208</td>
<td>*0.0000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>“(x'Qx + z'Wz)” Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>70</td>
</tr>
<tr>
<td>150</td>
</tr>
</tbody>
</table>

Table 3.2: Sample Upper bound results from M-Randomization Heuristic. Number of generated samples equals 5N, where N is the number of binary variables in the problem. A star(*) denotes that for the corresponding problem \(Z_{M-RH}\) was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, \(Z_{M-RH}/Z_{OPT} - 1\) is larger than or equal to the “starred” numbers shown in the Table.

entry appears in the Table. Note that for that problem we have only two estimates from heuristics. It turns out, however, that the M-Randomization heuristic estimate is very close to the lower bound available from SDPA for the same problem (see Table 2.5 in Chapter 2 and Table 4.2 in Chapter 4).

The M-Randomization Heuristic seems to be a very good candidate for use in exact branch and bound algorithms because it runs fast and provides tight upper bounds to the optimal solution. This is true especially for large problems, where other heuristic techniques turn out to be quite expensive computationally.

3.2.3 Positive Semidefinite Randomization Heuristic

In this section we develop a randomized heuristic technique that elaborates on the solution of a positive semidefinite relaxation algorithm of the symmetric matrix for the
binary variables $Z = zz'$. Recall from Chapter 2, that (quadratic) positive semidefinite algorithms are relaxations of the original problem, which they attempt to solve in terms of symmetric matrices formed by multiplying the vector of the original variables with its transpose. These matrices are guaranteed to be positive semidefinite at the solution.

We focus on matrix $Z$ that corresponds to the $\{0,1\}$ variables $z$. Because the positive semidefinite relaxations start from the relaxed version of the original problem, the elements of $Z$ are expected to lie between 0 and 1. Recall that $Z_{ii} = z_i$, when $z_i = \{0,1\}$. It is natural to assume that the higher $Z_{ii}$, the more probable it is for variable $z_i$ to equal 1. One could therefore devise a probability distribution that draws variables randomly with heavier probabilities assigned to higher $Z_{ii}$ variables. A first approach would be to pick variable $i$ with probability $p_i = Z_{ii}/M$. Note that $\sum Z_{ii} = M$ and $\sum p_i = 1$. This can easily be accomplished through a uniform random number generator. This approach, however, does not take into account information available from the off-diagonal elements of $Z$. A method that in effect treats $Z$ as a correlation matrix can prove quite efficient and is presented below.

- **Initial step**: Solve relaxation algorithm. Obtain positive semidefinite matrix $Z$.

- **Step 0**: Perform Cholesky factorization on $Z$. Then $Z = L'L$, where $L$ is a lower diagonal matrix. Set upper bound $upbnd = \infty$ and $k=0$.

- **Step 1**: Set $k = k + 1$. Generate a vector $u$ with elements uniformly distributed in the N-dimensional unit ball. Form vector $w = Lu$ and construct current sample so that it contains the variables with the $M$ largest values in $w$.

- **Step 2**: Solve problem and obtain cost. If cost $< upbnd$, set optimal sample equal to current sample.

- **Step 3**: If maximum number of iterations reached, then terminate, otherwise go to Step 1.

The Positive Semidefinite Randomization Heuristic (PSDH) is more involved computationally, compared with the SRH and M-RH. It involves solving the relaxation
algorithm and performing a Cholesky factorization step. Nevertheless, its appeal comes from the fact that it has been studied in a variety of combinatorial optimization problems where it has been proved to guarantee very good approximation results. For some examples, see [17], [6].

We apply the heuristic using both the interior-point and the Eigenvalue-Cut methods. We also try different accuracies for both. We denote by $Z_{PSDH}$ the upper bound provided by the method. Results are provided separately in Tables 3.3 and 3.4. We tried accuracies of $10^{-6}$ and $10^{-4}$ for the smaller size problems (up to 50 variables) and we found that there was an improvement in the upper bound in the $\langle x'Wz \rangle$ problems, when the "weaker" accuracy $10^{-4}$ was used. There was no difference in the $\langle x'Qx \rangle$ problems. The $10^{-4}$-accuracy running time for SDPA is shorter, since the algorithm terminates faster. This time is shown in the parentheses in the “time” column.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Time (Z_{PSDH}/Z_{OPT} - 1)(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>6</td>
<td>10(0.90) *4.21</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>55.95(6.46) *0.03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Time (Z_{PSDH}/Z_{OPT} - 1)(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>1.40(0.07) 36.07</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>7.97(0.09) 19.08</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>25.82(1.18) 14.36</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>143.4(7.2) &gt; 100</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>484.5(30.8) 0.00</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
<td>1,780.4(116.76) *0.00</td>
</tr>
<tr>
<td>150</td>
<td>25</td>
<td>9,494(615.9) *61.71</td>
</tr>
<tr>
<td>200</td>
<td>40</td>
<td>31,445(1,903) *0.00</td>
</tr>
</tbody>
</table>

Table 3.3: Sample Upper bound results from Positive Semidefinite Randomization Heuristic using the SDPA algorithm. The number of generated samples equals $5N$, where $N$ is the number of binary variables in the problem. The solution accuracy is $10^{-4}$. A star (*) denotes that for the corresponding problem $Z_{PSDH}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{PSDH}/Z_{OPT} - 1$ is larger than or equal to the “starred” numbers shown in the Table.

Note the excellent performance of the eigenvalue-cut method on the flat problems
Table 3.4: Sample Upper bound results from Positive Semidefinite Randomization Heuristic with the Eigenvalue-Cut method. A star (*) denotes that for the corresponding problem $Z_{PSDH}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{PSDH}/Z_{OPT} - 1$ is larger than or equal to the “starred” numbers shown in the Table.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Tolerance</th>
<th>Time (sec)</th>
<th>$(Z_{PSDH}/Z_{OPT} - 1)$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>$10^{-2}$</td>
<td>8.0</td>
<td>0.0021 (20 samples)</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>$10^{-2}$</td>
<td>989.0</td>
<td>0.0013 (10 samples)</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>$5 \times 10^{-2}$</td>
<td>1,196.0</td>
<td>*0.0141 (10 samples)</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>$5 \times 10^{-1}$</td>
<td>2,942.0</td>
<td>*0.0174 (250 samples)</td>
</tr>
</tbody>
</table>

"$x'Qx$". The tolerances used are not the strictest possible with this method, but not too weak also. We observed that the tighter the tolerance, the worse the upper bound. This behavior is similar to the one observed for SDPA and it can be attributed to the fact that in very accurate solutions some $Z_{ii}$'s tend to be closer to 1 than others. This somehow tilts the random samples towards assets corresponding to those $Z_{ii}$'s. These are usually not close to the optimal solution. On the other hand, in less accurate runs, there is more flexibility in random sample generation from the $Z$ matrix solution.

Excluding the relaxation solving part of PSDH, the random part of the heuristic is quite fast. In comparison to SRH and M-RH, it involves the additional step of Cholesky factorization. Random samples are generated through matrix multiplications rather than directly through a random number generator.

The quality of the results obtained with the SDPA method are good, but not as strong as the ones obtained with the Eigenvalue-Cut method. A big advantage of PSDH is that it can run right after a a lower bound generating relaxation algorithm. So both lower and upper bounds are available at almost no additional cost. This makes its application quite appealing, especially in large problems. The Eigenvalue-Cut PSDH method, however, cannot be applied to larger problems because of its inherent limitations.
3.3 Greedy Heuristics

In this section we examine various heuristic algorithms that are based on “greedy” ideas. In its primitive form, the idea consists of selecting variables to be included in the final solution depending on whether the inclusion of the variable contributes to the objective better than the exclusion of it. So, for example, if one considers a minimization problem with an upper bound on the number of variables to be included, one first picks the element that gives the lowest objective value among all elements assuming that only one is selected, then the process is carried on for the second element assuming that the best element from the previous step has already been incorporated in the solution and so on. Below we study three versions of the above idea applied to problems with \{0,1\} variables and we present some indicative results.

3.3.1 Simple Bottom-Up and Top-Down Greedy Heuristic

This is a simple greedy heuristic version. The final solution, including M elements, is constructed step by step. In the bottom-up version, one starts from the relaxed formulation of the problem. Then one fixes each binary variable, one at a time, and solves the resulting program. Then he picks the variable that produced the smaller value for the objective and fixes it to 1. Once this variable is selected the process continues for the rest of the variables fixed again one by one, this time to select the second variable, and so on. At each selection step one has the choice of either leaving all the not-yet-fixed variables relaxed between 0 and 1 (relaxed version of heuristic), or fixing all of them to 0 and changing the maximum-number-of-elements constraint to \( \sum_{i=1}^{i=N} z_i \leq k + 1 \), where k is the step of the algorithm (fixed version of heuristic).

We have observed that this second version of the algorithm does not work as well as the first one, where one allows the nonfixed variables to take any value between 0 and 1.

The Bottom-Up heuristic can be described as follows. In parentheses we show the additional steps required when the fixed rather than the relaxed version of the heuristic is implemented.
• **Step 0:** Consider the relaxed version of the combinatorial problem. Let us call the final solution vector by $F$ (of size $M$). Set $F = \emptyset, k = 1$.

• **Step 1:** Fix all variables in $F$ equal to 1. Consider variables NOT in $F$. Fix one variable equal to 1, leave the rest relaxed (fix them to 0 and set $M=k+1$). Then solve for the cost. Repeat until all variables (outside $F$) have been considered, one at a time. Identify variable with minimum cost. Add it to $F$.

• **Step 2:** Set $k = k + 1$. If $k > M$ terminate, otherwise go to Step 1.

The Top-Down approach is slightly different. Here one follows the opposite direction as shown below. Starting from all elements included, one seeks to identify those that, if excluded, will result in the smallest objective value when the rest are kept.

• **Step 0:** Consider relaxed version of combinatorial problem. Denote the unwanted variables vector by $F$ (of size $N-M$) and the set of all variables $Q$. Set $F = \emptyset, k = 1$.

• **Step 1:** Set all elements in $F$ equal to 0. Set $M = N - k$. Consider variables in $Q - F$. Set each one equal to 0, one at a time. Solve for cost. Repeat until all variables (in $Q - F$) have been considered. Identify variable that results in minimum cost. Add it to $F$.

• **Step 2:** Set $k = k + 1$. If $k < N - M$ terminate, otherwise go to Step 1.

We have observed that the Bottom-Up version works better than the Top-Down version in general. The fixed Bottom-Up version seems to give better results in a few cases compared to the relaxed version, but the relaxed version seems to work better in steeper problems. Below we present some results for the Bottom-Up Greedy heuristic. We denote by $Z_{BUGH}$ the upper bound provided by the method. In the last column, we show in parentheses whether the best solution came from the relaxed (r) or the fixed (f) version.
Table 3.5: Sample Upper bound results for the Simple Bottom-Up Greedy Heuristic. In parentheses in the last column we show whether the best bound has come from the relaxed (r) or the fixed (f) version of the heuristic. A star (*) denotes that for the corresponding problem \(Z_{BUGH}\) was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, \(Z_{BUGH}/Z_{OPT} - 1\) is larger than or equal to the "starred" numbers shown in the Table.

### "x'Qx" Problems

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Time (sec)</th>
<th>(Z_{BUGH}/Z_{OPT} - 1) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>0.8</td>
<td>0.0251 (r)</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>3.0</td>
<td>0.0246 (r)</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>10.0</td>
<td>*0.5448 (f)</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>65.0</td>
<td>*0.0211 (r)</td>
</tr>
</tbody>
</table>

### "x'Qx + z'Wz" Problems

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Time (sec)</th>
<th>Value (f)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>1.0</td>
<td>252.9611</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>5.5</td>
<td>2.0823</td>
</tr>
</tbody>
</table>

### 3.3.2 Greedy-Square \((G^2)\)

The Greedy-Square heuristic is an extension of the simple greedy heuristics presented above. It can run relatively fast and it is very efficient. Furthermore, it does not involve any parameters and can be applied without hesitation to any problem. Its power stems from the fact that one follows a greedy decision process on the outcome of an independent greedy scheme.

To illustrate the idea, consider a set of \(N\) binary variables. Take one of them, say \(z_i\) and set it equal to 1. Then run the Bottom-Up greedy for the rest of the variables. This will provide an upper bound, say \(V_1\). Now fix the same variable to 0 and run the Bottom-Up heuristic for the rest of the variables again. This will provide another value \(V_0\). If \(V_1 < V_0\), then pick \(z_i = 1\), otherwise \(z_i = 0\) and continue the same procedure for the remaining variables until \(M\) elements have been fixed to 1. This is the upper bound estimate of the Greedy-Square.

It turns out that if one applies the fixed version of the Bottom-Up heuristic at each step, the running time of the algorithm can be considerably reduced. This is because previous simple greedy runs come up with sample solutions that apply to later runs.
If the currently examined variable was 1 (0) in the best previous-step greedy run, then one does not need to run the Bottom-Up heuristic with this variable fixed to 1 (0), again. The solution is known. This way a considerable amount of running time can be saved. The cost, as evidenced in the results below, comes in the form of a less-tight upper bound estimate. We call this the *Fixed Greedy-Square* heuristic.

In Tables 3.6 and 3.7 we present results for a range of problems for the Greedy-Square heuristic. We consider both the relaxed and fixed versions. The upper bound is denoted as $Z_{G2r}$ for the relaxed and $Z_{G2f}$ for the fixed version of the heuristic.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Time (sec)</th>
<th>$(Z_{G2r}/Z_{OPT} - 1)$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>3.3</td>
<td>0.0022</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>32.0</td>
<td>0.0143</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>220.0</td>
<td>*0.1086</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>1,260.0</td>
<td>*0.0077</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>6,289.0</td>
<td>*10.386 $\times$ 10$^{-4}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Time (sec)</th>
<th>$(Z_{G2r}/Z_{OPT} - 1)$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>5.7</td>
<td>16.0578</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>64.0</td>
<td>0.1310</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>390.0</td>
<td>0.3066</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>3,024.0</td>
<td>0.2887</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>29,234.0</td>
<td>1.0220</td>
</tr>
</tbody>
</table>

Table 3.6: Sample Upper bound results for Greedy-Square Heuristic (Relaxed Version). A star(*) denotes that for the corresponding problem $Z_{G2r}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{G2r}/Z_{OPT} - 1$ is larger than or equal to the "starred" numbers shown in the Table.

Branching variables in a different order did not make any difference in the results. If there is, however, any relevant information indicating that some variables should be considered first, then that can be very easily (and should be) incorporated in the algorithm.

65
Table 3.7: Sample Upper bound results for Greedy-Square Heuristic (Fixed Version). A star (*) denotes that for the corresponding problem $Z_{G2f}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{G2f}/Z_{OPT} - 1$ is larger than or equal to the “starred” numbers shown in the Table.

<table>
<thead>
<tr>
<th>$x'Qx$ Problems</th>
<th>N</th>
<th>M</th>
<th>Time (sec)</th>
<th>($Z_{G2f}/Z_{OPT} - 1$) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 6</td>
<td>60.0</td>
<td>*0.1103</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 8</td>
<td>374.0</td>
<td>*0.0162</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70 10</td>
<td>4,848.0</td>
<td>*0.0000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x'Qx + z'Wz$ Problems</th>
<th>N</th>
<th>M</th>
<th>Time (sec)</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 4</td>
<td>3.1</td>
<td>19.9346</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 5</td>
<td>22.0</td>
<td>0.8400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30 6</td>
<td>123.4</td>
<td>3.3170</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 8</td>
<td>2,238.0</td>
<td>4.1477</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70 10</td>
<td>8,949.0</td>
<td>3.3474</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.3.3 Greedy-Cube ($G^3$)

The Greedy-Cube method follows exactly the same idea as the Greedy-Square. The difference now is that at each step one solves a Greedy-Square algorithm to decide whether to fix a variable at 1 or 0. This is expected to be more time-consuming computationally, but it is guaranteed to give smaller, or at least equal, upper bounds than the Greedy-Square.

In Table 3.8 we provide results for the G3 heuristic (relaxed version). The upper bound is denoted as $Z_{G3}$. Note the power of the method, but also its long running time which limits its use in larger problems. It is interesting to note, that the Greedy-Cube gives the best upper bound estimates in our “$z'Wz$” problems with up to 50 binary variables! A way to make Greedy-Cube faster would be to run the fixed instead of the relaxed version of Greedy-Square at each node of Greedy-Cube. As we saw above this will speed up the process considerably with a little sacrifice in the bound tightness.

Notice that Greedy-Cube is the only heuristic that has provided the optimal solution in two different problems! This comes as a support of the power inherent in the
<table>
<thead>
<tr>
<th></th>
<th>“$x'Qx$” Problems</th>
<th></th>
<th></th>
<th>“$x'Qx + z'Wz$” Problems</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>M</td>
<td>Time (sec)</td>
<td>$(Z_{G3}/Z_{OPT} - 1)(%)$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>13.0</td>
<td>0.0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>300.0</td>
<td>0.0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>2,233.0</td>
<td>*0.1083</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>13,387.0</td>
<td>*0.0039</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.8: Sample Upper bound results for Greedy-Cube Heuristic (Relaxed Version). A star(*) denotes that for the corresponding problem $Z_{G3}$ was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, $Z_{G3}/Z_{OPT} - 1$ is larger than or equal to the “starred” numbers shown in the Table.

"branch and greedy" idea present in the Greedy-Square and Greedy-Cube algorithms.

### 3.4 Local Search Heuristics

In this section we focus on heuristic algorithms based on local search. Central to this class of algorithms is the idea of the *neighborhood* of a particular solution. In local search one moves from one feasible solution to another trying to go to directions that result in an improvement in the objective value. Being at a feasible solution with a given cost, one first identifies its neighborhood, that is, a set of candidate solutions that could be considered next. For example, in the traveling salesman problem that could be a set of neighboring cities. Afterwards these neighbors are evaluated. One can replace the existing solution with the first improving neighbor (not necessarily the best) and continue from there, or try all neighbors and continue from the one that provides the best improvement from all candidates in the neighborhood. One can choose to adopt only improving moves, in which case the algorithm takes some form of steepest descent (or ascent). In this case, however, practical experience shows
that the algorithm tends to terminate itself in a local optimum which can very well be quite far from the optimal solution. One can escape such local optima by sometimes adopting solutions that are worse than the existing one. Furthermore, one can prevent going back to already examined cases by establishing some form of memory for past moves. This list acts like a 'tabu' list for moves. Along with certain elaborations this is the main idea behind the so called Tabu and Reactive Tabu search algorithms. This class of algorithms has been shown to perform quite well in a range of combinatorial optimization problems like, for example, the quadratic assignment, graph coloring, vehicle routing and course scheduling problems [22]. Nevertheless, Tabu search is more like an engineering approach to large and difficult optimization problems rather than an elegant algorithm of mathematical simplicity. Complexity is not only present in the problem but also in the technique itself.

A simpler yet effective approach to local search is the simulated annealing algorithm. This originated in the 1950s in condensed matter physics as a method to simulate annealing, the thermal process of obtaining low-energy states of a solid in a heat bath. In the context of optimization, simulated annealing belongs to the wider category of threshold local search algorithms. Threshold algorithms continually select a neighbor of a current solution and compare the difference in cost between these solutions to a threshold. If the cost difference is below the threshold the neighbor replaces the current solution. Otherwise the search continues from the current solution. Three variants of this approach can be distinguished [1]. One is iterative improvement in which the threshold is kept at zero all the time and corresponds to the classical local descent algorithm. A second approach is threshold accepting in which a nonincreasing sequence of thresholds is established. This sequence goes to zero eventually. Due to the use of positive thresholds neighboring solutions with larger costs are accepted in a limited way. As mentioned earlier, this facilitates escaping local optima. The third approach is simulated annealing. Here, one uses randomized thresholds with values between zero and infinity and the thresholds are established according to a certain distribution function. A parameter, called temperature $T$, is included in the distribution function. Temperature is gradually reduced along the course of the al-
algorithm. The purpose of doing that is to push thresholds to lower and lower values, so that eventually only improving neighbors are kept. A very popular and successful choice for the distribution is the exponential $\exp(-x/T)$. This approach has been used with success on a variety of problems and is much simpler than tabu search. On the other hand it involves certain control parameters which implies that experimentation is required for different problems. This comes in contrast with the greedy family of algorithms and especially the greedy-square which can be applied "as is" across different problems. The parameters involved in simulated annealing are the number of iterations (neighbor searches) at the same temperature, the number of temperature reductions, the actual amount of temperature reduction each time (cooling schedule) and the initial temperature. We continue with a more detailed description of the simulated annealing algorithm below.

- **Initial step**: Pick sample $S$ of $M$ variables randomly. Evaluate cost $f(S)$. Set upper bound $upbnd = f(S)$. Set temperature $T = |f(S)|/10, k = 0$ and $t = 0$.

- **Step 1**: Pick randomly variable from $S$ and select randomly a neighbor-variable of it that does not already belong to $S$. Form new sample $C$ and evaluate $f(C)$.

- **Step 2**: If $f(C) < f(S)$ set $S = C, upbnd = f(S)$ and go to Step 3, else generate uniform random variable $r \in [0,1]$. If $r < \exp[-(f(C) - f(S))/T]$, set $S = C$ and go to Step 3.

- **Step 3**: Set $t = t + 1$. If $t$ larger than specified maximum number of iterations at same temperature, then set $T = \alpha T$ (cooling), $k = k + 1$ and $t = 0$. If $k$ larger than number of temperature reductions, then terminate. Otherwise go to Step 1.

An initial sample in the above algorithm can be picked in different ways. One could be by randomly generating one sample of $M$ variables as suggested above. Different ways would be to set the initial sample equal to the outcome of another heuristic algorithm. That could be anything, from the SRH to greedy-square. As one expects, the better the initial sample is, the less room for improvement from the simulated
annealing technique there is. We feel it is preferable at this stage to start the method
from a randomly generated sample to get a better feeling of what it can do. To
pick a neighbor of variable \( i \), a direction is picked randomly between “upwards” and
“downwards”. If “upwards” is picked then we move to variables \( k = i+1, k = i+2, \ldots \)
until we find the first \( k \) that does not belong to the current sample. Similarly we move
to \( k = i - 1, k = i - 2, \ldots \), if the “downwards” direction has been selected. The choice
of the initial temperature and cooling schedule are quite important. Here we take
the initial temperature to be the initially available cost divided by 10. This is an
empirical rule and seems to work quite well in our problems. As for the cooling
schedule, this is controlled by \( \alpha \) and the number of temperature reductions allowed.
Values of \( \alpha \) are typically around 0.95.

In Table 3.9 we present some indicative results for the problems considered for
the previous heuristics as well. The upper bound is denoted by \( Z_{SAH} \). For the results
below, the number of temperature reductions is set to 100, the number of repetitions
at the same temperature is set at \( 2M \) and \( \alpha \) is 0.95. The starting sample is randomly
generated and the initial temperature is set equal to the \( 1/10 \) of its cost.

Regarding bound quality, the Simulated Annealing Heuristic works sometimes
better, sometimes worse than the Randomization Heuristics. It is slower but also
more reliable. With respect to the Greedy family, it is faster in general, but it does
not provide bounds as tight as these methods, especially in larger problems.

We note the general slightly worse performance, but faster running time for larger
problems in the simulated annealing versus the greedy-square heuristic. We also
mention that if one uses the greedy-square solution as initial sample, there is no
improvement resulting from simulated annealing in the problems considered above.
Compared to randomization heuristics, simulated annealing, as well as the greedy
techniques, can be considered to be more trustworthy, in that they are expected to
perform quite well (as in these examples) in a wide variety of problems. These can
not be said for randomization techniques in general.
<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Time (sec)</th>
<th>( \frac{Z_{SAH}}{Z_{OPT}} - 1 ) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>14</td>
<td>0.0424</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>52</td>
<td>0.0091</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>104</td>
<td>*0.1175</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>357</td>
<td>*0.0155</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>863</td>
<td>*99.81 \times 10^{-4}</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
<td>3,209</td>
<td>*0.4813</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Time (sec)</th>
<th>( \frac{Z_{SAH}}{Z_{OPT}} - 1 ) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>20</td>
<td>10.7205</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>75</td>
<td>0.5404</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>187</td>
<td>1.5460</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>882</td>
<td>1.8532</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>3,259</td>
<td>&gt; 100</td>
</tr>
</tbody>
</table>

Table 3.9: Sample Upper bound results for the Simulated Annealing Heuristic. The starting sample is randomly generated. The number of temperature reductions is set to 100, the number of repetitions at the same temperature to \( 2M \), where \( M \) is the number of “1” binary variables in the problem and \( \alpha = 0.95 \). A star(*) denotes that for the corresponding problem \( Z_{SAH} \) was divided by the best available upper bound instead of the optimal solution. An optimal solution was not available for these problems from Branch and Bound. In these cases, \( Z_{SAH}/Z_{OPT} - 1 \) is larger than or equal to the “starred” numbers shown in the Table.

### 3.5 Summary

We conclude this chapter by providing an overview of the heuristics performance. We consider “\( x'Qx \)” and “\( z'Wz \)” problems separately for all sizes up to 100 variables. The purpose for doing this is to provide a clearer picture of the solution accuracy versus running time trade-off across all heuristics and to illustrate the different behavior of our heuristics in the flatter “\( x'Qx \)” versus the steeper “\( z'Wz \)” problems.

In Figure 3-1 we present the “\( x'Qx \)” problem case. We plot the Heuristic Upper Bound divided by the Optimal Solution (or the best available upper bound for problems with \( N \geq 30 \)) versus the logarithm of the running time in seconds. In the Positive Semidefinite heuristic we have included additional runs when for example we have results available from both the SDPA and Eigenvalue-Cut methods. The same is true for the Greedy-Square Relaxed and Fixed versions. The y-axis of the figure
has been rescaled to show with more clarity the efficient frontier which is formed by the “south-most” points. These correspond to the heuristics that provide the best upper bound performance for a given amount of time. Due to the y-axis rescaling some data points are not shown in Figure 3-1.

Note that randomization heuristics (especially the SRH and M-RH) have quite a few points on the frontier. The same is true for the Greedy-Cube and the Greedy-Square methods. Their points always lie on, or close to, the frontier. The Greedy heuristics, however, are more expensive computationally. Notice also the abundance of good upper bound estimates. All runs in Figure 3-1 correspond to upper bounds closer than 0.15% from the optimal solution or the best available upper bound.

In Figure 3-2 we present similar results in the case of the steeper “$z'Wz$” problems. Here note the very good performance of the Greedy Heuristics (G2 and G3) as well as the M-Randomization Heuristic. These should be the heuristics of choice in such problems. Figure 3-2 has also been rescaled and all points correspond to upper bounds not further than 10% from the optimal solution or the best available upper bound. Note that fewer points now lie in this range than in the “$x'Qx$” problem case. This is another demonstration of the difficulty heuristics have dealing with steeper rather than flatter problems.
Figure 3-1: Algorithmic performance of the heuristics in the case of $x'Qx$ problems. All problem sizes from 10 to 100 variables are shown. We include the Simple Randomization Heuristic (SRH), M-Randomization Heuristic (M-RH), Positive Semidefinite Heuristic (PSDH), Greedy-Square (G2), Greedy-Cube (G3) and the Simulated Annealing Heuristic (SAH).
Figure 3-2: Algorithmic performance of the heuristics in the case of $z'Wz$ problems. All problem sizes from 10 to 100 variables are shown. We include the Simple Randomization Heuristic (SRH), M-Randomization Heuristic (M-RH), Positive Semidefinite Heuristic (PSDH), Greedy-Square (G2), Greedy-Cube (G3) and the Simulated Annealing Heuristic (SAH).
Chapter 4

Exact Algorithms

In this chapter we investigate exact solution algorithms for the mixed integer-quadratic problem. The main idea of the proposed methods is the branch and bound scheme. In Section 4.1 we provide a general description of the branch and bound algorithm and illustrate its strengths and limitations. In Section 4.2 we investigate various ways to improve the method’s efficiency by incorporating relaxation and heuristic algorithms. In Section 4.3 we apply exact methods to the Portfolio replication problem. Results are presented for both the “$x'Qx$” and “$z'Wz$” type of problems.

4.1 Branch and Bound

Branch and Bound uses a “divide and conquer” approach to span the set of feasible integer solutions to an integer programming problem [5], [27]. It does not consider the entire feasible set, however. Using bounds on the optimal costs it tries to avoid examining as many feasible solutions as possible. To illustrate the idea let us assume that our problem is a purely integer one, as shown below:

$$V_* = \text{Minimize } f(z)$$

$$s.t. \quad z \in F$$
In the above, $f$ is an arbitrary concave function of $z$, $z$ takes purely integer values and $F$ is the set of integer feasible solutions to the problem. One can partition $F$ into a finite collection of subsets $F_1, \ldots, F_k$ and solve each of the subproblems separately. Then, the solutions, which we denote by $V_i$, would be compared and the smallest one would be chosen. Solving a subproblem, however, can be as difficult as solving the original problem. This would suggest following the same splitting approach for each subproblem and so on. This results in branching, which manifests itself as a tree of subproblems.

Now assume that one has a quite efficient algorithm that, for each subproblem $F_i$, provides a lower bound $b(F_i)$ to the optimal solution, so that $b(F_i) \leq V_i$. The idea is that it might be a lot easier to obtain a lower bound to $V_i$, than $V_i$ itself. Such a bound could be obtained, for example, from the relaxed version of the subproblem. Now note that in the course of the algorithm one will be able to maintain an upper bound $U$ to the problem's optimal solution $V_*$ from solving certain subproblems to optimality or by simply evaluating the cost of certain feasible solutions.

The main point of Branch and Bound lies in the following observation. If the lower bound to a particular subproblem satisfies $b(F_i) \geq U$, then then this subproblem need not be considered further, since its optimal solution $V_i$ is going to be larger than $U$ which in turn is larger than the problem's optimal solution $V_*$. The algorithm keeps a set of outstanding (active) subproblems $L$ starting with $L = \emptyset$. The lowest upper bound $U$ is also kept. Initially it is set to $\infty$ or some other value obtained from an appropriate method, like for example a heuristic algorithm. The algorithm works as follows:

- **Initial step**: Set $U$. Break problem into set of subproblems $L \equiv F_i, i = 1, \ldots, k$.

- **Step 1**: If $L = \emptyset$ terminate. Otherwise, select an active subproblem $F_i \in L$.

- **Step 2**: If $F_i$ infeasible, delete it and go to Step 1, otherwise compute $b(F_i)$.

- **Step 3**: If $b(F_i) \geq U$ delete subproblem and go to Step 1. If $b(F_i) < U$ either obtain an optimal solution to the subproblem, or branch the subproblem further,
adding the resulting subproblems to \( L \). In case an optimal solution \( V_i \) to the subproblem is obtained, check whether \( V_i < U \). If yes, then set \( U = V_i \) and go to Step 1. If not, delete subproblem and go to Step 1.

There are several ways the above algorithm can be implemented in practice. Decision needs to be taken regarding how an active subproblem will be chosen, a lower bound will be obtained, and how a problem will be broken into subproblems. On the first question, quite a few choices exist, the most extremes being breadth-first search and depth-first search. Regarding the lower bound one approach is to use the corresponding relaxed version of the problem. Other methods could be to use positive semidefinite relaxation algorithms like those described in Chapter 2. Note, that the speed of obtaining a lower bound is quite crucial for the Branch and Bound algorithm to terminate at a decent time. A computationally expensive method that provides tight lower bounds might reduce significantly the number of considered subproblems, but it would increase the running time of Branch and Bound. More on this will follow shortly.

Now we concentrate on the mixed integer-quadratic problems studied in this thesis. The integer variables involved are binary, meaning that they can assume only value 0 or 1. We consider problem (2.1) which is restated here for convenience.

\[
Z_{OPT} = \text{Minimize} \quad x'Qx + z'Wz + c'_x x + c'_z z \\
\text{s.t.} \quad a'_i x \leq h_i, \quad i = 1, \ldots, m_x \\
b'_j z \leq r_j, \quad j = 1, \ldots, m_z \\
c'_l x + g'_l z \leq d_l, \quad l = 1, \ldots, m_{xz} \\
x \geq 0, \quad z \in \{0,1\}^N.
\]

A straightforward way to branch in this case is to pick a variable \( z_i \) and create two subproblems \( F_0 \) and \( F_1 \) with \( z_i = 0 \) and \( z_i = 1 \) respectively, with all other variables relaxed between 0 and 1. Repeating the procedure from these two subproblems for
another binary variable results in a tree like the one shown in Figure 4-1.

Note that at the first branching level (one variable fixed), one has two problems (nodes), at the second level 4 subproblems, at the $i$th level $2^i$ nodes etc. If the problem is N-dimensional, then at the $N$th tree-level a complete enumeration of all possible binary value combinations is available, that is $2^N$ combinations. The total number of tree nodes is $2^{N+1} - 1$, clearly indicating an exponential increase on the problem’s dimension. For $N = 100$ this amounts to $\sim 2 \times 10^{30}$ nodes! In the case of particular interest to us, where the $\sum z_i = M$ constraint is present, we note that the total number of feasible nodes would be equal to $\sum_{l=1}^{l=N} \sum_{k=\min(l,M)}^{k} \binom{l}{k}$ which for $N = 100, M = 20$ would be $\sim 3.5 \times 10^{21}$! These huge numbers give a first glimpse at the difficulty of solving even moderate-size problems using exact Branch and Bound algorithms.

The presence of continuous variables $x$ can complicate feasibility tests of the mixed constraints and checking even for purely-$z$ constraints can be quite involved in general. One should first check the purely $z$-constraints, then modify the mixed constraints according to the fixed $z$-values and check for consistency.

Two important decisions in the course of the algorithm are node selection and variable selection. The first refers to the node that will be selected next for consideration among all active subproblems. The second relates to the variable that will be chosen to branch upon next at the current node. Two possibilities for node selection are depth-first search and breadth-first search. In the first case, the next subproblem will be one of the children of the current node. In the second case, it will be one of the same-tree-level nodes. Depth-first emphasizes moving down the tree rather than across each level. A choice can also be made in depth-first, whether the “1” or the “0” child will be considered first. Other node selection rules can be devised according to the particular problem structure. For more details the reader is referred to [27] and references therein.

Regarding variable selection, the possibilities are numerous and no robust methods have been established for that purpose. One way this could be done is to assign user-
Figure 4-1: Schematic of the Branch and Bound method. All nonfixed variables are assumed to be relaxed.
specified priorities among variables according to what one feels should be considered first if a choice was possible. Other possibilities include assigning penalties to each variable, that provide an estimate of the increase in the objective value when this variable is fixed to 1 or 0 from its current relaxed value. The variable with the highest penalty would be considered next in that case. Note, however, that such an approach would be more appropriate in purely integer programs. In our case, it is very hard to obtain fast an estimated change in the objective value when a $z$ variable is set to 1 or 0 from its current relaxed value without reoptimizing for $x$. Another approach to variable selection would be to sort the nonfixed $z$ variables according to which one is closest to 0 (or 1) and branch that one first. If the problem is quite flat, however, the relaxed $z$ values can be quite close to each other thus invalidating the idea. For the problems we have examined in this thesis, we have found out that the ability to obtain tight lower and upper bounds is much more important for performance improvement than variable selection rules. The latter adds little value when good bound generating algorithms are available.

The outline of the Branch and Bound algorithm follows the general one presented above. It is shown below.

- **Initial step**: Set $U$. Select variable. Branch to 0 and 1, creating subproblems $F_0$ and $F_1$. Set $L = F_i$, $i = 0, 1$.

- **Step 1 (Node Selection)**: If $L = \emptyset$ terminate. Otherwise, select an active sub-problem $F_i \in L$.

- **Step 2 (Feasibility Test, Lower Bound Estimation)**: If $F_i$ infeasible, delete it and go to Step 1, otherwise compute $b(F_i)$.

- **Step 3 (Pruning or Branching)**: If $b(F_i) \geq U$ delete subproblem and go to Step 1. If $b(F_i) < U$ either obtain an optimal solution to the subproblem, or branch the subproblem further, adding the resulting subproblems to $L$. In case an optimal solution $V_i$ to the subproblem is obtained, check whether $V_i < U$. If yes, then set $U = V_i$ and go to Step 1. If not, delete subproblem and go to Step 1.
We still need to emphasize that the problem structure is very important. Problems with very flat objectives tend to have their relaxed $x$ and $z$ values close to each other and away from 0 or 1. In addition, pruning is difficult at early nodes since a number of variables has to be fixed first in order to push lower bound estimates close to the upper bound. In many cases, even fixing one variable does not lead to any significant lower bound improvement.

4.2 The Use of Bounds in Branch and Bound

As we stressed in the previous section, the use of efficient bound generating routines is crucial for Branch and Bound to be of any practical use. In most real life problems, even if the actual solution was known from the beginning, proving that it really is the optimum can take a lot of time. This is because relaxed suboptimal nodes can very well produce lower bound estimates that are quite smaller from the optimal solution, thus forbidding early pruning. Quite a few binary variables will have to be fixed for the lower bound to appreciate enough in value. This effect is especially pronounced in problems with relatively flat objectives.

One needs relaxation algorithms that provide lower bounds as close as possible to the actual integer-feasible solution of the subproblem. As one would expect, this comes at a cost. The running time of a good such algorithm like, for example, one that is based on a positive semidefinite formulation can be quite longer than an estimate obtained from just solving the corresponding relaxed version of the subproblem. The latter can fail reducing the number of nodes considered, but the Branch and Bound algorithm can still run much faster than a Branch and Bound version with a positive semidefinite relaxation algorithm. In that case, the number of nodes may be reduced significantly but the overall running time of the method can be much higher. To give the reader an idea, in a small problem with $N = 10, M = 4$ the total number of nodes is 1,022. Branch and Bound based simply on the relaxed version of each subproblem, yields the result in 7 seconds after considering 394 nodes. A version based on the eigenvalue-cut positive semidefinite algorithm of Chapter 2 yields the
result in 950 seconds after 206 nodes. The number of nodes is reduced in half but the overall running time is increased by two orders of magnitude! Nevertheless, significant tree pruning can prove quite beneficial in large problems even if an expensive relaxation algorithm is used. In addition, one can avoid running the expensive relaxation algorithm at every node. It can be run selectively at nodes at particular tree levels, where there is a good chance of pruning. For example, one can experiment and determine that when any two binary variables have been fixed to 1 there is a very good chance for pruning. Or he could just run the relaxation at every even-numbered level of the tree. Also note, that the positive semidefinite relaxations need not be run with strict primal-dual tolerances. This can save time and does not affect the bound considerably.

On the opposite side of early pruning, lies a good upper bound estimate. A tight upper bound will lie closer to the optimal solution and to lower bound estimates. Heuristic algorithms like the ones presented in Chapter 3 can be very valuable. One can run such a heuristic in the beginning, before the Branch and Bound, to provide a tight bound to start with. If the heuristic is fast, it can be repeatedly run during the course of Branch and Bound to accelerate improvement of the upper bound. Alternatively, a more expensive heuristic can be executed at selected points in the tree.

It is clear from the above that experimentation with different relaxation and heuristic schemes is required to arrive at a Branch and Bound algorithm that works well for different kinds of problems. On the practical implementation side, no matter what relaxation algorithm is used to obtain lower bounds, the problem will be formulated as demonstrated in Chapter 2. The only difference is that one imposes additional constraints on the $z$ variables that have been fixed. For example, in a positive semidefinite relaxation formulation, if at the current node $z_i = 0$ then one sets $Z_{ij} = 0$, $\forall j$, otherwise if $z_i = 1$ then one sets $Z_{ii} = 1$ and $Z_{ij} = Z_{jj}$, $\forall j \neq i$. The options for relaxation algorithms at one’s disposal are quite a few. One can simply solve the relaxed version of the subproblem, use the simple positive semidefinite formulation (2.6), or the tighter (2.10). The eigenvalue-cut method can also be used,
although it usually fails to provide strict lower bounds, especially for large problems. Finally one can employ the quadratic positive semidefinite approach, which provides stricter bounds than the previous formulations in general, sometimes with no significant increase in running time. Regarding the heuristics, one first sets the fixed binary variables at their corresponding values and applies the heuristic on the rest of them. The M-Randomization heuristic works quite well and is fast. The positive semidefinite randomization heuristic represents no significant computational cost when is executed after a positive semidefinite relaxation aimed at providing a lower bound. It also works quite well. The greedy-square heuristic is another possibility, although it can prove costly if run at every node.

An additional feature of the Branch and Bound algorithm can be a solution tolerance parameter specified by the user. This number is a measure of how close to the actual optimal solution is the Branch and Bound solution guaranteed to be upon termination. So, if the user specifies a tolerance parameter of 5%, then the Branch and Bound method guarantees a solution which is at the most 5% away from the optimal. This idea is common in commercial Branch and Bound programs available for linear mixed-integer programming problems. Its implication in the flow of the algorithm is that a node is now pruned when the $b(F_i) \geq (1 - \epsilon)U$ condition, instead of the more strict $b(F_i) \geq U$, is satisfied. Note, that $\epsilon$ is the solution tolerance parameter ($\epsilon = 0.05$ for 5% tolerance). It is clear that the introduction of the solution tolerance parameter speeds up the algorithm at the cost of guaranteeing a solution within a certain range of the optimal solution, instead of the optimal solution itself. This is not a drawback. On the contrary, in many cases the exact solution is impossible or pointless to find, for example when the problem is of very large size or flat. It is then preferable to have an $\epsilon$-accurate solution than nothing.
4.3 The Portfolio Problem - Branch and Bound Results

In this Section we apply the Branch and Bound algorithm in an attempt to solve the Index and Portfolio replication problem exactly. The Branch and Bound idea guarantees the exact solution at termination, but in its pure form it faces quite a few challenges in solving problems of even a moderate size. We start with a general discussion and we provide results in Sections 4.3.1 and 4.3.2.

To the best of our knowledge, Bienstock [7] is the only work published on methods to solve mixed integer quadratic problems. In particular, Bienstock examines the Portfolio problem as formulated in Section 1.4. He considers a Branch and Cut scheme and various heuristics to assist in tree pruning. He avoids to introduce binary variables and formulates additional constraints on the \( x \) variables, that, as he argues, serve as substitutes to the cardinality constraint \( \sum z_i = M \). He considers compact problems (low-rank \( Q \) problems) with up to 3897 variables and with \( M \) between 20 and 25. He mentions, however, that at most 50 variables are positive in the relaxed version of these problems. This is in contrast with our case, where all the \( x \) values lie close together and strictly within bounds. Non-compact problems are also studied of similar size and \( M = 25 \ldots 40 \). Again, he mentions that the number of positive variables lies between 40 and 100 with many of them having values smaller than \( 10^{-7} \). Bienstock acknowledges, that problems with a few hundred positive variables and fairly flat objectives will be difficult to solve with his method, since no cutting will take place. This agrees with our experience. The author also considers parallel implementations of Branch and Bound. Parallel computing approaches for solving mixed integer quadratic programs have been considered also by Lee and Mitchell [24] and Borchers and Mitchell [8, 9]. It is worth mentioning here the work of Patel et al. [31]. The authors provide a simple algorithmic approach for solving the cardinality constrained Portfolio problem exactly when all off-diagonal elements of the correlation matrix are equal to a constant. Although this may have limited practical use by itself, it can lead to a quite efficient randomization heuristic as we explained in Section 3.2.2.
To get a better feeling of the difficulty of solving the Index Replication problems exactly, recall that in the presence of the combinatorial constraint $\sum z_i = M$, the total number of tree nodes that one would have to consider (assuming no pruning takes place) is $\sum_{i=1}^{I} \sum_{k=0}^{\min(I,M)} \binom{i}{k}$. In Table 4.1 we show this number for the problems considered in this thesis. Multiplying with the time it takes to solve one such node (for the faster “$x'Qx$” problem), from Table 2.2, we also give an estimate of the overall required time.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Number of nodes</th>
<th>Time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>1.022000e+03</td>
<td>0.009</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>8.215800e+04</td>
<td>0.7</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>3.572222e+06</td>
<td>61</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>3.815481e+09</td>
<td>169,577</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>3.108886e+12</td>
<td>2.51e8</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
<td>1.957555e+18</td>
<td>4.4e14</td>
</tr>
<tr>
<td>150</td>
<td>25</td>
<td>1.427381e+29</td>
<td>1.07e26</td>
</tr>
<tr>
<td>200</td>
<td>40</td>
<td>1.343852e+43</td>
<td>2.6e40</td>
</tr>
</tbody>
</table>

Table 4.1: Total number of nodes and estimated running times in Branch and Bound assuming that no pruning takes place.

It can be said that problems with up to 20 binary variables are tractable. From there the number of nodes and running time estimate increase in an explosive manner. In this context it becomes clear that very efficient pruning techniques are needed to reduce the size of the tree. The node cutting ability of these techniques has to increase at a rate similar to the one observed above, if one is to solve large problems.

In the last two chapters we presented results for lower and upper bounds to the optimal solution. These bounds were produced by relaxation and heuristic algorithms. It can be said, that producing tight such bounds relatively fast provides the most important means of reducing the tree size significantly. Just to give the reader an idea, we present below the number $\lambda = 2 \frac{UB - LB}{|UB + LB|}$, where $UB$ and $LB$ are the upper and lower bounds provided by the above algorithms. This number gives us an idea of how close to the optimal solution we are after running just these approximation algorithms. If it is 0%, then the optimum has been determined with certainty. The
numbers are presented in Table 4.2.

From Table 4.2 observe the excellent performance of our lower and upper bounds in the case of the \( x'Qx + z'Wz \) problems with up to 150 variables. In these cases we can say where the optimal solution lies with more than 90% accuracy without even running Branch and Bound. For the \( x'Qx \) problems it is evident that there is a lot of work to be done in Branch and Bound to solve problems larger than 20 variables. It could be said with safety, that the problem has to do with the lower bound rather than with the upper bound estimates. Our heuristics seem to work very well for all problems with up to 150 variables. In the cases where an exact solution was found, this was not further than 1% from the best heuristic result. Branch and Bound for these problems should rather rely on heuristics rather than relaxation schemes to reduce the tree size.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>&quot;x'Qx&quot; ( \lambda(%) )</th>
<th>&quot;x'Qx + z'Wz&quot; ( \lambda(%) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>85.55</td>
<td>8.85</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>117.52</td>
<td>5.99</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>131.76</td>
<td>4.43</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>124.30</td>
<td>2.59</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>149.86</td>
<td>1.93</td>
</tr>
<tr>
<td>100</td>
<td>15</td>
<td>147.63</td>
<td>4.13</td>
</tr>
<tr>
<td>150</td>
<td>25</td>
<td>142.61</td>
<td>( 9.58 \times 10^{-4} )</td>
</tr>
<tr>
<td>200</td>
<td>40</td>
<td>n/a</td>
<td>97.23</td>
</tr>
</tbody>
</table>

Table 4.2: Total number of nodes and estimated running times in Branch and Bound assuming that no pruning takes place.

We consider various choices for node selection, of the relaxation algorithm for generating the lower bound at each node and on whether a variable will be branched first to 0 or 1 (branching value).

In particular the following options are considered:

* **Node Selection**
  - Depth-first search
• Breadth-first search

Relaxation Algorithm Selection

• Relaxed version

• Positive Semidefinite Relaxation using SDPA

We consider exact as well as $\epsilon$-approximation solutions. The value of the $\epsilon$ parameter is smaller than the $\lambda$ numbers of Table 4.2. In addition, we do not consider $\epsilon$-approximate solutions in problems that can be solved exactly.

Regarding the upper bound we do not consider running a heuristic at each node. This would add a lot of time in the overall running time of the method. Besides, our heuristics have provided very good upper bounds for the problems we are studying. We start with the best upper bound estimate from Chapter 3.

Also we are not considering the quadratic positive semidefinite relaxation for Branch and Bound. Although it has demonstrated stronger bounds than SDPA when variables are fixed, its implementation depends critically on the starting parameters and it is sensibly slower for problems larger than 30 variables in terms of running time.

As a general note, we mention that the Quadratic CPLEX solver is configured so that it optimizes using the previous run as a warm start. This can save a considerable amount of time. Also, the interior-point positive semidefinite solver SDPA is run with accuracy $10^{-4}$. This is a good compromise for saving time without losing on the lower bound’s quality. In smaller problems we have tried to run all node selection and branching choices, that is, depth-first with variable first branched to 1 (DF-1) or 0 (DF-0) and breadth-first with variable first fixed to 1 (BF-1) or 0 (BF-0).

In contrast with upper bound generating algorithms, Branch and Bound faces a challenge in problems with flat rather than steep objectives. In flat problems early pruning is more difficult, resulting in larger enumeration trees. It is interesting to have that in mind when assessing its performance in the following lines.

87
4.3.1 Branch and Bound with the Relaxed Version

In this subsection we run Branch and Bound using the simple relaxation formulation (2.17). This is the "cheaper" choice for obtaining a lower bound at each node in terms of computational time. The lower bounds are weak though. This is evident from the fact that we cannot consider problems larger than 30 variables with this method, due to the rapidly increasing number of nodes with the problem size.

Results are presented in Table 4.3 in the case of $\epsilon = 0$ (exact solution). We also show the percentage of the number of nodes considered relative to the total possible. Note that no heuristic upper bound was applied initially in these results. We did not observe much difference when such a bound was applied. Node reduction was 3% in the best case. For example, in the $N = 20$ "$z'Wz$" problem solved by DF-1 the method considered 1134 nodes instead of 1159 when the initial upper bound was set a little higher than the best available estimate. Observe the small decrease in the percent number of nodes with the problem's size. This demonstrates the limitation of this approach in larger problems.

Observe that solving the less flat "$x'Qx + z'Wz$" problems is much easier. In fact, such a problem with 30 variables could be solved, in contrast with the corresponding flatter "$x'Qx$" version, where the system run out of memory due to the increasing number of nodes in the tree.

Depth-first search with a branching choice to 1 seems to be working the best for problems "$x'Qx$" in this version of Branch and Bound. Branching to 1 rather than 0 means going down rather than across the tree allowing the algorithm to arrive at feasible nodes faster. This way upper bound improvement can occur earlier. Subsequent higher-level nodes may then be pruned earlier than if a breadth-first scheme or branching to 0 was employed. Note, that this behavior is not that evident in the "$z'Wz$" problems, where the most important choice seems to be branching value rather than node selection. Keep in mind, however, that branching to 0 and breadth-

\[^4\]Because of the flatness in the objective, branching a variable to 0 will have a minimal effect in the relaxed value of the resulting problem. This is not so when a variable is branched to 1, because of the $\sum z_i = M$ constraint.
<table>
<thead>
<tr>
<th>( x'Qx ) Problems</th>
<th>( x'Wz ) Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>( M )</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>( N )</td>
<td>( M )</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 4.3: Branch and Bound with use of the simple relaxed formulation. Exact solution version.

First search can prove better choices in a scheme where good lower bound algorithms are employed.

For the difficult \( x'Qx \) problems we tried to solve for an 0.1-approximate solution using Branch and Bound. In addition, the best available upper bound was used from the beginning. In the 10 and 20 variable cases, this coincides with the optimal solution, so the corresponding time and nodes show how costly it can be just to prove optimality in these problems. The results are presented in Table 4.4.

Notice from Table 4.4, that the reduction in nodes is at the most 9% compared to Table 4.3. This is not a significant improvement. Also, the number of nodes is the same regardless of the branching method.

However, there seems to be a significant improvement in the larger (30 variables) problem. The problem is now solved after 374,990 nodes and the 0.1-solution equals the best upper bound, which was also the starting one. We mention that when we
<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Problem type</th>
<th>Method</th>
<th>Number of nodes</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>$x'Qx$</td>
<td>DF-1</td>
<td>360</td>
<td>6.9</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>$x'Qx$</td>
<td>BF-1</td>
<td>360</td>
<td>6.9</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>$x'Qx$</td>
<td>DF-1</td>
<td>21,932</td>
<td>1,061.0</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>$x'Qx$</td>
<td>BF-1</td>
<td>21,932</td>
<td>1,061.0</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>$x'Qx$</td>
<td>DF-1</td>
<td>374,990</td>
<td>29.390.0</td>
</tr>
</tbody>
</table>

Table 4.4: Branch and Bound with use of the simple relaxed formulation. $\epsilon = 0.1$ solution version.

made an attempt to solve this problem exactly and without a starting upper bound, the system memory signaled overflow after considering 531,752 nodes. So the node reduction in this case is at least 30% and the number of nodes is 10.5% of the total possible. It is quite unprobable that the 50 variables problem can be solved this way however. The total number of nodes is three orders of magnitude larger than the $N = 30$ case and even if pruning reduces the tree by even an order of magnitude more, this is still not enough.

### 4.3.2 Branch and Bound with Positive Semidefinite Relaxation

We saw in the previous section that Branch and Bound, using the relaxed version of the problem as a lower bound generator, is quite limited. Only problems with up to 30 variables can be solved and for the flat "$x'Qx$" problems even using the optimal solution as a starting upper bound and requiring a 0.1-approximate solution is of help. The problem lies predominantly on the weak lower bounds that the relaxed formulation provides.

The Positive Semidefinite formulation provides much tighter lower bounds, at least for the “steeper” "$z'Wz$" problems. Despite the fact that this comes at the cost of increased running time, it can prove invaluable for larger problems where early node pruning is much more important. The benefit translates into the ability of solving larger problems than before in a reasonable amount of time.

In this section we will try to solve larger problems. We consider "$z'Wz$" problems
with up to 70 variables and "$x'Qx$" problems up to 50 variables. At each node the SDPA algorithm is applied to determine the lower bound. If this is larger or within $\epsilon$ (for a $\epsilon$-approximate solution) of the current upper bound, pruning occurs. The best available upper bound is used initially in all cases. In addition, a randomization heuristic is applied at each node, that takes into account which variables have already been branched and generates $N$ samples. We do not consider $\epsilon$-approximate solutions for the "$z'Wz$" problems, since the lower and upper bounds available at the beginning provide tight ranges for the optimum as it was demonstrated in Table 4.2. Such approximate solutions with $\epsilon = 0.1$ will be considered for the "$x'Qx$" problem with 30 and 50 variables. Accuracy for the SDPA is set to $10^{-4}$. The results are shown in Table 4.5.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Method</th>
<th>Number of nodes</th>
<th>% No.</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>DF-1</td>
<td>242</td>
<td>23.68</td>
<td>113.0</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>DF-1</td>
<td>14,257</td>
<td>17.39</td>
<td>69,819.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&quot;$z'Wz$&quot; Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>70</td>
</tr>
</tbody>
</table>

Table 4.5: Branch and Bound with use of the positive semidefinite algorithm SDPA. Exact solution for "$z'Wz$" and 0.1-approximate solution for "$x'Qx$" problems.

Note the dramatic reduction in the number of nodes, especially in the "$z'Wz$" case. Although this is accompanied with increased running time for smaller problems, it speeds up the overall algorithm in larger problems and allows one to solve problems with up to 70 binary variables. Note, however, that the "$x'Qx$" case is still quite difficult to tackle. The SDPA method becomes too expensive computationally for these problems because of poor node pruning. Even problems with up to 30 variables cannot be solved not because of memory overflow, but because of unreasonably long running times. Nevertheless, this approach constitutes a significant improvement.
over the previous results for problems \(z'Wz\). In the \(N = 50\) case, the optimum was 0.0056% lower than the best upper bound (available from Greedy-Cube), where in the \(N = 70\) case it was equal to the best upper bound (available from the PSD Randomization heuristic). The very small number of nodes in the 70 variables case is probably due to the very tight \(\text{[Lower Bound, Best Upper Bound]}\) interval as it was demonstrated by the corresponding \(\lambda\) parameter in Table 4.2. Indeed, we observed that in many cases even fixing only one variable to 1 was driving the lower bound higher than the best upper bound.
Chapter 5

Evaluation and Comparison of Approximate and Exact Algorithms

In this chapter we provide a comparison of the various algorithms considered in this thesis. The purpose is to give a clearer picture on the performance of each method with regard to the accuracy of its solution and its running time. To do this more effectively, we gather results from previous chapters in one Table. This way the trade-off between accuracy and running time becomes more evident. In Section 5.1 we consider lower bound relaxation algorithms, whereas in Section 5.2 we concentrate on feasible solution generating algorithms, namely, heuristic algorithms and branch and bound.

5.1 Relaxation Algorithms - Lower Bounds

Relaxation formulations attempt to solve the problem in a larger space than the original mixed integer-quadratic formulation and thus provide lower bounds for minimization problems. These bounds can prove of great value in exact Branch and Bound schemes when they are tight enough. Note, however, that the solution of a relaxation algorithm is not feasible in the sense that the binary variables $z$ will not
be strictly \( \{0,1\} \) upon termination.

We considered three different relaxation algorithms in Chapter 2. First was the relaxed version of the original problem. Then we had positive semidefinite relaxations, namely, the (linear) positive semidefinite problem solved by the interior point method, or the eigenvalue-cut algorithm and the quadratic positive semidefinite relaxation. Below we summarize what each one does.

1. **Relaxed Problem Formulation (RP)**

   This is simply the original problem (1.2) where one allows the \( \{0,1\} \)-integer variables to take any value between 0 and 1. This way the problem is transformed to a continuous variables problem with a quadratic objective and linear constraints. It can be solved fast using quadratic solvers, like CPLEX, but the lower bound estimate is expected to be quite weak in general.

2. **Linear Positive Semidefinite Relaxation (PSD)**

   Transforms the quadratic in the objective to a linear expression, by projecting the original space to a higher dimensional space of symmetric positive semidefinite matrices. It is solved by interior-point algorithms (SDPA), or the Eigenvalue-Cut (E-C) method.

3. **Quadratic Positive Semidefinite Relaxation (QPSD)**

   Strengthens the continuous variable \( x \) formulation by keeping the quadratic form in the \( x \) part of the objective. This improves the lower bound estimate since there is no approximation in the \( x \)-part of the problem. By introducing Lagrange multipliers for the mixed constraints in the problem, separability is achieved for the \( x \)- and \( z \)-parts. This results in a simple linearly constrained quadratic program for \( x \) and a much smaller-size positive semidefinite program for \( Z \).

In Tables 5.1 and 5.2 we present selected results for the aforementioned relaxation methods. In Table 5.1 we consider "\( x'Qx \)" problems and in Table 5.2 "\( z'Wz \)"
problems. Problems with up to 70 variables are considered. Solution accuracy and running time are placed together for easier comparison. By accuracy we refer to the ratio $LB/Optimum$ whenever the optimal solution is known, or $LB/UB$ whenever only an upper bound is available ($LB$ stands for the lower bound estimate and $UB$ for the best upper bound available). Note that for "$x'Qx$" problems with $N < 30$ and "$z'Wz$" problems with $N \leq 70$ the exact solution is known.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Accuracy</th>
<th>Time(sec)</th>
<th>RP</th>
<th>SDPA</th>
<th>E-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>40.03%</td>
<td>0.003</td>
<td>40.08%</td>
<td>0.09</td>
<td>37.98%</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>25.98%</td>
<td>0.030</td>
<td>25.98%</td>
<td>0.44</td>
<td>&lt;1%</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>*20.57%</td>
<td>0.060</td>
<td>*20.57%</td>
<td>1.26</td>
<td>1.196</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>*21.48%</td>
<td>0.160</td>
<td>*23.34%</td>
<td>7.33</td>
<td>2.938</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>*14.33%</td>
<td>0.290</td>
<td>*14.33%</td>
<td>25.56</td>
<td>n/a</td>
</tr>
</tbody>
</table>

"$x'Qx$" - FIXED VARIABLES (from Table 2.1)

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Accuracy</th>
<th>Time(sec)</th>
<th>RP</th>
<th>SDPA</th>
<th>E-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>50.28%</td>
<td>0.06</td>
<td>50.33%</td>
<td>0.9</td>
<td>n/a</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>31.81%</td>
<td>0.13</td>
<td>32.29%</td>
<td>2.2</td>
<td>n/a</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>*47.19%</td>
<td>0.35</td>
<td>*50.15%</td>
<td>7.6</td>
<td>n/a</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>*26.49%</td>
<td>0.84</td>
<td>*35.17%</td>
<td>17.0</td>
<td>n/a</td>
</tr>
</tbody>
</table>

Table 5.1: Results for solution accuracy and running time from relaxation algorithms in "$x'Qx$" problems. By solution accuracy we refer to the ratio $LB/Optimum$ whenever the optimal solution is known, or $LB/UB$ whenever only an upper bound is available ($LB$ stands for the lower bound estimate and $UB$ for the best upper bound available). In the second case, a star(*) is added before the entries above. Both non-fixed and fixed variable cases considered.

The message from Table 5.1 for the "$x'Qx$" problems is clear. No matter what relaxation formulation is used, the attainable lower bounds will be much smaller than the optimal solution or the best upper bound obtainable. One would rather use the
much less time consuming simple relaxed formulation of the problem and may be try SDPA in very large problems. Branch and Bound should rely on heuristics and $\epsilon$-approximate solutions. This certainly limits the applicability of the method for large problems.

<table>
<thead>
<tr>
<th>“$z'Wz$” - NO FIXED VARIABLES</th>
<th>RP</th>
<th>SDPA</th>
<th>QPSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>M</td>
<td>Accuracy</td>
<td>Time(sec)</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>37.72%</td>
<td>0.017</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>21.64%</td>
<td>0.060</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>19.92%</td>
<td>0.120</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>33.66%</td>
<td>0.450</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>52.38%</td>
<td>1.050</td>
</tr>
</tbody>
</table>

| “$z'Wz$” - FIXED VARIABLES (from Table 2.1) | | | | |
|-------------------------------|----|------|------|
| N    | M   | Accuracy | Time(sec) |     |     |
| 10   | 4   | 64.76%   | 0.09      | 122.29% | 48.7 |
| 20   | 5   | 47.62%   | 0.24      | 94.99% | 237.6 |
| 30   | 6   | 55.73%   | 0.72      | 99.30% | 1,418.0 |
| 50   | 8   | 60.10%   | 1.52      | 98.67% | 114.07% |

Table 5.2: Results for solution accuracy and running time from relaxation algorithms for “$z'Wz$” problems. By solution accuracy we refer to the ratio $LB/Optimum$ (LB denotes the Lower Bound). Note that in all problems above the exact solution is known from Branch and Bound.

For the less flat “$z'Wz$” problems it is clear that the Positive Semidefinite Relaxation solved by SDPA is preferable to the simple relaxed formulation. Despite its longer computational time the benefits can be tremendous in a Branch and Bound algorithm. For a little more amount of time than it takes Branch and Bound to solve a problem with 30 variables using the simple relaxed formulation, a problem with 70 variables can be solved when SDPA is used.
On the other hand, the Quadratic Positive Semidefinite method provides even stronger lower bounds than SDPA, but its running time is too high for the benefit it offers. Reducing its computational time would make it a very good candidate to substitute SDPA in Branch and Bound algorithms.

5.2 Feasible Solution Algorithms - Heuristics and Branch and Bound

In this section we compare algorithms that provide feasible solutions to the problem. These are heuristic techniques and the branch and bound method. Heuristics come up with an approximate solution after a certain time. Accuracy is defined here as the ratio \((HUB/optimum)-1\) (\(HUB\) stands for Heuristic Upper Bound), wherever an optimal solution is known and \((HUB/UB)-1\), when the optimal solution is not known (\(UB\) stands for the best available Upper Bound). Branch and Bound is an exact method (accuracy equals 0%), but can also provide \(\epsilon\)-approximate solutions when an exact solution is not possible to obtain. In that case the accuracy of the method will be \(\leq \epsilon\%\). We mention again, that for \(x'Qx\) problems with \(N < 30\) and \(z'Wz\) problems with \(N \leq 70\) the exact solution is known.

The following algorithms will be considered:

1. Simple Randomization Heuristic (SRH).
4. Greedy-Square (G2H).
5. Greedy-Cube (G3H).
7. Branch and Bound (BB).
In Tables 5.3 and 5.4 we present accuracy and running time data for the aforementioned algorithms. Again, \(x'Qx\) and \(z'Wz\) problems are considered separately. In the Positive Semidefinite Heuristic (PSDH) column we input the eigenvalue-cut method outcome for the \(x'Qx\) problems and the interior-point result for the \(z'Wz\) problems. In all randomization heuristics a number of \(5N\) samples was generated. For the Greedy-Square results we input the best outcome in terms of accuracy-time trade-off between the relaxed and fixed versions of the heuristic (see Section 3.3.2). The same holds for the Branch and Bound column, whenever both Relaxed and Positive Semidefinite results are available. Finally, note that the sign \(\sim\) in the Tables means "less than \(10^{-3}\)\%

Keep in mind once more, that the \(z'Wz\) problems present a bigger challenge to heuristic algorithms than the more flat \(x'Qx\) problems.

<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>SRH</th>
<th>M-RH</th>
<th>PSDH</th>
<th>G2H</th>
<th>G3H</th>
<th>SAH</th>
<th>BB</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>Acc.</td>
<td>.008%</td>
<td>.016%</td>
<td>.002%</td>
<td>.002%</td>
<td>.000%</td>
<td>.042%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T(sec)</td>
<td>0.8</td>
<td>0.9</td>
<td>8.0</td>
<td>3.3</td>
<td>13.0</td>
<td>14.0</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>Acc.</td>
<td>.010%</td>
<td>.011%</td>
<td>.001%</td>
<td>.014%</td>
<td>.000%</td>
<td>.009%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T(sec)</td>
<td>5.0</td>
<td>4.8</td>
<td>989.0</td>
<td>32.0</td>
<td>300.0</td>
<td>52.0</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>Acc.</td>
<td>*1.18%</td>
<td>*.128%</td>
<td>*.014%</td>
<td>*.109%</td>
<td>*.108%</td>
<td>*.118%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T(sec)</td>
<td>13.0</td>
<td>13.2</td>
<td>1,196.0</td>
<td>220.0</td>
<td>2,233.0</td>
<td>104</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>Acc.</td>
<td>*.003%</td>
<td>*.000%</td>
<td>*.017%</td>
<td>*.008%</td>
<td>*.004%</td>
<td>*.016%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T(sec)</td>
<td>55.0</td>
<td>79.6</td>
<td>2,942.0</td>
<td>374.0</td>
<td>13,387.0</td>
<td>357.0</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>Acc.</td>
<td>*\sim 0%</td>
<td>*.004%</td>
<td>n/a</td>
<td>*.001%</td>
<td>n/a</td>
<td>*.010%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T(sec)</td>
<td>152.3</td>
<td>151.0</td>
<td>n/a</td>
<td>4,848.0</td>
<td>n/a</td>
<td>863.0</td>
</tr>
</tbody>
</table>

Table 5.3: Results for solution accuracy and running time from heuristic algorithms and branch and bound for \(x'Qx\) problems. Accuracy is defined as the ratio \(HUB/optimum\)-1 (\(HUB\) stands for Heuristic Upper Bound), wherever an optimal solution is known and \(HUB/UB\)-1, when the optimal solution is not known (\(UB\) stands for the best available Upper Bound). In the second case, a star(*) is added in front of the entries above. The sign \(\sim\) means "less than \(10^{-3}\)\%".

In all problem cases in the Tables Branch and Bound seems to be too expensive computationally, especially in large problems. On the other hand, the heuristic algorithms work very well and provide tight upper bounds, sometimes even the optimal
<table>
<thead>
<tr>
<th>N</th>
<th>M</th>
<th>Acc.</th>
<th>SRH</th>
<th>M-RH</th>
<th>PSDH</th>
<th>G2H</th>
<th>G3H</th>
<th>SAH</th>
<th>BB</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>10.721%</td>
<td>.000%</td>
<td>36.07%</td>
<td>16.058%</td>
<td>16.058%</td>
<td>10.721%</td>
<td>.00%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T(sec)</td>
<td>1.2</td>
<td>1.3</td>
<td>1.4</td>
<td>5.7</td>
<td>21.0</td>
<td>20.0</td>
<td>3.4</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>1.620%</td>
<td>.052%</td>
<td>19.08%</td>
<td>.131%</td>
<td>.115%</td>
<td>.540%</td>
<td>.00%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T(sec)</td>
<td>8.0</td>
<td>7.7</td>
<td>8.0</td>
<td>64.0</td>
<td>607.0</td>
<td>75.0</td>
<td>1,159.0</td>
</tr>
<tr>
<td>30</td>
<td>6</td>
<td>.575%</td>
<td>2.015%</td>
<td>14.36%</td>
<td>.307%</td>
<td>.151%</td>
<td>1.546%</td>
<td>.00%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T(sec)</td>
<td>23.8</td>
<td>22.8</td>
<td>25.8</td>
<td>390.0</td>
<td>2,758.0</td>
<td>187.0</td>
<td>11,582.0</td>
</tr>
<tr>
<td>50</td>
<td>8</td>
<td>&gt; 100%</td>
<td>8.465%</td>
<td>137.22%</td>
<td>.289%</td>
<td>~ 0%</td>
<td>1.853%</td>
<td>.00%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T(sec)</td>
<td>157.4</td>
<td>146.0</td>
<td>143.4</td>
<td>3,024.0</td>
<td>21,243.0</td>
<td>882.0</td>
<td>55,264.0</td>
</tr>
<tr>
<td>70</td>
<td>10</td>
<td>&gt; 100%</td>
<td>&gt; 100%</td>
<td>.00%</td>
<td>3.347%</td>
<td>n/a</td>
<td>&gt; 100%</td>
<td>.00%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>T(sec)</td>
<td>561.3</td>
<td>557.6</td>
<td>484.5</td>
<td>8,949.0</td>
<td>n/a</td>
<td>3,259.0</td>
<td>159,623.0</td>
</tr>
</tbody>
</table>

Table 5.4: Results for solution accuracy and running time from heuristic algorithms and branch and bound for “$z'Wz$” problems. Accuracy is defined as the ratio $(HUB/optimum)$-1 ($HUB$ stands for Heuristic Upper Bound). Note that for all problems in this Table the optimal solution is known from Branch and Bound. The sign "~" means "less than $10^{-3}$%".

solution at much less time. This might downplay the importance of the Branch and Bound method. It is essential to keep in mind, that after it has been run, Branch and Bound guarantees that the optimal solution has been found. This is not so for the heuristics. In a general problem, there will be no way to know whether a heuristic estimate is satisfactorily close to the optimum, unless a relaxation algorithm or, even better, Branch and Bound has been run.

Concluding this section, we found that for flat “$x'Qx$” problems one would rather rely on heuristics and the simple relaxed formulation for the problem with Branch and Bound to estimate a solution. Branch and Bound is of limited use in larger problems though. Only problems with as many as 30 variables could be considered.

Regarding the less flat “$z'Wz$” problems, the Positive Semidefinite relaxation provides excellent lower bounds that along with heuristic estimates specify very tight ranges for the optimum. Branch and Bound can now be run for problems with up to 70 variables. Although heuristics provide very tight upper bounds (even the optimal solution in certain cases) in much less time, Branch and Bound is the only way to
guarantee that the optimum has indeed been calculated.

5.3 Summary

Three distinct relaxation formulations were considered. First, a simple relaxed version was introduced, where the \{0,1\} variables are allowed to take any value within the [0,1] interval. Then (linear) positive semidefinite relaxations that linearize the problem by projecting it onto a higher dimensional space for both $x$ and $z$ were formulated. These provide tighter lower bounds than the simple relaxed version, but require more computational time. Third, in an attempt to further strengthen the lower bounds, we formulated a quadratic positive semidefinite relaxation. Here the goal is to keep the quadratic part in the continuous variables and project only the binary part. Mixed constraints are introduced in the objective through Lagrange multipliers and the problem becomes a MaxMin program, which gives a lower bound at least as tight as the one from the linear positive semidefinite version. Finally we also considered the so called eigenvalue-cut method where the semidefiniteness constraint is removed and constraints are added to the problem explicitly to push the solution matrices towards the positive semidefinite cone. In Table 5.5 we provide a comparison of the above algorithms in terms of their ease of use, bound quality, maximum problem size they can consider, PSD heuristic performance and computational time required. We use a scale from 1 to 4 with 1 denoting the best performer.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Ease of use</th>
<th>Bound Q</th>
<th>Pr Size</th>
<th>PSDH</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Relaxed</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>n/a</td>
<td>1</td>
</tr>
<tr>
<td>(Linear) PSD</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Quadratic PSD</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Eig-Cut</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5.5: Comparison of relaxation algorithms of this thesis. A scale from 1 to 4 is used, where 1 means best performer.

For \("x'Qx"\) problems the improvement on the lower bounds from the positive semidefinite relaxations is not enough for pruning in branch and bound. This is not
true for \( x'Qx + z'Wz \) problems, where use of a positive semidefinite formulation results in considerable reduction in the enumeration tree and allows for solving larger problems.

Quite a few heuristic algorithms were developed with the purpose of providing strong upper bounds to the optimal solution. The first category included randomization heuristics, which are either stand alone (Simple Randomization and M-Randomization heuristics), or build upon information present in the positive semidefinite matrix solution of the corresponding positive semidefinite relaxation (PSD Randomization heuristic). These heuristics work well in general, but their performance is not guaranteed. In the second category we have the Greedy-Square and Greedy-Cube heuristics, which provided the most solid performance overall. These combine greedy search with variable branching. Finally, we considered simulated annealing, which is a local search method that randomly accepts neighbors based on their costs. Simulated annealing also performs well, but its performance depends on a few parameters and the starting sample. In Table 5.6 we rank the above heuristics in terms of ease of use, bound quality, problem sizes handled and computational time.

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Ease of use</th>
<th>Bound Q</th>
<th>Pr Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Random</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>M-Random</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PSD Random</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Greedy-Square</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Greedy-Cube</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Simul. Anneal.</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.6: Comparison of heuristic algorithms in this thesis. A scale from 1 to 5 is used, where 1 means best performer.

In many cases, the best upper bound found by the above heuristics coincided with the optimal solution. The randomization versions did not perform that well in problems with more than 70 variables. The most robust heuristics in terms of bound quality were the Greedy-Cube and Greedy-Square heuristics. The first provided the
exact optimal solution in two cases and the best upper bound in four out of the eight problems it was run for. We observed that in general the optimal solution lies closer to the upper rather than the lower bound estimate and this is true even for “$x'Qx + z'Wz$” problems where tight lower bounds exist.

Problem structure is very important when an exact branch and bound algorithm is considered. The more flat the objective, the more difficult it will be to cut nodes at an earlier stage in the tree. Both types of the problems we constructed are flat in the sense that there are many distinct solutions with costs very close to each other. In such circumstances, fixing variables does not alter the cost a lot (sometimes by less than $10^{-7}$). This means that one has to wait until a lot of variables have been fixed first in order to have the relaxed value close to the upper bound. This is the case in our “$x'Qx$” problems. Furthermore, in contrast with the Bienstock results [7], solving the relaxed version of the problem when no variables have been fixed gives a vector $x$ where its elements are very close to each other and all positive. This holds for the “$x'Qx + z'Wz$” case also, but the $x$ values now are a little further apart.

Using the simple relaxed formulation we can find the exact solution to problems “$x'Qx$” with up to 20 variables and problems “$x'Qx + z'Wz$” with up to 30 variables. Considering $\epsilon$-approximate solutions and the use of very strict upper bounds from the beginning results in a reduction in the number of nodes which is 9% in problems with $N \leq 20$. In larger problems node reduction can be higher though. The $N = 30$ “$x'Qx$” problem was solved that way as the number of nodes was reduced by at least 30%. The reason for the small node decrease lies in the poor lower bound quality of the simple relaxed version, especially in the more flat type of problems. Switching to the positive semidefinite relaxation improves lower bound quality in a drastic manner and this results in node reduction by even two orders of magnitude in the less flat problems! This kind of performance allows us to solve such problems with up to 70 variables, which is a serious improvement. As for the flat case, we are not able to go up to 30 variables as this comes at a large computational cost. The reason again lies in the fact, that although better, the lower bounds for these problems are still quite
weak to allow early pruning in the tree.

In a snapshot, we can say that we have been able to provide very tight ranges (less than 10%) for the optimal solution in problems with not very flat objectives and up to 150 variables. Furthermore, such problems have been solved exactly using branch and bound, when they involve up to 70 variables. Positive semidefinite relaxation and heuristics are employed to provide tight lower and upper bounds. In problems with flat objectives, we have provided good upper bounds in problems with up to 100 variables. We have been able to solve exactly or approximately using branch and bound only problems with up to 30 variables. This is due to poor lower bound quality, which is intimately related to the problem’s structure.
Chapter 6

Conclusions and Future Work

The goal of this work was to develop algorithms for solving linearly constrained mixed integer quadratic programs. These involve both continuous $x$ and binary $z$ variables which makes them considerably more difficult than purely integer quadratic problems. First we developed approximation algorithms aimed at providing tight lower and upper bounds to the optimal solution. Then we considered exact branch and bound schemes that make use of the bounds generated by such approximations to arrive at the optimal solution. We applied our methods to the problem of Index and Portfolio replication which arises in Finance. We constructed two different types of problems, one more flat, which was denoted as “$x'Qx$” and another steeper, denoted by “$x'Qx + z'Wz$”, or simply “$z'Wz$”. Very good approximation results were provided for problems with up to 150 variables and the exact solution could be found for problems with up to 70 variables.

Despite the positive developments towards solving mixed integer quadratic programs in this thesis, there are quite a few things that should be pursued for further improvement. We consider the areas of Relaxation Algorithms, Heuristics and Exact Methods separately below.

Relaxation Algorithms

A first step would be to incorporate some of the violated triangle inequalities into
the formulation. This can result in some improvement in the lower bound in the 
"x'Qx" case as well. Because the number of these inequalities can be quite large, an 
efficient way to add some of them is needed each time a problems is solved.

Some authors (see Helmberg [18]) seem to suggest, that in a \{-1,+1\} formulation 
of the problem, it is possible to determine the lower bound from the positive semidef-
inite relaxation from previous nodes in the tree without rerunning it. Although this 
has been proposed for purely integer quadratic programs, it would be interesting to 
explore possible application in the mixed case as well.

More attention should be given to the quadratic positive semidefinite algorithm. 
An interior-point method could probably be developed directly for that formulation. 
Keeping the quadratic objective for x and not incorporating the mixed constraints 
to the objective, one could probably come up with a nonlinear system of equations 
to determine the Newton step for x and Z. Developing an efficient algorithm could 
be quite challenging though.

One could also consider a piecewise-linear approximation of the objective and use 
solvers for mixed integer linear problems. That could provide an excellent estimate 
of the optimal solution especially in flat quadratic problems. Furthermore, the ap-
proximation can be from below or above indicating that tight bounds can possibly 
be obtained this way. Implementation, however, is expected to be costly.

Finally, it would be worth the effort to try and prove theoretical results on lower 
bound guarantees, that these relaxations could offer.

Heuristics

Despite the very good performance of our heuristics, a first step could be to 
improve the Greedy algorithms’ performance. For example, use the fixed Greedy-
Square version in the Greedy-Cube algorithm.

The synthetic “branch and greedy” approach can be applied to other heuristics 
as well. For example, consider a combination of variable branching and simulated 
annealing, or a randomization heuristic. It is almost certain that this would improve
performance and make those methods more robust. On the other hand, this would increase the computational time.

**Exact Methods**

Improvements in obtaining tight lower bounds will affect branch and bound in a positive way and allow for solving even larger problems. More experimentation with node and variable selection is needed. Incorporating a faster version of the quadratic positive semidefinite formulation and devising intelligent rules on when it is worth applying a relaxation and/or heuristic algorithm at a particular node could result in serious savings in terms of computational time.

A very interesting development would be in programming parallel implementations of branch and bound. Researchers have already diverted their attention into this area [7, 24]. This can potentially allow for solving much larger and more flat problems, than these considered in this thesis.

The area of solving mixed integer quadratic programs is still new and we believe it is going to attract a lot of attention in the coming years. This is to be attributed mainly to the wide applicability that such problems can find in many different areas and the special interest they present from a mathematical and computational point of view.
Bibliography


