Computational studies of collateral optimisation problems

PG Reynolds

orcid.org/0000-0002-4455-4254

Dissertation submitted in *partial* fulfilment of the requirements for the *Masters* degree in *Computer Science* at the North-West University

Supervisor: Prof SE Terblanche

Graduation May 2018

25807757

NWU®
NORTH-WEST UNIVERSITY
NOORDWES-UNIVERSITEIT
UNIBESITI YA BOKONE-BOPHIRIMA

Abstract

Collateral management is becoming an ever more complex area requiring sophisticated systems and technology. Collateral optimisation aims to optimise funding costs and balance sheet utilisation when allocating assets to meet a range of liabilities. The problem can be modelled as a mathematical optimisation problem, with the objective to minimise the cost of the posted collateral whilst meeting all collateral calls, and satisfying a diverse range of constraints. In practice, constraints such as lot sizes, concentration limits and eligibility criteria could result in a problem that becomes difficult to solve in a reasonable amount of time. This paper presents an integer linear programming formulation of the problem and investigates some of the computational aspects of collateral optimisation problems. Different types of problem instances are evaluated to establish their effect on the runtime performance of the solver. Various types of constraints are examined in order to explore the type of constraints that may make the problem difficult to solve. The results showed that with a basic set of constraints, the problem can be solved within a reasonable amount of time, even for relatively large problem sizes. However, certain types of constraints, such as those related to the diversification of assets, significantly affect the time taken to solve the model.

Key words: Collateral management, Collateral optimisation, Linear programming, Integer programming

Contents

List of Figures

List of Tables

Source Code Listing

Chapter 1

Introduction

Since the 2008 global financial crisis, financial institutions have implemented more stringent counterparty risk management processes. Regulation has increased the amount of collateral that has to be posted, the number of parties posting collateral and the frequency of posting. These factors are leading to an increasing demand for collateral and a scarcity of high quality liquid assets, making the optimal efficient use of collateral all the more important. The collateral management departments within these organisations are investing in complex information systems that attempt to make use of advanced algorithms to ensure that collateral is allocated optimally.

This dissertation focuses on the area of collateral optimisation, which can be modelled as a mathematical optimisation problem with the objective of minimising the cost of posted collateral, whilst meeting all collateral calls and satisfying a diverse range of requirements. Considerations such as lot sizes, concentration limits and eligibility criteria need to be incorporated into the model via appropriately formulated constraints.

The goal of this research is to formulate a mathematical model, implement and test the model, and perform an empirical evaluation of some of the computational aspects of collateral optimisation problems.

Different types of problem instances will be evaluated to establish their effect on the runtime performance of the solver. The various types of constraints will be examined in order to explore what type of constraints make the problem difficult to solve.

Many industry "white papers" have been published online, describing optimisation as part of the collateral management process. However, there appears to be limited existing academic literature dealing specifically with collateral optimisation problems. This research aims to show how the collateral optimisation problem is related to other known types of problems, and that existing techniques can be used to solve the problem.

Collateral management is a challenge because it is a relatively new imperative still evolving in line with legislation and industry standards. By presenting a formulation of the model, the intention is to provide insight into the types of data that are required as inputs into the model, and stimulate discussion around possible approaches to issues such as the identification of

instruments comprising the collateral inventory and the best way to represent funding costs.

1.1 Layout

Chapter 2 reviews the fundamental optimisation concepts that will be used in subsequent chapters to solve collateral optimisation problems. A general introduction to linear and integer programming is given. In its simplest form, a collateral allocation problem can be represented as a well known type of problem called the transportation problem. Methods for solving these types of problems are also reviewed.

Chapter 3 introduces the area of collateral and collateral management. Credit risk management and the use of collateral as a credit risk mitigation tool is discussed. The impact of regulation and the importance of technology in collateral management is highlighted.

Chapter 4 shows how the fundamental collateral allocation problem can be extended with a more realistic set of constraints. An integer programming formulation of the collateral optimisation problem is presented.

Chapter 5 discusses the data generation, model implementation and testing. The results of executing the model on various test data files are presented and the outcomes analysed.

Chapter 6 summarises the research presented in this dissertation and concludes with possible future work.

Appendix A contains an extract from the collateral documentation showing the selection of eligible collateral. The different types of financial transactions in which collateral plays an important role is also summarised.

Appendix B contains source code extracts of the algorithms that were implemented as part of this study.

Chapter 2

Mathematical optimisation

This chapter reviews the fundamental optimisation concepts that will be applied to solve collateral optimisation problems in subsequent chapters. The chapter gives a general introduction to mathematical optimisation and linear programming concepts. The simplex method will be reviewed using a graphical approach that, although not practical for large problems, assists in the understanding of optimal solutions to linear programming problems. Algebraic methods for the simplex method will also be discussed.

In its simplest form, a collateral allocation problem can be cast as a well known type of linear programming problem known as the transportation problem. The transportation problem and methods for constructing an initial basic feasible solution will be presented, as well as methods for progressing towards an optimal solution.

More complete formulations of the collateral optimisation problem have constraints that cannot be catered for by a basic transportation formulation. The section on integer programming demonstrates how some of these limitations can be overcome and an example of the branch and bound algorithm is provided.

Optimisation software and tools are discussed in the final section.

2.1 Linear programming

Mathematical models use mathematical relationships such as equations, inequalities and logical dependencies to describe the real-world relationships and constraints within a system. Mathematical programming is "programming" in that it relates to the "planning" or scheduling of activities in an optimal way. The main feature of mathematical programming models is that they involve optimisation. The quantity that is to be maximised or minimised is known as the objective function. Mathematical programming describes the minimisation or maximisation of an objective function of many variables, subject to constraints on those variables [1].

One of the most common and fundamental optimisation problems is the linear programming problem in which the objective function is linear and the constraints are linear equations and inequalities. The process of constructing

and solving this type of problem is known as linear programming [2]. Many real-world problems from a wide variety of disciplines can be formulated as linear programming problems and fast efficient methods exist for solving such problems even with thousands of variables and constraints [1].

Fourier was one of the first mathematicians to identify the relationship between linear equalities and optimisation. In 1827 he proposed a method for solving a system of linear inequalities [3]. Linear programming as a discipline started with the work of George B. Dantzig in 1947, motivated by the need to solve planning problems for the military [4]. The mathematical foundations were laid by von Neumann [3].

To illustrate how linear programming problems are formulated and solved, let us consider a simple example of a manufacturer that produces two types of products p_1 and p_2. Producing product p_1 requires 10 units of resource r_1 and 40 units of resource r_2 and generates 6 units of profit. Producing product p_2 requires 20 units of resource r_1 and 30 units of resource r_2 and generates 9 units of profit. Assuming there are 200 units of r_1 and 450 units of r_2 available, the manufacturer wants to determine how many units of p_1 and p_2 to produce to maximise profit. The resources r_1 and r_2 could be any resource such units of material or labour in hours.

The information in Table 2.1 can be formulated as a linear programming problem that determines how to optimally allocate the limited resources in order to maximise the profit.

	Products		
Resources	p_1	p_2	**Available Units**
r_1	10	20	200
r_2	40	30	450
Profit	6	9	

Table 2.1: Manufacturing example

Let x_1 and x_2 denote the units of p_1 and p_2 respectively. The resulting linear programming problem which maximises the profit is the following

maximise

$$Z = 6x_1 + 9x_2 \qquad \text{(objective function)}$$

subject to

$$10x_1 + 20x_2 \leq 200 \qquad (r_1 \text{ resource constraint})$$

$$40x_1 + 30x_2 \leq 450 \qquad (r_2 \text{ resource constraint})$$

The aim is to identify values for the *decision variables* x_1 and x_2 that satisfy

all the constraints and results in the greatest value for the objective function Z. A solution that satisfies all the constraints is known as a feasible solution. For example, the pair of values $(x_1, x_2) = (5, 5)$ is feasible, while $(10, 10)$ is infeasible as it violates both the constraints. The objective function value for the solution $(5, 5)$ is equal to 75 while the value for $(2, 2)$ is 30. Because we are trying to maximise the objective function, $(5, 5)$ is the better of the two feasible solutions. For a problem with n variables and m constraints, generating all possible feasible solutions is equivalent to finding all feasible combinations of m columns from n possibilities, given by $\binom{n}{m} = \frac{n!}{m!(n-m)!}$ [5]. As the number of variables and constraints grows larger, it becomes impractical to generate all possible feasible solutions and compare them to determine the best objective value. A more systematic method is required.

2.1.1 Graphical solution

The manufacturing example introduced in the previous section can be represented graphically by converting the inequalities to equalities and plotting the constraint lines. Each constraint makes a "cut" in the two-dimensional plane as in Figure 2.1. The feasible region is established by checking the validity of the constraint and determining if the feasible region lies above or below the line. The point $(0, 0)$ is valid, satisfying the constraint $10x_1 + 20x_2 \leq 200$, whereas the point $(10, 10)$ is invalid, violating the constraint and indicating that the feasible direction is below the line. The decision variables x_1 and x_2 are assumed to be non-negative otherwise it would be possible to allocate more resources than are available and thus the feasible region is also restricted to the positive quadrant of the Cartesian plane.

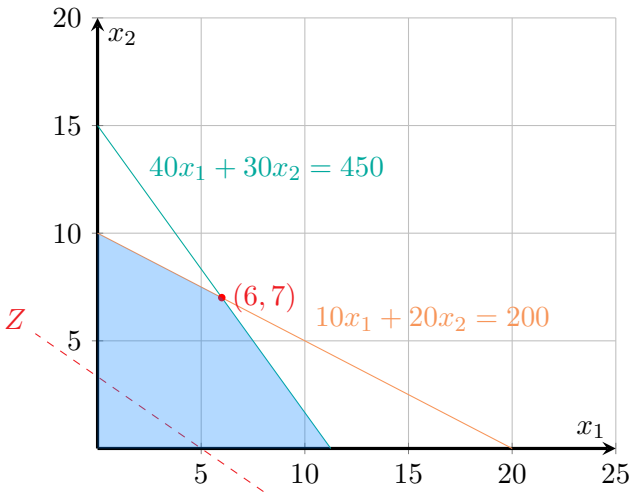


Figure 2.1: Feasible region

To determine which point in the feasible region is optimal, we consider the objective function and plot the line Z. Visually this line can be moved in parallel, i.e. increasing the value of Z, until it touches one of the corner points of the feasible set. This extreme point represents the optimal solution to the problem and lies at the intersection of the two constraint lines. By simultaneously solving the two equations we get the point $(6, 7)$ with $Z = 99$ and thus the maximum profit is obtained by producing 6 units of p_1 and 7 units of p_2.

In general, the optimal solution is always found at one of the extreme points of the feasible set, although there may be multiple optimal solutions. For problems with two and three decision variables, the problem can be represented and solved graphically using a multi-dimensional coordinate system. However, for larger linear programs with a greater number of variables and constraints it is not possible to solve graphically and a more general procedure for determining the extreme points is required.

2.1.2 Simplex method

Dantzig's invention of the simplex method solidified the effectiveness of linear programming in solving complex practical decision-making problems. Specific algorithms exist to solve special kinds of linear programming problems such as those described in Section 2.1.3, however the simplex algorithm

remains a primary computational tool in linear and mixed-integer programming to this day [6, 7].

Extreme points can be identified by simultaneously solving pairs of constraint equations to establish their intersections, and if the solution satisfies all the constraints then it can be selected as an extreme point solution. This approach of identifying all the extreme points of the feasible set and selecting the optimal solution forms the basis of the simplex algorithm. The algebraic procedure usually begins at the extreme point denoting no work or allocation of resources, and then repeatedly moves to the neighbouring extreme point that results in the biggest improvement in the solution, until no further improvements can be made.

The general linear programming problem can be written as follows

$$\begin{aligned} &\text{maximise} && Z = c_1x_1 + c_2x_2 + \ldots + c_nx_n && (2.1)\\ &\text{subject to} && a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 && \\ & && a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 && \\ & && \vdots \quad \vdots \quad \ddots \quad \vdots \quad \vdots && (2.2)\\ & && a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m && \\ & && x_j \geq 0, \text{ for } j = 1, 2, \ldots, n && (2.3) \end{aligned}$$

and in vector notation as

$$\begin{aligned} &\max cx && (2.4)\\ &\text{s.t } Ax \leq b && (2.5)\\ &x \geq 0 && (2.6) \end{aligned}$$

where $A \in \mathbb{R}^{m\times n}$ is a matrix, $b \in \mathbb{R}^m$ a column vector, $c \in \mathbb{R}^n$ a row vector and $x \in \mathbb{R}^n$ the column vector of decision variables.

Definition 2.1.1. A *basic solution* is a solution to (2.2) obtained by setting $(n - m)$ variables to zero and solving for the remaining m variables. The zero variables are called non-basic variables and the non-zero variables are called basic variables.

Definition 2.1.2. A *feasible solution* is a vector x that satisfies the constraints (2.2) and (2.3).

Definition 2.1.3. A *basic feasible solution* is a basic solution where all basic variables are non-negative.

For the simplex algorithm it is often convenient to write the linear program in the *standard* form by converting the linear inequalities to linear equalities. Minimisation problems can be converted to maximisation problems by multiplying the objective function by a negative constant. Simple transformations allow any linear program to be rewritten in the standard form

$$\begin{aligned} &\max\ cx \\ &\text{s.t } Ax = b \\ &\quad x \geq 0 \end{aligned}$$

The following example is used to demonstrate the application of the simplex method to the manufacturing example described previously. Inequalities are written as equalities by introducing *slack* variables s_1 and s_2.

$$\begin{aligned} \max \quad & Z = 6x_1 + 9x_2 + 0s_1 + 0s_2 \\ \text{s.t} \quad & 10x_1 + 20x_2 + s_1 + 0s_2 = 200 \\ & 40x_1 + 30x_2 + 0s_1 + s_2 = 450 \\ & x_1, x_2, s_1, s_2 \geq 0 \end{aligned}$$

The simplex method is an iterative algorithm that moves from one extreme point to an adjacent extreme point. Since extreme points of the feasible set correspond to basic feasible solutions [2], each iteration starts with a basic feasible solution that is not yet optimal. Then Gaussian elimination is used to generate a new basic feasible solution, which is tested for optimality to establish the need for further iterations. The simplex iterations are best described in a tabular form know as a *simplex tableau* containing the coefficients of the objective and constraint equations.

The first step of the simplex method is to find an initial basic feasible solution. Since there are two constraints with four unknowns it is not possible to solve the equations algebraically. Each basic feasible solution will have *nonbasic* variables that are given a value of zero in order to solve for the *basic* variables. Note, that depending on the problem data, a solution with slack variables equal to zero, may not constitute a basic feasible solution. In this case, alternative approaches should be used to generate basic feasible solutions.

The set of basic variables in the basic solution is known as the *basis*. Choosing x_1 and x_2 as nonbasic results in $s_1 = 200$ and $s_2 = 450$, and as the values

are non-negative the solution $(0, 0, 200, 450)$ is feasible and serves as the initial basic feasible solution. The initial simplex tableau corresponding to this basic feasible solution is provided in Table 2.2. The C_i column contains the coefficients of the basic variables representing the contribution per unit in the cost function. The slack variables can be thought of as surplus or unused resources that don't add any value to the objective function thus their coefficients are zero in the objective function. Thus the value of Z for the initial basic feasible solution is zero. The column C_j contains the coefficients of the variables in the objective function. The bottom two rows are used to determine if the solution can be improved or not, where $Zj = \sum_{i=1}^{m}(C_i)(a_{ij})$.

C_i	Basis	x_1	x_2 (E)	s_1	s_2	Solution	Ratio
0	s_1 (L)	10	20*	1	0	200	$200/20 = 10$
0	s_2	40	30	0	1	450	$450/30 = 15$
C_j		6	9	0	0		
Z_j		0	0	0	0		
$C_j - Z_j$		6	9	0	0		

Table 2.2: Initial simplex tableau

Once the initial tableau is formed, a variable to enter the basis is selected by evaluating the last row and identifying the largest positive contribution. This is known as the pivot column and x_2 becomes the *entering variable* (E). If all the values in the last row are non-positive, then there can be no further improvement and the current solution is optimal.

To establish which variable will leave the basis, the ratio of the right-hand-side value in the row and the coefficient in the pivot column is calculated. Only positive values are considered as non-positive values would impose no bound on the entering variable. Since $\min\{200/20, 450/30\} = 10$, s_1 is selected as the leaving variable (L) in the pivot row.

To construct the new tableau, the pivot element (*) in the intersection of the pivot row and column is used in row operations on the matrix. The entries in the pivot row are divided by the pivot element in order to get 1 in the position of the pivot element.

The formula for calculating the value of the remaining rows is

$$\text{New value} = \text{Old value} - \frac{\text{Corresponding Key Col Value} \times \text{Corresponding Key Row Value}}{\text{Key Element}}$$

For example:

$$40 - (30 \times 10/20) = 25$$
$$30 - (30 \times 20/20) = 0$$
$$0 - (30 \times 1/20) = -3/2$$
$$1 - (30 \times 0/20) = 1$$
$$450 - (30 \times 200/20) = 150$$

C_i	Basis	x_1 (E)	x_2	s_1	s_2	Solution	Ratio
9	x_2	1/2	1	1/20	0	10	20
0	s_2 (L)	25*	0	−3/2	1	150	6
C_j		6	9	0	0		
Z_j		9/2	9	9/20	0		
$C_j - Z_j$		3/2	−9/20	0	0		

Table 2.3: Second simplex tableau

From Table 2.3, x_1 is defined as the entering variable and s_2 becomes the leaving variable. By performing another pivot operation, the third simplex tableau is obtained in Table 2.4. As all the values of $C_j - Z_j$ are non-positive the optimal solution has been reached with $x_1 = 6$, $x_2 = 7$ and $Z = 99$.

C_i	Basis	x_1	x_2	s_1	s_2	Solution	Ratio
9	x_2	0	1	8/100	−1/50	7	
6	x_1	1	0	−3/50	1/25	6	
C_j		6	9	0	0		
Z_j		6	9	−36/100	−3/50		
$C_j - Z_j$		0	0	−36/100	−3/50		

Table 2.4: Third simplex tableau

Revised simplex algorithm

Instead of maintaining a tableau, an alternative matrix-oriented approach offers greater computational efficiency by performing a series of linear algebra computations [8, 9, 10]. If we consider the system of equations $Ax = b$ in n unknowns with m the rank of the matrix A and $(m < n)$, there exist m linearly independent column vectors. Selecting these vectors to form an $m \times m$ basis matrix B and leaving the remaining $n - m$ columns as non-basis N, A can be rearranged as $A \equiv [B, N]$. Similarly we can partition

the variable vector as $\begin{bmatrix} x_B \\ x_N \end{bmatrix}$ and the equality constraints can be rewritten as

$[B, N] \begin{bmatrix} x_B \\ x_N \end{bmatrix} = Bx_B + Nx_N = b$

$$x_B = B^{-1}b - B^{-1}Nx_N \tag{2.7}$$

With $x_N = 0$ and $x_B = B^{-1}b$ we have a basic solution and if $x_B \geq 0$ we have a basic feasible solution. x_B are the basic variables and x_N are the nonbasic variables [2].

Similarly, for the objective function $Z = cx$ we have $c = [c_B, c_N]$, thus

$$Z - [c_B, c_N] \begin{bmatrix} x_B \\ x_N \end{bmatrix} = 0 \tag{2.8}$$

$$Z - c_B x_B - c_N x_N = 0 \tag{2.9}$$

Substituting x_B from (2.7)

$$Z - c_B(B^{-1}b - B^{-1}Nx_N) - c_N x_N = 0 \tag{2.10}$$

$$Z - (c_N - c_B B^{-1}N)x_N = c_B B^{-1}b \tag{2.11}$$

The *reduced costs* vector $c_N - c_B B^{-1}N$ in (2.11) is used to establish which non-basic variable will enter the basis and if for a basic feasible solution $c_N - c_B B^{-1}N \leq 0$ then the solution is optimal [2]. For the purpose of demonstrating the application of the matrix-oriented approach, one iteration is performed on the manufacturing example. The first iteration begins by selecting an initial basis $x_B = \begin{bmatrix} x_3 \\ x_4 \end{bmatrix}$ and thus $x_N = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$,

$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, N = \begin{bmatrix} 10 & 20 \\ 40 & 30 \end{bmatrix}$, and $x_B = \begin{bmatrix} 200 \\ 450 \end{bmatrix}$.

The variable to enter the basis is the one for which the reduced costs vector is a maximum

$c_N - c_B B^{-1}N = [6 \quad 9] - [0 \quad 0] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 10 & 20 \\ 40 & 30 \end{bmatrix}$ thus x_2 will enter the basis

The variable to leave the basis is the one for which $\frac{B^{-1}b}{B^{-1}Nx_N}$ is a minimum, where only positive values of Nx_N are considered

$$\frac{x_B}{B^{-1}Nx_N} = \frac{\begin{bmatrix} 200 \\ 450 \end{bmatrix}}{\begin{bmatrix} 20 \\ 30 \end{bmatrix}} = \begin{bmatrix} 10 \\ 15 \end{bmatrix}$$ thus x_3 will leave the basis.

Source Code B.1 gives an example of the simplex algorithm implemented in Python using the NumPy scientific computing package for matrix operations.

2.1.3 Transportation problem

One of the oldest known types of linear programming problems is that of the transportation problem which involves the transporting of goods or materials from a set of origins to a set of destinations with the objective of minimising the transportation costs. A model of the problem takes into account constraints such as the amounts available at the origins and the amounts required at the destinations. Transportation models can be applied to a diverse range of applications and can be extended to deal with the allocation or assignment of any kind of resource, provided that the quantities can be measured and a movement cost per unit established.

As early as 1781 the French mathematician Gaspard Monge formulated a problem for the relocation of materials in the most efficient way, specifically dealing with the transportation of soil during the construction of forts and roads with minimal transportation costs [11, 12]. Schrijver [13, 14] reviews an article by A.N. Tolstoĭ from 1930, in which graphical and algorithmic methods are proposed to solve a cargo transportation problem along the railway network of the Soviet Union.

In 1939 the Soviet mathematician Leonid Kantorovich described the importance of certain linear programming problems for organising and planning production, and gave a simplex-like method for solving the transportation problem [3]. The history of the Monge–Kantorovich transportation problem is documented in [12].

The standard form of the problem was formulated by Hitchcock in 1941 [15], who also gave a computational procedure for solving the problem. Independently, Koopmans and his colleagues arrived at the same problem as part of their work during World War II and thus the problem is often referred to as the Hitchcock–Koopmans transportation problem [16, 17, 18].

The application of the simplex method to the transportation problem was given by Dantzig in 1951 [19] and this method has been widely used to solve systems involving hundreds of constraints with thousands of unknowns [18].

Let us consider an example where a supplier is required to transport a number of units of a product from m warehouses to n factories. Each origin has a given level of supply and each destination has a known level of demand. The capacities at each origin and the requirements at each destination as well as the transportation costs between each source and destination can be represented graphically as a network with m source nodes and n sink nodes where each pair of nodes is connected by a directed arc as in Figure 2.2.

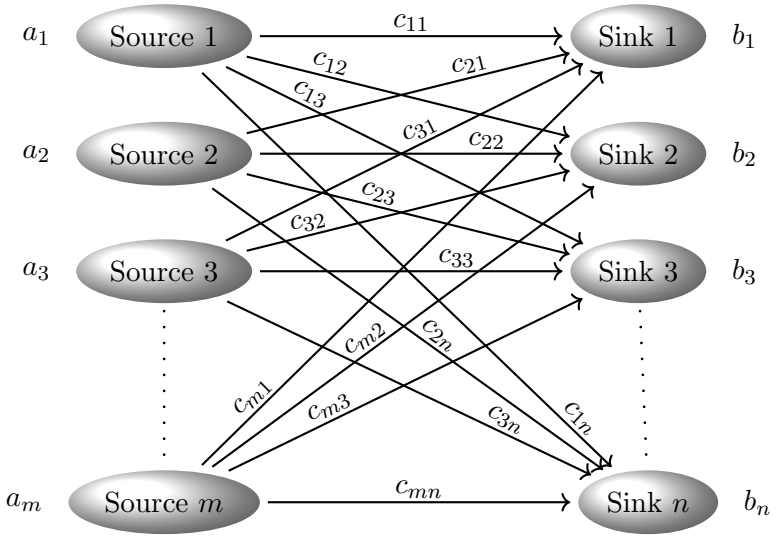


Figure 2.2: Balanced transportation flow network

A linear programming formulation is given by

$$\min \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$$

s.t.

$$\sum_{j=1}^{n} x_{ij} \leq a_i \quad \text{for } i = 1, 2, \ldots, m$$

$$\sum_{i=1}^{m} x_{ij} \geq b_j \quad \text{for } j = 1, 2, \ldots, n$$

$$x_{ij} \geq 0 \quad \text{for } i = 1, 2, \ldots, m \text{ and } j = 1, 2, \ldots, n$$

where
m is the number of sources,
n is the number of destinations,
a_i is the supply capacity of the i$^{\text{th}}$ source,
b_j is the demand requirement of j$^{\text{th}}$ destination,
c_{ij} is the cost per unit of transporting goods between the i$^{\text{th}}$ source and j$^{\text{th}}$ destination,
x_{ij} is the number of units of a product to be transported between the i$^{\text{th}}$ source and j$^{\text{th}}$ destination.

The simplex algorithm can be applied to solve the problem in the above form by introducing slack and surplus variables to convert the inequality

constraints to equalities as was done previously. However, by noting that for a given problem to have a feasible solution, the total supply cannot be less than the total demand and if the total supply equals the total demand, then any feasible solution satisfies the inequality constraints as equalities. Hence whenever the total supply equals the total demand the introduction of slack and surplus variables is no longer required. More formally, when $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} bj$, the transportation problem is said to be balanced and the standard form of the problem is given by

$$\begin{aligned}
\min \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{s.t.} \quad & \\
& \sum_{j=1}^{n} x_{ij} = a_i \quad \text{for } i = 1, 2, \ldots, m \\
& \sum_{i=1}^{m} x_{ij} = b_j \quad \text{for } j = 1, 2, \ldots, n \\
& x_{ij} \geq 0 \quad \text{for } i = 1, 2, \ldots, m \text{ and } j = 1, 2, \ldots, n
\end{aligned}$$

The original problem is defined by $m + n$ equality constraints and $m \times n$ variables. Balanced transportation problems, however, have the special characteristic that they have $m + n - 1$ independent constraint equations due to the fact that the sum of the supply and demand equations are equal. Having the problem in the standard form allows for the application of a simplified version of the simplex method.

It is always possible to balance an unbalanced transportation problem. Where there is excess supply, the problem can be balanced by artificially introducing a dummy sink with zero cost to absorb the difference. If there is a cost for storing surplus production, then the unit costs can be set equal to the inventory holding cost. Similarly, where the total supply is less than the total demand, a dummy source can be added with zero cost. If there is a penalty cost for the unsatisfied demand, the unit cost is set equal to the shortage penalty cost, thus accounting for the fact that units are artificially transported from the dummy source.

For a transportation problem with m sources and n sinks, the transportation tableau contains m rows representing the sources and their corresponding supply and n columns representing the sinks and their corresponding demand. Each cell (i, j) contains the decision variable x_{ij} indicating the number of units of the product that should be transported, as well as the transportation cost per unit c_{ij} in the upper right hand corner of the cell as in Table 2.5.

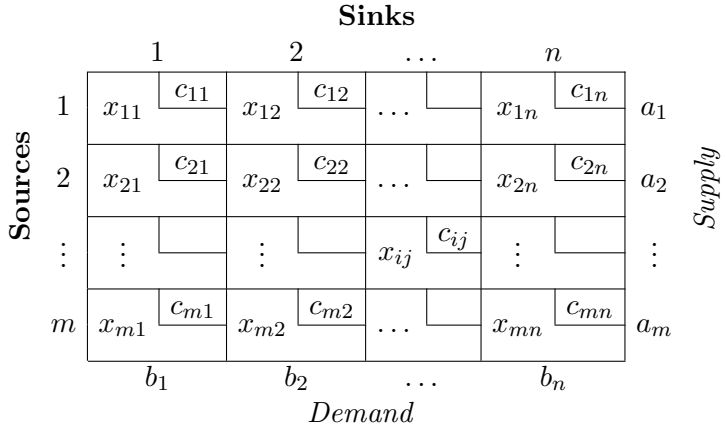


Table 2.5: Transportation tableau

Initial basic feasible solution

As with the simplex method, the first step of the transportation algorithm is to find an initial basic feasible solution. Consider an example of a balanced transportation problem with three sources and three destinations, and costs as depicted in Table 2.6. Three different methods for establishing an initial basic feasible solution are described below.

Northwest corner method As its name implies, this method begins in the northwest corner of the tableau and assigns as many units as possible to meet the demand given the available supply. The first iteration (Table 2.6), begins at cell $(1, 1)$ by allocating all available supply from row 1 to column 1, resulting in a remaining demand of 10. As the first row's supply has been exhausted, the row is ignored in future iterations. The available supply and remaining demand are updated in the relevant row and column respectively.

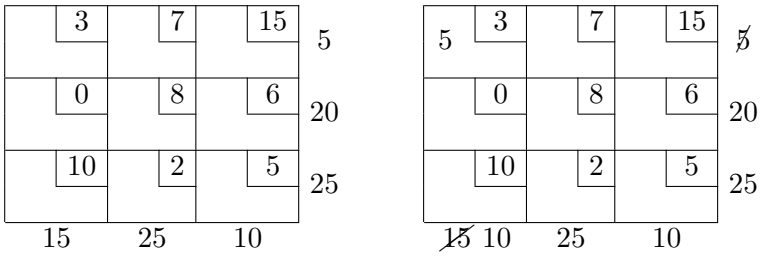


Table 2.6: Northwest corner method - initial iterations

In the next iterations (Table 2.7), cell $(2, 1)$ becomes the entering cell and

10 units are taken from the second row. The demand in column 1 is satisfied and the supply in the second row is reduced to 10. Then cell $(2, 2)$ is chosen as the next cell, assigning the remaining 10 supply from row 2, leaving a demand of 15 in column 2.

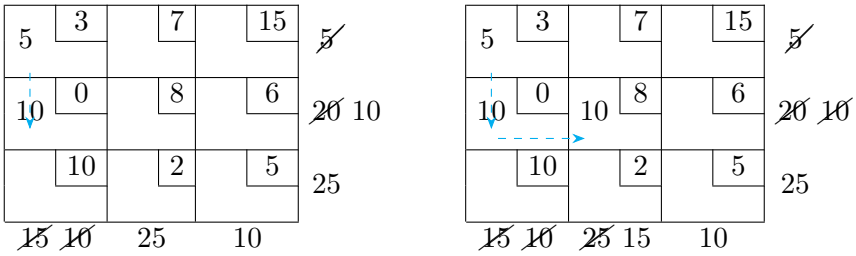


Table 2.7: Northwest corner method - intermediate iterations

In the final iterations (Table 2.8), 15 is assigned to cell $(3, 2)$ leaving 10 to be assigned to cell $(3, 3)$ in the last iteration. The fact that the final remaining supply and demand are equal is as a result of the balanced nature of the problem.

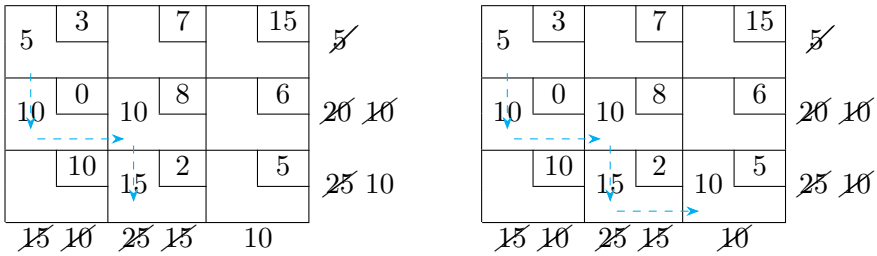


Table 2.8: Northwest corner method - final iterations

Five allocations have been made, resulting in $x_{11} = 5, x_{21} = 10, x_{22} = 10, x_{32} = 15, x_{33} = 10$, and it is evident that the basic variable count is equal to $m+n-1$. The remaining allocations are equal to zero and are thus non-basic. The objective function value is $(5 \times 3) + (10 \times 0) + (10 \times 8) + (15 \times 2) + (10 \times 5) = 175$.

The northwest corner method is a simple systematic procedure, that performs a minimal number of calculations. It does not, however, produce a very good initial feasible solution. The method described in the next section takes the transportation costs into account when selecting subsequent cells.

Least cost method This method selects the entering cell with smallest c_{ij} value. The algorithm begins in cell $(2,1)$, as it is the cell with the least cost of all cells in the tableau (Table 2.9). The demand in column one will be fully satisfied and the supply in row 2 reduced to 5. Cell $(3,2)$ will be the next entering cell, assigning the full amount from the available supply and excluding row 3 from future iterations.

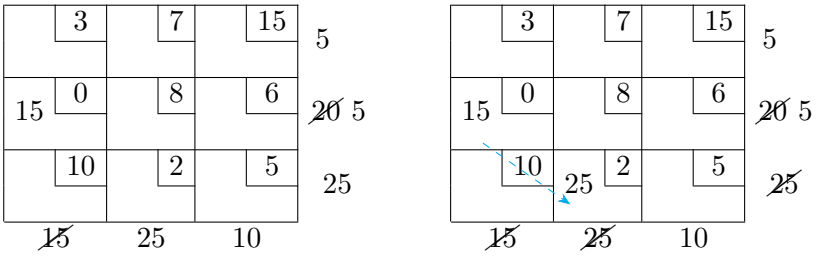


Table 2.9: Least cost method - initial iterations

Cell $(2,3)$ is the next cell in which 5 units are allocated, exhausting row 2's supply (Table 2.10). Finally cell $(1,3)$ is allocated the remaining 5 units, thus completing all allocations.

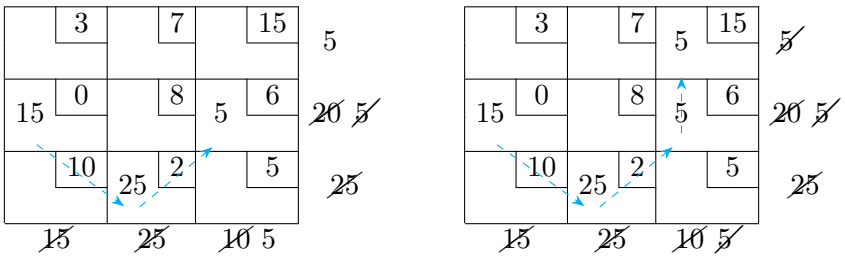


Table 2.10: Least cost method - final iterations

The final allocations are $x_{21} = 15, x_{32} = 25, x_{23} = 5, x_{13} = 5$ resulting in an objective function value of 155, which is better than the previous method. However, there are only 4 allocations as opposed to the 5 arrived at by the previous method. In a standard transportation problem, the test for optimality of any feasible solution requires that there are $m+n-1$ allocations and that these allocations are at independent cell locations, meaning that it should not be possible to increase or decrease an allocation without violating the row and column restrictions. If a solution has less than $m + n - 1$ independent allocations, the solution is said to be degenerate. Degeneracy can occur at the initial solution or during the testing of the optimal solution. In order to resolve degeneracy, an infinitesimally small positive amount ϵ is

allocated to one or more unoccupied cells with the lowest transportation costs until the $m + n - 1$ condition is satisfied [20].

Vogel approximation method Vogel's method calculates a set of penalty costs for each row and column and then selects the entering cell based on these penalties [21]. The penalty is calculated as the difference between the second lowest cost and the lowest cost in each row and column. The cell with the greatest penalty cost and smallest cost value c_{ij} is selected as the entering cell, with ties broken arbitrarily. The first row's penalty is $7-3=4$ and the first column's penalty is $3-0=3$, and similarly for the other rows and columns as indicated in Table 2.11. The greatest penalty is 6 in the second row, with cell $(2,1)$ having the least cost of zero. This cell is selected as the entering cell and 15 units are allocated to satisfy the demand, removing this column from consideration in future iterations and leaving a supply of 5 in row 2.

					Penalty
	3	7	15	5	4
15	0	8	6	~~20~~ 5	6
	10	2	5	25	3
	~~15~~ 0	25	10		
Penalty	3	5	1		

Table 2.11: Vogel approximation method - initial iteration

The next iteration (Table 2.12), calculates new penalties excluding the first column, hence the first row penalty is now $15 - 7 = 8$ and the second row penalty is $8 - 6 = 2$. The cell $(1, 2)$ with penalty 8 and cost 7 is selected as the next entering cell, allocating all 5 of the supply, reducing the demand to 20 and excluding the first row from future calculations. Recalculating the penalties results in cell $(3, 2)$ becoming the entering cell with penalty 6 and cost 2, with an allocation of 20.

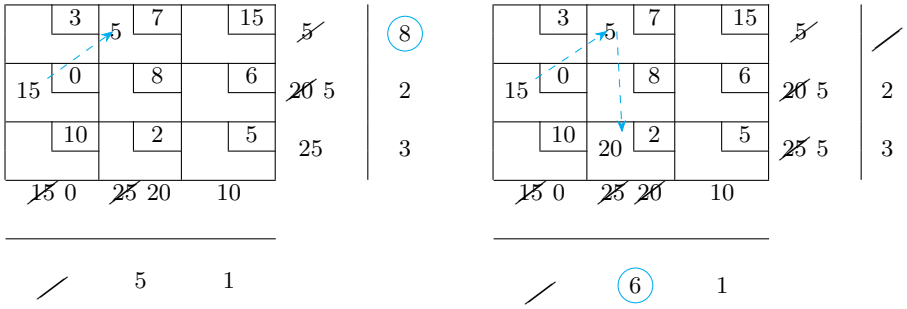


Table 2.12: Vogel approximation method - intermediate iterations

In the final two iterations, only two cells remain and no more penalties can be computed (Table 2.13). The cell $(3, 3)$ with least cost 5 is selected first, allocating the remaining supply of 5 from row 3 and then cell $(2, 3)$ is allocated the remaining 5 from row 2, hence satisfying the demand in column 3.

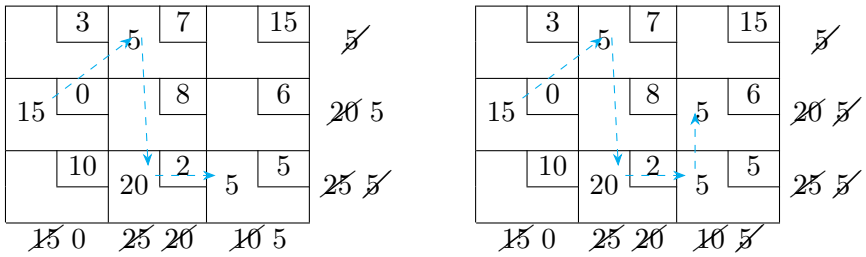


Table 2.13: Vogel approximation method - final iterations

The final allocations are $x_{21} = 15, x_{12} = 5, x_{32} = 20, x_{33} = 5$ and $x_{23} = 5$, resulting in an objective function value of $(15 \times 0) + (5 \times 7) + (20 \times 2) + (5 \times 5) + (5 \times 6) = 130$, which is better than the previous two methods.

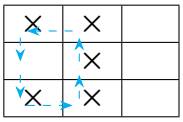
Test for optimality

Given an initial basic feasible solution, the next step is to determine an optimal solution. This section describes two methods that use an iterative process to progress towards optimality.

Stepping stone method Charnes and Cooper's stepping stone method is used to determine whether an initial basic feasible solution is optimal or not, and if not, to iterate towards an optimal solution [22]. The method involves evaluating the unallocated cells in the transportation tableau to establish if a change in allocation would result in a more optimal solution. The method's name is derived from the analogy of crossing a pond using stepping stones, where the allocated cells are the stones and the unoccupied cells are water.

Before the optimality test can be applied, it is first necessary to check that the initial feasible solution is not degenerate and satisfies the two conditions described in Section 2.1.3. Graphically, in terms of the transportation tableau, an allocated cell is considered independent if it is not possible to travel from the cell back to itself via a series of horizontal and vertical steps from one occupied cell to another without directly reversing the route (see Table 2.14).

(a) non-independent



(b) independent

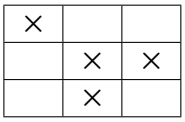


Table 2.14: Independent allocation paths

The stepping stone algorithm begins by evaluating all nonbasic cells (water cells) to determine if an allocation in that cell would reduce the overall allocation costs. An increase in one of the nonbasic cells will necessitate a decrease in one of the basic cells (stone cells) in order to maintain the feasibility of the solution and not violate the demand or supply constraints.

Given the basic feasible solution produced by the Vogel approximation method in Section 2.1.3, a closed path is formed from the selected water cell, via stone cells, back to the original cell as indicated in Table 2.15 for cell (2,2). The closed path is formed with horizontal and vertical lines and may skip stone or water cells, but right angles are only made at stone cells. It should be noted that every nonbasic cell has exactly one such stepping stone path. Alternating positive and negative signs indicate the addition and subtraction of units in order to maintain the feasibility. The reduced cost change of cell (2,2) is calculated as $8 - 2 + 5 - 6 = 5$, which has a positive sign, indicating an increase in cost if this change were to be made. Similarly (1,3) and (3,1) have positive reduced cost values of $(15 - 7 + 2 - 5 = 5)$ and $(10 - 5 + 6 - 0 = 11)$ respectively and it is not desirable to bring either of them into the basis. Cell (1,1) however has a more complicated closed path but results in a reduced cost of $3 - 0 + 6 - 5 + 2 - 7 = -1$, indicating that the solution is not optimal and that x_{11} should enter the basis. If more than

one cell has a negative reduced cost, then the one with the smallest negative cost is selected. If two cells have the same smallest negative number, the cell where the maximum allocation can be made is chosen. Finally, the smallest allocated amount is subtracted from each cell with a negative sign and added to the cells with a positive sign, resulting in a new tableau (Table 2.16) with an objective function value of $(5\times3)+(10\times0)+(25\times2)+(10\times6) = 125$. This process is repeated until all the reduced costs are positive and no further improvements can be made resulting in an optimal solution.

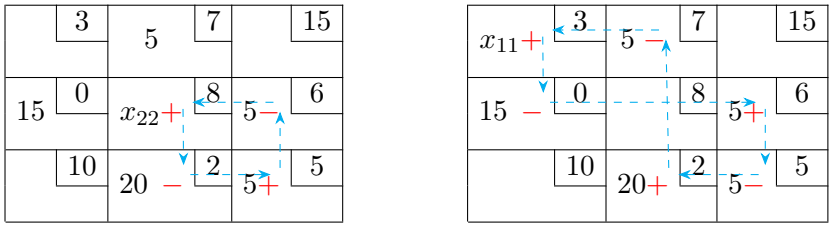


Table 2.15: Stepping stone method - closed paths

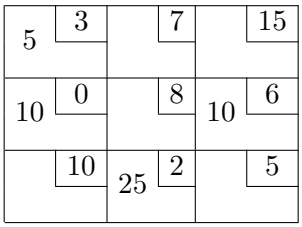


Table 2.16: Stepping stone method - optimal solution

Modified distribution method In the stepping stone method, a closed path is constructed for each unoccupied cell. In the modified distribution method, the nonbasic cells are evaluated simultaneously and only the path of the most negative cell is calculated [23]. Thus the modified distribution method requires computationally less effort that grows linearly with respect to the problem size as opposed to the exponential growth of the stepping stone method. See [24] for a review of the computational aspects of the various methods of solving transportation problems.

As with the previous method, before proceeding with the optimality test,

the first step is to assert that the two conditions in Section 2.1.3 are satisfied.

The next step is to assign the variables u_i to each of the rows and v_j to each of the columns, and to calculate the values of the variables for each of the allocated cells such that $u_i + v_j = c_{ij}$ as in Table 2.17. As there are $m+n$ variables and $m+n-1$ allocations, u_1 is set to zero, to solve for the remaining variables. These values are then used to calculate the reduced costs.

	v_1	v_2	v_3
u_1	3	5 7	15
u_2	15 0	8	5 6
u_3	10	20 2	5 5

Stone cell	$\boldsymbol{u_i + v_j = c_{ij}}$	$\boldsymbol{u_i, v_j}$
$(1,2)$	$u_1 + v_2 = 7$	$u_1 = 0, v_2 = 7$
$(2,1)$	$u_2 + v_1 = 0$	$u_2 = -4, v_1 = 4$
$(2,3)$	$u_2 + v_3 = 6$	$u_2 = -4, v_3 = 10$
$(3,2)$	$u_3 + v_2 = 2$	$u_3 = -5, v_2 = 7$
$(3,3)$	$u_3 + v_3 = 5$	$u_3 = -5, v_3 = 10$

Table 2.17: Modified distribution method - initial steps

The reduced costs $\bar{c}_{ij} = c_{ij} - (u_i + v_j)$ are equivalent to those produced by the stepping stone method in the previous section. The cell with the smallest reduced cost is selected to enter the basis and the closed path for that cell generated. The smallest allocated amount on the path is subtracted from cells with a negative sign and added to cells with a positive sign resulting in a new tableau, and the process is repeated. The resulting tableau in Table 2.18 is in fact the optimal solution.

	$v_1 = 4$	$v_2 = 7$	$v_3 = 10$
$u_1 = 0$	+5 3	5−5 7	15
$u_2 = -4$	15−5 0	8	5+5 6
$u_3 = -5$	10	20+5 2	5−5 5

Water cell	$\boldsymbol{c_{ij} - (u_i + v_j)}$	$\boldsymbol{\bar{c}}$
$(1,1)$	$3 - (0 + 4)$	-1
$(1,3)$	$15 - (0 + 10)$	5
$(2,2)$	$8 - (-4 + 7)$	5
$(3,1)$	$10 - (-5 + 4)$	11

Table 2.18: Modified distribution method - reduced costs

An implementation of these algorithms in Python is given in Source Code B.2. A linear programming implementation of the problem in OPL was used to validate the results, and demonstrates the succinct and mathematically natural way the problem can be expressed in such a language (Source Code B.3).

2.1.4 Limitations

Although a wide variety of practical problems can be expressed using linear programming, there are cases where the linearity assumption does not hold. This section describes some of the limitations of linear programming where alternative approaches are required to accurately model the problem domain.

Nonlinearity In cases where the objective function or constraints are non-linear, the resulting nonlinear programming problems are often more difficult to solve than similarly sized linear programming problems. Such cases arise where economies of scale come into play, for example where profit margins increase due to decreasing unit costs as a result of increases in production. The distinction between convex and non-convex problems plays an important role in determining which methods are used for solving such problems. With convex problems an optimum is guaranteed to be a global optimum, however for non-convex problems the possibility of finding a local optimum arises when using certain algorithms. For a discussion of convex optimisation methods as well as quadratic programming techniques for solving problems with quadratic objective functions see [25, 2, 26].

Uncertainty In cases where the coefficients of decision variables or constraints are not known with certainty, these probabilistic variables can be modelled using a technique called *stochastic programming*. This is not an alternative class of problem, but rather a technique for using random variables with known probability distributions to model the uncertain data before the stochastic program is converted into the deterministic equivalent linear or integer program. In finance such problems are common, for example modelling risk and return on investments.

Integrality Often the assumption that variables can be allowed to take fractional values also breaks down. For example, there are cases where it only makes sense to allocate integral quantities of resources, or produce to integral quantities of goods such as vehicles. Integer programming can be used to enforce integer variables, but also adds modelling capabilities such as the ability to model logical conditions using binary variables. Integer programming is discussed in more detail in the following section.

2.2 Integer programming

Many linear programming problems, require the decision variables to have non-fractional values, that represent units that cannot be divided. Integer

linear programming problems are ones in which some, or all of the decision variables are restricted to be integers. The problem is called a *pure integer program* when all the variables are restricted to be integers, and a *mixed integer program* when some of the variables are constrained to be integers while others are allowed to be non-integers [27].

2.2.1 Binary variables

Variables that can only take the values of 0 or 1 are a special type of integer variables known as binary variables. Binary variables can be used to capture yes/no type decisions such as whether a particular fixed cost project should be undertaken or whether a factory should be constructed at a particular location. Such a binary decision variable can be modelled as

$$y_j = \begin{cases} 1 & \text{if decision } j \text{ is yes} \\ 0 & \text{otherwise} \end{cases}$$

Since the values of y_j is 1 only where the condition is true, this variable can also then be used to restrict the size or cardinality of a certain set as follows

$$\sum_{j=1}^{n} y_j \leq maxSize$$

2.2.2 Logical constraints

Binary variables are also useful for modelling logical constraints that cannot be expressed with linear constraints alone.

Either-or constraints Given two constraints (2.12) and (2.13), in order to ensure that at least one of the constraints is satisfied, the additional constraints (2.14), (2.15) and (2.16) are added.

$$f_1(x_1, x_2, \ldots, x_n) \leq 0 \tag{2.12}$$
$$f_2(x_1, x_2, \ldots, xn) \leq 0 \tag{2.13}$$

$$f_1(x_1, x_2, \ldots, x_n) \leq M_1 y_1 \tag{2.14}$$
$$f_2(x_1, x_2, \ldots, xn) \leq M_2 y_2 \tag{2.15}$$
$$y_1 + y_2 \leq 1 \tag{2.16}$$
$$y_1, y_2 \in \{0, 1\}$$

Where M_1 and M_2 are constants chosen large enough so that when some $y_j = 1$ the constraints are satisfied for all values of $x_1, x_2, \ldots, x_n$. The constraint (2.16) ensures that at least one y_j variable is equal to 0, and thus at least one of the original constraints is satisfied. This constraint can be replaced with $y_1 + y_2 = 1$, since the constraint implies either y_1 or y_2 equals 0. Choosing the largest M, a simplified formulation is given by

$$f_1(x_1, x_2, \ldots, x_n) \leq My$$
$$f_2(x_1, x_2, \ldots, xn) \leq M(1 - y)$$
$$y \in \{0, 1\}$$

If-then constraints In order to ensure that
$f(x_1, x_2, \ldots, x_n) > 0 \implies g(x_1, x_2, \ldots, x_n) \geq 0$
the following constraints are included in the formulation

$$-g_1(x_1, x_2, \ldots, x_n) \leq My \tag{2.17}$$
$$f(x_1, x_2, \ldots, xn) \leq M(1 - y) \tag{2.18}$$
$$y \in \{0, 1\}$$

Where M is large enough so that $-g < M$ and $f \leq M$ are satisfied for all values of $x_1, x_2, \ldots, x_n$.

If $f > 0$ is satisfied, then (2.18) can only be satisfied if $y = 0$ and thus (2.17) implies $-g \leq 0$ and $g \geq 0$.

If $f > 0$ is not satisfied, then (2.18) allows $y = 0$ or $y = 1$ and $g < 0$ or $g \geq 0$ can only be satisfied if $y = 0$ and thus (2.17) implies $-g \leq 0$ or $g \geq 0$.

2.2.3 Branch and bound

A number of techniques have been developed for solving integer programs [28, 29]. The cutting-plane method was proposed by Ralph Gomory in 1958

[2]. This method removes undesirable fractional solutions by tightening the formulation. Constraints are added to the linear relaxation in order to cut off non-integer solutions. This is done during the solution process and does not create additional sub-problems as is the case with branching. The branch-and-bound algorithm was developed by A. Land and G. Doig in 1960 [2]. This method divides (branches) the problem into a number of sub-problems that can be solved using linear programming. These sub-problems are then evaluated and compared in a process called bounding.

Initially, in practice, the branch-and-bound algorithm was almost always considered to outperform the cutting-plane algorithm. However, in the mid-1990s Gérard Cornuéjols and colleagues showed that combining these techniques into a branch-and-cut procedure was very effective [2]. Cutting plane techniques are generally accepted to be one of the most important contributions in the computational advancement of integer programming.

The branch-and-bound procedure begins by removing all the integrality constraints from the mixed integer program (MIP). This results in a linear program called the *linear relaxation* of the original MIP which can be solved using the methods described in Section 2.1.2. The next step is to pick a variable in the relaxation with a fractional value that should be restricted to integer. For example, if the value in the relaxation is 3.7, this value can be excluded by adding the constraints $x \leq 3$ and $x \geq 4$. In this way, the original problem (P_0) can be split into two sub-problems, P_1 with $x \leq 3$ and P_2 with $x \geq 4$, which are solved and the better of the two solutions is selected. This process is then repeated on each of the sub-problems, resulting in a tree structure with P_0 as the root node, and the sub-problems as leaf nodes as in Figure 2.3.

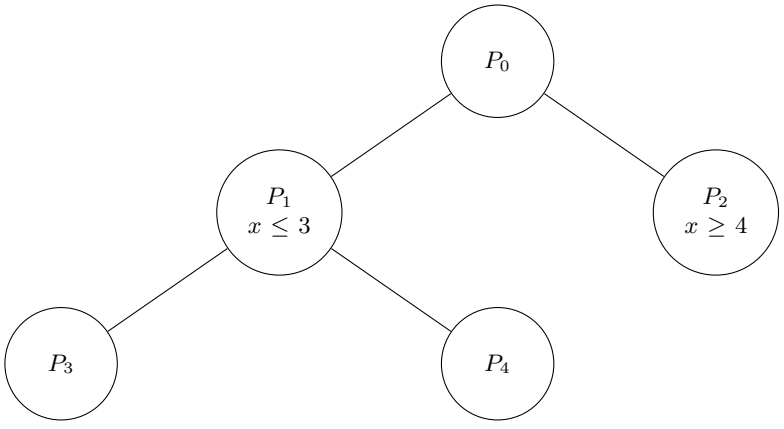


Figure 2.3: Branch and bound

As the search procedure progresses, the best feasible solution found so far, is

termed the *incumbent solution* and z^* denotes the value of z for the current incumbent. A node is said to be *pruned* and thus not explored any further, for three different reasons

- Pruned by *bounds*, when the node's objective value is worse than the incumbent.
- Pruned by *infeasibility*, when the linear program at that node is infeasible.
- Pruned by *integrality*, when the linear program at that node has an optimal solution that is integral.

Assuming a maximisation problem, the objective value for the incumbent is a lower bound on the optimal solution of the original MIP. Note however that, the maximum of the optimal objective values of all of the current leaf nodes also gives a valid upper bound. The difference between the current upper and lower bounds is known as the *gap* and when the gap is zero, optimality has been reached. When the gap is smaller than a certain threshold, the solution can be considered "good enough" for all practical purposes and can be very useful in reducing further unnecessary computations as seen in Section 5.

To illustrate the branch-and-bound algorithm consider the following integer program

$$\begin{aligned} \max \quad & 3x_1+4x_2 \\ \text{s.t.} \quad & x_1 - x_2 \leq 2 \\ & 3x_1 + 9x_2 \leq 23 \\ & x_1, x_2 \geq 0 \\ & x_1, x_2 \text{ integer} \end{aligned}$$

The root node of the search tree in Figure 2.4 contains the linear programming relaxation with $x_1 = 3.4167, x_2 = 1.4167$ and the objective value 15.9166. Branching on $x_2 \leq 1$ and $x_2 \geq 2$ results in the child nodes P_1 and P_2 respectively. Node P_1 has an integer solution so it is pruned and becomes the incumbent. Node P_2 is branched on the variable x_1 with node P_4 pruned due to infeasibility. Branching node P_3 on variable x_2 results in an infeasible solution for node P_6. Node P_5 has an integer solution and is pruned, however this solution is not better than the incumbent. As there are no further nodes to explore, the optimal solution is at node P_1 with $x_1 = 3, x_2 = 1$ and objective value $z = 13$.

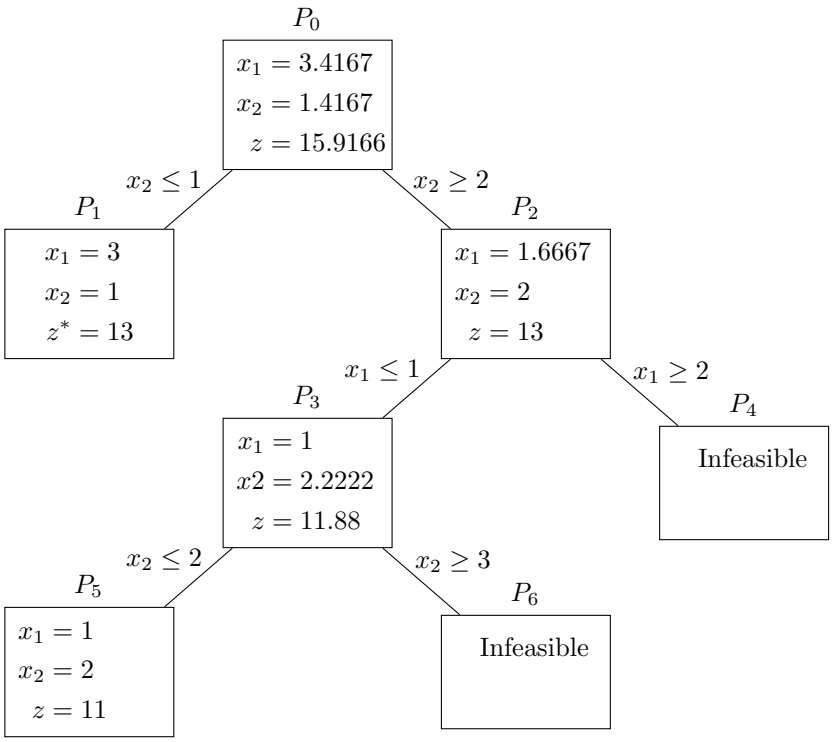


Figure 2.4: Branch and bound tree

The branch-and-bound algorithm can be extended to mixed integer problems, by only branching on variables that are required to be integer. For binary variables, a problem is split into two sub-problems by fixing a variable (say x_1) to $x_1 = 0$ for the one sub-problem and $x_1 = 1$ for the other sub-problem.

One of the main limitations of the branch-and-bound algorithm, is related to decisions such as the order in which nodes are examined, and the branching strategy. For example, at node P_0, x_1 could have also been selected as the branching variable instead of x_2. These decisions can significantly impact the performance of the algorithm in the worst case scenario, that might well occur in practice. Branching strategies and node selection criteria as well as the use of heuristics to find good starting feasible solutions are described in [2] and [29].

Finally, it is worth noting that *presolving* or preprocessing techniques are an important component of all modern MIP solvers. The aim of presolving is to simplify the given problem instance, before solving with more sophisticated and often time-consuming procedures such as the branch-and-bound and cutting-plane algorithms [30]. Presolve routines remove redundant information, and integer presolve methods such as bound tightening and coefficient

tightening, strengthen the model formulation, accelerating the subsequent solution process [31]. For example, given a binary variable x and the constraint $x \leq 1/2$. Then x can be fixed at zero since it can never take the value of 1.

2.3 Optimisation software

A variety of commercial and open source software packages are available that implement the models presented in this chapter. These products make use of algebraic modelling languages that have a syntax very similar to the mathematical notation for sets, indices and algebraic expressions. A user of the software is able to formulate the problem using the appropriate syntax, and the software takes care of invoking and configuring the underlying solver and related algorithms.

These software packages are the result of decades of research and development, and therefore offer the most computationally advanced implementations of an extensive range of algorithms. However, custom implementations in programming languages like Java and Python, such as the examples in Appendix B, can be used to solve certain fundamental forms of problems.

In modelling languages like IBM's OPL [32], SAS's OPTMODEL [33] and AMPL [1], relationships defined within the model can be independent of the data used with the model and thus the same model can be used with different data sets. In general, these tools make mathematical programming more efficient and reliable when developing, debugging and documenting models.

See Dantzig's notes on the derivation of certain terms and the contribution of William Orchard-Hays in the early development of these software tools [4]. Bixby [6] describes the progress in algorithm development, as modern computer software and hardware has evolved. Sophisticated open source solvers for linear and integer programming are available as part of the Computational Infrastructure for Operations Research (COIN-OR) project [34, 35].

Chapter 3

Collateral management

This chapter introduces the role of credit in financial markets and discusses the importance of credit risk management. The nature of counterparty credit risk and the use of collateral as a form of credit risk mitigation is discussed. The importance of collateral management as well as the benefits and risks of collateralisation are highlighted. The regulatory environment and the role of technology in collateral management is also explored.

As documentation plays such an intricate role in the collateral management process, key terminology useful in interpreting the relevant legal agreements is outlined. Chapter 4 will translate certain attributes of the collateral agreement into the constraints that uniquely characterise the collateral optimisation problem.

As the required amount of the call requirement is one of the main inputs into the collateral optimisation problem, an example of the required amount calculation is described.

3.1 Credit and credit risk management

Financial institutions such as banks, insurers and pension funds play an important role in the correct functioning of financial markets.
In the last few decades there has been tremendous growth in the development of sophisticated products offered by these financial institutions. Due to the complexity of these products, the risks are not always fully understood and can have a devastating impact on the financial system. In the wake of the 2008 financial crisis, driven by significant regulatory changes, there has been an increased awareness of the importance of credit risk management [36].

One particular type of risk, namely counterparty credit risk, arises due to a counterparty's inability to fulfil their contractual obligations. It is especially important because of its systemic nature and its complexity as it intersects both market risk[1] and credit risk[2]. Counterparty credit risk generally arises

[1]Market risk is the risk that arises from changes in the market price of financial instruments.

[2]Credit risk is the risk that money owed is not repaid and arises from a borrower's unwillingness or inability to make required payments.

from trading in the over the counter (OTC) derivatives markets, securities borrowing and lending (SBL) and repurchase (Repo) markets [37]. A high level explanation of the different types of transactions is given in Appendix A. There are various ways to mitigate counterparty credit risk, such as netting, margining and hedging. These concepts are discussed in more detail in Section 3.2.4. The following section focuses on collateralisation and the margin call process.

3.2 Collateral and collateral management

In general, collateral refers to assets that are used to secure a lending transaction that are forfeited in the event of default. Collateral serves as a form of guarantee that if the borrower is unable to pay the lender, the lender has the right to sell the collateral to recover the outstanding amounts owed by the borrower.

In the financial markets, trading between two counterparties creates counterparty credit exposure, which is a measure of the magnitude of loss or replacement cost in the event the counterparty defaults. Collateral is moved between trading counterparties as security against non-payment or default by a counterparty. Collateralisation serves as a form of bilateral insurance that is used to mitigate counterparty credit risk.

As an example, consider an interest rate swap transaction between two counterparties A and B, in which party A makes a mark-to-market (MTM)[3] profit and party B makes a corresponding loss. Then party B is required to post collateral to party A in order to offset the credit exposure that arises. The type of collateral posted can be cash or securities, the specific characteristics of which are agreed contractually before the swap transaction is entered into. As the collateral agreements are often bilateral, an institution with a positive MTM will call for collateral and be required to post collateral in the case of a negative MTM. Similarly, if the exposure decreases, an equivalent amount of collateral must be returned accordingly.

Collateral management is the function within the financial institution that is responsible for reducing counterparty risk and making efficient use of collateral. Collateral management began in the 1980s with Bankers Trust and Salomon Brothers using collateral to offset credit exposures [37]. Initially there were no standard legal agreements, but during the 1990s collateralisation of derivatives became more prevalent, resulting in significant documentation being produced by ISDA[4]. In 1994, the Credit Support Annex

[3]Mark-to-market or marking to market is an accounting method that records the value of an asset or liability according to the current market price or "fair value".

[4]International Swaps and Derivatives Association.

(CSA) was finalised, specifically documenting credit support (collateral) for derivative transactions (Appendix A.1). The collateral management team is responsible for day to day collateral operations such as collateral calculations, making and receiving margin calls, dispute resolution, delivering and receiving collateral, interpreting collateral agreements and liaising with internal departments and external counterparties. Collateral and collateral management, collateral optimisation and the convergence of collateral and liquidity are reviewed in [38]. Trends and developments in the collateral management space are considered in [39].

3.2.1 Benefits and risks

The main motivation for collateral management is to reduce counterparty risk and minimise losses in the event of default. The reduction of credit exposure allows the institution to do more business, and trading between counterparties with different credit worthiness is facilitated. The Basel II accord provides capital relief for collateralised exposures, thus capital requirements are reduced, freeing up capital for alternate investments. Pricing is also made more competitive by reducing the credit spread that is charged to a counterparty.

Although basic counterparty credit risk is reduced, it is important to note that collateralisation changes counterparty risk into other forms of financial risk. Collateralisation is operationally intensive and operational risks and costs in terms of human and technology resources is increased. Legal risks with regard to the ability to enforce contracts or differences in insolvency legislation across jurisdictions also needs to be taken into consideration. Collateral optimisation, re-use and rehypothecation and the transformation of collateral in the Dutch financial sector is studied in [40]. The paper highlights increases in operational and liquidity risks due to complex collateral allocation processes and systems, particularly optimisation models and algorithms.

3.2.2 Regulatory drivers

At the G20[5] Pittsburgh summit in 2009 it was noted that OTC derivatives had contributed significantly to the 2008 financial crisis due to certain market characteristics that exacerbate systemic risk. As part of their commitment to stabilise and safeguard the financial system, member countries

[5]The Group of Twenty (G20) is an international forum for the governments and central bank governors from the world's twenty leading industrialised and emerging economies.

agreed to gradually implement reforms of the global OTC derivatives markets. The main goals are to reduce systemic counterparty risk, protect regulated entities, improve fairness, efficiency and competitiveness and increase the overall transparency of the market.

Regulations were put in place to encourage trading of all OTC derivatives on an exchange or other electronic trading platforms. Standardised OTC derivatives contracts are to be cleared through a central counterparty (CCP) and reported to a trade repository. Bilateral derivatives contracts have generally involved exchanging variation margin over the life of the deal. Centrally cleared trades, however, will require a daily variation margin in cash as well as require both counterparties to pay an initial margin in cash or sovereign bonds with a high credit quality. Non-centrally cleared contracts are subject to higher capital requirements and bilateral derivatives that are too complex to be centrally cleared, such as inflation swaps, will need to be collateralised. The first of these regulations have been incorporated in the U.S. under the Dodd–Frank Wall Street Reform and Consumer Protection Act, and in Europe under the European Market Infrastructure Regulation (EMIR).

As a member of the G20, South Africa committed to making regulatory and legislative reforms to the OTC derivatives market in alignment with international standards. In July 2016, National Treasury published the third draft of regulations supporting the objectives of the Financial Markets Act, and noted in an explanatory memorandum that it expected the implementation of the reforms to continue beyond 2018 [41].

The Basel Committee on Banking Supervision (BCBS) and the International Organization of Securities Commissions (IOSCO) created the Working Group on Margining Requirements (WGMR), and in September 2013 the final policy framework for the margining of non-centrally cleared OTC derivatives was published. The framework was developed in consultation with the Committee on Payment and Settlement Systems (CPSS) and the Committee on the Global Financial System (CGFS) and imposes restrictions on eligible forms of collateral, segregation of initial margin and documentation requirements that govern the collateral relationships.

Basel III[6] is an extension to the existing Basel II framework, aimed at strengthening the regulation, supervision and risk management of the banking sector. In addition to strengthening the capital requirements and introducing a minimum leverage ratio, two new liquidity ratios were introduced. The Liquidity Coverage Ratio requires banks to have sufficient high-quality

[6]The Basel Accords are three sets of recommendations (Basel I, II and III) on banking regulations issued by the Basel Committee on Bank Supervision (BCBS). The Basel Committee is named after the city of Basel in Switzerland where the BCBS maintains its secretariat at the Bank for International Settlements (BIS).

liquid assets on their balance sheets to withstand a 30-day long stress scenario, and the Net Stable Funding Ratio incentivises banks to access longer terms stable funding sources.

In South Africa the Pensions Fund Act regulates the extent to which retirement funds may invest in particular assets. Regulation 28 of the act prescribes conditions for securities lending transactions and in particular that adequate collateral is held at all times. Rules concerning the margining and valuation of collateral and pledge or cession and outright transfer of collateral assets are stipulated. The Solvency Assessment Management project (a joint venture between the Financial Services Board and the South African insurance industry) will adopt many of the principles of the European Solvency II directive, which aims to coordinate insurance regulation and the amount of capital insurance companies should hold in order to reduce the risk of insolvency. With more complex collateral requirements, insurance companies will need to adopt more advanced collateral management solutions.

The challenges faced by financial institutions due to a changing regulatory environment and the impact on collateral and risk management are explored in [42]. Counterparty credit risk, the impact of regulation, collateral funding aspects and the effect of systemic risk when trading with central counterparties is reviewed in [43]. A study [44] of the collateral value chain shows how banks could reduce their Basel III capital requirements and optimise balance sheet usage.

3.2.3 Role of technology

Collateral management is a large and complex area requiring sophisticated systems and integration across many of an organisation's operational areas. An enterprise wide collateral management system includes inventory management and collateral tracking, reference data and reporting, advanced analytics, automation and straight-through processing.

Technology is at the core of collateral optimisation. Advanced algorithms, rule and scenario engines and data visualisation provide decision support to end users. Even though these systems can perform complex calculations and automate many manual processes, collateral allocation may still involve subjective decisions. As each organisation might want to apply slightly different business rules in how they manage their collateral, an ideal system should provide a user with the ability to add or change constraints without requiring them to be specialists in the underlying modelling language.

Electronic messaging standards such as SWIFT[7], used for money and secu-

[7]SWIFT (Society for Worldwide Interbank Financial Telecommunication) messages are

rities transfers, play a crucial role in automation and straight-through processing. The Financial Information eXchange (FIX) protocol is a messaging standard for the real-time electronic exchange of information related to securities transactions. FpML (Financial products Markup Language) is the messaging standard for OTC derivatives and is based on XML[8]. Similarly, XBRL (eXtensible Business Reporting Language) is an XML based standard to define and exchange business information such as financial reports. Collateral management in particular benefits from reducing operational risk through automation. As opposed to sending information via email, phone or fax, the margin call notices and requests for substitutions can be automated allowing, for scalability and enhanced security. SWIFT's bilateral collateral management messages are based on ISO 20022[9] standards.

Cloud computing[10] services provide an alternative for firms that may not have a collateral management platform in place. Regulation is changing rapidly and cloud solutions allow an organisation to manage technology costs and scale technology infrastructure in a more agile way. Complex new functionality can be deployed quickly and updated more easily in order to keep up with new and continually changing regulatory requirements. Collateral management requires interoperability between organisations, and cloud based solutions seem particularly suited to provide such functionality. The benefits of Software as a Service (SaaS) solutions for collateral management are mentioned in [45].

In the future, blockchain[11] technology is likely to play a greater role in the collateral management space. The movement of assets can be transparently recorded, and the smart contract capabilities of the technology, can be used for the calculation of collateral amounts and automatic triggering of margin calls.

3.2.4 Documentation

Prior to the commencement of trading activity between two counterparties, an ISDA Master Agreement is signed by the two parties. The document

used by financial institutions to send and receive information about financial transactions.

[8]XML (Extensible Markup Language) is used to encode data in a format that is both human and machine readable.

[9]ISO 20022 is an ISO (International Organization for Standardization) standard for electronic data interchange between financial institutions.

[10]Cloud computing is the access to shared computing resources such as storage, databases, networks, applications and services that can be rapidly provisioned, configured and managed over the Internet.

[11]A blockchain is a list of records (blocks) which are linked and cryptographically secured. A blockchain can act as a public distributed digital ledger that can record transactions between two parties.

outlines the standard terms that apply to all OTC derivatives transactions between the parties and is typically accompanied by a CSA. Repo transactions are most often documented using the Global Master Repurchase Agreement (GMRA), whereas the Global Master Securities Lending Agreement (GMSLA) relates to collateral for the securities lending and borrowing market.

Key terminology useful in interpreting the CSA documentation is listed below. For a more detailed explanation of the parameters in the collateral agreement and the mechanics of collateralisation see [46] and [47].

Netting is the process of aggregating all trades with a counterparty to establish a net MTM portfolio value and exposure estimate.

A *margin call* is a request usually from the party with a net positive MTM to the party with a net negative MTM to post additional collateral in order to offset the credit risk due to changes in the market value of the transactions.

The counterparty posting the collateral is the collateral *giver* and the party receiving the collateral is the collateral *taker*. The *call amount* is the amount of collateral being requested by the taker and is rounded up or down to a certain lot size.

The *independent amount* is an amount that is paid upfront usually in the form of cash. It can also be an agreed amount that is transferred later and serves as a form of overcollateralisation intended to offset credit risk due to a time delay between a collateral call and the delivery of collateral, or in cases where a counterparty's credit rating deteriorates.

The *threshold* is the level of exposure below which collateral will not be called for, and essentially represents the amount of uncollateralised exposure a party is willing to accept.

The *minimum transfer amount* is the smallest amount that can be transferred as collateral and assists in avoiding the frequent transfer of small amounts of collateral, which might be more costly than the benefits provided by collateralisation.

The *valuation percentage* or haircut is a discount applied to the MTM value of the collateral that reduces its value in order to protect the holder of the collateral from a deterioration in the value of the held collateral. For example, if a security attracts a haircut of 5%, only 95% of the value of this security will be credited for collateral purposes. Appendix A contains sample pages from a CSA illustrating eligible collateral and associated haircuts.

The types of collateral that a party is willing to accept are termed *eligible* collateral (A.2). ISDA has developed Collateral Asset Definitions in order to provide standardised short-hand terms of the various types of collateral.

A party may be unwilling or unable to accept certain types of collateral, for example, it may not have the accounts to hold cash or securities in a certain currency. Counterparties may also not accept certain collateral due to credit reasons stemming from concentration limits or percentage limits on the amount of a particular issue.

A counterparty has the ability to *substitute* one form of collateral for another under the terms of the agreement. For example, a counterparty may require certain securities posted as collateral to be returned, so that they can be used to meet other commitments. They can then issue a substitution request and post an amount of alternative collateral as long as the collateral meets the eligibility requirements under the agreement.

Non-cash collateral may also be sold or delivered as collateral to other counterparties. This secondary trading of collateral is known as *rehypothecation*. The terms rehypothecation and reuse of securities are often used interchangeably, but there is an important technical and legal difference. Outside the United States, the term *reuse* applies if collateral is posted on the basis of title transfer when legal ownership goes from the collateral giver to the collateral taker. The collateral becomes the unencumbered property of the taker and can be used as he/she deems fit. Repo and securities lending transactions are based on the transfer of the legal title of the collateral [40, 48].

The term rehypothecation is used when collateral is pledged, i.e. the collateral remains legally owned by the collateral giver. Rehypothecation is common in derivatives transactions, which are usually governed by an ISDA Master Agreement and associated CSA, which specifies whether rehypothecation of collateral is allowed under the terms of the contract. If a counterparty makes a substitution request and the original securities have been rehypothecated, then the party may need to purchase equivalent fungible[12] securities in order to return the requested collateral [40, 48, 49].

Required amount calculation

Trades are periodically marked to market and net exposure values are calculated. Based on the terms of the collateral agreements, a collateral calculation determines if a party has the right to call for collateral or if a party is obliged to post collateral.

[12]The *fungibility* of securities implies that different securities (not the same security with exactly the same serial numbers), but equivalent in terms of specification can be returned to the lender.

The collateral amount is calculated as follows [47]

$$max(MTM - TCP, 0) - max(-MTM - TP, 0) - C \tag{3.1}$$

where MTM is the current mark-to-market value of the positions, TP is the threshold of the party and TCP is the threshold of the counterparty. C is the value of collateral already held by the party under the agreement. The calculation is made by both parties and if they agree on the MTM and collateral parameters, they should agree on the required amount.

Consider a simplified example where a party trades with two counterparties under the terms of the collateral agreements listed in Table 3.2. Given the MTM values in Table 3.1, the collateral calculations from the perspective of Party A are shown in Table 3.3. In case *(a)* the net portfolio value is R−195,000. As the threshold represents uncollateralised exposure, only the exposure above the threshold will be collateralised. The amount is rounded and exceeds the minimum transfer amount, hence Party A will be required to post collateral to the value of R100,000. In case *(b)* the net portfolio value is R−200,000 which is reduced further by taking the value of held collateral into account. Applying thresholds and rounding results in Party A again being obliged to post collateral.

Trade Id	**Product**	**Counterparty**	**MTM**
1	Interest Rate Swap	Standard Bank	R−300,000
2	Forward Rate Agreement	Standard Bank	R50,000
3	Forward Rate Agreement	Standard Bank	R55,000
4	Interest Rate Swap	Rand Merchant Bank	R−300,000
5	Interest Rate Swap	Rand Merchant Bank	R100,000

Table 3.1: Trade exposures

Id	**Counterparty**	**Threshold**	**MTA**[13]	**Rounding**
SB CSA	Standard Bank	100000	50000	10000
RMB CSA	Rand Merchant Bank	50000	10000	5000

Table 3.2: Collateral agreements

[13]Minimum Transfer Amount.

(a) CSA with Standard Bank

	Party A
MTM	$-195{,}000$
Existing collateral	0
Required collateral	$-95{,}000$
Credit support amount	$-100{,}000$

(b) CSA with Rand Merchant Bank

	Party A
MTM	$-200{,}000$
Existing collateral	50,000
Required collateral	$-200{,}000$
Credit support amount	$-200{,}000$

Table 3.3: Collateral calculation

Chapter 4

Collateral optimisation

This chapter shows how the most basic form of a collateral optimisation problem can be formulated in a similar way to the transportation problem introduced in Section 2.1.3. An integer formulation is presented[1] with constraints that more accurately reflect the problem as explained in Chapter 3.

4.1 Problem definition

Collateral optimisation is concerned with the optimal allocation of collateral from a pool of available assets to meet margin call requirements as illustrated in Figure 4.1. The goal is to minimise the cost of using collateral and maximise the value of the assets that are retained in the collateral inventory. Certain restrictions such as the type of collateral that may be used or limitations on the quantities that can be allocated must also be taken into consideration.

For a high level overview of collateral optimisation, the driving factors and benefits of optimisation, see [51]. Some of the limitations of collateral optimisation, and the building blocks that need to be in place before advanced optimisation techniques can be applied are highlighted in [52].

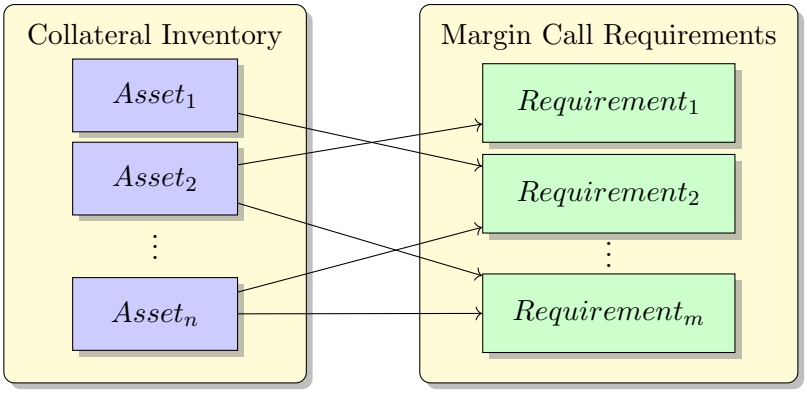


Figure 4.1: Asset allocation

[1] An earlier formulation was presented in [50].

Defining the cost of collateral assets is particularly important when specifying the objective function. The cost assigned to each asset is the primary driver of the allocation process. The simplest approach is to assign a preference weighting to each asset whereby the institution prefers to post lower ranked assets and retain assets with a higher rating. This weighting could be assigned manually or derived as function of certain properties of the asset itself, such as credit rating, liquidity or maturity of the asset.

If an institution has insufficient collateral to meet collateral requirements, it may engage in collateral *transformation* (also called collateral swaps or collateral upgrades), whereby collateral of the required type is traded for assets that are less easily used to secure transactions. Collateral transformation differs from collateral optimisation in that optimisation attempts to make the best use of existing assets in the collateral inventory, whereas with transformation, the inventory is adjusted by acquiring the required collateral [40]. For example an institution with a large corporate bond portfolio that is required to post cash for derivatives contracts can use a repo to borrow cash using the corporate bonds as collateral. If the institution is required to post government bonds, it can use the securities lending market to borrow sovereign bonds in exchange for the corporate bonds. Therefore, the cost of funding collateral is one of the key factors that needs to be considered when establishing which collateral is most valuable.

The economic and operational costs of collateral are important factors when specifying the cost of using assets in the collateral inventory. See [53] for a discussion of the limitations of a preference ranking cost model and the various factors and benefits of defining an institution specific economic cost model. For a detailed analysis of funding value adjustment (FVA) and the implications of funding collateral on derivatives pricing, see [47]. An in-depth analysis of optimising collateral allocation is given in [54]. The paper states that the goal of optimisation is not only to minimise the cost of posted collateral but also to minimise funding costs and maximise the liquidity of the retained inventory. The paper also quantifies the additional benefits of collateral optimisation, as well as other factors that act as constraints, that may need to be taken into consideration during the optimisation process.

The formulations in following sections assume that the cost function of assets has already been defined and focuses on the optimal allocation of collateral using mathematical optimisation methods.

4.2 Basic formulation

The fundamental problem can be modelled in a similar way to the transportation problem described in Section 2.1.3. The objective is to minimise

the cost of posting collateral when allocating assets from the collateral inventory to meet the margin call requirements.

Consider a simplified version of the model in which a number of assets $(A_1 \dots A_5)$ are used to meet requirements (R_1, R_2). Table 4.1 lists the available units of each asset and required amounts. Table 4.2 lists the cost per unit of allocating an asset to a specific requirement.

Assets	
Asset	Available Units
A_1	220
A_2	170
A_3	10
A_4	30
A_5	90

Requirements	
Requirement	Required Amount
R_1	200
R_2	300

Table 4.1: Assets and requirements

	Assets				
Requirements	A_1	A_2	A_3	A_4	A_5
R_1	10	7	3	2	1
R_2	10	8	4	2	1

Table 4.2: Allocation costs

Introducing the variable x_{ar} to represent the number of units of an asset A_i delivered to a requirement R_j, a linear programming formulation is given by

minimise

$$\begin{aligned} &10x_{11} + 10x_{12} + 7x_{21} + 8x_{22} + 3x_{31} \\ &+4x_{32} + 2x_{41} + 2x_{42} + x_{51} + x_{52} \end{aligned} \tag{4.1}$$

subject to

$$x_{11} + x_{12} \leq 220 \tag{4.2}$$

$$x_{21} + x_{22} \leq 170 \tag{4.3}$$

$$x_{31} + x_{32} \leq 10 \tag{4.4}$$

$$x_{41} + x_{42} \leq 30 \tag{4.5}$$

$$x_{51} + x_{52} \leq 90 \tag{4.6}$$

$$x_{11} + x_{21} + x_{31} + x_{41} + x_{51} = 200 \tag{4.7}$$

$$x_{12} + x_{22} + x_{32} + x_{42} + x_{52} = 300 \tag{4.8}$$

$$x_{ar} \geq 0 \tag{4.9}$$

For such a problem with m assets and n requirements the total number of constraints is $m+n$ and the total number of variables in the objective function is mn. The *availability constraints* (4.2) to (4.6) prevent the allocation of more units of an asset than are available. The *requirement constraints* (4.7) and (4.8) ensure that each requirement is met.

This problem can also be represented graphically as a bipartite graph with nodes divided into two disjoint sets, one for the assets and one for the requirements. As illustrated in Figure 4.2, the arcs represent the movement of assets across the network and the problem now is one of determining the minimum cost flow through the network. The positive asset nodes (A_i) can be seen as "sources" of flow entering the network and the negative nodes (R_j) as "sinks" where flow leaves the network. In order to ensure that the total flow into the network equals the total flow leaving the network, the available assets must be completely exhausted when meeting the requirements. In the linear programming formulation, the number of available assets ($220+170+10+30+90=520$) exceeds the required demand ($200+300=500$). In the network model, a dummy requirement R_3 is introduced to consume the excess 20 supply, giving the following equivalent linear program

minimise

$$\begin{aligned} &10x_{16}+7x_{26}+3x_{36}+2x_{46}+x_{56}+10x_{17}+8x_{27} \\ &+4x_{37}+2x_{47}+x_{57}+x_{18}+x_{28}+x_{38}+x_{48}+x_{58} \end{aligned} \tag{4.10}$$

subject to

$$x_{16}+x_{17}+x_{18}=220 \tag{4.11}$$

$$x_{26}+x_{27}+x_{28}=170 \tag{4.12}$$

$$x_{36}+x_{37}+x_{38}=10 \tag{4.13}$$

$$x_{46}+x_{47}+x_{48}=30 \tag{4.14}$$

$$x_{56}+x_{57}+x_{58}=90 \tag{4.15}$$

$$-1x_{16}+-1x_{26}+-1x_{36}+-1x_{46}+-1x_{56}=-200 \tag{4.16}$$

$$-1x_{17}+-1x_{27}+-1x_{37}+-1x_{47}+-1x_{57}=-300 \tag{4.17}$$

$$-1x_{18}+-1x_{28}+-1x_{38}+-1x_{48}+-1x_{58}=-20 \tag{4.18}$$

$$x_{ar}\geq 0 \tag{4.19}$$

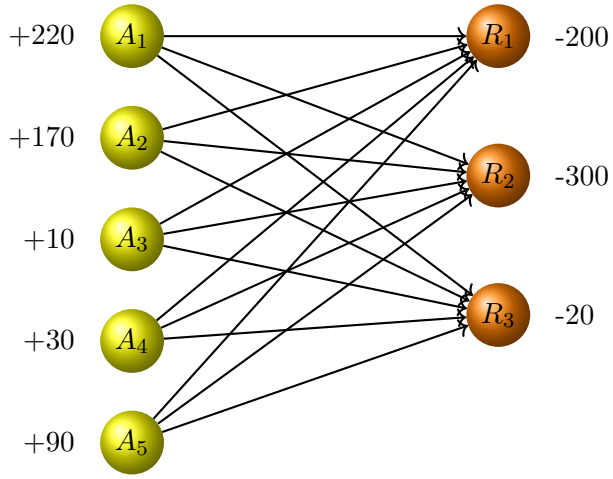


Figure 4.2: Flow Network

4.3 Integer formulation

A more realistic example with a more representative set of constraints requires the use of the integer programming techniques introduced in Section 2.2.3.

Let $\mathcal{A}$ be the index set of assets in the collateral inventory and let $\mathcal{R}$ be the index set of requirements that are obliged to be met. Let $x_{ar} \in \mathbb{Z}_+$ be a decision variable that represents the number of units of an asset $a \in \mathcal{A}$ allocated to requirement $r \in \mathcal{R}$ and let c_{ar} be the cost per unit of each movement. An integer linear programming formulation is given by

$$\min \sum_{a \in \mathcal{A}} \sum_{r \in \mathcal{R}} x_{ar} c_{ar} \tag{4.20}$$

s.t.

$$\sum_{r \in \mathcal{R}} x_{ar} \leq u_a \ \forall a \in \mathcal{A} \tag{4.21}$$

$$\sum_{a \in \mathcal{A}} x_{ar} m_a \geq v_r \ \forall r \in \mathcal{R} \tag{4.22}$$

$$x_{ar} \geq 0 \tag{4.23}$$

Availability constraints (4.21) prevent the allocation of more units of an asset than are available in the inventory where u_a is the number of available units for each asset.

Requirement constraints (4.22) ensure that the value of the posted collateral at least meets the value of the required amount v_r of the margin call, where m_a is the market value per unit of an asset.

As an example, consider the list of assets in the collateral inventory in Table 4.3 and the margin call requirements listed in Table 4.4. The cost of allocating an asset to a call is listed in Table 4.5, and the specific instruments the assets refer to are listed in Table 4.6.

Id	**Name**	**Market Value Per Unit**	**Available Units**	**Lot Size**	**Minimum Units**
A_1	ZAR	R100	1000	1	1
A_2	R186	R500	500	1	1
A_3	ES23	R200	0	1	1
A_4	TL20	R115	0	1	1
A_5	TKG	R110	0	1	1
A_6	AGL	R150	2000	1	1

Table 4.3: Collateral inventory

Id	**Required Amount**	**Agreement**	**Maximum Number Assets**
R_1	R100,000	SB CSA	3
R_2	R200,000	RMB CSA	3

Table 4.4: Margin call requirements

	Assets					
Requirements	A_1	A_2	A_3	A_4	A_5	A_6
R_1	5	3	2	2	1	1
R_2	5	3	2	2	1	1

Table 4.5: Cost matrix

Id	**Type**	**Issuer**	**Coupon**	**Maturity**	**ISIN**[2]
ZAR	Cash	Republic of South Africa			
R186	Government Bond	Republic of South Africa	10.5	2026/12/21	ZAG000016320
ES23	Corporate Bond	Eskom Holdings	10	2023/01/25	ZAG000074212
TL20	Corporate Bond	Telkom SA Limited	6	2020/02/24	ZAG000021528
TKG	Equity	Telkom SA Limited			ZAE000044897
AGL	Equity	Anglo American plc			GB00B1XZS820

Table 4.6: Cash and securities

[2]International Securities Identification Number (ISIN).

Implementing this basic case using IBM ILOG CPLEX (Appendix B.4 and B.5) results in the three collateral movements indicated in Figure 4.3. The two requirements are satisfied without using cash which had the highest cost. Requirement R_2 with a required amount of R200000 is 100% satisfied using 400 units of asset A_2 with a unit cost of 3. Requirement R_1 is slightly overcollateralised using 100 units of A_2 and 334 units of A_6 resulting in R100100. The resulting objective function value is $(400 \times 3 + 100 \times 3 + 334 \times 1 = 1834)$

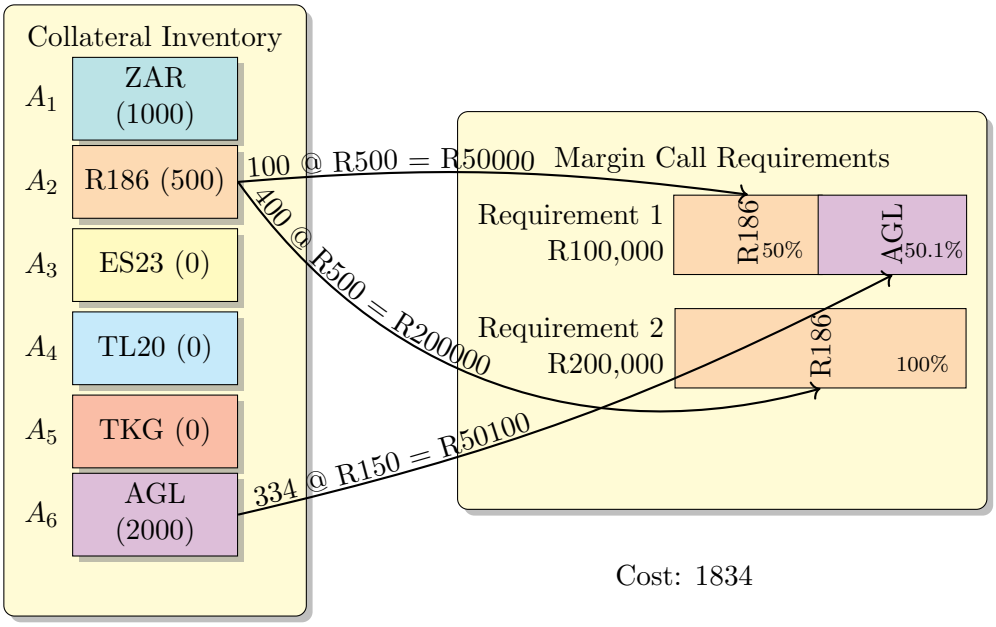


Figure 4.3: Allocation costs

4.3.1 Eligibility

An eligibility matrix captures the eligibility of an asset to be posted as collateral to a specific requirement. With $e_{ar} \in \{0, 1\}$ defined as in the matrix below, and incorporated into the requirement constraints (4.22), asset A_6 becomes ineligible for either requirement. The optimal solution is forced to draw upon asset A_1 to meet the requirements.

	A_1	A_2	A_3	A_4	A_5	A_6
R_1	1	1	1	1	1	0
R_2	1	1	1	1	1	0

Table 4.7: Eligibility matrix

4.3.2 Substitution

This basic model could be extended to support collateral substitution and recall by allowing the decision variable x_{ar} to take negative values. An alternative approach is to introduce an additional decision variable $z_{ar} \in \mathbb{Z}_+$ that represents the number of units of an asset $a \in \mathcal{A}$ that can be recalled from a requirement $r \in \mathcal{R}$.

Let $s_r \in \{0, 1\}$ indicate whether substitutions are allowed under the agreement or not, and let t_{ar} be the number of units of an asset that are currently allocated to a specific requirement. The objective function is altered to accommodate the additional decision variable and an additional constraint (4.24) ensures that it is not possible to recall more than is initially allocated.

$$z_{ar} \leq s_r t_{ar} \ \forall a \in \mathcal{A}, \forall r \in \mathcal{R} \tag{4.24}$$

Suppose requirement R_1 has an initial allocation of 5 units of A_5 with a total value of ($R110 \times 5 = R550$). It is cheaper to recall the 5 units and post an additional number of A_6 units to replace the recalled collateral. In this case 337 units of A_6 to the value of $R50550 are posted to R_1 and both calls are satisfied exactly. The total cost is reduced to ($100 \times 3 + 400 \times 3 + 337 \times 1 - 5 \times 1 = 1832$).

4.3.3 Lot Size

As assets are usually only allocated in discrete lot sizes, l_a is introduced into the objective function and the constraints, x_{ar} and z_{ar} will then represent the number of units allocated as a function of the lot size. A minimum lot size constraint is introduced with b_a giving the minimum number of units of an asset that can be allocated to meet a requirement. Let y_{ar} be a binary indicator variable such that

$$y_{ar} = \begin{cases} 1 & \text{if asset } a \text{ is allocated to requirement r} \\ 0 & \text{otherwise} \end{cases}$$

The minimum lot size constraint is given by

$$y_{ar} b_a \leq x_{ar} l_a \leq y_{ar} u_a \tag{4.25}$$

The logic employed in (4.25) ensures that if y_{ar} is 1 then $x_{ar} \geq b_a$ and if y_{ar} is 0 then x_{ar} is forced to 0.

4.3.4 Fragmentation

Reusing y_{ar}, the number assets assigned to a call, can be limited in order to reduce fragmentation. f_r places an upper bound on the number of different types of assets that are used to meet a call.

$$\sum_{a \in \mathcal{A}} y_{ar} \leq f_r \ \forall r \in \mathcal{R} \tag{4.26}$$

Similarly, concentration limits could be used to diversify the composition of assets by industry sector or issuer. Such constraints could be applied to individual requirements or to all assets in the collateral inventory.

4.3.5 Diversification

Diversification constraints limit the allocation of specific assets or from a specific issuer when meeting a particular call.

Asset level diversification Let d_{ar} be a limit on the proportion of asset a that is allowed to be assigned to requirement r.

$$\frac{x_{ar} e_{ar} m_a l_a}{\sum\limits_{k \in \mathcal{A}} x_{kr} e_{kr} m_k l_k} \leq d_{ar} \ \forall a \in \mathcal{A}, \forall r \in \mathcal{R} \tag{4.27}$$

Issuer level diversification Let d_{ir} be a limit on the proportion of issuer i that is allowed to be assigned to requirement r and let $\mathcal{A}(i)$ be an index set used in order to filter assets from a particular issuer $i \in \mathcal{I}$.

$$\frac{\sum\limits_{a \in \mathcal{A}(i)} x_{ar} e_{ar} m_a l_a}{\sum\limits_{k \in \mathcal{A}} x_{kr} e_{kr} m_k l_k} \leq d_{ir} \ \forall i \in \mathcal{I}, \forall r \in \mathcal{R} \tag{4.28}$$

4.3.6 Complete formulation

The complete formulation is given by

$$\min \sum_{a\in\mathcal{A}}\sum_{r\in\mathcal{R}} x_{ar}l_a c_{ar} - \sum_{a\in\mathcal{A}}\sum_{r\in\mathcal{R}} z_{ar}l_a c_{ar} \tag{4.29}$$

s.t.

$$\sum_{r\in\mathcal{R}} x_{ar}l_a \leq u_a \ \forall a \in \mathcal{A} \tag{4.30}$$

$$\sum_{a\in\mathcal{A}} x_{ar}l_a m_a e_{ar} - z_{ar}l_a m_a \geq v_r \ \forall r \in \mathcal{R} \tag{4.31}$$

$$z_{ar} \leq s_r t_{ar} \forall a \in \mathcal{A}, \forall r \in \mathcal{R} \tag{4.32}$$

$$y_{ar}b_a \leq x_{ar}l_a \leq y_{ar}u_a \forall a \in \mathcal{A}, \forall r \in \mathcal{R} \tag{4.33}$$

$$\sum_{a\in\mathcal{A}} y_{ar} \leq f_r \ \forall r \in \mathcal{R} \tag{4.34}$$

$$\frac{x_{ar}l_a m_a e_{ar}}{\sum\limits_{k\in\mathcal{A}} x_{kr}l_k m_k e_{kr}} \leq d_{ar} \ \forall a \in \mathcal{A}, \forall r \in \mathcal{R} \tag{4.35}$$

$$\frac{\sum\limits_{a\in\mathcal{A}(i)} x_{ar}l_a m_a e_{ar}}{\sum\limits_{k\in\mathcal{A}} x_{kr}l_k m_k e_{kr}} \leq d_{ir} \ \forall i \in \mathcal{I}, \forall r \in \mathcal{R} \tag{4.36}$$

$$x_{ar} \geq 0 \tag{4.37}$$

Chapter 5

Computational results

This chapter gives details of how the test data files were generated, and presents the results of implementing and testing the model on the various data sets. Different problem sizes are compared, and the effect of the sparsity of the eligibility matrix is shown. The influence of initial allocations and the impact of diversification constraints are reported.

5.1 Model implementation

The model for the integer formulation outlined in Section 4.3 was implemented and tested using IBM ILOG CPLEX Optimization Studio version 12.7.1. Comparative results were generated on the Amazon EC2 cloud-computing platform using a `m4.2xlarge` instance with 8 vCPUs and 32GiB of memory. CPLEX was configured with a mixed integer programming (MIP) emphasis that balances optimality and feasibility. A dynamic MIP search method in deterministic parallel mode was used.

5.2 Data file generation

Chapter 3 described the types of data required before collateral optimisation can take place. The required amount calculation, for instance, depends on information from the collateral agreements. This information is not always readily available, due to the fact that banks are still finalising the processes and systems for capturing data of this nature. The agreements themselves are also still evolving in line with legislation. For these reasons, synthetic data was generated for the purposes of this study.

To test the integer model, various test data files were generated. The number of assets and requirements were varied between 100 and 500, in increments of 100, resulting in $5 \times 5 = 25$ variations. The density of the eligibility matrix was varied between 25, 50, 75 and 100%, resulting in an additional 4 variations, where a sparse matrix implies that fewer assets are available to meet the requirements.

When initial allocations are present, the collateral giver is entitled to replace the existing collateral with cheaper eligible alternatives. A variation was added for call requirements where initial asset allocations were present, and another variation where no initial allocations were present.

The number of available units and the market value per unit for each of the assets were randomly generated. Similarly for the requirements, the required amounts were randomly generated. No bounds were placed on the diversification of assets or issuers. In order to establish an average runtime, 5 different random files were generated for each of the variations, resulting in $25 \times 4 \times 2 \times 5 = 1000$ test data files.

For cases where no more than a certain percentage of a call requirement can be satisfied by the same asset, asset diversification constraints are introduced. To test the effect of such constraints, a further set of files were generated with only 100 assets and a 100 requirements, in which the asset diversification percentage was varied between 70, 80, 90 and 100%. As above, 5 different random files were generated.

5.3 Transportation algorithm results

As described in Chapter 4, the basic collateral optimisation problem can be represented as a balanced transportation problem that can be solved using transportation algorithms.

In order to demonstrate the use of a transportation algorithm on the collateral optimisation problem, the modified distribution method was implemented in the Python programming language. An implementation of the stepping stone method is provided in Appendix B, but as stated in Section 2.1.3, the modified distribution method is computationally more efficient and therefore only results for this method were generated.

Runtime performance was compared across various problem instances. Test data files with an equal number of assets and requirements were generated. The problem instances were balanced before being passed to the algorithm and the northwest corner method was used to find the basic initial feasible solution. Table 5.1 shows the increase in runtime as the problem size increases. The performance of the Amazon cloud computing platform (AWS EC2) is compared to that of an Intel i5-3317U CPU @ 1.70GHz. Performance on the cloud platform is roughly five times faster than the dual core machine.

It should be noted that due to Python's Global Interpreter Lock (GIL) the implementation is not making full use of all the processor cores. The Python implementation makes use of the NumPy library's efficient arrays, and can

be refined and improved further for example, by using the multiprocessing package. However, the final column in Table 5.1 shows how fast a CPLEX implementation of the basic formulation is. These results are from the dual core machine, where CPLEX has been instructed to use only a single thread. On the cloud platform, the solving duration for CPLEX when multithreading is enabled is less than a second for all problem size variations of this basic model.

Figure 5.1 shows that the computational effort with respect to problem size. The following sections describes the performance of a more complete formulation of the problem implemented using the CPLEX optimisation software package.

Assets	**Requirements**	**AWS EC2**	**i5-3317U**	**CPLEX**
100	100	2	10	0
200	200	19	118	0
300	300	91	338	3
400	400	171	817	5
500	500	343	2548	10

Table 5.1: Modified distribution method solving duration in seconds

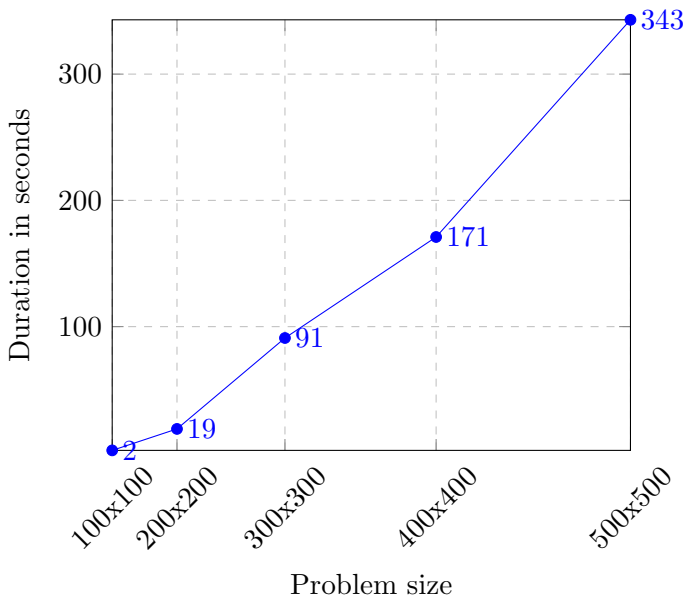


Figure 5.1: Modified distribution method solving duration

5.4 Integer program results

5.4.1 Problem size variations

Table 5.2 shows the average solving duration in seconds as the number of assets and requirements is varied. The number of initial allocations per call has been set to zero and all assets are eligible to be posted to every call. These values are plotted in Figure 5.2. As described in Section 2.2.3, the CPLEX relative MIP gap tolerance[1] was set to 0.01, instructing the solver to stop as soon as a feasible integer solution proved to be within 1% of the optimal is found.

It is evident from the heatmap in Table 5.2 that the solver is quite easily able to find an optimal solution, even with 500 assets in the inventory and 500 requirements that need to be met. Figure 5.2 shows the increase in time taken to solve the problem as the number of assets and requirements increase, and gives an indication as to the exponential nature of the problem.

	Number of requirements				
Number of assets	100	200	300	400	500
100	0.00	1.00	2.00	3.00	4.40
200	1.80	4.00	7.00	9.80	13.20
300	4.00	8.60	13.60	18.60	25.20
400	6.00	13.40	21.20	28.80	44.80
500	9.60	20.20	32.20	49.40	78.40

Table 5.2: Average solving duration in seconds

[1] IloCplex.DoubleParam EpGap - Sets a relative tolerance on the gap between the best integer objective and the objective of the best node remaining.

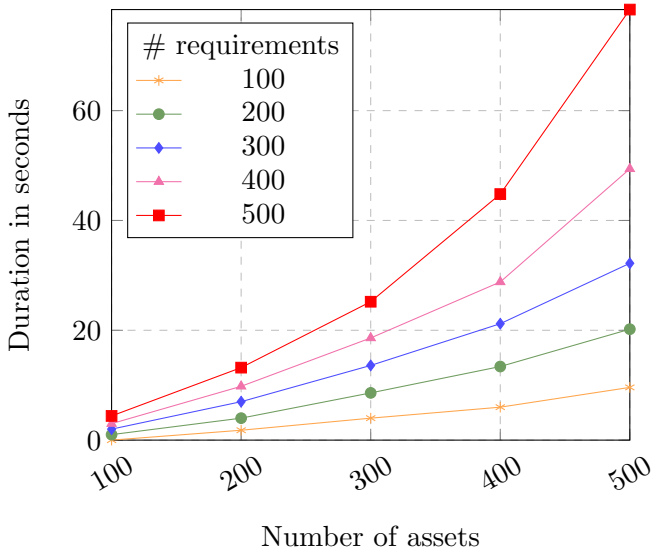


Figure 5.2: Average solving time for assets and requirements

5.4.2 Eligibility matrix density

The impact of varying the density of the eligibility matrix between 25% and 100% is illustrated in Figure 5.3, which shows that the runtimes decrease linearly as the eligibility matrix becomes more sparse.

This is likely due to the fact that the solver preprocessor is simplifying the problem. As discussed in Section 2.2.3, the presolver is able to reduce the size of the problem by decreasing the number of rows and columns based on the binary eligibility value. This is similar to the way that certain arcs could be excluded from a network flow formulation of the problem, as there is no path from a certain asset to a particular requirement. A custom preprocessor could also exclude these values from the input data before passing the data to the solver.

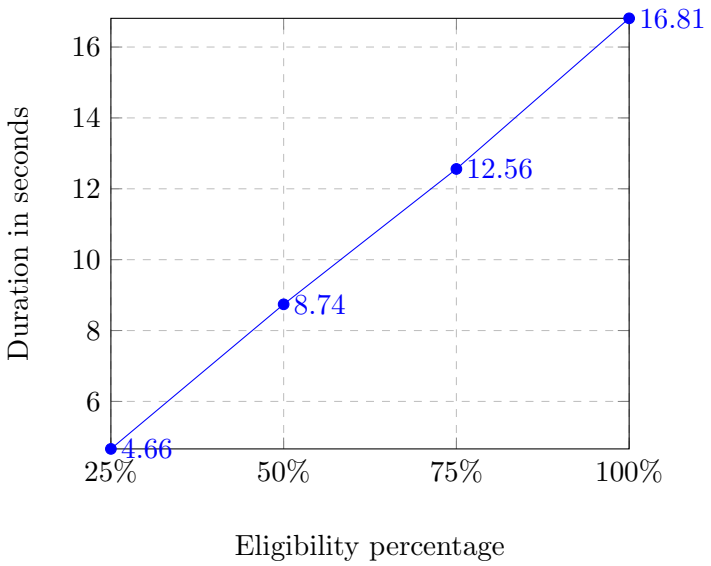


Figure 5.3: Eligibility matrix density average solving time

5.4.3 Initial allocations

To test the effect of substitutions, the runtimes of call requirements with initial allocations were compared to those where there were no initial asset allocations. Table 5.3 shows the average solving time in seconds with "False" indicating that no initial allocations were present and "True" indicating a random number of initial allocations were generated per call. These values are plotted in Figure 5.4.

As expected, when initial allocations are present, additional computational effort is required to solve the problem. From Figure 5.4 it appears that taking previously allocated collateral into consideration could possibly increase the exponential growth rate of the problem.

	Initial allocation	
Number of Req	False	True
100	4.28	4.56
200	9.44	12.24
300	15.20	19.76
400	21.92	29.40
500	33.20	42.20

Table 5.3: Initial allocation average solving duration in seconds

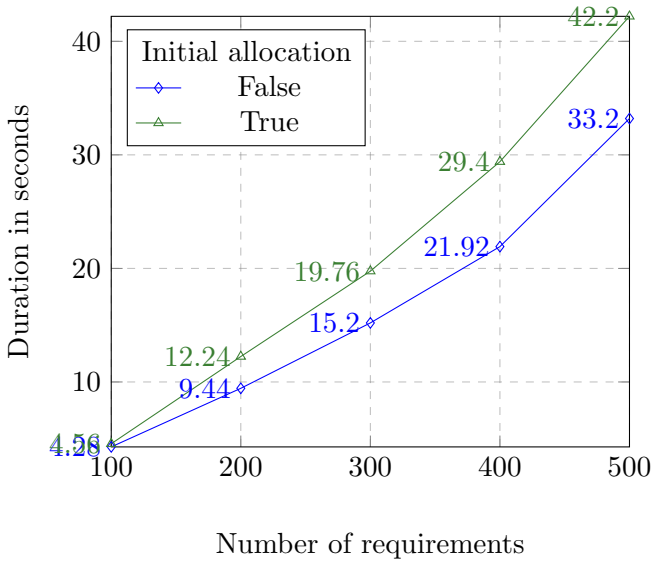


Figure 5.4: Initial allocation average solving time

5.4.4 Diversification

The results of varying the asset diversification percentage between 70% and 100% are shown in Table 5.4. A time limit of 10 minutes was set in CPLEX for finding a solution to the problem[2]. Once this time limit is reached, the program reports the objective value of the current best solution and the relative MIP gap. The status indicates whether an optimal solution was returned or whether solving was aborted due to the time limit being reached.

With the diversification percentage set to 100%, i.e. where there is no limit on the percentage composition of assets that are used to meet a particular call, the optimal solution is found very quickly. However, when the proportion of an asset that is assigned to meet a requirement is constrained, the solution is not found in a reasonable amount of time. It is clear that introducing a diversification percentage significantly affects the time taken to solve the model. Table 5.4 shows the cases where CPLEX has stopped due to the time limit, and was only able to find a feasible integer solution that was within 20% of the optimal solution.

[2]IloCplex.DoubleParam TiLim - Sets the maximum time, in seconds, for a call to an optimiser.

% Asset Diversification	Solving Duration	Cplex Status	MIP Relative Gap	Objective Value
70%	600	AbortTimeLim	0.228312	234.816107
70%	600	AbortTimeLim	0.236872	233.422710
70%	600	AbortTimeLim	0.238918	233.754236
70%	600	AbortTimeLim	0.244675	231.709513
70%	600	AbortTimeLim	0.256874	228.881757
80%	600	AbortTimeLim	0.219213	228.306086
80%	600	AbortTimeLim	0.222089	226.875527
80%	600	AbortTimeLim	0.225224	230.925027
80%	600	AbortTimeLim	0.232585	228.719331
80%	600	AbortTimeLim	0.248602	227.452867
90%	600	AbortTimeLim	0.173254	228.489343
90%	600	AbortTimeLim	0.177279	225.563920
90%	600	AbortTimeLim	0.177478	228.207200
90%	600	AbortTimeLim	0.183292	228.728429
90%	600	AbortTimeLim	0.217746	227.045425
100%	0	Optimal	0.0	108.884805
100%	0	Optimal	0.0	109.538293
100%	0	Optimal	0.0	109.980338
100%	0	Optimal	0.0	110.967328
100%	0	Optimal	0.0	111.069271

Table 5.4: Asset diversification percentage

Chapter 6

Conclusion

This dissertation gave an overview of linear and integer programming and related algorithms.

The function of collateral management and the factors driving the increased demand for collateral were discussed. In particular, the impact of regulation and the role of technology were highlighted. The importance of the optimal allocation of collateral and the need for advanced information systems were also noted.

The various types of constraints that are applicable to the collateral optimisation problem were discussed and a mathematical formulation of the problem was presented. The model was implemented and tested on various data sets. The results showed that the collateral optimisation problem with a set of basic constraints can be solved within a reasonable amount of time, even for relatively large problem sizes. However, certain types of constraints, such as those related to the diversification of assets, can significantly affect the time taken to solve the model.

Future work could explore the applicability of the model to real-world data sets. The model could also be extended to include various other types of constraints such as recalling collateral when it becomes ineligible, for example after a ratings downgrade. A multiperiod model could be developed that not only takes the current collateral requirements into consideration, but also future collateral requirements. Future collateral requirements could be forecast based on future cashflows or corporate actions.

This study only considered computation time as a measure of the behaviour and performance of the model, but both memory usage and solution time rise as the number of integer variables grow. For mixed integer programs, the set of active nodes in the branch and bound tree can consume large amounts of memory. The problem size alone may not be a good indicator of the lower bound on memory usage. It could be useful to analyse the CPLEX log file to gauge how variations in the data impact memory usage. The log file contains the time spent in the current MIP optimisation as well as the amount of memory used by branch-and-cut. More detailed experiments could be performed with different CPLEX parameters that control variable selection and node selection as described in 2.2.3.

The effectiveness of constraint programming at expressing institutional-specific requirements could also be explored. For example, the fact that a certain type of asset was previously posted to satisfy a requirement, may imply that more of the same asset should be posted, even if the allocation is not the most optimal.

In order to overcome the limitations seen in the diversification constraints, heuristics methods might be employed to find good approximate solutions in cases where the standard algorithms were not able to find a solution in a reasonable amount of time.

Institutions that are faced with the challenge of selecting and implementing a collateral management system at significant expense, should consider using open source solvers to develop an in-house optimisation model. Such a prototype could ensure that the necessary building blocks are in place before trying to optimise the allocation process. For example, collateral agreements need to be electronically captured, with eligibility criteria, netting sets etc. and the composition of assets in the inventory and their associated costs needs to be well defined. Once these foundations are in place, the organisation can consider spending further capital on more advanced commercial software and solvers to handle the more complicated types of requirements and constraints.

Bibliography

[1] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming, 2^{nd} Edition*. Duxbury/Thomson, 2003.

[2] Gérard Cornuéjols and Reha Tütüncü. *Optimization Methods in Finance*. Mathematics, Finance and Risk. Cambridge University Press, 2007.

[3] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley Series in Discrete Mathematics & Optimization. John Wiley & Sons, 1998.

[4] George B. Dantzig. Linear programming. *Operations Research*, 50(1):42–47, 2002.

[5] A. Arbel. *Exploring Interior-point Linear Programming: Algorithms and Software*. Foundations of computing. MIT Press, 1993.

[6] Robert E Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, pages 107–121, 2012.

[7] Richard Elwes. The algorithm that runs the world. *New Scientist*, (2877):32–37, August 2012.

[8] M.S. Bazaraa, J.J. Jarvis, and H.D. Sherali. *Linear Programming and Network Flows*. Wiley, 2010.

[9] M. Jeter. *Mathematical Programming: An Introduction to Optimization*. Chapman & Hall/CRC Pure and Applied Mathematics. Taylor & Francis, 1986.

[10] H.S. Kasana and K.D. Kumar. *Introductory Operations Research: Theory and Applications*. Springer, 2004.

[11] S.T. Rachev and L. Rüschendorf. *Mass Transportation Problems: Volume 1: Theory*. Probability and Its Applications. Springer New York, 2006.

[12] A. M. Vershik. Long History of the Monge-Kantorovich Transportation Problem. *The Mathematical Intelligencer*, 35(4):1–9, 2013.

[13] Alexander Schrijver. On the history of the transportation and maximum flow problems. *Mathematical Programming*, 91(3):437–445, 2002.

[14] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Number v. 1 in Algorithms and Combinatorics. Springer Berlin Heidelberg, 2003.

[15] Frank L. Hitchcock. The distribution of a product from several sources to numerous localities. *Journal of Mathematics and Physics*, 20(1-4):224–230, 1941.

[16] Tjalling C. Koopmans. Optimum utilization of the transportation system. *Econometrica*, 17:136–146, 1949.

[17] A. Galichon. *Optimal Transport Methods in Economics*. Princeton University Press, 2016.

[18] L. R. Ford and D. R. Fulkerson. Solving the transportation problem. *Management Science*, 3(1):24–32, 1956.

[19] G. B. Dantzig. Application of the simplex method to a transportation problem. *Activity Analysis of Production and Allocation, Cowles Commission Monograph 13*, pages 359–373, 1951.

[20] A. Shafaat and S. K. Goyal. Resolution of degeneracy in transportation problems. *The Journal of the Operational Research Society*, 39(4):411–413, 1988.

[21] N.V. Reinfeld and W.R. Vogel. *Mathematical Programming*. Prentice-Hall, 1958.

[22] A. Charnes and W. W. Cooper. The stepping stone method of explaining linear programming calculations in transportation problems. *Management Science*, 1(1):49–69, 1954.

[23] D.S. Hira. *Operations Research*. S. Chand Limited, 2008.

[24] Saul I. Gass. On solving the transportation problem. *Journal of the Operational Research Society*, 41(4):291–297, Apr 1990.

[25] H. Paul Williams. *Model Building in Mathematical Programming, 5^{th} Edition*. Wiley, 2013.

[26] Wayne L. Winston and Munirpallam Venkataramanan. *Introduction to Mathematical Programming, Operations Research: Volume One, Fourth Edition*. Thomson Learning, 2002.

[27] S.P. Bradley, A.C. Hax, and T.L. Magnanti. *Applied Mathematical Programming*. Addison-Wesley Publishing Company, 1977.

[28] W.L. Winston and J.B. Goldberg. *Operations Research: Applications and Algorithms*. Thomson Brooks/Cole, 2004.

[29] Hamdy A. Taha. *Operations Research: An Introduction, 10th Edition*. Pearson, 2017.

[30] Ashutosh Mahajan. *Presolving Mixed–Integer Linear Programs*. John Wiley & Sons, Inc., 2010.

[31] Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. Technical report, Technical Report 16-44, ZIB, Takustr. 7, 14195 Berlin, 2016.

[32] Modeling with OPL (IBM Optimization Programming Language). `https://www-01.ibm.com/software/commerce/optimization/modeling/` . Accessed: 2017-03-20.

[33] SAS/OR(R) 13.2 User's Guide: Mathematical Programming - Overview: OPTMODEL Procedure. `http://support.sas.com/documentation/cdl/en/ormpug/67517/HTML\/default/viewer.htm` . Accessed: 2017-03-20.

[34] Matthew J. Saltzman. *Coin-Or: An Open-Source Library for Optimization*, pages 3–32. Springer US, Boston, MA, 2002.

[35] Andrew J. Mason. *OpenSolver - An Open Source Add-in to Solve Linear and Integer Progammes in Excel*, pages 401–406. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[36] Panayiota Koulafetis. *Modern Credit Risk Management: Theory and Practice*. Palgrave Macmillan UK, 2017.

[37] Jon Gregory. *Counterparty Credit Risk: The new challenge for global financial markets*. The Wiley Finance Series. John Wiley & Sons, 2010.

[38] Shyam Venkat and Stephen Baird. *Liquidity Risk Management: A Practitioner's Perspective*. Wiley Finance. Wiley, 2016.

[39] Bilgehan Aydin. Evolution of collateral 'management' into collateral 'optimisation'. *Journal of Securities Operations & Custody*, 8(3), 2016.

[40] Jeannette Capel and Anouk Levels. Collateral optimisation, re-use and transformation - Developments in the Dutch financial sector. *DNB Occasional Studies*, 12(5), 2014. https://www.dnb.nl/binaries/415234_DX0_DNB_OS_12-05_eng-WEB_tcm46-309555.pdf
Accessed: 2017-08-17.

[41] National Treasury Republic of South Africa. Explanatory Memorandum on the Financial Markets Act Regulations 2016: Regulating over-the-counter derivative markets in South Africa.
http://www.treasury.gov.za/OTC/Explanatory%20Memo%20%20Regulating%20OTC%20Markets_July2016.pdf
, 2016. Accessed: 2017-08-01.

[42] DTCC white paper: Trends, Risks and Opportunities in Collateral Management.
http://www.dtcc.com/~/media/Files/Downloads/WhitePapers/CollateralMGMT_WhitePaper.ashx
, 2014. Accessed: 2017-08-17.

[43] IBM - The optimization of everything: OTC derivatives, counterparty credit risk and funding.
https://www-935.ibm.com/services/multimedia/SIS_-_The_Optimization_of_everything_whitepaper.pdf
, 2013. Accessed: 2017-08-01.

[44] Cyril Louchtchay de Fleurian and Didier Bensaid. Clearstream and Elton-Pickford, Collateral Optimistion - The value chain of collateral: liquidity, cost & capital perspectives.
http://www.clearstream.com/blob/66616/9250c222407c9aad032ff51f8e4befa3/eltonpickford-data.pdf
, 2014. Accessed: 2017-08-01.

[45] Martin Seagroatt, David Field, and Paul Wilson. 4sight and Rule Financial white paper: Buyside Collateral Management - Challenges and Opportunities.
http://www.4sight.com/media/9412/4sight%20and%20Rule%20Financial%20Whitepaper%20-%20Buy%20Side%20Collateral%20Challenges%20and%20Opportunities.pdf
, 2015. Accessed: 2017-08-17.

[46] 2005 ISDA Collateral Guidelines.
http://www.isda.org/publications/pdf/2005isdacollateralguidelines.pdf
. Accessed: 2017-07-11.

[47] Jon Gregory. *The XVA Challenge: Counterparty Credit Risk, Funding, Collateral, and Capital*. The Wiley Finance Series. Wiley, 2015.

[48] Committee on the Global Financial System (CGFS). *Asset encumbrance, financial reform and the demand for collateral assets*. CGFS Papers No 49. Bank for International Settlements, 2013.
http://www.bis.org/publ/cgfs49.pdf
Accessed: 2017-08-17.

[49] Jean-Marc Bottazzi, Jaime Luque, and Mário R. Páscoa. Securities market theory: Possession, repo and rehypothecation. *Journal of Economic Theory*, 147(2):477 – 500, 2012.

[50] P.G. Reynolds and S.E. Terblanche. An integer linear programming formulation for collateral optimisation. In *Proceedings of the 44th Annual ORSSA Conference*, pages 54–61. Operations Research Society of South Africa.

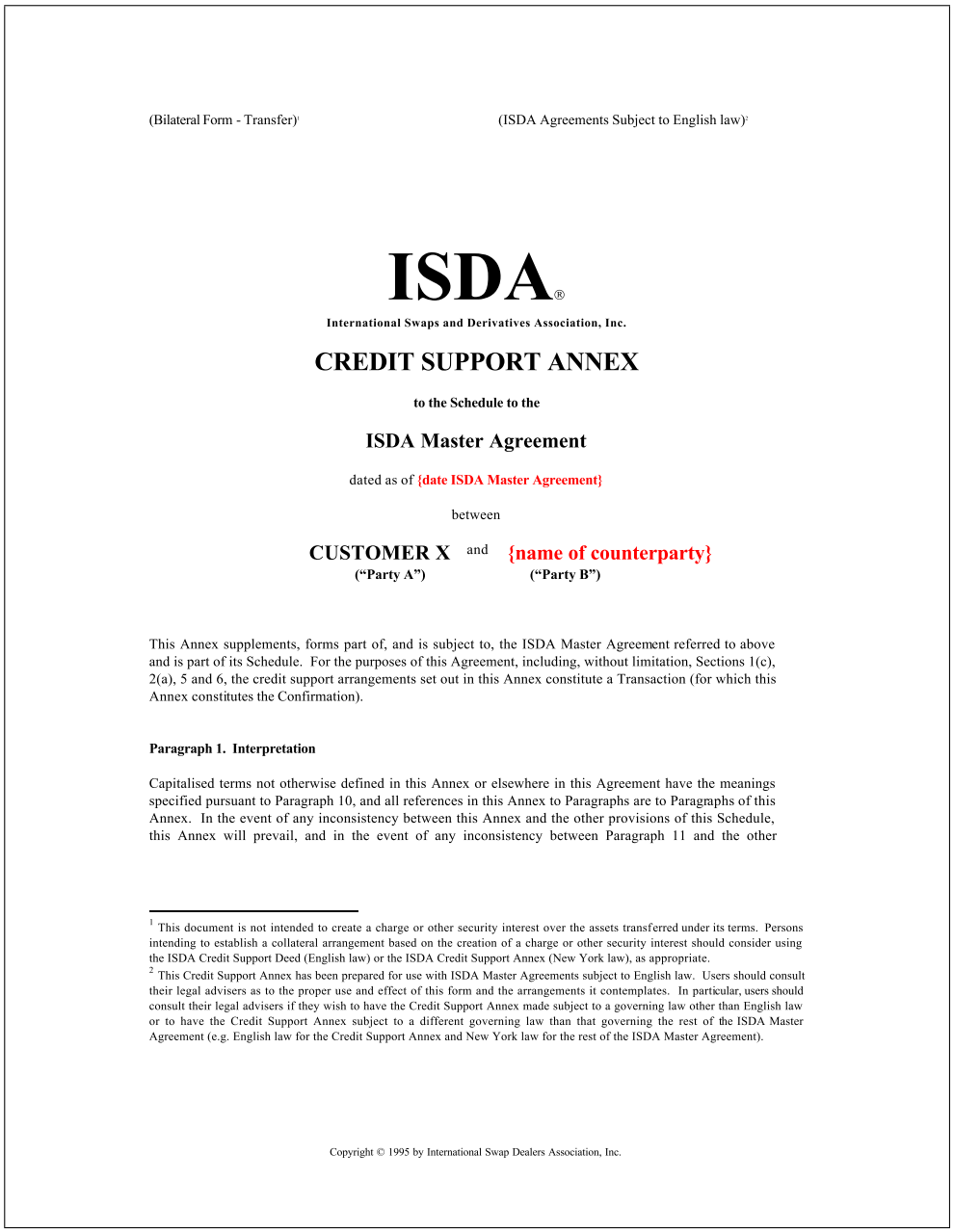
[51] Martin Seagroatt. 4sight white paper: Collateral Optimisation in a Centrally Cleared World.
http://www.4sight.com/media/2310/4sight%20Whitepaper%20-%20Collateral%20Optimisation%20in%20a%20Centrally%20Cleared%20World.pdf
, 2012. Accessed: 2017-08-17.

[52] Martin Seagroatt and Paul Wilson. 4sight white paper: Collateral Optimization - Beyond Cheapest to Deliver and the Big Red Button.
http://www.4sight.com/media/8202/4sight%20Whitepaper%20-%20Beyond%20Cheapest%20to%20Deliver%20and%20the%20Big%20Red%20Button.pdf
, 2015. Accessed: 2017-08-17.

[53] Ted Allen and Ed Hellaby. SunGard white paper: Collateral Optimization - How it really works.
http://finance.flemingeurope.com/webdata/4201/WP_CollateralOptimization_HowItReallyWorks_2013.pdf
, 2013. Accessed: 2017-09-01.

[54] Thomas Schiebe, Sendi Cigura, Ted Allen, and Sven Ludwig. Sapient Global Markets and FIS: Techniques for post-trade collateral optimization.
https://www.fisglobal.com/solutions/institutional-and-wholesale/asset-management/-/media/fisglobal/files/report/techniques-for-post-trade-collateral-optimization.pdf
, 2016. Accessed: 2017-08-01.

Appendix A: Exhibits



(Bilateral Form - Transfer)[1] (ISDA Agreements Subject to English law)[2]

ISDA®

International Swaps and Derivatives Association, Inc.

CREDIT SUPPORT ANNEX

to the Schedule to the

ISDA Master Agreement

dated as of {date ISDA Master Agreement}

between

CUSTOMER X ("Party A") and **{name of counterparty}** ("Party B")

This Annex supplements, forms part of, and is subject to, the ISDA Master Agreement referred to above and is part of its Schedule. For the purposes of this Agreement, including, without limitation, Sections 1(c), 2(a), 5 and 6, the credit support arrangements set out in this Annex constitute a Transaction (for which this Annex constitutes the Confirmation).

Paragraph 1. Interpretation

Capitalised terms not otherwise defined in this Annex or elsewhere in this Agreement have the meanings specified pursuant to Paragraph 10, and all references in this Annex to Paragraphs are to Paragraphs of this Annex. In the event of any inconsistency between this Annex and the other provisions of this Schedule, this Annex will prevail, and in the event of any inconsistency between Paragraph 11 and the other

[1] This document is not intended to create a charge or other security interest over the assets transferred under its terms. Persons intending to establish a collateral arrangement based on the creation of a charge or other security interest should consider using the ISDA Credit Support Deed (English law) or the ISDA Credit Support Annex (New York law), as appropriate.

[2] This Credit Support Annex has been prepared for use with ISDA Master Agreements subject to English law. Users should consult their legal advisers as to the proper use and effect of this form and the arrangements it contemplates. In particular, users should consult their legal advisers if they wish to have the Credit Support Annex made subject to a governing law other than English law or to have the Credit Support Annex subject to a different governing law than that governing the rest of the ISDA Master Agreement (e.g. English law for the Credit Support Annex and New York law for the rest of the ISDA Master Agreement).

Figure A.1: ISDA Credit Support Annex

Paragraph 11. Elections and Variables

(a) ***Base Currency and Eligible Currency.***

(i) "Base Currency" means United States Dollars unless otherwise specified here:
.....

(ii) "Eligible Currency" means the Base Currency and each other currency specified here:
.....
.....

(b) ***Credit Support Obligations.***

(i) ***Delivery Amount, Return Amount and Credit Support Amount.***

(A) ***"Delivery Amount"*** has the meaning specified in Paragraph 2(a), unless otherwise specified here: .....

(B) ***"Return Amount"*** has the meaning specified in Paragraph 2(b), unless otherwise specified here: .....

(C) ***"Credit Support Amount"*** has the meaning specified in Paragraph 10, unless otherwise specified here: .....

(ii) ***Eligible Credit Support.*** *The following items will qualify as "Eligible Credit Support" for the party specified:*

		Party A	Party B	Valuation Percentage
(A)	cash in an Eligible Currency	[ ]	[ ]	[ ]%
(B)	negotiable debt obligations issued by the Government of [ ] having an original maturity at issuance of not more than one year	[ ]	[ ]	[ ]%
(C)	negotiable debt obligations issued by the Government of [ ] having an original maturity at issuance of more than one year but not more than 10 years	[ ]	[ ]	[ ]%

Figure A.2: ISDA CSA eligible collateral

Securities borrowing and lending

- Securities or stock lending refers to the lending of securities by one party to another in exchange for collateral.
- Motivated by the need to borrow securities.
- Market in Europe represented by the International Securities Lending Association (ISLA).
- Legal contract: Global Master Securities Lending Agreement (GMSLA).

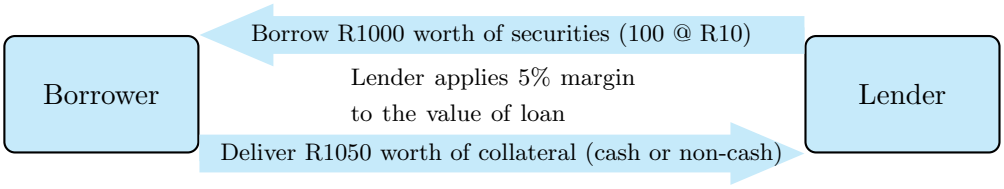
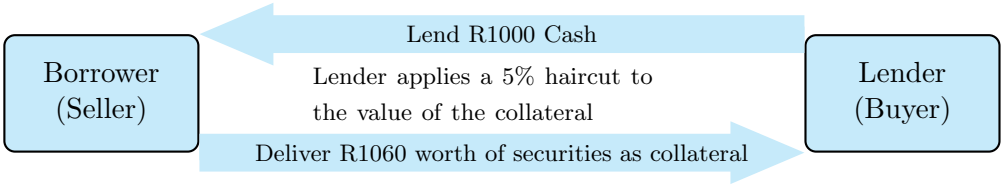


Figure A.3: Securities borrowing and lending

Repurchase agreement

- A repurchase agreement (repo) is the sale of securities together with an agreement from the seller to buy them back at a future date and price. If the seller defaults during the life of the deal, the buyer can sell the asset to offset the loss. The asset therefore acts as collateral and mitigates the credit risk that the buyer has on the seller.
- Motivated by the need to borrow and lend cash.
- Market in Europe represented by the European Repo and Collateral Council (ERCC) of the International Capital Market Association (ICMA).
- Legal contract: Global Master Repurchase Agreement (GMRA).



106 units @ R10$\times(1-0.05)$ = R1007 (market value = R1060)

Figure A.4: Repurchase agreement

Over the counter derivative

- A derivative is a security that's value is derived from the performance of an underlying asset.
- Trading is done directly between two counterparties as opposed to trading on an exchange.
- Terms are documented in an International Swaps and Derivatives Association (ISDA) Master Agreement with Credit Support Annex (CSA).

Example: An Interest Rate Swap (IRS) is an interest rate derivative that can be used to hedge exposure to changes in interest rates or to take advantage of a declining interest rate environment by changing from a fixed to a floating rate. Party A agrees to pay Party B a fixed rate of interest based on a specified notional amount and party B agrees to make floating rate payments to Party A linked to a rate index such as 3 month JIBAR[1].If interest rates rise, Party A benefits as they are paying a fixed rate but now receiving the difference between the fixed rate and the higher floating rate. Party B gains if interest rates drop.

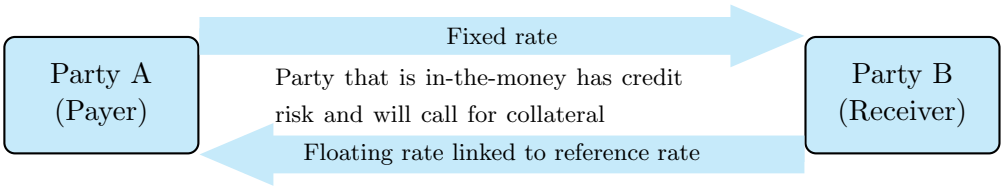


Figure A.5: OTC derivative

[1]The Johannesburg Interbank Average Rate (JIBAR) benchmark average interest rate at which South African banks are prepared to lend to one another in Rand with a maturity of 3 months.

Appendix B: Source Code

```
from numpy import dot, array, linalg, subtract, min, amax,
    where
from numpy.linalg import inv

def simplex_method():
    c = array([6, 9, 0, 0])
    A = array([
         [10, 20, 1, 0],
         [40, 30, 0, 1]])
    b = array([[200], [450]])

    nonbasic_idx = array([0, 1]);
    basic_idx = array([2, 3]);

    optimal = False
    iteration = 0;
    while not optimal:
        iteration += 1
        print "iteration", iteration
        B = A[:, basic_idx];
        N = A[:, nonbasic_idx];
        cb = c[basic_idx];
        cn = c[nonbasic_idx];
        xn = array([[0], [0]])
        xb = inv(B).dot(b);
        reduced_costs = subtract(cn, cb.dot(inv(B)).dot(N))
        max_rc = amax(reduced_costs)
        if max_rc <= 0:
            print "optimal"
            optimal = True
            print "Z", dot(cb, xb)
        else:
            entering_idx = where(reduced_costs == max_rc)
            entering_variable = nonbasic_idx[entering_idx];
            entering_column = N[:, entering_variable];
            ratios = xb / inv(B).dot(entering_column);
            positive_ratios = ratios[ratios > 0]
            min_ratio = min(positive_ratios)
            ratios = ratios.tolist()
            leaving_idx = ratios.index(min_ratio)
            tmp = basic_idx[leaving_idx]
            basic_idx[leaving_idx] = nonbasic_idx[entering_idx]
            nonbasic_idx[entering_idx] = tmp
```

Source Code B.1: Python simplex algorithm

```
def northWestCornerMethod(supply, demand, matrix):
    supply = np.array(supply)
    demand = np.array(demand)
    for r in range(0, len(supply)):
        for c in range(0, len(demand)):
            quantity = min(supply[r], demand[c])
            if quantity > 0:
                matrix[r][c] = quantity
                supply[r] -= quantity;
                demand[c] -= quantity;
            if supply[r] == 0:
                r += 1
                break;

def leastCostMethod(supply, demand, costs, matrix):
    supply = np.array(supply)
    demand = np.array(demand)
    costs = np.array(costs)
    cells = []
    removedSupply = set()
    removedDemand = set()
    for r in range(0, len(supply)):
        for c in range(0, len(demand)):
            cells.append((r, c));
    cellsSortedByCost = sorted(cells, key=lambda (a, b): costs[
    a][b])
    for cell in cellsSortedByCost:
        r = cell[0]
        c = cell[1]
        if supply[r] > 0 and demand[c] > 0:
            quantity = min(supply[r], demand[c])
            if quantity > 0:
                matrix[r][c] = quantity
                supply[r] -= quantity;
                demand[c] -= quantity;

def getAdjacentCells(cell, cells):
    adjacentCells = [None, None]
    for c in cells:
        if c != cell:
            if adjacentCells[0] is None and c[0] == cell[0]:
                adjacentCells[0] = c
            elif adjacentCells[1] is None and c[1] == cell[1]:
                adjacentCells[1] = c
            if adjacentCells[0] is not None and adjacentCells
    [1] is not None:
                break
    return adjacentCells

def removeCellsWithNoAdjacentCells(cells):
    removed = False
    for cell in cells:
        adjacentCells = getAdjacentCells(cell, cells)
        if adjacentCells[0] == None or adjacentCells[1] == None
```

```
    :
            cells.remove(cell)
            removed = True
    return removed

def getClosedPath(cell, cells):
    path = [cell]
    path.extend(cells)
    while True:
        result = removeCellsWithNoAdjacentCells(path)
        if result == False:
            break
    stones = []
    prev = cell
    for i in range(len(path)):
        stones.append(prev)
        prev = getAdjacentCells(prev, path)[i % 2]
    return stones

def fixDegeneracy(supply, demand, matrix, cells):
    epsilon = 0.00000000000000000000001
    for r in range(0, len(supply)):
        for c in range(0, len(demand)):
            if matrix[r][c] == 0:
                path = getClosedPath((r, c), cells)
                if len(path) == 0:
                    matrix[r][c] = epsilon
                    return

def steppingStoneMethod(matrix):
    steppingStones = []
    for r in range(0, len(supply)):
        for c in range(0, len(demand)):
            if matrix[r][c] != 0:
                steppingStones.append((r, c))
    if (len(supply) + len(demand) - 1) != len(steppingStones):
        fixDegeneracy(steppingStones)

    maxReducedCost = 0;
    finalPath = None
    leavingVariable = None
    for r in range(0, len(supply)):
        for c in range(0, len(demand)):
            if matrix[r][c] != 0:
                continue

            path = getClosedPath((r, c), steppingStones)
            reducedCost = 0
            minQuantity = sys.maxsize
            leavingCandidate = None
            sign = 1;
            for p in path:
                change = sign * (costs[p[0]][p[1]])
                reducedCost = reducedCost + change
```

```
                if sign == -1:
                    if matrix[p[0]][p[1]] < minQuantity:
                        minQuantity = matrix[p[0]][p[1]];
                        leavingCandidate = p
                sign = sign * -1;

            if reducedCost < maxReducedCost:
                finalPath = path
                leavingVariable = leavingCandidate
                maxReducedCost = reducedCost

    if leavingVariable is not None:
        quantity = matrix[leavingVariable[0]][leavingVariable
    [1]]
        sign = 1
        for p in finalPath:
            matrix[p[0]][p[1]] = matrix[p[0]][p[1]] + (sign *
    quantity)
            sign = sign * -1;

        steppingStoneMethod(matrix)

def modifiedDistributionMethod(supply, demand, costs, matrix):
    supply = np.array(supply)
    demand = np.array(demand)
    costs = np.array(costs)
    while True:
        stoneCells = []
        for r in range(0, len(supply)):
            for c in range(0, len(demand)):
                if matrix[r][c] != 0:
                    stoneCells.append((r, c))

        if (len(supply) + len(demand) - 1) != len(stoneCells):
            fixDegeneracy(supply, demand, matrix, stoneCells)

        uv = []
        stoneCosts = []
        for r in range(0, len(supply)):
            for c in range(0, len(demand)):
                if matrix[r][c] != 0:
                    if len(uv) == 0:
                        u = [0] * len(supply)
                        v = [0] * len(demand)
                        u[r] = 0
                        v[c] = 1
                        u.extend(v)
                        uv.append(u)
                        stoneCosts.append(costs[r][c])

                    u = [0] * len(supply)
                    v = [0] * len(demand)
                    u[r] = 1
                    v[c] = 1
```

```
                    u.extend(v)
                    uv.append(u)
                    stoneCosts.append(costs[r][c])

        a = np.array(uv)
        b = np.array(stoneCosts)
        x = np.linalg.solve(a, b)
        minReducedCost = None
        enteringVariable = None
        for r in range(0, len(supply)):
            for c in range(0, len(demand)):
                if matrix[r][c] == 0:
                    c_ij = costs[r][c]
                    u_i = x[r]
                    v_j = x[len(supply) + c]
                    reducedCost = c_ij - (u_i + v_j)
                    if minReducedCost is None or reducedCost <
    minReducedCost:
                        minReducedCost = reducedCost
                        enteringVariable = (r, c)

        if minReducedCost < 0:
            path = getClosedPath(enteringVariable, stoneCells)
            minQuantity = None
            plus = True
            for cell in path:
                r = cell[0]
                c = cell[1]
                quantity = matrix[r][c]
                if not plus and (minQuantity == None or
    quantity < minQuantity):
                    minQuantity = quantity
                plus = not plus

            plus = True
            for cell in path:
                r = cell[0]
                c = cell[1]
                if plus:
                    matrix[r][c] = matrix[r][c] + minQuantity
                else:
                    matrix[r][c] = matrix[r][c] - minQuantity
                plus = not plus

        else:
            break
```

Source Code B.2: Python transportation algorithms

```
dvar int x11;
dvar int x12;
dvar int x13;
dvar int x21;
dvar int x22;
dvar int x23;
dvar int x31;
dvar int x32;
dvar int x33;

minimize 3 * x11 + 7 * x12 + 15 * x13 + 0 * x21 + 8 * x22 + 6 *
     x23 + 10 * x31 + 2 * x32 + 5 * x33;

subject to {
 x11 + x12 + x13 == 5;
 x21 + x22 + x23 == 20;
 x31 + x32 + x33 == 25;
 x11 + x21 + x31 == 15;
 x12 + x22 + x32 == 25;
 x13 + x23 + x33 == 10;
 x11 >= 0;
 x12 >= 0;
 x13 >= 0;
 x21 >= 0;
 x22 >= 0;
 x23 >= 0;
 x31 >= 0;
 x32 >= 0;
 x33 >= 0;
}
```

Source Code B.3: OPL transportation example

```
tuple AssetTuple {
  string id;
  string name;
  float marketValuePerUnit;
  float availableUnits;
}

tuple RequirementTuple {
  string id;
  string name;
  float requiredAmount;
}

{AssetTuple} Assets = ...;
{RequirementTuple} Requirements = ...;

float Costs[Assets][Requirements] = ...;
int Eligibility[Assets][Requirements] = ...;

dvar int+ x[Assets][Requirements];

minimize
  sum(a in Assets, r in Requirements)
    x[a][r] * Costs[a,r];

subject to
{
  AvailabilityConstraints:
  forall (a in Assets)
    sum(r in Requirements) x[a][r] <= a.availableUnits;

  RequirementConstraints:
  forall(r in Requirements)
    sum(a in Assets) (x[a][r] * a.marketValuePerUnit *
    Eligibility[a,r]) >= r.requiredAmount;
}

execute DISPLAY {
  for(var a in Assets) {
    for(var r in Requirements) {
        writeln(a.name," ",r.name," ",x[a][r]," ",x[a][r] * a.
    marketValuePerUnit);
    }
  }
};
```

Source Code B.4: OPL collateral optimisation basic example

```
Assets = {
<A1,"ZAR",100,1000>,
<A2,"R186",500,500>,
<A3,"ES23",200,0>,
<A4,"TL20",115,0>,
<A5,"TKG",110,0>,
<A6,"AGL",150,2000>,
};

Requirements = {
<R1,"Call1",100000>,
<R2,"Call2",200000>,
};

Costs = [
[5,5],
[3,3],
[2,2],
[2,2],
[1,1],
[1,1],
];

Eligibility = [
[1,1],
[1,1],
[1,1],
[1,1],
[1,1],
[0,0],
];
```

Source Code B.5: OPL collateral optimisation basic example data