Optimizing marine insurance brokerage

by
Gunvor Lemvik

Master of Science Thesis in
Informatics

Department of Informatics
University of Bergen

June 2019

Abstract

Insurance coverages for marine shipping fleets are often created by a broker, who negotiates underwriter insurance offers on behalf of the fleet owner. The broker evaluates different combinations of offers, searching for the best insurance coverage for their customers. The coverage must have a minimal price, while earning the broker a large enough commission. This time consuming and difficult task is currently performed manually.

We present a bilinear model of the broker problem, and prove that solving it is NP-hard. We propose three different solution methods: (1) solving the model exactly using the commercially available software Baron, (2) alternatingly solving linearized subproblems and (3) using a metaheuristic. Results from computational experiments show that the second solution method outperforms Baron, both with respect to running time and objective function value.

The model and the alternating algorithm constitute a powerful tool for the brokers. It yields solutions with prices close to provable lower bounds in less than a second for most of the test instances.

Acknowledgements

I would like to thank my supervisor Professor Dag Haugland. I greatly appreciate his detailed feedback, encouragement and dedication in guiding me through this process. I could not have wished for a better supervisor. A special thanks to Edge Group AS, and especially Tore Ingvar Hellebø, Ida Abrahamsen-Sangedal and Nils B. Rokne, for a fruitful cooperation. Their hospitality and the time they have given me has resulted in a very interesting and enjoyable project.

I would also like to thank Tommy Odland for reading through a thesis draft and providing valuable feedback. Finally, I would like to thank my fellow students at the Faculty of Mathematics and Natural Sciences for wonderful years in Bergen, filled with laughter and fun.

Contents

List of Figures

List of Tables

Notation

The following notation will be used throughout the thesis.

Mathematical model

$\mathcal{P}$ – All products.

$\mathcal{U}$ – All underwriters.

$\mathcal{S}$ – All ships.

$\mathcal{U}_p$ – Underwriters offering product p.

$\mathcal{S}_p$ – Ships to be insured for product p.

$\mathcal{I}_p$ – Underwriters included in product p.

$\mathcal{C}_p$ – Claims lead candidates for product p.

$\mathcal{W}_p$ – Underwriters wanting shares smaller than or equal to claims lead share for product p.

τ – Total rating of an insurance coverage.

δ – Largest discount split difference of an insurance coverage.

Abbreviations

uwr – Underwriter.

BP – Broker Problem.

MC – Model Core.

Edge – Edge Group AS.

UiB – University of Bergen.

AMPL – A Mathematical Programming Language.

Baron – Branch-And-Reduce Optimization Navigator.

CBC – Coin-or Branch and Cut.

ALT – Alternating algorithm.

PSO – Particle swarm optimization.

SA – Simulated annealing.

Chapter 1

Introduction

With a fleet of ships, *fleet owners* transport cargo by sea. Marine shipping, like all other shipping industries, involves risk, and so it is important for the fleet owners to insure their fleet and cargo. *Underwriters* provide the fleet owners with insurance offers, in which their insurance price and conditions are specified. With fleet owners as customers, *brokers* negotiate these offers. Their goal is to create an insurance coverage at a minimal customer price, while also earning a commission. We refer to this task as the *Broker Problem (BP)*.

Marine brokerage is a traditional field where most tasks are performed manually. New underwriter offers are usually negotiated from previous offers. As a result, the brokerage relies heavily upon past experience and relationships with the underwriters. Because of this, the brokers must consider both quantitative and qualitative properties when evaluating the offers.

This thesis is the result of a collaboration between the marine brokerage company Edge Group AS (Edge) and the University of Bergen (UiB). Edge wants to explore the possibility of using optimization for creating insurance coverages on behalf of their customers. The BP can be formulated as an optimization problem, where the objective is to minimize the customer price, and the constraints correspond to the conditions specified by the customer and the underwriters. To the best of our knowledge, there does not exist any optimization model describing this problem, nor any solution approaches.

In this thesis we formulate a model describing the BP, and prove that solving it is NP-hard. We perform computational experiments for comparing different solution methods. The best solution method finds close to optimal solutions in few seconds for instances of varying sizes—outperforming the proprietary global optimization solver Baron.

We mainly focus on the quantitative aspect of the insurance coverage. To incorporate the qualitative aspects into the insurance coverage, an interplay between the broker and the model is necessary. The insurance coverage created by the model will serve as a basis for further brokering, rather than as a final solution. Re-solving the model for new offers gives a new basis for further brokering, and so forth. Because of this interplay, finding a good solution quickly may be more valuable than performing an extensive search for an optimal solution. We examine the trade-off between running time and solution quality, by comparing several solution methods.

The model and solution methods are all implemented from scratch, using the programming languages *A Mathematical Programming Language (AMPL)*, which is used for algebraic modeling, and *Python*, which is one of the fastest growing programming languages. AMPL has great readability for implementation of mathematical models and offer a wide range of solvers, but it is limited with respect to implementing more complex solution methods. With Python we have great freedom for implementing solution methods, and the optimization package `or-tools` created by Google provide a variety of solvers. Since Python is not a modeling language, the readability is arguably worse than for AMPL.

The structure of the thesis is presented below.

Chapter 1 – Introduction The first part of this chapter contains necessary background information, and presents the goal of the thesis. Next we describe the motivation for solving the BP, and then we give a simple problem description by means of examples. Finally we present previous work regarding the use optimization in the field of marine insurance and marine brokerage.

Chapter 2 – Model formulation In this chapter we give the complete BP description, and present a mathematical model which defines the problem in its entirety. We continue by discussing a couple of model extensions and some model properties, before proving that solving the BP is NP-hard.

Chapter 3 – Solution methods This chapter presents three different solution approaches for solving the BP model, along with one solution method for each of the model extensions. For solving the model, we examine a global optimization algorithm, a linearization technique and a metaheuristic algorithm. The model extensions are solved by a heuristic algorithm and a linear program, respectively.

Chapter 4 – A simulated annealing approach In this chapter we present the problem specific choices for the metaheuristic approach to solving the BP. This

includes generation of an initial solution, design of heuristics and parameter tuning. We devote a chapter to this solution method because of the many problem-specific choices it involves, as opposed to the straightforward application of the other solution methods proposed in Chapter 3.

Chapter 5 – Computational experiments We begin this chapter by describing the experimental setup and the test instances for the computational experiments. We report the results from solving the BP model and its extensions using the proposed solution methods, and finally we present our observations regarding these results.

Chapter 6 – Conclusion and future work In this final chapter, we summarize our work by restating the thesis goal and explaining how we reached it. We also propose some areas for future work.

1.1 Motivation

To avoid the risk of big losses, marine insurance underwriters usually do not want to provide insurance coverage for an entire fleet. This means that several underwriter offers must be combined in order to provide a complete insurance coverage. The broker examines which combination of offers results in the best solution, and consequently which underwriters to include in the insurance coverage. Consider the case where a broker is provided with 15 underwriter offers, and wants to create an insurance coverage consisting of 8 out of these offers. In this modest example, there are 6,435 possible combinations of underwriter offers for the broker to consider. This is a daunting manual task.

To complicate matters further, the price of each underwriter offer can only be calculated after the combination of underwriters is determined. This is because the calculation of the price involves two variables for each underwriter offer, whose values are dependent on the corresponding variables of all the other involved underwriters. This means that it is not possible to examine an underwriter offer separately—all the 6,435 possible combinations must be examined in order to guarantee that the best solution has been found.

The BP is also a multi-objective problem—an insurance coverage should have a minimal customer price and yield a maximal broker commission. The commission constitutes a certain percentage of the price, which means that finding a low-price insurance coverage while earning a significant commission is a contradictory goal.

It is clear from this section that finding a good solution to the BP by hand is a difficult task. Brokers currently spend a significant portion of their time attempting to solve this problem. Using an optimization model to quickly generate good insurance coverages would let the brokers spend their time more efficiently, i.e., use more of their time negotiating better offers. Since both the model and the solution methods are implemented from scratch in a free programming language, with the possibility of using free solvers, this approach for solving the BP is free of charge. The brokers can save both time and money, which would provide them with a competitive advantage of the other marine brokerage companies.

1.2 Simple problem description

In this section we give a *minimal working example* of the BP problem. This is a smallest possible example demonstrating the key properties of the problem. By solving increasingly difficult subproblems, we demonstrate different properties of the problem. This is done in an effort to let the reader gradually familiarize himself with the BP, before going into the details in the next chapter.

Let $\mathcal{S}$ be the set of ships constituting the customer fleet and let $\mathcal{U}$ be the set of candidate underwriters. For every ship $s \in \mathcal{S}$, each underwriter $u \in \mathcal{U}$ offers an insurance rate ($\text{rate}_{s,u}$), which indicates the percentage of the ship value (value_s) that must be paid for the underwriter to insure the ship. We define the *total price* (total_price_u) of an underwriter offer to be the cost of insuring the entire fleet, if that underwriter were to cover the entire insurance. For each underwriter $u \in \mathcal{U}$, the total price is calculated as the sum over all the customer ships $s \in \mathcal{S}$, of the ship rate multiplied with the ship value,

$$\text{total\_price}_u = \sum_{s \in \mathcal{S}} \text{value}_s \cdot \left(\text{rate}_{s,u} \cdot 10^{-2}\right). \tag{1.1}$$

The rates are conventionally presented as percentages rather than as decimals. For the calculation of total_price_u, the rates are therefore multiplied by 10^{-2}.

In addition to the rate, each underwriter $u \in \mathcal{U}$ offers a price discount (total_discount_u). This discount is split between the customer and the broker, such that the customer discount and the broker discount sums up to the total discount. Furthermore, the underwriters specify the minimal and maximal acceptable shares (share_u) they may be assigned. The customer specifies the broker share (broker_share), which is the percentage of the fleet for which the customer requests insurance. For each underwriter $u \in \mathcal{U}$, the customer price (customer_price_u) and the broker commission ($\text{broker\_commission}_u$) is calculated as products of the total price, the share and the

discount split, divided by the broker share

$$\text{customer\_price}_u = \frac{\text{total\_price}_u \cdot \text{share}_u \cdot (1 - \text{customer\_discount}_u)}{\text{broker\_share}}, \tag{1.2}$$

$$\text{broker\_commission}_u = \frac{\text{total\_price}_u \cdot \text{share}_u \cdot \text{broker\_discount}_u}{\text{broker\_share}}. \tag{1.3}$$

The broker aims to minimize the customer price, while earning a commission that is at least 5 % of the customer price.

Problem 1.1 (Minimal Working Example). *Consider a customer with a fleet consisting of two ships, and three candidate underwriters. The customer wants a complete insurance coverage, i.e.,* broker_share *is equal to 1. The ship values, the rates offered by the underwriters and the underwriter conditions are presented in the tables below.*

Table 1.1: Ship values and underwriter rates.

		rate		
	value (€)	uwr_1	uwr_2	uwr_3
ship_1	1,000,000	0.02	0.02	0.01
ship_2	1,500,000	0.01	0.01	0.02

Table 1.2: Underwriter conditions.

	min share	max share	total discount
uwr_1	0.2	0.4	0.05
uwr_2	0.2	0.4	0.10
uwr_3	0.3	0.6	0.20

The goal is to distribute shares among the underwriters such that the shares sum up to the broker share, and split the total discount between the customer and the broker in such a way that the customer price is minimal while the broker commission is at least 5 % of the customer price. ⌟

We solve Problem 1.1 step by step, where each step is presented as an example. For each example we provide the corresponding optimal solution, which changes as we consider more properties of the problem.

Example 1.2 (Rates). As a first step we disregard the underwriter conditions in Table 1.2 and the constraint specifying that the broker commission must be at least 5 % of the customer price. We minimize the customer price by concentrating solely on the ship values and the rates provided by the underwriters—both found in Table 1.1 Using the formula from Equation (1.1), we calculate the total price of the complete insurance coverage for the three underwriters.

$$\begin{aligned}
\text{uwr}_1 &: \ (1/1) \cdot (1{,}000{,}000 \cdot 0.02 \cdot 10^{-2} + 1{,}500{,}000 \cdot 0.01 \cdot 10^{-2}) = 350 \\
\text{uwr}_2 &: \ (1/1) \cdot (1{,}000{,}000 \cdot 0.02 \cdot 10^{-2} + 1{,}500{,}000 \cdot 0.01 \cdot 10^{-2}) = 350 \\
\text{uwr}_3 &: \ (1/1) \cdot (1{,}000{,}000 \cdot 0.01 \cdot 10^{-2} + 1{,}500{,}000 \cdot 0.02 \cdot 10^{-2}) = 400
\end{aligned}$$

We see that uwr_1 and uwr_2 provide the cheapest offers, while the offer from uwr_3 is more expensive. An optimal insurance coverage at this step excludes uwr_3, and includes either uwr_1 or uwr_2, or both, at the customer price of 350 euros. ⌟

Example 1.3 (Rates and shares). In the second step, we distribute shares among the underwriters, satisfying the conditions in the first two columns of Table 1.2. In the previous example we found that the optimal price for an insurance coverage is achieved by including either uwr_1 or uwr_2, or both. At the current step this turns out to be infeasible, since both underwriters have a condition of covering at most 40 % of the insurance, which sums to less than 100 %. We are forced to include uwr_3 in the insurance coverage. Since its offer is the most expensive, we want to give it the smallest possible share, i.e., the remaining 20 %. However, uwr_3 wants to cover at least 30 % of the insurance, and we need to reduce the share of either uwr_1 or uwr_2 to 30 % in order to satisfy all underwriter conditions.

We have found an optimal distribution of shares, and calculate the corresponding customer price at this step, using the formula in Equation (1.2) (where $\text{customer\_discount}_{u_j}$ is set to zero).

$$\sum_{u \in \mathcal{U}} \text{customer\_price}_u = 350 \cdot 0.4 + 350 \cdot 0.3 + 400 \cdot 0.3 = 365.$$

The minimal customer price at this step is 365 euros, which is slightly higher than in the previous example. ⌟

Example 1.4 (Rates, shares and discounts). In the final step, we additionally split the discount provided by the underwriters between the customer and the broker, and we include the condition that the broker commission must be at least 5 % of the customer price. The total discount is specified in the third column of Table 1.2.

We can no longer solve the problem by inspection or by a greedy approach, as we did in the preceding examples. Even for an instance as small as Problem 1.1, there are infinitely many combinations of share distributions and discount splits. An

algebraic model of the problem instance is formulated below, where customer_price_u and $\text{broker\_commission}_u$ is computed by Equations (1.1)-(1.3).

$$\begin{aligned}
\text{minimize} \quad & \sum_{u \in \mathcal{U}} \text{customer_price}_u \\
\text{subject to} \quad & \sum_{u \in \mathcal{U}} \text{broker_commission}_u \geq 0.05 \cdot \sum_{u \in \mathcal{U}} \text{customer_price}_u \\
& \sum_{u \in \mathcal{U}} \text{share}_u = \text{broker_share} \\
& \text{customer_discount}_u + \text{broker_discount}_u = \text{total_discount}_u, \quad \text{for all } u \in \mathcal{U}
\end{aligned}$$

$$\begin{aligned}
\text{where } \text{customer_discount}_u &\in [0, \text{total_discount}_u], && \text{for all } u \in \mathcal{U} \\
\text{broker_discount}_u &\in [0, \text{total_discount}_u], && \text{for all } u \in \mathcal{U} \\
\text{share}_u &\in \{0\} \cup [\text{min_share}_u, \text{max_share}_u], && \text{for all } u \in \mathcal{U}
\end{aligned}$$

We find an optimal solution by using the nonlinear solver Baron. This solution results in a customer price of 337 euros, and a broker commission of 17 euros. The corresponding share distribution and discount splits are presented in Table 1.3.

Table 1.3: Optimal solution to Problem 1.1. The values of shares and discounts are displayed with 1 and 2 decimals of precision, respectively.

	share	customer_discount	broker_discount
uwr_1	0.2	0.02	0.03
uwr_2	0.4	0.06	0.04
uwr_3	0.4	0.14	0.06

⌟

This section illustrates the difficulty of manually creating an optimal insurance coverage—even when the problem instance is simple.

1.3 Previous work

In this section we present previous work regarding the use of optimization and mathematical modeling in the field of marine insurance.

The most important job of a marine insurance broker is to negotiate underwriter insurance offers on behalf of its customer. The broker and the underwriters have opposite goals in this negotiation—the underwriters want to earn as much profit as

possible, while the broker wants to achieve the lowest possible price. Kihlstrom and Roth (1982) propose a game-theoretic model as a way to negotiate the terms of an insurance contract. The model is based on the solution to the classic bargaining problem described by Nash (1950), and studies how the degree of risk aversion affects the result of the insurance contract negotiation. The key takeaway is that when bargaining with a risk neutral insurer, a risk aversive client will pay a higher premium, for less coverage of the potential loss, than a less risk averse client.

Borch (1961) explains how mathematical modeling can be used to describe the traditional way of creating insurance contracts, where company supervisors make subjective choices, partly based on input from actuaries. He states that we can assume that there is some consistency to their subjective choices, and that this consistency can be expressed by utility functions, which the insurance company aims to maximize. These utility functions, which he refers to as the "utility of money", are used further in Borch (1962), where a model of the reinsurance market is proposed. He argues that a *Pareto optimal* insurance contract arrangement can be reached by seeing the problem as an n-person game. An arrangement is Pareto optimal if all involved parties receive the maximal possible utility, in the sense that a larger utility for any of them will lead to a smaller utility for at least one of the others. Extensive research of utility theory and competitive equilibrium within insurance is found in Chapters 2 and 3 of Borch et al. (1990).

Much effort has been put into solving bargaining problems, both within insurance and in general. This theory is useful for the brokering of insurance contracts, but does not solve the problem of creating an optimal insurance coverage with respect to the customer price. Smith (1968) addresses the current lack of literature on optimization of insurance purchase, for protection against casualty, which he compares to the problem of optimal inventory stockage under uncertainty. He derives several theorems for determining the optimal amount of insurance coverage for different probabilities of loss. Raviv (1979) addresses the problem of insurance purchases further by developing a general model. The model contains utility functions for both the insurance purchaser and the insurer. He shows that Pareto optimal insurance contracts involve a deductible and coinsurance of the losses above the deductible. This work is extended by Szpiro (1985), who derives explicit expressions for the optimal amounts of insurance to be purchased.

It is clear that the negotiation of insurance contracts is a popular area of research. The area of insurance purchasing, while not as popular, has also been studied in depth. For the latter, most of the focus has been upon the amount of insurance to purchase, in relation to the probability of risk and the amount of deductible. For the problem of constructing an insurance coverage with a minimal customer price, without the consideration of risk and deductible, there does to the best of our knowledge not exist any documented optimization model.

Chapter 2

Model formulation

In this chapter we give a complete problem definition by expanding the minimal working example defined in Problem 1.1. We continue by describing a mathematical model of the BP, defining a couple of model extensions and discussing some model properties. At the end of the chapter we prove that the BP is NP-hard, by constructing a reduction from the Subset-Sum problem.

2.1 Problem definition

Let the BP be equal to the simple problem description defined by the minimal working example in Problem 1.1. So far, we have not discussed what kind of insurance the underwriters offer. We refer to the different types of insurance as *products*. Examples of products within marine insurance are insurance of hull and machinery and insurance of loss of hire. The customer chooses one or several insurance products for each ship, and specifies the broker share for each of the products. Each underwriter specifies the products they offer, along with corresponding rates and discounts for each product. Every product is now associated with a subset of ships and a subset of underwriters, which means that the BP expands to creating several insurance coverages—one for each product.

The minimal working example states that the broker commission must be at least 5 % of the customer price. We now expand to let this *minimal commission factor* vary for the different products. The value of this factor typically lies in the interval 5 %-10 %, and is determined by the broker.

In any insurance coverage, one underwriter is chosen to be *claims lead*. The main task of the claims lead is to handle the salvage operation on behalf of all the involved

underwriters, in case of a casualty. For each product, a subset of underwriters are chosen as claims lead candidates.

The customer needs to specify whether or not the insurance coverage constructed by the broker is to have a claims lead. If the broker share is set to 100 %, the coverage must contain exactly one claims lead. If the broker share is less than 100 %, for instance 60 %, this means that the remaining 40 % of the insurance coverage is being handled by someone else. In this case, the claims lead can be in either the first or the second insurance coverage, but not in both.

All underwriters must specify the minimal and maximal acceptable share they may be assigned. In addition to specifying these limits, they have the option of restricting their share size to be smaller than or equal to the claims lead share. This may be desirable, since they do not have control of the salvage operation.

Since the brokers must consider qualitative properties of the insurance coverage, as well as quantitative, they may have preferences regarding which underwriters are included in the different products. Because of this, a set of *underwriter preferences* is associated with every product, so that each underwriter in this set must be included in the coverage of the corresponding product.

The underwriters may also have *inter-product demands*. Consider the set of products $\mathcal{P} = \{p, p', p^*\}$ and the underwriter $u \in \mathcal{U}$ who provides all three products. Underwriter u may define a condition of offering product p if and only if it is included in the insurance coverage of product p'. This makes the task of creating an insurance coverage for several products significantly harder. It links all the products together in such a way that the insurance coverage of each product is influenced by the insurance coverage of every other product. These inter-product demands also exist for potential claims leads.

Formally, the problem under study is defined as follows.

Definition 2.1 (The Broker Problem (BP)).

For each product,

- *find a distribution of shares that sum up to the corresponding broker share, while not violating the share limits specified by the underwriters,*
- *split the underwriter discounts between the customer and the broker,*
- *include the correct number of claims lead,*
- *satisfy all inter-product demands,*

in such a way that the broker commission is larger than or equal to the specified percentage of the customer price, and the total customer price is minimal.

⌟

2.2 Mathematical model

In this section we present the mathematical model describing the BP. We list all sets, parameters and decision variables, define the objective function, and present all the constraints along with descriptions.

2.2.1 Sets

$\mathcal{P}$	All products p to be included in the insurance coverage.
$\mathcal{S}$	All ships s constituting the customer fleet.
$\mathcal{U}$	All underwriters u that are candidates for the insurance coverage.
$\mathcal{S}_p$	The subset of ships to be insured for product $p \in \mathcal{P}$.
$\mathcal{U}_p$	The subset of underwriters who offer product $p \in \mathcal{P}$.
$\mathcal{I}_p$	The subset of underwriters who must be included in covering product $p \in \mathcal{P}$.
$\mathcal{C}_p$	The subset of underwriters who are claims lead candidates for product $p \in \mathcal{P}$.
$\mathcal{W}_p$	The subset of underwriters who want shares smaller than or equal to the claims lead share for product $p \in \mathcal{P}$.

2.2.2 Parameters

$\text{value}_{p,s}$	Value of ship s for product p.
$\text{rate}_{p,s,u}$	Rate provided by underwriter u for ship s and product p.
$\text{min\_share}_{p,u}$	Lowest acceptable share for underwriter u and product p.
$\text{max\_share}_{p,u}$	Highest acceptable share for underwriter u and product p.
$\text{total\_discount}_{p,u}$	Total discount provided by underwriter u and product p.
min_ratio_p	Lowest acceptable ratio of the broker commission divided by the customer price, for each product p.
max_price	Highest acceptable total customer price.
max_commission	Highest acceptable total broker commission.
broker_share_p	The insurance coverage percentage to be covered by the broker for product p.
num_claims_lead_p	Indicates whether the insurance coverage of product p contains the claims lead or not.

$\text{u\_demands}_{p,u,p'}$	Indicates whether underwriter u offers to cover product p if and only if it also covers product p'.
$\text{cl\_demands}_{p,u,p'}$	Indicates whether underwriter u offers to be claims lead of product p if and only if it is also claims lead of product p'.

2.2.3 Decision variables

$\text{share}_{p,u} \in [0,1]$	Amount of coverage of product p assigned to underwriter u.
$\text{customer\_discount}_{p,u} \in [0,1]$	Amount of discount given to the customer from underwriter u for product p.
$\text{broker\_discount}_{p,u} \in [0,1]$	Amount of discount given to the broker from underwriter u for product p.
$\text{included}_{p,u} \in \{0,1\}$	Equal to 1 if underwriter u is included in the insurance coverage of product p, equal to 0 if not.
$\text{claims\_lead}_{p,u} \in \{0,1\}$	Equal to 1 if underwriter u is claims lead for product p, equal to 0 if not.

2.2.4 Derived parameters and variables

The total price of assigning the complete insurance coverage of product p to underwriter u, for each product $p \in \mathcal{P}$ and each underwriter $u \in \mathcal{U}_p$:

$$\text{total\_price}_{p,u} = \sum_{s \in \mathcal{S}_p} \text{value}_{p,s} \cdot \left(\text{rate}_{p,s,u} \cdot 10^{-2}\right). \tag{2.1}$$

Computation of the customer price and the broker commission for each product $p \in \mathcal{P}$ and each underwriter $u \in \mathcal{U}_p$.

$$\text{customer\_price}_{p,u} = \frac{\text{total\_price}_{p,u} \cdot \text{share}_{p,u} \cdot (1 - \text{customer\_discount}_{p,u})}{\text{broker\_share}_p} \tag{2.2}$$

$$\text{broker\_commission}_{p,u} = \frac{\text{total\_price}_{p,u} \cdot \text{share}_{p,u} \cdot \text{broker\_discount}_{p,u}}{\text{broker\_share}_p} \tag{2.3}$$

2.2.5 Objective function

The goal of the broker is to minimize the total customer price.

$$\text{minimize} \sum_{p \in \mathcal{P}} \sum_{u \in \mathcal{U}_p} \text{customer\_price}_{p,u} \tag{2.4}$$

2.2.6 Constraints

The total customer price and the total broker commission must be smaller than or equal to their respective upper bounds.

$$\sum_{p \in \mathcal{P}} \sum_{u \in \mathcal{U}_p} \text{customer\_price}_{p,u} \leq \text{max\_price} \tag{2.5}$$

$$\sum_{p \in \mathcal{P}} \sum_{u \in \mathcal{U}_p} \text{broker\_commission}_{p,u} \leq \text{max\_commission} \tag{2.6}$$

The broker commission must be larger than or equal to the minimal acceptable percentage of the customer price, for each product $p \in \mathcal{P}$.

$$\sum_{u \in \mathcal{U}_p} \text{broker\_commission}_{p,u} \geq \text{min\_ratio}_p \cdot \sum_{u \in \mathcal{U}_p} \text{customer\_price}_{p,u} \tag{2.7}$$

For each product $p \in \mathcal{P}$ and each underwriter $u \in \mathcal{U}_p$, the customer discount and broker discount must sum to the total discount provided by underwriter u, if it is included in the coverage of product p, and to zero otherwise.

$$\text{broker\_discount}_{p,u} + \text{customer\_discount}_{p,u} = \text{total\_discount}_{p,u} \cdot \text{included}_{p,u} \tag{2.8}$$

Analogously, for each product $p \in \mathcal{P}$ and each underwriter $u \in \mathcal{U}_p$, the assigned share must lie in between the minimal and maximal acceptable shares for underwriter u, if it is included in the coverage of product p, and be equal to zero otherwise.

$$\text{min\_share}_{p,u} \cdot \text{included}_{p,u} \leq \text{share}_{p,u} \leq \text{max\_share}_{p,u} \cdot \text{included}_{p,u} \tag{2.9}$$

All shares must add up to the broker share, for each product $p \in \mathcal{P}$.

$$\sum_{u \in \mathcal{U}_p} \text{share}_{p,u} = \text{broker\_share}_p \tag{2.10}$$

The insurance coverage must include the specified number of claims lead, for each product $p \in \mathcal{P}$.

$$\sum_{u \in \mathcal{C}_p} \text{claims\_lead}_{p,u} = \text{number\_of\_claims\_lead}_p \tag{2.11}$$

An underwriter can only be claims lead of a product if it is also claims lead of all other products in its claims lead demands. The underwriter must also be included in the insurance coverage in order to be claims lead. This must hold for each product $p \in \mathcal{P}$, each claims lead candidate $u \in \mathcal{C}_p$ and all products $\{p' \in \mathcal{P} \mid \text{cl\_demands}_{p,u,p'} = 1\}$.

$$\text{claims\_lead}_{p,u} \leq \text{claims\_lead}_{p',u} \tag{2.12}$$
$$\text{claims\_lead}_{p,u} \leq \text{included}_{p,u} \tag{2.13}$$

All underwriters $u \in \mathcal{I}_p$ must be included in the insurance coverage, for each product $p \in \mathcal{P}$.

$$\text{included}_{p,u} = 1 \tag{2.14}$$

The share assigned to underwriter u can not be larger than the share of the claims lead share. This must hold for each product $p \in \mathcal{P}$, each underwriter $u \in \mathcal{W}_p$ and each claims lead candidate $u' \in \mathcal{C}_p$.

$$\text{share}_{p,u} \leq \text{share}_{p,u'} + (1 - \text{claims\_lead}_{p,u'}) \tag{2.15}$$

An underwriter can only be included in a product if it is also included in all other products of its product demands. This must hold for each product $p \in \mathcal{P}$, each underwriter $u \in \mathcal{U}_p$ and all products $\{p' \in \mathcal{P} \mid \text{u\_demands}_{p,u,p'} = 1\}$.

$$\text{included}_{p,u} \leq \text{included}_{p',u} \tag{2.16}$$

2.3 Model extensions

The primary objective of the BP is to minimize the customer price. If there are several solutions with the same customer price, all of them will be considered by the mathematical model to be equally good. A broker would however favor some solutions over others, because of certain desirable properties. In this section we introduce two model extensions which describe such solution properties. Both of these extensions are handled by solution methods, rather than being incorporated into the mathematical model, so that they are treated as secondary objectives.

2.3.1 Underwriter ratings

Every underwriter is associated with a *priority rating* (priority_rating_u). This rating is a real number whose value is determined based on previous experiences and relationships. The rating represents to what degree the underwriter is desirable to include in an insurance coverage. One factor of desirableness is how quickly the underwriter usually has issued insurance payouts in the past.

The mathematical model only distinguishes between underwriters in regards to their contribution to the customer price, but a broker would additionally consider the underwriter priority ratings. A slightly higher customer price could be favorable, if it leads to a smoother handling of casualties.

For an arbitrary insurance coverage, we let $\mathcal{U}^* \subseteq \mathcal{U}$ denote the set of underwriters that are included in at least one product of the insurance coverage. We denote the *total rating* of an insurance coverage by τ, and define it to be the sum of the priority ratings of all underwriters $u \in \mathcal{U}^*$,

$$\tau = \sum_{u \in \mathcal{U}^*} \text{priority\_rating}_u. \tag{2.17}$$

If an underwriter with low priority rating is included in one product of the insurance coverage, the "damage is already done". Including it in more products will not worsen the casualty handling. It is desirable to maximize τ, so that underwriters with low priority are not included in the insurance coverage.

2.3.2 Even discount splits

When a customer is presented with the price of an insurance coverage, he may ask the broker how much discount he received from the different underwriters. For this reason, it is desirable to split the total discount evenly, rather than giving the customer zero discount from one underwriter and all the discount from another. The mathematical model does not take this into account. Even though an uneven split does not need to affect the customer price, it affects the relationship between the customer and the underwriters.

Let $\mathcal{U}'_p \subseteq \mathcal{U}_p$ denote the set of underwriters that are included in the insurance coverage of product $p \in \mathcal{P}$. We denote the *largest discount split difference* by δ, and define it as the largest difference between the smallest and largest customer discount for the included underwriters $u \in \mathcal{U}'_p$ within each product $p \in \mathcal{P}$

$$\delta = \max_{p \in \mathcal{P}} \left(\max_{u \in \mathcal{U}'_p} (\text{customer\_disocunt}_{p,u}) - \min_{u \in \mathcal{U}'_p} (\text{customer\_discount}_{p,u}) \right). \tag{2.18}$$

It is desirable to minimize δ so that the discount split is as even as possible.

2.4 Model properties

A common way to discuss the size of a mathematical model is to consider the number of decision variables and constraints, which depend on the input instance size. We derive the number of decision variables and the number of constraints in the BP model with regards to a general instance. To see how the number of decision variables and constraints grow as the instance size increases, we use big-$\mathcal{O}$ notation.

The number of variables is:

$$\mathcal{O}\left(5\left|\mathcal{P}\right|\left|\mathcal{U}_p\right|\right) = \mathcal{O}\left(\left|\mathcal{P}\right|\left|\mathcal{U}\right|\right)$$

The number of constraints is:

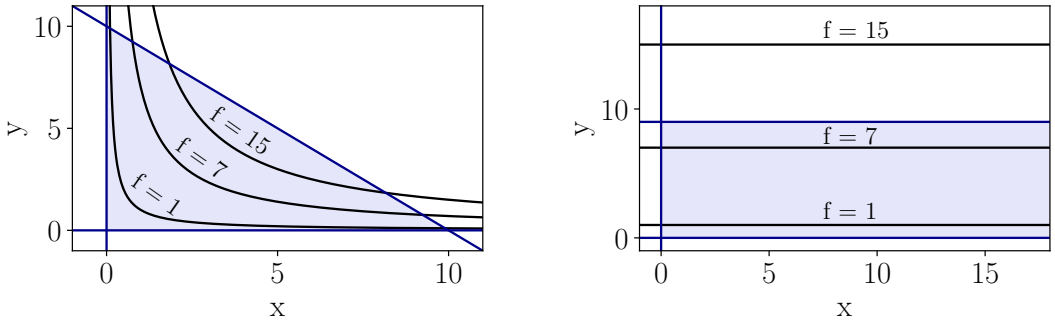
$$\begin{aligned}&\mathcal{O}\left(2+3\left|\mathcal{P}\right|+4\left|\mathcal{P}\right|\left|\mathcal{U}_p\right|+\left|\mathcal{P}\right|\left|\mathcal{I}_p\right|+\left|\mathcal{P}\right|\left|\mathcal{W}_p\right|\left|\mathcal{C}_p\right|+\left|\mathcal{P}\right|^2\left|\mathcal{U}_p\right|+\left|\mathcal{P}\right|^2\left|\mathcal{C}_p\right|\right)\\ =\ &\mathcal{O}\left(\left|\mathcal{P}\right|\left|\mathcal{U}\right|^2+\left|\mathcal{P}\right|^2\left|\mathcal{U}\right|\right)\end{aligned}$$

As the instance size increases, the number of decision variables and the number of constraints are bounded by a polynomial of degree 2 and 3, respectively. The number of constraints grows with a factor of $\mathcal{O}\left(\left|\mathcal{P}\right|+\left|\mathcal{U}\right|\right)$ faster than the number of variables.

The BP model contains real variables and binary variables. Most of the constraints in the model are linear, since no variables are multiplied by each other. However, in the computation of $\text{customer\_price}_{p,u}$ and $\text{broker\_commission}_{p,u}$ (Equations (2.2)-(2.3)) there are products involving the real variables $\text{share}_{p,u}$, $\text{customer\_discount}_{p,u}$ and $\text{broker\_discount}_{p,u}$. This makes the model nonlinear. There are no other variables causing the nonlinearity, which means that if either the variable $\text{share}_{p,u}$ or the variables $\text{customer\_discount}_{p,u}$ and $\text{broker\_discount}_{p,u}$ had been parameters, the problem would be linear. Such a problem is called a *bilinear problem*. Recognizing the bilinearity is important with regards to the choice of solution method. We give a simple example showing how a bilinear problem turns into a linear problem when one of the bilinear variables are fixed.

Example 2.2. Consider the problems P and Q below. We recognize that P becomes the same problem as Q if we fix $x = 1$. This simple modification turns a bilinear problem into a linear problem. Figure 2.1 visualizes how this affects the problem. The problem constraints are represented by blue lines, and the filled area represents the solution space. The black lines mark the objective function for different objective function values.

$$P: \quad \begin{array}{rl} \text{maximize} & f(x,y) = x \cdot y \\ \text{subject to} & x + y \leq 10 \\ \text{where} & x, y \geq 0 \end{array} \qquad Q: \quad \begin{array}{rl} \text{maximize} & f(y) = y \\ \text{subject to} & y \leq 9 \\ \text{where} & y \geq 0 \end{array}$$



(a) Graphic representation of P.

(b) Graphic representation of Q.

Figure 2.1

We see that the optimal solution to P is located somewhere along one of the constraints, while the optimal solution to Q has the favorable property of being located at a vertex. ⌟

2.5 Model complexity

In this section we examine the BP model complexity. We begin by defining a simplified version of the model, which we refer to as the *Model Core*. Because the Model Core is similar to the well-known *Subset-Sum Problem*, we use this fact to determine the BP model complexity. This is achieved by proving that solving the Model Core is at least as hard as solving the Subset-Sum Problem.

The BP model consists of several sets and parameters. To discuss the complexity of the model, we simplify by considering a single product. We omit the product index of sets and parameters for the duration of this section, as it is not needed. Furthermore, we leave some sets empty, and assign values of zero or infinity to certain parameters—see Table 2.1 for these assignments. As a result, several of the model constraints become redundant and are disregarded. The Model Core is defined by the remaining constraints.

Table 2.1: Set and parameter assignments for the Model Core.

Sets			Parameters		
$\mathcal{P}$	:=	$\{p\}$	min_ratio_p	:=	0
$\mathcal{I}_p$	:=	$\emptyset$	max_price	:=	∞
$\mathcal{C}_p$	:=	$\emptyset$	max_commissoin	:=	∞
$\mathcal{W}_p$	:=	$\emptyset$	num_claims_lead_p	:=	0

Problem 2.3 (Model Core (MC)).

Does there exist a subset of underwriters $\mathcal{U}' \subseteq \mathcal{U}$, along with a share distribution and discount splits for all underwriters $u \in \mathcal{U}'$, such that the following constraints are satisfied?

1. *The share of each underwriter u must either be zero or lie between* min_share_u *and* max_share_u*, for all $u \in \mathcal{U}'$:*

$$\text{min\_share}_u \leq \text{share}_u \leq \text{max\_share}_u. \tag{2.19}$$

2. *The sum of all shares must be equal to* broker_share*:*

$$\sum_{u \in \mathcal{U}'} \text{share}_u = \text{broker\_share}. \tag{2.20}$$

3. *The discount split from underwriter u must add up to* total_discount_u*, for all $u \in \mathcal{U}'$:*

$$\text{customer\_discount}_u + \text{broker\_discount}_u = \text{total\_discount}_u. \tag{2.21}$$

⌟

2.5.1 Determining existence of a solution

Whether or not a solution to the MC exists is determined by the share distribution and discount splits among the underwriters. The discount split from a single underwriter does not affect the other underwriters, nor does it imply that this underwriter should be included in the subset of underwriters $\mathcal{U}'$ (the insurance coverage). This means that we can choose discount splits satisfying Equation (2.21), for all the underwriters in $\mathcal{U}$, without affecting the existence of a solution. Distributing the shares among the underwriters, however, turns out to be a hard problem. Since the sum of all shares must be equal to broker_share, the assignment of a share to a single underwriter will affect all the other underwriters.

We need to determine whether there exists a share distribution satisfying the constraints (2.19) and (2.20). From these two constraints, it is clear that broker_share must lie between the sum of minimal shares and the sum of maximal shares.

$$\begin{aligned} \text{min\_share}_u \leq \quad & \text{share}_u \leq \quad \text{max\_share}_u, \text{ for all } u \in \mathcal{U}', \\ \text{which implies } \sum_{u \in \mathcal{U}'} \text{min\_share}_u \leq \quad & \sum_{u \in \mathcal{U}'} \text{share}_u \leq \sum_{u \in \mathcal{U}'} \text{max\_share}_u, \\ \text{and we get } \sum_{u \in \mathcal{U}'} \text{min\_share}_u \leq \text{broker\_share} \leq & \sum_{u \in \mathcal{U}'} \text{max\_share}_u. \end{aligned} \tag{2.22}$$

It is important to note that any solution to Equations (2.19) and (2.20) is a solution to Equation (2.22), but the converse does not hold. However, we are interested in the existence of a solution, and we give a proof that for any value of broker_share, for which Equation (2.22) holds, there must also exist a solution to Equations (2.19) and (2.20).

Proposition 2.4. *For any value of* broker_share *satisfying Equation* (2.22)*, there must exist a share distribution for which Equations* (2.19) *and* (2.20) *also hold.*

Proof. Any value of broker_share that satisfies Equation (2.22) must lie in the interval,

$$\text{broker\_share} \in \left[\sum_{u \in \mathcal{U}'} \text{min\_share}_u, \sum_{u \in \mathcal{U}'} \text{max\_share}_u \right].$$

We let the variable $\alpha \in [0, 1]$ indicate the position of broker_share in this interval,

$$\text{broker\_share} = \sum_{u \in \mathcal{U}'} \text{min\_share}_u + \alpha \left(\sum_{u \in \mathcal{U}'} \text{max\_share}_u - \sum_{u \in \mathcal{U}'} \text{min\_share}_u \right).$$

We use the value of α to determine a share distribution satisfying Equation (2.19).
For all $u \in \mathcal{U}'$, we let α determine the position of share_u in the interval $[\text{min\_share}_u, \text{max\_share}_u]$ of feasible values, i.e.,

$$\text{share}_u := \text{min\_share}_u + \alpha \left(\text{max\_share}_u - \text{min\_share}_u \right).$$

Since α is a variable in the interval $[0, 1]$, Equation (2.19) must be satisfied for this share distribution.

In order to verify that the proposed share distribution also satisfies Equation (2.20), we compute the sum of shares for all underwriters $u \in \mathcal{U}'$,

$$\begin{aligned}\sum_{u\in\mathcal{U}'} \text{share}_u &= \sum_{u\in\mathcal{U}'} (\text{min\_share}_u + \alpha\,(\text{max\_share}_u - \text{min\_share}_u)) \\ &= \sum_{u\in\mathcal{U}'} \text{min\_share}_u + \alpha \sum_{u\in\mathcal{U}'} (\text{max\_share}_u - \text{min\_share}_u) \\ &= \text{broker\_share}.\end{aligned}$$

Equation (2.20) is also satisfied, and the proof is complete.

□

We rewrite Equation (2.22) by splitting it into two, and recognize that the MC has a solution if and only if there exists a subset $\mathcal{U}' \subseteq \mathcal{U}$ for which the following inequalities are satisfied

$$\sum_{u\in\mathcal{U}'} \text{min\_share}_u \leq \text{broker\_share}, \tag{2.23}$$

$$\sum_{u\in\mathcal{U}'} \text{max\_share}_u \geq \text{broker\_share}. \tag{2.24}$$

The problem of finding such a subset is similar to the well-known *Subset-Sum Problem*.

2.5.2 Subset-Sum Problem

We introduce the Subset-Sum Problem (SSP) by giving a classic example. Suppose Bob is going hiking and wants to pack a backpack which will provide him with maximal comfort during the trip. His choice of items are constrained by the total capacity of the backpack. Bob considers the total value of his knapsack to be determined by how well this capacity is exploited. In other words, his goal is to maximize the total weight of his knapsack, preferably to the point where the knapsack weight is exactly equal to its capacity. This is an instance of the SSP, formulated as an optimization problem. The corresponding decision problem is defined as follows.

Definition 2.5 (Subset-Sum Problem (SSP)). *Given a knapsack with capacity $W \in \mathbb{Z}_+$, a finite set of items $\mathcal{N}$, where all items $n \in \mathcal{N}$ have weights $w(n) \in \mathbb{Z}_+$; does there exist a subset of items $\mathcal{N}' \subseteq \mathcal{N}$ such that*

$$\sum_{n\in\mathcal{N}'} w(n) = W? \tag{2.25}$$

⌟

The SSP has been proved to be *NP-complete* (Martello, 1990). This means that there is currently no known polynomial time algorithm for solving the SSP, however it has not been proven that no such algorithm exists. Furthermore, a proposed solution to the SSP can be verified in polynomial time. We will not go into more detail on NP-completeness, and refer the interested reader instead to Chapter 6 in Wolsey (1998) or Chapter 8 in Dasgupta and Vazirani (2006) for an introduction.

2.5.3 Reduction

We are ready to determine the complexity of the MC. In order to prove that it is NP-complete, we need to

1. show that a proposed solution can be verified in polynomial time,
2. perform a polynomial reduction from an NP-complete problem to the MC.

Verifying a proposed solution to the problem is done by checking whether or not the constraints in its definition are satisfied. It is clear that all three constraints in Problem 2.3 can be verified in $\mathcal{O}(|\mathcal{U}|)$ time, where $|\mathcal{U}|$ refers to the number of underwriters. Since the set of underwriters is finite, the complexity of verifying a solution is linear.

For the second step, we need to perform a polynomial reduction from an NP-complete problem to the MC, i.e., to Equations (2.23) and (2.24). When a problem A reduces to a problem B, one of the problems have a solution if and only if the other problem also has a solution. Furthermore, the reduction proves that problem B is at least as hard to solve as problem A.

The similarity between the SSP and the MC becomes more clear if we split Equation (2.25) in two:

$$\sum_{n\in\mathcal{N}'} w(n) \leq W,$$
$$\sum_{n\in\mathcal{N}'} w(n) \geq W.$$

Comparing these equations to Equations (2.23) and (2.24), we see a strong similarity. In order to perform a reduction from the SSP to the MC, we must define the sets and parameters of the MC from an arbitrary SSP input.

We define a bijection $f : \mathcal{N} \mapsto \mathcal{U}$ such that the set of knapsack items corresponds to the set of underwriters. Furthermore, we let the item weights correspond to the

minimal underwriter shares and to the maximal underwriter shares, i.e.,

$$\begin{aligned} \text{min\_share}_{f(n)} &= w(n), \\ \text{max\_share}_{f(n)} &= w(n), \end{aligned}$$

for all $n \in \mathcal{N}$. Finally, the knapsack capacity (W) corresponds to the broker share (broker_share).

For all underwriters u, the reduction results in MC instances where min_share_u is equal to max_share_u, which clearly satisfies the condition of min_share_u being less than or equal to max_share_u. A solution to the the MC exists if and only if there exists a solution to the SSP, and so the reduction from the SSP to the MC is complete. This means that solving the MC is at least as hard as solving the SSP, and so, the MC must also be NP-complete.

Chapter 3

Solution methods

In this chapter we go into detail about the different solution methods used for solving the BP model. We begin by presenting a global optimization method. Even for relatively small problem instances, this method is time consuming. Because of this, we investigate two local optimization methods, which are able to find solutions to larger instances of the BP in reasonable time. Even though these methods do not guarantee optimality, they may produce solutions close to optimality.

We conclude the chapter by presenting solution methods for handling the two secondary objectives described in Section 2.3. Their purpose is to tune the optimal solution, in an attempt to fulfill the secondary objectives to the best possible extent, without worsening the value of the primary objective function.

3.1 Exact algorithm

Two widely used methods for solving integer programs are *cutting plane* algorithms and *branch-and-bound* algorithms (Wolsey, 1998). These are global optimization methods, which use different techniques for searching the solution space for the optimal solution. Both methods solve the relaxation of integer programming problems, in such a way that the solution of the relaxed problem is integer.

Cutting plane algorithms solve the relaxed problem by adding hyperplanes that modify the solution space. This procedure is demonstrated in Figure 3.1. The purpose of these hyperplanes is to cut away parts of the solution space that do not contain feasible solutions of the original integer programming problem, i.e., fractional numbers. We keep adding cutting planes as long as the optimal solution to the modified solution space has fractional coordinates. Eventually, the optimal solution

will have all-integer coordinates, and an optimal solution to the original integer programming problem is found.

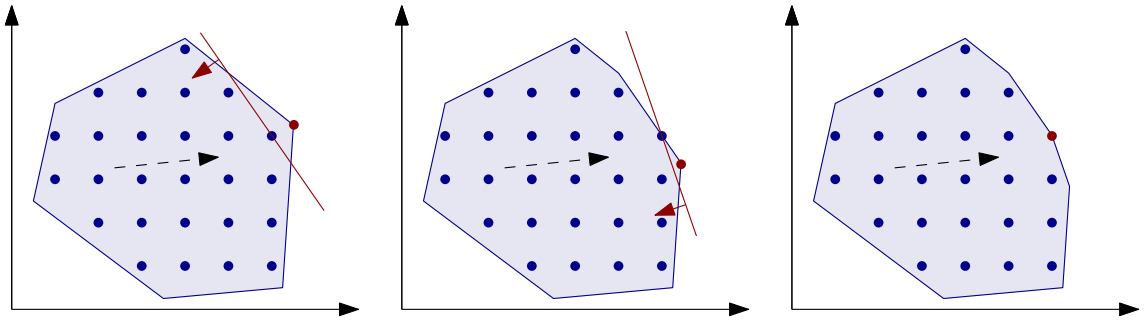


Figure 3.1: The cutting planes (red lines) modify the solution space (blue area) of the relaxed problem, until the optimal solution (red point) is integral.

The branch-and-bound method works by systematically examining sub-areas of the solution space. Analytical upper and lower bounds on the objective function are calculated for each sub-area. The procedure starts by finding the optimal solution of the entire solution space. If this solution contains a fractional coordinate, the solution space is split by hyperplanes corresponding to the floor and ceiling of the fractional coordinate (branching), as illustrated in Figure 3.2. The upper and lower bounds on the objective function are considered for each sub-area. If these bounds indicate that the sub-area may contain a better solution than yet has been found, the sub-area is searched for its optimal solution. Some sub-areas may however be disregarded (bounding), because the corresponding upper and lower bounds imply that no better solution exists in these areas.

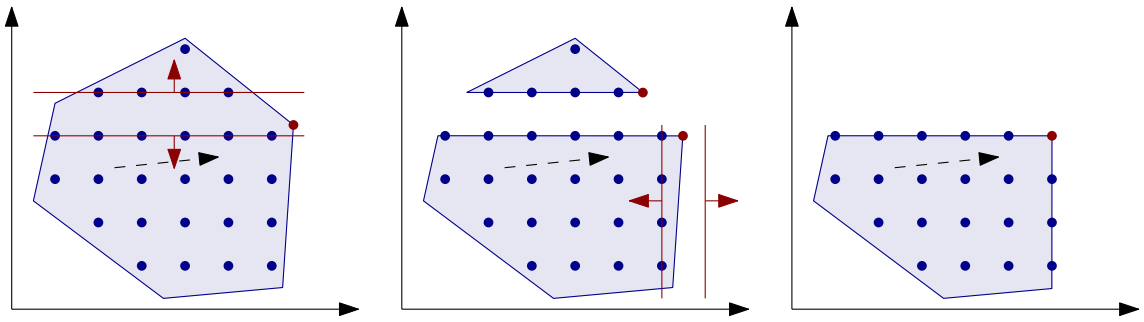


Figure 3.2: The solution space (blue area) is split by hyperplanes (red lines) into sub-areas, by removing fractional coordinates of optimal solutions (red point), and is searched until the optimal integer solution is found.

There exist many implementations of different variations and combinations of these two solution methods. These implementations are referred to as *solvers*. Some well-known high quality solvers in this category are Cplex, Gurobi and Baron. It is

neither common nor recommended to implement solvers from scratch, but rather to use the ones available. Some solvers are free, others are proprietary.

For solving the BP, we have chosen to use the solver *Branch-And-Reduce Optimization Navigator (Baron)*, created by Ryoo and Sahinidis (1996). As is clear from its name, this is a variation of the branch-and-bound method. For each branch, Baron applies a range reduction of the continuous variables, which leads to improved lower and upper bounds on the value of the global optimum—hence the name Branch-and-Reduce.

Implementing the mathematical model in AMPL, we choose Baron as the solver. Given enough time, Baron is guaranteed to find an optimal solution. For solving the BP, this is unfortunately too time consuming. Baron solves the Minimal Working Example (Problem 1.1) instantly, but as the instance size increases the solution time grows rapidly.

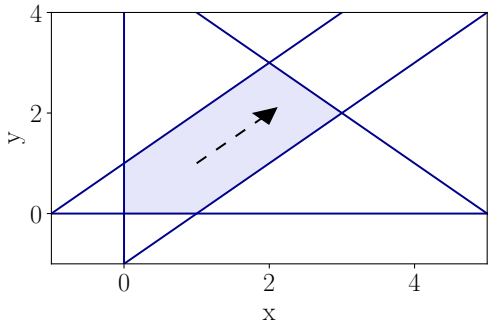
3.2 Linear approximation

Solving linear problems is considerably easier than solving nonlinear problems, and finding good linear approximations can help us solve nonlinear problems. Even though this approach does not guarantee global optimality, it may yield solutions close to optimal. For problems where computational time or power are factors, in addition to solution quality, this approach may be beneficial.

We approximate the BP by using the alternating algorithm (ALT) first constructed by Haverly (1978), and further developed by Audet et al. (2004). ALT is a simple algorithm for solving bilinear problems, by iteratively solving linear subproblems. Consider a bilinear problem, where the variables causing the bilinearity are x and y. In the first iteration, we fix variable x at some initial value. We now have a linear problem in y, which is easily solved. For the next iteration, we fix variable y at the solution value achieved in the previous iteration, and unfix variable x. Again, we have a linear problem. The algorithm keeps on in this manner until there is no change in objective function value, or until the change is below some specified threshold. In order to make the alternating process clear, we use an example to demonstrate how ALT solves a bilinear problem.

Example 3.1 (Iteratively solving linear subproblems). Consider the bilinear problem:

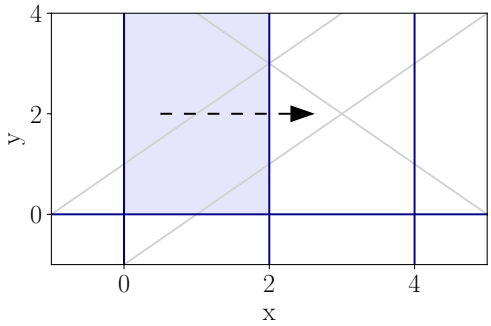
$$
\begin{array}{lrcrcl}
\text{maximize} & x \cdot y & & & & \\
\text{subject to} & x & + & y & \leq & 5 \\
 & x & - & y & \leq & 1 \\
 & -x & + & y & \leq & 1 \\
 & x & , & y & \geq & 0
\end{array}
$$



In the figure, the constraints are represented by blue lines, and the filled area represents the solution space. In order to use ALT to solve this problem, we must decide on some initial value for either x or y. We choose to fix $y = 1$, which means that the algorithm begins by optimizing the variable x. For each iteration, we demonstrate how the objective function and constraints change, along with a visualization of which part of the solution space is being searched. The gray lines show the original optimization problem, but are not "active" in the current iteration.

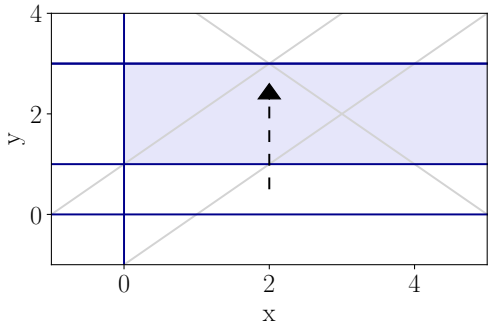
Iteration 1: We fix $y = 1$ and optimize for x. The optimal solution is at $x = 2$, with objective value $x \cdot 1 = 2$.

$$\begin{array}{lrcrcl} \text{maximize} & x & & & & \\ \text{subject to} & x & + & 1 & \leq & 5 \\ & x & - & 1 & \leq & 1 \\ & -x & + & 1 & \leq & 1 \\ & x & & & \geq & 0 \end{array}$$



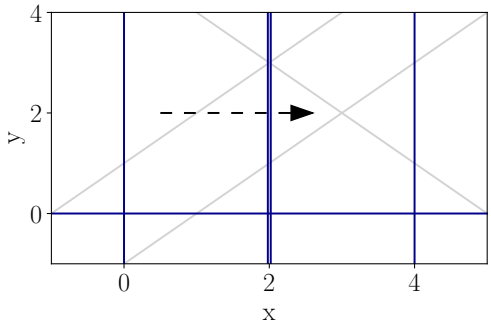
Iteration 2: We fix $x = 2$ and optimize for y. The optimal solution is at $y = 3$, with objective value $2 \cdot y = 6$.

$$\begin{array}{lrcrcl} \text{maximize} & 2y & & & & \\ \text{subject to} & 2 & + & y & \leq & 5 \\ & 2 & - & y & \leq & 1 \\ & -2 & + & y & \leq & 1 \\ & & & y & \geq & 0 \end{array}$$



Iteration 3: We fix $y = 3$ and optimize for x. The optimal solution is at $x = 2$, with objective value $x \cdot 3 = 6$.

$$\begin{array}{lrcrcl} \text{maximize} & 3x & & & & \\ \text{subject to} & x & + & 3 & \leq & 5 \\ & x & - & 3 & \leq & 1 \\ & -x & + & 3 & \leq & 1 \\ & x & & & \geq & 0 \end{array}$$



The objective value is equal to that of the previous iteration, and so the algorithm terminates and provides $(x, y) = (2, 3)$ as the best solution, with objective value $2 \cdot 3 = 6$.

⌟

Note that the solution found by ALT in the example above is not optimal—the optimal solution is at $(x, y) = (2.5, 2.5)$ with objective value $2.5 \cdot 2.5 = 6.25$. This example also shows that the solution is dependent on the initial solution. If we had fixed $x = 1$ in stead of y, the algorithm would have terminated in a different local optimum. Furthermore, if we had fixed any of the variables at zero, the objective function of the first iteration would have vanished. In this case, it is possible that the other variable would also be set equal to zero, there would be no change in the objective function value, and the algorithm would terminate at the objective function value of zero—as far from the optimal solution as possible.

By creating good initial solutions for fixing one of the variables, this need not be an issue. Because the problem is "translated into" linear programs, reaching the final solution is very fast. The algorithm can be run several times from different starting solutions, and it may still be solved considerably faster than by using a global optimization solver. ALT is intuitively easy to understand and just as easy to implement, see Algorithm 1 for the pseudocode.

Algorithm 1: Pseudocode for the alternating algorithm.

Input : Problem instance with bilinear variables x, y and an initial solution.
Output : Solution.

```
// Initialize
optimize_x = true
solution = initial solution
// Alternate
while change in objective function value do
    if optimize_x then
        fix y at solution
        solution = optimize(x)
    else
        fix x at solution
        solution = optimize(y)
    optimize_x = not optimize_x
return solution
```

We need to define a general method for constructing an initial solution, for an arbitrary BP instance. The probability of this initial solution being feasible should be as high as possible. Since the variables $\text{share}_{p,u}$ are dependent on each other, the easiest way to create an initial solution is by fixing the variables $\text{customer\_discount}_{p,u}$ and $\text{broker\_discount}_{p,u}$. We want to maximize the probability of the broker commission being sufficiently large, i.e., the probability of Equation (2.7) being satisfied. To

achieve this, we split the total discount in such a way that the broker receives all the discount, and the customer receives zero discount

$$\begin{aligned} \text{broker\_discount}_{p,u} &= \text{total\_discount}_{p,u}, \\ \text{customer\_discount}_{p,u} &= 0, \end{aligned} \tag{3.1}$$

for all products $p \in \mathcal{P}$ and all underwriters $u \in \mathcal{U}_p$. Unfortunately, this locks all underwriters $u \in \mathcal{U}_p$ as included in the insurance coverage of product $p \in \mathcal{P}$. This is because Equation (2.8) forces the variables $\text{included}_{p,u}$ to be equal to 1 for all underwriters $u \in \mathcal{U}_p$, given the initial solution. In the first iteration of ALT, Equation (2.9) will consequently force the variables $\text{share}_{p,u}$ to be nonzero for all underwriters $u \in \mathcal{U}_p$, which may very well lead to infeasibility because of Equation (2.10). In other words, the $\text{included}_{p,u}$ variables force all underwriters to be included in the insurance coverage, without the possibility of removing them from the coverage at a later point.

To avoid the issue described above, we make a slight modification to ALT. We let Equation (2.8) be active for the iterations optimizing $\text{customer\_discount}_{p,u}$ and $\text{broker\_discount}_{p,u}$, and inactive otherwise. Analogously, we let Equations (2.9) and (2.10) be active for the iterations optimizing $\text{share}_{p,u}$, and inactive otherwise. Removal of included underwriters is now possible, since the $\text{included}_{p,u}$ variables affect the current iteration only. This results in a flexible algorithm, with the construction of an initial solution having a high probability of being feasible.

3.3 Heuristic approach

Heuristic solution methods are expected to work well in practice, without the support of theoretical proofs. These approaches find solutions to problems fast by sacrificing the guarantee of finding an optimal solution. The solution space is searched by use of *heuristics*, which are problem-specific rules of thumb for finding good solutions. The heuristics are combined in algorithms, referred to as *metaheuristics*, which we define as is done by Sörensen and Glover (2013).

Definition 3.2 (Metaheuristics). *A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies for developing heuristic optimization algorithms.*

⌟

Many metaheuristics are inspired by animal behavior or natural processes. Particle swarm optimization (PSO), developed by Kennedy and Eberhart (1995), searches the solution space by simulating the social behavior of the individual organisms in a

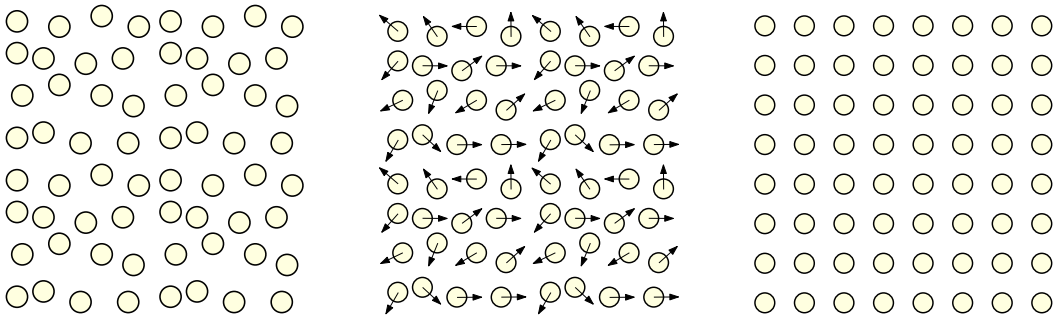
bird flock or fish school. Simulated annealing (SA), described in its original form by Kirkpatrick et al. (1983) and Černý (1985), uses the temperature in connection to the process of annealing solids to determine the intensity of the solution space search. Furthermore, there exist metaheuristics based on ant colony behavior, mutations and heritage of genetics and different types of search strategies.

Erbeyoğlu and Bilge (2016) examined the performance of SA and PSO for solving bilinear problems, and found that both methods perform well on large instances. By its construction PSO is a natural choice for handling continuous variables. On the other hand SA can work well for both continuous and binary variables. Because the BP is a mixed integer problem we solve it using the SA algorithm.

3.3.1 Simulated annealing

Simulated annealing is widely researched and has been applied to a large variety of optimization problems. As the interest for SA grew during the 1980s, van Laarhoven and Aarts (1987) decided to write a review of the theory and applications of SA.

Annealing is a physical process in which a solid with defects is heated to a temperature where the particles in the solid freely rearrange themselves. Slowly cooling the solid down, the particles arrange themselves in a low energy (symmetric) state. We demonstrate the stages of this process in Figure 3.3.



(a) Structure with defects. (b) Heat and movement. (c) Symmetric structure.

Figure 3.3: Annealing process.

Simulated annealing starts at an initial solution, that is either provided or generated. At each iteration a *neighboring solution* is compared to the current solution, and possibly accepted as the new current solution. The set of neighboring solutions is defined to be any solution reachable from the current solution by applying a heuristic. The energy state E of a solution is represented by its objective function, and so the change of energy ΔE between two solutions is defined as the change in the objective function value,

$$\Delta E = \text{new solution objective value} - \text{current solution objetive value}. \tag{3.2}$$

Whenever ΔE is less than zero for a neighboring solution, the algorithm accepts this solution as the new current solution, since it leads to a decrease in energy. If ΔE is larger than or equal to zero, the neighboring solution may still be accepted as the new current solution, even though this leads to an increase in energy. The acceptance of worse solutions corresponds to the free rearrangement of particles in the solid. When the temperature is high, there is more movement in the solid, and the probability of accepting a worse solution should be higher than when the temperature is low—when the solid is crystallizing. To achieve this behavior, the probability of accepting worse solutions is represented by the *Boltzmann probability*

$$\rho = \exp\left(-\frac{\Delta E}{T}\right), \tag{3.3}$$

where T is the current temperature. Accepting worse solutions allows SA to escape local minima, as opposed to a local search which would get stuck as shown in Figure 3.4.

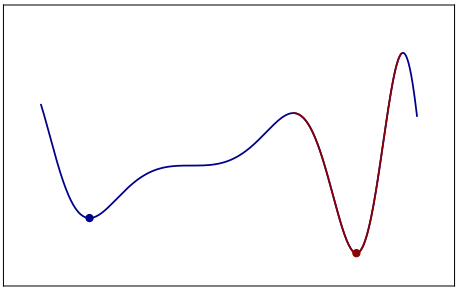


Figure 3.4: Performing a local search starting from anywhere along the blue line will lead to the local optimum (blue point), in stead of the global optimum (red point).

When the temperature is high SA explores the entire solution space, and when the temperature is low it moves towards a local search of the neighborhood at the current solution. Since the set of heuristics determine the neighborhoods of the solutions, it is important to design heuristics that together allow for searching the entire solution space. Heuristics that yield neighboring solutions close to the current solution are said to increase the *intensification* of the search, while heuristics that yield neighbors far away from the current solution are said to increase the *diversification* of the search. Having only intensification heuristics leads to a pure local search, while only diversification heuristics leads to a random search. Balancing the intensification and diversification is important for SA to produce good solutions. The choice of *cooling schedule* for the temperature also greatly influences the quality of the solutions found. Some of the most common cooling schedules are shown in Figure 3.5. The pseudocode for simulated annealing is shown in Algorithm 2.

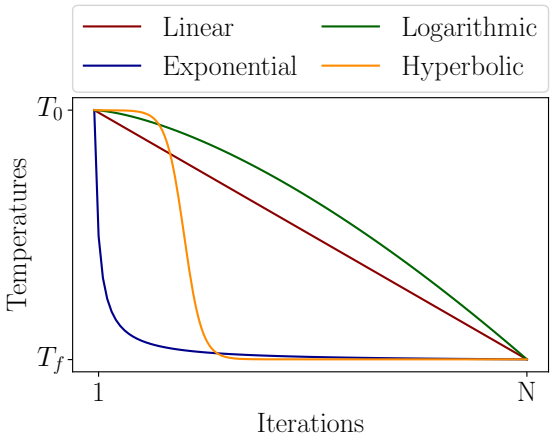


Figure 3.5: Common cooling schedules for simulated annealing.

3.4 Solving the model extensions

In this section we present the solution methods created for handling the secondary objectives of the BP. A solution found by either one of the solution methods described in the previous sections is given as input to these extension solution methods. For the remainder of this chapter, we shall refer to this solution as the *initial solution*. Each of the extension solution methods fix the customer price at the value of the initial solution. In this way the primary objective function value is not affected, but the final solution is tuned in order to fulfill the secondary objectives defined in Section 2.3 to the best possible extent. In other words, the extension solution methods choose which optima with equal primary objective function value to return as the final solution. If there exists only one such optimum, the solution extensions are without effect.

3.4.1 Maximizing the total rating τ

Recall the definition of the total rating τ of an insurance coverage from Equation (2.17). We use a heuristic approach for maximizing the total rating of the initial solution. For 1,000 iterations, we apply a heuristic for swapping included underwriters with low rating, by currently not included underwriters with higher rating. These underwriters are chosen from sets of candidates for removal and addition. We define *feasibility checks* that are used for constructing these candidate sets. The purpose of the checks is to minimize the probability of choosing underwriter combinations that yield infeasible solutions. We present the checks in detail in Section 4.2.2.

Algorithm 2: Pseudocode for simulated annealing.

Input : Problem instance, initial_solution, temperatures T_0, T_f, cooling_schedule.
Output : Solution.

```
// Initialize
solution = initial_solution
best_solution = initial_solution
T = T_0
// Simulate the annealing process
while T > T_f do
    repeat
        Generate new_solution = heuristic(solution)
        Compute change in objective function value ΔE
        if ΔE < 0 then
            // Always accept better solutions
            solution = new_solution
            if new best solution then
                best_solution = new_solution
        else
            // Accept worse solutions with Boltzman probability
            if random(0,1) < exp(-ΔE/T) then
                solution = new_solution
    until max_iterations is reached or new_solution is accepted
    Cool down T according to cooling_schedule
return best_solution
```

Each iteration of the heuristic approach consists of three steps:

1. Choosing a product for performing the swap,
2. Choosing the underwriter to be removed from the product,
3. Searching for an underwriter to replace the removed underwriter.

are constructed in order to minimize the probability of choosing underwriters whose swaps leads to infeasible solutions.

Step 1 is straightforward, as we randomly choose a product for which the swap is to be made. For step 2, we wish to remove an underwriter with low rating from the product. It is not guaranteed that we will be able to replace the removed underwriter,

and so we should not try to remove the underwriter with the lowest rating at each iteration—this may potentially get us nowhere. Because of this, we randomly choose among a set of underwriters with lowest ratings, where the cardinality of the set is variable.

The set cardinality is initially equal to one, so that we start by attempting to remove the underwriter with the worst rating. If no swap can be made for this underwriter, we increase the cardinality by one, and move to the next iteration. We keep increasing the set cardinality by one whenever the number of successive iterations with no successful swap is larger than the set cardinality.

When we have chosen an underwriter to remove, we move to step 3. In this step we sort the set of potential replacement underwriters from highest to lowest rating. For each of these underwriters, in this order, we attempt to swap it with the underwriter chosen to be removed. We then use ALT to search for a feasible solution for this combination of underwriters, with the customer price fixed at the initial solution value. If a feasible solution is found, we keep this as our new current solution, reset the set cardinality to one and the number of successive unsuccessful iterations to zero, before moving to the next iteration. If no feasible solution is found, we move to the next underwriter in the set of potential replacement underwriters. If no replacement is found, we increment the number of successive iterations with no successful swap by one, and move to the next iteration. We present the pseudocode for this solution method in Algorithm 3.

3.4.2 Minimizing the discount split difference δ

Recall the definition of the largest discount split difference δ of an insurance coverage from Equation (2.18). We use a linear program to minimize δ for the initial solution.

In ALT, we alternate between a model optimizing the underwriter share distribution and a model optimizing the discount splits between the customer and the broker. We can use the latter model to solve this model extension. We fix the variables $\text{share}_{p,u}$ to the values of the initial solution for all products $p \in \mathcal{P}$ and all underwriters $u \in \mathcal{U}_p$, and change the objective function to minimize δ. The customer price is fixed at the initial solution value. Optimizing this linear model, the variables $\text{customer\_discount}_{p,u}$ will be assigned values as similar to each other as possible, for all underwriters $u \in \mathcal{U}_p$ within each product $p \in \mathcal{P}$.

Algorithm 3: Pseudocode for maximizing τ.

Input : Problem instance, initial_solution, max_iterations.
Output : Solution.

```
// Initialize
current_solution = initial_solution
successive_fails = 0
size = 1
for i in range max_iterations do
    // Step 1: Randomly choose product p
    p = random(P)
    // Step 2: Choose underwriter u_r to be removed
    u_r = random(removal_candidates, size)
    // Step 3: Attempt to perform swaps with underwriter u_a to be added
    foreach u_a in sorted(addition_candidates) do
        new_solution = current_solution.swap(u_a, u_b)
        result = ALT(new_solution)
        if result is feasible then
            current_solution = result
            size = 1
            successive_fails = 0
            continue
    if no successful swap then
        successive_fails += 1
        if successive_fails > size then
            size += 1
return current_solution
```

Chapter 4

A simulated annealing approach

Even though SA is a problem-independent framework, as stated in Definition 3.2, the choice of values for its many parameters strongly influences the solution quality for different problems. Its performance is also highly dependent on the set of heuristics. Because of this, we devote a chapter to describing all the problem-specific components and choices made in our SA approach to solving the BP.

We begin by describing our choice of initial solution, and proceed by introducing feasibility checks and heuristics used to generate neighboring solutions. Finally we address the choices made for tuning the SA parameters to fit the BP problem.

4.1 Initial solution

As SA iteratively applies heuristics to the current solution, it requires an initial solution. This solution can either be generated inside the SA algorithm, or it can be provided. For the BP it would be unnecessarily complicated to generate an initial solution randomly or greedily, since it would be hard to guarantee feasibility. An easy and fast way to create a feasible solution is to simply use ALT.

Like for SA we still need to provide an initial solution for ALT, but only for half of the variables causing bilinearity. This is a much simpler task than generating an initial solution for SA. We use the procedure for creating an initial solution for ALT described in Section 3.2, and provide the solution found by ALT as the initial solution for SA. We illustrate this architecture in Figure 4.1.

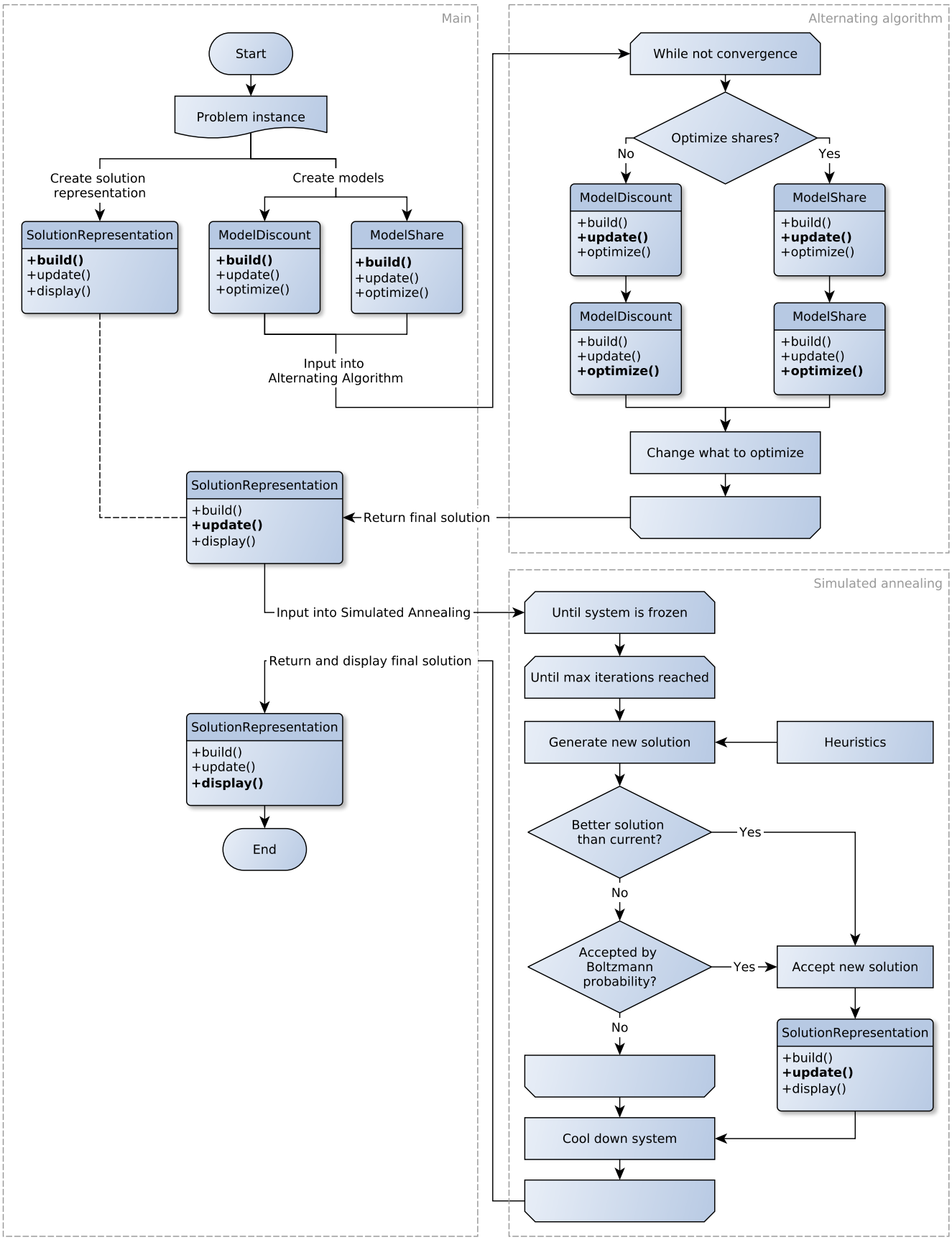


Figure 4.1: SA implementation architecture.

4.2 Neighboring solutions

Neighboring solutions are generated in two steps:

1. We apply a heuristic to the current solution.

   This modifies the combination of included underwriters—the heuristic either adds more underwriters to the solution, removes underwriters from it or swaps underwriters.

   Since the different products need not have the same sets of underwriter candidates, and especially not the same set of included underwriters, the heuristics are applied to one product at a time. Both the heuristic and the product are chosen uniformly at random among the set of heuristics and the set of products, respectively.

2. We use ALT to generate a solution for the modified combination of underwriters. In addition to the new solution, ALT returns a Boolean value indicating whether or not a feasible solution was found.

When we determine the combination of included underwriters using heuristics, the variables $\text{included}_{p,u}$ become redundant, and we consequently remove them from the model. As a result of this, the model constraints containing only these variables vanish. We design *feasibility checks* for replacing these constraints, in addition to decreasing the probability of generating underwriter combinations which have no feasible solutions. The purpose of these checks is not to guarantee feasibility—which is not possible without solving the model itself—it is to decrease the probability of generating infeasible neighboring solutions. We attempt to find a balance between decreasing the probability of infeasibility and the extensiveness of the feasibility checks.

4.2.1 Feasibility checks

The feasibility checks are used when creating sets of candidates for being added to or removed from the current solution, by filtering the set of all underwriters. Below we describe the feasibility checks performed in the heuristics.

Share distribution

We check that there exists a share distribution among the combination of included underwriters. For the product of interest, we compute the sum of minimal shares and the sum of maximal shares, over all the included underwriters. If the broker share is contained in the interval of these two sums (inclusively), there exists at least

one feasible share distribution among the included underwriters. The candidates for removal from the solution are the underwriters that do not cause the broker share to lie outside of this interval if they are removed (analogously for the set of candidates for addition to the solution).

Underwriter preferences
Any underwriter that is in the set $\mathcal{I}_p$ of underwriters that must be included for the specified product, must be included in the insurance coverage. The set of candidates for removal from the product never includes an underwriter which is in this set.

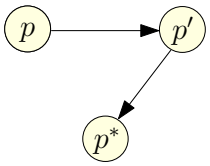
Claims lead candidates
If the product of interest is to contain a claims lead underwriter, the set of claims lead candidates should never be empty. If there is only one claims lead candidate, the set of candidates for removal from the solution never includes this underwriter.

This feasibility check does not guarantee feasibility. If a heuristic removes more than one underwriter, it can remove all the claims lead candidates since they may all be in the set of candidates for removal. We choose to allow for such infeasibilities to occur, in order to keep the balance between the probability of feasibility and extensiveness of the checks.

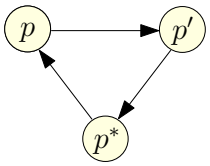
Underwriter inter-product demands
We check that the underwriter inter-product demands are satisfied by representing the demands as directed graphs. For each underwriter u, we create a product dependency graph, where the set of nodes corresponds to the set of all products offered by the underwriter, and the set of arcs represents the inter-product demands. Examples of such dependency graphs are shown in Figure 4.2. An arc (p, p') indicates that underwriter u can be included in product p only if it is also included in product p'. This means that if there exists any path from product p to product p^*, the underwriter can be included in product p only if it is also included in product p^* and all other products corresponding to nodes along the path. We use breadth first search to check these connectivities for the sets of candidates for removal and addition.

4.2.2 Heuristics

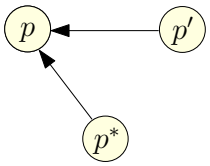
The heuristics take as input the current solution and the product for which the combination of included underwriters is to be modified. For some inputs, the sets of candidates for removal and addition are empty. Together with the neighboring



(a) To be included in product p, the underwriter must be included in both p' and p^*.



(b) The underwriter must either be included in all products p, p' and p^* or in none of them.



(c) If the underwriter is not included in product p, it can not be included in any of the other products.

Figure 4.2

solution, the heuristics return a Boolean value indicating whether or not the heuristic was successful in modifying the combination of underwriters. We present the heuristics below.

Random remove underwriter
An underwriter is chosen uniformly at random among the candidates for removal, and is subsequently removed from the solution. The set of candidates for removal contains every included underwriter that:

1. does not break share feasibility if removed,
2. is not contained in the set of underwriter preferences,
3. is not the only claims lead candidate, and
4. does not break underwriter inter-product demands if removed.

Random add underwriter
An underwriter is chosen uniformly at random among the candidates for addition, and is subsequently added to the solution. The set of candidates for addition contains every underwriter that:

1. is currently not included in the solution
2. does not break share feasibility if added, and
3. does not break underwriter inter-product demands if added.

Random swap underwriters
One underwriter is chosen uniformly at random among the candidates for removal, another underwriter is chosen at random among the candidates for addition, and they are subsequently swapped.

The set of candidates for removal contains every included underwriter that satisfies conditions 2.-4. specified for the heuristic "random remove underwriter." The share feasibility limits are updated by subtracting the corresponding minimal share and maximal share from the sums. The set of candidates for addition contains every underwriter that satisfies all the conditions specified for the heuristic "random add underwriter."

Remove the two underwriters with smallest share, and replace by one
The two underwriters among the candidates for removal with the smallest corresponding shares are replaced by a randomly chosen underwriter among the candidates for addition.

The set of candidates for removal contains every included underwriter that satisfies conditions 2.-4. specified for the heuristic "random remove underwriter." The share feasibility limits are updated by subtracting the corresponding minimal share and maximal share from the sums. The set of candidates for addition contains every underwriter that satisfies all the conditions specified for the heuristic "random add underwriter."

4.3 Parameter tuning

The SA parameters are: the initial and final temperature, the cooling schedule for decreasing the temperature, the total number of cooling iterations (the number of iterations used for cooling the temperature from initial temperature to final temperature), and the number of searching iterations (the maximal number of iterations used to search for a feasible solution at every cooling iteration). Finding good values for these parameters is critical for SA to perform well.

There are several different methods for parameter tuning, ranging from trial-and-error to statistical methods. We tune the initial temperature and final temperature from the behavior of the first α iterations of SA, and we choose the cooling schedule and the number of cooling and searching iterations by trial-and-error.

4.3.1 Tuning the initial and final temperature

We tune the temperatures by examining the changes in energy ΔE between neighboring solutions for the specific instance we are solving. Let N denote the number of cooling iterations, and define $\alpha := \max\{10, 0.2N\}$. For the first α iterations of SA, all neighboring solutions are accepted, regardless of whether they are better or

worse than the current solution. We compute ΔE at every iteration using Equation (3.2).

After α iterations, we calculate the average energy change ΔE_{avg} for these first iterations. We let $T := T_0$ and $\Delta E := \Delta E_{\text{avg}}$, and rewrite the Boltzmann probability Equation (3.3), in order to calculate the initial temperature T_0

$$\rho = \exp\left(-\frac{\Delta E_{\text{avg}}}{T_0}\right) \quad \Leftrightarrow \quad T_0 = \frac{-\Delta E_{\text{avg}}}{\ln(\rho)}.$$

By assigning a value ρ_0 to ρ, i.e., determining the initial probability of accepting a worse solution, we can calculate the value of T_0, and set $T_f = 1$. The initial probability is typically set to a value close to 1 (van Laarhoven and Aarts, 1987), but we find that setting $\rho_0 = 0.5$ works better for the BP.

4.3.2 Choosing the cooling schedule and the number of cooling iterations and searching iterations

The remaining parameters are the number of cooling iterations N, the number of search iterations M, and the cooling schedule for decreasing the temperature from T_0 to T_f. Among the cooling schedules displayed in Figure 3.5, we choose the commonly used exponential cooling schedule (van Laarhoven and Aarts, 1987).

We set the default number of cooling iterations to 500, and the default number of searching iterations to 10 % of the cooling iterations, in order to keep the running time reasonable.

Chapter 5

Computational experiments

In this chapter we present results from the following experiments:

- **Evaluation of solution methods for the BP model**

  We solve the BP model using the solution methods presented in Chapter 3, and compare the solution quality and running time of ALT and SA. We use the global optimization solver Baron to obtain upper and lower bounds on the objective function value.

- **Evaluation of the effect of solving the model extensions**

  We solve the BP model extensions for the feasible test instance solutions. For each extension, we consider the running time, the number of instances for which the corresponding solution is improved, and the extent of solution improvement.

Before presenting the results of these experiments, we specify the tools used for the experimentation and provide information about the test instances. We continue by describing each experiment in more detail, explaining the corresponding results, and finally sharing our observations regarding the experiment results.

5.1 Experimental setup

All experiments were run on a 64 bit Ubuntu 16.04 computer, with a 2.70 GHz speed quad-core i5-6400 processor and 8 GB RAM.

The BP model is implemented in AMPL (version 20181217), as is specified in Section 2.2, using Baron as the solver. The two linearized BP submodels used in ALT, for optimizing share distribution and discount split, respectively, are implemented in

Python (version 3.6.7). These linearizations of the BP model are implemented as specified at the end of Section 3.2. We use the optimization package `or-tools` (version 6.8.5452) created by Google, together with the solver *Coin-or Branch and Cut (CBC)*. This solver is among the free solvers that can be used together with `or-tools`. ALT, SA and the solution methods for the model extensions are also implemented in Python.

5.2 Test instances

The experiments are carried out on a total of 68 test instances: 3 of which are based on real data provided by Edge, and 65 of which are randomly generated. Any real data must undergo a process of complete anonymization with regards to the fleet owners and the underwriters involved. This is a time consuming task. We use instances of varying size in order to challenge the solution methods. For these reasons, most of the instances are randomly generated. Unless specified differently, we use uniform probabilities for random choices. We combine the mathematical notation for the floor function ($\lfloor \cdot \rfloor$) and the ceiling function ($\lceil \cdot \rceil$) to represent rounding a number to the nearest integer, e.g., $\lfloor 0.1 \rceil = 0$ and $\lfloor 0.5 \rceil = 1$.

5.2.1 Instances based on real data

The real data provided by Edge contains information about the number $|\mathcal{P}|$ of products and for each product $p \in \mathcal{P}$, the number $|\mathcal{S}_p|$ of ships, the number $|\mathcal{U}_p|$ of underwriters, and values of the following parameters: $\text{value}_{p,s}$, $\text{rate}_{p,s,u}$, $\text{min\_share}_{p,u}$, $\text{max\_share}_{p,u}$ and $\text{total\_discount}_{p,u}$. This information does not define a complete BP instance, since there are several more sets and parameters that need to be specified. To handle this, we generate the remaining sets and parameters randomly. For each product $p \in \mathcal{P}$:

- there is a 75 % probability that the parameter num_claims_lead_p is equal to 1,
- the parameter broker_share_p is randomly chosen from the interval $[0.7, 1.0]$,
- the parameter min_ratio_p is randomly chosen from the interval $[0.05, 0.10]$,
- the set $\mathcal{C}_p$ is a randomly chosen subset of $\mathcal{U}_p$, with cardinality in the interval $[\lfloor 0.3 \cdot |\mathcal{U}_p| \rceil, \lfloor 0.5 \cdot |\mathcal{U}_p| \rceil]$,
- the set $\mathcal{I}_p$ is a randomly chosen subset of $\mathcal{U}_p$, with cardinality 1, 2 or 3,
- the set $\mathcal{W}_p$ is a randomly chosen subset of $\mathcal{U}_p$, with cardinality equal to $\lfloor 0.1 \cdot |\mathcal{U}_p| \rceil$.

The only remaining parameters are the inter-product demands for underwriters and potential claims leads. Recall that for all products $p, p' \in \mathcal{P}$ and underwriters $u \in \mathcal{U}_p \cap \mathcal{U}_{p'}$, if the value of $\text{u\_demands}_{p,u,p'}$ is equal to 1, the constraint (2.16) becomes active. Analogously for claims lead candidates $u \in \mathcal{C}_p \cap \mathcal{C}_{p'}$, if the value of $\text{cl\_demands}_{p,u,p'}$ is equal to 1, the constraint (2.12) becomes active. We are interested in how the number of such constraints affects the "tightness" of the solution space. For this reason we vary the probability of $\text{u\_demands}_{p,u,p'}$ and $\text{cl\_demands}_{p,u,p'}$ being equal to 1, for all combinations of products and underwriters. We let $\mathcal{D}$ be the set of probabilities, and generate a test instance for each $d \in \mathcal{D}$, i.e., we get $|\mathcal{D}|$ different test instances from each real dataset.

5.2.2 Randomly generated instances

When we create randomly generated test instances, we vary their size by specifying the number $|\mathcal{P}|$ of products and the number $|\mathcal{U}|$ of underwriters. The number of ships is kept constant. For each product we randomly choose a subset of underwriters $\mathcal{U}_p$ with cardinality in the interval $[\lfloor 0.6 \cdot |\mathcal{U}| \rceil, \lfloor 1.0 \cdot |\mathcal{U}| \rceil]$, and a subset of ships $\mathcal{S}_p$ with cardinality in the interval $[\lfloor 0.6 \cdot |\mathcal{S}| \rceil, \lfloor 1.0 \cdot |\mathcal{S}| \rceil]$. Values for the following parameters are randomly generated: $\text{value}_{p,s}$, $\text{rate}_{p,s,u}$, $\text{min\_share}_{p,u}$, $\text{max\_share}_{p,u}$ and $\text{total\_discount}_{p,u}$. The intervals from which these values are generated are based on the parameter values observed in real datasets. The remaining sets and parameters are generated in the same way as described for the instances based on real data.

We create one random test instance for every combination of the following set cardinalities and probabilities of inter-product demands

$$\begin{aligned} |\mathcal{P}| &\in \{1, 3, 10, 15, 50\}, \\ |\mathcal{U}| &\in \{5, 15, 40, 100, 300\}, \\ \mathcal{D} &= \{0.0, 0.10, 0.30\}, \end{aligned}$$

with the exception of when $|\mathcal{P}| = 1$, which is not combined with the different probabilities $d \in \mathcal{D}$. This is because an instance with only one product cannot contain inter-product demands. As a result, we get $(|\mathcal{P}| - 1) \cdot |\mathcal{U}| \cdot |\mathcal{D}| + 1 \cdot |\mathcal{U}| = 65$ randomly generated instances.

The instance names are on the format: $\mathcal{P}_{|\mathcal{P}|}\,\mathcal{U}_{|\mathcal{U}|}\,\mathcal{D}_{d \cdot 100}$. As an example, an instance with 3 products, 15 underwriters and a probability of inter-product demands equal to 0.10, is given the name $\mathcal{P}_3\,\mathcal{U}_{15}\,\mathcal{D}_{10}$. The symbol $*$ is appended to the names of instances based on real data.

5.3 Results

5.3.1 Evaluation of solution methods for the BP model

This experiment concerns the BP primary objective function, in which the customer price is minimized. For each test instance, we compare the running time and the objective function value corresponding to the solutions found by ALT and SA, whose implementations follow the descriptions in Section 3.2 and Chapter 4, respectively. Since the optimal solutions are unknown for all the test instances, we run Baron for 40 minutes in order to obtain upper and lower bounds on the optimal customer price. The lower bounds reported by Baron are theoretically deduced, as described in Section 3.1. The upper bounds are equal to the best objective function values found by Baron.

The results are presented in Table 5.1, along with a visual representation in Figure 5.1. Columns 2 and 4 show the running time in seconds, and columns 3, 5, 6, and 7 show the objective function value in euros. For each test instance, the best objective function value is displayed in bold. Note that the objective function value returned by SA is written in bold only if it belongs to a solution found by SA—it is not written in bold if it belongs to the initial solution provided to SA. If no solution was found, the corresponding cell contains the symbol "-", and if a solution method proves that no solution exists, the corresponding cell contains the text "infeasible". Column 8 shows the relative gap between the best objective function value found by either one of the solution methods and the lower bound reported by Baron. The gap is computed as the difference between the objective function value and the lower bound, divided by the lower bound.

It is important to note that if ALT or SA fails to find a feasible solution, it does not imply infeasibility. Solutions found by these methods should be interpreted as upper bounds on the optimal solution. The lack of a feasible solution simply means that the upper bound is infinitely large. Baron is the only one among the considered solution methods that can determine whether an instance is infeasible.

Table 5.1: Results comparing the running time and objective value of ALT and SA, together with upper and lower bounds provided by Baron.

	ALT		SA		Baron		
Instance	Time	Obj. value	Time	Obj. value	Upper b.	Lower b.	Gap
$\mathcal{P}_1\,\mathcal{U}_5\,\mathcal{D}_0$	0	**547,442**	4	547,442	**547,442**	547,438	0.00
$\mathcal{P}_1\,\mathcal{U}_{15}\,\mathcal{D}_0$	0	**492,820**	262	492,820	**492,820**	492,671	0.00
$\mathcal{P}_1\,\mathcal{U}_{40}\,\mathcal{D}_0$	0	**367,618**	14	367,618	**367,618**	367,475	0.00

(Continued on next page)

Table 5.1 – continued from previous page

	ALT		SA		Baron		
Instance	Time	Obj. value	Time	Obj. value	Upper b.	Lower b.	Gap
$\mathcal{P}_1\mathcal{U}_{100}\mathcal{D}_0$	0	**286,497**	361	286,497	**286,497**	286,146	0.00
$\mathcal{P}_1\mathcal{U}_{300}\mathcal{D}_0$	0	**230,588**	60	230,588	**230,588**	226,351	0.02
$\mathcal{P}_3\mathcal{U}_5\mathcal{D}_0$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_3\mathcal{U}_5\mathcal{D}_{10}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_3\mathcal{U}_5\mathcal{D}_{30}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_3\mathcal{U}_{15}\mathcal{D}_0$	0	**1,256,358**	17	1,256,358	**1,256,358**	1,255,910	0.00
$\mathcal{P}_3\mathcal{U}_{15}\mathcal{D}_{10}$	0	**1,301,902**	21	1,301,902	**1,301,902**	1,294,400	0.01
$\mathcal{P}_3\mathcal{U}_{15}\mathcal{D}_{30}$	0	**1,346,834**	12	1,346,834	**1,346,834**	1,334,470	0.01
$\mathcal{P}_3\mathcal{U}_{15}\mathcal{D}_0*$	0	**445,502**	16	445,502	445,636	437,434	0.02
$\mathcal{P}_3\mathcal{U}_{15}\mathcal{D}_{10}*$	0	**447,688**	14	447,688	448,384	439,707	0.02
$\mathcal{P}_3\mathcal{U}_{15}\mathcal{D}_{30}*$	0	**450,450**	14	450,450	450,626	443,190	0.02
$\mathcal{P}_3\mathcal{U}_{40}\mathcal{D}_0$	0	1,196,272	52	**1,196,266**	**1,196,266**	1,191,980	0.00
$\mathcal{P}_3\mathcal{U}_{40}\mathcal{D}_{10}$	0	1,233,065	36	**1,233,038**	**1,233,038**	1,231,810	0.00
$\mathcal{P}_3\mathcal{U}_{40}\mathcal{D}_{30}$	0	**1,392,726**	39	1,392,726	**1,392,726**	1,392,690	0.00
$\mathcal{P}_3\mathcal{U}_{100}\mathcal{D}_0$	0	**1,039,741**	130	1,039,741	1,040,035	1,030,510	0.01
$\mathcal{P}_3\mathcal{U}_{100}\mathcal{D}_{10}$	0	**1,065,224**	50	1,065,224	1,065,572	1,047,290	0.02
$\mathcal{P}_3\mathcal{U}_{100}\mathcal{D}_{30}$	0	**1,181,373**	1,470	1,181,373	1,185,536	1,176,460	0.00
$\mathcal{P}_3\mathcal{U}_{300}\mathcal{D}_0$	0	**941,814**	135	941,814	**941,814**	923,672	0.02
$\mathcal{P}_3\mathcal{U}_{300}\mathcal{D}_{10}$	0	**972,476**	111	972,476	975,303	949,522	0.02
$\mathcal{P}_3\mathcal{U}_{300}\mathcal{D}_{30}$	1	**997,969**	115	997,969	1,006,664	966,040	0.03
$\mathcal{P}_{10}\mathcal{U}_5\mathcal{D}_0$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{10}\mathcal{U}_5\mathcal{D}_{10}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{10}\mathcal{U}_5\mathcal{D}_{30}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{10}\mathcal{U}_{15}\mathcal{D}_0$	0	**4,441,115**	31	4,441,115	4,447,567	4,406,620	0.01
$\mathcal{P}_{10}\mathcal{U}_{15}\mathcal{D}_{10}$	0	**4,647,035**	49	4,647,035	**4,647,035**	4,631,850	0.00
$\mathcal{P}_{10}\mathcal{U}_{15}\mathcal{D}_{30}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{10}\mathcal{U}_{40}\mathcal{D}_0$	0	**4,319,639**	63	4,319,639	4,355,703	4,236,940	0.02
$\mathcal{P}_{10}\mathcal{U}_{40}\mathcal{D}_{10}$	1	**4,563,599**	57	4,563,599	4,601,240	4,501,560	0.01
$\mathcal{P}_{10}\mathcal{U}_{40}\mathcal{D}_{30}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{10}\mathcal{U}_{100}\mathcal{D}_0$	1	**3,629,311**	192	3,629,311	3,651,154	3,558,160	0.02
$\mathcal{P}_{10}\mathcal{U}_{100}\mathcal{D}_{10}$	1	**4,106,276**	213	4,106,276	4,116,773	4,071,480	0.01
$\mathcal{P}_{10}\mathcal{U}_{100}\mathcal{D}_{30}$	1	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{10}\mathcal{U}_{300}\mathcal{D}_0$	4	**3,923,137**	336	3,923,137	-	1,589,972	1.47
$\mathcal{P}_{10}\mathcal{U}_{300}\mathcal{D}_{10}$	4	**4,385,803**	424	4,385,803	-	2,727,417	0.61
$\mathcal{P}_{10}\mathcal{U}_{300}\mathcal{D}_{30}$	1	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{15}\mathcal{U}_5\mathcal{D}_0$	0	-	-	-	infeasible	infeasible	-

(Continued on next page)

Table 5.1 – continued from previous page

	ALT		SA		Baron		
Instance	Time	Obj. value	Time	Obj. value	Upper b.	Lower b.	Gap
$\mathcal{P}_{15}\mathcal{U}_{5}\mathcal{D}_{10}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{15}\mathcal{U}_{5}\mathcal{D}_{30}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{15}\mathcal{U}_{15}\mathcal{D}_{0}$	0	**6,342,925**	43	6,342,925	6,363,018	6,279,850	0.01
$\mathcal{P}_{15}\mathcal{U}_{15}\mathcal{D}_{10}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{15}\mathcal{U}_{15}\mathcal{D}_{30}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{15}\mathcal{U}_{40}\mathcal{D}_{0}$	1	**6,807,110**	87	6,807,110	6,872,438	6,672,640	0.02
$\mathcal{P}_{15}\mathcal{U}_{40}\mathcal{D}_{10}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{15}\mathcal{U}_{40}\mathcal{D}_{30}$	1	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{15}\mathcal{U}_{100}\mathcal{D}_{0}$	1	5,982,884	197	**5,977,339**	6,016,655	5,836,720	0.02
$\mathcal{P}_{15}\mathcal{U}_{100}\mathcal{D}_{10}$	1	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{15}\mathcal{U}_{100}\mathcal{D}_{30}$	1	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{15}\mathcal{U}_{300}\mathcal{D}_{0}$	6	**5,361,786**	576	5,361,786	-	1,113,329	3.82
$\mathcal{P}_{15}\mathcal{U}_{300}\mathcal{D}_{10}$	6	**6,466,810**	676	6,466,810	6,484,165	6,156,720	0.05
$\mathcal{P}_{15}\mathcal{U}_{300}\mathcal{D}_{30}$	4	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{50}\mathcal{U}_{5}\mathcal{D}_{0}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{50}\mathcal{U}_{5}\mathcal{D}_{10}$	0	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{50}\mathcal{U}_{5}\mathcal{D}_{30}$	1	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{50}\mathcal{U}_{15}\mathcal{D}_{0}$	1	21,160,672	268	**21,160,288**	21,221,801	20,865,000	0.01
$\mathcal{P}_{50}\mathcal{U}_{15}\mathcal{D}_{10}$	2	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{50}\mathcal{U}_{15}\mathcal{D}_{30}$	3	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{50}\mathcal{U}_{40}\mathcal{D}_{0}$	3	**21,078,835**	286	21,078,835	21,373,853	20,691,600	0.02
$\mathcal{P}_{50}\mathcal{U}_{40}\mathcal{D}_{10}$	2	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{50}\mathcal{U}_{40}\mathcal{D}_{30}$	4	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{50}\mathcal{U}_{100}\mathcal{D}_{0}$	6	**19,369,622**	597	19,369,622	-	5,786,498	2.35
$\mathcal{P}_{50}\mathcal{U}_{100}\mathcal{D}_{10}$	9	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{50}\mathcal{U}_{100}\mathcal{D}_{30}$	16	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{50}\mathcal{U}_{300}\mathcal{D}_{0}$	27	**18,645,219**	2,155	18,645,219	-	5,714,325	2.26
$\mathcal{P}_{50}\mathcal{U}_{300}\mathcal{D}_{10}$	71	-	-	-	infeasible	infeasible	-
$\mathcal{P}_{50}\mathcal{U}_{300}\mathcal{D}_{30}$	257	-	-	-	infeasible	infeasible	-

5.3.2 Evaluation of the effect of solving the model extensions

This experiment concerns the BP secondary objectives described in Section 2.3 and their corresponding solution methods described in Section 3.4. For each feasible test instance, we choose the best solution found by either ALT or SA. For this solution,

we apply the maximization of the total rating τ followed by the minimization of the largest discount split difference δ.

The results from this experiment are presented in Table 5.2. Columns 2 and 4 show the running time in seconds, and columns 3 and 5 show the *relative improvement* of solving the model extensions. The relative improvement of a solution method, with respect to an objective function f, an initial solution x_0 and a new solution x_1, is defined as

$$\text{relative improvement} = \frac{f(x_0) - f(x_1)}{f(x_0)}.$$

The relative improvement of maximizing τ is the change in total rating, divided by the total rating of the solution found by ALT or SA. Analogously, the relative improvement of minimizing δ is the change in the largest discount split difference, divided by the largest discount split difference in the solution found by ALT or SA.

Table 5.2: Results presenting the effect of maximizing the total rating τ and minimizing the largest discount difference δ of the feasible test instances.

	Maximizing τ		Minimizing δ	
Instance	Time	Rel. impr.	Time	Rel. impr.
$\mathcal{P}_1\mathcal{U}_5\mathcal{D}_0$	0	0.0	0	0.996
$\mathcal{P}_1\mathcal{U}_{15}\mathcal{D}_0$	36	0.0	0	0.993
$\mathcal{P}_1\mathcal{U}_{40}\mathcal{D}_0$	56	0.0	0	0.995
$\mathcal{P}_1\mathcal{U}_{100}\mathcal{D}_0$	152	0.0	0	0.996
$\mathcal{P}_1\mathcal{U}_{300}\mathcal{D}_0$	126	0.0	0	0.991
$\mathcal{P}_3\mathcal{U}_{15}\mathcal{D}_0$	11	0.0	0	0.996
$\mathcal{P}_3\mathcal{U}_{15}\mathcal{D}_{10}$	13	0.0	0	0.996
$\mathcal{P}_3\mathcal{U}_{15}\mathcal{D}_{30}$	11	0.0	0	0.996
$\mathcal{P}_3\mathcal{U}_{15}\mathcal{D}_0*$	10	0.0	0	0.987
$\mathcal{P}_3\mathcal{U}_{15}\mathcal{D}_{10}*$	9	0.0	0	0.987
$\mathcal{P}_3\mathcal{U}_{15}\mathcal{D}_{30}*$	5	0.0	0	0.987
$\mathcal{P}_3\mathcal{U}_{40}\mathcal{D}_0$	89	0.0	0	0.990
$\mathcal{P}_3\mathcal{U}_{40}\mathcal{D}_{10}$	64	0.0	0	0.994
$\mathcal{P}_3\mathcal{U}_{40}\mathcal{D}_{30}$	27	0.0	0	0.990
$\mathcal{P}_3\mathcal{U}_{100}\mathcal{D}_0$	362	0.0	0	0.996
$\mathcal{P}_3\mathcal{U}_{100}\mathcal{D}_{10}$	256	0.0	0	0.996
$\mathcal{P}_3\mathcal{U}_{100}\mathcal{D}_{30}$	422	0.0	0	0.996
$\mathcal{P}_3\mathcal{U}_{300}\mathcal{D}_0$	778	0.0	0	0.996
$\mathcal{P}_3\mathcal{U}_{300}\mathcal{D}_{10}$	949	0.0	0	0.996
$\mathcal{P}_3\mathcal{U}_{300}\mathcal{D}_{30}$	729	0.0	0	0.991

(Continued on next page)

Table 5.2 – continued from previous page

	Maximizing τ		Minimizing δ	
Instance	Time	Rel. impr.	Time	Rel. impr.
$\mathcal{P}_{10}\mathcal{U}_{15}\mathcal{D}_0$	54	0.0	0	0.990
$\mathcal{P}_{10}\mathcal{U}_{15}\mathcal{D}_{10}$	17	0.0	0	0.990
$\mathcal{P}_{10}\mathcal{U}_{40}\mathcal{D}_0$	287	0.0	0	0.991
$\mathcal{P}_{10}\mathcal{U}_{40}\mathcal{D}_{10}$	124	0.0	0	0.991
$\mathcal{P}_{10}\mathcal{U}_{100}\mathcal{D}_0$	466	0.0	0	0.990
$\mathcal{P}_{10}\mathcal{U}_{100}\mathcal{D}_{10}$	538	0.0	0	0.990
$\mathcal{P}_{10}\mathcal{U}_{300}\mathcal{D}_0$	4,842	0.0	0	0.990
$\mathcal{P}_{10}\mathcal{U}_{300}\mathcal{D}_{10}$	2,314	0.0	0	0.990
$\mathcal{P}_{15}\mathcal{U}_{15}\mathcal{D}_0$	69	0.0	0	0.991
$\mathcal{P}_{15}\mathcal{U}_{40}\mathcal{D}_0$	283	0.0	0	0.992
$\mathcal{P}_{15}\mathcal{U}_{100}\mathcal{D}_0$	877	0.0	0	0.992
$\mathcal{P}_{15}\mathcal{U}_{300}\mathcal{D}_0$	5,756	0.0	0	0.992
$\mathcal{P}_{15}\mathcal{U}_{300}\mathcal{D}_{10}$	4,998	0.0	0	0.990
$\mathcal{P}_{50}\mathcal{U}_{15}\mathcal{D}_0$	197	0.0	0	0.990
$\mathcal{P}_{50}\mathcal{U}_{40}\mathcal{D}_0$	986	0.0	0	0.990
$\mathcal{P}_{50}\mathcal{U}_{100}\mathcal{D}_0$	3,202	0.0	0	0.990
$\mathcal{P}_{50}\mathcal{U}_{300}\mathcal{D}_0$	39,495	0.0	1	0.990

5.4 Observations

We see from the results in Table 5.1 that among the three solution methods, ALT is the method that most frequently finds the best solution. It returns the best objective function value for 33 out of the 37 feasible test instances, 26 of which are *unique solutions*. By a unique solution, we mean that none of the other solution methods are able to find this solution. It is also able to find a solution to all 5 feasible instances for which Baron is unable to find a feasible solution during the 40 minutes of searching:

- $\mathcal{P}_{10}\mathcal{U}_{300}\mathcal{D}_0$,
- $\mathcal{P}_{10}\mathcal{U}_{300}\mathcal{D}_{10}$,
- $\mathcal{P}_{15}\mathcal{U}_{300}\mathcal{D}_0$,
- $\mathcal{P}_{50}\mathcal{U}_{100}\mathcal{D}_0$,
- $\mathcal{P}_{50}\mathcal{U}_{300}\mathcal{D}_0$.

ALT finds these solutions in at most 27 seconds, while Baron searches for 40 minutes before terminating. SA returns the best objective function value for 4 instances, 2 of which are unique, and Baron returns the best objective function value for 13 instances, none of which are unique. Almost all of the solutions found by Baron are also found by ALT.

As seen in Figure 5.1, the differences between the best objective function values found by the three solution methods are modest. To visualize the size of the gaps, we use a line plot even though we are visualizing discrete data. The largest gap between the best objective function values found by ALT and SA is 0.001, for test instance $\mathcal{P}_{15}\mathcal{U}_{100}\mathcal{D}_0$. Between Baron and either ALT or SA the largest gap is 0.014, for test instance $\mathcal{P}_{50}\mathcal{U}_{40}\mathcal{D}_0$.

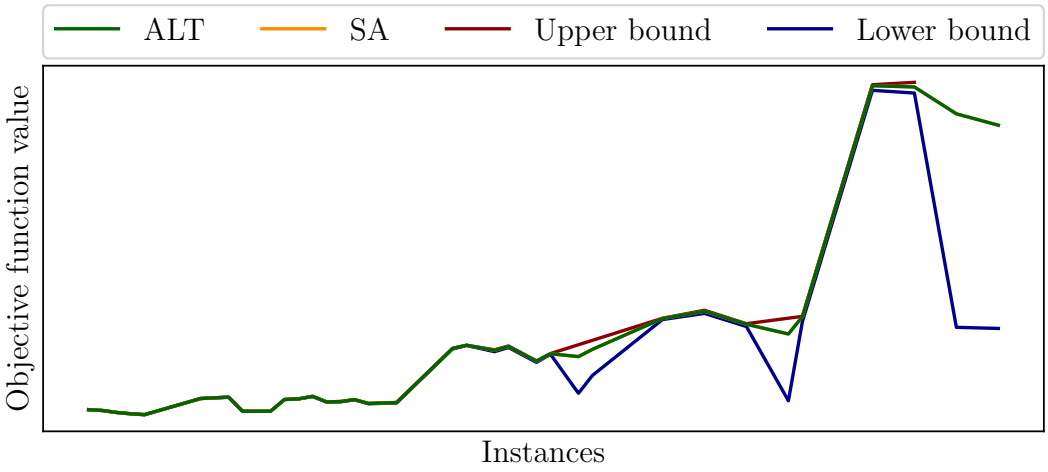


Figure 5.1: The best solutions found by ALT, SA and Baron (upper bound) are all close to the lower bounds. The three visibly large gaps correspond to instances where Baron is unable to find a feasible solution.

In addition to finding the largest amount of best solutions, we see that ALT outperforms both SA and Baron when it comes to running time. The longest running time for ALT is 4 minutes, while SA runs for a maximal of 36 minutes. The median running time of ALT is 0 seconds, while the median running time of SA is 87 seconds. Baron often finds feasible solutions during a preprocessing phase, long before the 40 minutes have elapsed. However, by stopping Baron from searching further when reaching the corresponding running times for ALT, the number of instances for which it is unable to find a solution increases from 5 instances to 28 instances.

Compared to SA and Baron, ALT is a light-weight solution method. It is easy to implement, does not require parameter tuning, is intuitively easy to understand and is very fast. Recall the possible disadvantages of ALT demonstrated in Example 3.1.

The most serious disadvantage is arguably the possibility of bad initial solutions leading to solutions far from optimality. For BP instances it appears that we have found a good initial solution for ALT. The gaps between the lower limit provided by Baron and the best solutions found by ALT are small. They range from 0.00 to 3.82, where the largest gaps are for the instances for which Baron is unable to find a solution. For these instances, the lower bounds are considerably lower than for instances of similar size for which Baron is able to find a feasible solution. Among the instances for which Baron finds feasible solutions, the largest gap between the solutions found by ALT and the lower bounds is 0.05.

We see from Table 5.2 that the maximization of the total rating τ fails to improve any of the feasible instance solutions, even though it often runs for several minutes or up to a couple of hours, and in the worst case for almost 11 hours. Recall that in order to improve the total rating, it is necessary to remove an underwriter from all the products it is included in. Since the heuristic used to maximize τ greedily swaps underwriters for one product at a time, it is not unexpected that the heuristic is unsuccessful more often than not. We would however expect that the total rating could be improved occasionally, especially for the instances with only one product.

Table 5.2 shows that minimizing the largest discount split difference δ yields good results. Not only is it very fast, it also consistently yields an improvement of 99 % for all instances. The fact that a secondary objective is consistently improved to such an extent clearly substantiates our belief that the BP instances typically have several local optima with identical primary objective function values. In order to visualize the effect this model extension has on the solution, we show the details of the best solution to instance $\mathcal{P}_3\,\mathcal{U}_{15}\,\mathcal{D}_{10}*$ before and after the minimization of δ in Table 5.3. We include several decimals to show the precision of the effect. The first three columns contain information about the insurance products, the combination of underwriters included in each product and the share distribution for each product. Column 4 and 5 show the discount split between the customer (c_discount) and the broker (b_discount) corresponding to the optimal solution of the primary objective function. Column 6 and 7 show the discount split between the customer and the broker after minimizing δ.

It is important to note that all the presented observations are connected to randomly generated data. In real data instances, there are correlations between the parameter values and subsets. For the generated instances, the values are chosen at random and independently of each other.

Table 5.3: The effect of minimizing δ displayed for test instance $\mathcal{P}_3\mathcal{U}_{15}\mathcal{C}_{10}*$.

			Before min. δ		After min. δ	
product	underwriters	share	c_discount	b_discount	c_discount	b_discount
HM	uwr_3	0.130	0.1000	0.0000	0.0533	0.0467
	uwr_9	0.100	0.0000	0.1000	0.0533	0.0467
	uwr_{10}	0.100	0.0000	0.1000	0.0533	0.0467
	uwr_{11}	0.050	0.0000	0.1000	0.0533	0.0467
	uwr_{12}	0.050	0.1000	0.0000	0.0533	0.0467
	uwr_{14}	0.200	0.0133	0.0892	0.0546	0.0479
	uwr_{15}	0.250	0.1025	0.0000	0.0546	0.0479
HIFI	uwr_2	0.100	0.0000	0.1000	0.0008	0.0992
	uwr_3	0.150	0.0000	0.1000	0.0008	0.0992
	uwr_5	0.100	0.0000	0.1000	0.0008	0.0992
	uwr_6	0.250	0.0000	0.1025	0.0008	0.1017
	uwr_9	0.100	0.0000	0.1000	0.0008	0.0992
	uwr_{10}	0.100	0.0000	0.1000	0.0008	0.0992
	uwr_{11}	0.050	0.0139	0.0861	0.0008	0.0992
LOH	uwr_1	0.245	0.0000	0.1000	0.0007	0.0993
	uwr_9	0.100	0.0000	0.1000	0.0007	0.0993
	uwr_{10}	0.100	0.0000	0.1000	0.0007	0.0993
	uwr_{13}	0.125	0.0000	0.1000	0.0007	0.0993
	uwr_{14}	0.200	0.0028	0.0997	0.0007	0.1018

Chapter 6

Conclusion and future work

6.1 Conclusion

We have examined the possibility of using optimization as a tool for creating marine insurance coverages at a minimal price. A thorough literature review indicated no documentation of any similar optimization tool in the field of marine insurance brokerage. The software we have developed is valuable for marine insurance brokers, because it gives them a competitive advantage as it is both time saving and unique.

We have defined the problem of creating a marine insurance coverage with minimal price as a mathematical model, and proved that finding a solution is NP-hard. To solve the model for instances of significant size, it was necessary to consider different solution methods. We have proposed an exact solution method, where we use the proprietary solver Baron, and two solution methods that do not guarantee optimality: the simple linearization method ALT, and the more complex metaheuristic approach SA. ALT and SA were adapted to the problem at hand, and implemented from scratch. By experimenting on instances of varying size, we found that ALT performs best out of the three solution methods proposed. ALT found solutions with objective function values close to corresponding theoretical lower bounds. For most of the test instances these solutions were found in less than one second. In at most 27 seconds, ALT found feasible solutions to large test instances for which Baron was unable to find a feasible solution during a 40 minute search.

We expected that the problem instances typically contained several local optima with equal objective function values. Because of this, we defined two model extensions with the purpose of affecting the "choice" of such local optima with equal primary objective function value (customer price). The first extension aims to maximize the total rating of an insurance coverage and the second extension aims to minimize

the largest discount split difference. We proposed a heuristic solution method for maximizing the total rating, and a linear program for minimizing the largest discount split difference. The linear program was highly effective for choosing local optima, while the heuristic solution method did not affect the solution for any of the test instances. We can not conclude whether the heuristic solution method was unsuccessful, or whether the test instances did not contain several local optima with equal primary objective function value and different total rating.

In the end, we were able to create an effective tool for the marine insurance brokers. Solving the model using ALT followed by a minimization of the total discount split provides good solutions, with the desirable property of an even total discount split, in very little time.

6.2 Future work

There are especially two areas for which further exploration would be valuable.

Extending the computational experiments to include more real-life instances. Even though the solution methods found good solutions to the test instances, all but three of the instances were generated randomly. We do not have any reason to believe that the solution methods should produce worse results on real instances. They perform well on a large variety of instance sizes, and the results regarding the instances based on real data do not stand out from the others. Even so, we encourage experimentation on more instances from real data, in order to verify what we have seen in randomly generated instances.

Other solution methods could be considered for maximizing the total rating. One possibility is to design a new heuristic, which considers both the priority rating of the underwriters and the number of products from which the underwriters must be removed in order to improve the total rating. Another possibility could be to define an objective function which favors high priority rating, and solve the model with the customer price fixed at the current solution.

Bibliography

Audet, C., Brimberg, J., Hansen, P., Le Digabel, S., and Mladenović, N. (2004). Pooling Problem: Alternate Formulations and Solution Methods. *Management Science*, 50(6):761–776.

Borch, K. (1961). The Utility Concept Applied to the Theory of Insurance. *ASTIN Bulletin*, 1(05):245–255.

Borch, K. (1962). Equilibrium in a Reinsurance Market. *Econometrica*, 30(3):424–444.

Borch, K. H., Sandmo, A., and Aase, K. K. (1990). *Economics of Insurance*. Elsevier.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51.

Dasgupta, S. and Vazirani, U. (2006). *Algorithms*. McGraw-Hill Education, Boston, 1 edition edition.

Erbeyoğlu, G. and Bilge, Ü. (2016). PSO-based and SA-based metaheuristics for bilinear programming problems: an application to the pooling problem. *Journal of Heuristics*, 22(2):147–179.

Haverly, C. A. (1978). Studies of the Behavior of Recursion for the Pooling Problem. *SIGMAP Bull.*, (25):19–28.

Kennedy, J. and Eberhart, R. (1995). Particle Swarm Optimization. *Proceedings of the IEEE international conference on neural networks*, 4:1942–1948.

Kihlstrom, R. E. and Roth, A. E. (1982). Risk Aversion and the Negotiation of Insurance Contracts. *The Journal of Risk and Insurance*, 49(3):372–387.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598):671–680.

Martello, S. (1990). Knapsack Problems: Algorithms and Computer Implementations. *Wiley-Interscience series in discrete mathematics and optimization*.

Nash, J. F. (1950). The Bargaining Problem. *Econometrica*, 18(2):155–162.

Raviv, A. (1979). The Design of an Optimal Insurance Policy. *The American Economic Review*, 69(1):84–96.

Ryoo, H. S. and Sahinidis, N. V. (1996). A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138.

Smith, V. L. (1968). Optimal Insurance Coverage. *University of Chicago Press*, 76:68–77.

Sörensen, K. and Glover, F. W. (2013). Metaheuristics. In Gass, S. I. and Fu, M. C., editors, *Encyclopedia of Operations Research and Management Science*, pages 960–970. Springer US, Boston, MA.

Szpiro, G. G. (1985). Optimal Insurance Coverage. *The Journal of Risk and Insurance*, 52(4):704–710.

van Laarhoven, P. J. M. and Aarts, E. H. L. (1987). *Simulated Annealing: Theory and Applications*. Mathematics and Its Applications. Springer Netherlands, Dordrecht.

Wolsey, L. A. (1998). *Integer Programming*. Wiley-Interscience, New York, 1 edition edition.