TECHNISCHE UNIVERSITÄT WIEN

DIPLOMARBEIT

Hedging under integer constraints

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Finanz- und Versicherungsmathematik

eingereicht von

Robert Bajons

Matr.Nr.: 1226422

ausgeführt am
Institut für Stochastik und Wirtschaftsmathematik
der Fakultät für Mathematik und Geoinformation
der Technischen Universität Wien

Betreuung durch
Assoc. Prof. Dipl.-Ing. Dr. techn. Stefan Gerhold

Wien, am 20.08.2020

(Unterschrift Verfasser) (Unterschrift Betreuer)

Abstract

We study the problem of hedging claims under a minimum variance criterion and with respect to integer constraints. The model we use is a discrete time model where the price states evolve as multinomial trees. The optimal hedging strategy is then derived by two different approaches; first a Branch and Bound type algorithm is employed to obtain an integer solution. In a second approach we rewrite the problem as a closest vector problem and use this algorithm from lattice theory to compute an optimal strategy. We will present the solutions for the classical binomial model and also for the trinomial model, which are the most well known discrete time tree models. The outcome of these approaches are compared with the naive idea of simply rounding the strategy to next integer to see if it is possible to improve the solution. Finally we analyze the integer strategy in the binomial model in more detail, where we inspect the limiting behaviour of the strategy numerically.

Kurzfassung

Diese Arbeit befasst sich mit dem Problem des Hedges von verschiedenen Claims nach dem Minimum-Varianz-Prinzip bei der zugleich eingeführten Einschränkung auf die optimale Strategie nur aus ganzzahligen Positionen zu bestehen (integer constraint). Wir betrachen ein zeitdiskretes Modell, in dem sich die Preise nach einem multinomialen Baum entwickeln. Die optimale Integer-Strategie wird hierbei mit zwei verschiedenen Ansätzen erörtert; zunächst einem Algorithmus vom Branch and Bound Typ, einer Klasse von Algorithmen aus dem Bereich des integer programming. In einem zweiten Ansatz wird das Problem auf ein Closest-Vector-Problem zurückgeführt und dieses wird mit aus der Gittertheorie bekannten Algortihmen gelöst. Wir betrachen diese Ansätze für praktisch relevante und bekannte Modelle, das Binomialmodell und das Trinomialmodell, sowie Erweiterungen dieser beiden Modelle. Wir vergleichen hierbei die Leistungen und Resultate der zwei vorgestellten Algortihmen mit der naiven Herangehensweise, dem Runden der klassischen Strategie. Zu guter Letzt analysieren wir die optimale Integer-Hedging-Strategie im Binomialmodell etwas genauer und betrachten hierbei ob sich im Grenzfall eine Konvergenz zur klassischen Strategie beobachten lässt.

Danksagung

An dieser Stelle möchte ich mich gerne bei allen bedanken, die mich während des Studiums persönlich, sowie fachlich unterstützt haben. Ein großes Dankeschön geht hierbei an meinen Diplomarbeit-Betreuer Assoc. Prof. Dipl.-Ing. Dr. techn. Stefan Gerhold, der mich zu dem Thema geführt hat und mich immer unterstützt hat.

Des Weiteren möchte ich mich bei meine Eltern bedanken, die mir dieses Studium ermöglicht haben und mich in auf so viele Arten und Weisen unterstützt haben. Auch meinen beiden Geschwistern gebührt herzlicher Dank, vor allem meiner Schwester, die mich ein Leben lang aushalten musste.

Darüber hinaus möchte ich mich bei all meinen Freunden bedanken, die mich immer wieder von den Anstrengungen des Studiums ablenken und mir in vielen Belange helfen. Ein besonderer Dank geht an Markus und Philip.

Am allermeisten aber danke ich meiner Verlobten, Hanna. Ohne dich wäre ich jetzt nicht hier und hätte sicherlich nicht all das geschafft. Deine Liebe und dein Rückhalt hat mir immer wieder geholfen, auch wenn ich geglaubt habe, dass alles zu viel wird. Du hast mich dazu gebracht immer das Beste aus mir herauszuholen und hast mir jeden Tag aufs Neue Motivation gegeben. Te amo mi Huevilitas!

Contents

Chapter 1

Introduction

The work of Gerhold and Krühner [17] tackles an interesting problem when it comes to classical pricing and hedging theory. In the general no arbitrage sense, there are a lot of simplifying assumptions on the financial market. The one particularly looked in is the one that position sizes may be arbitrary real numbers, which is an often violated assumption in practice. Real-life trading follows various rules, and it is often only possible to trade an integer amount of some assets. Even if there are no such restrictions it can be more reasonable to buy only integer amounts of assets due to fees or poor liquidity. Especially for small inverstors this is often useful. But also for big players it is often practice to buy round lots consiting of several hundreds of shares.

In this work we will follow the idea of Gerhold and Krühner [17]. They impose a framework for integer contraints and also present results which adapt the findings of the classical no arbitrage theory to this framework. Furthermore they look at a toy example for hedging under integer constraints, which only takes into account one timestep.

The main goal of this thesis is to extend the idea of the above mentioned toy example and develop and solve a multi-time period version of the integer hedging problem, which in this context has not been done yet. Thus, the structure of the thesis is as follows: in chapter 2 we will establish the framework in which we operate. We will look at the problem of minimizing the hedging error when imposing integer contraints. Therefore we will revise this so called minimum variance hedging (or variance oprimal hedging, cf. Föllmer and Schied [16]) theory and also state some useful results. In chapter 3 we will look at various approaches for solving the problem developed in the preceding chapter. As we will see we face a classical integer programming problem. One approach to solve this is the Branch and Bound (BB) algorithm. Another idea to look at the problem, that is also considered in the Gerhold and Krühner [17] paper for the one step problem, is to rewrite it as an instance of the closest vector problem and to solve the problem accordingly. We will revise the theory behind these

integer programming approaches as well as some special characteristics regarding our specific framework.

Chapter 4 focuses on the application side of the integer hedging problem. Here we study common multi-period models, like the binomial model (or often called CRR model, after Cox, Ross and Rubinstein [12]) and the trinomial model. Furthermore we will look at two other models which account for more than one risky asset. The basis for them are the trinomial and binomial model. We will analyse how the algortihm developed behave in each part of the application and where they can be useful to implement.

Finally in chapter 5 we will look at the integer hedging problem in the binomial model in more detail. The goal here is to analyze the limiting behaviour of the integer constraints strategy in a sequence of binomial model. We will see that numerically, there is a converging behaviour of the integer strategy to the classical strategy as the number of timsteps of the model increases. This is an interesting point as it suggests that we can achieve a completeness in the incomplete integer constraints model.

Chapter 2

The integer constraints problem

In this chapter we want to establish a framework which will serve as a basis for the whole thesis. To do so we will first impose general assumptions and recall the usual notations and definitions necessary for our setting. Furthermore we will revise some important results on variance optimal hedging, which we mostly take from Föllmer and Schied [16]. As for the notations we will mainly adopt the one from Gerhold and Krühner [17], from which we also take some useful result.
After these preparation we will then present the general model for multi-period time hedging under integer constraints in section 2.3.

2.1 General settings

The foundation of our analysis is the filtered probability space $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$, with a filtration $\mathbb{F} = (\mathcal{F}_t)_{t\in\mathbb{T}}$, $\mathbb{T} = \{0, \dots, T\}$, $T \in \mathbb{N}$, for which it holds that $\mathcal{F}_T = \mathcal{F}$ and $\mathcal{F}_0 = \{\emptyset, \Omega\}$. Furthermore we also fix the following assumption:

Assumption 2.1.1. *Ω is finite, $\mathcal{F}$ is the powerset of Ω, $P[\{\omega\}] > 0$ for any $\omega \in \Omega$, and we choose an enumeration $\omega_1, \dots, \omega_n$ of its elements.*

We consider a frictionless market model with d risky assets and denote their price process by $S_t = (S_t^1, \dots, S_t^d)$, $t \in \mathbb{T}$. Furthermore the market contains a riskless asset, whose price is given as $S_t^0 = (1 + r)^t$, $t \in \mathbb{T}$, and we denote the market price process by $\tilde{S} = (S^0, S)$. Furthermore in general we assume for the risk-free rate r that $r > -1$ and for the risky asset it holds that $S_t \geq 0$ and S_t is $\mathcal{F}_t$-measurable, $\forall t$.

As usual we consider a trading strategy $\tilde{\phi}_t = (\phi_t^0, \dots, \phi_t^d) = (\phi_t^0, \phi_t) \in \mathbb{R}^{d+1}$, $t \in \mathbb{T}\backslash\{0\}$, as a predictable and square integrable process, where ϕ_t^i denotes the number of shares of asset

i held in time $(t-1,t]$. Furthermore the value process of the portfolio denoted by $V_t(\tilde{\phi})$ is given as:

$$V_t(\tilde{\phi}) = \tilde{\phi}_t \cdot S_t = \sum_{i=0}^{d} \phi_t^i S_t^i \tag{2.1}$$

We will only consider selffinacing trading strategies, meaning that:

$$\sum_{i=0}^{d} \phi_t^i S_t^i = \sum_{i=0}^{d} \phi_{t+1}^i S_t^i \tag{2.2}$$

Often it is much more convenient to work with the discounted processes:

$$\hat{S}_t = \frac{\tilde{S}_t}{S_t^0}, \text{ and } \hat{V}_t(\tilde{\phi}) = \frac{V_t(\tilde{\phi})}{S_t^0} \tag{2.3}$$

A simple yet very useful result is given by the following proposition:

Proposition 2.1.1. *Let $\tilde{\phi}$ be a trading strategy, then*

$$\tilde{\phi} \textit{ is selffinancing} \Leftrightarrow \hat{V}_t(\tilde{\phi}) = V_0(\tilde{\phi}) + \sum_{k=1}^{t} \tilde{\phi}_k \cdot \Delta\hat{S}_k, \textit{ where } \Delta\hat{S}_k = \hat{S}_k - \hat{S}_{k-1} \tag{2.4}$$

Proof.

$$\begin{aligned}
\tilde{\phi} \text{ is selffinancing} &\Leftrightarrow \sum_{i=0}^{d} \phi_t^i S_t^i = \sum_{i=0}^{d} \phi_{t+1}^i S_t^i \Leftrightarrow \tilde{\phi}_t \cdot S_t = \tilde{\phi}_{t+1} \cdot S_t \\
&\Leftrightarrow \hat{V}_t(\tilde{\phi}) = \tilde{\phi}_t \cdot \hat{S}_t = V_0(\tilde{\phi}) + \sum_{k=1}^{t} (\tilde{\phi}_k \cdot \hat{S}_k - \underbrace{\tilde{\phi}_{k-1} \cdot \hat{S}_{k-1}}_{\overset{\phi \text{ sf}}{=} \tilde{\phi}_t \cdot \hat{S}_{t-1}}) \\
&\Leftrightarrow \hat{V}_t(\tilde{\phi}) = V_0(\tilde{\phi}) + \sum_{k=1}^{t} \tilde{\phi}_k \cdot \Delta\hat{S}_k
\end{aligned}$$

□

Remark. Note that since $\hat{S} = (1, \hat{S}^1, \ldots, \hat{S}^d)$ it follows that $\Delta\hat{S}^0 = 0$ and therefore this part plays no role in the equation (2.4). Therefore it is sufficient to only define $\Delta\hat{S}$ for the risky assets and rewrite the (2.4) as $\hat{V}_t(\tilde{\phi}) = V_0 + \sum_{k=1}^{t} \phi_k \cdot \Delta\hat{S}_k$.

As the main focus of this thesis is interger hedging strategies we also want to adjust our strategies to that:

Definition 2.1.1. An *integer strategy* is a predictable process $(\tilde{\varphi}_t)_{t\in\mathbb{T}\setminus\{0\}}$ that takes values in $\mathbb{R}\times\mathbb{Z}^d$. We will always assume that $\tilde{\varphi}$ is selffinancing, so (2.2) holds.

Our definition of an integer strategy assumes, that the amount of shares in the riskless asset can be any real number, which goes in line with the idea that it is usually possible to hold any kind of amount in the bank account, or at least a value that is not necessarily an integer amount or any kind of round lot. Furthermore defining the holdings in the bank account like that facilitates the computation as it will help to find a selffinaning strategy via the formula given of the value process given in equation (2.4).

Finally we recall one of the main concepts studied within the course of financial mathematics, namely arbitrage. Intuitively an arbitrage opportunity is a portfolio, which yields positive profit without any kind of downside risk. This leads to the following definition:

Definition 2.1.2. A selffinacing trading strategy is called an *arbitrage opportunity* if the value process for this strategy satisfies the following conditions:

$$V_0(\tilde{\phi}) \leq 0, \quad V_T(\tilde{\phi}) \geq 0 \quad \text{and } P(V_T(\tilde{\phi}) > 0) > 0 \tag{2.5}$$

Of course this concept can be easily transferred to our integer case, i.e. an integer strategy φ is an arbitrage opportunity if it fullfills definition 2.1.2.

2.2 Hedging in incomplete markets

After having established the settings and notations of the market model we quickly want to revise the idea of hedging. To do so we recall that a claim C is an $\mathcal{F}_T$-measurable random variable with $C \geq 0$. We will often write H for the discounted claim and use this notation instead of C.

The main definition of hedging is to find a replicating strategy $\tilde{\phi}$, such that $\hat{V}_T(\tilde{\phi}) = H$. The idea of hedging is closely related to the one of determining the fair price of a claim. Thus it is a highly studied problem in the financial field.

Finding a perfect hedging strategy is only possible in a complete market model, i.e. a market model where any claim can be attained in a way as described before. In reality however, there are a lot of frictions in the market which prevents it from being complete. Thus it is not possible to replicate claims. One example for such a market with frictions is our integer hedging model. Through imposing integer constraints our market model is incomplete, thus we need to find strategies to cope with this incompleteness. In this section we will look at ideas to treat this issue.

There are various different approaches to cope with hedging in incomplete markets, such as finding superreplication strategies or using a quadratic criterion. In this thesis we will look at this most common idea of finding a strategy which minimizes the hedging error.

2.2.1 Minimum variance hedging

For this section we consider a model which satisfies the standard no abritrage condition. We consider a martingale measure P and a claim $H \in \mathcal{L}^2(P)$, so H is a square integrable discounted claim. Furthermore the discounted price process $\hat{S}$ is also square integrable w.r.t. P for all $t \in \mathbb{T}$. The goal of minimal variance hedging is then given by minimizing the hedging error under the $L^2(P)$-norm, i.e. minimizing:

$$\|(H - \hat{V}_T)\|_2^2 = \mathbb{E}[(H - \hat{V}_T)^2] = \mathbb{E}[(H - V_0 + \sum_{t=0}^{T} \phi_t \cdot \Delta\hat{S}_t)^2] \tag{2.6}$$

Thus a minimum variance strategy, or variance optimal strategy can be characterized in the following way:

Definition 2.2.1. A *variance optimal strategy* for a (discounted) claim H is a pair (V_0^*, ϕ^*) such that

$$\mathbb{E}[(H - V_0^* - \sum_{t=0}^{T} \phi_t^* \cdot \Delta\hat{S}_t)^2] \leq \mathbb{E}[(H - V_0 - \sum_{t=0}^{T} \phi_t \cdot \Delta\hat{S}_t)^2] \tag{2.7}$$

holds, for any initial capital $V_0 \in \mathbb{R}$ and any admissible trading strategy ϕ.

Remark. It can be shown that such an optimal strategy always exists for a fixed martingale measure P. Under this martingale measure the minimum variance strategy is equivalent to the so called *locally risk minimizing strategy*, whose existence can be shown by the discrete time version of the *Kunita Watanabe decomposition*. Furthermore to derive such a strategy a recursive scheme can be applied where one determines V_t and ϕ_{t+1} by regressing V_{t+1} on $\Delta\hat{S}$. Since we are not particularily interested in the classical minimum variance case, but only want to apply the general idea for our integer hedging problem, the interested reader is referred to Föllmer and Schied [16, chap. 10] for more details.

2.3 The multi period model

The model we primarily will work with is in general a multinomial lattice model. The idea of this is basically derived by generalizing the binomial model of Cox et al. [12]. Lattice

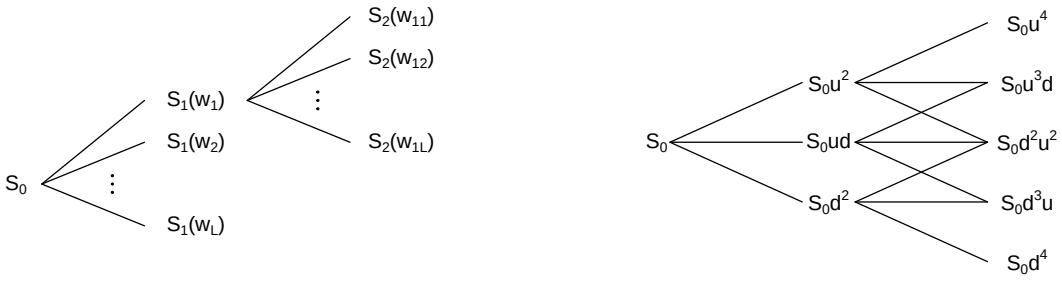

Figure 2.1: Evolution of a multinomial tree: *left:* general case; *right:* the trinomial model for the special case of equation 2.8.

models have been studied a lot in an financial context and have various nice properties. They are easily applicable and are thus often used in real application. Besides that they allow for valuation of all kinds of options and are also able to account for other problems that often arrive in a financial context such as the volatility smiles or heavy tails (see for example Yamada and Primbs [32]).

In our model, the price of each risky asset at each trading time $t \in \mathbb{T}\backslash\{0\}$ evolves randomly in $L \in \mathbb{N}$ different directions, thus building a lattice that evolves with each time step as seen in figure 2.1.

For our analysis it often will be useful to look at the less general model of Yamada and Primbs [32], where the price process can take the L possible future values in the following way:

$$S_{t+1} = u^{L-l} d^{l-1} S_t \text{ , where } l = 1, \dots, L \tag{2.8}$$

Here $u > d > 0$ usually relate to an up and down movement of the price. It is easy to see that the Binomial model is a special case where $L = 2$. Another special case of the above formulation is the trinomial model, where in each step the price has three possibilites $\{\bar{u}, \bar{m}, \bar{d}\}$ of moving. This model corresponds to the case of $L = 3$, $\bar{u} = u^2$, $\bar{m} = ud$ and $\bar{d} = d^2$.

The primarily goal will be to hedge a (discounted) Claim H according to the minimum variance criterion introduced in section 2.2.1, where the integer constraints are imposed on

the hedging strategy.

Therefore lets first look at said strategy φ: from our definition we know that it should be a predictable process. In each multinomial step the price of each risky asset S^i, $i \in \{1, \dots, d\}$, jumps according to L different points, thus in each timestep $t \in \mathbb{T}\backslash\{0\}$, φ_t^i takes $L^{(t-1)}$ values. Finally each of these φ_t^i should be an integer and so $\varphi_t \in \mathbb{Z}^{dL^{t-1}}$, or if we put it together, we are searching for $\varphi^i = (\varphi_1^i, \varphi_2^i, \dots, \varphi_T^i) \in \mathbb{Z}^{1+L^2+\dots+L^{(T-1)}} = \mathbb{Z}^{(\frac{L^T-1}{L-1})}$.

If we look at the less general case of equation 2.8, the space, in which the φ^i lives, changes, so the number of components to estimate decrease. This is due to the fact that in each step the price process evolves in a way, such that the price states jump only according to u and d. Thus each price of the process at time t, has a form of $S_0 u^N d^{M-N}$ where N is the number of ups and M is the maximum number of ups. This maximum number is clearly given as $M = t(L-1)$, since in each time step the highest upward jump is given as $u^{(L-1)}$. Therefore the number of different prices in each time step is given by $t(L-1)+1$. For each time component φ_t^i of the strategy, this means that $\varphi_t^i \in \mathbb{Z}^{(t-1)(L-1)+1}$. The number of values for the full trading strategy φ^i can thus be found by summing up the possibilities up to time $T-1$. Doing this shows that the strategy we are searching for is in $\mathbb{Z}^{(L-1)T(T-1)/2+T}$.
In the following we will write $\varphi^i \in \tilde{\mathbb{Z}}$ and refer to this as being either $\mathbb{Z}^{(\frac{L^T-1}{L-1})}$ or being $\mathbb{Z}^{(L-1)T(T-1)/2+T}$.

With the notation and derivation of the previous paragraph we can now state the key problem that we will try to solve throughout this thesis:

$$\inf_{\varphi \in \tilde{\mathbb{Z}}} ||H - V_T||_2^2 = \inf_{\varphi \in \tilde{\mathbb{Z}}} \mathbb{E}[(H - \hat{V}_T)^2] \tag{2.9}$$

Considering the L different possible ways the price can develop in each step. We can rewrite the general problem from equation 2.9 in 2 different ways:

1) rewrite as one sum:

$$\inf_{\varphi \in \tilde{\mathbb{Z}}} ||H - V_T||_2^2 = \inf_{\varphi \in \tilde{\mathbb{Z}}} \sum_{j=1}^{L^T} P(\omega_j)\big(\tilde{H}(\omega_j) - \sum_{t=1}^{T} \varphi_t(\omega_j) \cdot \Delta \hat{S}_t(\omega_j)\big)^2 \tag{2.10}$$

2) or we can rewrite it as multiple sums (cp. with the notation from figure 2.1):

$$\begin{aligned} &\inf_{\varphi \in \tilde{\mathbb{Z}}} ||H - V_T||_2^2 = \\ &\inf_{\varphi \in \tilde{\mathbb{Z}}} \sum_{j_1=1}^{L} \cdots \sum_{j_n=1}^{L} P(\omega_{j_1,\dots,j_n})\big(\tilde{H}(\omega_{j_1,\dots,j_n}) - V_0 - \sum_{t=1}^{T} \varphi_t(\omega_{j_1,\dots,j_n}) \cdot \Delta \hat{S}_t(\omega_{j_1,\dots,j_n})\big)^2 \end{aligned} \tag{2.11}$$

In the equation we have $\tilde{H} = H - V_0$, so it represents the discounted payoff of the claim minus the starting capital V_0, which can be derived in a standard way by computing the expectations under the risk neutral measure.

Using the notations from equation 2.11 and considering the structure of the strategy φ it is clear that the individiual φ_t are not dependent on all the sums but only on the first ones. Furthermore also the $\Delta\hat{S}_t$ are not dependent on all the sums, and thus to solve the problem it will be necessary to find a good way to take these properties into account.

Chapter 3

Approaches to solve the problem

The most simple and naive approach to solve the problem formulated in the previous chapter would be to look at the problem without the integer restrictions and then simply rounding the results of this relaxed problem to the next integer value. Since the multinomial model for $L \geq 3$ is already an incomplete market model, one could also use the minimum variance criterion to obtain an optimal strategy for the relaxed problem. The resulting task of finding this optimum has already been studied a lot and can be solved by using the dynamic programming idea developed by Bellman [5] (see also e.g. Fedotov and Mikhailov [15], Yamada and Primbs [31] or Föllmer and Schied [16] for pricing and hedging in incomplete markets using the dynamic programming principle).

Since this naive approach might not give the best results, we also want to look at more sophisticated ways of solving the interger hedging problem. Considering the nature of the problem, the most obvious idea is to look at integer programming techniques. One of the most well known methods for such kind of problems is the Branch and Bound (BB) algorithm, orginated in the 1960's by Land and Doig [21], which is able to solve a vast amount of optimization problems. Another way to look at our problem is using lattice theory and searching for the element of a given lattice which is closest to a vector, also known as closest vector problem (CVP), which is a well known computational problem with many applications. While there are probably more methods for finding an optimal integer hedging strategy, we will follow these two ideas within this thesis.

3.1 Branch and Bound (BB) algorithm

Developed in the early 60's these types of algorithms have already been studied a lot throughout the history (see e.g. Lawler and Wood [22], Nemhauser and Wolsey [26]) and more recently Morrison et al. [25] have given a sound survey of the advances regarding BB.

In a financial context, the BB algorithm has been used to solve integer pricing and hedging problems by Bonami and Lejeune [8].

3.1.1 Algorithm overview

Lets begin with the type of problem that should be solved by the algorithm:

$$\min_x f(x) \text{ subject to } x \in X \tag{3.1}$$

where X is a space of valid solutions, which in our case will be of course $\mathbb{Z}^n$, but X can also contain more constraints. The BB algorithm is a tree search algorithm which iteratively builds trees acording to subproblems that are searched for solutions to the problem.

BB starts by solving the classical (relaxed) optimization problem without any (integer) constraints on the solution space. Additionally a feasible solution $\hat{x} \in X$ is stored globally; this is called the **incumbent solution**, which can also be an initial target objective value of the function to maximize. Now let x^* be a solution to the relaxed problem, then we have the following scenarios: if x^* is already a feasible solution, i.e. it already only contains integer values, and is better than the incumbent solution, i.e $f(x^*) < f(\hat{x})$, then we have found a solution to problem 3.1. Otherwise the algorithm creates subproblems $S_1, \ldots, S_r \subset X$ and tries to find a solution in each of these subproblems. The algorithm selects one of these subproblems, S_i, and explores it. If the algorithm finds a feasible solution, which has better objective value than the actual optimal value $\hat{x}$, then this solution will replace $\hat{x}$ as new incumbent solution. If the algorithm cannot find any better solution than the incumbent, the subproblem is **pruned** and another subproblem is explored. If the selected subproblem cannot be pruned, then child subproblems are created and stored in in the set of the subproblems. The algorithm searches then through all of the subproblems, and once it is done it returns the best incumbent solution.

Morrison et al. [25] provide pseudocode for the BB procedure as seen in algortihm 1.

From the Pseudocode and the brief overview, we see that there are three key factors that influence the BB algorithm and the effectiveness and performance of it. First there is the search strategy, i.e. the order in which the subproblems are searched. Second there are branching rules, so which variable is chosen for branching, i.e how the children are generated from a subproblem. And lastly the pruning rules, which indicate when the algorithm can stop exploring a specific subproblem and thus explore the remaining and more relevant problems. In the following section we will take at look at these issues for our specific case of integer

Algorithm 1 Pseudocode for the BB procedure

```
Set L = X and initialize x̂
while L ≠ ∅ do
    Select a subproblem S from L to explore
    if a solution x̄ with f(x̄) < f(x̂) can be found then Set x̂ = x̄
    if S cannot be pruned then
        Partitions S agian in subproblems S_1, ..., S_r
        Insert S_1, ..., S_r into L
    Remove S from L
return x̂
```

hedging.

3.1.2 Branching, searching and pruning

In this section we will look a bit closer into the details of the BB algorithm. As mentioned previously there are three important factors that impair the runtime and performance of the algorithm and we want to discuss them in a bit more detail. However since this is not the core of the thesis we will only discuss the general ideas for each of the factors and refer to Achterberg [1] or Morrison et al. [25] for more insight into these ideas.

1. Branching:

   This part of the algorithm examines the problem of how new subproblems, also called children, are created from the current problem considered. As choosing a proper node to branch on, might have a big impact on the performance of the algortihm, this aspect has been studied a lot. In general, there are two components of the branching strategy; one is the selection of the branching variable and the other is creating the subproblems.

   The selection component for the branching is done by picking a suitable variable regarding a criterion, i.e. calculate a score for each possible variable to branch on and then choose the variable with the highest score. In integer programming, the only possible variables to branch on are integer variables that have a fractional value in the current relaxed problem. After selecting a branching variable x_j, it is common to create the new subproblems by setting the bounds for x_j as follows: Let $\hat{x}_j$ be the (fractional) value obtained from solving the relaxed problem. Then one subproblem is such that $x_j \geq \lceil \hat{x}_j \rceil$ and the other is such that $x_j \leq \lfloor \hat{x}_j \rfloor$.

   For the runtime of the algorithm the more important component is of course the selection procedure. Thus there have been many ideas in this direction. In the following we will

mention the most common ones.

One of the most traditional branching rules is the most infeasible or most fractional branching rule. Here the variable selected is the one with the most fractional part, meaning that its fractional part is closest to the value 0.5. The idea is to pick a variable, where it is the least clear in which direction, namely up or down, the rounding is optimal. As opposed to this rule another idea for branching is the least feasible or least fractional rule, where the selected variable is closest to next integer number. However though these rules are the most common, they do not prove to be very efficient, as Achterberg [1] has shown. A more efficient rule has goes back to Benichou et al. [6]. The pseudocost branching rule attempts to calculate the per unit change in the objective function, when using a candidate variable to branch on, based on past experience in the tree. Of course this branching method works not so well in the beginning, as there is no information about past branching. Another more sophisticated way of branching is strong branching which was developed by Applegate et al. [3]. Here the variable that produces the most change in the objective function is selected. Obviously this branching strategy can affect the runtime quite heavily as computing the value of the objective function can be time consuming. Thus a mix of strong branching and hybrid branching has been proposed by Linderoth and Savelsbergh [24].
Finally it is interesting to mention that Bonami and Lejeune [8], who used a Branch and Bound algortihm in a financial context, worked out their own branching strategies for the portfolio optimization case, showing that coming up with a good approach for the specific problem can increase the performance drastically.

2. Searching:

The searching component of the algorithm determines, how to proceed one after one with the next subproblem to explore. As branching creates new subproblems, the set of these increases and the algorithm has to check all of the created problems to find an optimal solution. The strategy of how these subproblems are combed through can impact not only the runtime of the algorithm, but sometimes it can also be handy to use a specific apprach to save memory storage.

One very intuitive searching variant is the depth-first search (DFS). Here the idea is to explore the subproblems in a hierarchical way, thus always exploring a child of the previous subproblem. If there is a subproblem that has no child, meaning that it is pruned, then the algorithm jumps back to most recent parent problem. However, if there is still an unexplored subproblem left in this parent node, then it goes on to process this one.

Otherwise it will again trace back to the next ancestor. The DFS strategy has various advantages over other methods; first of all, by always considering a child of the ancestor as next problem, it is not necessary to adjust the relaxation of the problem drastically, as the child will have a similar structure. Often it might even be possible to reuse information from the parent problem. Another advantage is its low cost of memory, which comes from the fact that it is only necessary to trace the nodes in the path from the root started. However there are also problems that can arive with DFS. The main disadvantage is that depth-first can spend a large amount of time in exploring regions that are not close to the problem. This can especially happen when the search tree is extremly unbalanced, meaning that the optimal value can be found in a subproblem near the tree. In an unlucky case, the search strategy can spend a long time exploring irrelevant paths.

As opposed to depth-first, there is the idea of breadth-first search (BrFS). In this case problems are explored in such way that first all the problems with a fixed length to the root are reviewed. If all of the problems in one level have been examined and if there are new subproblems with a greater distance to the root, the algorithm continues by exploring these. This type of searching strategy is not as memory efficient as the DFS and in general does not make good use of pruning rules and is thus not very often used in BB algorithms.

Another commonly used search stratey is the best-first search (BFS) strategy . In this case the concept is to always choose the one node to explore, which is the best regarding some kind of measure, i.e. minimizes the value of the measure. Typically such a measure-of-best function considers a lower bound on the value of the best solution in the subproblem. The advantages of the BFS approach in comparison to other methods is clearly that it is not tied to a specific region of the search tree and thus is often able to find good solutions very fast. As a matter of fact Achterberg [1] showed that – given a constraint integer program with a fixed branching rule – there always exists a best first type of search which minimizes the number of nodes to be processed. However there are also disadvantage such as storage problems for very big search trees, as well as employing a proper measure for the strategy.

Finally it is important to mention that the best working ideas, DFS and BFS, both have its advantages and disadvantages. Therefore more recently various approaches of mixing these strategies in order to make use of the individual benefits have been developed. We again refer to Achterberg [1] for more details on this issue.

3. Pruning:

Besides searching and branching strategies there is a third factor that might influence

the runtime of the algorithm, namely the pruning rules. These define how the algorithm selects the problems which are to explore and the ones that are not. If not for the pruning rules, the algorithm would have to go through every subproblem created by branching. This could cause a lot of runtime as exploring suboptimal regions gives no valuable input. These rules are often very closely related to a specific type of problem. Thus they might substantially vary in each case. However there are some general rules which often come into play for integer constraints problem.

The most common and intuitive pruning rule – besides having found a valid solution to the constraint problem, which of course also prunes a branch – is introducing bounds on the objective value. In general if a valid solution to the problem is found, i.e. in an integer constraint case a solution, where the relevant elements are integers and all possible other constraints are also satisfied, then the objective value might be saved and used as a lower bound. If at any node, the relaxed problem alreday has a worse (i.e lower) objective value, then the subproblem can be pruned and thus need not to be explored.

As mentioned there is a lot of variety in the pruning rules that are usually very problem specific. However another idea that is closely related to pruning rules is the idea of cutting planes (which was introduced by Gomory [18]). The cutting plane is an added constraint to tighten the feasible region without affecting the integrality constraints. This approach leads to a variant of the branch and bound algortihm, called branch and cut.

3.1.3 BB procedure for integer hedging

In this section we will have a closer look at the mode of operation of the BB algorithm for our specific problem.

As already mentioned in section 2.3, we operate in a multinomial model, where we will primarily look at the special case of equation 2.8. The goal is to minimize the hedging error of a claim under the $L^2(P)$-norm:

$$\min_{\varphi} ||H - V_T||_2^2 \text{ s.t. } \varphi \in \mathbb{Z}^{\tilde{d}}$$

where $\tilde{d} = (L-1)T(T-1)/2 + T$. For our algorithm we will use the classical idea for solving interger constraint problems with the BB approach. This means that we will first solve the relaxed problem without any integrality constraint and then choose a suitable (non-integer) variable to branch on. In the following we will discuss the main features of the algorithm regarding the 3 main components of the BB approach as discussed in the previous section.

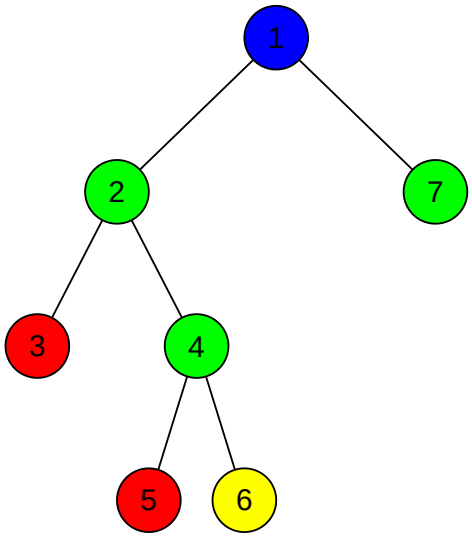

Figure 3.1: Mode of operation of the used BB algorithm. The blue node is the root node, green nodes are the ones that are yet to be explored in detail, red nodes are the ones that are pruned, yellow node is the current node. Numbers inside the node signify the order in which the algorithm explores them.

A visualization of the algorithm can be found in figure 3.1.

1. Branching:

   From the previous section we know that the most common rule to use for branching is the most fractional rule, which selects the variable φ_i whose fractional part is the closest to 0.5 to do the branching on. As Achterberg [1] has shown, this method is in general no better than selecting a variable at random. Therefore our attempt for our BB algorithm does exactly this, namely choosing a non integer variable to branch on at random.

   After the variable is selected, the algortihm follows the classial approach where it creates two new branches. First a branch is created where the selected variable φ_i is set to $\lfloor \varphi_i \rfloor$ and in the same step a branch is generated where φ_i is set to $\lceil \varphi_i \rceil$. For each branch the

relaxed solution is again computed now under the restriction for φ_i and the procedure start over again and thus the search tree is produced accordingly.

2. Searching:

As discussed the most common searching strategies are depth-first searching (DFS), best-first searching (BFS), where each of its strategies has its own advantages. For the time being our algorithm follows a depth first strategy, where we pursue the rule that the first explored subproblem after branching is the branch with $\lfloor \varphi_i \rfloor$. From our standpoint of view this is sufficient for the problems we discuss in this thesis. Furthermore the property of using less memory is very helpful and as has been shown in Linderoth and Savelsbergh [24] often integral solutions are found deep in the search tree, thus preferring a depth search. Nevertheless it might be adequate to introduce a hybrid of both strategies, but since our idea is not to optimize runtime we leave that for the time being.

3. Pruning:

Lastly we discuss the pruning rules used in our implementation of the BB algorithm. Here we follow the ideas of Bonami and Lejeune [8], which are the most intuitive and also widely use rules for stopping the iterative search in a branch of the tree. Pruning of a node is done when one of the following 3 criteria is met:

- the optimal solution of the sub-problem consist of integer variables (pruning by optimal integer feasibility)
- the optimal solution of the sub-problem is not better than the value of an already (in another branch of the tree) found best integer solution (pruning by bounds)
- the optimal solution of the subproblem is infeasible, i.e. no solution exists (pruning by infeasibility)

As we have already mentioned there might be improvements in the Branch and Bound algorithm used for this thesis, but nevertheless it works perfectly well for our case. Details about runtime and performance issues are discussed in the corresponding application sections below.

3.2 Closest Vector (CVP) algorithm

The closest vector problem is a well known lattice based problem that has been used in various fields such as cryptography or number theory. It has been proven to be an NP-hard problem and thus a lot of research has been centered on developing fast algorithms to solve the problem. A survey with an profound introduction to the problem as well as state of the art approaches to solve it can be found in Hanrot et al. [19]. Another nice approach for the problem has been given by Becker et al. [4] . For this thesis we mainly follow the algorithm described in Agrell et al. [2], where their implementation is based on the Schnorr-Euchner method [29].

3.2.1 Algorithm overview

Before starting to decribe the CVP algortihm used in this work, we need to recap shortly some lattice theory. Since we use the algorithm from Agrell et al. [2], we will therefore mainly use their notation.

A lattice is always generated by a matrix with real entries and linearly independent rows (or colums, depending on the represention). Given such a generator matrix G with n rows and m columns, where $n \leq m$, a lattice is given as:

$$\Lambda(G) = uG : u \in \mathbb{Z}^n$$

As a generator matrix of a lattice needs not to be unique, we say that generator matrices G_1 and G_2 create identical lattices, i.e. $\Lambda(G_1) = \Lambda(G_2)$ if and only if

$$G_1 = WG_2$$

where W is a square matrix with integer entries such that $|\det(W)| = 1$. Often it is also useful to change the representation of the generator matrix G_1 to a lower triangular matrix. Thus we will often look at

$$G_1 = G_2Q$$

where G_2 is square and lower triangular and $Q^tQ = I$. G_2 is then said to be the lower triangular representation of G_1.

The reason why it is necessary to discuss the above representations of a generator matrix, is that for many CVP algorithms it is necessary or advantageous to select a proper basis for the lattice. The idea of finding a good representation of the generator matrix for a

CVP algorithm is called reduction. The two most commonly used reduction techniques are the Korkine-Zolotaroff (KZ) reduction [20] and Lenstra-Lenstra-Lovasz (LLL) reduction [23]. Since we do not use one of these techniques we refrain for discussing them in detail here and refer to the original papers, or Agrell et al. [2] for more details on reduction issues.

Finally we can describe a closest vector problem as the problem of finding the element of the lattice $\hat{u} \in \Lambda(G)$ such that

$$||x - \hat{u}|| \leq ||x - u||, \forall u \in \Lambda(G)$$

In this case x is a given point in $\mathbb{R}^m$, which the lattice element should be closest to.

After having established the general problem and some basic ideas from lattice theory we can now start inspecting the algorithm we use in this thesis in more detail. Therefore let us again consider the $n \times m$ generator matrix G for a lattice. We can think about G as

$$G = \begin{bmatrix} G^* \\ \boldsymbol{v}_n \end{bmatrix}$$

where G^* are the first $(n-1)$ rows of the generator matrix and $\boldsymbol{v}_n = \boldsymbol{v}^* + \boldsymbol{v}^\perp$ where $\boldsymbol{v}^*$ is in the row-space of G^* and $\boldsymbol{v}^\perp$ is in the null space. This decomposition is particularly easy if G is a triangular matrix, then we have $\boldsymbol{v}^* = (v_{n1}, \ldots, v_{n(n-1)}, 0)$ and $\boldsymbol{v}^\perp = (0, \ldots, 0, v_{nn})$.

With the above terminology it is possible to rewrite the lattice in the following way:

$$\Lambda(G) = \bigcup_{u_n=-\infty}^{\infty} \{\boldsymbol{c} + u_n \boldsymbol{v}^* + u_n \boldsymbol{v}^\perp : c \in \Lambda(G^*)\}$$

Thus we can represent the lattice as $(n-1)$-dimensional sublattices, which are called layers. Here the vector $\boldsymbol{v}^\perp$ is normal to the layers and the distance of two adjacent layers is $||\boldsymbol{v}^\perp||$, which for triangular matrices is conveniently given as $||\boldsymbol{v}^\perp|| = |v_{nn}|$.

All of the existing search algortihms for lattice problems are iteratively searching through a finite number of layers to solve the problem. The Schnorr-Euchner method hereby searches through the layers via the following sequenece

$$u_n = \lfloor \hat{u}_n \rceil, \lfloor \hat{u}_n \rceil - 1, \lfloor \hat{u}_n \rceil + 1, \lfloor \hat{u}_n \rceil - 2, \ldots$$

where $\hat{u}_n = \frac{x(v^\perp)^t}{||\boldsymbol{v}^\perp||^2}$ and x is again the point to which the closest lattice element is searched. This strategy proves to be very efficient in encountering the closest element early on and is an optimized variant of other searching strategies.

As already mentioned earlier this algorithm is taken from Agrell et al. [2], where they also describe in more detail other searching methods and provide pseudocode for the algorithm. Lastly it is necessary to mention that for the closest point search via the described technique a triangular generator matrix is needed. This can be easily achieved by performing a QR-decomposition on the original matrix.

3.2.2 CVP for integer hedging

In this part we want to discuss how we can use the CVP algorithm for our integer hedging problem. To do so we need to take a closer look at the problem described via equation 2.10. For the case, where the asset price evolves according to a multinomial lattice with L possible future values of the price in the next timestep, there are L^T different paths the asset can move.

The idea for being able to use a CVP type algorithm is to order the paths appropriately, in such a way that the strucure of the dependencies of the desired trading strategy φ on the different ω_i with $i \in \{1, \dots, L^T\}$ becomes more clear. As already mentioned and as can easily be seen from the representation of our integer hedging problem via equation 2.11, the φ are not dependent on every single path ω_i. In fact φ_1 is even deterministic and we have already shown how the sturcture of strategy in each timestep depends on the multinomial tree of the model (c.f. section 2.3). Therefore, by properly ordering the paths ω_i and by remembering that each time component φ_t of the strategy has $(t-1)(L-1)+1$ relevant components, it is possible to rewrite equation 2.10 in terms of each individual element of $\varphi = (\varphi_1, \varphi_{2_1}, \dots, \varphi_{2_L}, \dots, \varphi_{T_1}, \dots, \varphi_{T_{(T-1)(L-1)+1}})$

In combining all these findings one can see that our integer hedging problem amounts to a closest vector problem where the lattice is an $L^T \times \tilde{d}$ matrix which has the following form:

$$\left\{ \varphi_1 \begin{pmatrix} \Delta S_1(\omega_1) \\ \vdots \\ \\ \\ \vdots \\ \\ \\ \vdots \\ \Delta S_1(\omega_{L^T}) \end{pmatrix} + \quad \varphi_{2_1} \begin{pmatrix} \Delta S_2(\omega_1) \\ \vdots \\ \Delta S_2(\omega_{L^{T-1}}) \\ 0 \\ \vdots \\ \\ \\ \vdots \\ 0 \end{pmatrix} + \ldots \; \varphi_{2_L} \begin{pmatrix} 0 \\ \vdots \\ \\ \\ \vdots \\ 0 \\ \Delta S_2(\omega_{(L-1)L^{T-1}}) \\ \vdots \\ \Delta S_2(\omega_{L^T}) \end{pmatrix} + \ldots \right.$$

$$\left. + \; \varphi_{T_1} \begin{pmatrix} \Delta S_T(\omega_1) \\ 0 \\ \vdots \\ \\ 0 \\ \vdots \\ \\ \vdots \\ 0 \end{pmatrix} + \ldots \qquad \ldots + \; \varphi_{T_{(T-1)(L-1)+1}} \begin{pmatrix} 0 \\ \vdots \\ \\ \\ 0 \\ \vdots \\ \\ 0 \\ \Delta S_T(\omega_{L^T}) \end{pmatrix} : \varphi \in \mathbb{Z}^{\tilde{d}} \right\} \tag{3.2}$$

where $\tilde{d}$ is again given as $(L-1)T(T-1)/2+T$. To solve the problem the algorithm searches for the lattice point closest to the vector:

$$\begin{pmatrix} \tilde{H}(\omega_1) \\ \\ \tilde{H}(\omega_{L^t}) \end{pmatrix} \tag{3.3}$$

Having written down the problem we can now solve it by using a CVP algorithm. Here it is worth to mention that we do not use any preprocessing of the lattice using some reduction techniques, since for the examples we used the runtime was not yet an issue and thus we have refrained from that. Further details about the performance and other issues regarding the CVP approach are described in the applications section below.

Chapter 4

Applications to specific models

In the preceding chapters we have derived a framework in which we work and also presented two approaches to solve an integer hedging problem. In this chapter we try to implement the previously derived ideas and apply them to specific discrete time models.

The most well known and also most used discrete time lattice models are the binomial model, derived by Cox et al. [12], and the trinomial model which is a natural extension of it. These models have various properties which make them very popular. First of all the structure of them is very easy and intuitive thus these models have been studied a lot. Second of all it is possible to show that they are able to approximate continuous time models such as the well known Black Scholes model [7].

In the following we will recap the most important porperties of the two above mentioned models and see how the two algorithms apply to theses specific case.

4.1 The Binomial model

As mentioned before this model is a special case of our general model from 2.8, where $L = 2$. Thus, besides the two parameters u and d we also want to find the probability p of moving up. Since we are interested in risk neutral valuation for our claims, we need to find conditions which p has to fullfill.

Theorem 4.1.1. *The binomial model is arbitrage free if and only if $d < 1 + r < u$. The martingale measure defined through the probability $p = \mathbb{P}[S_t = S_{t-1} * u]$ is unique and thus the model is a complete market model. Furthermore we have*

$$p = \frac{(1+r) - d}{u - d}$$

Proof. In order for the model to be arbitrage free we need to find a martingale measure for our price process, that is the following has to hold:

$$\mathbb{E}[\hat{S}_t|\mathcal{F}_{t-1}] = \frac{pS_{t-1}u}{(1+r)^t} + \frac{(1-p)S_{t-1}d}{(1+r)^t} \overset{!}{=} \hat{S}_{t-1} = \frac{S_{t-1}}{(1+r)^{t-1}}$$

Reordering the terms gives the equivalent condition that

$$p = \frac{(1+r)-d}{u-d}$$

Thus $d < (1+r) < u$ is a necessary and sufficient conditon for the measure defined by p and since the term for p is independent of the time t we find that this expression is unique. □

Since the binomial model is complete, it is possible to perfectly hedge any given claim. To find the replicating strategy a recursive scheme can be applied. Let $\tilde{\phi}_t = (\phi_t^0, \phi_t)$, $t \in \mathbb{T}\backslash\{0\}$, be a replicating strategy, we can start by finding the value for ϕ_T via the equation:

$$\hat{V}_T = \phi_T^0 + \phi_T \hat{S}_T$$

Rewriting this we get:

$$\hat{V}_T - \phi_T \hat{S}_T = \phi_T^0$$

And since the right hand side is predictable, thus known at time $T-1$, also the left hand side must be known at $T-1$ and we can find ϕ_T by solving the equation

$$V_T^u - \phi_T S_{T-1} u = V_T^d - \phi_T S_{T-1} d \Leftrightarrow \phi_T = \frac{V_T^u - V_T^d}{S_{T-1}u - S_{T-1}d}$$

where V_T^u and V_T^d are the values of the claim if the underlying moves up from time $T-1$ respectively the down movement. Finally we can compute ϕ_T^0 via $\hat{V}_T - \phi_T \hat{S}_T = \phi_T^0$. This procedure can be repeated until we arrive at $\tilde{\phi}_1$.

Thus far we have revisited some results for the binomial model, which are very helpful and important when dealing with pricing and hedging under this model. Before discussing the main part, namely the application of the two algorithms of the previous section for the integer hedging problem, we briefly want to discuss another important aspect of the model.

As we have seen there are basically two parameters we have to choose, for this model to work, namely u and d. A priori it is not so clear how to select these parameters in order

to use the model in practice. Intuitively the model should capture the market structure of the asset. Thus it is of interest to determine the up and down rates such that the model fits the actual market data as close as possible. The usual idea to achieve this is to match these parameters with the moments of the underlying distribution of the asset, see e.g. Yamada and Primbs [31] for details on that.

Suppose we are looking at an asset in a time period of $[0, T]$ (e.g. any kind of time unit like days, weeks, etc.). To illustrate the usage of the model lets fix this interval to be 1 day. The price of a stock can evolve in many directions and can take many different values at the end of the day, especially more than two values, as the simple binomial model would assume. Thus one typically divides the interval into $N+1$ periods, and let the price evolve as a binomial tree over these $0 = t_0, \ldots, t_N = T$ periods. Using the information of the moments of the distribution of an underlying stock we can try to come up with values for u and d such that our binomial model fits to this distribution.

In pratice, the most common assumption is that the asset follows a geometric brownian motion, as in the Black Scholes model. Thus the distribution of the asset price follows a lognormal distribution. For this case Cox et al. [12] already showed how the parametrization needs to be chosen in order to achieve approximation of the binomial model price to the Black-Scholes price. Assuming equally spaced timesteps, $t_i - t_{i-1} = T/N$, for $i = 1, \ldots, N$, they presented the choices for the price movements as

$$u = e^{\sigma\sqrt{T/N}}, \quad d = e^{-\sigma\sqrt{T/N}}, \text{and thus } p = \frac{1}{2} + \frac{1}{2}\frac{\mu}{\sigma}\sqrt{T/N}$$

where μ and σ are the drift respectively the volatility in the Black Scholes model. Note that it also necessary to adjust the evolvement of the riskless asset to $(1+r)^{T/N}$ in each period. Letting $N \to \infty$, i.e. increasing the steps of the binomial lattice, the price converges to the Black-Scholes price.

4.1.1 Integer hedging results

We saw in the previous section why the binomial model is a popular and very often used in real life trading. For our analysis we are nevertheless more interested in the performance of the integer hedging algorithms when using this model. Thus we calculated the optimal integer hedging strategy for different numbers of timesteps, and also varied the values of the parameters u,d and r to get a feeling of the behavior of the approaches.

Besides varying the parameters, we also investigated two different options as claims; first a simple European call option, i.e. an option whose payoff is given as $(S_T - K)^+ = \max(S_t - K, 0)$. Secondly we looked at a corridor option, i.e. an option where the price stays

within a given corridor from the starting value. The payoff of such an option is given as $(|S_T - S_0|)\mathbb{1}_{\{S_T \in [S_0-\delta, S_0+\delta]\}}$. The choice of these two types of options are very arbitrary. The european call is an obvious choice since this is the most common type of option, whereas the other one is a bit more special.

The first approach was always the naive approach where we simply rounded the results of the classical optimal strategy. In a next step we computed the optimal strategy by using the BB algorithm described in section 3.1. Finally we also computed the results by using a CVP approach described in section 3.2.The code to both approaches can be found in the appendix A.

Before presenting the results we will briefly discuss some of the properties of the BB algortihm and the CVP algorithm when using them for this model.

BB approach

It is worth to mention that the BB approach used highly depends on the initial solution provided, which gives a first bound for the optimal value, and thus will trigger one of the pruning rules, namely the pruning by bounds (c.f. section 3.1.3). So if one already provides a nice initial value then the algorithm is very fast, even for a high number N of timesteps and thus a high-dimesional value of φ. If the initial value provided is far from optimal then the runtime of the algortihm increases drastically with the number of timesteps and thus is not very efficient. For N up to 5, the algorithm works very well and fast, even when providing a bad starting value. Therefore we will only consider these cases for the moment. Nonetheless it is important to mention that it might be useful to improve the performance by using different approaches for the 3 key performance factors; branching, searching and pruning.

One idea is of course to use the solution of the naive approach, so simply rounding the optimal strategy, as an initial value. As we will see this is already the optimal starting value and thus with this value the BB algorithm will be very fast.

CVP approach

In the case of the CVP approach we have found that there are similarities to the BB approach. In general the algorithm is fast for up to 7 timesteps, without any adaption. Nevertheless the runtime of the algorithm increases exponentially with the increase of timesteps, thus suggesting that for $N > 7$ a preprocessing of the lattice would be helpful to decrease the runtime.

Due to the structure of the algorithm and also due to the fact that the simply rounded

N	2	3
Classical	(0.59,0.97,0.00)	(0.59,0.79,0.28,1,0.46,0)
rounded	(1,1,0)	(1,1,0,1,0,0)
BB	(1,1,0)	(1,1,0,1,0,0)
CVP	(1,1,0)	(1,1,0,1,0,0)

N	4
Classical	(0.63, 0.81, 0.34, 0.97, 0.57, 0, 1, 0.93, 0, 0)
rounded	(1, 1, 0, 1, 1, 0, 1, 1, 0, 0)
BB	(1, 1, 0, 1, 1, 0, 1, 1, 0, 0)
CVP	(1, 1, 0, 1, 1, 0, 1, 1, 0, 0)

T	5
Classical	(0.63, 0.78, 0.41, 0.91, 0.57, 0.16, 1, 0.78, 0.26, 0, 1, 1, 0.43, 0, 0)
rounded	(1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0)
BB	(1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0)
CVP	(1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0)

Table 4.1: European Call option results

value is already the optimal value the algorithm however is able to find the correct solution very quickly. The high runtime of the algorithm comes due to checking the criterion implemented for the layers, thus even though the optimal solution is found very fast, the algorithm can take a while. Thus is it possible to insert a check into the algorithm which speeds the procedure up, but also prevents the algortihm from checking every layer. This would increase the performance of the algorithm also for high values of N and can be used in our case.

Results

We will now present some results from the different algorithms for the different options and different number of timesteps N. We will fix the parameters to

$$u = 1.07,\ S_0 = 100$$
$$d = 0.93,\ K = 100$$
$$r = 0.01,\ \delta = 25$$

This is only due to convenience and the choice has no particular reason. Furthermore it is essential to mention that we varied the parameters a lot, but the outcomes have not changed and thus we only provide the results for this particuar case in this work. To make the structure of the table more clear we ask the reader to notice that we arrange the results in a way such that $\varphi = (\varphi_1, \varphi_{2_1}, \varphi_{2_2}, \ldots, \varphi_{T_1}, \ldots, \varphi_{T_T})$.

Table 4.1 shows the results of our algorithms, as well as the simply rounded strategy for

N	2	3
Classical	(0.17,0.93,-1)	(0.18,0.58,-0.44,1,-0.07,-1)
rounded	(0,1,-1)	(0,1,0,1,0,-1)
BB	(0,1,-1)	(0,1,0,1,0,-1)
CVP	(0,1,-1)	(0,1,0,1,0,-1)

N	4
Classical	(-0.01, -0.04, 0.03, -0.15, 0.13, -0.12, -0.81, 0.87, -1, 1.24)
rounded	(0, 0, 0, 0, 0, 0, -1, 1, -1, 1)
BB	(0, 0, 0, 0, 0, 0, -1, 1, -1, 1)
CVP	(0, 0, 0, 0, 0, 0, -1, 1, -1, 1)

N	5
Classical	(0.04, 0.07, 0, 0.02, 0.14, -0.23, -0.33, 0.55, -0.48, 0.15, -1.19, 1, -0.14, -1, 1.91)
rounded	(0, 0, 0, 0, 0, 0, 0, 1, 0, 0, -1, 1, 0, -1, 2)
BB	(0, 0, 0, 0, 0, 0, 0, 1, 0, 0, -1, 1, 0, -1, 2)
CVP	(0, 0, 0, 0, 0, 0, 0, 1, 0, 0, -1, 1, 0, -1, 2)

Table 4.2: Corridor option results

the european call option. Here one can already see how the integer constraints affect the optimal strategy. As can be seen, in the binomial model, it is sufficient to simply round the outcome of the classical hedging problem to find the integer optimum. Neither of our more eloborated algorithms used is able to come up with a better strategy for the problem. Table 4.2 confirms this result for the second type of option used in our example cases, the corridor option.

The results for the Binomial model are far from being unexpected. In fact in his master thesis, Breese [11, sec. 3] shows as one of his main results that in the Binomial model the rounded strategy is the optimal integer strategy.

4.2 The Trinomial model

The Trinomial model is a natural extension of the binomial model of the previous section. As opposed to the binomial model, in this case the price can evolve in three different ways in each time step, thus we have a multinomial lattice model from section 2.3 with $L = 3$. We will define the three states as $\{\tilde{u}, \tilde{m}, \tilde{d}\}$, where $\tilde{u} = u^2$, $\tilde{m} = ud$ and $\tilde{d} = d^2$, resulting in an evolution of the price that can be represented as follows:

$$S_t = \begin{cases} S_{t-1}\tilde{u} & \text{with prob. } p_u \\ S_{t-1}\tilde{m} & \text{with prob. } p_m \\ S_{t-1}\tilde{d} & \text{with prob. } p_d \end{cases} \tag{4.1}$$

As usual we are interested in risk neutral valuation of the claims in this model, so we need to determine the corresponding risk neutral probabilites, p_1, p_2, p_3, of ending up in state $\tilde{u}, \tilde{m}, \tilde{d}$. To do so, we need to solve the following system of equations:

$$p_1\tilde{u} + p_2\tilde{m} + p_3\tilde{d} = (1+r),$$
$$p_1 + p_2 + p_3 = 1.$$

If $d < 1 + r < u$ there is no unique solution to the above system of equations, thus the trinomial model is not a complete market model and we have infinetely many possible choices for the martingale measure. In order to find one parametrization it is usually common to fix one value of the probabilities and then determine the other ones.

As in the case of the binomial model, also the trinomial model finds a lot of attention in practice, since it is easy to handle and quite flexible. To use the model in practice there are various possibilities to calibtrate the parameters by matching them with the moments of the underlying distribution of the assets (we again refer to Yamada and Primbs [31] for the general procedure).

A special case is again the one where the asset follows a geometric Brownian motion, so the Black-Scholes model. In this case we can follow Boyle [9] to determine the risk neutral probabilities and the parameters u and d. Assuming equally spaced timesteps, $t_i - t_{i-1} = T/N = \Delta t$, for $i = 1, \ldots, N$, we have the following representations:

$$u = e^{\sigma\sqrt{2\Delta t}} \quad d = \frac{1}{u} \quad \text{and thus} \quad m = 1$$
$$p_1 = \left(\frac{e^{r\Delta t} - e^{-\sigma\sqrt{\frac{\Delta t}{2}}}}{e^{\sigma\sqrt{\frac{\Delta t}{2}}} - e^{-\sigma\sqrt{\frac{\Delta t}{2}}}}\right)^2$$
$$p_2 = 1 - p_1 - p_3$$
$$p_3 = \left(\frac{e^{\sigma\sqrt{\frac{\Delta t}{2}}} - e^{r\Delta t}}{e^{\sigma\sqrt{\frac{\Delta t}{2}}} - e^{-\sigma\sqrt{\frac{\Delta t}{2}}}}\right)^2$$

where σ is again the volatility of the asset in the Black-Scholes model. Letting now $N \to \infty$, i.e. increasing the time steps, the price of the trinomial model converges to the Black-Scholes price.

4.2.1 Integer hedging results

The analysis here is pretty similar to the previous case of the binomial model. As we have seen in the previous part there is no unique martingale measure, thus besides fixing the parameters u, d and r, we will also specify p_2 (which is the probability for the price to evolve as $\bar{m} = ud$, c.f. section 2.3) and caluclate the other two probabilites accordingly. We again look at the same two option used in the binomial case, an european call option and a corridor option. For the presentation of the results we will again fix the parameters to the following values

$$u_1 = 1.07,\ u_2 = 0.93$$
$$S_0^1 = 100,\ K = 100$$
$$r = 0.01,\ \delta = 25$$
$$p_2 = 0.25$$

Although the results are presented for the above fixed values, it is important to mention that we have tried to vary the paratemers a lot, but since the results are similar in any case we do not look more closely into the selection of parameters. For the results we proceed as usual, i.e. we compare the results of the naive approach to the ones from the algorithms described in chapter 3. The codes for the results can again be found in the appendix A.

Before having a closer look at the results we will again briefly discuss some of the properties of the BB algorithm and the CVP algorithm when using them for this model.

BB approach

The BB approach for the trinomial model performs pretty similar to the case of the binomial model. The runtime of the algorithm is again dependent on the initial bound provided for the optimal value. Thus using the naive approach, which is the best one as seen below, results in a very fast performance, which increases quite quickly when providing suboptimal solutions. We only present results for N up to 4, for which the runtime is manageable even when providing a "bad" initial bound.

CVP approach

Also the behavior of the CVP algorithm for the trinomial model is pretty similar to the one of the binomial model. Up to a number of timesteps N of 5 the algorithm is quite fast and does not really need adaption. When increasing the number the runtime increases

N	3
Classical	(0.59, 0.84, 0.56, 0.23, 1, 0.96, 0.5, 0, 0)
rounded	(1, 1, 1, 0, 1, 1, 0, 0, 0)
BB	(1, 1, 1, 0, 1, 1, 0, 0, 0)
CVP	(1, 1, 1, 0, 1, 1, 0, 0, 0)

N	4
Classical	(0.6, 0.82, 0.58, 0.3, 0.98, 0.83, 0.55, 0.22, 0, 1, 1, 0.94, 0.48, 0, 0, 0)
rounded	(1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0)
BB	(1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0)
CVP	(1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0)

Table 4.3: European Call option results for the trinomial model

N	3
Classical	(-0.02, -0.18, -0.03, 0.21, -0.35, -0.07, 0, 0.02, 0.71)
rounded	(0, 0, 0, 0, 0, 0, 0, 0, 1)
BB	(0, 0, 0, 0, 0, 0, 0, 0, 1)
CVP	(0, 0, 0, 0, 0, 0, 0, 0, 1)

N	4
Classical	(-0.03, -0.11, -0.03, 0.09, -0.11, -0.19, -0.04, 0.21, 0.20, 0, -0.34, -0.09, -0.04, 0.04, 0.73, 0)
rounded	(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0)
BB	(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0)
CVP	(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0)

Table 4.4: Corridor option results for the trinomial model

drastically, thus it might be of advantage using some preprocessing mechanisms to boost the performance.

Results

The results of our exercise can be found in tables 4.3 and 4.4. The first table shows the results for the european call option, the second one shows the analysis of the corridor option. As we can see the naive approach of simply rounding the classical results, provides us with the same strategy as the more sophisticated approaches, thus leading to the conclusion that it is not to recommend the more costly algorithms to solve the integer hedging problem in the trinomial model.

4.3 Models with more than one asset

In the previous sections we studied the most commonly used models for pricing and hedging claims. As we have seen the outcome for the integer hedging problem in these classical models were not very elaborate, the simple approach of rounding to the next integer already gave the desired best results, thus the complex algorithms developed in section 3 of this work were not necessary.

In this section we will look at more sophisticated models, which are based on the two previously defined models. The key modification within these model is that we will not restrict the number of assets modelled to only one. We will look at two distinct models which model two assets at once, but which can easily be adapted for a higher number of assets. Considering two or more assets not only complicates the model, but also leads to a bigger variety of claims which can be considered.

In the following, we examine two models and inspect the impact of the integer constraint on a hedging strategy. To find the optimal strategy we proceed as before, so we will first look at the naive approach and then consider a branch and bound approach as well as a closest vector problem approach.

4.3.1 A lattice framework for 2 Assets

The first model we want to study goes back to the work of Boyle [10]. He developed a lattice framework for two assets which is based on the CRR model and trinomial model. We will adapt the ideas of his paper such that we can use our developed framework, since the concepts are pretty similar.

Boyle [10] proposed a five-jump process to capture the evolvement of the two stocks, S^1 and S^2, in time. In each timesteps these assets move according to multivariate binomial lattice, thus shifting up and down with rates u_1,u_2, respectively d_1,d_2. Accordingly in each timestep there are four possible movements from the state before. Furthermore, to get the five-jump structure, there is also the possibility that the assets do not move in this timestep. Suppose at time t the values of the assets are given by S_t^1 and S_t^2, then the five possible states in time $t+1$ are the following:

Asset 1	Asset 2
$S_t^1 u_1$	$S_t^2 u_2$
$S_t^1 u_1$	$S_t^2 d_2$
$S_t^1 d_1$	$S_t^2 u_2$
$S_t^1 d_1$	$S_t^2 d_2$
S_t^1	S_t^2

From this structure it is easy to see the connenction to our framework described in section 2.3. Implementing the model is nothing else than using an appropriate lattice for the possible events.

Before looking at the outcomes of the integer hedging approaches within this model, it is worthwhile to mention some more properties and some motivation for the usage of such a model.

One very nice feature is that within this framework it is easy to calibrate the model parameters to market data. If one uses the standard assumption that the two assets S^1 and S^2 follow a joint lognormal distribtuion and the variance and covariance structure can be estimated from data, then Boyle [10] shows how to select the values u_1 and u_2 and how to calculate the according risk neutral probabilites $p_1, \ldots, p_5$ for each state.

Using the parametrization of the Boyle [10] paper has another very useful property. It can be shown that within this framework it is possible to approximate the prices for european claims. When using a generalisation of the Black-Scholes model for 2 Assets as proposed by Stulz [30], one can derive analytical results for the European type option. Then by proceeding in a similar fashion as when calibrating the binomial model to market data, i.e. by dividing the time interval into a number of timesteps and using moments to fit the parameters (c.f. section 4.1), the analytical prices are approximated.

Beside the various useful properties this model has also drawbacks such as the possibility of negative probabilities and slow convergence. Thus Ekvall [14] has proposed some modifications on the model to overcome these flaws, making the model easy to implement and very practical for usage in various applications.

Approach	Optimal Strategy	Error
Classical	(0.288,0.304,0.645,-0.298,0.075,0.289,0.249,0.339,-0.164,0.764,-0.010, 0.181)	9.012
rounded	(0,0,1,0,0,0,0,0,0,1,0,0)	19.391
BB	(0,0,1,0,0,1,0,0,0,1,0,0)	19.087
CVP	(0,0,1,0,0,1,0,0,0,1,0,0)	19.087

Table 4.5: Optimal integer strategy in the 2 Asset lattice framework for an European call on the maximum of two assets.

Numerical results

For our numerical studies of the model in our integer constraints framework we will look at an example where we fix the number of timesteps to $N = 2$. Furthermore we analyse the problem of hedging an European call option on the maximum of the two assets, thus the payoff is given as $(\max(S_T^1, S_T^2) - K)^+$. The other parameters used are:

$$u_1 = 1.0551,\ u_2 = 1.0692$$
$$d_1 = 0.9478,\ d_2 = 0.9353$$
$$S_0^1 = 100,\ S_0^2 = 98$$
$$r = 0.025,\ K = 99$$

It is necessary to mention that, as we are only looking for an example to use our integer hedging algortihms, also the parameters for the probabilties are chosen by hand. Since we do not use any real underlying and thus are not interested in calibrating the model to specific data we do not follow the procedure of deriving the according risk neutral probabilites, but instead choose values for the probabilites such that they dont vary to much for every outcome. This goes in line with Boyle [10] and Ekvall [14], who mention that the events should have similar probability weights.

Table 4.5 shows the result of the numerical exercise. In the simple case where we only consider two timesteps, there are 6 different values for each of the two assets for the hedging strategy. The results in the table are ordered in a way such that the first 6 entries are the values for the strategy for S_1 and the last 6 ones are the corresponding entries for S_2. When observing the values one can see that the integer hedging approaches developed in chapter 3 work better than the naive approach which is simply rounding. Between the two algorithms we can see that they both come with the same result and also from a runtime performance there seems to be no preference, at least for this small example.

4.3.2 A complete trinomial model

The second model we want to look at is a modification of the trinomial model for two assets. We follow here the idea of Pascucci and Runggaldier [27], they propose to look at two independent assets S_1 and S_2 which both follow a trinomial model. To follow the notation from section 4.2 (and to keep the strucure of section 2.3), this means that each asset S_i, $i = 1, 2$ has tree possibilites $\{\tilde{u}_i, \tilde{m}_i, \tilde{d}_i\}$ to move in each step, where $\tilde{u}_i = u_i^2$, $\tilde{m}_i = u_i d_i$ and $\tilde{d}_i = d_i^2$, thus:

$$S_t^i = \begin{cases} S_{t-1}^i \tilde{u}_i & \text{with prob. } p_u^i \\ S_{t-1}^i \tilde{m}_i & \text{with prob. } p_m^i \\ S_{t-1}^i \tilde{d}_i & \text{with prob. } p_d^i \end{cases} \tag{4.2}$$

Doing so we can find the martingale measure by solving the following system of equations:

$$\begin{aligned} p_1\tilde{u}_1 + p_2\tilde{m}_1 + p_3\tilde{d}_1 &= (1+r) \\ p_1\tilde{u}_2 + p_2\tilde{m}_2 + p_3\tilde{d}_2 &= (1+r) \\ p_1 + p_2 + p_3 &= 1 \end{aligned} \tag{4.3}$$

Pascucci and Runggaldier [27] showed that the system 4.3 is uniquely solveable and that the parameters can be chosen in such a way that p_1, p_2, p_3 are really valid probabilites. Thus the market model is complete and is straightforward to calculate the optimal hedging strategy.

Numerical results

For the numerical example we will study a similar case as before, i.e. we want to find the optimal integer strategy for a european call on the maximum of the two assets S_1 and S_2. The parameter we choose are similar to the ones before, they are given as:

$$\begin{aligned} u_1 = 1.07,\ u_2 &= 1.084 \\ d_1 = 0.93,\ d_2 &= 0.91 \\ S_0^1 = 100,\ S_0^2 &= 98 \\ r = 0.025,\ K &= 99 \end{aligned}$$

We will again only look into the simple case where the number of timesteps is fixed to $N = 2$. Thus we have in this case eight different values for the hedging strategy where the first four values belong to the strategy for S_1 and the last four to the one for S_2.

Approach	Optimal Strategy	Error
Classical	(-15.74,-7.91,-26.45,0.00,13.51,7.23,22.39,0.00)	0
rounded	(-16,-8,-26,0,14,7,22,0)	16.192
BB	(-15,-9,-26,0,13,8,22,0)	2.039
CVP	(-20,-5,-26,0,17,5,22,0)	0.837

Table 4.6: Optimal integer strategy in the 2 Asset trinomial model for an European call on the maximum of two assets.

Table 4.6 shows the outcome of the example used. In this model we see that the optimal integer strategies vary for each case. By looking at the error it is clear that the naive apporach of simply rounding performs way worse than the BB approach and the CVP algorithm. Furthermore we find that the there are also differences between the two more sophisticated approach. The error of the CVP solution is clearly the smallest, thus this method works best. The reason why the solution of the BB approach differs that much from the CVP one is due to the fact of how the algorithm works. The algorithm always rounds the optimal value received from the relaxed solution. Thus is not able to reach values far away from the classical solution. We see that the CVP optimal value differs substantially from the ones of the rounded solutions, being able to take more values for the strategy into account. Nevertheless the outcome of the BB solution is already a huge improvement to the one of the naive approach.

Chapter 5

Analysis of limiting behaviour

Up to now we have explored the integer hedging problem for various models and to find an optimal strategy under the imposed constraints we have studied two numerical methods. Nevertheless we have seen that for standard models like the binomial and the trinomial model it is sufficient to simply round the results of the classical strategy to find an optimal solution to the integer hedging problem.

In this chapter we want to analyse the behaviour of the integer strategy in the binomial model and see if we can recover completeness in the limit. The idea here comes from the fact mentioned in section 4.1 that the prices for options in the binomial model converge to the price of the Black-Scholes model. The goal is to consider sequences of the Binomial model and observe whether there is a similar kind of convergence for the integer hedging strategy to the classical hedging strategy. To achieve this we will try to infer from numerical outcomes whether the hedging error vanishes when considering more and more timesteps.

5.1 Completeness for a sequence of binomial models

For our analysis we recall the details of the model. We consider an asset that evolves over a fixed timeperiod $[0, T]$ as a binomial tree with $N + 1$ different equally spaced timepoints $0 = t_0, t_1 = \frac{T}{N}, \ldots, t_N = T$ such that $t_i - t_{i-1} = \frac{T}{N}$. The asset evolves in each time step according to

$$S_t^{(N)} = S_{t-1}^{(N)} m, \quad t = 0, \frac{1}{N}, \ldots, \frac{T}{N}$$

where $m \in \{u, d\}$ with $u = e^{\sigma\sqrt{T/N}}$ and $d = e^{-\sigma\sqrt{T/N}}$.

In order to analyse how the optimal integer strategy evolves for this sequence of binomial models as $N \to \infty$, we will analyse the hedging error $H - V_T^{(N)}$ made in each time steps. To

be more precise we will look at the maximum error made, i.e.

$$E_N = \max_{\omega \in \Omega} |H(\omega) - V_{t_N}^{(N)}(\omega)|$$

where H is the discounted claim and $V_T^{(N)}$ is the value process for the selffinancing integer hedging strategy in a binomial model with $N+1$ periods at time $t_N = T$.

As initially mentioned, the idea is to find out what happens to E_N when $N \to \infty$. To start, we will consider a classical European call option, i.e. the payoff is given as $(S_T - K)^+$. The other parameters are chosen as

$$S_0 = 100,\ K = 100$$
$$T = 1,\ \sigma = 0.2.$$

N even	E_N	N odd	E_N	N even	E_N	N odd	E_N
2	5.680	3	13.160	24	6.213	25	7.484
4	9.705	5	14.775	26	5.403	27	6.570
6	11.434	7	15.634	28	4.639	29	5.692
8	11.683	9	15.279	30	3.921	31	4.878
10	11.656	11	14.726	32	3.235	33	4.122
12	11.365	13	13.861	34	2.600	35	3.423
14	10.630	15	12.733	36	2.013	37	2.781
16	9.693	17	11.598	38	1.456	39	2.186
18	8.836	19	10.557	40	0.962	41	1.634
20	7.910	21	9.479	42	0.508	43	1.122
22	7.069	23	8.454	44	0.089	45	0.761

Table 5.1: Maximal error E_N for different number of timesteps

In the following we will present the results of the analysis in two ways. First we present some numerical results of the maximal hedging error E_N for different numbers of N. Table 5.1 shows how the values of E_N evolve as N increases. We make a distinction between an even and odd numbers of N as there results are more visible in that way. The table confirms what we had hope to achieve. After the first few timesteps (after $N > 7$), we observe a decreasing behavior of E_N, which seems to converge slowly to 0. At the last timepoints considered we see that the hedging error is already very close to 0, indicating that the integer constraint strategy converges to the classical solution of the binomial model.

Figure 5.1 gives a graphical representation of what is decribed in table 5.1. The decreasing behavior of the hedging error is clearly visible (until the last timepoints where the values

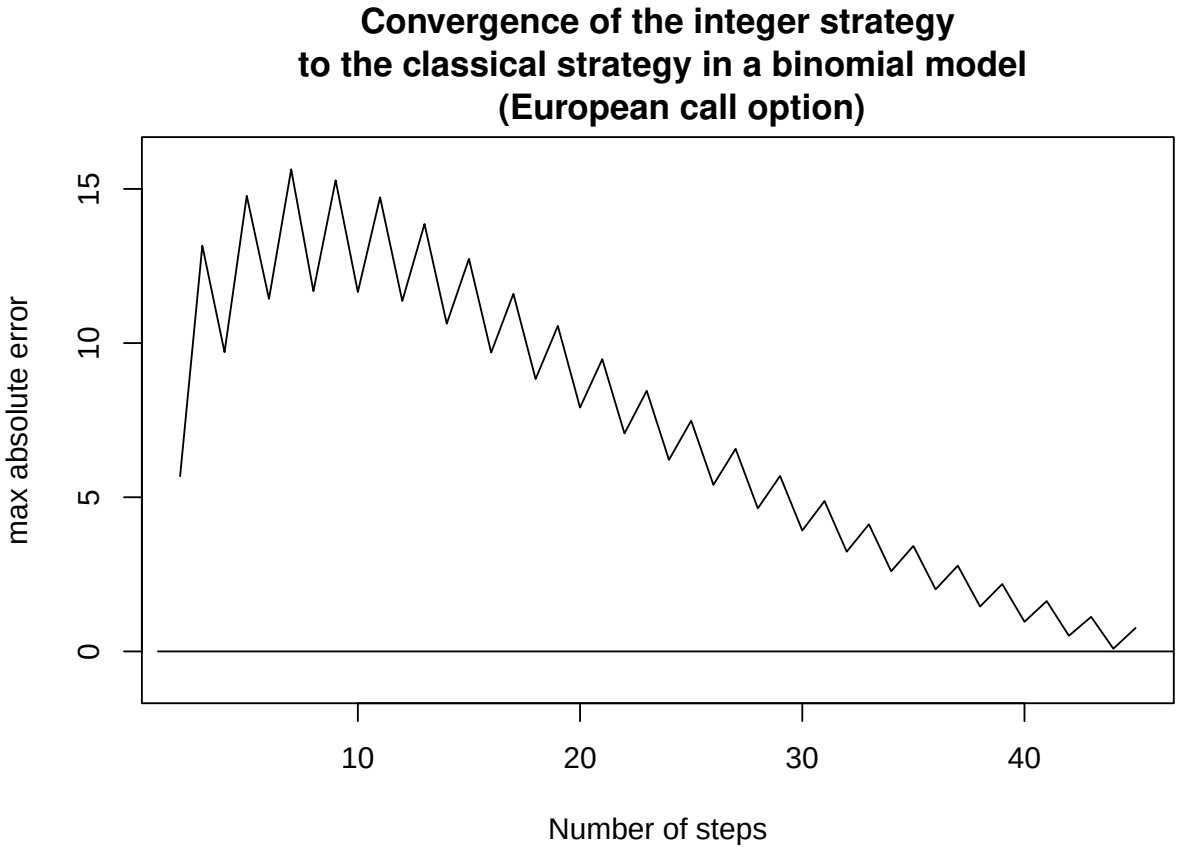

Figure 5.1: Maximal error E_N for increasing number N of timesteps

closely approximate zero).

From the above results we see an interesting tendency for the hedging error in a binomial model. In a next step we will extend the analysis to different claims and also vary the parameters to see if this form of limiting behavior can be observed for different scenarios. The most obviuos choice to extend the analyses is to consider a European put option instead of a call option. However in our simulations it turned out that the behaviour of the error is exactly the same as the one for the call option, thus we will not present these results here. Instead of that we will analyze the behavior when we change the parameters σ and K.

Figure 5.2 shows the results for this analysis. We observe a very interesting behavior of the error E_N. First of all there seems to be a convergence, i.e. it seems that the error decreases in all cases. However for a higher choice of parameters of σ the decrease is slower than for lower values. A similar behavior is obtained when increasing K. In fact for $\sigma = 0.4$ we observe a decreasing behavior but still have relative high values of E_N in comparison to other cases. When choosing $\sigma = 0.15$ we have a different picture. The error decreases relatively fast and reaches almost zero. However after a number of timesteps ($N > 30$) we observe a strange behavior with high volatility in the error.

In a final analysis we will consider a digital call option, i.e. an option with payoff function given as $\mathbb{1}_{\{S_T \geq K\}}$. The results of this hedging error with various parameter choices of σ and K are presented in figure 5.3. In the case of the digital option we achieve convergence, i.e. the hedging error decreases and approaches 0 for the number of timeperiods presented. We observe a similar behavior as beforehand, where the speed of decrease is again dependent on the choice of parameters; for a smaller value of σ and K the convergence to 0 takes fewer timesteps, whereas for higher values it takes longer.

The results of the analysis show an interesting behavior for the hedging error of the integer hedging strategy, indicating that there could exist a limiting property in a binomial model. Nevertheless one has to be careful as we also have seen that for the call/put option the performance of the error shows different behavior for small values of σ. In any case the picture for the digital call option strongly indicates that the error converges to 0.

Of course it would be of interest to analyze the behavior in more detail, and especially to give a sound mathematical analysis on the behavior of the integer hedging strategy in the binomial model. However in this thesis we refrain from doing so and only provide this numerical analysis, which gives a nice overview of the integer strategy of the most classical options in the very popular binomial model.

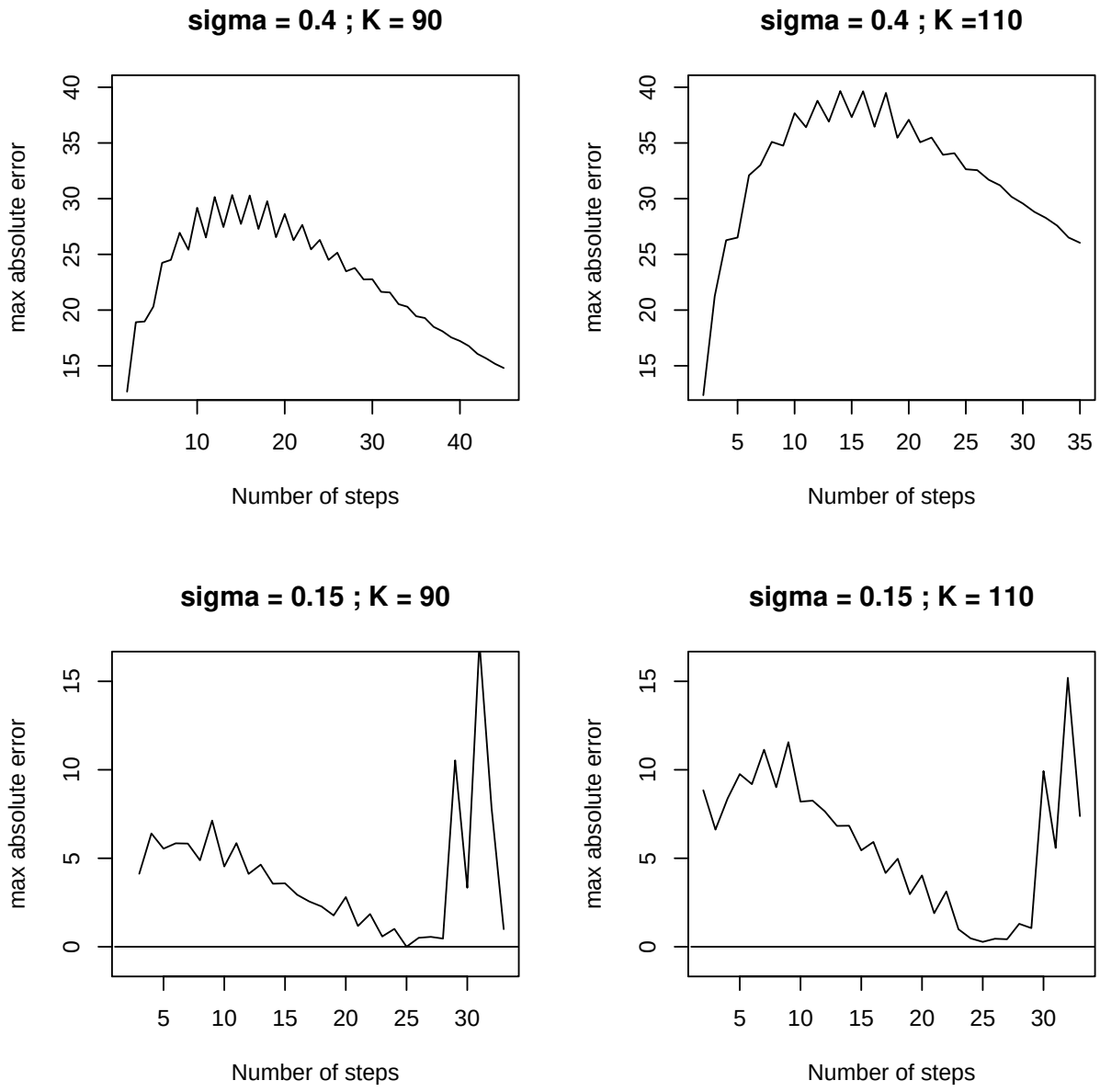

Figure 5.2: Maximal error E_N for increasing number N of timesteps and varying parameters σ and K (Put/Call option)

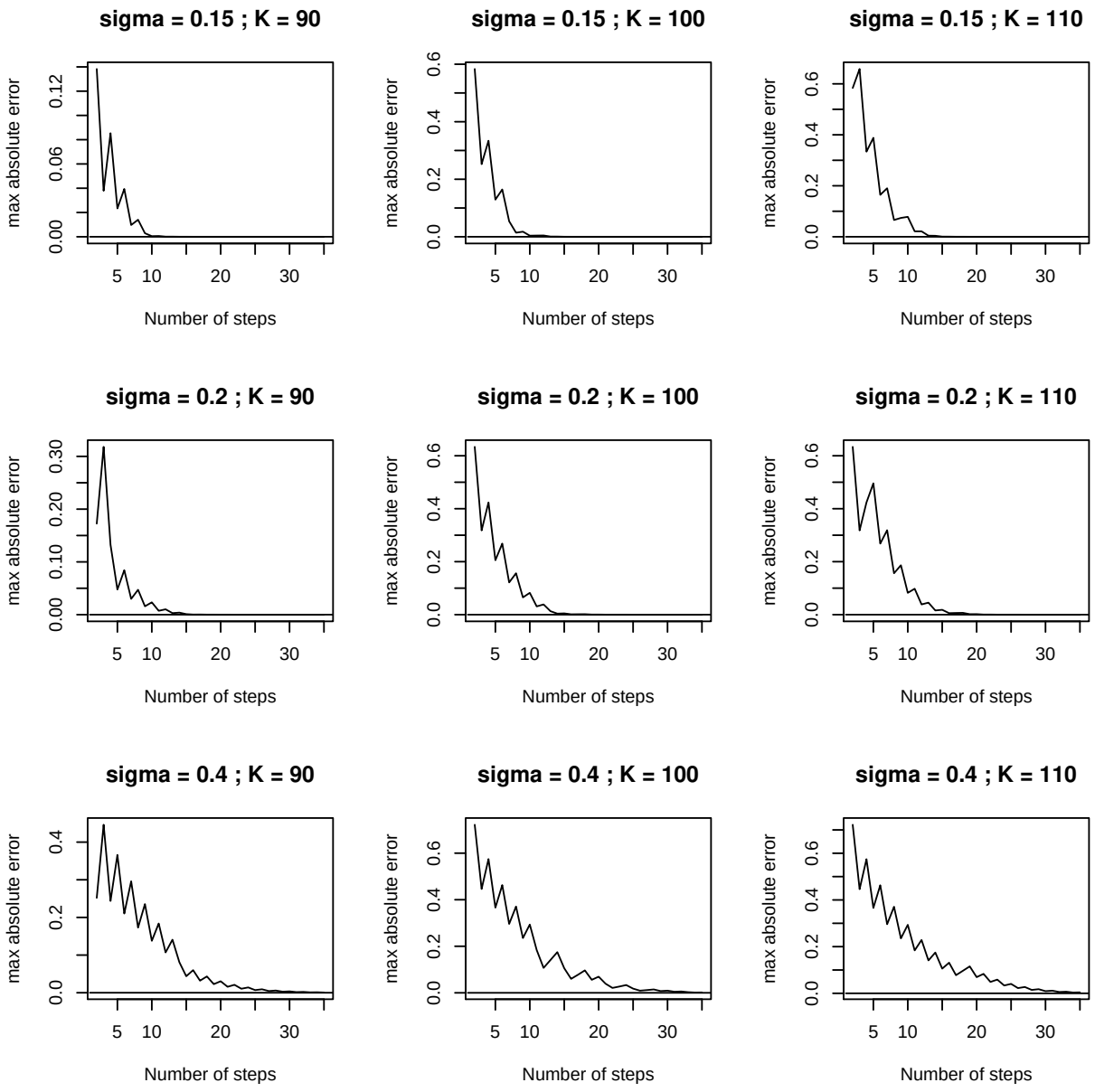

Figure 5.3: Maximal error E_N for increasing number N of timesteps and varying parameters σ and K (Digital Call option)

Chapter 6

Conclusion

In the context of financial mathematics the market theories and models often underly a variety of different assumptions that might be inadequate in the real world. Such violations of the assumptions or market fricitons may have a big impact on practitioners and thus cannot be neglected. One of these assumptions, that is basically always violated in the real world is that under the majority of market models one can buy any kind of position size, i.e. every share of assets in a portfolio can be described by a real number. However, in real life trading it is often only possible or at least more reasonable to buy round lots or integer amounts of an asset.

The goal of this thesis was to consinder these assumption where it is only possible to buy integer amounts of assets and to see how this would affect hedging strategies and optimal portfolio choices. To this end, we developed a framework which builds on the usual market assumption but also takes into account the integer restriction for the portfolios. To solve a kind of integer constraints problem for varous different models it was necessary to consider ideas from integer programming and lattice theory. We developed two methods to calculate the optimal hedging strategy for different models, a Branch and Bound (BB) approach and a Closest Vector problem (CVP) approach, providing the algorithm for it as well as the code written in the statistical programming language `R` (R Core Team [28]).

One of the ideas of the thesis was to look at practical applications for the two developed procedures of finding an optimal hedging strategy for the case of integer restricitions. Thus we looked at popular models used in practise and also variations of them. We began by inspecting the classical binomial model and then also the more complex trinomial model. The outcome of our developed approaches for these two models were not particularly spectacular. It turned out that the optimal integer strategy derived by our approaches does not differ from the one that can be deduced by simply rounding the optimal strategy of the classical case, i.e. the case without integer constraints. However when extending the trinomial model to models

with more than one underlying asset, we showed that the the developed algorithm performs far better than by simply rounding the classical solution in terms of the hedging error. It turned out that the CVP approach is even performing a bit better then the BB approach, where the performance of the algorithm does not differ very much considering runtime.

Finally in the last chapter of the thesis we analyzed integer strategies in the binomial in more detail. The idea was to see if a limiting behaviour for the integer hedging strategy can be observed when increasing the timesteps of the binomial model. Interestingly, in a numerical exercise, we indeed found that the hedging error decreased and converges to zero for the majority of our applications, indicating that we can achieve some kind of completeness in the limit. However it is necessary to be careful before hastily coming to conclusions as we have seen that the behavior of the analysed hedging error can vary depending on the parameter choice of the model.

This thesis adresses some of the problems that integer constraints impose on the optimal strategy with respect to hedging. However there are still a lot of possible extensions that can be done in this direction. One could be concerned with the optimization of the algorithm presented to solve the integer constraints problem for various models. As far as we went in this work, the algorithms used to solve the integer constraints are not yet fully elaborate and can be very slow for too high number of timesteps N. Another very interesting research could go into a deeper and more mathematical elaboration on chapter 5. The convergence analysis in this thesis is done by performing a numerical exercise, but a more mathematical approach with some convergence proof could be very interesting. Furthermore the research work on integer problems in mathematical finance is still not very profound and thus there is a lot of tasks in this field that are yet to tackle.

Appendix A

Appendix

A.1 Codes

For this work all the computations are done within the statistical programming language `R` (R Core Team [28]). The codes provided here are selfwritten algorithms in `R`, which follow the ideas mentioned in the different chapters of the thesis. To make the code work, the only requirement besides the base packages of `R` is the package `data.table` (Dowle and Srinivasan [13]). Apart from that all the functions needed are provided here. The functions are categorized in the order of appearance in the thesis, i.e. first we present the Branch and Bound algorithm used, secondly we provide code for the closest vector algorithm, which are both described in chapter 3. After that we procede with the code used for the applications to the Binomial model and the trinomial model from chapter 4. Finally further necessary functions are provided, so that the code can be run properly.

A.1.1 Code for the BB algorithm

In the following we present the code for the Branch and Bound algorithm as described in section 3.1.3 of the thesis. For the actual computations in chapter 4 there are 3 different versions of this algorithm, since we considered different models. However, since the basic structure of all 3 BB functions is the same and the difference is only in the arguments passed to the function, we only present one version of the algorithm.

```
BB_algorithm_1 <- function(optim_func,asset_paths,option_paths,delta_Asset,
                           N,x_start,phi_init,n_iter = F,keep_track = T){
  #### important: optimfunct need to have the following arguments:
  #### phi,asset_paths,delta_Asset,option_paths,N,restr = NULL,restr_vals = NULL !!!!!
```

```
#### initialize solution
x_hat <- x_start
opt_val <- optim_func(x_hat,asset_paths,delta_Asset,option_paths,N)

phi_opt <- optim(phi_init, optim_func,asset_paths = asset_paths,
                 delta_Asset = delta_Asset,option_paths = option_paths,N = N,
                 method = "BFGS",
                 control = list(factr = 10^(-11)))

relaxed_sol <- ifelse(abs(phi_opt$par-1) < 10^-6,1,
                      ifelse(abs(phi_opt$par-0) < 10^-4,0,phi_opt$par))

problems <- which(abs(relaxed_sol) %% 1 != 0)

#### select a node to branch on, first at random, but look at branching methods

branch_node <- sample(problems,1)

layer <- 1

open_nodes <- list()
first_node <- list(branch_node,c(floor(relaxed_sol[branch_node]),
                                 ceiling(relaxed_sol[branch_node])))

open_nodes[[1]] <- first_node
if(n_iter){
  iter <- 1
}

#set.seed(100)
while(length(open_nodes) != 0){
  if(keep_track){
    cat(paste0(" ",layer))
  }
  subprob <- open_nodes[[layer]][[2]][1]
```

```
    node <- open_nodes[[layer]]
    restr <- sapply(open_nodes,function(n)n[[1]])
    restr_vals <- sapply(open_nodes,function(n)n[[2]][1])
    new_opt <- optim(phi_init, optim_func,asset_paths = asset_paths,
                     delta_Asset = delta_Asset,option_paths = option_paths,N = N,
                     restr = restr, restr_vals = restr_vals,
                     method = "BFGS",control = list(factr = 10^(-11)))
    new_relaxed_sols <- ifelse(abs(new_opt$par-1) < 10^-4,1,
                               ifelse(abs(new_opt$par-0) < 10^-4,0,new_opt$par))
    new_relaxed_sols[restr] <- restr_vals
    new_problems <- which(abs(new_relaxed_sols %% 1 - 0) > 10^-10)
    if(length(new_problems) == 0){
      if(new_opt$value < (opt_val-10^-10)){#### if true: interger sol is better, adjust
        opt_val <- new_opt$value
        x_hat <- new_relaxed_sols
        if(keep_track){
          cat(" inside the if ")
        }
      }
      node[[2]] <- node[[2]][-(which(node[[2]] == subprob))]
      open_nodes[[layer]] <- node
    }else{
      if(new_opt$value < (opt_val-10^-10)){ #### careful about tolerance here!!
        #cat(" how often here? ")
        layer = layer+1
        new_branch_node <- ifelse(length(new_problems) == 1,
                                  new_problems,sample(new_problems,1))
        new_node <- list(new_branch_node,c(floor(new_relaxed_sols[new_branch_node]),
                                           ceiling(new_relaxed_sols[new_branch_node])))
        open_nodes[[layer]] <- new_node
      }else{
        node[[2]] <- node[[2]][-(which(node[[2]] == subprob))]
        open_nodes[[layer]] <- node
      }
    }
```

```
    #cat(paste0(" lnew: ",layer," &&"))
    while(length(open_nodes[[layer]][[2]]) == 0 && layer > 1){
      open_nodes <- open_nodes[-layer]
      if(layer > 1){
        layer = layer-1
        open_nodes[[layer]][[2]] <- open_nodes[[layer]][[2]][-1]
      }
    }
    if(layer == 1 && length(open_nodes[[layer]][[2]]) == 0){
      open_nodes <- open_nodes[-layer]
    }
    if(n_iter){
      iter <- iter+1
    }
  }

  return(list("par" = x_hat,"value" = opt_val))
}
```

A.1.2 Code for the CVP algorithm

We present the CVP algorithm used as described in section 3.2.2 of the thesis. As before there are two versions of the `CVP_algo` functions, but again the main difference is only the input arguments provided and thus we only present one version.

First we provide the `decode` algorithm which mainly follows the idea of the Agrell et al. [2]:

```
decode <- function(x,H,n_iter){
  n = dim(H)[1]
  bestdist = 10^10
  k = n
  dist = rep(0,n)
  e = matrix(0,nrow = n, ncol = n)
  e[k,] = x%*%H
  u = rep(0,n)
```

```
u[k] = round(e[k,k])
y = (e[k,k]-u[k])/H[k,k]
sgn <- function(x){
  1*(x>0)-1*(x <= 0)
}
step = c(rep(0,n-1),sgn(y))
u_hat = 0
runs = 0
while(runs < n_iter){######### this line needs adjustment, maybe insert another break
  #cat(paste0(" ", k))
  newdist = dist[k] + y^2
  if(newdist < bestdist){
    if(k != 1){
      e[k-1,] = e[k,]-y*H[k,]
      k = k-1
      dist[k] = newdist
      u[k] = round(e[k,k])
      y = (e[k,k]-u[k])/H[k,k]
      step[k] = sgn(y)
    }else{
      u_hat = u
      bestdist = newdist
      k = k+1
      u[k] = u[k]+step[k]
      y = (e[k,k]-u[k])/H[k,k]
      step[k] = -step[k]-sgn(step[k])
    }
  }else{
    if(k == n){
      print(paste0("algo breaks after ", runs, " steps"))
      break
    }else{
      k = k+1
      u[k] = u[k]+step[k]
      y = (e[k,k]-u[k])/H[k,k]
```

```
        step[k] = -step[k]-sgn(step[k])
      }
    }
    runs = runs+1
  }
  return(list(u_hat,k))
}
```

The function for the CVP algorithm is given as:

```
CVP_algo_1 <- function(lattice,probs,option_paths){
  mat_used <- lattice*probs^(1/2)

  option_paths <- as.data.frame(option_paths)
  C_tilda <- option_paths[,(N+1)]-option_paths[,1]

  dec <- qr(mat_used)
  Q <- qr.Q(dec)
  R <- qr.R(dec)

  x_1 = C_tilda*probs^(1/2)
  x = x_1%*%Q

  H = solve(t(R))

  val <- decode(x,H)
  val
}
```

A.1.3 Code for application to binomial model

In this section we will provide code used to calculate the hedging strategies for the binomial model. To come up with the final solution of the integer hedging problem under the two different approaches there are various functions that have to be defined.

- Creating the asset paths in the binomial model

```
CRR_paths <- function(N,U,D,S0){
  CRR_grid <- expand.grid(lapply(1:N,function(i)c(U,D)))
  CRR_grid <- CRR_grid[N:1]
  paths <- S0*t(apply(CRR_grid,1,function(x)sapply(1:N,function(i)prod(x[1:i]))))
  cbind(rep(S0,2^N),paths)
}
```

- Calculating the prices of an option

```
Prices <- function(N,K,S_n,r,p,opt){
  ### opt needs to be a function for the payoff of an option
  disc_opt <- opt(N,S_n,K)*exp(-r*N)

  prices <- list()
  prices[[N+1]] = disc_opt
  for(i in N:1){
    prices[[i]] <- prices[[i+1]][-1]*(1-p)+p*prices[[i+1]][-length(prices[[i+1]])]
  }
  prices
}
### example for opt function
E_call <- function(N,S_n,K){
  pmax(S_n[[N+1]]-K,0)
}
```

- Calculating option price paths

```
CRR_option_paths <- function(N,asset_paths,help_fun){
  option_paths <- as.data.table(t(apply(asset_paths,1,function(x){
    c(x[1:(length(x)-1)],max(x[length(x)]-K,0)*exp(-r*N))})))
  names(option_paths) = paste0(rep("col",N+1),as.character(1:(N+1)))

  for(i in N:1){
    sup <- option_paths[,lapply(.SD, help_fun),
```

```
                          by = eval(paste0("col",i)),.SDcols = eval(paste0("col",i+1)))
    option_paths <- merge(option_paths,sup,by = paste0("col",i),sort = F)
    option_paths <- option_paths[,-1]
    if(i == 1){
      setcolorder(option_paths,c(N+1,i:N))
    }else{
      setcolorder(option_paths,c(1:(i-1),N+1,i:N))
    }
    names(option_paths) = paste0(rep("col",N+1),as.character(1:(N+1)))
  }

  option_paths
}

help_fun_call <- function(x){ #### for Call option
  max(x)*p+min(x)*(1-p)
}

#### if not all values needed (often the case!)

CRR_option_paths_2 <- function(asset_paths,opt_prices,N){
  option_paths_2 <- as.data.frame(matrix(0,nrow = 2^N,ncol = N+1))
  swap_vals <- unique(asset_paths[,N+1])
  for(k in 1:length(swap_vals)){
    option_paths_2[asset_paths == swap_vals[k]] <- opt_prices[[N+1]][k]
  }
  option_paths_2[,1] <- opt_prices[[1]]
  option_paths_2
}
```

- Calculating the replicating strategy

```
replicating_strat <- function(N,K,S,r,p,prices){
  strat <- list()
  for(i in N:1){
    strat[[i]] <- exp(r*i)*
```

```
      (prices[[i+1]][-length(prices[[i+1]])]-prices[[i+1]][-1])/
      (S[[i+1]][-length(prices[[i+1]])]-S[[i+1]][-1])
  }
  strat
}
```

A.1.4 Code for application to trinomial model

Similar to the previous case of the binomial model there are various functions needed for the trinomial model.

- Creating the asset paths in the trinomial model

```
Multinom_paths <- function(N,U,D,S0,L){
  grid_vec <- c()
  for(l in 1:L){
    grid_vec <- c(grid_vec,U^(L-l)*D^(l-1))
  }
  grid <- expand.grid(lapply(1:N,function(i)grid_vec))
  grid <- grid[N:1]
  paths <- S0*t(apply(grid,1,function(x)sapply(1:N,function(i)prod(x[1:i]))))
  cbind(rep(S0,L^N),paths)
}
```

- Calculating the prices of an option in the trinomial model

```
Prices_tm <- function(N,K,S_n,r,p,opt){

  disc_opt <- opt(N,S_n,K)/(1+r)^N
  #disc_opt <- opt(N,S_n,K)*exp(-r*N)

  prices <- list()
  prices[[N+1]] = disc_opt
  for(i in N:1){
    ind <- length(prices[[i+1]])
```

```
    prices[[i]] <- prices[[i+1]][-c(ind-1,ind)]*p[1]+
      p[2]*prices[[i+1]][-c(1,ind)]+
      prices[[i+1]][-c(1,2)]*p[3]
  }
  prices
}
```

- Calculating option price paths in the trinomial model

```
Multinom_option_paths <- function(asset_paths,opt_prices,N,L){ #only price end end
  option_paths_2 <- as.data.frame(matrix(0,nrow = L^N,ncol = N+1))
  swap_vals <- unique(signif(asset_paths[,N+1],7))
  for(k in 1:length(swap_vals)){
    option_paths_2[signif(asset_paths,7) == swap_vals[k]] <- opt_prices[[N+1]][k]
  }
  option_paths_2[,1] <- opt_prices[[1]]
  option_paths_2
}
```

- Calculating the minimum variance strategy via dynammic programming approach:

```
dyn_optim_tm <- function(phi,opt_prices,S_N,K,r,p,time){
  l <- length(S_N[[time]])
  disc_price_1 <- S_N[[time+1]]/(1+r)^(time)
  disc_price_2 <- S_N[[time]]/(1+r)^(time-1)
  p[1]*sum((opt_prices[[time+1]][1:l] -
              phi*(disc_price_1[1:l]-disc_price_2)-opt_prices[[time]])^2)+
    p[2]*sum((opt_prices[[time+1]][2:(l+1)] -
                phi*(disc_price_1[2:(l+1)]-disc_price_2)-opt_prices[[time]])^2)+
    p[3]*sum((opt_prices[[time+1]][3:(l+2)] -
                phi*(disc_price_1[3:(l+2)]-disc_price_2)-opt_prices[[time]])^2)
}

#### procedure for derivation below
```

```
#opt_strat_tm <- list()
#for(i in N:1){
#  phi_init <- rep(1,(i-1)*(L-1)+1)
#  opt_strat_tm[[i]] <- optim(par = phi_init,fn = dyn_optim_tm,
#                             opt_prices = call_prices_tm,
#                             S_N = S_N_tm, K = K, r = r,p = ps,time = i,
#                             method = "BFGS",
#                             control = list(reltol = 10^(-10)))$par
#}
```

A.1.5 Further necessary Code

For the BB algorithm it is necessary to provide an optimization function, thus it is necessary to write a proper one for the different integer hedging problems of chapter 4.

- Optimization function for the binomial model

```
optim_func <- function(phi,asset_paths,delta_Asset,option_paths,N,
                       restr = NULL,restr_vals = NULL){
  option_paths <- as.data.frame(option_paths)
  temp <- phi
  temp[restr] <- restr_vals #### tester!
  phi_list <- list()
  count = 1
  for(i in 1:N){
    phi_list[[i]] <- temp[1:i]
    temp <- temp[-(1:i)]
  }
  phi_mat <- asset_paths[,1:N]
  for(j in 1:N){
    swap_vals <- unique(asset_paths[,j])
    for(k in 1:length(swap_vals)){
      phi_mat[phi_mat == swap_vals[k]] <- phi_list[[j]][k]
    }
  }
  sum(probs*((option_paths[,(N+1)]-option_paths[,1])-
```

```
                rowSums(as.matrix(delta_Asset)*phi_mat))^2)
}
```

- Optimization function for the trinomial model

```
optim_func_Multinom <- function(phi,asset_paths,delta_Asset,option_paths,N,
                                probs = probs_tm,restr = NULL,restr_vals = NULL){
  option_paths <- as.data.frame(option_paths)
  temp <- phi
  temp[restr] <- restr_vals #### tester!
  phi_list <- list()
  phi_list[[1]] <- temp[1]
  temp <- temp[-1]
  ind <- 1
  for(i in 1:(N-1)){
    ind <- ind+(L-1)
    phi_list[[i+1]] <- temp[1:ind]
    temp <- temp[-(1:ind)]
  }
  phi_mat <- signif(asset_paths[,1:N],7)
  for(j in 1:N){
    swap_vals <- unique(signif(asset_paths[,j],7))
    for(k in 1:length(swap_vals)){
      phi_mat[phi_mat == swap_vals[k]] <- phi_list[[j]][k]
    }
  }
  sum(probs*((option_paths[,(N+1)]-option_paths[,1])-
               rowSums(as.matrix(delta_Asset)*phi_mat))^2)
}
```

A.1.6 Code for the Chapter 5

In this subsection we present the code for the calculation and results of chapter 5. It is important to note, that the code depends to some extend on the code presented in the sections before. Furthermore we only present the code for the European call option as the code for the other options follow a similar logic.

```
conv_checker_2 <- function(N){

  U = exp(sigma*sqrt(Mat/N))
  D = exp(-sigma*sqrt(Mat/N))

  p = calc_MM(U,D,r)

  S_N <- S_n(N,U,D,S0)
  plot_CRR(S_N,N,U,D)

  E_call <- function(N,S_n,K){
    pmax(S_n[[N+1]]-K,0)
  }

  call_prices <- Prices(N,K,S_N,r,p,E_call)
  plot_CRR_options(S_N,N,U,D,call_prices)

  strat_list_Asset <- replicating_strat(N,K,S_N,r,p,call_prices)

  unique_strat_paths <- function(strat_list_Asset,N){
    #disc_asset_paths <- as.data.frame(sapply(0:N,function(x){as.data.table(asset_path
    rounded <- lapply(strat_list_Asset,round)
    uniques <- matrix(0,nrow = 1,ncol = N)
    uniques[1] <- rounded[[1]]
    for(j in 2:(N)){
      if(length(unique(rounded[[j]])) == 1){
        uniques[j] <- rounded[[j]][1]
      }else{
        if(nrow(uniques) == 1){
          add_row <- uniques
          uniques[j] <- unique(rounded[[j]])[1]
          uniques <- rbind(uniques,add_row)
        }else{
          n_j <- nrow(uniques)
          sel_0 <- which(rounded[[j-1]] == unique(rounded[[j-1]])[2])[1]
```

```
          sel_1 <- sel_0-1
          val_0 <- rounded[[j]][c(sel_0,sel_0+1)]
          val_1 <- rounded[[j]][c(sel_1,sel_1+1)]
          for(k in 1:n_j){
            w_val_k <- uniques[k,j-1] ### check if entry before (j-1) in row k is 1 or
            if(w_val_k == 1){
              if(length(unique(val_1))==1){
                uniques[k,j] <- val_1[1]
              }else{
                add_row <- uniques[k,]
                uniques[k,j] <- val_1[1]
                uniques <- rbind(uniques,add_row)
              }
            }else{
              if(length(unique(val_0)) == 1){
                uniques[k,j] <- val_0[1]
              }else{
                add_row <- uniques[k,]
                uniques[k,j] <- val_1[1]
                uniques <- rbind(uniques,add_row)
              }
            }
          }
        }
      }
    }
    uniques
}


unique_s_paths <- unique_strat_paths(strat_list_Asset,N)
rownames(unique_s_paths) <- 1:nrow(unique_s_paths)

for(i in N:1){
  ordered_paths <- unique_s_paths[order(-unique_s_paths[,i]),]
```

```
}

get_disc_asset_mat <- function(S_N,ordered_paths,N,strat_list_Asset){
  rounded <- lapply(strat_list_Asset,round)
  disc_S_N <- lapply(0:N,function(x){S_N[[x+1]]*exp(-r*x) })
  if(is.null(nrow(ordered_paths))){
    ordered_paths <- t(as.matrix(ordered_paths))
  }
  d_asset_mat <- ordered_paths
  d_asset_mat[,1] <- S_N[[1]]
  for(i in 2:N){
    col_p <- ordered_paths[,i]
    oz_vec <- unique(col_p)
    if(length(oz_vec) == 1){
      d_asset_mat[,i] <- disc_S_N[[i]][length(disc_S_N[[i]])]
    }else{
      sel_0 <- which(col_p == oz_vec[2])
      sel_1 <- which(col_p == oz_vec[1])
      rounded_0 <- which(rounded[[i]] == unique(rounded[[i]])[2])[1]
      rounded_0_val <- disc_S_N[[i]][rounded_0]
      help_r1 <- which(rounded[[i]] == unique(rounded[[i]])[1])
      rounded_1 <- help_r1[length(help_r1)]
      rounded_1_val <- disc_S_N[[i]][rounded_1]
      d_asset_mat[sel_0,i] <- rounded_0_val
      d_asset_mat[sel_1,i] <- rounded_1_val
    }
  }
  d_asset_mat
}


d_asset_ps <- get_disc_asset_mat(S_N,ordered_paths,N,strat_list_Asset)

if(is.null(nrow(ordered_paths))){
  ordered_paths <- t(as.matrix(ordered_paths))
```

```
  }
  BA_rounded_ps <- matrix(0,nrow = nrow(ordered_paths),ncol = N)
  BA_rounded_ps[,1] <- call_prices[[1]]-ordered_paths[,1]*d_asset_ps[,1]

  for(i in 2:N){
    BA_rounded_ps[,i] <- BA_rounded_ps[,i-1]+d_asset_ps[,i]*
      (ordered_paths[,i-1]-ordered_paths[,i])
  }

  disc_S_N <- lapply(0:N,function(x){S_N[[x+1]]*exp(-r*x) })
  rounded <- lapply(strat_list_Asset,round)
  p_val <- which(rounded[[N]] == 0)[1]
  opt_val_rel <- rep(call_prices[[N+1]][p_val])
  d_asset_rel <- rep(disc_S_N[[N+1]][p_val],nrow(ordered_paths))

  err2 <- opt_val_rel- (ordered_paths[,N]*d_asset_rel+BA_rounded_ps[,N])

  return(max(abs(err2)))
}
```

Appendix B

Bibliography

[1] Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2009.

[2] E. Agrell, T. Eriksson, A. Vardy, and K. Zeger. Closest point search in lattices. *IEEE Transactions on Information Theory*, 48(8):2201–2214, Aug 2002. ISSN 1557-9654. doi: 10.1109/TIT.2002.800499.

[3] D. Applegate, R. Bixby, V. Chvatal, and B. Cook. Finding cuts in the TSP (a preliminary report). Technical report, Center for Discrete Mathematics & Theoretical Computer Science, 1995.

[4] Anja Becker, Nicolas Gama, and Antoine Joux. Solving shortest and closest vector problems: The decomposition approach. *IACR Cryptol. ePrint Arch.*, 2013:685, 2013.

[5] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957. URL http://books.google.com/books?id=fyVtp3EMxasC&pg=PR5&dq=dynamic+programming+richard+e+bellman&client=firefox-a#v=onepage&q=dynamic%20programming%20richard%20e%20bellman&f=false.

[6] M. Benichou, J. M. Gauthier, P. Girodet, G. Hentges, G. Ribiere, and O. Vincent. Experiments in mixed-integer linear programming. *Mathematical Programming*, 1(1): 76–94, 1971. doi: 10.1007/BF01584074. URL https://doi.org/10.1007/BF01584074.

[7] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973. doi: 10.1086/260062. URL https://doi.org/10.1086/260062.

[8] P. Bonami and M. A. Lejeune. An exact solution approach for portfolio optimization problems under stochastic and integer constraints. *Operations Research*, 57(3):650–670, 2009. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/25614782.

[9] P. P. Boyle. Option valuation using a three jump process. *International Options Journal*, (3):7–12, 1986.

[10] Phelim P. Boyle. A lattice framework for option pricing with two state variables. 23 (1):1–12, 1988. doi: DOI:10.2307/2331019. URL https://www.cambridge.org/core/article/lattice-framework-for-option-pricing-with-two-state-variables/CEB5D729BBC4D7C7B3EE0479AAFF8817.

[11] Mark Breese. Integer arbitrage for the small player. Master's thesis, University of Liverpool, 2019.

[12] John C. Cox, Stephen A. Ross, and Mark Rubinstein. Option pricing: A simplified approach. *Journal of Financial Economics*, 7(3):229 – 263, 1979. ISSN 0304-405X. doi: https://doi.org/10.1016/0304-405X(79)90015-1. URL http://www.sciencedirect.com/science/article/pii/0304405X79900151.

[13] Matt Dowle and Arun Srinivasan. *data.table: Extension of 'data.frame'*, 2019. URL https://CRAN.R-project.org/package=data.table. R package version 1.12.8.

[14] Niklas Ekvall. A lattice approach for pricing of multivariate contingent claims. *European Journal of Operational Research*, 91(2):214 – 228, 1996. ISSN 0377-2217. doi: https://doi.org/10.1016/0377-2217(95)00279-0. URL http://www.sciencedirect.com/science/article/pii/0377221795002790.

[15] Sergei Fedotov and Sergei Mikhailov. Option pricing for incomplete markets via stochastic optimization: transaction costs, adaptive control and forecast. *International Journal of Theoretical and Applied Finance*, 4:179–195, 02 2001. doi: 10.1142/S0219024901000912.

[16] Hans Föllmer and Alexander Schied. *Stochastic Finance*. De Gruyter, Berlin, Boston, 2008. ISBN 978-3-11-021207-5. URL https://www.degruyter.com/view/title/17490.

[17] Stefan Gerhold and Paul Krühner. Dynamic trading under integer constraints. *Finance and Stochastics*, 22(4):919–957, Oct 2018. ISSN 1432-1122. doi: 10.1007/s00780-018-0369-3. URL https://doi.org/10.1007/s00780-018-0369-3.

[18] Ralph Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 09 1958. doi: 10.1090/S0002-9904-1958-10224-4.

[19] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Algorithms for the shortest and closest lattice vector problems. In Yeow Meng Chee, Zhenbo Guo, San Ling, Fengjing Shao, Yuansheng Tang, Huaxiong Wang, and Chaoping Xing, editors, *Coding and Cryptology*, pages 159–190, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-20901-7.

[20] A. Korkine and G. Zolotareff. Sur les formes quadratiques. *Mathematische Annalen*, 6(3):366–389, 1873. doi: 10.1007/BF01442795. URL https://doi.org/10.1007/BF01442795.

[21] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. 28(3):497–520, 2020/04/10/ 1960. doi: 10.2307/1910129. URL www.jstor.org/stable/1910129.

[22] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966. doi: 10.1287/opre.14.4.699. URL https://doi.org/10.1287/opre.14.4.699.

[23] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982. doi: 10.1007/BF01457454. URL https://doi.org/10.1007/BF01457454.

[24] J. T. Linderoth and M. W. P. Savelsbergh. A computational study of search strategies for mixed integer programming. *INFORMS Journal on Computing*, 11(2):173–187, 1999. doi: 10.1287/ijoc.11.2.173. URL https://doi.org/10.1287/ijoc.11.2.173.

[25] David Morrison, Sheldon Jacobson, Jason Sauppe, and Edward Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 02 2016. doi: 10.1016/j.disopt.2016.01.005.

[26] George Nemhauser and Laurence Wolsey. *The Scope of Integer and Combinatorial Optimization*, chapter I.1, pages 1–26. John Wiley & Sons, Ltd, 1988. ISBN 9781118627372. doi: 10.1002/9781118627372.ch1. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118627372.ch1.

[27] Andrea Pascucci and Wolfgang Runggaldier. *Financial Mathematics - Theory and Problems for Multi-period Models*. Springer-Verlag Mailand, 01 2012. ISBN 9788847025370. doi: 10.1007/978-88-470-1442-8.

[28] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017. URL https://www.R-project.org/.

[29] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathematical Programming*, 66(1):181–199, 1994. doi: 10.1007/BF01581144. URL https://doi.org/10.1007/BF01581144.

[30] RenéM. Stulz. Options on the minimum or the maximum of two risky assets: Analysis and applications. *Journal of Financial Economics*, 10(2):161 – 185, 1982. ISSN 0304-405X. doi: https://doi.org/10.1016/0304-405X(82)90011-3. URL http://www.sciencedirect.com/science/article/pii/0304405X82900113.

[31] Yuji Yamada and James A. Primbs. Construction of multinomial lattice random walks for optimal hedges. In Vassil N. Alexandrov, Jack J. Dongarra, Benjoe A. Juliano, René S. Renner, and C. J. Kenneth Tan, editors, *Computational Science — ICCS 2001*, pages 579–588, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-45545-5.

[32] Yuji Yamada and James A. Primbs. Properties of multinomial lattices with cumulants for option pricing and hedging. *Asia-Pacific Financial Markets*, 11(3):335–365, 2004. doi: 10.1007/s10690-005-9005-2. URL https://doi.org/10.1007/s10690-005-9005-2.