Utrecht University

Department of Information and Computing Sciences

Benders' Decomposition vs. Column & Constraint Generation, a Closer Look

Master Thesis

Author:
Abel Komen
Utrecht University

First Supervisor:
Dr. J. A. Hoogeveen

Second Supervisor:
Dr. Ir. J. M. van den Akker

June 6, 2017

Abstract

In this thesis an attempt is made to find out how the type of uncertainty (discrete and finite or polyhedral) influences performance of Benders' decomposition [4] and Column & Constraint Generation [24] when solving the demand robust location-transportation problem. A generalization of Benders' decomposition is presented to make it applicable to a large group of demand robust optimization problems. Also, Column & Constraint Generation is adapted to be used on discrete and finite uncertainty sets. In [24] it was shown that Column & Constraint Generation is able to solve the problem a lot better than a standard implementation of Benders' decomposition. The performance comparison for discrete and finite uncertainty sets made in this thesis is new. On top of that, a number of techniques for making Benders' faster are applied. Special attention is paid to the role of the MIP-solver that is used as a black box for both algorithms.

Contents

Chapter 1

Introduction

Real world optimization problems often include uncertainty in some form. The easiest way to deal with this uncertainty is to use a deterministic model whose solutions are close enough to the real outcome. Unfortunately this is not always possible or it could be that capturing the uncertainty in an optimization problem yields superior results. A number of ways to do this will be covered here.

There is ample literature regarding real world optimization problems that deal with uncertainty. In the field of robustness a couple of examples are problems in railway related optimization problems ([8], [16]), airline scheduling problems problems ([19]) and problems regarding electricity grid stability ([1], [14], [15], [23], [25]).

To illustrate the possibilities, consider the cutting-stock problem. The cutting-stock problem concerns the divisions of some standard-sized pieces of supplied material into pieces of a specific size. The objective is to minimize the waste that is left over after the demand is fulfilled. This problem could for example be encountered in a company that is supplied with pieces of sheet metal of a fixed size that have to be cut into numerous pieces of a specific size. The company has m different sizes, and requires q_j pieces of size with index $j \in \{1, \ldots, m\}$. Each standard-sized piece can be cut using a so-called pattern. The company has n patterns and for each pattern $i \in \{1, \ldots, n\}$ parameter a_{ij} indicates the number of pieces of size j it produces. Also, each pattern i has waste c_i associated with it. The variable x_i indicates how often pattern i is utilized. Of course, x_i is integer. The deterministic version of cutting stock is given by
$\min_{x \in \{0,1,\ldots\}} \{\sum_{i=1}^{n} c_i x_i | \sum_{i=1}^{n} a_{ij} x_i \geq q_j \quad \forall j \in \{1, \ldots, m\}\}$.

In the cutting-stock problem, uncertainty can arise in a number of ways. It could be that demand q_j for the cuts is unknown at the time of decision making or that the number of cuts a_{ij} produced by a pattern is uncertain (due to low quality of supplied material for example). Also, the amount of waste produced by pattern i can be uncertain. For ease of exposition, assume that the uncertainty can be adequately captured by some K scenarios {1, …, K}. This means that scenario $k \in \{1, , \ldots, K\}$ is a set of parameters (c^k, q^k, a^k)

The most strict way to deal with this uncertainty is given by *robust opti-*

mization. In robust optimization the chosen solution has to be feasible for all possible scenarios and the objective is to minimize the costs of the most expensive scenario. So the robust optimization version of cutting stock is given by $\min_{x\in\{0,1,\dots\}}\{\max_{k\in\{1,\dots,K\}}\{\sum_{i=1}^n c_i^k x_i \mid \sum_{i=1}^n a_{ij}^k x_i \geq q_j^k \, \forall j \in \{1,\dots,m\}\}\}$.

Robust optimization is a one stage approach to handling uncertainty, because all decisions have to be made at once before the outcome of the uncertain processes is known and it is not possible to adjust the solution afterwards. On the other hand, there are also two stage approaches. These allow for a decision maker to observe the outcome of his plans and adjust where necessary. For cutting-stock this means that it is possible to make a plan and check whether all demands are met after the plan is executed and the results can be checked. If not, some more pieces of material can be cut.

A well known two stage approach is *stochastic programming.* The goal of two stage stochastic programming is to minimize the cost of the first stage decisions plus the average cost over all scenarios of the second stage decisions. Let the first stage decisions be called y_i. The second stage decisions for scenario k are called x_i^k. The cost for the first stage decisions are denoted by d_i and to avoid a trivial situation where all decisions can be postponed to the second stage it is necessary that $d_i < c_i^k$. The two stage stochastic programming version of cutting-stock is given by

$$\begin{aligned}
\min_{y,x^k} \quad & \sum_{i=1}^n d_i y_i + \frac{1}{K}\sum_{k=1}^K \sum_{i=1}^n c_i^k x_i^k \\
\text{s.t.} \quad & \sum_{i=1}^n a_{ij}^k (y_i + x_i^k) \geq q_j^k \quad \forall j \in \{1,\dots,m\}, k \in \{1,\dots,K\} \\
& y, x^k \geq 0 \, \text{and integer}
\end{aligned}$$

A middle ground between stochastic programming and robust optimization is presented by *recoverable robustness* ([3], [16]). Recoverable robustness is like two stage stochastic programming in that decisions can be made before and after it is known what the scenario is. However, the value to be optimized is not the first stage costs plus the expected second stage costs, but the first stage costs plus the cost of the worst case scenario in the second stage, which makes it look more like robust optimization. Also, the algorithm to determine the second stage decisions should be easy and the allowed actions should be limited. In this case the set of patterns could be limited to the patterns used in the first stage and the recovery problem becomes a cutting stock problem with fixed patterns. A recoverable robust version of cutting-stock is given by

$$\begin{aligned}
\min_{y} \quad & \sum_{i=1}^{n} d_i y_i + \max_{k} \min_{x^k} \sum_{i=1}^{n} c_i^k x_i^k \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_{ij}^k (y_i + x_i^k) \geq q_j^k \quad \forall j \in \{1, \ldots, m\}, k \in \{1, \ldots, K\} \\
& y, x^k \geq 0 \, \text{and integer}
\end{aligned}$$

This thesis focuses on the narrower concept of *demand robustness* as presented in [11]. In demand robustness the uncertainty is only in the right-hand-side (so the demand) of some subset of constraints that need to be satisfied. Two examples of these kind of problems are given in Chapter 2: the demand robust minimum-cut ([11]) problem and the robust facility location problem ([24]). A demand robust version of cutting stock is given by

$$\begin{aligned}
\min_{y} \quad & \sum_{i=1}^{n} d_i y_i + \max_{k} \min_{x^k} \sum_{i=1}^{n} c_i x_i^k \\
\text{s.t.} \quad & \sum_{i=1}^{n} a_{ij} (y_i + x_i^k) \geq q_j \quad \forall j \in \{1, \ldots, m\}, k \in \{1, \ldots, K\} \\
& y, x^k \geq 0 \, \text{and integer}
\end{aligned}$$

A variety of optimization techniques can be applied to demand robust optimization problems. The optimization algorithm of choice is highly dependent on the way the problem is modelled. In this thesis I will focus on two families of optimization techniques:

1. Benders' decomposition [4] (see Chapter 3)

2. Column & constraint generation [24] (see Chapter 4)

Both techniques rely on decomposition of problems into a 'hard' and an 'easy' part and are therefore a natural choice to solve two stage demand robust problems. Benders' decomposition has been applied to optimization problems under uncertainty in for example [6] and [13]. Column & constraint generation has been applied in [1], [14], [15], [23] and [25].

In [24] the performance of the column & constraint generation algorithm is compared to that of Benders' decomposition on a robust location-transportation problem. The column & constraint generation algorithm appears to perform much better. However, part of this advantage could be caused by a sub-optimal implementation of Benders decomposition. Since its publication numerous methods of making Benders' faster have been published, such as a better cut selection ([17]) and faster starting procedures ([18]). Also, column & constraint generation may be at an advantage in [24] due to the nature of the uncertainty in that paper. A polyhedral scenario set is used which could put Benders' at a disadvantage because the problem no longer can be solved by a classical Benders' but has to be solved by a variant of the algorithm.

Therefore, in this thesis Benders' decomposition will be compared to column and constraint generation for demand robust location-transportation with a discrete scenario set and with a polyhedral uncertainty set. This will hopefully shed some light on the strengths and weaknesses of both algorithms and could help with future algorithm selection.

Also, the problem with discrete uncertainty sets is solved by a standard Gurobi MIP-solver to compare the performance of the specialized algorithms to an off-the-shelf standard solver. The results of these experiments can be found in Chapter 5.

In Chapter 6 we will take a closer look at how to interpret the results of the computational experiments.

Chapter 2

Demand Robust Optimization

This chapter provides a detailed look at demand robust optimization problems and how to model them using (mixed integer) linear programs. According to [3] real-world optimization problems often have the following properties:

- The data are not exactly known.
- Small data perturbations can heavily affect feasibility/optimality properties of solutions.
- Efficient solution methods are necessary because data and decision vectors are large.

In [3] Ben-Tal et al. describe two ways of dealing with the listed properties using uncertain linear programs: *Robust Optimization* (RO) and *Adjustable* RO. The uncertainty of parameters is captured in an *uncertainty set* that contains all possible realizations of the uncertain parameters. The definition of an uncertain linear program for some uncertainty set $\mathcal{Z}$ is

$$\left\{ \min_{x \geq 0} \{cx | Ax \geq u\} \right\}_{[A,u,c] \in \mathcal{Z}} \tag{2.1}$$

In robust optimization feasibility and objective value of a solution x are affected by the uncertain parameters in uncertainty set $\mathcal{Z}$. The aim is to find an optimal solution x^* that is feasible for all possible realizations in $\mathcal{Z}$ and minimizes the maximal cx among all corresponding realizations of c. This leads to the *Robust Counterpart* (RC, [3]) of 2.1:

$$\min_{x \geq 0} \left\{ \max_{[A,u,c]} \{cx | Ax \geq u, [A,u,c] \in \mathcal{Z}\} \right\} \tag{2.2}$$

The RC formulation requires that a solution is feasible for all realizations of $\mathcal{Z}$. In practice, this might not be strictly necessary. It could be that the value of some decision variables can be chosen after the uncertainty is lifted. From now one, variables that have to be chosen before the uncertainty is lifted will be denoted by y and variables that can be chosen after the uncertainty is lifted will be marked x, unless explicitly stated otherwise. The variables y are called *non-adjustable* or *first stage* and the variables x are called *adjustable* or *second stage*. It is possible that there are constraints that only affect the y variables. The set Y will be the set of allowed values of y and is defined by $Y = \{y|Dy \geq e\} \cap \mathbb{N}$ (throughout this report, the symbol $\mathbb{N}$ will be used to denote the natural numbers including 0). The *Adjustable* RC (ARC, [3]) can now be formulated as

$$\min_{y\in Y}\{dy|Ax + By \geq u; x \geq 0\}_{[A,B,u,d]\in\mathcal{Z}} \tag{2.3}$$

The matrix A is called the *recourse* matrix. The problem is called a *fixed recourse* problem when A is not uncertain. The vector u is called the *demand vector*.

The concept of *demand robustness* (DR) is introduced in [11]. Whitout calling it demand robustness, [24] also gives a description of demand robustness. The description provided here is modeled after the description in [24] but continuing the notation used in the beginning of [3]. A DR problem aims to optimize the total cost of the non-adjustable variables y plus the worst case costs of the adjustable variables x while only the demand vector u is uncertain. To fit this into the framework of 2.3 a variable η is added to the non-adjustable variables accompanied by constraints of the form $\eta \geq cx$. In this case, the costs of the second stage decisions (captured by c) can also vary. The uncertainty set $\mathcal{Z}$ is renamed U to emphasize this property and furthermore [11] only deals with uncertainty sets that consist of *scenarios*, meaning that U is a discrete and finite set. So $U = \{u^1, \ldots, u^k\}$. The discrete and finite nature of U allows for the introduction of a separate vector of variables x^k to be introduced for every scenario u^k.

A general DR problem in the style of 2.3 can be formulated as follows:

$$\min_{y\in Y,\eta}\{dy + \eta|Ax^k + By \geq u^k, \eta \geq c^k x^k, x^k \geq 0\}_{u^k\in U} \tag{2.4}$$

A different kind of uncertainty set for demand robust problems was introduced in [5] where the uncertain values are not in a discrete and finite set but in a special type of polyhedron. Every element u_j of $u \in U$ has a base value $\underline{u}_j$ and some extra demand of at most $\bar{u}_j$ can be added to that. So every u_j takes a value in $\underline{u}_j + g_j\bar{u}_j$ with $g_j \in [0, 1]$. On top of that there is a limit Γ to the amount of total deviation from the base values which is enforced by the constraint $\sum_j g_j \leq \Gamma$. The uncertainty set can now be defined as

$$U = \{u|\underline{u} + g \odot \bar{u}, \mathbf{1}g \leq \Gamma\} \tag{2.5}$$

Vector $\mathbf{1}$ denotes the row vector of ones $(1 \cdots 1)$ and $\odot$ is element-wise multiplication of two vectors of the same length.

A DR problem with an uncertainty set as in 2.5 can be summarized as

$$\min_{y \in Y} \left\{ dy + \max_{u \in U} \min_{x \geq 0} \{cx | Ax \geq u - By\} \right\} \tag{2.6}$$

There is a large difference in solving 2.4 and 2.6. The discrete and finite uncertainty set of 2.4 allows the problem to be formulated as one linear program. This is not the case for 2.6. The way to solve the problem for some fixed $\bar{y}$ can be found in [13] and is repeated here. How to use this to solve the complete problem is explained in the chapters on Benders' decomposition (Chapter 3) and Column & Constraint Generation (Chapter 4).

For fixed $\bar{y}$, problem 2.6 reduces to ([13])

$$\begin{aligned} \max_{u \in U} \min_{x} \quad & cx && (2.7) \\ \text{s.t.} \quad & Ax \geq u - B\bar{y} && (2.8) \\ & x \geq 0 && (2.9) \end{aligned}$$

Taking the dual of the inner minimization of 2.7 - 2.9 and combining the maximization problems yields

$$\begin{aligned} \max_{u,\pi} \quad & \pi(u - B\bar{y}) && (2.10) \\ \text{s.t.} \quad & \pi A \leq c && (2.11) \\ & u \in U, \quad \pi \geq 0 && (2.12) \end{aligned}$$

Using the uncertainty set defined by 2.5 leads to

$$\begin{aligned} \max_{g,\pi} \quad & \pi\underline{u} + \pi(g \odot \bar{u}) - \pi B\bar{y} && (2.13) \\ \text{s.t.} \quad & \pi A \leq c && (2.14) \\ & \mathbf{1}g \leq \Gamma && (2.15) \\ & g \in \{0,1\}, \quad \pi \geq 0 && (2.16) \end{aligned}$$

Note that $g \in \{0,1\}$ and not $g \in [0,1]$ because an optimal solution will always be in an extreme point of the uncertainty set when Γ is integer ([13], [24]).

Problem 2.13 - 2.16 contains the quadratic terms $\pi(g \odot \bar{u})$. Due to these terms the problem cannot be solved by a standard MIP-solver. These quadratic terms can be eliminated by introducing a new variable vector ω such that $\omega_j = \pi_j$ when $g_j = 1$ and 0 otherwise. Because $\bar{u} \geq 0$ this can be achieved by the constraints $\omega \leq \pi$ and $\omega \leq Mg$ for some sufficiently large real number M. This gives the following problem

$$\max_{g,\pi} \quad \pi\underline{u} + \omega\bar{u} - \pi B\bar{y} \tag{2.17}$$
$$\text{s.t.} \quad \pi A \leq c \tag{2.18}$$
$$\omega \leq \pi \tag{2.19}$$
$$\omega \leq Mg \tag{2.20}$$
$$\mathbf{1}g \leq \Gamma \tag{2.21}$$
$$g_j \in \{0,1\}\forall j, \quad \pi \geq 0, \quad \omega \geq 0 \tag{2.22}$$

The solution to Problem 2.17 - 2.22 can be used for Column & Constraint Generation or Benders' decomposition.

2.1 Problem Examples

2.1.1 Demand Robust Min-Cut

A simple example of a demand robust problem is the robust minimum-cut problem ([11]). The regular min-cut problem consists of a directed graph $G = (V, A)$ with a root vertex r and a terminal vertex t. A set of arcs $A' \subseteq A$ is to be removed such that r and t are no longer connected in $G' = (V, A - A')$. Every edge $a \in A$ has costs c_a and the goal is to minimize the costs $\sum_{a \in A'} c_a$. The linear programming formulation of this problem has a variable x_a for all $a \in A$ that is 1 if $a \in A'$ and 0 otherwise. For all $v \in V$ there is an additional variable p_v that is 1 if v is connected to the root r and 0 otherwise. This is enforced by a constraint that makes sure $p_i - p_j = 0$ when vertices i and j are connected, so $x_{(i,j)} - (p_i - p_j) \geq 0$ for all $(i,j) \in E$. Disconnecting the root from the terminal translates to constraint $p_r - p_t \geq 1$. A linear program is given by

$$\min \quad \sum_{(i,j) \in A} c_{(i,j)} x_{(i,j)} \tag{2.23}$$
$$\text{s.t.} \quad x_{(i,j)} - (p_i - p_j) \geq 0 \qquad \forall (i,j) \in A \tag{2.24}$$
$$p_r - p_t \geq 1 \tag{2.25}$$
$$p_i \geq 0 \quad \forall i \in V \tag{2.26}$$
$$x_{(i,j)} \geq 0 \quad \forall (i,j) \in A \tag{2.27}$$

Problem 2.23 - 2.27 can be expanded by the realization that implicit constraints $p_r - p_i \geq 0$ for all $i \neq t$ are left out. This makes sense for the deterministic version of the problem. Mentioning them explicitly makes it easier to see how this problem can be cast into the demand robust framework of 2.4. The demand robust version of min-cut is such that the terminal vertex t is uncertain and it is cheaper to select edges before the terminal is known. Due to the nature of this problem (a vertex is the terminal or not) there is only a demand robust version of this problem with a discrete and finite uncertainty set. For every edge there is a variable $y_{(i,j)}$ that indicates if (i, j) is selected before the terminal is known. For every edge in every scenario k there is a variable $x^k_{(i,j)}$ that indicates if edge (i, j) is selected after the terminal is revealed in scenario k. Selecting (i, j) before the terminal is revealed costs $c_{(i,j)}$ and selecting (i, j) after the terminal is known costs $\sigma^k c_{(i,j)}$ in scenario k for some $\sigma^k \geq 1$. The uncertain demand vector consists of the right-hand sides of constraints $p_r - p^k_i$ with $p_r - p^k_i \geq 0$ if $i \neq t$ in scenario k and 1 otherwise. If t^k denotes the terminal in scenario k the demand robust equivalent of 2.23 - 2.27 is

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in E} c_{(i,j)} y_{(i,j)} + \eta && && (2.28)\\
\text{s.t.} \quad & y_{(i,j)} + x^k_{(i,j)} - (p^k_i - p^k_j) \geq 0 && \forall (i,j) \in E, \quad \forall k && (2.29)\\
& p_r - p^k_{t^k} \geq 1 && \forall k && (2.30)\\
& \eta - \sum_{(i,j)\in E} \sigma^k c_{(i,j)} x^k_{(i,j)} \geq 0 && \forall k && (2.31)\\
& p^k_i \geq 0 \quad \forall i \in V, \quad \forall k && && (2.32)\\
& x^k_{(i,j)} \geq 0 \quad \forall (i,j) \in E, \quad \forall k && && (2.33)
\end{aligned}$$

2.1.2 Demand Robust Location-Transportation

The formulation of the robust location-transportation problem in this section is due to [24]. The idea is that some commodity has to be supplied to a set of customers. Before delivery it will be stored at a number of potential warehouses. Each warehouse has a fixed building cost and variable costs for the chosen capacity. For each warehouse there is a maximal possible capacity. Supplying a customer from a warehouse also carries with it a variable cost. Each customer has an uncertain demand.

To formulate this problem as a mixed integer program each possible warehouse has binary variable y_i to indicate whether it is operational and continuous variable z_i to indicate its chosen capacity. Continuous variable x_{ij} indicates the amount that is supplied to customer j from warehouse i. The parameters are f_i and z_i for the fixed and variable cost of warehouse i respectively and c_{ij} for the variable cost of supplying customer j from warehouse i. Each customer has demand u_j and the largest possible capacity for warehouse i is K_i. The deterministic version of this problem is given by

$$\min_{(y,z,x)} \quad \sum_i (f_i y_i + a_i z_i) + \sum_{ij} c_{ij} x_{ij} \tag{2.34}$$

$$\text{s.t.} \quad z_i \leq K_i y_i \tag{2.35}$$

$$\sum_j x_{ij} \leq z_i \tag{2.36}$$

$$\sum_i x_{ij} \geq u_j \tag{2.37}$$

$$y_i \in \{0,1\}, \quad z_i \geq 0, \quad x_{ij} \geq 0 \tag{2.38}$$

In the demand robust version of this problem the variables y_i and z_i are the non-adjustable or first stage variables and the variables x_{ij} are the adjustable or second stage variables. The uncertain demand vector u is drawn from a polyhedral uncertainty set U defined as in 2.5. Let u_{max} be the largest possible total demand for all scenarios. The constraint $\sum_i z_i \geq u_{max}$ is added to make sure that the second stage problem is always feasible. The demand robust location-transportation problem is given by

$$\min_{(y,z,x)} \quad \sum_i (f_i y_i + a_i z_i) + \max_{u \in U} \min \sum_{ij} c_{ij} x_{ij} \tag{2.39}$$

$$\text{s.t.} \quad z_i \leq K_i y_i \qquad \forall i \tag{2.40}$$

$$\sum_i z_i \geq u_{max} \tag{2.41}$$

$$\sum_j x_{ij} \leq z_i \qquad \forall i \tag{2.42}$$

$$\sum_i x_{ij} \geq u_j \qquad \forall j \tag{2.43}$$

$$y_i \in \{0,1\}, \quad z_i \geq 0, \quad x_{ij} \geq 0 \tag{2.44}$$

For a discrete and finite uncertainty set $U = \{u^1, \ldots, u^K\}$ of size K the problem is

$$\min_{(y,z,\eta)} \quad \sum_i (f_i y_i + a_i z_i) + \eta \tag{2.45}$$

$$\text{s.t.} \quad z_i \le K_i y_i \qquad \forall i \tag{2.46}$$

$$\sum_i z_i \ge u_{max} \tag{2.47}$$

$$\sum_j x_{ij}^k \le z_i \qquad \forall i, k \tag{2.48}$$

$$\sum_i x_{ij}^k \ge u_j^k \qquad \forall j, k \tag{2.49}$$

$$\eta - \sum_{ij} c_{ij} x_{ij}^k \ge 0 \qquad \forall k \tag{2.50}$$

$$y_i \in \{0,1\}, \quad z_i \ge 0, \quad x_{ij}^k \ge 0 \tag{2.51}$$

Chapter 3

Benders' Decomposition

Benders' Decomposition as described here is a method to solve mixed integer programming problems by splitting the original problem into a master problem that contains the 'hard' (integer) variables and a subproblem that contains the 'easy' (continuous) variables. The master and subproblem are iteratively solved until a solution is found.

Benders' decomposition was first published in [4] and has since been successfully applied to a wide range of integer linear programs and mixed integer programs. It is particularly well suited for network design problems [9], but has also been applied to, for example, airline scheduling [19]. The following explanation of Benders' decomposition is based on [9]. The general problem that is suited for being solved by Benders' decomposition is as follows:

$$\begin{aligned} \min \quad & cx + dy && (3.1)\\ \text{s.t.} \quad & Ax + By \geq b && (3.2)\\ & Dy \geq e && (3.3)\\ & x \geq 0, \quad y \in \mathbb{N} && (3.4) \end{aligned}$$

For ease of exposition the set Y is defined as $Y = \{y | Dy \geq e, y \in \mathbb{N}\}$. Problem 3.1 - 3.4 can also be written as

$$\min_{y \in Y}\{dy + \min_{x \geq 0}\{cx | Ax \geq b - By\}\} \tag{3.5}$$

The minimization problem $\min_{x \geq 0}\{cx | Ax \geq b - By\}$ is a linear program so it can be substituted by its dual $\max_{\pi \geq 0}\{\pi(b - By) | \pi A \leq c\}$ which leads to

$$\min_{y \in Y}\{dy + \max_{\pi \geq 0}\{\pi(b - By) | \pi A \leq c\}\} \tag{3.6}$$

Problems 3.5 and 3.6 are equivalent. The inner maximization problem of 3.6 is a linear program so it can be either bounded, unbounded or infeasible. If it is infeasible the inner minimization problem of 3.5 is either unbounded or infeasible. This means that the original problem 3.1 - 3.4 is unbounded or infeasible in that case. Therefore, the inner maximization of 3.6 is assumed to be feasible. Note that feasibility does not depend on y, so the inner maximization problem is infeasible for all y or feasible for all y. If it is unbounded for some $\bar{y} \in Y$ then the inner minimization problem of 3.5 is infeasible and that means there exists no solution for the original problem with $y = \bar{y}$. Also, an unbounded dual problem means there exists some extreme ray ρ with $\rho(b - By) > 0$. This situation can be avoided by adding a constraint

$$\rho(b - By) \leq 0 \tag{3.7}$$

Now suppose the inner maximization problem of 3.6 is bounded. Then the optimal solution is an extreme point of the feasible region. Suppose there are P extreme points called π^p for $p \in \{1, \dots, P\}$. The maximization problem $\max_{\pi \geq 0}\{\pi(b - By) | \pi A \leq c\}$ can also be written as

$$\max\{\pi^p(b - By) | p \in \{1, \dots, P\}\} \tag{3.8}$$

Suppose there are Q extreme rays ρ^q with $q \in \{1, \dots, Q\}$. Combining constraints of the form 3.7 with expression 3.8 and an auxiliary variable means the original problem can also be expressed as

$$\begin{array}{lllr}
\min & dy + \zeta & & (3.9) \\
\text{s.t} & \zeta \geq \pi^p(b - By) & \forall p \in \{1, \dots, P\} & (3.10) \\
 & \rho^q(b - By) \leq 0 & \forall q \in \{1, \dots, Q\} & (3.11) \\
 & y \in Y & & (3.12)
\end{array}$$

3.1 Benders' Algorithm

In itself the reformulation of problem 3.1 - 3.4 into 3.9 - 3.12 is not very useful. Due to the possibly large number of extreme points and/or extreme rays it is still not easy to solve the problem. However, the reformulation does give rise to a potentially useful algorithm.

The algorithm starts by removing all constraints 3.10 and 3.11 from problem 3.9 - 3.12. This problem will be called the *master problem*. At each iteration a constraint of type 3.11 (feasibility constraint) or 3.10 (optimality constraint) will be added. So after t iterations some $P' \leq P$ optimality constraints and $Q' \leq Q$ feasibility constraint have been added with $P' + Q' = t$. At that point, the master problem is

$$\begin{array}{lllr}
\min & dy + \zeta & & (3.13) \\
\text{s.t} & \zeta \geq \pi^p(b - By) & \forall p \in \{1, \dots, P'\} & (3.14) \\
 & \rho^q(b - By) \leq 0 & \forall q \in \{1, \dots, Q'\} & (3.15) \\
 & y \in Y & & (3.16)
\end{array}$$

The master problem after t iterations is the same as problem 3.9 - 3.12 with a number of constraints removed, and hence the objective value of an optimal solution for the master problem is a lower bound (LB) for problem 3.9 - 3.12. Therefore, the master problem after t iterations provides a LB for the original problem 3.1 - 3.4.

The feasibility and optimality constraints are generated by solving the inner maximization of 3.6. The y variables in the objective function get the values of the y variables of the most recent optimal solution for the master problem. This is called the *subproblem*. Optimization of the subproblem will yield an extreme ray or extreme point that can be used to either build a feasibility or an optimality constraint, respectively. When the subproblem has been solved to optimality (so it is not unbounded), the optimal objective function plus dy are an upper bound (UB) for the original problem. If this UB is smaller than the incumbent UB, then a new upper bound has been found.

So the algorithm starts by solving the master problem and storing the LB. Then the solution to the master problem is used to build a new subproblem. The subproblem is solved and either a feasibility or optimality constraint is added to the master. If a new UB is found and it is better than the old one the UB is updated. Now the master (including the new constraint) is solved, which leads to a new LB and a new subproblem to be solved. The algorithm stops when LB=UB, because at that point the solution to the original problem has been found. Pseudocode of the algorithm can be found in Algorithm 3.1.

Algorithm 1 Benders Decomposition

```
upper bound $UB = \infty$
lower bound $LB = -\infty$
while $UB - LB > 0$ do
    $\bar{y}, \bar{\xi} = \text{argmin}\{dy + \xi | y \in Y, \xi \geq 0\}$
    $LB = d\bar{y} + \bar{\xi}$
    solve $\max_{u \geq 0}\{u(b - B\bar{y}) | uA \leq c\}$
    if subproblem is unbounded then
        get direction $r$ and add constraint
        $r(b - By) \leq 0$ to master problem
    else
        get optimal solution $u$ and objective
        value $u(b - B\bar{y})$, set
        $UB = \min\{UB, d\bar{y} + u(b - B\bar{y})\}$
        and add constraint $\xi \geq u(b - By)$ to master
    end if
    solve master, set new $y$ and LB
end while
```

3.2 Benders' Decomposition for Demand Robust Optimization

Benders' is also well suited to be applied to robust optimization problems. For discrete uncertainty sets the general Benders' decomposition as described in Section 3.1 can be used in a straightforward way. For polyhedral uncertainty sets Benders' has to be adapted to the fact that the subproblem is no longer a linear program. In the case described in this report the subproblem is a bilinear program which calls for a different implementation of Benders'.

3.2.1 Discrete Uncertainty Sets

Using Benders' decomposition to solve a demand robust optimization problem with a discrete and finite uncertainty set is quite straightforward. This section is based on the disaggregation of Benders' cuts presented in [22]. In [22] Benders' decomposition is applied to a facility location problem with capacity expansion and multiple products. The resulting subproblem is then split into two by seperating the transportation and facility expansion problems and it is further split into seperate problems for products and facilities. In [20] the subproblem is split for every time period T. A seperate subproblem is created for every scenario in [21]. In [21] the average costs of the second stage are optimized, as opposed to the worst case costs. In this section a seperate subproblem is created for every scenario.

Disaggregation can be applied to subproblems that can be split into independent parts ([22]). Matrix notation is used in this section to show that the scenario subproblems are independent except for the cost constraint. Let the standard problem be defined as

$$\min \quad dy + \eta \tag{3.17}$$

$$\text{s.t.} \quad Ax^k + By \geq b^k \qquad \forall k \in \{1, \dots, K\} \tag{3.18}$$

$$\eta - c^k x^k \geq 0 \qquad \forall k \in \{1, \dots, K\} \tag{3.19}$$

$$y \in Y, x^k \geq 0 \quad \forall k \in \{1, \dots, K\} \tag{3.20}$$

where y are the first stage variables and x^k are the second stage variables for scenario k. The set Y is defined as $Y = \{y | y \in \mathbb{N}, Dy \geq e\}$. An expanded representation of 3.17 - 3.20 is given by

$$\min \quad dy + \eta \tag{3.21}$$

$$\text{s.t.} \quad \begin{pmatrix} 0 & A & 0 & \cdots & 0 \\ 0 & 0 & A & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A \\ 1 & -c^1 & 0 & \cdots & 0 \\ 1 & 0 & -c^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & -c^K \end{pmatrix} \begin{pmatrix} \eta \\ x^1 \\ \vdots \\ x^K \end{pmatrix} + \begin{pmatrix} B \\ \vdots \\ B \\ 0 \\ \vdots \\ 0 \end{pmatrix} y \geq \begin{pmatrix} b^1 \\ \vdots \\ b^K \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{3.22}$$

$$\eta \geq 0, x^1, \ldots, x^K \geq 0, y \in Y \tag{3.23}$$

For some fixed $y = \bar{y}$ this can be reduced to

$$\min \quad \eta \tag{3.24}$$

$$\text{s.t.} \quad \begin{pmatrix} 0 & A & 0 & \cdots & 0 \\ 0 & 0 & A & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A \\ 1 & -c^1 & 0 & \cdots & 0 \\ 1 & 0 & -c^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & -c^K \end{pmatrix} \begin{pmatrix} \eta \\ x^1 \\ \vdots \\ x^K \end{pmatrix} \geq \begin{pmatrix} b^1 \\ \vdots \\ b^K \\ 0 \\ \vdots \\ 0 \end{pmatrix} - \begin{pmatrix} B \\ \vdots \\ B \\ 0 \\ \vdots \\ 0 \end{pmatrix} \bar{y} \tag{3.25}$$

$$\eta \geq 0, x^1, \ldots, x^K \geq 0 \tag{3.26}$$

The dual of problem 3.24 - 3.26 is the Benders' subproblem. The dual variables will be

$$\pi = \begin{pmatrix} \pi^{u,1} & \cdots & \pi^{u,K} & \pi^{\eta,1} & \cdots & \pi^{\eta,K} \end{pmatrix} \tag{3.27}$$

In 3.27 $\pi^{u,k}$ refers to a set of variables that correspond to the scenario constraints $Ax^k \geq b^k - B\bar{y}$ for every scenario k, and the $\pi^{\eta,k}$ variables are the dual variables of the cost constraint $\eta - c^k x^k \geq 0$ for every k. The Benders' subproblem for some master solution $\bar{y}$ can now be expressed as

$$\max \quad \pi \left(\begin{pmatrix} b^1 \\ \vdots \\ b^K \\ 0 \\ \vdots \\ 0 \end{pmatrix} - \begin{pmatrix} B \\ \vdots \\ B \\ 0 \\ \vdots \\ 0 \end{pmatrix} \bar{y} \right) \tag{3.28}$$

$$\text{s.t.} \quad \pi \begin{pmatrix} 0 & A & 0 & \cdots & 0 \\ 0 & 0 & A & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A \\ 1 & -c^1 & 0 & \cdots & 0 \\ 1 & 0 & -c^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & -c^K \end{pmatrix} \leq \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix} \tag{3.29}$$

$$\pi \geq 0 \tag{3.30}$$

The objective 3.28 can be reduced because for all dual variables of cost constraints $\pi^{\eta,k}$ the objective parameter is 0 so 3.28 can also be expressed as

$$\begin{pmatrix} \pi^{u,1} & \cdots & \pi^{u,K} \end{pmatrix} \left(\begin{pmatrix} b^1 \\ \vdots \\ b^K \end{pmatrix} - \begin{pmatrix} B \\ \vdots \\ B \end{pmatrix} \bar{y} \right) \tag{3.31}$$

The problem can further be reduced in the following way. Suppose k^* is the most expensive scenario for some optimal solution to 3.24 - 3.26 for fixed $\bar{y}$. In other words, $\eta \geq c^{k^*}x^{k^*} > c^k x^k$ for all $k \neq k^*$ (if multiple scenarios are tied for most expensive then the first one that is encountered is chosen). This means that the cost constraints $\eta - c^k x^k$ are not binding and therefore the dual variables $\pi^{\eta,k}$ are 0 for $k \neq k^*$ due to complementary slackness. The constraint that linked the dual problem together is now no longer necessary so the problem can be split into K independent problems.

For all $k \neq k^*$ the problem is

$$\max \quad \pi^{u,k}(b^k - B\bar{y}) \tag{3.32}$$

$$\text{s.t.} \quad \pi^{u,k} A \leq 0 \tag{3.33}$$

$$\pi^{u,k} \geq 0 \tag{3.34}$$

For the most expensive scenario k^* the problem is

$$\max \quad \pi^{u,k^*}(b^{k^*} - B\bar{y}) \tag{3.35}$$
$$\text{s.t.} \quad \pi^{u,k^*}A - \pi^{\eta,k^*}c^{k^*} \leq 0 \tag{3.36}$$
$$\pi^{u,k^*} \geq 0, \quad 0 \leq \pi^{\eta,k^*} \leq 1 \tag{3.37}$$

Assuming $c^k \geq 0$ for all k choosing $\pi^{\eta,k^*} = 1$ gives the largest feasible region, so problem 3.35 - 3.37 reduces to

$$\max \quad \pi^{u,k^*}(b^{k^*} - B\bar{y}) \tag{3.38}$$
$$\text{s.t.} \quad \pi^{u,k^*}A \leq c^{k^*} \tag{3.39}$$
$$\pi^{u,k^*} \geq 0 \tag{3.40}$$

Solving problem 3.28 - 3.30 is equivalent to solving the $K-1$ problems 3.32 - 3.34 and problem 3.38 - 3.40. Solving 3.32 - 3.34 is trivial because if it is unbounded an extreme ray has to be found but when the subproblem is bounded, the optimal objective is 0 with trivial solution $\pi^{u,k} = 0$. Due to the way the Demand Robust Location-Transportation problem is defined in Section 2.1.2 the subproblem is always bounded and therefore its dual is too. This report focuses on that problem so from now on the dual will always be assumed to be bounded. Solving 3.38 - 3.40 can be done by recognizing that its dual is

$$\min \quad c^{k^*}x^{k^*} \tag{3.41}$$
$$\text{s.t.} \quad Ax^{k^*} \geq b^{k^*} - B\bar{y} \tag{3.42}$$
$$x^{k^*} \geq 0 \tag{3.43}$$

This observation can be used to device a solution strategy for 3.28 - 3.30. It starts by solving $\min_{x^k \geq 0}\{c^k x^k | Ax^k \geq b^k - B\bar{y}, x^k \geq 0\}$ for every scenario k. All objective values for the optimal solution are then compared, and the most expensive scenario is called k^*. For all other scenarios $k \neq k^*$ the following values are assigned: $\pi^{\eta,k} = \pi^{u,k} = 0$. For k^*: $\pi^{\eta,k^*} = 1$ and π^{u,k^*} equals the vector of shadow prices of the constraints 3.42.

This yields the following Benders' reformulation

$$\min \quad dy + \zeta \tag{3.44}$$
$$\text{s.t.} \quad \zeta \geq \pi^{u,k}\left(b^k - By\right) \qquad \forall p \in \{1, \ldots, P\} \tag{3.45}$$
$$y \in Y \tag{3.46}$$

3.2.2 Polyhedral Uncertainty Sets

As said in Chapter 2, the most expensive scenario for a scenario set as in 2.5 will always be an extreme point of that scenario set. So if all extreme points of the scenario set are explicitly generated the solution method of the previous section could be applied because the set of extreme points of the scenario set is discrete and finite. Unfortunately this is infeasible in practice. Therefore the solution to problem 2.17 - 2.22 has to be used. An optimal solution π^*, g^* is sufficient to generate an optimality constraint $\pi^*(u^* - By)$ for the Benders' master problem.

3.3 Demand Robust Location-Transportation

In this section Benders' decomposition is applied to the Demand Robust Location-Transportation problem.

3.3.1 Discrete Uncertainty Sets

The original problem in the case of discrete uncertainty sets is given by 2.45 - 2.51. The following exposition is a slight adaptation of the work in [24]. The only difference is the discrete instead of polyhedral uncertainty set. The starting master problem is

$$\min_{(y,z,\zeta)} \quad \sum_i (f_i y_i + a_i z_i) + \zeta \tag{3.47}$$

$$\text{s.t.} \quad z_i \leq K_i y_i \qquad \forall i \tag{3.48}$$

$$\sum_i z_i \geq u_{max} \tag{3.49}$$

$$y_i \in \{0,1\}, \quad z_i \geq 0 \tag{3.50}$$

In the starting master problem variable ζ is unconstrained, which leads to an unbounded problem. This can be prevented by choosing $\zeta = 0$ or removing ζ from the master and introducing it the first time an optimality constraint is added. Be careful to ignore the LB found by optimizing the master problem with $\zeta = 0$ because it could be too high.

The K subproblems (one for every scenario k) based on an (optimal) solution (y^*, z^*) of the master problem are

$$\min \quad \sum_{ij} c_{ij} x_{ij}^k \tag{3.51}$$

$$\text{s.t} \quad -\sum_j x_{ij}^k \geq -z_i^* \qquad \forall i \tag{3.52}$$

$$\sum_i x_{ij}^k \geq u_j^k \qquad \forall j \tag{3.53}$$

$$x_{ij}^k \geq 0 \tag{3.54}$$

Optimizing the subproblem for every scenario k yields a solution $\pi^{sup,k*}, \pi^{dem,k*}$ for the most expensive scenario where $\pi^{sup,k*}$ are the dual values of supply constraints 3.52 and $\pi^{dem,k*}$ are the dual values of demand constraints 3.53. This gives Benders' extreme point constraint $\zeta - \sum_j \pi_j^{dem,k*} u_j^{k*} + \sum_i \pi_i^{sup,k*} z_i \geq 0$ that can be added to the master. So after T iterations the master problem is given by

$$\min_{(y,z,\eta)} \quad \sum_i (f_i y_i + a_i z_i) + \zeta \tag{3.55}$$

$$\text{s.t.} \quad z_i \leq K_i y_i \qquad \forall i \tag{3.56}$$

$$\sum_i z_i \geq u_{max} \tag{3.57}$$

$$\zeta - \sum_j \pi_j^{(dem,k*),t} u_j^{k*} + \sum_i \pi_i^{(sup,k*),t} z_i \geq 0 \quad \forall t \in \{1,\ldots,T\} \tag{3.58}$$

$$y_i \in \{0,1\}, \quad z_i \geq 0 \tag{3.59}$$

3.3.2 Polyhedral Uncertainty Sets

The master problem for Demand Robust Location-Transportation with a polyhedral uncertainty set is the same as the one used for discrete uncertainty set. The subproblem is based on 2.17 - 2.22 and is ([13])

$$\max \quad \sum_j \underline{u}_j \pi_j^{dem} + \sum_j \bar{u}_j \omega_j - \sum_i z_i^* \pi_i^{sup} \tag{3.60}$$

$$\text{s.t.} \quad \pi_j^{dem} - \pi_i^{sup} \leq c_{ij} \qquad \forall i,j \tag{3.61}$$

$$\sum_j g_j \leq \Gamma \tag{3.62}$$

$$\omega_j \leq \pi_j^{dem} \qquad \forall j \tag{3.63}$$

$$\omega_j \leq M g_j \qquad \forall j \tag{3.64}$$

$$\pi_j^{dem}, \pi_i^{sup}, \omega_j \geq 0, g_j \in \{0,1\} \tag{3.65}$$

M is some really big number. According to [13] it can be set to the value $\pi_j^{dem,*}$ where $\pi_j^{dem,*}$ is the value of π_j^{dem} in the optimal solution of the subroblem for Γ equals the number of customers, so all demands are at their maximum value. This means that $g_j = 1$ for all j and $\omega_j = \pi_j^{dem}$. So these values can be found by solving

$$\max \quad \sum_j ((\underline{u}_j + \bar{u}_j)\pi_j^{dem}) - \sum_i z_i^* \pi_i^{sup} \tag{3.66}$$

$$\text{s.t.} \quad \pi_j^{dem} - \pi_i^{sup} \leq c_{ij} \qquad \forall i,j \tag{3.67}$$

$$\pi_j^{dem}, \pi_i^{sup} \geq 0 \tag{3.68}$$

3.4 Algorithmic Enhancements

Benders' decomposition can work well for some problems but solving numerous (increasingly large) MIPs can hurt performance. A number of ways of overcoming this problem exist. Below are two ways of improving Benders' decomposition that can be used separately or combined. The 2-phase method described in 3.4.1 and the method of using incumbent solutions (3.4.2) are ways to utilize the observation that optimal subproblem solutions based on non-optimal master solutions also provide valid constraints. This can be used to reduce the number of times the master MIP has to be solved.

3.4.1 2 Phase Method

Repeatedly solving the master problem (a MIP) to optimality can slow down the convergence of Benders decomposition. A way to possibly mitigate this performance issue is described in [18]. The method is based on the fact that the feasible region of the dual of the subproblem is independent of the solution to the master problem. This means that optimizing the (bounded dual) subproblem will always yield an extreme point of the feasible region and will lead to a valid constraint for the master problem. So a non-optimal or infeasible solution for the master problem still leads to a valid constraint. This follows from the fact that the dual of the subproblem looks like this:

$$\begin{aligned} \min \quad & \pi(b - By) && (3.69) \\ \text{s.t.} \quad & \pi A \leq c && (3.70) \\ & \pi \geq 0 && (3.71) \end{aligned}$$

The feasible region $\{\pi : \pi A \leq c, \pi \geq 0\}$ of the dual of the subproblem 3.69 - 3.71 is independent of master problem solution y; y only influences what extreme point of the feasible region is returned after optimization. Therefore it is not strictly necessary to optimize the dual of the subproblem with an optimal master solution to generate a valid constraint for the master. It is not even necessary that the vector y is feasible for the master problem. Simply put, solving the subproblem to optimality yields a valid constraint for Benders decomposition regardless of the feasibility or optimality of y for the master problem.

A substitute for an optimal master solution could be a solution to the linear programming relaxation of the master problem. The LP relaxation of the master problem is obtained by simply ignoring the integer (or binary) constraints. A solution to the relaxed master problem will most likely not be a feasible solution for the master problem but it can be used in the subproblem to generate constraints. The major drawback of this method is that the solution of the dual of the subproblem can no longer be used as a valid upper bound in Benders' algorithm. Therefore, this way of generating constraints has to be implemented in a 2-phase method. In the first phase the LP relaxation of the master problem is repeatedly solved and the solutions are used to generate constraints while in the second phase the integer constraints are reinstated and the master problem is solved.

The time spent in the first phase can be chosen in a number of ways:

1. Continue until the LP relaxed algorithm has converged.

2. Use the relaxed master problem for a fixed number of iterations.

3. Stop when the ζ-variable of the master and the objective value of the subproblem are close enough.

Method 1 leads to the longest possible first phase. Once the first phase has converged no new constraints can be found using this method. Method 2 is really straightforward. Before starting a number of iterations for the first phase is chosen and after that the algorithm reapplies the integer constraints to the master variables that were removed in the first phase. Method 3 compares the value of the ζ-variable in the master problem and the objective value of the optimized subproblem. Once these are deemed close enough the algorithm continues to the second phase.

It is hard to say in advance which method of stopping the first phase gives the biggest performance boost (if any). The best implementation of this 2-phase algorithm can only be determined experimentally, also because it is likely to be problem dependent.

3.4.2 Using Incumbent Solutions

As explained in section 3.4.1, it is not strictly necessary to use optimal master solutions in the subproblem to generate valid constraints for the master problem. Suboptimal but feasible solutions of the master problem can also be used to generate constraints ([10]). When the master is solved with a modern MIP-solver (using Branch-and-Bound) callback functions can be used to get a (at that time) feasible solution for the master problem every time a new best incumbent solution is found. This new incumbent solution is used in the subproblem and the resulting constraint is added as a lazy constraint. This can be continued until the MIP-solver converges and has found the optimal solution.

Chapter 4

Column & Constraint Generation

A rather novel way of solving robust optimization problems called *Column & Constraint Generation*(C&CG) was first presented in [24]. C&CG is a decomposition strategy that uses a master and subproblem framework similar to Benders' decomposition. The main difference between Benders' decomposition and C&CG is that Benders' is a general solving procedure that can be applied to a wide range of mixed integer programming problems while C&CG is specifically suited to solving robust optimization problems.

Despite being quite new there are already a number of applications for C&CG. It has been mostly used in robust optimization problems concerning power grids ([1], [14], [15], [23], [25]) but it also seems to work well for facility location problems ([2], [24]).

The name Column & Constraint Generation is based on the fact that the algorithm iteratively adds variables (columns) and constraints to the problem until the optimal solution is found. The variables that are added correspond to second stage decision variables and the constraints are from scenarios that are added to the problem. The scenarios are selected on basis of being the *worst case* at some point in the optimization.

4.1 Discrete Uncertainty Sets

This explanation of Column & Constraint Generation is derived from [24]. The difference is the use of discrete uncertainty sets instead of polyhedral ones. Suppose there is a robust optimization problem with K scenarios and the problem can be modelled as follows:

$$\begin{aligned}
\min_{y,\eta} \quad & dy + \eta && && (4.1)\\
\text{s.t.} \quad & Ax^k + By \geq b^k && \forall k \in \{1,\ldots,K\} && (4.2)\\
& \eta \geq c^k x^k && \forall k \in \{1,\ldots,K\} && (4.3)\\
& Dy \geq e && && (4.4)
\end{aligned}$$

In the problem formulation above the vector y consists of the first stage variables and the vector x^k is made up of the second stage variables for scenario k. c^k is the cost vector for the second stage variables in scenario k.

The main idea of C&CG is iteratively adding scenarios and the corresponding variables until an optimal solution is found. The optimal solution is found when the *lower bound* (LB) and *upper bound* (UB) maintained by the algorithm are equal, so when $LB = UB$.

The LB is based on the idea that the objective value of the optimal solution for a restricted set of scenarios is never worse than the objective value of the optimal solution for the complete set of scenarios. Let U be the complete set of scenarios for problem (4.1)-(4.4) and let $U' \subseteq U$ be some restricted scenario set, then the objective value of the optimal solution for the problem with a restricted scenario set is a lower bound for the original problem if the subproblem is bounded and has a feasible solution. Suppose that $y^*, x^{1,*}, \ldots, x^{K,*}$ is an optimal solution for the problem with scenario set U, then y^* can be used as a partial solution for the problem with scenario set U' that has an objective value that is never larger than the objective value for this first stage solution with scenario set U. Therefore, the optimal solution of the restricted problem leads to a lower bound for the complete problem.

UB is found by solving the restricted problem and using the resulting optimal first stage vector y'^* to solve the second stage problem for all scenarios. This leads to a solution $(y'^*, x'^*, \ldots, x'^K)$ that is feasible for the problem with complete scenario set U and the objective value of this solution is therefore an upper bound on the objective value of the optimal solution with scenario set U.

The above leads to the C&CG algorithm where the master problem is the original problem with a restricted set of scenarios and the subproblem is optimizing the second stage variables x^k for all scenarios separately given the first stage decision y found by optimizing the master problem. The scenario k' that has the highest second stage costs $c^{k'} x^{k'}$ is then added to the master problem and the master problem is solved again. This continues until UB=LB.

4.2 Polyhedral Uncertainty Set

The approach described in the previous section works for discrete and finite uncertainty sets but not for polyhedral uncertainty sets as the one defined in 2.5 in Chapter 2. This is because the number of possible scenarios (all extreme points of U) is potentially way too big to result in a solution in a decent amount of time. Therefore, scenarios will be generated by solving problem 2.17 - 2.22. Solving this problem also results in finding the most expensive scenario.

4.3 Demand Robust Location-Transportation

In this section the Column & Constraint Generation algorithm is applied to the Demand Robust Location-Transportation problem. This part is also based on [24] with the difference being the discrete uncertainty sets instead of polyhedral ones.

4.3.1 Discrete Uncertainty Set

The starting master problem for C&CG (with a discrete uncertainty set) is the same as the starting master problem for Benders' decomposition (see 3.47 - 3.50). At every iteration $t \in \{1, \ldots, T\}$ one scenario is added to this master problem, so after t' iterations the master problem is the same as the Demand Robust Location-Transportation for a discrete uncertainty set with scenario set $\{1, \ldots, t'\}$. So it is

$$
\begin{aligned}
\min_{(y,z,\eta)} \quad & \sum_i (f_i y_i + a_i z_i) + \eta && && (4.5)\\
\text{s.t.} \quad & z_i \le K_i y_i && \forall i && (4.6)\\
& \sum_i z_i \ge u_{max} && && (4.7)\\
& \sum_j x_{ij}^t \le z_i && \forall i, t && (4.8)\\
& \sum_i x_{ij}^t \ge u_j^t && \forall j, t && (4.9)\\
& \eta - \sum_{ij} c_{ij} x_{ij}^t \ge 0 && \forall t && (4.10)\\
& y_i \in \{0,1\}, \quad z_i \ge 0, \quad x_{ij}^t \ge 0 && && (4.11)
\end{aligned}
$$

Similar to Benders' decomposition, K independent subproblems are solved. After solving them the most expensive scenario is added to the master problem. For every scenario k the subproblem is given by

$$\begin{aligned}
\min \quad & \sum_{ij} c_{ij} x_{ij}^k && && (4.12) \\
\text{s.t} \quad & -\sum_{j} x_{ij}^k \geq -z_i^* && \forall i && (4.13) \\
& \sum_{i} x_{ij}^k \geq u_j^k && \forall j && (4.14) \\
& x_{ij}^k \geq 0 && && (4.15)
\end{aligned}$$

After solving the subproblem, demands u_j^{k*} are sufficient to add a new scenario to the master problem.

4.3.2 Polyhedral Uncertainty Set

For polyhedral uncertainty sets the starting master is again the same as 3.47 - 3.50. At every iteration a scenario is generated by solving the same subproblem as is done for Benders' decomposition with polyhedral uncertainty sets (see 3.60 - 3.65). After solving the subproblem a scenario can be generated using the variables g_k^* from the optimal solution and that scenario can be added to the master problem. The master problem for polyhedral uncertainty sets after T iterations is the same as the one used for discrete uncertainty sets.

Chapter 5

Computational Results

To compare Benders' decomposition and C&CG a robust location-transportation will be solved. The problem is the same as in [24]. Several implementations of Benders' decomposition with a number of enhancements will be tested to find the optimal implementation for this problem for discrete and polyhedral uncertainty sets. Benders' performance is then compared to the performance of C&CG.

The demand robust location-transportation problem was given in Chapter 2 Section 2.1.2 as

$$\min_{(y,z,x)} \quad \sum_i (f_i y_i + a_i z_i) + \max_{u \in U} \min \sum_{ij} c_{ij} x_{ij} \tag{5.1}$$

$$\text{s.t.} \quad z_i \leq K_i y_i \tag{5.2}$$

$$\sum_j x_{ij} \leq z_i \tag{5.3}$$

$$\sum_i x_{ij} \geq u_j \tag{5.4}$$

$$y_i \in \{0,1\}, \quad z_i \geq 0, \quad x_{ij} \geq 0 \tag{5.5}$$

5.1 Problem Generation

The way to generate instances of the robust location-transportation problem described in [24] is derived from [13]. The first thing to determine is the number of facilities n_f and the number of demands n_d. At one point [13] states that it is more realistic to have the number of facilities to be larger than the number of demands. However, later on in [13] tests are performed on instances with $n_f = n_d$. In [24] tests are also performed on instances with $n_f = n_d$. This example will be followed and test are performed on instances with $n_f = n_d = 30$.

To generate the polyhedral uncertainty set U base demands $\underline{u}_j$ are drawn from $[10, 500]$ with maximal deviation $\bar{u}_j \in [01.\underline{u}_j, 0.5\underline{u}_j]$ Fixed costs f_i are drawn from $[100, 1000]$, variable facility costs per unit a_i from $[10, 100]$ and maximal allowable capacity K_i from $[200, 700]$. Transportation costs c_{ij} are in the interval $[1, 1000]$.

To ensure feasibility the inequality $\sum_i K_i \geq \max_{u \in U}\{\sum_j u_j\}$ has to be respected. Neither [24] nor [13] make clear how this is taken care of during generation so instances that violate the feasibility constraint will just be ignored. So an instance is generated, the feasibility is checked and if it is feasible it is entered into the set of test instances.

The scenarios for the version of the problem with discrete uncertainty sets were generated based on the corresponding polyhedral uncertainty sets. To generate a scenario Γ customers are chosen to have their maximum demand in that scenario. All other customers have base demand. All discrete problems have 100 scenario's.

All algorithms are implemented in C# and executed on a computer with an Intel Core2Duo 2.20 GHz processor.

5.2 Results

To evaluate the performance of the various optimization algorithms 10 instances of size 30x30 were generated. Each of these problems was extended with 100 random scenario's for every value of Γ. The values of Γ are $3, 6, 9, 12, 15, 18, 21, 24$ and 27.

5.2.1 Discrete Uncertainty Sets

The results of the algorithmic experiments for problems with a discrete uncertainty set are presented in this subsection. The performance of the three variations of Benders' decomposition are presented first. After that the performance of the standard Gurobi MIP-Solver is discussed and finally they are compared with Column & Constraint Generation.

Benders' Decomposition

Three variations of Benders decomposition are compared. A classic implementation (denoted by BenClass) that alternatingly solves the master and subproblem to optimality until the algorithm converges, an implementation where the subproblem is solved every time the solver of the master problem encounters a new incumbent solution (BenCB) and an implementation that implements the 2 Phase method (Ben2Phase). The 2 Phase method was implemented in such a way that the relaxed master problem was solved for 5 iterations before the standard Benders' algorithm took over. As you can see in Table 5.2.1, the classic implementation outperforms BenCB and is comparable to Ben2Phase. This can be explained by the fact that both enhancement methods are aimed at reducing the burden caused by repeatedly solving the master problem to optimality. Apparently, the master problem is not such a huge bottleneck for this problem that these methods provide a significant boost in performance.

	Γ	3	6	9	12	15	18	21	24	27
BenClass	time(s)	33.5	69.3	63.5	58.5	57.6	51.7	48.0	45.0	17.6
	iterations	67.4	58.7	55.1	53	52.5	48.8	47.7	45.7	45.4
Ben2Phase	time(s)	36.4	69.5	24.0	48.2	31.5	19.1	42.1	39.1	38.7
	iterations	64.2	58	52	46	43.2	44.3	42	40	40.2
BenCB	time(s)	133.9	105.4	106.5	123.4	68.2	89.1	102.9	87.7	84.0
	iterations	653.3	517.3	506.7	464.4	427.6	433.7	383.3	339.7	370.5

Table 5.1: Performance of variations of Benders' decomposition, discrete uncertainty sets, 100 scenarios

Gurobi MIP-solver

The Gurobi MIP-solver provides a wide range of settings that can be used to tune its performance. All were left in their default setting, except for one: the optimality gap. The optimality gap is used to determine when the algorithm can stop optimizing. When the relative gap between the lower bound and the upper bound is smaller than the optimality gap the algorithm terminates. It would be more elegant if the algorithm terminates when the upper and lower bound are equal but in practice this is not feasible, mainly due to rounding errors. It could be that a rounding error causes the upper bound to be slightly larger than the lower bound even though the optimal solution is found and this would cause a failure to terminate. So the optimality gap is a necessary evil.

The standard optimality gap in Gurobi is 10^{-4}. There is no fundamental reason for the gap to have this value. A value of 10^{-4} still allows for termination before an optimal solution is certain to be found, as can be seen in the results. Therefore it is justified to see how this parameter affects algorithmic performance. A comparison is made to solution times obtained when setting the optimality gap to 10^{-2}.

The results are quite surprising. Solution times are a lot better for the larger optimality gap. However, it could be that solutions obtained by the larger gap could be a lot worse because 10^{-2} is a lot bigger than 10^{-4}. This seems to be the case when looking at the results in Table 5.2.1. The solver terminates with a solution that is guaranteed to be optimal for a considerable number of instances when the allowed gap is 10^{-4} and the average final optimality gap is roughly 100 times worse when the allowed gap is 10^{-2} as would be expected.

	Γ	3	6	9	12	15	18	21	24	27
MIP (gap= 10^{-4})	time(s)	37.8	68.1	70.0	71.8	58.4	49.2	48.5	44.2	52.8
	exact	5/10	3/10	4/10	6/10	5/10	7/10	3/10	2/10	2/10
	avg gap ($*10^{-5}$)	2.29	4.40	3.68	1.77	3.00	1.09	4.14	4.89	4.83
MIP (gap= 10^{-2})	time(s)	5.0	9.1	9.2	9.3	9.2	9.0	9.3	9.4	7.3
	exact	0/10	0/10	0/10	0/10	0/10	0/10	0/10	0/10	0/10
	avg gap ($*10^{-5}$)	379	362	310	298	266	259	247	220	214

Table 5.2: Performance of Gurobi MIP-solver, discrete uncertainty sets, 100 scenarios

The actual difference between the solutions that are found by both methods are a lot smaller. The average relative difference (see Table 5.2.1) is less than 1 in 1000. Based on the difference between the allowed optimality gaps this relative difference could have been almost 1 percent. The relatively small difference can be explained by the way an optimality gap is calculated. It is simply (UB - LB)/UB so a larger optimality gap does not necessarily mean that the current solution is bad. A difference in optimality gap can also be caused by a lower bound that is less tight. For this problem that seems to be the case. This shows that performance can be dramatically increased while the obtained solution is, on average, less than 1 in a 1000 worse.

Γ	3	6	9	12	15	18	21	24	27
avg diff. ($*10^{-5}$)	44.5	69.6	37.1	39.5	37.9	31.1	22.5	2.0	1.9

Table 5.3: Relative difference between optimal solutions obtained by the Gurobi MIP-solver with optimality gap 10^{-2} and 10^{-4}. The number for each value of Γ is the average relative difference over ten instances. The realtive difference is calculated by $(UB^{10^{-2}} - UB^{10^{-4}})/UB^{10^{-4}}$

Table 5.2.1 makes it clear that a large part of the difference in optimality gap can be explained by a worse lower bound. The performance of the Gurobi MIP-solver could potentially be sped up by instructing it to focus more on improving the lower bound. Fortunately it provides this option. There is a parameter called *MIPFocus* that can make the solver spend more resources on improving the lower bound. According to Gurobi's documentation: "If you believe the solver is having no trouble finding good quality solutions, and wish to focus more attention on proving optimality, select MIPFocus=2." The result can be found in Table 5.2.1. For this comparison the allowed optimality gap was left at its default value of 10^{-4}. The results show that changing the focus of the solver has a positive influence on its performance. The average time it takes to solve a problem decreases.

	Γ	3	6	9	12	15	18	21	24	27
MIP (default)	time(s)	37.8	68.1	70.0	71.8	58.4	49.2	48.5	44.2	52.8
	exact	5/10	3/10	4/10	6/10	5/10	7/10	3/10	2/10	2/10
	avg gap ($*10^{-5}$)	2.29	4.40	3.68	1.77	3.00	1.09	4.14	4.89	4.83
MIP (MIPFocus = 2)	time(s)	40.0	13.2	20.9	18.3	19.1	20.9	14.4	20.2	16.0
	exact	5/10	4/10	4/10	5/10	1/10	4/10	3/10	1/10	3/10
	avg gap ($*10^{-5}$)	3.44	3.47	2.44	4.03	4.18	4.23	2.82	4.80	1.88

Table 5.4: Result of difference focus settings for Gurobi MIP-solver, discrete uncertainty sets, 100 scenarios

The experiments with a different optimality gap and a different focus show that the performance of a solver can be influenced changes in its settings. These experiments were not aimed at finding the optimal settings for the Gurobi MIP-solver but are an indication that performance of this solver can be tuned and that this tuning can have a large effect on its performance.

Both Benders' decomposition and Column & Constraint Generation use a MIP-solver and a LP-solver as a subroutine. When comparing the performance of these algorithms it is important to keep in mind that solver performance can be greatly affected by changes in settings.

Column & Constraint Generation

Only one version of C&CG was implemented, but it has the best performance of all algorithms. The only algorithm that is a bit close is the MIP-solver with a large optimality gap. C&CG outperforms Benders' but it should be noted that the performance of Benders' decomposition is on par with the performance of the MIP-solver with the standard optimality gap.

	Γ	3	6	9	12	15	18	21	24	27
BenClass	time(s)	33.5	69.3	63.5	58.5	57.6	51.7	48.0	45.0	17.6
	iterations	67.4	58.7	55.1	53	52.5	48.8	47.7	45.7	45.4
C&CG	time(s)	3.2	3.0	5.2	4.1	3.4	3.7	4.6	3.3	1.6
	iterations	4.4	3.2	4.2	3.8	3.5	3.6	4.1	3.3	3.2
MIP Gap= 10^{-2}	time(s)	5.0	9.1	9.2	9.3	9.2	9.0	9.3	9.4	7.3

Table 5.5: Performance of Benders' decomposition, Column & Constraint Generation, Gurobi MIP-solver, discrete uncertainty sets, 100 scenarios

5.2.2 Polyhedral Uncertainty Sets

The experiments with the polyhedral uncertainty sets were conducted to verify the results of [24]. The results can be found in Table 5.2.2. In [24] the CPlex MIP-solver was used to solve the subproblem for C&CG and Benders' decomposition. Apparently this solver is much better at solving this specific problem than the Gurobi solver. Therefore, the results presented here are a lot slower than those presented in [24]. However, the number of iterations needed by either C&CG or Benders' decomposition to solve the problem is comparable to the number of iterations neede in [24]. This means that the number of iterations needed for C&CG is a lot smaller than the number needed for Benders' decomposition. The polyhedral nature of the uncertainty sets causes the subproblem to be a MIP instead of a LP, as is the case for discrete uncertainty sets. This turns the subproblem into the bottleneck and given that the subproblem for both algorithms is the same (only the information extracted from it is different) the algorithm that needs the smallest number of iterations is preferable.

The solution times marked with a * indicate that at least one of the tested instances failed to converge before the time limit of 3600 seconds was reached. In general nearly all time needed by both algorithms was spent solving the subproblem. Therefore the low number of necessary iterations gives C&CG a large advantage over Benders' decomposition. It should be noted that adding a scenario to the master problem when performing Column & Constraint Generation involves adding 900 continuous variables and 60 constraints to the master. During Benders' decomposition no variables are added to the master problem. The advantage that C&CG has is created by the smaller number of iterations while its master problem does not become so slow that it becomes a disadvantage. The fact that the master problems can be expanded with so many variables and constraints and not have its performance crippled is an interesting observation.

	Γ	3	6	9	12	15	18	21	24	27
BenClass	time(s)	249.44	5536.17	2620.42	3718.39*	3698.02*	3706.67*	1083.17*	1867.93*	20.45
	iterations	65	59.5	34.5	34.5	22.5	18.5	33.5	5770	27
C&CG	time(s)	10.42	101.43	509.25	611.17	477.70	230.56	97.75	14.40	3.17
	iterations	4.7	4.8	6.5	6.5	5.9	5.6	5.4	4	3.4

Table 5.6: Performance of Benders Decomposition and C&CG, polyhedral uncertainty sets. Only 2 problems per value of Gamma were solved by Benders' decomposition due to large solving times.

Chapter 6

Interpretation of Results

The computational experiments of Chapter 5 are performed in a way many experiments in optimization are carried out (see for example [24] and [13]). A number of instances is generated or obtained in some other ways and some algorithms are used to solve those problems. Then the performance of the various algorithms is compared.

The question to answer at this point is what the results presented in Chapter 5 (or those obtained by similar experiments) actually mean. By comparing the increase in performance of various versions of the CPLEX MIP-solvers, Bixby ([7]) shows that there has been a tremendous improvement in the CPLEX solver over the years. So the constant emergence of new ideas and the usually small scale testing of new algorithms and new versions of existing algorithms has led to great results.

However, it is still hard to generalize the results of experiments to larger groups of problems. On a fundamental level, the question remains what the outcome of the experiments in this thesis means for performance of the tested algorithms on other problem instances and problems. Preferably, one would like to make the claim that the results presented in this report are representative for all instances of the Demand Robust Location-Transportation problem or something similar. In other scientific fields such as medicine tests performed on a sample of the population are generalized to the complete population using statistics. The quality of these results for the population is affected by the representativeness of the sample for the population. Such an elevation of specific experiments to general rule cannot be made using the kind of experiments that are common in the field of combinatorial optimization and also performed in this report. Why this is impossible and possible ways to mitigate the consequences of this observation are explained in this chapter.

It would be nice to be able to make a claim along the lines of: ‘Column & Constraint Generation is superior to Benders decomposition for the optimization of the Demand Robust Location-Transportation problem.’ Based on the results of the previous chapter that looks like a reasonable statement, but that is not really what was proven. What has been shown is that Column & Constraint

Generation performs better than Benders decomposition for the instances tested using the code and computer system of the author. So to claim a general result, you would have to prove the following:

1. In general, solver performance is the same for a random instance of Demand Robust Location-Transportation as it was in the instances tested.
2. The code and underlying algorithmic choices are optimal.
3. All computer systems behave the same as the one used.

In the next three sections, the challenges accompanying these three points will be explained. After that, some possible ways to mitigate these effects will be discussed.

6.1 Challenges

6.1.1 Problem Selection

It is unclear how the performance of solver on a number of problem instances should be elevated to a general result for some problem. The generation of problems for this report was done using the description of [24], which is based on [13]. The parameters used to generate problem instances seem to be picked at random, or at least they are not thoroughly justified. Such a justification could for example be that these problems are a very good representation of a general problem or that they are close to a problem that is relevant in the real world. For example, when solver performance for the recourse problem is analysed section 4.1 of [13] the number of customers is larger than the number of warehouses 'to be closer to reality'. However, when the full problem is solved the number of customers equals the number of warehouses. This way of generating problems is not wrong in any way but it will not lead to a general result, simply because it is unclear how representative the sample is for what population.

A question that arises when thinking about the problem selection is the required number of tested instances. No attempt is made to link the sample size to the reliability of the result. So something that could be given more attention in papers like [24] and [13] is what set of problems was tested and how do the algorithmic performance figures translate to a larger group of problems.

Another possible issue that arises from the lack of a transparent way selecting test problems is possibility of cherry picking results. If authors are free to choose the problem instances on which they test their algorithms there is the possibility of trying an algorithm on a set of instances and only publishing the results for the subset of instances on which it performed well. It is hard to say how often this is done, but the major problem is that it is impossible to check because everyone can select their problem instances without having to justify their choices.

6.1.2 Code and Algorithmic Choices

An algorithm like Benders' decomposition is not so much one algorithm as it is a family of algorithms. That can be seen in this report for example by the fact that Benders' can be implemented with the 2 Phase-method or with a technique that uses incumbent solutions during the solving of the master problem to generate constraints. The Gurobi MIP-solver that was used for the problem with discrete uncertainty set has a wide range of settings that can be changed, one of which was the optimality gap that affected performance quite a bit. Even more, both Benders' decomposition and C&CG employ MIP- and LP-solvers to solve the master problem and subproblem respectively. These solvers can be configured in a lot of ways that can affect performance.

Besides algorithmic choices that influence performance there is also the way in which an algorithm is implemented that affects how well it does. On top of that, the programming language itself in which the algorithm was written can have an effect on its speed.

Two specific implementations of an algorithm that are given the same name by their respective authors could have differences in performance that are caused by different algorithmic choices and differences in implementation that are not immediately clear to the reader of an academic paper.

A fundamental problem with the way results are usually presented is that it is unclear how much an algorithm was tuned to perform optimally on the problem instances it is tested on. This could lead to a problem that is equivalent to the statistical notion of overfitting. This implies that it is possible for an algorithm that was tuned to perform optimally on the test instances might not perform very well on instances that it was not initially tested on.

6.1.3 Performance Variability

A more surprising source of possible performance variability is shown in [12]. In that paper, it is explained that an MIP-solver that is run on the same problem on two different computing systems can show a very large difference in performance. Even more surprising, this variability can also occur on the same system. The explanation for this phenomenon is that MIP-solvers rely on scores to make choices (e.g. branching decisions) among candidates while looking for a solution. These scores can show some small rounding errors that vary between computing systems and in the case of actual ties some arbitrary choices is made. This arbitrary choice could be based on something as simple as the order of an array. For example, a branching decision among seemingly equivalent variables can be influenced by which one was added to the model first.

The above is not a hypothetical or rare phenomenon. In [12], it is shown permuting the order of constraints and variables can strongly affect algorithmic performance.

6.2 Possible Solution

The best way to deal with these issues is transparency. Transparency comes in many different shades and it is not the objective of this paper to present a ready-made solution, but it could be beneficial to future scientific endeavors to think about how results are made public.

To be able to judge the results published in an academic paper it is important that they can be reproduced. This is by definition impossible if the problem instances are not available in some form. If the problem instances are generated in some random and reproducible way the experiment can be repeated, but there is still no way to repeat the experiments of the original publication. This could be dealt with by demanding that the set of problem instances is published in an easily accessible way along with the article. Some kind of public repository is an option for this. An initiative like `miplib.zib.de` is a start, but as of now it only contains 361 problem instances spread out over a large number different problems.

However, due to the proprietary nature of some data not all problem instances can be made public. Depositing such a problem at some kind of neutral third party that is able to run an algorithm for an interested researcher could be solution in these cases.

Some kind of database that shows what kind of algorithms have been applied to some problem would speed up the way in which research is done. This would provide an overview over the large amount of papers that are produced. An example from the medical science shows what this could look like. The International Committee of Medical Journal Editors requires that all medical trials are registered with `clinicaltrials.gov` as a prerequisite for publication of trial results.

Such a database does not completely prevent the possibility of cherry picking but it will at least result in a nice overview of what has been tried and also allows for negative results to be made public without being published in a journal article.

The greatest opportunity lies in the sharing of code. If code is required to be made public along with an article presenting results, doubts about code quality and algorithmic choices can be easily addressed. The availability of code together with the publication of problem instances leads to simple reproduction and verification of results. Testing how well the results translate to other instances is made easier.

A comprehensive open source optimization library would be even better. Open source projects can be very successful, see for example the Python programming language, the R project for statistical computing or the Linux operating system. If such a thing would exist for optimization it would make applying different algorithms to problems a lot easier. No longer would scientists have to write their own implementations of algorithms to be able to test them. Also, comparison of performance would be made a bit easier if an algorithm is available in an open source library so there are no doubts about the way it was implemented.

Of course, computer code can also be deemed proprietary in some cases. Here, a neutral third party could also serve the purpose of making experiments reproducible without making everything public.

Overfitting of an algorithm on a set of problem instances can be dealt with in two ways. The simplest one is applying the algorithm to problem instances that it was not initially tested on. This could be done after a result is published, but it can also be incorporated in the development of the algorithm by splitting the available problem instances into a training set and a test set. Similarly to the way in which this scheme is applied in machine learning, the algorithm can be tuned on the training set and its performance judged on the test set.

Addressing the issue of performance variability is the toughest nut to crack. This issue will always exist in some form unless all computational experiments are run within the same (virtual) computing system. However, such a standardized system would be one among many and it is hard to see how it could be designed to allow maximal crossover between the results of experiments and applications in the real world. That does not mean performance variability is something that can just be ignored when publishing results. More research could shed light on what algorithms are particularly affected by it and how the effects could be alleviated.

Chapter 7

Conclusion

This report has attempted to compare the performance of Benders' decomposition, Column & Constraint Generation and the standard Gurobi MIP-solver for the Demand Robust Location-Transportation problem with discrete uncertainty sets and the performance of Benders' decomposition and Column & Constraint Generation on said problem with polyhedral uncertainty sets. The general picture is that Column & Constraint Generation performs very well for both classes of uncertainty sets. Especially the low number of required scenarios for an optimal solution is noteworthy. Also, the small number of required iterations protects the algorithm form being bogged down by a hard subproblem. If that property holds for a wider range of problems it is a nice addition to the toolkit of demand robust optimization.

The remarkable difference in the performance of the MIP-solver for changes in its settings is something that raises questions about the very specific nature of results obtained by experiments like these. This is important to note because both Column & Constraint Generation and Benders' decomposition lean heavily on general MIP and LP solvers such as Gurobi or Cplex. This is a trait that is shared among a large number of decomposition approaches to solving mixed integer programs.

The performance of these algorithms is affected by the choice of solver and its settings. Therefore it is strange that usually these choices are merely mentioned and not justified when they are used in an algorithm. In practice they can greatly affect how fast a problem is solved and they should not be overlooked when evaluating the results of an experiment.

The reproducibility and general nature of results obtained by computational experiments could be improved by a larger emphasis on openness regarding problem instances and algorithmic implementation within the scientific community.

Bibliography

[1] Y. An and B. Zeng. “Exploring the modeling capacity of two-stage robust optimization: Variants of robust unit commitment model”. In: *IEEE Transactions on Power Systems* 30.1 (2015), pp. 109–122.

[2] Y. An, B. Zeng, Y. Zhang, and L. Zhao. “Reliable p-median facility location problem: two-stage robust models and algorithms”. In: *Transportation Research Part B: Methodological* 64 (2014), pp. 54–72.

[3] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. “Adjustable robust solutions of uncertain linear programs”. In: *Mathematical Programming* 99.2 (2004), pp. 351–376.

[4] J. F. Benders. “Partitioning procedures for solving mixed-variables programming problems”. In: *Numerische mathematik* 4.1 (1962), pp. 238–252.

[5] D. Bertsimas and M. Sim. “Robust discrete optimization and network flows”. In: *Mathematical programming* 98.1-3 (2003), pp. 49–71.

[6] D. Bertsimas, E. Litvinov, X. A. Sun, J. Zhao, and T. Zheng. “Adaptive robust optimization for the security constrained unit commitment problem”. In: *IEEE Transactions on Power Systems* 28.1 (2013), pp. 52–63.

[7] R. E. Bixby. “A brief history of linear and mixed-integer programming computation”. In: *Documenta Mathematica* (2012), pp. 107–121.

[8] S. Cicerone, G. DAngelo, G. Di Stefano, D. Frigioni, A. Navarra, M. Schachtebeck, and A. Schöbel. “Recoverable robustness in shunting and timetabling”. In: *Robust and online large-scale optimization.* Springer, 2009, pp. 28–60.

[9] A. M. Costa. “A survey on benders decomposition applied to fixed-charge network design problems”. In: *Computers & operations research* 32.6 (2005), pp. 1429–1450.

[10] G. Cote and M. A. Laughton. “Large-scale mixed integer programming: Benders-type heuristics”. In: *European Journal of Operational Research* 16.3 (1984), pp. 327–333.

[11] K. Dhamdhere, V. Goyal, R. Ravi, and M. Singh. “How to pay, come what may: Approximation algorithms for demand-robust covering problems”. In: *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS’05)*. IEEE. 2005, pp. 367–376.

[12] M. Fischetti, A. Lodi, M. Monaci, D. Salvagnin, and A. Tramontani. “Improving branch-and-cut performance by random sampling”. In: *Mathematical Programming Computation* 8.1 (2016), pp. 113–132.

[13] V. Gabrel, M. Lacroix, C. Murat, and N. Remli. “Robust location transportation problems under uncertain demands”. In: *Discrete Applied Mathematics* 164 (2014), pp. 100–111.

[14] R. A. Jabr, I. Džafić, and B. C. Pal. “Robust optimization of storage investment on transmission networks”. In: *IEEE Transactions on Power Systems* 30.1 (2015), pp. 531–539.

[15] C. Lee, C. Liu, S. Mehrotra, and M. Shahidehpour. “Modeling transmission line constraints in two-stage robust unit commitment problem”. In: *IEEE Transactions on Power Systems* 29.3 (2014), pp. 1221–1231.

[16] C. Liebchen, M. Lübbecke, R. Möhring, and S. Stiller. “The concept of recoverable robustness, linear programming recovery, and railway applications”. In: *Robust and online large-scale optimization*. Springer, 2009, pp. 1–27.

[17] T. L. Magnanti and R. T. Wong. “Accelerating Benders decomposition: Algorithmic enhancement and model selection criteria”. In: *Operations research* 29.3 (1981), pp. 464–484.

[18] D. McDaniel and M. Devine. “A modified Benders’ partitioning algorithm for mixed integer programming”. In: *Management Science* 24.3 (1977), pp. 312–319.

[19] A. Mercier, J. F. Cordeau, and F. Soumis. “A computational study of Benders decomposition for the integrated aircraft routing and crew scheduling problem”. In: *Computers & Operations Research* 32.6 (2005), pp. 1451–1476.

[20] R. H. Pearce and M. Forbes. “Disaggregated Benders Decomposition for solving a Network Maintenance Scheduling Problem”. In: *arXiv preprint arXiv:1603.02378* (2016).

[21] T. Santoso, S. Ahmed, M. Goetschalckx, and A. Shapiro. “A stochastic programming approach for supply chain network design under uncertainty”. In: *European Journal of Operational Research* 167.1 (2005), pp. 96–115.

[22] L. Tang, W. Jiang, and G. K. Saharidis. “An improved Benders decomposition algorithm for the logistics facility location problem with capacity expansions”. In: *Annals of operations research* 210.1 (2013), pp. 165–190.

[23] W. Wei, F. Liu, S. Mei, and Yunhe Hou. “Robust energy and reserve dispatch under variable renewable generation”. In: *IEEE Transactions on Smart Grid* 6.1 (2015), pp. 369–380.

[24] B. Zeng and L. Zhao. “Solving two-stage robust optimization problems using a column-and-constraint generation method”. In: *Operations Research Letters* 41.5 (2013), pp. 457–461.

[25] M. Zugno and A. J. Conejo. “A robust optimization approach to energy and reserve dispatch in electricity markets”. In: *European Journal of Operational Research* 247.2 (2015), pp. 659–671.