NTNU – Trondheim
Norwegian University of
Science and Technology

Exact Optimization Methods for the Mixed Capacitated General Routing Problem

Kevin Anders Gaze

Master of Science in Physics and Mathematics
Submission date: July 2013
Supervisor: Jo Eidsvik, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

ABSTRACT

This thesis is about using exact optimization algorithms to solve the routing problem known as the Mixed Capacitated General Routing Problem (MCGRP) that is a generalization of many other well known routing problems. The goal is to find an optimal set of routes servicing a set of required entities on a mixed network. The entities being vertices, directed arcs and undirected edges.

The mathematical programming model formulation developed in this thesis is a novel path flow formulation inspired by a formulation for another well known routing problem by Letchford and Oukil (2009). The solution method is based on the exact optimization techniques Column Generation (CG) and Branch & Price (B&P).

The algorithm is implemented in the programming language C# with the help of the BCL XPRESS libraries. A comparison has been given to the results by an exact algorithm by Bosco et al. (2012) as well as the currently best results known in the literature.

The algorithm is tested on 158 benchmark instances, were 83 of them where solved to optimum and 16 for the very first time. The algorithm is in addition an excellent lower bounding algorithm giving 58 improved lower bounds for previously unsolved instances. There is still a lot of research that can be done on the MCGRP and the hope is that this thesis will motivate to further research.

SAMMENDRAG

Denne masteroppgaven handler om å bruke eksakte optimaliserings-algoritmer til å løse rute-optimaliserings-problemet kjent som "Mixed Capacitated General Routing Problem". Dette problemet er en generalisering av mange andre kjente rute-optimaliserings-problemer.

Målet er å finne et optimalt set av ruter som kan betjene en mengde pålagte enheter i et blandet nettverk. Enhetene er punkter, rettede buer og urettede kanter.

Den matematiske formuleringen som blir utviklet i denne masteroppgaven er en ny flytformulering inspirert av en formulering til et annet velkjent rute-optimaliserings-problem av Letchford og Oukil (2009). Løsningsmetoden er basert på de eksakte optimaliserings-teknikkene, kolonnegenerering og "Branch & Price".

Algoritmen er implementert i programmeringsspråket C# ved hjelp av BCL-XPRESS-bibliotekene. Resultatene sammenlignes med den nøyaktige algoritmen til Bosco et al. (2012) og de, for tiden, beste resultatene kjent fra litteraturen.

Algoritmen blir testet på 158 benchmark-tilfeller, hvorav 83 av dem ga optimal løsning. 16 av dem ga optimal løsning for aller første gang. Algoritmen er i tillegg en utmerket nedre-grense-algoritme som gir forbedrede nedre grenser for 58 tilfeller som tidligere har vært uløst. Det er fortsatt mye forskning som kan gjøres på dette problem og håpet er at denne masteroppgaven vil motivere til videre forskning.

ACKNOWLEDGEMENTS

The master's thesis was written in cooperation with SINTEF, and is my completion of a Master's degree in Applied Mathematics at the Norwegian University of Science and Technology.

I would first and foremost like to thank my supervisors, Dr. Geir Hasle, Dr. Carlo Mannino, Dr. Elin Halvorsen-Weare and Dr. Jo Eidsvik. Especially, I would like to thank my main supervisor Geir for giving me the opportunity to write my thesis at SINTEF and presenting my work at the WARP1 conference in Copenhagen.

Finally, I would like to thank my good friend, Camilla. I would not be here today if it were not for you.

Kevin Gaze Oslo, July 2013

CONTENTS

LIST OF ABBREVIATIONS

BHW	Benchmark Instances for the MCGRP
BIP	Binary Integer Program
B&B	Branch & Bound
B&C	Branch & Cut
B&P	Branch & Price
CARP	Capacitated Arc Routing Problem
CBMix	Benchmark Instances for the MCGRP
CG	Column Generation
CPP	Chinese Postman Problem
CVRP	Capacitated Vehicle Routing Problem
DINEARP	Benchmark Instances for the MCGRP
DW	Dantzig-Wolfe
ESPPRC	Elementary Shortest Path Problem with Resource Constraints
IP	Integer Program
LB	Lower Bound
LP	Linear Program
MCGRP	Mixed Capacitated General Routing Problem
MGGDB	Benchmark Instances for the MCGRP
MGVAL	Benchmark Instances for the MCGRP
MILP	Mixed Integer Linear Program
MP	Master Problem
RMP	Restricted Master Problem
RPP	Rural Postman Problem
SPPRC	Shortest Path Problem with Resource Constraints

TSP	Travelling Salesman Problem
UB	Upper Bound

CHAPTER 1

INTRODUCTION

Traditionally, the routing literature has been devoted to studying the two families of problems: *node routing problems* and *arc routing problems* depending on the entities that need to be serviced in the problem. The Capacitated Vehicle Routing Problem or CVRP is the most common representation of the node routing problems. The CVRP is usually defined on an undirected network where some of the nodes correspond to customers. Each customer has a demand or weight for commodity. The edges in the network have travel costs. A fleet of homogeneous vehicles, located at a depot are used to service the nodes. Each vehicle has a maximum capacity that limits the amount of total demand or commodity it can service. A trip for a vehicle starts at the depot, visits a set of customers and returns to the depot. The cost of this trip is the total edge distance travelled by the tour. The CVRP consists of designing a set of tours of least cost, while all the customers are serviced. The CVRP has many important applications, for instance logistics, and has a long research history behind it.

Arc Routing has been much less investigated in the literature, but there has been growing interest the last two decades. This is mainly because of their applications such as snow ploughing and salt spreading in winter. The corresponding arc routing variation of the CVRP is the CARP or Capacitated Arc Routing Problem defined by Golden and Wong (1980). The definition of the CARP is very similar to the CVRP, but for the CARP a set of edges needs to be serviced by the vehicles, rather than nodes.

The focus of this thesis is on the MCGRP or the Mixed Capacitated General Routing Problem that is a combination of the VRP and the CARP. The MC-

GRP is a problem with required entities that include nodes, directed arcs and undirected edges.

1.1 Motivation

The MCGRP was defined by Pandit and Muralidharan (1995) and the problem has only been investigated a handful of times since then. Despite the great success of many algorithms for solving the CVRP and the CARP the two problems cannot formulate the requirements of many real world scenarios. Consider for instance urban waste collection. Most of the tasks can be considered as servicing streets, but there are some punctual accumulations of waste that cannot be considered as anything else than nodes, for instance at schools and hospitals.

The CVRP, CARP and the MCGRP are all NP-hard problems, which introductory means that big instances are very hard to solve exactly. The biggest problem instances we can solve to optimality today are in the region of a few hundred customers. For this reason heuristic algorithms have become very popular. These are methods that can quickly get satisfactory solutions, even for big problem instances, but cannot guarantee optimal solutions. Until Bosco et al. (2012) there had not been defined a mathematical model formulation for the MCGRP and no exact methods had been implemented. The motivation for this thesis is to develop a new and hopefully better mathematical model formulation. We will be investigating a solution method based on the Branch & Price (B&P) algorithm for the MCGRP and compare it to the previous exact method, based on a Branch & Cut (B&C) algorithm, by Bosco et al. (2012).

1.2 The Contribution of this thesis

In this thesis a new Mathematical Programming model formulation for the MCGRP is created. The model is solved by an exact optimization algorithm, the B&P algorithm. To the best of our knowledge, this thesis is the second time the MCGRP has been solved by exact optimization techniques and the first time it has been tackled by a B&P algorithm. The algorithm has been used to solve several benchmark instances of the MCGRP. A total of 158 instances has been solved. The results will be compared to results given by Bosco et al. (2012) as well as the currently best results from the literature.

1.3 The Structure of this thesis

Chapter 2 introduces the field of optimization and necessary background material on Mathematical Programming is gained. In Chapter 3 we go deeper into the theory and explain the methods of Branch & Bound (B&B), B&C and B&P as well as a short description of heuristics.

In Chapter 4 the three types of routing problems are described: CVRP, CARP and MCGRP. Previous work in the literature on the MCGRP is presented.

Two mathematical programming model formulations for the MCGRP are presented in Chapter 5: The *arc flow* formulation created by Bosco et al. (2012) is presented and a new *path flow* formulation. Chapter 6 is the main contribution of this thesis and explains how to solve the path flow formulation by a B&P algorithm.

The computational experiments are described and a discussion of the results are presented in Chapter 7. Finally a conclusion and some ideas for further work is given in Chapter 8.

The appendix includes the abstract from WARP1 (1st Workshop on Arc Routing Problems) in Copenhagen 22-24 May 2013 where the author of this thesis was invited to present the new algorithm (WARP1,2013).

CHAPTER 2

OPTIMIZATION AND MATHEMATICAL PROGRAMMING

The optimization field belongs to the field of applied mathematics where mathematical models are used in decision making situations to find the best alternative. Optimization models are often used in economic systems where the goal is either to minimize costs or maximize profits. The word *optimum* comes from the Latin word "optimus" which means "best, very good". We will mostly limit the scope of this thesis to discrete optimization. Discrete optimization is the science of making the best decision or making the best possible decision when working with discrete decision variables, see e.g. Lundgren et al. (2010).

When using an optimization model we have a set of *decision variables* that are controlled by the *decision maker*. The decision maker is given an *objective function* that depends on these decision variables. The problem also includes a set a constraints that restrict the variables in one way or another. A simple example of this could be a factory producing toys. Each toy will have a decision variable that decides how many toys of its kind to produce. The objective function includes the income that can be gained from selling the toys, while the constraints are the restrictions set on producing the toys that could be production costs, time limitations etc. Problems as these are often modelled as *Mathematical Programs.*

The rest of this chapter will be organized as follows. We start by introducing the mathematical program in chapter 2.1. Computational complexity and a discussion about **NP-hard** problems will be given in chapters 2.2 and 2.3 respectively. In chapter 2.4 an introduction to solving Mathematical Programs will be given.

2.1 What is a Mathematical Program?

A general mathematical program (P) can be represented by the equations (2.1)-(2.2).

$$(P) \quad max \quad f(\mathbf{x}) \tag{2.1}$$

$$s.t. \quad \mathbf{x} \in X. \tag{2.2}$$

Were $f(\mathbf{x})$ is the *objective function* that depends on the *decision variables* $\mathbf{x} = (x_1 ... x_n)^T$. The set X defines the feasible solutions to the problems. The goal when solving a mathematical program is to either maximize or minimize $f(\mathbf{x})$ given $\mathbf{x} \in X$. This problem could for instance represent the famous knapsack problem. The goal in the knapsack problem is to fill your knapsack with valuable items such that the total value of the knapsack is maximized. Each valuable has a specific weight or size and value associated with it . The objective function will contain information about the respective values and X would represent information about weights of the items and restrictions for the knapsack. Usually X is expressed by a set of functional constraints so an alternative formulation to (P) could be described by equation (2.3)-(2.4).

$$max \quad f(\mathbf{x}) \tag{2.3}$$

$$s.t. \quad g_i(\mathbf{x}) \leq b_i, \quad i = 1, ..., m, \quad \mathbf{x} \in \mathbb{R}^n \tag{2.4}$$

Here m would represent the number of constraints. Another specification that will be used in this paper is that the problem is linear. This is called a Linear Program (LP). This means that both functions f and g must be linear. An LP can be formulated as Equation (2.5)-(2.7)

$$max \quad z = \sum_{j=1}^{n} c_j x_j \tag{2.5}$$

$$s.t. \quad \sum_{j=1}^{n} a_j x_j \leq b \tag{2.6}$$

$$x_j \geq 0 \quad j = 1, ..., n \tag{2.7}$$

In this formulation the $c_j's$, a_j's and b are all known constants. They are input constants for a given problem, say our knapsack problem. This would mean that c_j represents the profit per unit amount x_j of an item j. a_j can represent the weight of each element and b can represent the total capacity of the knapsack. When the variables are continuous the problem is trivial. Say the variables are gold, silver and bronze dust. A simple greedy algorithm chooses the most profitable available dust and fills up the knapsack to the rim. Most problems have decision variables that represent real assets and since you cannot have negative amounts of assets in your knapsack it is common to include non negativity constraints, as equation (2.7).

The problems we will focus on in this thesis are of this linear form and knowing this we can use algorithms specifically designed for solving LPs. There are many cases were reality will fit better to a quadratic, or even higher ordered program, but most commercial optimization software requires that the problem is an LP. Therefore when using higher ordered programs we need to linearise, for example by interpolating, see Nocedal et al. (2009). This will not be described further because it is not used in this thesis.

We will also define our variables as integer variables. In most problems we work with discrete decisions. We can not do half a job, visit a customer 1.5 times or put a fraction of an item in the knapsack. Therefore we impose that our decision variables be integer $x_j \in \{0, 1, 2...\}$ or binary $x_j \in \{0, 1\}$. The LP is now called an Integer LP or a Binary Integer LP (IP or BIP). If, on the other hand, we include both integer variables and continuous variables we get a Mixed Integer LP (MILP). As stated earlier these problems can either be minimization or maximization problems. The constraints can either be less than ($\leq$), more than ($\geq$) or equality ($=$) constraints. The binary knapsack problem will now take the form:

$$max \quad z = \sum_{j=1}^{n} c_j \; x_j \tag{2.8}$$

$$s.t. \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \qquad i = 1, ..., m \tag{2.9}$$

$$x_j \in \{0, 1\} \quad j = 1, ..., n \tag{2.10}$$

The problem is then to find the best combination of x_j's to include. Constraint (2.10) is called a binary constraint that means the variable can only take the values 1 or 0. By using a greedy method here we might not find the optimal solution because a combination of less valuable items might be more profitable than always picking the most valuable item available. We will see how to solve these problems in chapters 2.4 and 3.

2.2 Computational Complexity

Let us now have a look at the computational complexity of the types of problems we will solve in this thesis. A famous routing problem is known as the travelling salesman problem (TSP). In the TSP we have n customers that we need to visit. We start at an initial depot, visit all the customers, and we return to the same place.

It is easy to demonstrate just how difficult the TSP can be. Imagine a vehicle that has to deliver to 3 different locations A, B and C. The goal is to decide which order the vehicle should visit each location to minimise the overall travel distance.

There are 6 possibilities:

A-B-C

A-C-B

B-A-C

B-C-A

C-A-B

C-B-A

So, the simplistic approach is to consider all 6 cases, work out the distance travelled for each one and choose the shortest. Actually often only 3 of the cases need to be considered because the distance A-B-C is likely to be the same as the distance C-B-A, unless one-way streets are involved. This simple problem would take a modern computer almost no time to solve. However the difficulty increases surprisingly quickly as the number of deliveries increases:

4 locations have 24 possible solutions

5 locations have 120 possible solutions

6 locations have 720 possible solutions

...

n locations have n x (n-1) x (n-2) x .... 3 x 2 x 1 = n! solutions.

This is known as a factorial dependence. For a parcel delivery van which might make 80-100 deliveries in a day the number of possible routes/sequences is extremely large as can be seen by the Table 2.1. Taking into account that there are approximately ”only” 10^{80} atoms in the observable universe, the simplistic approach of trying all possible combinations would therefore take ”longer than the lifetime of the universe” to come up with an answer. There are however methods to solve this quicker by only checking a few of the feasible tours, but as will be discussed in the next section, there is still no guaranteed algorithm that can find optimal solutions for all problem instances of this form in polynomial time. This is where the discussion of problem sets comes into play. Most routing problems have factorial dependence and are known as **NP-hard** problems.

Table 2.1: This table shows how polynomial dependencies compare to exponential and factorial dependencies. As one can see, polynomial dependencies are much more preferable.

n	n^2	n^{10}	2^n	$n!$
10	10^2	10^{10}	1.02×10^3	3.6×10^6
100	10^4	10^{20}	1.27×10^{30}	9.33×10^{157}
1000	10^6	10^{30}	1.07×10^{301}	4.02×10^{2567}

To describe what is meant by **NP-hard** some information about the more basic problem sets **P** and **NP** need to be explained.

2.3 P vs NP

The problem set **P** includes all *decision* problems that can be solved in polynomial time. Decision problems are problems that can be answered with either ”yes” or ”no”. This means that for a given input size n and for some constant p there is at most n^p deterministic operations to solve the problem, see Table 2.1. Lundgren et al. (2010) describe these as ”easy” problems.

The problem set **NP** (non deterministic polynomial) is the set of all problems where a solution can be checked in polynomial time. A **NP** problem does not require a solution to be found in polynomial time so it can be exponential or even factorial, but the solution must be *checkable* in polynomial time. All problems in **NP** are decision problems.

The **P** vs **NP** is one of the famous Millennium Prize Problems in mathematics. The problem is to show that either $\mathbf{P} = \mathbf{NP}$ or $\mathbf{P} \neq \mathbf{NP}$. We know that $\mathbf{P} \subseteq \mathbf{NP}$, but can not say more than this, see Figure 2.1, Cormen et al. (2009)

The next set of problems to be explained are the **NPC** (**NP-complete**) problems. These problems are defined to include all the problems that are at least as difficult to solve as all the other **NP** problems.

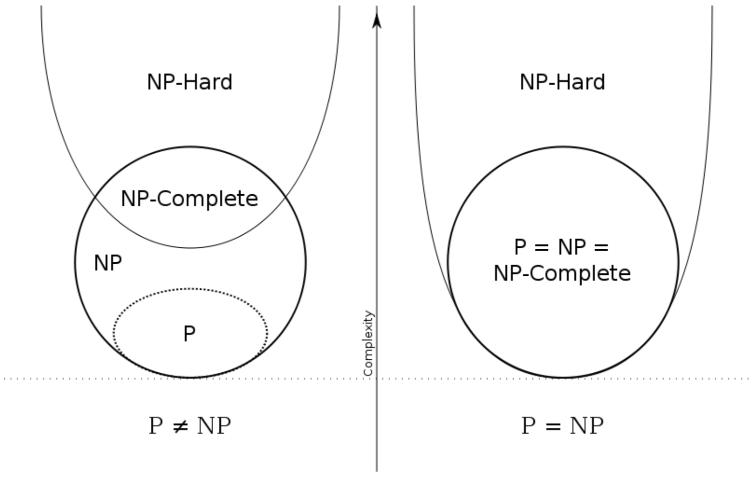


Figure 2.1: P vs **NP**. The left image represents the fact that there exists problems that can be checked in polynomial time, but cannot be optimally solved in polynomial time. The image to the right represents that all problems that can be checked in polynomial time can also be solved in polynomial time. It is still not known which image is correct.

Finally we have the **NP-hard** problems. These problems include the **NPC**, but do not require the problem to be a decision problem. Hence the solution does not need to be checked in polynomial time see Figure 2.1. It is in the **NP-hard** problems we find the TSP, CVRP, CARP and the MCGRP. This is the first step to understanding how complex these problems are.

2.4 How to solve a Mathematical Program

When solving a Mathematical Program, an LP in our case, we find the best feasible solution for our problem. The feasible region is the area of all possible solutions given the set of restrictions. If one restriction is violated, we get an infeasible solution. We may have one, none or infinitely many solutions. An LP can easily be visualized when we have 2 decision variables. Figure 2.2 shows this for the LP from Equation (2.11)-(2.14).

$$max \quad z = x_1 + x_2 \tag{2.11}$$
$$x_1 - x_2 \leq 3 \tag{2.12}$$
$$x_1 + 2x_2 \leq 6 \tag{2.13}$$
$$x_1, x_2 \geq 0 \tag{2.14}$$

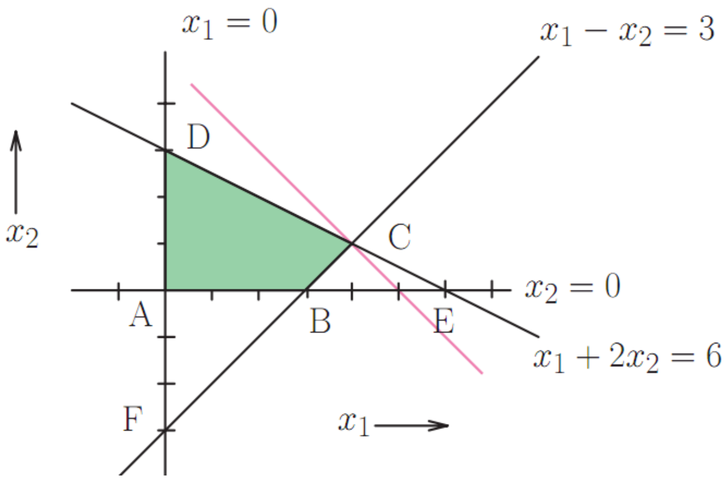


Figure 2.2: A visualization of the LP from equation (2.11)-(2.14). The green area indicates all the feasible solutions. Black lines indicate the constraints for the given problem. The red line represents a line were the objective value is constant. The value of the objective function is increased by moving the line to the right. The optimal value is reached at the position shown.

The feasible region in an LP is always a convex region. In Euclidean space, an object is convex if for every pair of points within the region, every point on the straight line segment that joins them is also within the region, see Figure 2.3. A unique optimal point is always on an extreme point. An extreme point is a point in a set that cannot be represented as a strict combination of two other points in the set. They are known as vertices of the set. If the optimal point is on an edge we get infinitely many optimal points, hence it is not unique.

The first property is obvious because of the linearity of the constraints. The second property can be seen to at least work from Figure 2.2, but we will prove this.

Proof. We assume a unique optimal solution and that this solution is not an extreme point. This implies that there must be two other feasible solutions such

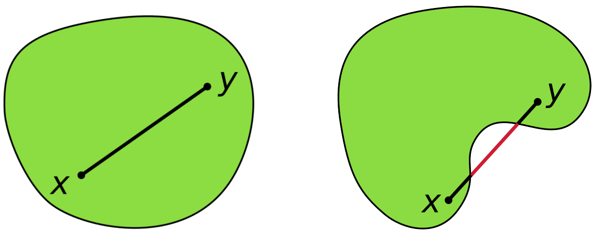


Figure 2.3: Convex Set to the left. Non-Convex Set to the right.

that the line segment connecting them contains the optimal solution. We call these two solutions $\mathbf{x}_1$ and $\mathbf{x}_2$ with objective value z_1 and z_2 respectively. For all these points on the line segment between $\mathbf{x}_1$ and $\mathbf{x}_2$ we must have that

$$\mathbf{x}^* = \alpha\mathbf{x}_1 + (1-\alpha)\mathbf{x}_2, \tag{2.15}$$

where α is a proportion constant, $0 \le \alpha \le 1$. Because we are looking at a linear function, we know that the objective function for x' also has the form.

$$z^* = \alpha z_1 + (1-\alpha)z_2. \tag{2.16}$$

because α and $1-\alpha$ adds to 1, it means that the relationship between the three objective function values has to be one of the three,

$$z^* = z_1 = z_2 \tag{2.17}$$
$$z_1 \le z^* \le z_2 \tag{2.18}$$
$$z_2 \le z^* \le z_1. \tag{2.19}$$

If 2.17 is true, we have more than one optimal solution which is a contradiction to the initial statement that we have a unique optimal solution. If either 2.18 or 2.19 is true, z^* is not optimal, which is also a contradiction. This results in that the optimal solution must be an extreme point.

$\square$

For an LP with n decision variables, each of the extreme points lie in the intersection of n constrained boundaries. This is easy to visualize for two and even three dimensions. The two dimension case shown in Figure 2.2 would have the following extreme points: *(0,0), (0,3), (4,1) and (3,0).*

The most famous solution method for solving LPs is the Simplex Method. Almost all optimization software packages for solving LPs use this method, see Lundgren et al. (2010). The method was developed by Dantzig (1947) and is an extremely efficient algorithm that is used routinely to solve huge problems. There are many exact optimization methods that are used as will be discussed in Chapter 3.

CHAPTER 3

OPTIMIZATION METHODS FOR INTEGER PROGRAMMING

In this Chapter we will go through the theory most commonly used to solve IPs. The *Simplex method* is used for most LP optimization problems and the *B&B* algorithm is most commonly used when the problem is integer. For bigger problems more complex methods are common. *B&C* is described in chapter 3.3 while Column Generation (CG) and *B&P* will be explained in chapters 3.4 and 3.5 respectively. Chapter 3.6 is a short description of heuristics referred to in this thesis.

3.1 The Simplex Method

In this section we will describe mathematically how the Simplex method solves an LP. The LP from (2.5)-(2.7) can be written on the matrix form:

$$\begin{aligned} max \quad z = \mathbf{c}^T\mathbf{x} & \qquad (3.1)\\ s.t. \quad \mathbf{A}\mathbf{x} \leq \mathbf{b} & \qquad (3.2)\\ \mathbf{x} \geq 0 & \qquad (3.3) \end{aligned}$$

First we convert the LP to standard form by adding new non zero variables to each row in the **A** matrix from Equation (3.2) which becomes,

$$\mathbf{A}\mathbf{x} = \mathbf{b}. \tag{3.4}$$

The newly added variables are called *slack variables* and measure the variation between the right and left side in each constraint.

For the standard form,

n is called the dimension,

m is called the order,

variables $\mathbf{x}$ satisfying equation (3.4) are called feasible solutions.

Suppose rank($\mathbf{A}$) $= m$, and the first m columns of $\mathbf{A}$, $\mathbf{A}_i = (\mathbf{A}_{1,i}....\mathbf{A}_{mi})$, are linearly independent then $\mathbf{B} = (\mathbf{A}_1, .., \mathbf{A}_m)$ is non-singular. Call $\mathbf{B}$ a basis matrix. The linear system $\mathbf{B}\mathbf{x}_b = \mathbf{b}$ has a unique solution $\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b}$. Define $\mathbf{x} = (\mathbf{x}_B, 0)$, then $\mathbf{x}$ satisfies equation (3.4). $\mathbf{x}$ is called a basic solution and basic feasible solution if it is feasible. The total number of basic feasible solutions is given by the combinatorial formula:

$$C_n^m = \frac{n!}{m!(n-m)!}$$

I.e. the number of possible ways m variables can be chosen from n variables. For an LP with $n = 150$ and $m = 50$ (in practice a rather small problem) the theoretical number of basic solutions exceeds 2×10^{40}.

The Simplex method systematically searches through the basic solutions by using the fact that the optimal solution will be on an extreme point. Lundgren et al. (2010) describe a 5 stage general algorithm for the Simplex:

- **Step 0** Start from a basic solution $\mathbf{x}^{(0)}$. Let $k = 0$
- **Step 1** Determine the search direction and determine the reduced costs of adding the variable needed. The reduced cost $\overline{\pi_j}$ is the cost of including the variable as basic.
- **Step 2** Check convergence criterion: The point $x^{(k)}$ is an optimal solution if

$$\overline{\pi_j} \geq 0, \forall j \text{ (minimization problem)}$$
$$\overline{\pi_j} \leq 0, \forall j \text{ (maximization problem)}$$

  i.e. for a minimization problem if we include any variable the objective will increase, hence we have optimum.

- **Step 3** Determine entering variable according to these criteria

$$\overline{\pi_p} = \min_j \{\overline{\pi_j} | \overline{\pi_j}\} \leq 0 \text{ (minimization problem)}$$
$$\overline{\pi_p} = \max_j \{\overline{\pi_j} | \overline{\pi_j}\} \geq 0 \text{ (maximization problem)}$$

  which gives the entering basic variable x_p and a search direction $\mathbf{d}^{(k)}$.

- **Step 4** Determine step length

$$t^{(k)} = \frac{x_r^{(k)}}{-d_r^{(k)}} = \min_j \{\frac{x_j^{(k)}}{-d_j^{(k)}} | d_j^{(k)} \leq 0\}$$

  which means x_r becomes the leaving basic variable. The step length will be the length to the adjacent extreme point.

- **Step 5** The new point is $\mathbf{x}^{(k+1)} + \mathbf{x}^{(k)} + t^{(k)}\mathbf{d}^{(k)}$ and the new basic solution x_r is replaced with x_p. Update $k := k + 1$ and go to Step 1.

Example 4.1

The Simplex method will be explained by solving the LP from (2.11)-(2.14), see Table 3.1. The standard form is given by equations (3.5) -(3.8).

$$max \quad z = x_1 + x_2 \tag{3.5}$$
$$x_1 - x_2 + x_3 = 3 \tag{3.6}$$
$$x_1 + 2x_2 + x_4 = 6 \tag{3.7}$$
$$x_i \geq 0, i = 1, 2, 3, 4 \tag{3.8}$$

As described before, an optimal solution can always be found on a vertex. The Simplex method uses this fact and it searches for a solution by going from one vertex to the next in the feasible region. Even if the solution is not unique, and the optimal solutions lie on an edge, we can still do the same because it is sufficient to find one of the vertices that lie on the edge. When tackling an IP with Simplex, the solution will often be fractional which is obviously not feasible. We cannot simply round off to closest natural number even though this is tempting. This might not give the correct answer. To solve IP the B&B algorithm is most commonly used.

Table 3.1: Solution to an LP using the Simplex method.

Iteration 0	Step 0:	Start by choosing (0,0) as initial solution. $\mathbf{x}^{(0)} = (0,0,3,6)^T$, $\mathbf{B} = (\mathbf{A}_3, \mathbf{A}_4)$ $z = 0$.
Iteration 0	Step 1:	From Figure 2.2 we know that the search directions are $(1,0)^T$ and $(0,1)^T$. $\overline{\pi_1} = \overline{\pi_2} = 1$
	Step 2:	$\overline{\pi_1}, \overline{\pi_1} \geq 0$
	Step 3:	The search directions give equal increase in objective value so we choose $\overline{\pi_1}$ arbitrarily. Entering variable x_1
	Step 4:	$d_2^{(0)} = 0$, $d_3^{(0)} = -1$, $d_4^{(0)} = -1$ which means that by increasing x_1 by 1 we decrease x_3 and x_4 by 1. $t^{(0)} = \frac{x_r^{(0)}}{-d_r^{(0)}} = \min\{\frac{x_3^{(0)}}{-d_3^{(0)}}, \frac{x_4^{(0)}}{-d_4^{(0)}}\} = \{\frac{3}{1}, \frac{6}{1}\} = \frac{x_3^{(0)}}{-d_3^{(0)}} = 3$. Leaving variable is x_3.
	Step 5:	$\mathbf{x}^{(1)} = (0,0,3,6)^T + 3 * (1,0,-1,-1)^T = (3,0,0,3)$, $\mathbf{B} = (\mathbf{A}_1, \mathbf{A}_4)$ $z = 3$.
Iteration 1	Step 1:	From Figure 2.2 we know that the search directions are $(-1,0)^T$ and $(1,1)^T$. $\overline{\pi_3} = -1$ $\overline{\pi_2} = \sqrt{2}$
	Step 2:	$\overline{\pi_2} \geq 0$
	Step 3:	Only search direction $(\sqrt{2}, \sqrt{2}, 0, -\sqrt{5})^T$ gives increase in objective value. Entering variable x_2
	Step 4:	$t^{(1)} = \frac{x_r^{(1)}}{-d_r^{(1)}} = \{\frac{3}{\sqrt{2}}\} = \frac{x_4^{(1)}}{-d_4^{(1)}} = \frac{3\sqrt{2}}{2}$. Leaving variable is x_4.
	Step 5:	$x^{(2)} = (3,0,0,3)^T + \frac{3\sqrt{2}}{2} * (\sqrt{2}, \sqrt{2}, 0, -\sqrt{2})^T = (4,1,0,0)$, $B = (a_1, a_2)$ $z = 5$.
Iteration 2	Step 1:	The search directions are, $(-1,-1)^T$ and $(-1,-2)^T$.
	Step 2:	No better solution can be found. We have optimum.

3.2 Branch & Bound

Solving an IP can be a lot more complicated than one might think. The B&B algorithm is most commonly used to tackle IPs. The idea is to split the feasible region into smaller regions and optimize these sub regions. In each area we solve the LP relaxed problem to optimality generating new *optimistic bounds*, i.e. lower bounds for minimization problems. Whenever an integer solution is found we get a candidate for the *pessimistic bounds*, i.e. upper bounds for a minimization problem. We continue until the gap between the bounds shrink to zero. Lundgren et al. (2010), page 393) states:

The term Branch and Bound comes from the fact that the feasible region is divided into several smaller areas (branching) and the fact that we use optimal solution for each area for bounding the optimal objective function value.

We will explain this with an example (IP).

$$\begin{aligned}(IP)\quad & max \quad z = 4x_1 + 5x_2\\ & x_1 + 3x_2 \leq 12\\ & 4x_1 + 3x_2 \leq 24\\ & x_i \in \mathbb{N}, i = 1, 2\end{aligned}$$

We continue using two dimensional problems because it is easy to visualize, see Figure 3.1. Extending this problem to more dimensions is trivial, but harder to visualize.

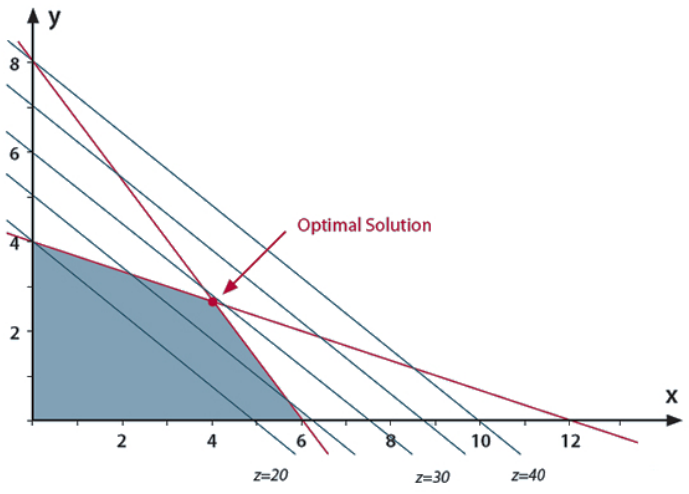


Figure 3.1: Graphical solution to (IP) using Simplex method. This results in non integer optimal solution. B& B must be used to solve. The objective value is constant along each of the diagonal lines.

The optimal solution to (IP), as can be seen graphically in Figure 3.1, is not integer. We need a way to " Cut " this solution away so Simplex can find the correct solution. The initial solution here is $x = 4, y = 2.67$. Now divide the area into two new areas were we for instance say $y \leq 2$ and $y \geq 3$ for the two areas respectively, and re-optimize both areas with Simplex. We can continue like this until we find the best integer solution. By dividing the feasible area into regions we will get integer solutions on some of the vertices and the Simplex method will find them.

To manage all the new solutions we create a search tree, called a B&B tree. These trees consist of sub problems denoted as nodes and new constraints added defined as edges. We start at the root node by simply solving the LP with no extra constraints. When nodes are found that have a worse solution than the pessimistic bound we can cut them away because these sub problems cannot provide better solutions. Also if we find an integer solution in one region which is better than a fractional solution in another region, there is no reason to search the region with the fractional solution. The B&B tree is shown in Figure 3.2 and the optimal integer solution can be seen from Figure 3.3. This is an example where rounding to the nearest solution would not work. This would give the solution from node 3 in Figure 3.2 which is clearly not as good as node 7 in the B&B tree.

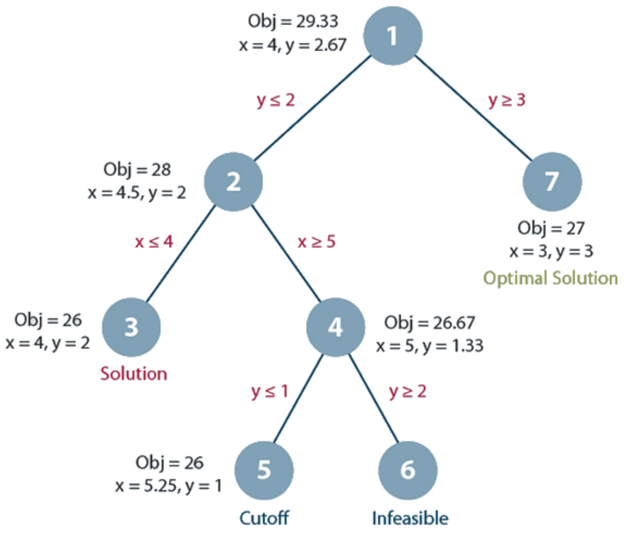


Figure 3.2: The B&B tree for (IP). Start by optimal solution from Figure 3.1. Branch on y and search in the region with the best objective value (Best region first strategy). Stop search when no better solutions can be found. The new optimal solution is visualized in Figure 3.3

The number of nodes in the search tree grows exponentially with the number of decision variables, which means it is still a hard problem to solve and solv-

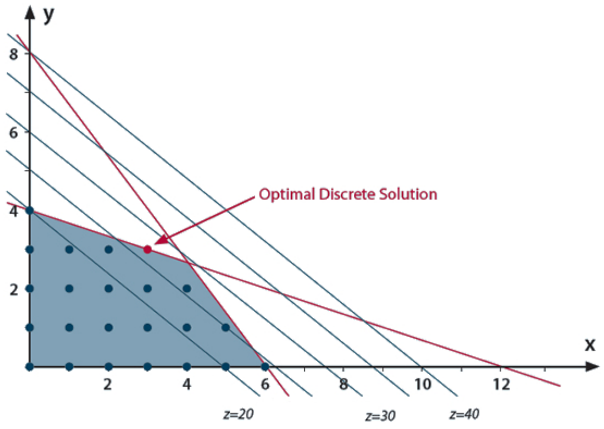


Figure 3.3: The optimal integer solution to (IP) found using Simplex and B&B.

ing large problems with this method may be very time consuming. A popular improvement to the B&B is to add extra binding constraints before branching. This often dramatically decreases the size of the B&B tree. This procedure is known as B&C.

3.3 Branch & Cut

The B&C algorithm is designed to create a "short-cut" in the branching tree. There may be a lot of unnecessary searching in the B&B tree so by restricting the areas with some good cuts, the searching might go faster since less branching is needed. The algorithm of adding new constraints or cuts is called a *Cutting-Plane* algorithm. The challenge is to find constraints that cut away as much of the non-integer solutions without taking away any of the integer solutions. Take the example used in section 3.2. It can easily be seen from Figure 3.3 that the constraint: $x + y \leq 6$ is a legal and good cut. This takes away the optimal LP solution. The new LP solution will find the optimal IP solution without needing to branch at all, see Figure 3.4.

The method of adding cutting planes is equivalent to adding new constraints or new rows to the IP. Good applications for this method are when using problems that have many constraints relative to the number of variables. We can then initially relax some of the constraints, since they are not all needed, and solve the LP. By adding the required rows before we branch we can find optimal solutions

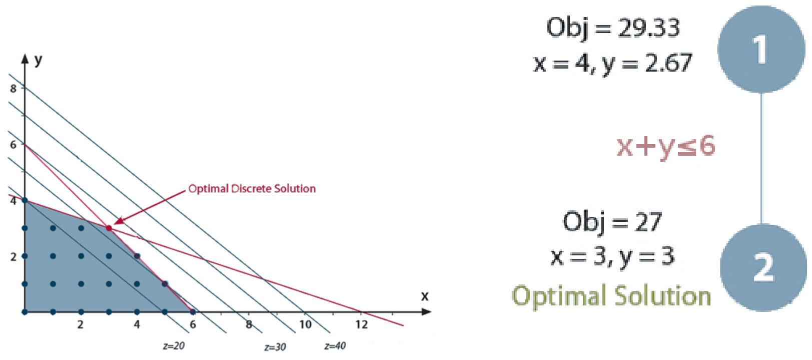


Figure 3.4: The optimal integer solution to (IP) found using Simplex and B&C. No branching is needed.

without using unnecessary constraints. We can also call this procedure Row Generation, since we add rows. Row Generation is used when a problem has relatively many constraints compared to variables. When it is the other way around, we use CG.

3.4 Column Generation

To describe how CG is used we first need to describe more thoroughly how we can represent a problem. There are two ways of describing the set of feasible solutions to an optimization problem, outer- and interior representation. Outer representation represents the region as an intersection of half spaces. This is for instance represented by constraints in an LP. Interior representation describes the feasible region as convex combinations of all the extreme points. An outer representation of the LP described in (3.1)-(3.3) would be:

$$X = \{\mathbf{x} : Ax \leq b, \mathbf{x} \in \mathbb{N}^n\} \tag{3.9}$$

The extreme points of $conv(X)$ can be used for the interior representation, were conv(X) is the smallest convex set that includes X. Let $x^{(k)}, \quad k \in 1, ..., K$ be the extreme points of $conv(X)$, K being the number of extreme points. Define a weighted variable λ_k for each extreme point k. We thus get.

$$X = \{\mathbf{x} : \mathbf{x} = \sum_{k=1}^{K} x^{(k)}\lambda_k, \sum_{k=1}^{K} \lambda_k = 1, \lambda_k \geq 0; k = 1, ..., K, \mathbf{x} \in \mathbb{N}^n\} \tag{3.10}$$

K is often very large, but only a few of the extreme points are normally needed to describe a specific point x^k. The *Dantzig-Wolfe* (D-W) *reformulation* is one of the most used tools to attack large and complex optimization problems and it uses a correspondence between outer and interior representations to reformulate a problem. A D-W *decomposition* is a D-W reformulation combined with a method for solving the reformulated problem. The idea with D-W reformulation is to express parts of the problem by an interior representation and then use this to substitute variables in the rest of the problem. Start with an LP:

$$\begin{aligned} max \quad z = \mathbf{c}^T\mathbf{x} && (3.11) \\ \mathbf{Ax} \leq \ \mathbf{b} && (3.12) \\ \mathbf{Dx} \leq \mathbf{e} && (3.13) \\ \mathbf{x} \in \mathbb{N}^n && (3.14) \end{aligned}$$

were $X_A = \{\mathbf{x} : \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \in \mathbb{N}^n\}$ has a simple structure, but $Dx \leq e$ destroys this structure. We describe X_A via an interior representation. and get:

$$\begin{aligned} max \quad z = \mathbf{c}^T(\sum_{k=1}^{K} x^{(k)}\lambda_k) && (3.15) \\ \mathbf{D}(\sum_{k=1}^{K} x^{(k)}\lambda_k) \leq e && (3.16) \\ \sum_{k=1}^{K} \lambda_k = 1 && (3.17) \\ \lambda_k \geq 0, \quad k = 1, ..., K && (3.18) \end{aligned}$$

This reformulation has fewer constraints but a lot more variables and we call this problem the Master Problem (MP) in the D-W decomposition. The Simplex method introduced concepts of basis and calculating reduced costs for non basic variables. The main idea behind CG is to use a D-W reformulation and start with a small number of variable or columns and generate new ones only when needed. Instead of having K variables, we start with $K' << K$, giving us a restricted MP (RMP). Every feasible solution to RMP is feasible in MP. Associate the reduced costs v with constraints (3.16) and u with constraint (2.17). We use a so called *Pricing Problem* to add new columns by using these reduced costs and find the best column the same way as in the Simplex method. The brilliance behind this method is that we can find the best columns without actually needing to generate them. The pricing problem becomes the following:

$$\begin{aligned} max \quad \bar{\mathbf{c}} = (\mathbf{c}^T - \mathbf{v}^T\mathbf{D})\mathbf{x} - u && (3.19) \\ \mathbf{Ax} \leq \mathbf{b} && (3.20) \\ \mathbf{x} \geq 0 && (3.21) \end{aligned}$$

The result of the pricing problem is the highest reduced cost (maximization problem) and the corresponding **x** solution. If there are no positive reduced costs, we have an optimal solution. If we find a solution, this extreme point is added to the RMP.

A CG algorithm iterates between solving the RMP and the pricing problem resulting in the same optimal solution as the Simplex method, namely the optimal LP solution. For solving IPs we need the B&P method.

3.5 Branch & Price

The idea when using B&P is similar to that of B&C except that the procedure focuses on generating columns rather than rows. Both pricing and cutting are complimentary procedures for tightening an LP. The B&P procedure allows CG to be applied throughout a branch and bound tree. There are several reasons for considering models with huge number of variables.

- A compact MILP may have weak LP relaxations and these can be tightened by a reformulation with more variables.
- A compact MILP may have symmetry issues that make the B&B algorithm perform badly. A reformulation can eliminate this symmetry.
- A formulation with a huge number of constraints may be our only choice.

It may appear at first glance that the B&P algorithm simply involves combining CG and B&B, however it is not straightforward. There are fundamental difficulties when using B&P, these include:

- Conventional IP branching on variables may not be effective because it destroys the structure of the pricing problem.
- Solving the LP relaxed MP and the pricing problem to optimality may not be effective, in which different rules apply for handling the B&P tree

There are many different ways to branch in a B&P environment and it differs from model to model. Therefore further discussion of branching will be discussed for the MCGRP in Chapter 6. However a quick overview of the procedure will be given here, see Figure 3.5. We start by using the CG procedure as explained in Chapter 3.4. If the solution is integer we have an optimal solution, else we create a branching tree similar to the one described in Chapter 3.2. Whenever an integer solution is found it becomes a candidate for upper bound for a minimization problem. We terminate the algorithm when no better solution can be found, similarly described for the B&B algorithm. For the interested reader we refer to Barnhart et al. (1998).

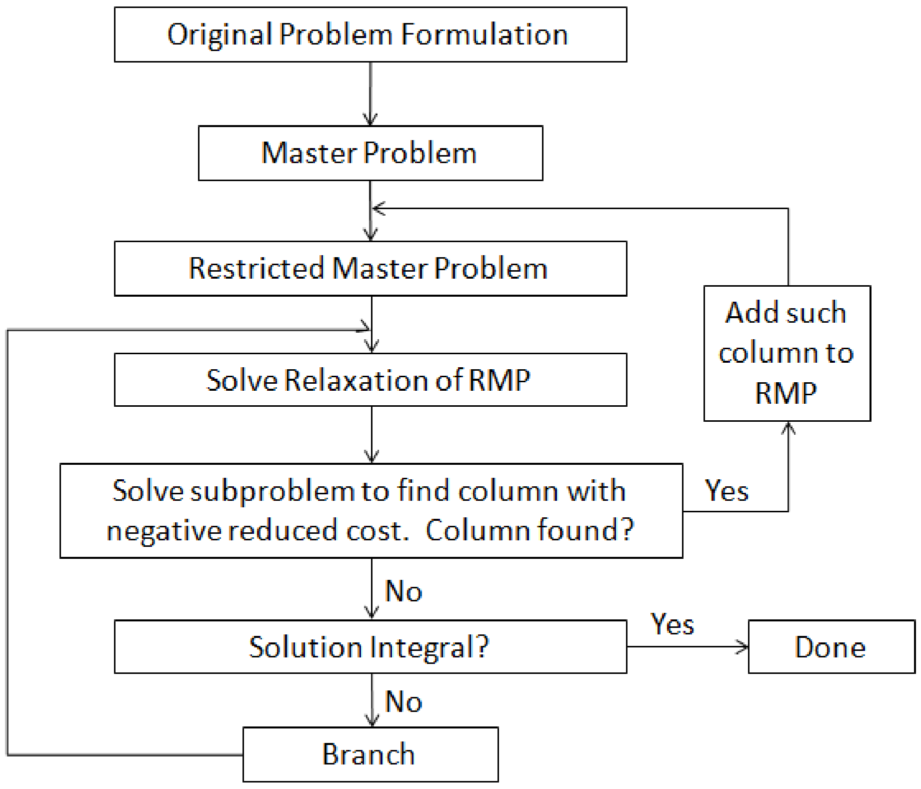

Original Problem Formulation
Master Problem
Restricted Master Problem
Add such column to RMP
Solve Relaxation of RMP
Solve subproblem to find column with negative reduced cost. Column found?
Yes
No
Solution Integral?
Yes
Done
No
Branch

Figure 3.5: The B&P procedure for finding integer solutions. We first restrict the MP to RMP by starting with a few variables. Add new columns until we need to branch. The optimal solution is found when the there are no fractional solutions that are better than the best integer one.

3.6 Heuristics

All methods discussed so far in this chapter have been exact algorithms. However, if an optimal solution is not necessary, but rather a good solution found quickly is sought for, we can use heuristics. Heuristic is a Greek word and means "find" or "discover". Heuristics use experience based methods to find solutions and are not exact procedures. When exact methods are impractical, heuristic methods are often implemented to give good and satisfactory solutions via various short cuts to ease the load of making a decision. Heuristics often work in the way humans think, for instance by using a "rule of thumb", taking an "educated guess" or simply using common sense. Heuristics cannot guarantee optimality, but will often give good solutions. In this thesis only exact methods are investigated therefore only a brief introduction to heuristics will be given and only the heuristics discussed in this thesis will be presented here.

The Greedy heuristics are very simple and can easily be described from the knapsack problem from Section 2.1. The idea is to make local decisions that seem best at that specific point. Say for instance you want to fill your knapsack with gold, silver and bronze figurines. Since we know that gold has the highest worth we start filling our sack with these and continue with silver and then the bronze until our sack is full. This will give a feasible solution that is probably very good. If we combine this with the use of common sense, say "do not pick a figurine that has a low weight:size ratio we end up with the simple algorithm used by most burglars when robbing a jewellery shop.

Metaheuristics is a class of heuristics that uses additional heuristics and overlying strategies to find solutions. The goal is to efficiently explore the search space in order to find near optimal solutions and not terminate at bad local optimal solutions. The techniques used can range from simple search procedures to complex learning strategies. For more information about Heuristics the interested reader is referred to Kochenberger (2003).

CHAPTER 4

ROUTING PROBLEMS

In this chapter we will describe three groups of routing problems more thoroughly. The first two sections are about the well known node- and arc routing problems and the final part is about the combination of the two, and the focus of this thesis, the MCGRP.

4.1 Node Routing Problems

The node routing problems are the problems where nodes, also known as vertices, within a defined area are to be visited by a vehicle. The most basic node routing problem is the TSP. The problem is as follows: Given a list of nodes and the distances between each pair of nodes, find the shortest possible route that visits each node exactly once and returns to the origin node. It is one of the most studied problems in optimization and even Google maps has a TSP algorithm implemented to calculate the routes between destinations, an example TSP route from Trondheim is shown in Figure 4.1.

Another well known node routing problem is the CVRP, presented by Dantzig and Ramser (1959). The CVRP is an extension of the TSP seeking to service a number of customers with a fleet of identical vehicles. Based at a central depot, the vehicles are to be optimally routed to supply customers with known demands subject to vehicle capacity constraints.

The CVRP itself is in the VRP family and there are many variations of the CVRP in the literature, the most famous being VRP with time windows, VRP

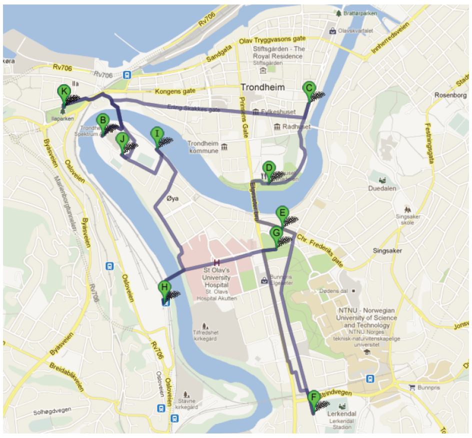


Figure 4.1: A possible solution to a small TSP solved by google maps with 10 destinations in Trondheim, Oslo.

with pick-up and delivery and the CVRP. CVRP is an important problem in the fields of transportation, distribution and logistics. Often the context is that of delivering goods located at a central depot to customers who have placed orders for such goods. An example of a possible CVRP can be seen from Figure 4.2, e.g. the vehicle can only visit 5 vertices in each tour.

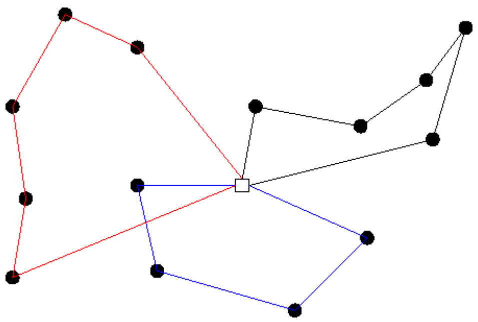


Figure 4.2: A possible solution to a small VRP problem solved using 3 vehicles. All the vertices are required to be visited exactly once by exactly one vehicle.

Typically exact and heuristic solutions have been proposed for each new VRP variant and there has been tremendous increase in the ability to solve these problems over the past 50 years. Today, a Google Scholar search of the words "vehicle routing problem" yields about 159,000 entries, the same search in google yields almost a million. In practise the CVRP is one of the absolute biggest success stories in optimizing history. For instance, each day over 100,000 drivers from the United Parcel Service in USA follow computer generated routes Golden et al. (2008). A few years back the best exact methods could solve CVRP problems up to 70 customers to optimality in reasonable time. Today, this number is well over 100!

4.2 Arc Routing Problems

The next family of routing problems to discuss are the *arc routing* problems. Arc routing problems regard the edges or arcs as required entities, not the vertices. It all started almost three centuries ago with the bridges of Königsberg. The city of Königsberg in Preussia had two big islands in the river that divided the city. There where a total of seven bridges connecting the two islands with the main land on each side, as can be seen from Figure 4.3. The problem was to cross all the bridges exactly one time each and return on the same side as started. By proving that this was impossible, Euler is considered to have created the first theorem of graph theory which also led to the problems known as arc routing.

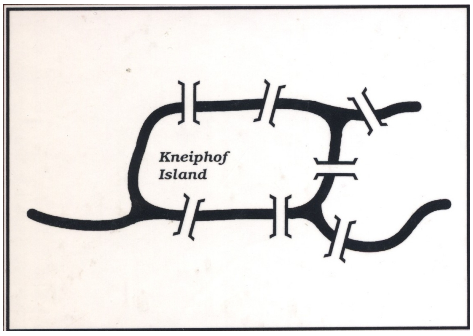


Figure 4.3: The seven bridges on Königsberg, Dror (2000)

Arc routing problems consist of determining the lowest costing traversed route of some edges or arcs, subject to side constraints. Much like node routing problems and the TSP, we have the Chinese Postman Problem (CPP) for arc routing. The CPP is defined as follows. Let $G = (V, E \bigcup A)$ be a graph where V is the set of vertices, E is a set of undirected edges and A is the set of directed arcs. With each edge or arc it is defined a cost for traversing c_{ij}. There are several cases of this problem. For instance: The undirected CPP, where $A = \emptyset$, the directed CPP, where $E = \emptyset$, and the mixed CPP, where $A \neq \emptyset, E \neq \emptyset$. CPP is the arc routing equivalent of the TSP. The bridges of Königsberg can be solved by using the undirected CPP, a variation at least. Find the shortest route that covers all the bridges.

CVRPs aim to optimize the routes with a set of vertices that need to be visited, arc routing problems involve covering parts of network in the most effective manner, e.g. for a snowplough. A snowplough needs to clean a set of streets that can easily be modelled via arc routing. An extended variant of the CPP where we only need to visit a sub set of the edges or arcs and may traverse the rest is called the Rural Postman Problem (RPP). A natural further extension of the RPP is the CARP. The CARP is defined on a similar graph G where each arc or edge has a quantity q_{ij} associated with it. A fleet of m vehicles, each with a capacity Q, must traverse all edges or arcs and must deliver the associated quantities without exceeding Q. Not all links must have a non-negative quantity and need therefore not be visited. The CARP is the arc routing equivalent of the CVRP and was defined by Golden and Wong (1981). Some variants of the problem can include:

- Those in which not all links need to be traversed,
- Cases where some or all links are directed,
- Cases in which links have variable traversal costs depending on direction,
- Hierarchical problems, in which one set of edges must be served before another.

Arc routing has a long history, but it is only in the recent decades that there has been a widespread use of software in the area of arc routing. More and more post offices, school bus operators electricity and gas companies etc. are adopting such systems. In recent years column generation has been given some attention with good results, Letchford and Oukil (2009) and a branch and price method has been introduced by Bode and Irnich (2012). Numerous researchers have investigated the conversion of arc-routing problems to node routing problems (Longo et al. (2006), Baldacci and Maniezzo, (2006)). These approaches transform each of the edges or arcs in the CARP to 2 or 3 nodes in the CVRP. This transformation shows that the CARP can be just as hard or even harder to solve as the CVRP.

4.3 The Mixed Capacitated General Routing Problem

The last routing problem that will be explained is a mix between arc and node routing problems. If routing is to be done on a specific subset of edges and arcs and a specific subset of vertices then the problem will be a mix between RPP and a TSP. This problem class has been named the General Gouting Problem (GRP). The extension to multiple vehicles with a capacity was named the Capacitated GRP (CGRP) and was first investigated by Jansen, K. (1993). Two years later Pandit and Muralharan (1995) investigated the CGRP on a mixed graph with both edges and arcs, they called this problem the Mixed Capacitated General Routing Problem. The MCGRP is a generalization of CARP and the CVRP.

There are some problems where neither the CVRP nor CARP seem to be good formulations to use. Prins & Bouchenoua (2005) state:

Despite the success of heuristics for the VRP and the CARP, it is clear that these two problems cannot formalize the requirements of many real world scenarios.

The example used by them is urban waste collection, where most demand may easily be modelled on street segments, but it may also be demand located at points, for instance at hospitals or shopping centres. An example of a MCGRP instance can be seen from 4.4.

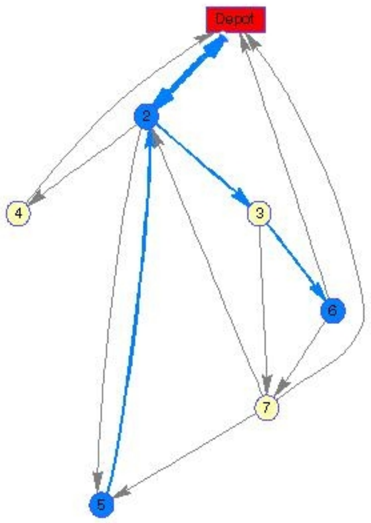


Figure 4.4: Circles: Vertices, Double arrow heads: Edges, Single arrow heads : Arcs. Blue colour for required entities. Red is the depot vertex. The goal is to minimize the distance travelled when servicing all the required entities.

There has been little research on the MCGRP, but a short summary of the previous work will be given. Pandit and Muralidharan (1995) and Gutiérrez et al. (2002) propose heuristics for variants of the MCGRP. Prins and Bouchenoua (2005) studied the MCGRP under the alias The Node, Edge, and Arc Routing Problem (NEARP). They define a benchmark (CBMix), and propose a memetic algorithm that provided the first upper bounds for the CBMix and also good results for well-known CVRP and CARP instances. Kokubugata et al. (2007) develop a metaheuristic with several new best CBMix upper bounds. The MC-GRP with turning penalties was investigated by Bräysy et al. (2011). This problem is created motivated by the fact that vehicle turns are troublesome and avoiding them is desirable. Their solution is to transform the MCGRP into a CVRP and solve it with well known heuristics. The first lower bounding procedure for the MCGRP was proposed by Bach et al. (2012). They also define two new MCGRP benchmarks: the BHW and the DI-NEARP. Bosco et al. (2012) propose the first integer programming formulation for the MCGRP and develop a B&C algorithm that was tested on 12 new sets of instances derived from CARP benchmarks, as well as small CBMix instances, providing two optimal solutions.

CHAPTER 5

MODELLING THE MCGRP AS INTEGER PROGRAMS

In this section we will describe two mathematical model formulations for the MCGRP. The first model is an *Arc Flow* model formulation created by Bosco et al. (2012). The term arc flow is used when the decision variables are defined on arcs in the network. This is a model that is suited for B&B- or B&C based solution methods. The second model is a novel *Path Flow* model formulation in the set partitioning form. The term path flow refers to models were the decision variables refer to paths in the network. B&P can be well suited to solve this formulation.

5.1 Arc Flow Model Formulation

Formulating a mathematical problem can be complex and there can be numerous formulations for the same problem. Prior to this thesis there was only one formulation, that the writer knows about, of the MCGRP in the literature. This formulation is an arc flow formulation created by Bosco et al. (2012). The name arc flow comes from the fact that the decision variables are the flow on each arc or edge.

We start by defining a mixed graph, see Definition 5.1.1 and Figure 5.1.
Definition 5.1.1: *Mixed Graph.* $G = (V, A, E)$ defined by a set of vertices or nodes $V = \{1, ..., n\}$, a set of arcs (directed) $A = \{(i, j) \subseteq V \times V\}$ and a set of

edges (undirected) $E = \{(i,j) \subseteq V \times V : i < j\}$.

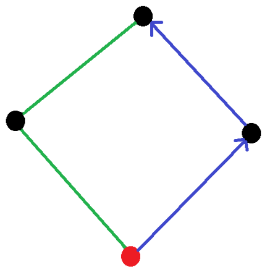


Figure 5.1: An example of a mixed graph. Black balls are the vertices and red ball is the depot. Blue links are arcs and green links are the edges.

Further, a subset of the edges and arcs, are denoted $E_R \subseteq E$ and $A_R \subseteq A$ respectively and are required links. This means they must be serviced by one of the tours, but any link in $A \cup E$ may be traversed any number of times. Similarly, we have a subset of required vertices, $V_R \subseteq V$. The required entities cannot be split, which means that you have to complete an entity that has been initialized. Each link has a non-negative traversal cost c_{ij}, each required link has in addition a non-negative demand d_{ij} and finally the required vertices have a demand q_i.

In order to ensure feasibility it is assumed that the demand of a single required entity does not exceed the capacity Q of a vehicle. Now for the more complicated notation. We will be needing all possible subsets

$$S \subseteq V$$

and their complementary subsets of vertices when modelling our problem,

$$\overline{S} = V \setminus S.$$

A few more sets are needed:

$$\delta^+(S) = \{(i,j) \in A : i \in S \land j \in \overline{S}\} \tag{5.1}$$

$$\delta^-(S) = \{(i,j) \in A : i \in \overline{S} \land j \in S\} \tag{5.2}$$

$$\delta_R^+(S) = \{(i,j) \in A_R : i \in S \land j \in \overline{S}\} \tag{5.3}$$

$$\delta_R^-(S) = \{(i,j) \in A_R : i \in \overline{S} \land j \in S\} \tag{5.4}$$

$$\delta(S) = \{(i,j) \in E : i \in S \land j \in \overline{S}\} \tag{5.5}$$

$$\delta_R(S) = \{(i,j) \in E : i \in S \land j \in \overline{S}\} \tag{5.6}$$

These new sets can be described as follows: Equation (5.1) and (5.2) represent the sets of all arcs leaving and entering the subset S, respectively. Equation (5.3) and Equation (5.4) represent the sets of all required arcs leaving and entering the subset S respectively. And finally Equation (5.5) and Equation (5.6) are the sets of all edges and all required edges incident to subset S, respectively.

Moreover, let $S_R = S \cap V_R$ be the set of required vertices belonging to S, $A_R(S) = \{(i,j) \in A_R : i \in S \wedge j \in S\}$ the set of required arcs with both endpoints in S, and $E_R(S) = \{(i,j) \in E_R : i \in S \wedge j \in S\}$ the set of required edges with both endpoints in S. The variables

$$x_{ij}^k, y_{ij}^k, z_i^k,$$

where x_{ij}^k is a binary variable equal to 1 if and only if the link (i,j) is serviced by vehicle k, y_{ij}^k is a integer variable equal to the number of times a link (i,j) is dead-headed by vehicle k. For a required vertex i and vehicle k, we have z_i^k is a binary variable equal to 1 if and only if i is serviced by k. We now have enough information to formulate the MCGRP. The mathematical program is defined by the equations (5.7) - (5.16).

$$\begin{aligned}\text{Min } \lambda \ &= \sum_{k\in K}\sum_{(i,j)\in E_R} c_{ij}(x_{ij}^k + x_{ji}^k) + \sum_{k\in K}\sum_{(i,j)\in A_R} c_{ij}x_{ij}^k \\ &+ \sum_{k\in K}\sum_{(i,j)\in E} c_{ij}(y_{ij}^k + y_{ji}^k) + \sum_{k\in K}\sum_{(i,j)\in A} c_{ij}y_{ij}^k \qquad (5.7)\end{aligned}$$

$$\sum_{k\in K}(x_{ij}^k + x_{ji}^k) = 1 \qquad \forall\ (i,j)\ \in E_R \quad (5.8)$$

$$\sum_{k\in K} y_{ij}^k = 1 \qquad \forall\ (i,j)\ \in A_R \quad (5.9)$$

$$\sum_{k\in K} z_i^k = 1 \qquad \forall\ i\ \in V_R \quad (5.10)$$

$$\sum_{i\in V_R} q_i z_i^k + \sum_{(i,j)\in E_R} d_{ij}(x_{ij}^k + x_{ji}^k) + \sum_{(i,j)\in A_R} d_{ij}x_{ij}^k \leq Q \qquad \forall\ k\ \in K \quad (5.11)$$

$$\sum_{j:(i,j)\in\delta_R^+(i)} x_{ij}^k + \sum_{j:(i,j)\in\delta^+(i)} y_{ij}^k - \sum_{j:(i,j)\in\delta_R^-(i)} x_{ji}^k - \sum_{j:(i,j)\in\delta^-(i)} y_{ji}^k =$$
$$\sum_{j:(i,j)\in\delta_R(i)} x_{ji}^k + \sum_{j:(i,j)\in\delta(i)} y_{ji}^k - \sum_{j:(i,j)\in\delta_R(i)} x_{ij}^k - \sum_{j:(i,j)\in\delta(i)} y_{ij}^k, \ \forall\, k\, \in K, \ \forall\, i\, \in V \tag{5.12}$$

$$\sum_{(i,j)\in\delta_R^+(S)} x_{ij}^k + \sum_{(j,i)\in\delta_R^-(S)} x_{ji}^k + \sum_{(i,j)\in\delta_R(S)} (x_{ij}^k + x_{ij}^k) + \sum_{(i,j)\in\delta^+(S)} y_{ij}^k$$
$$+ \sum_{(j,i)\in\delta^-(S)} y_{ji}^k + \sum_{(i,j)\in\delta(S)} (y_{ij}^k + y_{ji}^k) \geq \begin{cases} 2 \quad (x_{uv}^k + x_{vu}^k), & \forall\,(u,v)\, \in E_R(S), \\ 2 \quad x_{uv}^k, & \forall\,(u,v)\, \in A_R(S), \\ 2 \quad z_h^k, & \forall\, h\, \in S_R, \end{cases}$$
$$\forall\, k \in K, S \subseteq C \tag{5.13}$$

$$x_{ij}^k \in \{0,1\} \qquad \forall i,j,k \tag{5.14}$$
$$z_i^k \in \{0,1\} \qquad \forall i,k \tag{5.15}$$
$$y_{ij}^k \in \mathbb{N} \qquad \forall i,j,k \tag{5.16}$$

- The objective function from Equation (5.7) minimizes the total routing cost of the tours.
- Constraints (5.8) - (5.10) are known as the assignment constraints and ensure that the required nodes and links are visited by exactly one vehicle.
- The constraints in (5.11) are the so called knapsack constraints which model the demand limitations of each vehicle. Each vehicle can maximum satisfy a demand Q.
- Equation (5.12) represents the flow constraints. These model the symmetry condition at each vertex, the flow conservation. Much like Kirchhoff's junction rule that we know from electro physics (Young and Freedman, 2013) namely the sum of all vehicles going into a vertex must also leave this vertex.
- Finally we have the connectivity constraints from Equation (5.13). These constraints make sure that we reach all the required jobs. They also make sure that no subtours are created.
- The variables are given in (5.14)-(5.16)

The model (5.7) - (5.16) is an arc flow model for the MCGRP. It is a highly complex model and can only be solved to optimality with a small number of vertices and vehicles. One of the reasons that the formulation is so complex is because there are an exponential amount of constraints used to remove all possible subtours, formulated by (5.13). By solving this model by itself only B&B is needed because all possible constraints have already been created. Without these constraints a solution to the problem would most probably result in subtours. The number of subsets that can be created from a set of vertices is exponential proportional to the number of vertices in the set, see Table 2.1. Many of these eliminations might be unnecessary and we can save a lot of work by not creating them all. We propose an algorithm to take care of these constraints..

We will now explain a simple subtour elimination algorithm that can be implemented together with the Mathematical Program (5.7)-(5.12) and (5.14)-(5.16). The idea is to relax constraint (5.13) and iteratively only add the sub-tour eliminations that are binding. Relaxing a constraint basically means to ignore it. We find sub tours by looking for *connected components* in the graph.

Definition 5.2.1 *Connected components.* A graph is *connected* if every vertex is reachable from from all other vertices. A connected component is a subset of vertices with this property.

Figure 5.2 shows a graph with 2 connected components, 1-3 and 4-5.

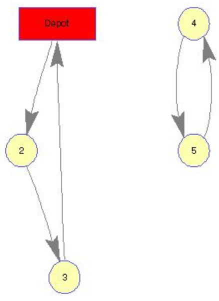


Figure 5.2: An example a graph with 2 connected components.

A three-stage algorithm is proposed to eliminate sub tours.

- **Step 1** Solve the LP relaxation of the mathematical problem from (5.7) subject to the constraints from equation (5.8) - (5.12).
- **Step 2** For all m. Find the connected components by using depth first search. If no components are found, go to step 4, else go to step 3.

- **Step 3** Create new constraints by using the subsets from the connected components and Equation (5.13). Resolve the LP and go back to step 2.
- **Step 4** If integer, Exit, else branch and resolve the LP and go back to step 2.

The goal of this simple subtour elimination algorithm is to avoid adding unnecessary constraints. Say we want to make a single TSP tour from Figure 5.2. Here we have 2 connected components. By using the subtour elimination algorithm we would force links between the two sets, defined by new constraints. This might make new subtours and we could have to impose further constraints.

The procedure explained here is a possible way of solving the the arc formulation of the MCGRP. A similar algorithm was developed by Bosco et al. (2012) and their results will be discussed in Chapter 7.

5.2 Path Flow Model Formulation

The following model is a novel path flow model for the MCGRP, defined by this thesis. In this case the paths are route variables. Each variable refers to a specific route. It is a relatively straight forward formulation of the known *Set Partitioning* form.

Let Ω denote the set of all possible feasible routes for a given vehicle with capacity Q. Define a binary variable λ_r for each $r \in \Omega$ taking the value 1 if the route is used or 0 if it is not used. $a_{re} = 1$ if edge e is included in route r and $a_{re} = 0$ if it is not included. This is similar for the required arcs and vertices, a_{ra} and a_{rv} respectively. c_r is the cost for each route. m is the number of vehicles. The formulation can be seen from (5.17)-(5.22).

$$\min \sum_{r\in\Omega} c_r \lambda_r \tag{5.17}$$

$$\text{s.t.} \sum_{r\in\Omega} a_{re}\lambda_r = 1 \quad \forall e \in E_R \quad (\pi(E)_e) \tag{5.18}$$

$$\sum_{r\in\Omega} a_{ra}\lambda_r = 1 \quad \forall a \in A_R \quad (\pi(A)_a) \tag{5.19}$$

$$\sum_{r\in\Omega} a_{rv}\lambda_r = 1 \quad \forall v \in V_R \quad (\pi(V)_v) \tag{5.20}$$

$$\sum_{r\in\Omega} \lambda_r \leq m \tag{5.21}$$

$$\lambda_r \in \{0,1\} \quad \forall r \in \Omega \tag{5.22}$$

- The objective function from Equation (5.17) minimizes the total routing cost of the tours.
- Constraints (5.18) - (5.20) are the covering constraints which forces all entities to be satisfied. $(\pi(E)_{ij})$ are the dual variables associated with each edge covering constraint. This is similar for $(\pi(A)_{ij})$ and $(\pi(V)_i)$
- Constraint (5.21) makes sure only the prescribed number of vehicles are used.

The path flow formulation is a D-W reformulation of the arc flow formulation from Chapter 5.1. By defining constraints (5.8)-(5.10) as the constraints from $\mathbf{Dx} \leq \mathbf{e}$ and constraints (5.11)-(5.13) as the constraints in $\mathbf{Ax} \leq \mathbf{b}$ from Chapter 3.4 and using D-W reformulation we get the path flow formulation. The new formulation has a lot fewer constraints, but a lot more variables. As will be discussed in Chapter 7, this formulation will, by itself, be weak. The number of routes is of factorial dependence of the number of entities to service. The path flow model is a relatively straight forward model to explain, however as we will see in the next chapter, the solution strategy can be rather complex.

CHAPTER 6

THE BRANCH & PRICE PROCEDURE

In this chapter we will go through the solution method for the path flow formulation model based on a B&P procedure. The CG algorithm used in this thesis is inspired by an algorithm created for the CARP by Letchford and Oukil (2009). CG solves an LP relaxation of the RMP and to ensure integer solutions we need to branch. Letchford and Oukil (2009) do not propose a branching scheme in their paper since their focus is solely on comparing various pricing schemes. However Bode and Irnich et al. (2011) expand the research and define the first B&P algorithm for the CARP. In their paper they focus on a pricing problem based on the so called *non-elementary* routes which means they need to use a relatively complicated branching scheme. In this thesis a new branching scheme is proposed. The B&P algorithm is the first of its kind for the MCGRP. In Chapter 6.1 the CG formulation will be presented and the full B&P algorithm will be described in Chapter 6.2.

6.1 Column Generation

The reason we use CG is because it is too costly to generate all the possible routes. We only want to create the necessary routes and ignore the others. This idea fits perfectly to a column generation environment with our MP becoming the path flow problem described in Chapter 5.2. We will restrict the MP by only starting with a few routes, giving us the the RMP. The RMP uses $\Omega_s \subset \Omega$ where Ω_s includes a subset of the routes. We start with a sufficiently large pool of a priori generated routes. Enough routes for a feasible solution to the RMP. We solve an LP relaxation of the RMP and use a pricing problem to add new variables by creating new feasible routes.

The pricing problem used in this thesis is inspired on a mathematical program created for the CARP by Letchford and Oukil (2009). The formulation includes a RMP in the form known as set partitioning and a pricing problem based on solving the Shortest Path Problem with Resource Constraints (SPPRC), explained by Feillet et al. (2004). The different pricing methods used by Letchford and Oukil (2009) range from being fast with weaker bounds to slower with stronger bounds, see Table 6.1. The pricing problem model used in this thesis is the one with strongest bounds, the model that uses the elementary routes. Elementary routes are basically legal routes in the way that they have no cycles and that they fulfil the capacity and entity covering requirements of the MCGRP. This is compared to a non-Elementary route that can service the same entity multiple times and have cycles. The reason to choose the Elementary SPPRC (ESPPRC) is because we get the best lower bounds and since we also generate legal routes, feasible upper bounds will also be generated at each iteration.

Table 6.1: Sub problem results for the CARP. The first four rows are all non-elementary routes with various eliminations or relaxations.

Method	**Speed**	**Bound Strength**
Without cycles elimination (SPPRC)	very fast	weak bounds
With 2-cycles elimination (2-cyc-SPPRC)	slower	stronger bounds
With k-cycles elimination (k-cyc-SPPRC)	slower	stronger bounds
ng-routes relaxation (ng-SPPRC)	slower	stronger bounds
Elementary routes (ESPPRC)	slowest	strongest bounds

The pricing problem is the following: For each $\{i,j\} \in E_R$ define two binary variables x_{ij} and x_{ji}, such that $x_{ij} + x_{ji} = 1$ if this edge is serviced. Then for each $\{i,j\} \in E$ define two binary variables y_{ij} and y_{ji} such that $y_{ij} + y_{ji} = 1$ if this edge is traversed. For each $\{i,j\} \in A_R$ define one binary variable $x_{ij} = 1$ if this arc is serviced. Then for each $\{i,j\} \in A$ define one binary variable $y_{ij} = 1$ if this arc is traversed. For each $i \in V_R$ define the variables $z_i = 1$ if this vertex is serviced. Finally define for each $\{i,j\} \in \{E \cup A\}$ the variables f_{ij} (as well as f_{ji} for the Es). $f_{ij} = 0$ if $y_{ij} = 0$, but if $y_{ij} = 1$, f_{ij} represents the remaining load on the vehicle when the vehicle arrives at j from i. We will also use the notation $y(\delta^+(i))$ and $y(\delta^-(i))$ which represent the sum of the traversal variables leaving and entering i respectively. Similar notation for f will be used.

$$\begin{aligned} \min \; & \sum_{\{i,j\} \in E} c_{ij}(y_{ij} + y_{ji}) + \sum_{\{i,j\} \in A} c_{ij} y_{ij} \\ & - \sum_{\{i\} \in V_R} \pi(V)_i z_i - \sum_{\{i,j\} \in E_R} \pi(E)_{ij}(x_{ij} + x_{ji}) - \sum_{\{i,j\} \in AR_{ij}} \pi(A)_{ij} x_{ij} \end{aligned} \tag{6.1}$$

subject to the following constraints:

$$x_{ij} + x_{ji} \le 1 \qquad \forall \{i,j\} \in E_R \tag{6.2}$$
$$y_{ij} \ge x_{ij}, y_{ji} \ge x_{ji} \qquad \forall \{i,j\} \in E_R \tag{6.3}$$
$$y_{ij} \ge x_{ij} \qquad \forall \{i,j\} \in A_R \tag{6.4}$$
$$y(\delta^+(i)) = y(\delta^-(i)) \qquad \forall i \in V\text{'} \tag{6.5}$$
$$f(\delta^+(i)) = f(\delta^-(i)) - \sum_{\{i,j\} \in \delta_R^+(i)} q_{ij} x_{ij} - qV_i z_i \qquad \forall i \in V\text{'} \tag{6.6}$$
$$f(\delta^+(0)) - f(\delta^-(0)) + \sum_{\{0,j\} \in \delta_R^+(0)} q_{0j} x_{0j} \le Q \tag{6.7}$$
$$f_{ij} \le Q y_{ij} - q_{ij} x_{ij} - z_i q V_i, \quad f_{ji} \le Q y_{ji} - q_{ji} x_{ji} - z_j q V_i \qquad \forall \{i,j\} \in E_R \tag{6.8}$$
$$f_{ij} \le Q y_{ij} - q_{ij} x_{ij} - z_i q V_i \qquad \forall \{i,j\} \in A_R \tag{6.9}$$
$$f_{ij}, f_{ji} \ge 0 \qquad \forall \{i,j\} \in E \tag{6.10}$$
$$f_{ij} \ge 0 \qquad \forall \{i,j\} \in A \tag{6.11}$$
$$y_{ij}, y_{ji} \in \mathbb{N} \qquad \forall \{i,j\} \in E \tag{6.12}$$
$$y_{ij} \in \mathbb{N} \qquad \forall \{i,j\} \in A \tag{6.13}$$
$$x_{ij}, x_{ji} \in \{0,1\} \qquad \forall \{i,j\} \in E_R \tag{6.14}$$
$$x_{ij} \in \{0,1\} \qquad \forall \{i,j\} \in A_R \tag{6.15}$$
$$z_i \in \{0,1\} \qquad \forall \{i\} \in V_R \tag{6.16}$$

- The Pricing objective function in equation (6.1). This is very similar to each step in the Simplex method. By using the dual variables we can find the best new column for the RMP.
- Constraint (6.2) ensures that each edge is serviced at most once.
- Constraints (6.3) and (6.4) ensure that an edge that is serviced will also be traversed.
- Constraint (6.5) ensures that each vehicle leaves each vertex that it enters.
- Constraint (6.6) ensures that a service reduces the load on the vehicle and (6.7) ensures that the total demand is less than the vehicle capacity Q.
- Constraints (6.8) and (6.9) restrict the vehicle load en route.
- The rest of the constraints are trivial non-negativity, integer and binary constraints.

6.2 The Branching

A difficult part about using CG for integer programs is the development of branching rules to ensure integrality. Rules that are appropriate for IPs, where the entire set of columns is explicitly available, do not work well with restricted IPs, where the columns are generated by implicit techniques, as discussed in Chapter 3.5.

When using the B&B algorithm for the arc flow model from Chapter 5.1 it is very easy to branch. There are relatively few variables in this formulation. One possible strategy is the following:

1. Solve the LP relaxation
2. Find a decision variable x_{ij}^k that is non-integer and non zero
3. Create the two branches $x_{ij}^k = 0$ and $x_{ij}^k = 1$ and resolve the LP relaxations with these two extra constraints.

When no fractional variables are found we have an integer solution. The best integer solution becomes an Upper bound (UB). The best unexplored fractional solution becomes the Lower Bound (LB). All costs and demands in our problem are integer, therefore when

$$UB - LB < 1$$

we have an optimal solution. This procedure for B&B is relatively easy and most linear programming software can handle this search tree by itself. However this strategy is poor in a B&P environment, see Chapter 3.5.

In the B&P environment we can have multiple strategies that might work. The goal is to find a strategy that guarantees convergence. When branching is done we in general divide the solution space into smaller parts. If we for some reason miss to search some parts of the solution space, we might not find an optimal solution. We call it a *complete branching* scheme if we can find all possible integer solutions.

The most common approach is to branch on the MP so as to keep the pricing problem unchanged or branch on the pricing problem and keep the MP unchanged. Bode and Irnich (2011) use a rather complicated *follower*, *non-follower* branch methodology in their branching for the CARP. With this approach they change the underlying graph structure by deleting some edges and adding others. They explain that this is the only way to ensure an integer solution for the CARP when using non elementary paths as their pricing problem. However, they have used a different pricing method to the one presented here. When pricing with elementary routes we will show that it is sufficient with *pairing*- and *non pairing* constraints to ensure the integer solutions.

A pairing constraint means that two required entities must be in the same route. Non-pairing is simply that they cannot be in the same route. The idea is to add new constraints to the pricing problem from Chapter 6.1 that enforces new routes with these properties. A way to do this is simply to add suitable constraints to the pricing problem . An important aspect is that we do not need to do anything with the MP with this branching strategy, however we need to delete some of the previously generated variables where the associated routes do not satisfy the new constraints. Now we need to ensure that this is a complete branching scheme.

Let $W_R = \{E_R \cup A_R \cup V_R\}$ be the set of all required entities and let the binary variables $w_{r,i} = 1$ if route r services entity i. Let each route have a corresponding binary variable $\lambda_r = 1$ if it is used in the optimal solution. A pairing relationship between entity i and j is when the two respective entities are serviced by the same route. This can be represented by the constraint:

$$w_{ri} = w_{rj} \quad \forall r, \quad i \neq j$$

A non- pairing relationship is when maximum one of the entities is serviced by any route. This can be represented by the constraint:

$$w_{ri} + w_{rj} \leq 1 \quad \forall r, \quad i \neq j.$$

The route columns w_r are what the pricing problem generates for the RMP at every iteration. When no more routes are found with negative reduced cost, we have the optimal solution to the LP relaxed RMP. We will then have a set of route variables with values $0 \leq \lambda_r^* \leq 1$, $\forall r$. A route variable with a positive λ_r-value we call an *active route*.

Branching Theorem :
For every pair of entities there can only be active routes that include either a non-pairing relationship or a pairing relationship for the solution to be integer. Further, if there only exists active routes that fulfil this requirement we have a guaranteed integer solution.

Proof. Let b be the current branching node and Ω_b be the set of associated routes. Assume that for some pair of entities (i, j), we have an active route r with a pairing relationship between i and j and an active route q with a non-pairing relationship between i and j. The corresponding routing variables are $\lambda_r > 0, \lambda_q > 0$. From the covering constraints (5.18)-(5.20) we know that:

$$\lambda_r w_{ri} + \lambda_q w_{qi} = 1 \text{ or}$$
$$\lambda_r w_{rj} + \lambda_q w_{qj} = 1$$

Since the w's are binary the λ's must be fractional.

For the next part, say for all entity pairs and all active routing variables there will only be one active route with either a pairing- or a non-paring relationship. Say entity j is serviced by the active route r. To have a fractional solution j will also be serviced by another route, say q. For the two routes r and q not to be identical we say route r also services entity i and route q does not. However this means that there exists both a non-pairing and pairing relationship of the entity pair (i, j) after all. This is a contradiction and the routes r and q must be identical. Further the CG procedure would not generate the same route twice because the reduced cost of variables in the basis is 0. We can therefore conclude that each entity will only be serviced by exactly one route and we have an integer solution

$\square$

To select the entity pair to branch on we need to check if there exists active routes that include both pairing and non-pairing relationships. This is done by adding up all the pairings and all the non-pairings. R is the total number of routes in the current branching node b.

Matrix N includes all the non-pairing relationships. N:=

$$\begin{array}{c|ccccc}
 & x_1 & x_2 & x_3 & \cdots & x_n \\
\hline
x_1 & 0 & \sum_{i=1}^{R} \lambda_i \mid x_{i1} - x_{i2} \mid & \sum_{i=1}^{R} \lambda_i \mid x_{i1} - x_{i3} \mid & \cdots & \sum_{i=1}^{R} \lambda_i \mid x_{i1} - x_{in} \mid \\
x_2 & - & 0 & \sum_{i=1}^{R} \lambda_i \mid x_{i2} - x_{i3} \mid & \cdots & \vdots \\
x_3 & - & - & 0 & \ddots & \vdots \\
\vdots & - & - & - & 0 & \sum_{i=1}^{R} \lambda_i \mid x_{i(n-1)} - x_{in} \mid \\
x_n & - & - & - & - & 0
\end{array}$$

Matrix P includes all the pairing relationships. P:=

$$\begin{array}{c|ccccc}
 & x_1 & x_2 & x_3 & \cdots & x_n \\
\hline
x_1 & 1 & \sum_{i=1}^{R} \lambda_i (x_{i1} \cdot x_{i2}) & \sum_{i=1}^{R} \lambda_i (x_{i1} \cdot x_{i3}) & \cdots & \sum_{i=1}^{R} \lambda_i (x_{i1} \cdot x_{in}) \\
x_2 & - & 1 & \sum_{i=1}^{R} \lambda_i (x_{i2} \cdot x_{i3}) & \cdots & \vdots \\
x_3 & - & - & 1 & \ddots & \vdots \\
\vdots & - & - & - & 1 & \sum_{i=1}^{R} \lambda_i (x_{i(n-1)} \cdot x_{in}) \\
x_n & - & - & - & - & 1
\end{array}$$

The lower left corners of the matrix are ignored because of symmetry. We want to choose a pair of entities that have non-zero entries in both matrices. In normal variable branching it is most common to branch on either the most fractional variable or the fractional variable closest to 1. The general idea is then that more fractional solutions are "cut" away. However the research from Achterberg et al. (2004) says otherwise. Their research show that there is no significant increase in performance for choosing any particular variable to branch on. Therefore it is just as efficient to choose a random pair of variables that fulfil the given requirements. Our algorithm terminates when no better integer solutions can be found.

CHAPTER 7

COMPUTATIONAL EXPERIMENTS AND RESULTS

This chapter is about the computational experiments performed in this thesis and the results produced by them. The purpose of this thesis was to investigate a new and hopefully better algorithm to solve the MCGRP and the results will be produced here. In Chapter 7.1 we start by describing and visualizing an MCGRP instance. In Chapter 7.2 the benchmark instances created for the MCGRP will be introduced. Chapter 7.3 will describe the experimental set-up used for the experiments.

In Chapter 7.4 we present the final results produced by the B&P algorithm as well as the earlier results produced in this thesis. A total of 158 instances were solved by the algorithm. All the results will be presented and discussed.

7.1 Visualized example

Let us have a closer look at one of the instances. The instance described here is one of the smaller instances investigated in this thesis with a total of 8 vertices, 18 arcs, 2 edges where 3, 6 and 1 of them are required respectively.

- *Required Vertices*: 2, 5 and 6
- *Required Edges*: 1$\leftrightarrow$2
- *Required Arcs*: 1$\rightarrow$4, 2$\rightarrow$3, 2$\rightarrow$7, 5$\rightarrow$7, 6$\rightarrow$8 and 7$\rightarrow$3

- *Non-Required Edges*: 2↔4

- *Non-Required Arcs*: 1→5, 1→6, 2→4, 3→2, 3→7, 4→1, 5→1, 5→2, 6→1, 7→2, 7→5 and 8→6

To better understand what we are dealing with, a Matlab plot of the instance is visualized in Figure 7.1.

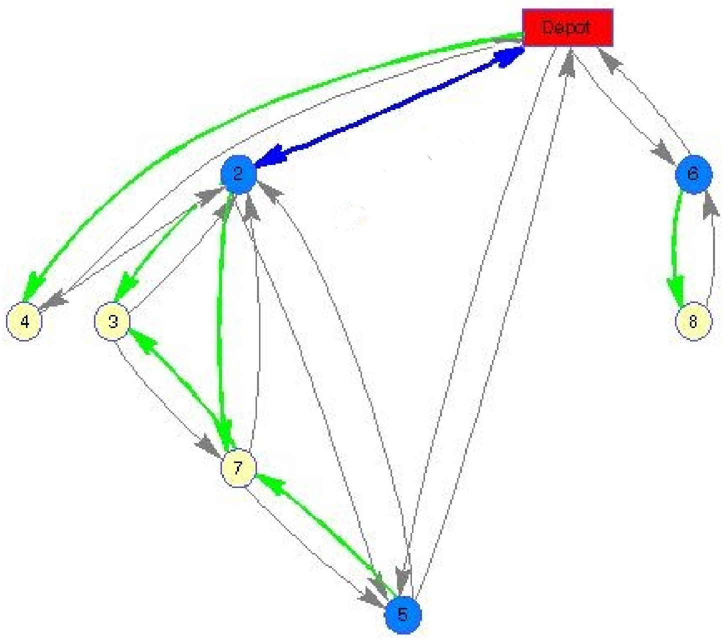
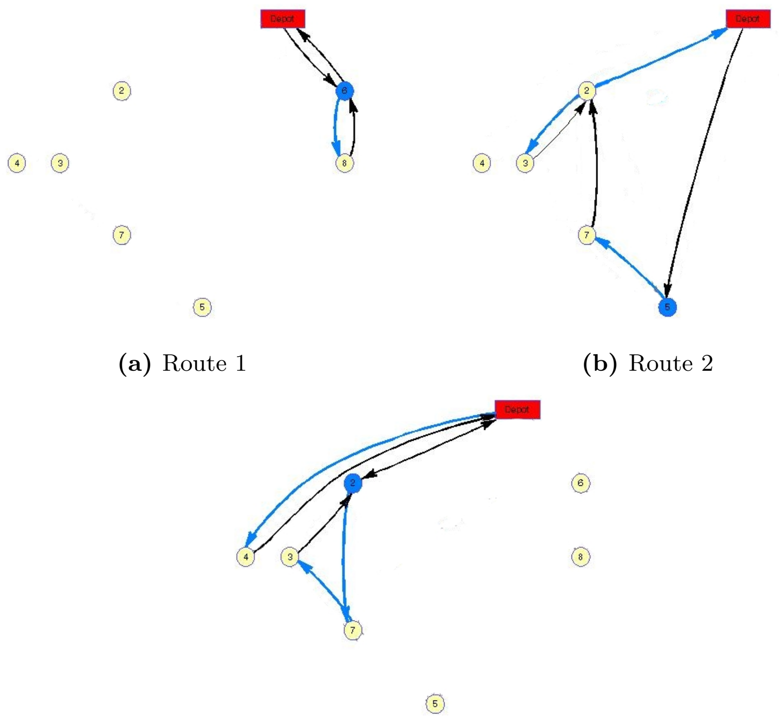


Figure 7.1: A graphical representation of the instance by using the biograph function in MATLAB. Blue vertices, green arcs and blue edges are all required entities. Note that the entities are placed to give a best possible visualization of the data and not correct with respect to actual position of vertices or lengths of the links.

The instance has access to three vehicles. A possible optimal solution can be seen from Figure 7.2.



(a) Route 1 **(b)** Route 2

(c) Route 3

Figure 7.2: Optimal solution to the instance. Blue colour means that the link/vertex is serviced. Black links represent the traversed links.

7.2 The Instances

In this section we will discuss the instances that exist as well as the ones used in this thesis. For the MCGRP there are currently 5 Benchmark instance sets that are used. Prins and Bouchenoua (2005) created the CBMix sets which include 23 instances. Bach et al (2012) developed 2 new benchmarks. The 20 BHW instances based on well known instances from the CARP literature and the 24 DI-NEARP instances taken from real-world newspaper distribution cases in Norway. Bosco et al. (2012) created the two last classes of sets, the MGGDB and the MGVAL. The first class is derived from previous undirected CARP instances by Golden et al. (1983). The sets are modified in the following way. Firstly, $\lceil 0.75|E| \rceil$ of the edges are replaced by pairs of opposite arcs and the demand is moved to a randomly chosen arc. Were $\lceil 0.75|E| \rceil$ means 75% of all edges rounded up to the closest integer. Then, 6 different datasets were created by shifting the demands of $\lceil \beta\pi \rceil$ randomly selected required links to $\lceil \beta\pi \rceil$ randomly selected adjacent vertices, where π is the number of required links in the respective mixed graphs. The 6 sets get the assigned $\beta = \{0.25, 0.30, 0.35, 0.40, 0.45, 0.50\}$. This results in 144 new instances. The second class of datasets is derived from datasets designed for the mixed CARP by Belenguer et al. (2006). These sets are transformed in the similar way as the MGGDB and we get a total of 204 new datasets from this class. The instances, as well as the currently bestbest solutions known from the literature can be found at SINTEF's TOP website (TOP, 2013).

In this thesis the main focus has been on solving the instances from the six MGGDB datasets. These sets will propose a good comparison between the B&C and the B&P based models. Additionally two instances have been chosen from each of the six MGVAL datasets, an instance that was easy and one that was hard to solve for Bosco et al. (2012). To inspire further research also one instance from each of the datasets BHW and CBMix were chosen. None of the DI-NEARP instances were chosen because they are very large.

7.3 The Computational set up

Computational experiments were carried out on a PC equipped with an Intel(R) Core(TM) i7 CPU @2.93GHz, with 8 GB RAM. The full B&P algorithm, including handling the branching and search tree was coded in the C# using Visual studios 2010. All the LP and MILP calculations were solved by using the BCL XPRESS library, release September 13 2012.

Xpress Optimization Suite is a development environment for mathematical modelling and optimization which consists of these three parts:

- IVE, Interactive Visual Environment
- Mosel language, libraries and modules
- Xpress Optimizer

Only the XPRESS Optimizer was used by the algorithm presented in this thesis. Xpress is a high level programming language for optimization which mainly uses Simplex and B&C, to solve linear programming problems.

The MGGDB instances were given a time cap of 60 minutes. Further the MGVAL, BHW and CBMix instances were given 2 hours because the instances were larger. Comparatively Bosco et al. (2012) had a time cap of 6 hours with approximately identical computer specifications.

7.4 Numerical Results

This section bears the fruits from the research performed by this thesis, the results. We start this section with some early results in Chapter 7.4.1. The results by creating all routes and by the CG method will be presented here. In Chapter 7.4.1 the final results from the B&P algorithm will be presented and discussed.

7.4.1 Early Results

The first step in this research was to investigate the path formulation by creating all possible routes. Only 6 of the 144 MGGDB-, and none of the other benchmark instances were solved. For all other instances out computer ran out of memory before all the routes were generated. The 6 instances solved are marked by a $(*)$ in Table 7.1-7.6. The biggest instance that was solved was the one described in detail in Chapter 7.1. Even this small example has almost half a million different routes. It was clear that we could not create all routes so the CG procedure inspired by Letchford and Oukil (2009) was created.

The next results that were produced by this thesis were by using CG. An initial pool of tours were created for a feasible starting point. In routing problems without constraints on the number of vehicles it is common to create as many routes as there are required entities. With each route only servicing one required entity each. This could not be done here because of the vehicle constraints from (5.21). This was solved by a random generator that created new routes until all required entities were covered, meaning that the constraints (5.18)-(5.20) are fulfilled as well as (5.21).

First we present the results in Table 7.1-7.8. Tables 7.1-7.6 present the six sets from the MGGDB instances and Table 7.7 present 12 MGVAL instances. These results are compared to the B&C results from Bosco et al. (2012). Table 7.8 presents 1 BHW- and 1 CBMix instance comparing them to the best known solutions from the literature. None of the MGGDB instances took more than 3 minutes to solve and they had an average solution time of approximately 1 minute in total for the 144 instances. For the bigger instances from Table 7.7 and 7.8 the average solution time was about 10 minutes. With respect to the MGGDB instances, the number of optimally solved instances is equal to 2 for $\beta = \{0.25, 0.30, 0.45\}$, 3 for $\beta = \{0.35, 0.40\}$ and 4 for $\beta = 0.50$; the average percentage gaps are equal to 7.25 for $\beta = 0.25$, 7.20 for $\beta = 0.30$, 5.61 for $\beta = 0.35$, 4.26 for $\beta = 0.40$, 3.73 for $\beta = 0.45$ and 4.58 for $\beta = 0.50$. A total of 19 MGGDB instances gave optimal solutions. For the bigger instances from

Table 7.7 and 7.8 the average GAP was 11.17% and no new optimal solutions were found. The column headings are defined as follows:

- Ins. denotes the instance number in the respective set of instances.
- m denotes the maximum number of vehicles available for each instance
- LB_{CG} and UB_{CG} denote the lower- and upper bounds respectively found by the CG method.
- UB_{BC} denotes the upper bounds found by the B&C method by Bosco et al. (2012), they have not presented lower bounds in their paper.
- GAP_{CG} and GAP_{BC} denotes the percentage gap provided by the respective methods.
- In Table 7.8 the under-case BC is changed to *best* meaning the best known results from the literature.

Table 7.1: CG results from the MGGDB-benchmark set with $\beta = 0.25$

Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}	Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}
1	5	274	285	280	3.86	0	13	6	388	392	388	1.02	3.38
2	6	339	359	349	5.57	5.58	14	5	107	107	107	0	0
3	5	270	282	278	4.26	0	15	4	54	56	55	3.57	0
4	4	273	292	289	6.51	0	16	5	97	169	98	42.6	0
5	6	384	404	394	4.95	6.41	17	5	70	78	71	10.26	0
6	5	291	295	292	1.36	0	18	5	139	157	144	11.46	3.47
7	5	288	290	290	0.69	0	19*	3	53	53	53	0	0
8	10	328	356	-	7.87	-	20	4	116	122	116	4.92	0
9	10	305	331	-	7.85	-	21	6	144	161	146	10.56	0.68
10	4	265	276	265	3.99	0	22	8	160	172	-	6.98	-
11	5	347	366	356	5.19	3.09	23	10	181	234	-	22.65	-
12	7	456	459	459	0.65	3.74							

Table 7.2: CG results from the MGGDB-benchmark set with $\beta = 0.30$

Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}	Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}
1	5	262	276	273	5.07	0	13	6	479	529	486	9.45	8.02
2	6	294	307	301	4.23	5.74	14	5	100	104	101	3.85	0
3	5	269	287	270	6.27	0	15	4	43	44	44	2.27	0
4	4	253	271	260	6.64	0	16	5	105	169	105	37.87	0
5	6	384	416	388	7.69	2.82	17	5	65	75	65	13.33	0
6	5	265	283	276	6.36	0	18	5	142	153	144	7.19	0
7	5	258	281	273	8.19	0	19*	3	51	51	51	0	0
8	10	329	352	-	6.53	-	20	4	94	100	94	6	0
9	10	278	290	-	4.14	-	21	6	121	121	121	0	0.83
10	4	241	248	242	2.82	0	22	8	151	156	-	3.21	-
11	5	382	430	387	11.16	1.55	23	10	167	188	-	11.17	-
12	7	462	472	467	2.12	6.36							

Table 7.3: CG results from the MGGDB-benchmark set with $\beta = 0.35$

Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}	Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}
1	5	236	252	252	6.35	0	13	6	417	417	417	0	2.98
2	6	280	284	284	1.41	0	14	5	83	84	84	1.19	0
3	5	236	243	243	2.88	0	15	4	44	44	44	0	0
4	4	240	256	242	6.25	0	16	5	73	75	75	2.67	0
5	6	299	323	309	7.43	6.11	17	5	61	63	62	3.17	0
6	5	249	262	262	4.96	0	18	5	132	157	135	15.92	0
7	5	263	282	272	6.74	0	19*	3	51	51	51	0	0
8	10	311	338	-	7.99	-	20	4	93	96	96	3.13	0
9	10	260	275	-	5.45	-	21	6	116	125	120	7.2	2.07
10	4	267	306	268	12.75	0	22	8	138	146	-	5.48	-
11	5	301	315	303	4.44	0	23	10	178	233	-	23.61	-
12	7	461	461	461	0	0							

Table 7.4: CG results from the MGGDB-benchmark set with $\beta = 0.40$

Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}	Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}
1	5	273	286	279	4.55	0	13	6	394	407	405	3.19	7.25
2	6	297	309	308	3.88	1.86	14	5	61	66	62	7.58	0
3	5	222	225	225	1.33	0	15	4	37	37	37	0	0
4	4	238	238	238	0	0	16	5	84	90	84	6.67	0
5	6	334	354	344	5.65	6.97	17	5	64	66	65	3.03	0
6	5	263	275	270	4.36	0	18	5	113	125	119	9.6	0
7	5	269	299	282	10.03	0	19*	3	38	38	38	0	0
8	10	322	349	-	7.74	-	20	4	93	94	94	1.06	0
9	10	269	280	-	3.93	-	21	6	104	104	104	0	0
10	4	189	194	191	2.58	0	22	8	129	129	-	0	-
11	5	277	287	283	3.48	2.6	23	10	160	193	-	17.1	-
12	7	403	412	412	2.18	0							

Table 7.5: CG results from the MGGDB-benchmark set with $\beta = 0.45$

Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}	Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}
1	5	257	262	259	1.91	0	13	6	403	407	423	0.98	10.19
2	6	292	298	298	2.01	3.78	14	5	64	67	66	4.48	0
3	5	234	237	237	1.27	0	15	4	34	34	34	0	0
4	4	222	228	228	2.63	0	16	5	70	73	70	4.11	0
5	6	347	370	350	6.22	4.43	17	5	52	53	53	1.89	0
6	5	215	218	218	1.38	0	18	5	115	124	123	7.26	7.19
7	5	225	249	243	9.64	0	19*	3	48	48	48	0	0
8	10	296	316	-	6.33	-	20	4	76	77	78	1.3	0
9	10	274	284	-	3.52	-	21	6	121	128	122	5.47	0
10	4	212	231	214	8.23	0	22	8	135	137	-	1.46	-
11	5	285	306	297	6.86	4.41	23	10	142	146	-	2.74	-
12	7	386	411	393	6.08	5.48							

Table 7.6: CG results from the MGGDB-benchmark set with $\beta = 0.50$

Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}	Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}
1	5	209	223	214	6.28	0	13	6	259	259	259	0	8.52
2	6	259	269	269	3.72	6.81	14	5	74	75	75	1.33	0
3	5	208	221	218	5.88	0	15	4	37	39	37	5.13	0
4	4	216	223	219	3.14	0	16	5	65	67	66	2.99	0
5	6	285	292	292	2.4	0	17	5	53	60	53	11.67	0
6	5	269	279	276	3.58	0	18	5	117	127	121	7.87	6.63
7	5	260	275	265	5.45	0	19*	3	44	44	44	0	0
8	10	305	336	-	9.23	-	20	4	81	81	81	0	0
9	10	260	275	-	5.45	-	21	6	83	98	86	15.31	0
10	4	194	194	194	0	0	22	8	123	124	-	0.81	-
11	5	267	292	275	8.56	4.16	23	10	125	132	-	5.3	-
12	7	440	446	445	1.35	0							

Table 7.7: CG results from the MGVAL-benchmark instances. The instances 1A and 3C from each set.

Ins.	m	LB_{CG}	UB_{CG}	UB_{BC}	Gap_{CG}	Gap_{BC}
$\beta = 0.25$: 1A	2	157	210	177	25.24	0
$\beta = 0.25$: 3C	7	146	154	153	5.19	11.01
$\beta = 0.30$: 1A	2	159	196	170	18.88	0
$\beta = 0.30$: 3C	7	146	165	153	11.52	12.08
$\beta = 0.35$: 1A	2	152	185	158	17.84	0
$\beta = 0.35$: 3C	7	143	149	150	4.03	6.84
$\beta = 0.40$: 1A	2	154	191	165	19.37	0
$\beta = 0.40$: 3C	7	142	152	148	6.58	12.82
$\beta = 0.45$: 1A	2	158	189	168	16.40	0
$\beta = 0.45$: 3C	7	142	146	143	2.74	8.7
$\beta = 0.50$: 1A	2	138	155	145	10.97	0
$\beta = 0.50$: 3C	7	135	145	137	6.90	7.94

Table 7.8: Computational results BHW3 and CBMix1. Comparing to the best known results from the literature

Ins.	m	LB_{CG}	UB_{CG}	UB_{best}	Gap_{CG}	Gap_{best}
BHW 3	-	405	418	415	3.11	24
CBMix 1	-	2513	2718	2589	7.54	7.2

The tables show that we get a clear decrease in percentage gaps when we increase the β. This is likely because we shift more of the required links to vertices. The number of route variables in the path formulation depend on the number of entities, but since edges can be serviced both ways a lot more routes can be created by having more required edges compared to arcs or vertices. This can be seen from the tables, for instance 4 of the instances with $\beta = 0.5$ give optimal solutions compared to only 2 for $\beta = 0.25$. This differs from the results found by Bosco et al. (2012). They conclude that changing β does not affect the solvability of the algorithm and explain this by their B&C algorithm. However they see a clear trend that by increasing the number of vehicles the instances become much harder to solve. They consider instances with $m > 7$ as hard and do not solve these instances in their paper. The reason for this is that the variables in the arc formulation are proportional to the number of vehicles, making it much harder when there are many vehicles. As discussed earlier this is not a problem with our formulation, i.e. instance 22 with $\beta = 0.40$ and $m = 8$ was solved to optimality by the CG algorithm, see Table 7.4.

As one can see from the tables, many of the problems with many vehicles have high percentage gaps, but we believe this is because the instances themselves are much larger and not because of the number of vehicles available. This fact can further be confirmed by Table 7.7. Bosco et al. (2012) has no problems solving the 6 instances with 2 available vehicles, however the gaps on the instances with 7 available vehicles are rather large. The opposite is true for our model. The average gap for the instances with more vehicles is lower than for the instances with fewer vehicles. The number of total- and required entities are higher in the instances with few vehicles, making them easier for our model.

In Table 7.8 we have included the instances from BHW3 and CBMix1. The CG algorithm dramatically improves the percentage gap of the best known solutions to the BHW3 instance ass well as the lower bound for the CBMix instance.

To give an overview that can further compare the CG results to Bosco et al. (2012) we have created Table 7.9. The table gives us information of the total optimal solutions found for the CG- and B&C algorithms respectively. The third column shows the improved lower bounds found by the CG. None of the Upper Bounds were improved, thus this is not in the table. Finally the average GAPs are given for the two methods. Note that some of the 24 extra MGGDB instances solved in this thesis (not included in Bosco et al. (2012)) increase the average GAP significantly.

It is clear from Table 7.9 that the B&C algorithm gives better results than the LP relaxed CG method. More than four times as many solutions were solved to optimum and the average GAP is much better for the B&C algorithm. However a very interesting part of the table are the improved lower bounds. The 48 improved lower bounds correspond to almost all the unsolved instances from the

Table 7.9: Computational results comparing CG with B&C for the MGGDB instances

SET	# Optimal Solutions	# Optimal Solutions BC	# Improved LB	Average GAP CG (%)	Average GAP BC (%)
MGGDB	19	86	48	5.44	1.11

B&C algorithm. The fact that so many lower bounds are improved within 3 minutes compared to the 6 hour cap by Bosco et al. (2012) shows that this is a very good Lower bound procedure. These were the results presented at the WARP1 conference in Copenhagen 22-24 May 2013 (WARP1, 2013).

These results fit well to the prediction by Letchford and Oukil (2009) that a CG method based on elementary routes gives good lower bounds, however they also state that this is a relatively slow procedure. This is not noticable for the MGGDB instances, but start noticing this for the MGVAL instances. The average solution time increases from 1 minute to 10 minutes for the MGGDB- and MGVAL instances respectively.

7.4.2 Branch & Price Results

The final results produced by this thesis are the B&P results. Even though the Lower bounds from the CG method were good, the upper bounds are not good and so the average gap is rather large. By the proof shown in in Chapter 6.2 we know that the branching scheme will provide optimal solutions as the B&C algorithm from Bosco et al. (2012) also does. The interesting part is how many solutions can be found within the time cap of 1 hour.

First we present the results in Table 7.10-7.17. Tables 7.10-7.15 presenting the six sets from the MGGDB instances and Table 7.16 presenting the 12 MGVAL instances. Table 7.17 presents 1 BHW instance and 1 CBMix instance. With respect to the MGGDB instances, the number of optimally solved instances is equal to 12 for $\beta = 0.35$, 13 for $\beta = \{0.30, 0.45\}$, 14 for $\beta = \{0.25, 0.40\}$ and 15 for $\beta = 0.50$; the average percentage gaps are equal to 0.69 for $\beta = 0.25$, 0.89 for $\beta = 0.30$, 1.06 for $\beta = 0.35$, 1.19 for $\beta = 0.40$, 1.15 for $\beta = 0.45$ and 1.32 for $\beta = 0.50$. A total of 81 MGGDB instances gave optimal solutions. Note that Bosco et al (2012) has an error in their table marked with a ($**$), see Table 7.12. This has been confirmed by implementing their model.

The bigger instances have an average GAP = 5.59% and two new optimal solutions were found.

The tables show that we do not get the same decrease in percentage gaps when we increase the β as we saw in the early results. In fact it can seem that it is

Table 7.10: B&P results from the MGGDB-benchmark set with $\beta = 0.25$

Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}	Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}
1	5	280	280	280	0	0	13	6	378	384	388	1.56	3.38
2	6	339	339	349	0	5.58	14	5	107	107	107	0	0
3	5	271	278	278	2.52	0	15	4	55	55	55	0	0
4	4	249	249	249	0	0	16	5	98	98	98	0	0
5	6	386	397	394	2.77	6.41	17	5	71	71	71	0	0
6	5	292	292	292	0	0	18	5	139	140	144	0.71	3.47
7	5	290	290	290	0	0	19	3	53	53	53	0	0
8	10	328	336	-	2.38	-	20	4	116	116	116	0	0
9	10	305	309	-	1.29	-	21	6	144	146	146	1.37	0.68
10	4	265	265	265	0	0	22	8	160	160	-	0	-
11	5	347	356	356	2.53	3.09	23	10	181	181	-	0	-
12	7	456	459	459	0.65	3.74							

Table 7.11: B&P results from the MGGDB-benchmark set with $\beta = 0.30$

Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}	Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}
1	5	263	273	273	3.66	0	13	6	473	474	486	0.21	8.02
2	6	295	301	301	1.99	5.74	14	5	101	101	101	0	0
3	5	270	270	270	0	0	15	4	44	44	44	0	0
4	4	259	260	260	0.38	0	16	5	105	105	105	0	0
5	6	384	388	388	1.03	2.82	17	5	65	65	65	0	0
6	5	270	278	276	2.88	0	18	5	144	144	144	0	0
7	5	260	273	273	4.76	0	19	3	51	51	51	0	0
8	10	328	339	-	3.24	-	20	4	94	94	94	0	0
9	10	279	282	-	1.06	-	21	6	121	121	121	0	0.83
10	4	242	242	242	0	0	22	8	152	152	-	0	-
11	5	382	387	387	1.29	1.55	23	10	167	167	-	0	-
12	7	467	467	467	0	6.36							

Table 7.12: B&P results from the MGGDB-benchmark set with $\beta = 0.35$.

Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}	Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}
1	5	240	252	252	4.76	0	13	6	417	417	417	0	2.98
2	6	280	284	284	1.41	0	14	5	84	84	84	0	0
3	5	237	243	243	2.47	0	15	4	44	44	44	0	0
4	4	242	242	242	0	0	16	5	75	75	75	0	0
5	6	301	309	309	2.59	6.11	17	5	62	62	62	0	0
6	5	262	262	262	0	0	18	5	135	135	135	0	0
7	5	266	272	272	2.21	0	19	3	51	51	47**	0	0
8	10	311	322	-	3.42	-	20	4	96	96	96	0	0
9	10	261	267	-	2.25	-	21	6	118	120	120	1.67	2.07
10	4	268	268	268	0	0	22	8	138	139	-	0.72	-
11	5	301	303	303	0.66	0	23	10	178	182	-	2.2	-
12	7	461	461	461	0	0							

Table 7.13: B&P results from the MGGDB-benchmark set with $\beta = 0.40$

Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}	Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}
1	5	274	279	279	1.79	0	13	6	396	405	405	2.22	7.25
2	6	300	309	308	2.91	1.86	14	5	62	62	62	0	0
3	5	225	225	225	0	0	15	4	37	37	37	0	0
4	4	238	238	238	0	0	16	5	84	84	84	0	0
5	6	334	346	344	3.47	6.97	17	5	65	65	65	0	0
6	5	264	270	270	2.22	0	18	5	113	119	119	5.04	0
7	5	282	282	282	0	0	19	3	38	38	38	0	0
8	10	321	338	-	5.03	-	20	4	94	94	94	0	0
9	10	269	275	-	2.18	-	21	6	104	104	104	0	0
10	4	191	191	191	0	0	22	8	129	129	-	0	-
11	5	276	283	283	2.47	2.6	23	10	160	160	-	0	-
12	7	412	412	412	0	0							

Table 7.14: B&P results from the MGGDB-benchmark set with $\beta = 0.45$

Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}	Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}
1	5	259	259	259	0	0	13	6	423	423	423	0	10.19
2	6	294	298	298	1.34	3.78	14	5	66	66	66	0	0
3	5	237	237	237	0	0	15	4	34	34	34	0	0
4	4	224	228	228	1.75	0	16	5	70	70	70	0	0
5	6	347	350	350	0.86	4.43	17	5	53	53	53	0	0
6	5	218	218	218	0	0	18	5	116	123	123	5.69	7.19
7	5	232	243	243	4.53	0	19	3	48	48	48	0	0
8	10	296	296	-	0	-	20	4	78	78	78	0	0
9	10	273	278	-	1.8	-	21	6	122	122	122	0	0
10	4	214	214	214	0	0	22	8	135	136	-	0.74	-
11	5	286	303	297	5.61	4.41	23	10	143	146	-	2.05	-
12	7	385	393	393	2.04	5.48							

Table 7.15: B&P results from the MGGDB-benchmark set with $\beta = 0.50$

Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}	Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}
1	5	214	214	214	0	0	13	6	259	259	259	0	8.52
2	6	262	269	269	2.6	6.81	14	5	75	75	75	0	0
3	5	218	218	218	0	0	15	4	37	37	37	0	0
4	4	219	219	219	0	0	16	5	66	66	66	0	0
5	6	292	292	292	0	0	17	5	53	53	53	0	0
6	5	271	276	276	1.81	0	18	5	117	121	121	3.31	6.63
7	5	261	268	265	2.61	0	19	3	44	44	44	0	0
8	10	305	328	-	7.01	-	20	4	81	81	81	0	0
9	10	260	267	-	2.62	-	21	6	83	89	86	6.74	0
10	4	194	194	194	0	0	22	8	123	123	-	0	-
11	5	268	278	275	3.6	4.16	23	10	125	125	-	0	-
12	7	445	445	445	0	0							

Table 7.16: B&P results from the MGVAL-benchmark instances. The instances 1A and 3C from each set.

Ins.	m	LB_{BP}	UB_{BP}	UB_{BC}	Gap_{BP}	Gap_{BC}
$\beta = 0.25$: 1A	2	157	188	177	16.49	0
$\beta = 0.25$: 3C	7	146	151	153	3.31	11.01
$\beta = 0.30$: 1A	2	160	186	170	13.98	0
$\beta = 0.30$: 3C	7	146	147	153	0.68	12.08
$\beta = 0.35$: 1A	2	155	173	158	10.40	0
$\beta = 0.35$: 3C	7	144	147	150	2.04	6.84
$\beta = 0.40$: 1A	2	155	174	165	10.92	0
$\beta = 0.40$: 3C	7	142	148	148	4.02	12.82
$\beta = 0.45$: 1A	2	159	170	168	6.47	0
$\beta = 0.45$: 3C	7	142	142	143	0	8.7
$\beta = 0.50$: 1A	2	138	147	145	6.12	0
$\beta = 0.50$: 3C	7	136	136	137	0	7.94

Table 7.17: Computational results for 12 MVAL-, 1 BHW- and 1 CBMix instances. Comparing the Branch & Price results to the Branch & Cut results.

Ins.	m	LB_{BP}	UB_{BP}	UB_{best}	Gap_{BP}	Gap_{best}
BHW 3	-	410	418	415	1.91	24
CBMix 1	-	2529	2575	2589	1.79	7.2

the opposite that happens here, the percentage gaps actually increase from from 0.69-1.32 from $\beta = 0.25$-$\beta = 0.50$. However we find most optimal solutions when $\beta = 0.50$ and by closer inspection we see that two instances have much higher gaps than the rest in this set, 7.01 and 6.74 for instances 8 and 21 respectively. There can be many reasons for particular instances being hard to solve. A random shift of required entities can change an instance for the easier and harder. We come to the conclusion that by using B&P we tend to the conclusion found by Bosco et al. (2012), that changing β does not affect the solvability of the algorithm. More tests should be made on this matter.

It is even more clear here than in Chapter 7.1 that the number of vehicles does not affect the solvability of the B&P. A total of 9 of the instances Bosco et al. (2012) did not give results for were solved to optimality here. Again we also look at the results from the MGVAL instances. The instances with $m = 2$ are hard for the B&P algorithm to solve. It is from Table 7.16 we notice a big difference between the B&P compared to the B&C algorithms. Instance $1A$ with $\beta = 0.25$ is solved to optimality by the B&C algorithm but as a percentage gap of 16.49 for our algorithm while instance $3C$ with $\beta = 0.45$ has a percentage gap of 8.7 by the B&C algorithm, but an optimal solution was found by using B&P.

In Table 7.8 we have included the instances from BHW3 and CBMix1. The B&P algorithm further improves the percentage gaps of the best known solutions for BHW3 and CBMix1. Further research on these instances are advices.

Finally to give an overview that can compare our results to Bosco et al. (2012) we have created Table 7.17. The table gives us information of the total optimal solutions found for the B&P- and B&C algorithms respectively. The third column shows the new optimal solutions found by this thesis. It can be mentioned that additionally 2 more lower bounds were improved, 50 in total. Finally the average GAPs are given for the two methods.

Table 7.18: Computational results BHW3 and CBMix1. Comparing to the best known results from the literature

Set	# Optimal Solutions	# Optimal Solutions BC	# New Optimal Solutions	Average GAP CG (%)	Average GAP BC (%)
MGGDB	81	83	14	1.02	1.11

The results from Table 7.17 show a very significant improvement for the MGGDB instances when using B&P. A total of 81 instances were solved to optimality and 14 of them solved for the very first time. This is almost as many as the 83 solved by the B&C algorithm that used 6 hours. However this means that there are 16 cases solved by the Bosco et al. (2012) that were not solved here. This further shows that different instances might pose difficulties to one of the models and not to the other and vice-versa as mentioned. Further proof for this comes from the fact that Bosco et al. (2012) did not solve 24 of the instances, they were considered to difficult as they had more than seven vehicles. Nine of these 24 instances were solved to optimality by the B&P algorithm. While some instances that where solved to optimum very quickly by Bosco et al. (2012) had a relatively large Gap in this thesis, see for instance number 7 in Table 7.14. It can be many reasons for this, but it is relatively clear that the B&P does not have same problems with the number of vehicles. This can also be seen from the mathematical programs. In the arc flow model, the number of variables is proportional to the number of vehicles. In the Path Model the number of vehicles only acts as a restricting constraint. In fact the more vehicles we have the more relaxed the problem becomes and easier to solve. From our results we are led to a conclusion that the new model solved by B&P can solve many problems and the CG algorithm gives excellent lower bounds. However it is hard to say weather it is better than the B&C from the research done here. The average GAP for the MGGDB instances is smaller for our algorithm, but Table 7.16 hints to better results for B&C when using the MGVAL instances. The two algorithms work good on different problems.

CHAPTER 8

CONCLUSION AND FURTHER WORK

In this thesis we have solved 158 benchmark instances of the MCGRP. We formulated the problem as a *path flow* model formulation and used the ESPPRC as a *pricing problem* in the *CG* method. A novel B&P algorithm was finally implemented to find optimal solutions. The conclusions that are drawn from this thesis will be given in Chapter 8.1. Finally suggestions for further work will be given in Chapter 8.2.

8.1 Conclusion

A novel mathematical model formulation for the MCGRP has been created and presented. As far as we know it is currently the second model for this problem and for the first time the MCGRP has been tackled by a B&P algorithm. The B&P procedure uses ESPPRC as a pricing problem in the CG method and we have used a branching scheme based on *pairing* and *non-pairing* relationships. A proof has been given to show that the chosen branching scheme is complete.

The main goal of this thesis was to create an exact algorithm that was hopefully better than the B&C algorithm by Bosco et al. (2012). The main comparisons were made on the 144 benchmark instances called MCGRP, a total of 81 and 83 instances were solved with percentage average gaps of 1.02 and 1.11 for the B&P- and B&C algorithms respectively. Additionally we found 14 new optimal solutions and improved 50 lower bounds on previously unsolved instances. 12 bigger instances were solved from the MGVAL benchmark instances and 2 new optimal solutions were found. Dramatically improvements were made on the benchmark instances BHW3 and CBMix1 were the percentage gaps were reduced from 24 to 1.91 and 7.2 to 1.79 for the BWH3 and CBMix1 instances respectively.

The algorithm created in this thesis has some weaknesses for the bigger instances with few vehicles. Some of the problems that Bosco et al. (2012) could solve within few seconds we ended up with percentage gaps more than 10 after 2 hours computation time. This fits well with the results found by Letchford and Oukil (2011) that pricing with elementary paths is slow.

8.2 Further Work

As mentioned earlier it has been written very little about exact methods for the MCGRP and there is still an unknown limit of extensions that can be made. We saw in this thesis that pricing with elementary paths gave relatively good lower bounds, but noticed the limitations when we solved the bigger instances. The various pricing methods from Table 6.1 should be investigated.

As mentioned in Chapter 3.5, it can be hard to find a complete branching scheme for the B& P environment. For other pricing problems new schemes need to be developed. This is absolutely worth investigating as it has been for the CARP by Bode and Irnich (2011). More inspiration on the matter can be found in the research done by Bode and Irnich (2013).

Procedures for adding cuts can be implemented. Bode and Irnich (2011) add initial cuts to their B&P algorithm creating a Cut first, B&P second procedure which also is possible to implement for the MCGRP. Cuts can also be added at each branching iteration which would create a Branch, Cut and Price procedure. This can potentially combine the strengths from both the B&P and the B&C algorithms discussed in this thesis.

In this thesis a time cap was put on the algorithm. This time cap can be increased to see if better solutions are obtained, but it is likely that small improvements will require high increase in computational time. An improvement can be to use parallelization for the implementation. The different nodes at the same level in the B&P search tree are independent and can easily be solved by different processes.

Summarizing for improvements of the B&P we can,

- investigate alternative pricing problems and alternative branching schemes,
- create cuts and investigate the Branch, Cut and Price method and
- increase time cap and/or parallelize implementation.

Lastly heuristic solution methods can be noted. This topic has only been mentioned in this thesis since the focus has been on exact algorithms, however the population and technology growth is huge and real life problems become bigger and more complex continuously. As has been seen in this thesis there are limitations to exact optimization methods so heuristics methods may be necessary to solve real life real sized problems.

BIBLIOGRAPHY

[1] Top web pages. http://www.sintef.no/TOP. Accessed: 29/06/2013.

[2] 1st workshop on arc routing problems. http://econ.au.dk/warp-1/, May 2013. Accessed: 29/06/2013.

[3] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[4] L. Bach, G. Hasle, and S. Wøhlk. A lower bound for the node, edge, and arc routing problem. *Computers & Operations Research*, 2012.

[5] R. Baldacci and V. Maniezzo. Exact methods based on node-routing formulations for undirected arc-routing problems. *Networks*, 47(1):52–60, 2006.

[6] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

[7] C. Bode and S. Irnich. Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations Research*, 60(5):1167–1182, 2012.

[8] C. Bode and S. Irnich. The shortest-path problem with resource constraints with (k, 2)-loop elimination and its application to the capacitated arc-routing problem. Technical report, Technical Report LM-2013-01, Chair of Logistics Management, Mainz School of Management and Economics, Johannes Gutenberg University, Mainz, Germany. Available at http://logistik. bwl. uni-mainz. de/158. php, 2013.

[9] A. Bosco, D. Laganà, R. Musmanno, and F. Vocaturo. Modeling and solving the mixed capacitated general routing problem. *Optimization Letters*, pages 1–19, 2012.

[10] O. Bräysy, E. Martínez, Y. Nagata, and D. Soler. The mixed capacitated general routing problem with turn penalties. *Expert Systems with Applications*, 38(10):12954–12966, 2011.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2001.

[12] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 80(6), 1959.

[13] M. Dror. *Arc routing: theory, solutions, and applications*. Kluwer Academic, 2000.

[14] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, 44(3):216–229, 2004.

[15] B. Golden, S. Raghavan, and E. Wasil. The vehicle routing problem: Latest advances and new challenges, vol. 43 of operations research. *Computer Science Interfaces, Springer*, 2008.

[16] B. L. Golden, J. S. DeArmon, and E. Baker. Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10(1):47–59, 1983.

[17] B. L. Golden and R. T. Wong. Capacitated arc routing problems. *Networks*, 11(3):305–315, 1981.

[18] J. C. A. Gutiérrez, D. Soler, and A. Hervás. The capacitated general routing problem on mixed graphs. *Revista Investigacion Operacional*, 23(1):15–26, 2002.

[19] Y. H.D. and F. R.A. *University Physics*. Pearson, 2008.

[20] K. Jansen. Bounds for the general capacitated routing problem. *Networks*, 23(3):165–173, 1993.

[21] G. A. Kochenberger et al. *Handbook of metaheuristics*. Springer, 2003.

[22] H. Kokubugata, A. Moriyama, and H. Kawashima. A practical solution using simulated annealing for general routing problems with nodes, edges, and arcs. In *Engineering Stochastic Local Search Algorithms. Designing,*

Implementing and Analyzing Effective Heuristics, pages 136–149. Springer, 2007.

[23] A. N. Letchford and A. Oukil. Exploiting sparsity in pricing routines for the capacitated arc routing problem. *Computers & Operations Research*, 36(7):2320–2327, 2009.

[24] H. Longo, M. P. de Aragão, and E. Uchoa. Solving capacitated arc routing problems using a transformation to the cvrp. *Computers & Operations Research*, 33(6):1823–1837, 2006.

[25] J. Lundgren, M. Rönnqvist, and P. Värbrand. Optimization. studentlitteratur. *Lund, Sweden*, 2010.

[26] J. Nocedal and S. J. Wright. *Numerical optimization.* Springer Science+ Business Media, 2006.

[27] R. Pandi and B. Muralidharan. A capacitated general routing problem on mixed networks. *Computers & operations research*, 22(5):465–478, 1995.

[28] C. Prins and S. Bouchenoua. A memetic algorithm solving the vrp, the carp and general routing problems with nodes, edges and arcs. In *Recent advances in memetic algorithms*, pages 65–85. Springer, 2005.

[29] A. Schrijver. On the history of combinatorial optimization (till 1960). handbook of discrete optimization, 1–68, 2005.

[30] L. A. Wolsey. Integer programming. *IIE Transactions*, 32, 2000.

APPENDIX A

ABSTRACT FROM WARP1

This is an abstract handed in to the WARP1 conference from Copenhagen 22-24 May 2013.

Column Generation for the Mixed Capacitated General Routing Problem

Kevin Anders Gaze

Department of Mathematics, Norwegian University of Science and Technology

Email: gaze@stud.ntnu.no

Geir Hasle

Department of Applied Mathematics, SINTEF ICT, Oslo, Norway

Carlo Mannino

Department of Applied Mathematics, SINTEF ICT, Oslo, Norway

We study the Mixed Capacitated General Routing Problem (MCGRP) in which a fleet of capacitated vehicles has to serve a set of requests by traversing a mixed weighted graph. The requests may be located on nodes, edges, and arcs. This problem generalizes the classical Capacitated Vehicle Routing Problem (CVRP), the Capacitated Arc Routing Problem (CARP), and the General Routing Problem, and captures important aspects of real life situations that cannot be properly modeled by VRP or CARP.

Pandit and Muralidharan [6] and Gutiérrez et al. [3] propose heuristics for variants of the MCGRP. Prins and Bouchenoua [7] studied the MCGRP under the alias The Node, Edge, and Arc Routing Problem (NEARP). They define a benchmark (CBMix), and propose a memetic algorithm that provided the first upper bounds for the CBMix and also good results for well-known CVRP and CARP instances. Kokubugata et al. [4] develop a metaheuristic with several new best CBMix upper bounds. The first lower bounding procedure for the NEARP was proposed by Bach et al. [1]. They also define two new MCGRP benchmarks: the BHW based on well known instances from the CARP literature, and the DI-NEARP taken from real-world newspaper distribution cases in Norway. Bosco et al. [2] propose the first integer programming formulation for the NEARP and develop a branch-and-cut algorithm that was tested on 12 new sets of instances derived from CARP benchmarks, as well as small CBMix instances, providing two optimal solutions.

We describe ongoing work towards exact methods for the MCGRP based on column generation. Inspired by Letchford and Oukil's work on the CARP [5], we investigate a model with an elementary shortest path subproblem. Preliminary results include new and better lower bounds for several of the Bosco et al. instances.

References

[1] L. Bach, G. Hasle, S. Wøhlk. "A Lower Bound for the Node, Edge, and Arc Routing Problem". Computers & Operations Research 40 (2013) 943–952.

[2] A. Bosco, D. Lagana, R. Musmanno, F. Vocaturo, "Modeling and solving the mixed capacitated general routing problem", Optimization Letters (forthcoming).

[3] J.C.A. Gutiérrez, D. Soler, and A. Hérvas. "The capacitated general routing problem on mixed graphs". Revista Investigacion Operacional, 23:15–26, 2002.

[4] H. Kokubugata, A. Moriyama, and H. Kawashima. "A practical solution using simulated annealing for general routing problems with nodes, edges, and arcs". In "Engineering Stochastic Local Search Algorithms" volume 4638, pages 136–149.

[5] A.N. Letchford and A. Oukil, "Exploiting sparsity in pricing routines for the capacitated arc routing problem". *Computers & Operations Research,* **36(7)**, 2320-2327, 2009

[6] R. Pandit and B. Muralidharan. "A capacitated general routing problem on mixed networks". Computers & Operations Research, 22:465–478, 1995.

[7] C. Prins and S. Bouchenoua. "A memetic algorithm solving the VRP, the CARP and general routing problems with nodes, edges and arcs". In Recent Advances in Memetic Algorithms, volume 166, pages 65–85. Springer Berlin / Heidelberg, 2005.