Master's Thesis

Optimal Order Placement when trading bitcoins at orderbook level, using Reinforcement Learning

Axel Perschmann

July 17, 2017

Albert-Ludwigs-University Freiburg im Breisgau
Faculty of Engineering
Department of Computer Science
Machine Learning Lab

Writing period

12. 01. 2016 – 31. 07. 2017

Examiners

Dr. Joschka Bödecker

Dr. Frank Hutter

External Adviser:

Manuel Blum, *Psiori GmbH*

Declaration

I hereby declare, that I am the sole author and composer of my Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Freiburg, July 17, 2017
Place, date

Axel Perschmann

Acknowledgements

I like to thank the following people for their help and support:

- Manuel Blum, who directed me and provided me with great feedback throughout the whole project.
- All employees of the data science company *PSIORI GmbH*, for the pleasant working atmosphere and their occasional feedback on my topic.

Abstract

English abstract

Zusammenfassung

Deutsche Zusammenfassung

Contents

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Objectives . . . 1
- 1.3 Related Work . . . 2
- 1.4 Contributions . . . 2
- 1.5 Outline of Contents . . . 2

2 Background 3
- 2.1 Trading Basics . . . 3
- 2.2 Bitcoin . . . 6
- 2.3 Supervised Learning . . . 8
- 2.4 Reinforcement Learning . . . 8

3 Orderbook Trading Simulator 15
- 3.1 Data Origin . . . 15
- 3.2 Data preprocessing . . . 16
- 3.3 Simulator . . . 17

4 Reinforcement Learning Approach 23
- 4.1 General Framework . . . 23
- 4.2 Backward learning . . . 25
- 4.3 Forward learning . . . 28
- 4.4 Discussion . . . 29

5 Orderbook Agents 33
- 5.1 Implementation . . . 33
- 5.2 Backward Sampling Experiments . . . 34
- 5.3 Forward Sampling Experiments . . . 42

6 Conclusion 45

A Appendix 46

B Glossary i

C References i

D List of Figures iii

E List of Tables iii

F List of Code iv

1 Introduction

1.1 Motivation

In the domain of computational finance, much research is performed to find and improve algorithms that help maximize revenue. One possibility to maximize revenue is to minimize inevitably occurring costs.

In the first place, investors participating in exchange markets, must expect fees, charged by the respective market place organizer in return for granting access to their infrastructure. Additionally, there are hidden costs to be considered as well. Most markets function after the microeconomic supply and demand [**todo**] model, where a universe of opposing trading interests determines the current price of a commodity.

While trades with little capital (relative to the whole market liquidity) usually cause minor impact on the current market situation, large-scale investors must be cautious when it comes to order placement. Large orders can have a major impact on supply and demand, which leads to diminishing availability, worsening prices and as such this so called slippage must be seen as hidden costs.

Well considered trading strategies help large-scale investors to reduce their impact and avoid costly market turbulences by unwinding large orders of shares over time. Nevmyvaka et al. [10] applied reinforcement learning to optimally distribute the trading activity over a fixed time horizon.

1.2 Objectives

This thesis tackles the important problem of optimized trade execution, which frequently occurs in the domain of financial computing. In its simplest form, the problem is defined by a particular financial instrument (here: bitcoins), which must be bought or sold within a fixed time horizon, while minimizing the expenditure (share price) for doing so.

The scope of this thesis is to transfer Nevmyvakas [10] reinforcement learning approach from traditional stock markets with expensive, proprietary data sources, to the relatively young market of bitcoin trading and to improve it's general ability to solve the important problem of optimized trade execution. In contrast to their experiments this thesis builds on inexpensive, publicly retrievable bitcoin exchange data. Snapshots of the current market situation are retrieved on a low-resolution, minute-scale basis from the open bitcoin exchange platform Poloniex. As such the usability of the retrieved dataset remains

to be shown.

Additional market features, describing the current market situation as well as historic market performance, are evaluated in terms of cost impact. An Orderbook Trading Simulator (OTS), which simulates the individual traders influence on the current market situation, is implemented and used in order to learn and evaluate various trading strategies.

1.3 Related Work

x

1.4 Contributions

- An orderbook trading simulator framework is presented which takes into account the individual traders influence on the current market situation.

1.5 Outline of Contents

The remainder of this thesis is structured as follows:
Section 2 gives a general introduction into the vocabulary of financial computing and the machine learning techniques employed, Section 3 describes the Orderbook Trading Simulator, developed within the scope of this thesis, and Section 4 covers the machine learning part. Section 6 closes with a conclusion and discussion.

2 Background

This chapter gives a general introduction into the vocabulary of financial computing and the machine learning techniques employed.

2.1 Trading Basics

In order to understand the objective of this thesis, fundamental knowledge about the domain of financial computing is obligatory. This chapter provides a quick overview.

2.1.1 Financial Instruments

Financial instruments represent legal agreements of monetary value between parties and can be traded as assets. These assets span from actual cash, through evidence of entity or share ownership, to contractual rights to receive or deliver another type of financial instrument.

2.1.2 Markets

Financial instruments are typically traded through one of two basic market types:

Over-The-Counter (OTC) In an OTC or off-exchange market, transactions take place directly between two parties, without a mediator. Prices are negotiated directly between buyer and seller and typically not published to the public.

Exchange In exchange markets, transactions are executed by a so called broker. A broker, which can be both, an individual or a firm, executes buy and sell orders on behalf of traders for a certain fee or commission. The respective prices are determined by the current market situation, in particular by supply and demand.

2.1.3 Exchange Markets

Specialized exchanges concentrate on certain sub-types of financial instruments and offer a trading venue for those willing to buy and sell these instruments. Some of them are listed below:

Stock Exchange Market A stock exchange or bourse provides companies access to investment capital in exchange to a share of ownership. Especially in times with notoriously low interest rates, investors tend to accept the greater risk of business development over a risk free, but faint investment, to grow their assets.
E. g. NASDAQ, Deutsche Börse, ...

Commodity exchange market Commodity exchange markets allow for speculations with goods like oil, gold, corn, ...
E. g. Eurex, ...

Foreign exchange market Foreign exchange (short: forex) is considered the largest financial market in the world. The forex market is responsible for determining currency exchange rates.
E. g. FXCM, ...

Digital assets exchange market Digital asset exchange markets allow for speculations with virtual goods, like digital documents, motion picture, audible content, or digital currencies (i. e. electronic money).
E. g. Poloniex, ...

In the past, exchanges were physical locations. In many cases, these physical locations have been replaced by fully electronically organized markets, accessible through the internet. As a result, more traders can execute orders in a higher frequency.

2.1.4 Ask and Bid

Most exchange markets function after the so called auction market model [5], where the exchange acts as a mediator between buyers and sellers to ensure fair trading. Here buyers can *bid* a price they are willing to pay for a certain number of shares and sellers can *ask* a price they are aiming to make with a number of shares. The highest of all bids is called the *bid price*, the lowest of all offers is called the *ask price*. Together they represent the current price at which an instrument is traded.

The difference between bid price and ask price is called *bid-ask spread*, often abbreviated to *spread*.

2.1.5 Limit Order Book and Market Depth

A limit orderbook reflects supply (asks) and demand (bids) for a particular financial instrument. It is usually maintained by the trading venue and lists the number of shares being bid or offered, organized by price levels in two opposing books. Incoming orders are constantly appended to this highly dynamic list, while a matching engine cautiously resolves any inconsistencies (i. e. overlaps) between asks and bids by mediating between the involved parties.

It is usually not before the matching engine has arranged an actual trade, that a trading venue claims a certain percentage of the turnover as a service fee. To encourage active market participation, the pure submission, revision and cancelation of orders is typically free of charge.

1 Acme Internet Widget Company

	Amount	Type	Volume	VolumeAcc	norm_Price
31.00	200.0	ask	6200.0	8425.0	1.074533
30.00	50.0	ask	1500.0	2225.0	1.039871
29.00	25.0	ask	725.0	725.0	1.005208
28.85	NaN	center	NaN	NaN	NaN
28.70	200.0	bid	5740.0	5740.0	0.994810
28.50	100.0	bid	2850.0	8590.0	0.987877
28.00	300.0	bid	8400.0	16990.0	0.970546

Table 1: Exemplary snapshot of a limit orderbook for stocks of AIWC[1]

Table 1 shows a limit orderbook snapshot up to a market depth of 3, as seen by market participants. Here Alice offers 25 shares per 29$, Bob and Cedar offer 20 and 30 shares respectively per 30$ and David offers 200 shares per 31$.

Based on their trading needs, traders can typically choose between multiple levels of real-time market data.

Level 1 Market Data Basic informations only:
Bid price + size, Ask price + size, Last price + size

Level 2 Market Data Additional access to the orderbook.
Usually data providers display the orderbook only up to a certain market depth m, i. e. the lowest m asks and the highest m bids.

Level 3 Market Data Full data access.
Typically only accessible for the market maker.

2.1.6 Slippage

Slippage is defined as the difference between expected and achieved price at which a trade is executed. Slippage may occur due to delayed trade execution. Especially during periods of high volatility, markets might change faster than the order takes to be executed. Slippage is also linked to the order size, as larger orders tend to *eat* into the opposing book and are fulfilled at successively worse price levels. Slippage can be both positive or negative, depending on the current market movements and must be taken into account by serious investors.

2.1.7 Order Types

Investors can execute orders of different types, of which the most common ones are described below:

Market Orders are the most simple form of orders. Here, the investor only specifies the number of shares he want's to buy/sell and the full order is executed immediately, at any price. Especially for large-scale traders or traders with level 1 data access only, these simple market orders are rather hazardous, since the achieved price can significantly differ from the expected price due to sparse supply and demand.

Limit Orders additionally feature a worst price, i. e. the highest price a buyer is willing to pay per share or respectively the lowest price a seller is willing to make per share. Limit orders are immediately placed into the orderbook and (partially) executed, once the matching engine finds a corresponding trade in the opposing book. Limit orders reduce the risk of slippage, but do not guarantee execution.

Hidden Orders are placed into the market makers internal orderbook, but not displayed to other market participants with level 2 market data access. They represent a simple solution to large-scale investors seeking anonymity in the market, aiming to obfuscate their trading intention from other market participants.

2.1.8 Trading strategies

An order placement typically originates from a carefully considered *trading strategy*. An *active* trading strategy buys and sells instruments frequently based on short-term price movements, whereas a *passive* trading strategy such as *Buy-And-Hold* believes in long-term price movements eventually outweighting any short-term fluctuations.

As the execution of trades typically implies trading costs and slippage, these have to be taken into account. Particularly active traders with a high order quantity and large-scale investors with high order volumes are concerned with this burden. The order type chosen has a major impact on speed of execution and slippage generated.

While *limit orders* reduce the risk of slippage, they do not guarantee full order execution. This leads to the important problem of *optimized trade execution*, which frequently occurs in the domain of financial computing. In its simplest form, the problem is defined by a particular financial instrument (here: Bitcoins), which must be bought or sold within a fixed time horizon, for the best achievable share price.

In [11] Nevmyvaka et al. introduce a Submit & Leave Strategy (S&L), which cleverly combines market and limit orders: After an initial limit order submission, the order is left on the market for a predefined time horizon, after which it's unexecuted part is transformed into a market order and thus executed completely. They later extended their strategy to a Submit & Revise Strategy (S&R) [10], where the order limit may be revised at discrete time steps, depending on trade progress and market changes.

2.2 Bitcoin

Bitcoin is a digital cryptocurrency, released in 2009 as open-source software under the name Satoshi Nakamoto [3]. Motivated in part by anger over the foregone financial crisis, the underlying peer-to-peer system[9], provides a decentralized payment system. To this point, electronic transactions always required involving a third party to validate

transactions, an inevitable necessity to forestall double-spending bitcoin tokens. Double-spending is a problem unique to digital currencies, as digital informations, in contrast to physical currencies, may be replicated arbitrarily. Rather than entrusting a central authority or central server with this validation task, a distributed public transaction log, known as the *block chain*[14] is maintained among all participants.

The block chain is a continuously growing list of confirmed and timestamped transaction records, called *blocks*. Every block includes a complex proof-of-work and a cryptographic hash of the previous block, making it extremely resistant against retrospective modifications. To motivate others to validate transactions, newly created bitcoins are rewarded to the first miner (or mining party), successfully serving the computationally demanding proof-of-work.

Through a manifold of legal and black markets, bitcoins may be exchanged for other currencies, products and services. In contrast to traditional exchanges, no regular trading hours exist. Transactions can be executed at all times, mostly unregulated and pseudonymous[2][7]. As a consequence, Bitcoin are particularly sensitive to extraordinary events or news, frequently causing immediate (panic) reactions. This effect is possibly leveraged by relatively low entry barriers, attracting amateur and professional traders likewise.

2.2.1 Volatility

Bitcoin is a highly volatile currency, capable of massive price changes towards both directions within short periods of time. A high volatility provides profitable opportunities, but comes at the price of higher risk. Due to a very high trading volume[3], limit orderbooks are typically more condensed around the current best price than limit orderbooks of less frequently traded stocks. As a consequence, less slippage is to be expected from eating into the orderbook, while more slippages originates from general price fluctuations.

Figure 1 compares volatilities of currency pairs USDT/BTC (blue line) and USD/Pound (red line).

2 Owners of bitcoin tokens are not explicitly identified, but bitcoin exchanges may collect personal data on their customers if required by law.

3 $\sim$ 40.000.000$ per 24h for currency pair BTC/USDT on Poloniex.
Source: https://coinmarketcap.com/exchanges/volume/24-hour (accessed 12-July-2017)

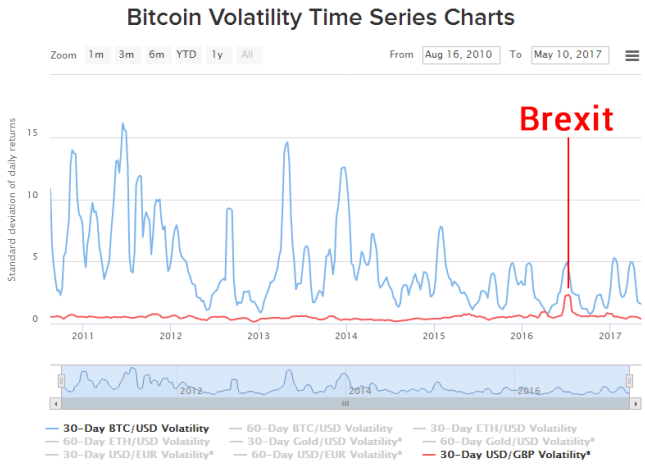


Figure 1: BTC/USDT and USDT/Pound volatility compared.
Source: https://99bitcoins.com/bitcoin-volatility-explained

2.3 Supervised Learning

Supervised learning is a subdomain of machine learning, where a function is learned from labeled training data $\{(x_1, y_1), ..., (x_N, y_N)\}$. Each training sample maps a feature vector $x_i \in X$ to a desired target value or label $y_i \in Y$. Target values may either be categorial, making the learning task a *classification* problem, or continuous, making the learning task a *regression* problem.

A supervised learning algorithms seeks to find a general function $g_\theta()$ (or it's parameters θ), such that $g_\theta(x_i) \approx y_i | i \leq N$. The learned function should ideally avoid overfitting by finding a generalization to previously unseen data.

Keep or Remove?! Or Move below Section 2.4.7

2.3.1 Random Forest

bla

2.4 Reinforcement Learning

Reinforcement Learning (RL) is a subdomain of machine learning. It was first defined by Sutton and Barto[15] as the problem of learning from interactions with an environment in order to achieve long-term goals. In contrast to Supervised Learning (SL), no labeled training data is present, such that sub-optimal actions can not be corrected explicitly. Instead, in each time step t, the agent takes the action a_t which maximizes[4] his future expected reward R_t, based on empirical knowledge.

4 Alternatively, it is possible to *minimize* the future expected cost: $C_t = -R_t$

Figure 2 shows the typical RL scenario. An agent (partially) observes a state s_t and a reward r_t at every discrete time step t. It takes an action a_t and hereon receives a new state s_{t+1} and reward r_{t+1}.

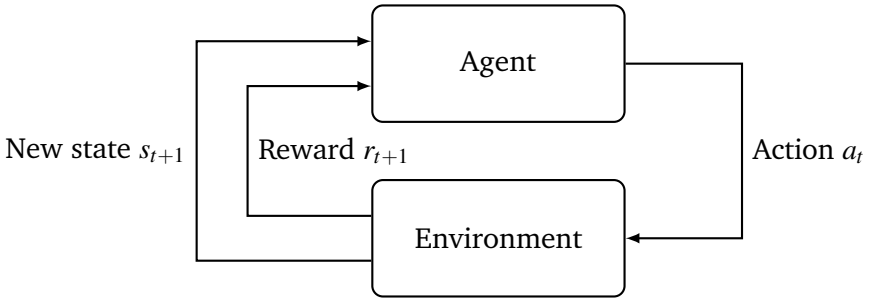


Figure 2: The discrete RL scenario defined by Sutton and Barto[15]

Rather than learning from labeled training data, the agent applies a *trial and error* pattern and exploits (delayed) external rewards to find actions, maximizing the expected future reward. As actions may not necessarily show an immediate effect, rewards are often delayed and must be accounted for in the so called *credit assignment problem*[8].

The expected future reward R_t can be expressed by the cumulative reward of all successive rewards. The discounting factor $\gamma \in [0,1]$ represents the importance between long-term and short-term rewards. With a high γ, the agent aims to maximize long-term rewards, whereas a small γ leads to greedy actions, favouring short-term rewards.

$$R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \tag{1}$$

2.4.1 Markov Decision Process

In reinforcement learning, the environment is typically formulated as Markov Decision Process (MDP) [1], which provides a mathematical framework for optimization problems.

An MDP is a discrete time stochastic control process, written as 5-tuple (S,A,P,R,γ). It consists of a (finite) set of environment states $s \in S$, a (finite) set of actions $a_t \in A$, a transition probability matrix $P : S \times A \to \mathbb{P}(S)$, which maps from state-action pairs (s_t, a_t) to a probability distribution over all successor states $s_{t+1} \in S$,

$$P^a_{ss'} = \mathbb{P}[s_{t+1} = s' | s_t = s, a_t = a] \tag{2}$$

and a reward (or cost) function $R : S \times A \to \mathbb{R}$, that computes the immediate reward received after applying action a_t in state s_t.

$$R^a_s = \mathbb{E}[r_{t+1} | s_t = s, a_t = a] = -\mathbb{E}[c_{t+1} | s_t = s, a_t = a] \tag{3}$$

The discount factor $\gamma \in [0,1]$ eventually defines the level of foresightedness. It is obligatory in MDPs with an infinite time horizon, to impede infinite values for accumulated long-term returns.

The general goal in reinforcement learning is to find optimal policies $\pi^* : S \to A$, that map states to the best applicable actions, such that the (discounted) expected long-term reward is maximized, or respectively the (discounted) expected long-term cost is minimized.

$$\pi^*(s) = arg \max_{a \in A(s)} \mathbb{E}[R_t | s_t = s] = arg \min_{a \in A(s)} \mathbb{E}[C_t | s_t = s] \tag{4}$$

2.4.2 Markov Property

An important condition of MDPs is the *Markov Property*, which defines the probability distribution of future states to be independent of the past (s_{t-n}), given the present (s_t). As such, states must capture all relevant information from the history, to deliver sufficient statistics of the future. A state s_t is *Markov* if and only if:

$$\mathbb{P}(S_{t+1} = s' | s_t, a_t) = \mathbb{P}(s_{t+1} = s' | s_t, a_t, s_{t-1}, a_{t-1}, \dots) \tag{5}$$

2.4.3 Value Function and Bellmann Equation

In order to find the optimal strategy π^*, the value of states and actions must be estimated. The state-value function $V^\pi : S \to \mathbb{R}$ returns the expected reward (or cost) when following π from state s:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}[R_t | s_t = s] \\ &= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma \sum_{k=1}^{\infty} \gamma^k r_{t+k+1} | s_t = s] \\ &= \sum_{s' \in S} P(s' \,|\, s, \pi(s)) \left[r(s, \pi(s), s') + \gamma V^\pi(s') \right] \end{aligned} \tag{6}$$

where $0 \leq \gamma \leq 1$ is the discount rate. Assuming an optimal policy π^*, the optimal value function may be rewritten as $V^{\pi^*} = V^*$ and Bellmann's *principle of optimality* holds:

> "*An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*"
>
> Richard Bellman, 1957 [2]

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s' \,|\, s, a) \left[r(s, a, s') + \gamma V^*(s') \right] \tag{7}$$

The *Bellman equation* shown in Equation (7) rewrites the value estimation problem as a recursive definition of the value function. This recursion breaks the complex optimization problem into simpler subproblems, that can be solved via *Dynamic Programming*[2].

2.4.4 Value Iteration

Bellmanns *value iteration algorithm* recursively applies the Bellman equation to an arbitrarily initialized value function, as shown in Algorithm 1. The algorithm repeatedly computes V^{k+1} for all states $s \in S$, until convergence: $\lim_{k\to\infty} V^k = V^*$. In practice, value iteration is stopped, once the Bellmann residual (i. e. the difference between both sides of the equation) falls below a predefined threshold θ.

Algorithm 1: Value Iteration.

Input: MDP, $\theta > 0$
Output: approximate of V^*
$V^0(s) \leftarrow$ arbitrary values
$k \leftarrow 0$
repeat
  $k \leftarrow k+1$
  foreach *state* $s \in S$ **do**
    $V^k(s) = \max_{a\in A} \sum_{s'\in S} P(s' \mid s,a)\,[r(s,a,s') + \gamma V^{k-1}(s')$
  end
until $\forall s \left|V^k(s) - V^{k-1}(s)\right| < \theta$

Once the optimal value function $V^*(s)$ is known, the optimal policy π^* can simply be extracted thereof:

$$\pi^*(s) = \arg\max_{a\in A} \sum_{s'\in S} P(s' \mid s,a)\,[r(s,a,s') + \gamma V^*(s')] \tag{8}$$

2.4.5 Q-learning

A more challenging problem arises, if transition probability matrix $P : S \times A \to \mathbb{P}(S)$ and reward function $R : S \times A \to \mathbb{R}$ are unknown a priori. In such case, the *model-free* reinforcement learning technique *Q-learning* [16] can be utilized to learn an action-value function Q, based on previously observed state transitions. The action value function

$Q^\pi : S \times A \to \mathbb{R}$ returns the expected reward (or cost) starting from state s, taking action a, and following π thereafter:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}[R_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma \sum_{k=1}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a] \\ &= \sum_{s' \in S} P(s' \,|\, s,a) \left[r(s,a,s') + \gamma Q^\pi(s', \pi(s')) \right] \end{aligned} \tag{9}$$

Similar to above, the optimal action-value function may be rewritten as $Q^{\pi^*} = Q^*$:

$$Q^*(s,a) = \sum_{s' \in S} P(s' \,|\, s,a) \left[r(s,a,s') + \gamma \max_{a' \in A} Q^*(s', a') \right] \tag{10}$$

The *Q-learning algorithm*, shown in Algorithm 2, repeatedly computes Q^{k+1} from a set of previously observed transition tuples $D = \{(s,a,r,s'), \dots | t = 1..n\}$, until convergence: $\lim_{k \to \infty} Q^k = Q^*$. Convergence is guaranteed if all state-actions pairs are visited an unbounded number of times and an appropriate learning rate is chosen [17].

Algorithm 2: Q-learning [16].

Input: D=$\{(s,a,r,s'), \dots\}$, $\alpha > 0$, $\theta > 0$
Output: approximate of Q^*
$Q^0(s,a) \leftarrow$ arbitrary values
$k \leftarrow 0$
repeat
  $k \leftarrow k+1$
  foreach *sample* $(s,a,r,s') \in D$ **do**
    $Q^k(s,a) = (1-\alpha) Q_{k-1}(s,a) + \alpha(r + \gamma \max_{a' \in A} Q^{k-1}(s', a'))]$
  end
until $\forall s \forall a \left| Q^k(s,a) - Q^{k-1}(s,a) \right| < \theta$

Once the optimal state-action function $Q^*(s,a)$ is known, the optimal policy π^* can simply be extracted thereof:

$$\pi^*(s) = \arg\max_{a \in A} Q^*(s,a) \tag{11}$$

2.4.6 Exploration

In environments with a large (or infinite) number of states, is is not feasible to visit all state-action pairs an unbounded number of times. Instead, the environment must be examined at the best possible rate. The agent has to carefully balance between *exploration* and *exploitation*, i. e. between testing out previously unknown paths and refining of promising actions. As the latter may get stuck in local optima, exploration helps to find

the global optimum.

This balancing act can be tackled by a so called ε-greedy strategy. With a probability of $p = \varepsilon \in [0,1]$ the agent takes a random action and with the remaining probability $q = 1 - \varepsilon$ the agent takes the presumably optimal action, as deduced from his current model. As the agent consolidates his model over time, ε may be slowly reduced.

2.4.7 Function Approximation

In small environments, V and Q-values are usually stored in a lookup table. For continuous or (infinitely) large environments function approximators are used to generalize the V and Q-functions to previously unseen states-action pairs. Besides scalability towards arbitrarily large environments, this may even speed up the learning process, as earlier experiences are generalized to previously unseen states.

More formally V and Q are replaced as follows:

$$V(s) = \hat{V}(s;\theta) \tag{12}$$

$$Q(s,a) = \hat{Q}(s,a;\theta) \tag{13}$$

where θ refers to function parameters, that must be learned through supervised learning methods.

2.4.8 Batch Reinforcement Learning

In the general reinforcement learning problem, the agent observes a state s_t, executes an action a_t and incrementally improves its policy π in conformity with the obtained reward r_t and successor state s'_t in an infinite loop. In *batch reinforcement learning* [6], an optimal strategy should be learned out of a fixed set of known transition samples $D = \{(s,a,r,s'), \ldots | t = 1..n\}$. As shown in Figure 3, the samples are collected a priori in an exploration phase, that is not part of the actual batch learning task.

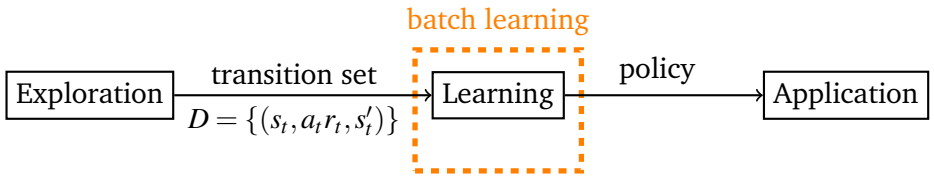


Figure 3: Batch Reinforcement Learning

Because exploration has an important impact on the achieved policy quality, more modern *growing batch reinforcement learning* approaches frequently alternate between exploration and learning phase as shown in Figure 4. In doing so, the agent gets a rough idea of a good policy in order to focus on *interesting* regions of the state space.

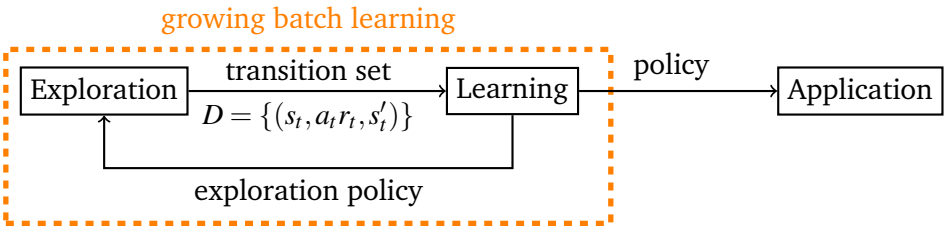


Figure 4: Growing Batch Reinforcement Learning

2.4.9 Tree-Based Batch Mode Reinforcement Learning

Randomized decision trees can be used to approximate the parameters of $Q(s,a;\theta)$, as proposed by Damien Ernst [4]. The batch reinforcement learning algorithm is shown in Algorithm 3.

Algorithm 3: Fitted Q Iteration [4].

Input: $D = \{(x_t, a_t, s_t, r_t), \ldots | t = 1..n\}$
$\hat{Q} \leftarrow 0$
$k \leftarrow 0$
repeat
  $k \leftarrow k+1$
  generate pattern set $P = \{(i^t, o^t) | t = 1,..,n\}$ where
    $i^t = (x^t, a^t), \qquad o^t = r_t + \gamma * \max_u \hat{Q}^k(s_t, u)$
  $\hat{Q}^{k+1} \leftarrow^{fit} P$
until *stopping condition reached*

3 Orderbook Trading Simulator

This chapter describes the Orderbook Trading Simulator (OTS) and it's underlying OrderbookContainers, implemented within the scope of this thesis. Fed with historic orderbook data it serves as a backtesting framework for testing out various trading strategies. The OTS provides detailed feedback in terms of trading progress, achieved prices and accrued costs.

3.1 Data Origin

Since typical financial data providers must make an earning from their treasures, they typically only deliver delayed market data on a complimentary basis. Investors dependent on real time or level 2 market data (see Section 2.1.5) are usually charged horrendous monthly subscription fees.

A costless alternative exists in open cryptocurrencies, like bitcoins (see Section 2.2). The digital asset exchange platform Poloniex [12] provides an open API for querying detailed market data in real time. As their push API, to receive live order book updates and trades, was rather error-prone and buggy when this project started, the decision was made, to query full orderbooks on a minutely basis.

On Nov, 10th 2016, 10:00 am, a daemon was started, to fetch orderbook snapshots up to a market depth of 5.000 from Poloniex via HTTP GET requests (see Listing 1). The volume of recorded orderbook snapshots for nine distinct currency pairs[5] has since grown to roughly 100GB (as per 2017-06-20). This thesis is based on a condensed version of the currency pair USDT/Bitcoin.

Listing 1: Data fetched from Poloniex via HTTP GET request

```
# https://poloniex.com/public?command=returnOrderBook&currencyPair=USDT_BTC&depth=5000
{"asks" :[[ "705.450000" ,2.772181], [ "705.450196", 0.139212] ,["706.170000" ↩
    ,0.052838] , ... ], "bids":[["705.000000",0.158232],["703.700000" ,0.001250], ↩
    ... ], "isFrozen": 0, "seq": 63413296}
```

Figure 5 shows the price evolution of currency pair USDT/Bitcoin over the period of recording. Prices have burst from roughly 700$ to more than 2.000$. Red and green lines depict the relative difference between the worst price paid/received for an immediate market order of $\pm$70.000$ respectively[6].

5 Recorded currency pairs include USDT/BTC, BTC/ETH, BT/XMR, BTC/XRP, BTC/FCT, BTC/NAV, BTC/-DASH, BTC/MAID, BTC/ZEC

6 With a total trading volume of roughly 40.000.000$ per day (see Section 2.2.1), these $\pm$70.000$ refer to roughly 4.2% thereof.

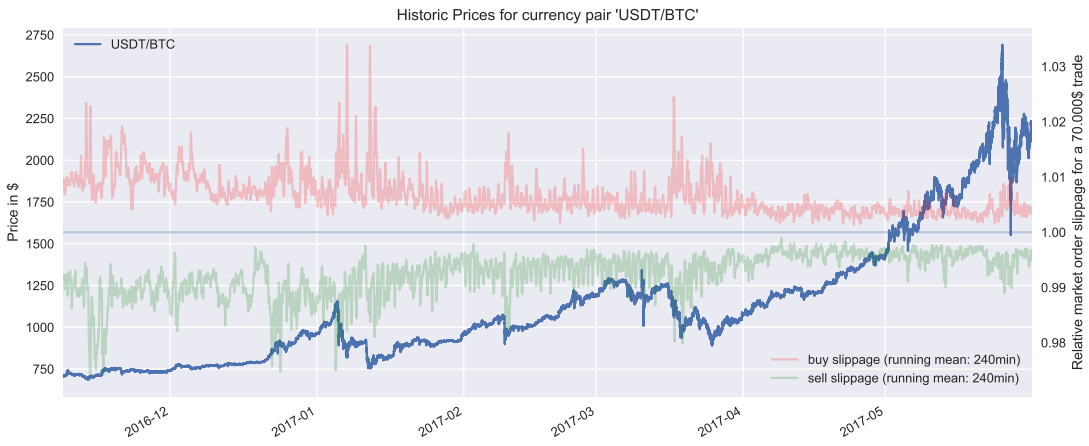


Figure 5: Historic center prices between Nov, 10th 2016 and May, 31 2017, as fetched from Poloniex.

3.2 Data preprocessing

The python `class OrderbookContainer` aggregates all informations contained in an individual orderbook snapshot. It enforces correct price ordering in the two opposing bid and ask books and provides additional methods for market visualization and feature extraction. To restrict wasteful memory usage, orderbook snapshots are condensed in several ways:

- Almost identically price levels are round to the second decimal and their respective order volumes merged.

$$\left.\begin{aligned} 0.139212 * 705.450000 \\ 2.632969 * 705.450196 \end{aligned}\right\} = 2.772181 * 705.45$$

- Market depth is capped just above the threshold of 100 bitcoins, roughly corresponding to a market depth of 100-140 prices levels in both books. This threshold allows to simulate trades up to a market order price of 70.000$ at any time throughout the whole recording period.
- Erroneous orderbook snapshots have been discarded. Occasional errors may occur, due to Poloniex API failures.

These measurements reduce the average individual orderbook snapshots size from 30KB to approximately 6.6KB. As for the december 2016, this results into 44.640 snapshots with a total size of 295MB instead of 1.35GB, clearly reducing the memory consumption.

Listing 2 shows the most important functions, provided by the OrderbookContainer class. OrderbookContainer instances are vigorously used by the OTS. Figure 6 shows a plain visualization of an individual orderbook snapshot.

Listing 2: OrderbookContainer

```
ob = OrderbookContainer(timestamp="2016-11-08T10:00",
                        bids=pd.DataFrame([200., 100., 300.],
                        columns=['Amount'], index=[28.7, 28.5, 28]),
                        asks=pd.DataFrame([25., 50., 200.],
                        columns=['Amount'], index=[29., 30., 31.]))
# Available methods
ob.plot(outfile='sample.pdf')  # plt.show or plt.savefig
ob.asks  # pd.DataFrame
ob.bids  # pd.DataFrame
ob.features  # returns a dict of precomputed features
ob.get_bid(), ob.get_ask(), ob.get_center()  # float
ob.get_current_price(volume=100)  # achievable cashflow by market order
ob.get_current_sharecount(cash=70000) # number of shares aquirable by market order
ob.compare_with(other_ob) # returns orderbook deltas used by the OTS
ob.enrich()  # computes Volume, VolumeAcc and norm_Price
ob.head(depth=3)  # returns the orderbook, capped at a market depth of 3
ob.plot()
```

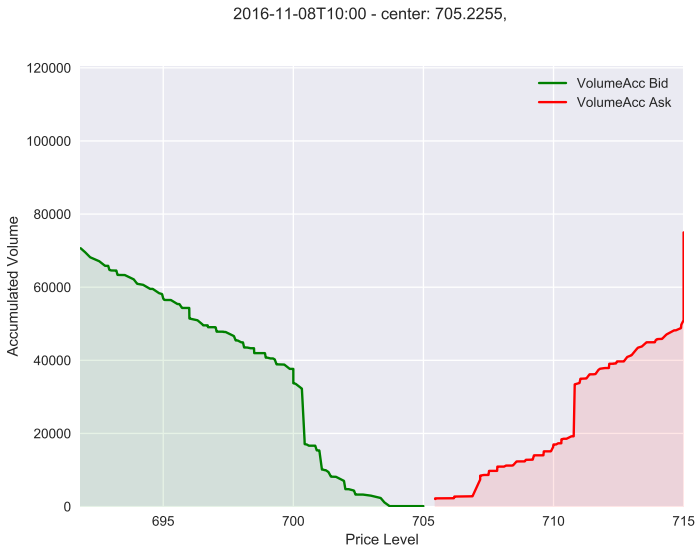


Figure 6: A simple visualization of an limit orderbook.

3.3 Simulator

The OTS framework serves as base for all preceding experiments and evaluations. Each simulator instance is fed with an array of subsequent OrderbookContainers (*orderbook windows*) and a targeted trading volume V, which it pretends to trade into cash or vice versa within a fixed time horizon H, according to an external strategy.

In the rare case of missing orderbook snapshots (see Section 3.2), the *real* time horizon may be larger than usual, since always H subsequent orderbooks are selected. By default, the OTS refuses to work with orderbook windows, whose actual length differs from the

presumed length by more than two minutes.

Limit orders may be placed at predefined, discrete time steps within the trading horizon H. This is done though the simulators main interface method `trade(limit=...)`. The simulator is done, once the remaining trading volume is zero, which is enforced at the very last time point. Any remaining volume at $H-1$ is transformed into a simple market order and executed immediately, at any price. Additional parameters control the simulators precise behavior:

volume : The targeted trading volume V.
Positive values indicate buy orders, negative values indicate sell orders.

consume : `'cash'` or `'volume'`
Defines whether `volume` should be interpreted as *cash* (goal: buy/sell shares for V dollars), or as *sharecount* (goal: buy/sell V shares).

period_length : `default=15`
Defines the duration at which a limit order is executed. After a trade has been placed, the simulator iterates over the next `period_length` orderbooks, before the results are reported and a reviewed order may be placed.

tradingperiods : `default=4`
Defines the number of trade reviews, that can be made within the time horizon $H =$ `period_length * tradingperiods`.

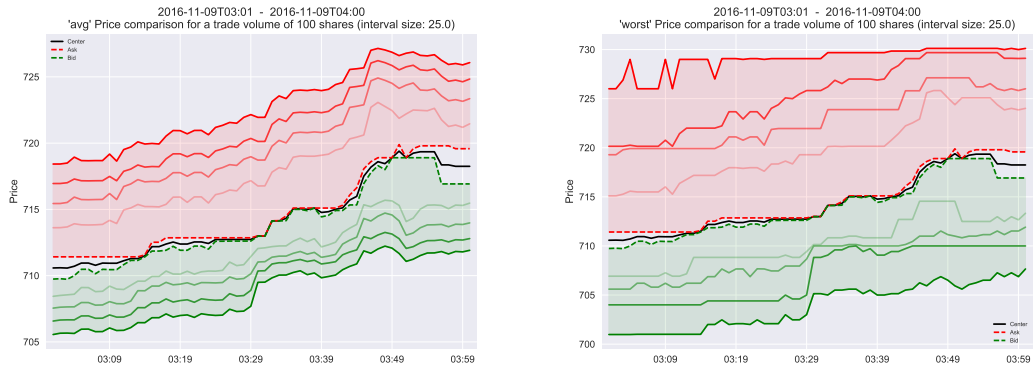
max_lengh_tolerance : `default=2`
Defines the accepted tolerance between actual and presumed trading horizon in minutes. Throws `ValueError`, if exceeded.

costtype : `default='slippage'`
Defines which of multiple cost functions to use in the reports returned.

3.3.1 Orderbook and strategy visualization

Figure 7 visualizes a 60 minutes long orderbook window, where the solid red lines mark the *average* (7a) and *worst* (7b) price, that has to be paid at a given time point, in case of an immediate market order of 100, 75, 50 and 25 bitcoins respectively. Analogously, the solid green lines represent achieved prices for sale orders of -25, -50, -75 and -100 bitcoins.

As can be seen in this graph, ask prices deviate more from the center price than bid prices. An plausible inference might be, that imbalances between demand and supply might serve as a valuable indicator for future price trends.



(a) Average market price. **(b)** Worst market price.

Figure 7: An orderbook window over a period of 60 minutes.

3.3.2 Masterbook

During instantiation, the OTS creates a copy of the first orderbook, called the *masterbook*. Hereinafter executed trades do only affect this internal *masterbook*. The remaining orderbooks are then converted into *deltabooks*, containing only changes between subsequent orderbooks.

The OTS may be reset to its initial state at any time via `ots.reset()`. This avoids computational overhead, when testing out multiple strategies on the same *window*, as only *masterbook* and *history* are reset, while *deltabooks* need not be recomputed and *orderbooks* need not be retransferred. `ots.reset()` provides optional parameters for modifying the simulators start conditions. As such the simulator might be instructed to start at a `custom_starttime` or with a `custom_startvolume`, to simplify the exploration of possible strategies.

3.3.3 Trade execution

The simulated trade execution is triggered by an external call to `trade(limit=...)`. The OTS expects a `limit` and iterates over the next `period_length` orderbooks, matching all eligible orders. The `limit` represents the highest accepted price level for buy orders and the lowest accepted price level for sell orders respectively. If `limit=None`, a simple market order is performed.

In a first step, all eligible orders, bound by the given limit and the total trade_volume, are cut from the internal masterbook and pasted into the `ots.trade_history`, as such these orders are assumed be fulfilled. The `volume` and `cash` variables are updated accordingly. The simulator then moves to the next time point and adds the corresponding *deltabook* to the masterbook.

In case of order size reductions[7], negative order sizes may appear in the masterbook. They are silently dropped, assuming they where matched before they actually vanished from the market. This is a possible source of trouble, as this assumption can not be proven to be valid. As such, matching orders that are simultaneously updated to another price level, are virtually doubled. They are perceived as a new order, even though the responsible market participant has already realized an execution and no basis to submit another one.

(1) `master += diff(ob[t] - ob[t-1])`, drop negative share counts.
(2) perform trade: buy until given limit
(3) `master -= bought bitcoins`
(4) done if `volume==0` or $t == T - 1$ (`forced=True` a)

Figure 8: Figure describing the masterbook adjustments as a graph?!

The masterbook is then ready to be queried again. Any eligible *new* orders are cut from the masterbook and past into the `ots.trade_history`. After `period_length` steps, a detailed trading report, as shown in Table 2 is returned. The upper case columns represent internal variables and orderbook statistics observed at the particular period start. The lower case columns summarize the actual trade in terms of highest, lowest and average price achieved, traded volume, cash flow and observed costs.

	ASK	BID	CENTER	SPREAD	LIMIT	T	VOLUME	volume_traded	CASH	cash_traded	...	avg	forced	initialMarketAvg	low	high	cost
03:01	711.42	709.74	710.58	1.68	713.0	15	100.00	46.77	0.00	-33280.72	...	711.51	False	718.42	711.42	713.00	43.48
03:16	712.52	711.86	712.19	0.66	715.0	15	53.23	28.90	-33280.72	-20630.99	...	713.99	False	718.42	712.52	715.00	98.53
03:31	715.10	712.98	714.04	2.12	717.5	15	24.33	6.68	-53911.71	-4780.95	...	716.16	False	718.42	715.10	717.41	37.28
03:46	718.60	717.77	718.18	0.83	720.0	15	17.65	17.65	-58692.66	-12706.15	...	719.73	False	718.42	718.60	720.00	161.57

Table 2: Trading history, as returned after four consecutive calls of `ots.trade()`.

3.3.4 Visualization

A visual representation of the same trading strategy, which underlies Table 2, is shown in Figure 9. Here, the OTS was instructed to buy 100 bitcoins within a period of 60 minutes and called with four consecutive limits prices 713, 715, 717.5 and 720, which are shown as grey step function in the upper subplot.

The second subplot displays trade executions and declining trade volume over time and the bottom subplot displays induced costs. Expected costs conform to the respective accumulated costs, extrapolated to the full trade volume.

7 Order size may be reduced due to order fulfillment, order updates or order cancelations through the other market participants.

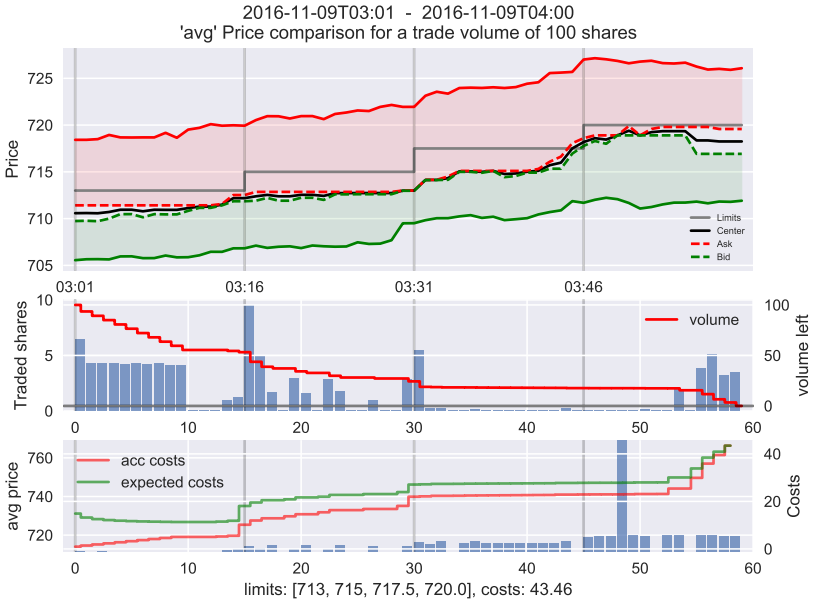


Figure 9: Visualization of an exemplary trading strategy.

3.3.5 Model correctness

The presented model does not account for market reactions, induced by the currently executed trading strategy. It moronically follows the market trend, as it would have evolved without any intervention. Some sources of troubles are pictures below:

- As stated in Section 3.3.3, the presented model can not distinguish between a limit order being removed from the market makers orderbook and a limit order essentially only being updated to another price level. In the latter case, the order presumably wouldn't have reappeared in the orderbooks after the simulator matched it.
- Other market participants typically monitor market activity thoroughly, which is particularly true for purely electronic markets of digital assets, like bitcoins. It is delusive to assume, that no other market participants or trading bots react to large orders, that eat significantly into the orderbook.

[placeholder]

Figure 10: Sample of a *curious masterbook shape*.
large part of order fulfilled, eaten deeply into the orderbook, deltabook brings in new orders close to original centerprice ⇒ uncommon gap.

- Orderbooks on a minutely basis miss a great part of the markets volatility.
  In addition to orderbook snapshots, professional data providers typically grant access to market level 2 data (see Section 2.1.5) in form of log files as well. The log files consist of timestamped orderbook updates (typically of type *remove* and

modify), which allow the reconstruction of the orderbook for an arbitrary time point. As mentioned in Section 3.1, Poloniex push API, to receive live order book updates and trades, was rather error-prone and occasionally failed to report important orderbook updates. As the valid reconstruction of orderbooks is highly vulnerable to missing logs, inconsistencies arose and the decision was made, to query orderbooks on a minutely basis only.

- Hidden Orders, as introduced in Section 2.1.7, are not accounted for.

As a consequence, trading on the masterbook can only be seen as an rough approximation of true market behaviour. Curious masterbook shapes, resulting from the described simulation process, encourage to perform actual feature extraction on the original orderbooks, examining the market as it would have evolved without any interventions.

4 Reinforcement Learning Approach

This chapter describes a RL approach, used to tackle the important problem of optimized trade execution. To a large extend, it is based on the RL formulation, as described by Nevmyvaka et al. [10]. Q-learning and dynamic programming are fused to find an optimal, state-based strategy from a training set of historic orderbook date. Sampling the discretized state space[8] in a brute force manner, the authors achieved an impressive 50% gain over the more simple Submit & Leave Strategy (see Section 2.1.8).

Some of the assumptions made in the original approach deserve a discussion and leave room for potential improvements. As their work was based on a rather large, proprietary dataset of 1.5 years of millisecond time-scale limit order data from NASDAQ, it was furthermore intriguing to evaluate its performance on a smaller, self-recorded dataset of limited resolution.

4.1 General Framework

In the first place, the general framework of the reinforcement learning problem is described.

4.1.1 State space

The state space consists of various variables, describing the current trade progress (*private variables*) and the current market situation as observable from orderbook data (*market variables*). The two private variables *remaining time* (`time`) and *remaining inventory* (`volume`) make the base for all subsequent experiments, while optional market variables may be enclosed to potentially assist the process of decision finding.

Each `state` $s \in < time, volume, o_1, o_2, ... >$ forms a vector of at least both private variables, plus a variable number of market variables. More specifically, the following market variables were evaluated in terms of improvement over a state space based on two private variables only.

Bid-Ask Spread : spread between best bid price and best ask price.

Bid-Ask Volume Misbalance : volume imbalance between orders at the best bid price and the best ask price.

Immediate Market Order Cost : costs, if remaining volume would be executed immediately, at the current market price.

8 The state space describes actual trade progress (i. e. *remaining time* and *remaining inventory*) as well as current market situation.

Signed Transaction Volume : signed volume of all trades executed within last 15 seconds. A positive value indicates more buy orders, while a negative value complies to more sell orders being executed.

In the original approach, market variables were discretized into 0 (low), 1 (medium) and 2 (high), while the concrete category mapping process was not further described. Market variables are extracted from the original orderbooks, as if the market had evolved without our impact. Private variables where discretized in 4 to 8 bins, while greater resolutions led to strictly better results and linearily higher computation times.

4.1.2 Action space

Actions define the level of trading aggression to be performed. In the original approach, action $a \in \mathbb{R}$ defines the deviation between current best price and chosen limit price, as $bid + a$ (for buy orders) and $ask - a$ (for sell orders). As such, actions must cross the full bid-ask-spread to find matching orders in the opposing book. Large actions a result in larger fractions of the order volume being matched immediately. Smaller or even negative actions a help to prolong the actual execution, and may help to profit from opportune market movements.

In case of the market situation as shown in Table 3, a buy order with an aggressive action $a = 1.4$ would map into $limit = bid + a = 28.7 + 1.4 = 30.1$. This limit would allow trading up to 75 shares instantaneously.

	Amount	Type	Volume	VolumeAcc	norm_Price
31.00	200.0	ask	6200.0	8425.0	1.074533
30.00	50.0	ask	1500.0	2225.0	1.039871
29.00	25.0	ask	725.0	725.0	1.005208
28.85	NaN	center	NaN	NaN	NaN
28.70	200.0	bid	5740.0	5740.0	0.994810
28.50	100.0	bid	2850.0	8590.0	0.987877
28.00	300.0	bid	8400.0	16990.0	0.970546

Table 3: Action $a = 1.4$ translates into $limit = 28.7 + 1.4 = 30.1$.

The employed number of selectable actions and their actual value range was not further specified.

4.1.3 Costs

Costs are defined as the slippage induced from the previously chosen actions. The baseline is given by the initial center price. The following formula is used to compute (partial)

costs in terms of price deviation from the idealized case of buying all shares at the initial center price:

$$cost_{im} = (avg\_paid - initial\_center) * volume\_traded \tag{14}$$

$$initial\_center = (\frac{ask_t + bid_t}{2} | t = 0 \tag{15}$$

Since the complete trade execution happens within a finite time horizon and full execution of the `volume` is mandatory, partial costs can simply be summed up without any discounting. Utilizing the OTS, it is not possible to compute occurring costs in advance exactly, as they depend on how the market evolves within the subsequent `trading_period`.

4.2 Backward learning

In order to learn the optimal limit for each possible situation, orderbook windows are examined in a backward, brute-force manner as described in Algorithm 4. Each orderbook window from the training data set is sampled $T * I * L$ times, where T is the number of performed limit revisions, I is the number of discrete volume states and L is the number of available actions.

Algorithm 4: Optimal_strategy, as described in [10].

```
Input: V=70.000$, H=60min, T=4, I=[12.5%..100%], L=[-4..10]
for t=1 to T do
    while not end of data do
        Transform (orderbook) → o_1..o_R
        for i=0 to I do
            for a=0 to L do
                Set x = [t, i, o_1, ..., o_R]
                Simulate transition x → y
                Calculate immediate cost_im(x, a)
                Look up argmax cost(y, p)
                Update cost([t, v, o_1, ..., o_R], a)
            end
        end
        Select the highest-payout action argmax cost(y, p) in every state y to output optimal policy
    end
end
```

While the algorithms running time depends solely on the resolution of the two private variables, the chosen action space and the size of the training data set, it is approximately independent of the number of market variables chosen. The precise transition simulation was not further described, but for this thesis the model described in Section 3.3.3 is utilized. The cost update rule is given below:

$$cost(x_t, a) = \frac{n}{n+1} cost(x_t, a) + \frac{1}{n+1} [cost_{im}(x_t, a) + \arg\min_p cost(x_{t-1}, p)] \tag{16}$$

The algorithm assumes the individual trading_periods to be of an (approximately) Markovian nature, where the optimal action to choose at state x_t with $t = \tau$ is completely independent of actions chosen previously ($t \geq \tau$).

As such, the state-action function can be computed inductively via dynamic programming. In a first round, expected costs for all actions in states being immediate predecessors of end states (i. e. $t = 1$) are computed according to Equation (16). Figure 11 visualizes the resulting optimal costs (right) and corresponding actions (left) after the first round has finished.

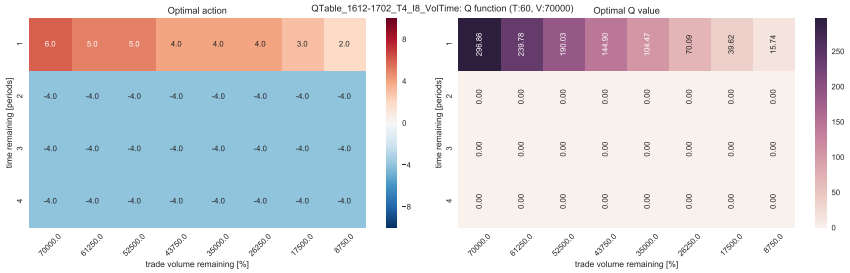


Figure 11: State-Action function, visualized after the first training round.
$T = 4$, $I = 8$, L=15

Knowing the optimal state-action values for all states with $t = 1$, all informations are given to compute the optimal state-action values for their predecessor states with $t = 2$ (see Figure 12).

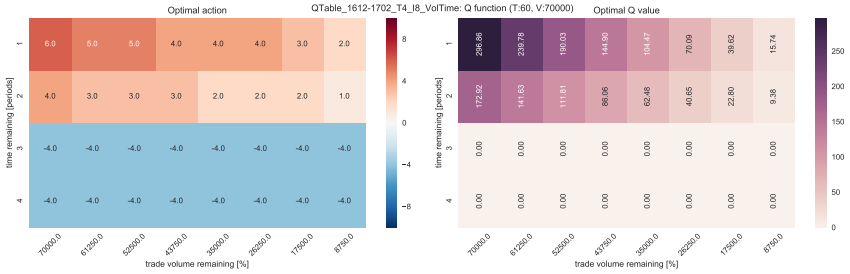


Figure 12: State-Action function, visualized after the second training round.
$T = 4$, $I = 8$, L=15

After T iterations a globally optimal policy, as shown in Figure 13, has been found. The annotated q values (right) denote the corresponding minimum over all available actions. The visualized state-action function was trained over orderbook snapshots from Nov, 10th 2017 10am to May, 31st 2017, partitioned into 4.154 orderbook windows of 60 minutes length each. Over a trading horizon of $T = 4$ `trading_periods`, 70.000$ cash (discretized in $I = 8$ intervals) had to be traded into Bitcoins. The state space included the two private variables `time` and `volume` only, while the action space comprised $L = 15$ actions. As such, a total of $4.154 * 4 * 8 * 15 = 1.993.920$ transition tuples were generated.

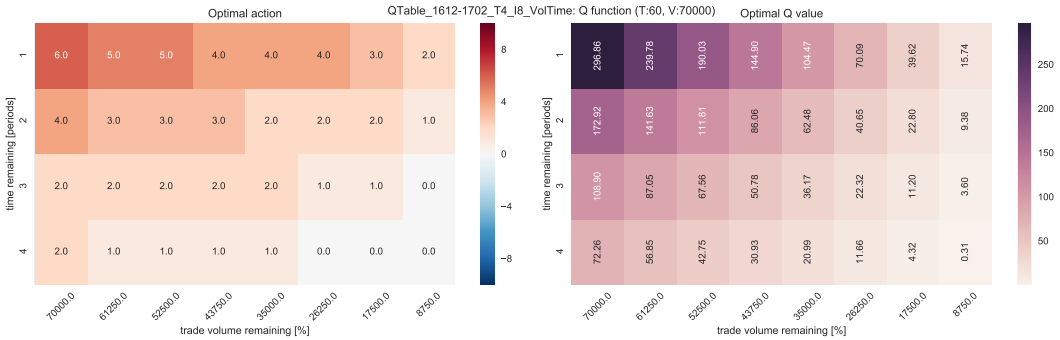


Figure 13: Final State-Action function.
$T = 4$ (60min), $I = 8$ (70.000$), L=15 ([-4, -3, . . ., 9, 10])

Figure 13 illustrates clearly, how the optimal strategy becomes more aggressive as time ceases and a large portion of the trading volume remains unexecuted.

4.3 Forward learning

In order to collect more realistic transition samples (see subsequent discussion in Section 4.4.4), an alternative, *forward sampling* method is presented.

4.3.1 Forward Sampling & Learning

Rather than examining all available actions on the great variety of combinational possibilities the state space has to offer, the forward approach repeatedly simulates trades in their entirety. The once again utilized OTS always starts with $V = 100\%$, but may be initialized at random start points (e. g. `t=0`, `t=15`, `t=30`, `t=45`) and keeps going until the full trade is executed.

As such, the obtained transitions stem from a more natural and furthermore fully continuous environment. Only actually attainable states, that could arise in the (simulated) reality, are used as transition start points, potentially increasing their significance. As major disadvantage, this forward sampling approach does not assure exhaustive exploration of the state space. A thorough exploration is not self-evident and must be supervised.

The pseudo-code of the forward sampling approach is shown in Algorithm 5. The algorithm iterates over the training data, starts E exploration phases per orderbook window and follows an ε-greedy exploration strategy. Each exploration phase starts with $\varepsilon = 1.0$ which finally decays to $\varepsilon = 0.05$ according to $\varepsilon = 0.05^{e/E}$.

If ε-greedy proposes an actions, that previously led to an end state (i. e. trade completed), an alternative action is sampled according to a gaussian distribution with its peak at the originally proposed action. Before the actual random draw, probabilities of all dead-end-actions are set to zero, to forestall pointless resampling. In case of repeatedly running into dead-ends (e. g. `max_dead_ends=4`) the exploration phase is aborted and continues with the next orderbook.

After every `retrain` exploration phases, the new samples are fit into the agents model (e. g. `retrain=256`) and the updated model used for future ε-greedy exploration. Prior to the sampling phase, the training set is shuffled.

In contrast to the backward sampling method, market variables may be queried on the go, such that they entail the strategies impact. Because sample trajectories stem from a continuous state space function approximations prove handy. Here a RandomForestRegressor (i. e. BatchTree (BT)) is utilized and retrained from scratch every `retrain` exploration phases. The alternative approach of exploiting Neural Fitted Q-iteration [13],

has been postphoned to future experiments.

Algorithm 5: Forward sampling and learning approach.

```
Input: data, V=70.000$, H=60min, T=4, L=[-4..10], E=60, retrain=256
Shuffle(data)
Split data into chunks of length retrain
while not end of chunks do
    for orderbook window in chunks do
        Init OTS(orderbook_window, V, H, T, L)
        for epoch=0 to E do
            Reset OTS to V = 100% and random time point (in H)
            repeat
                Set x_t = [time_left, volume_left, o_1, ..., o_R]
                Enquire ε-greedy action from model
                if action chain led to an end state previously then
                    choose other action
                end
                Apply action
                Remember transition {x_t, action, cost, x_{t+1}}
            until V = 0%
        end
        Retrain model from collected transitions (growing batch)
    end
end
```

4.4 Discussion

While the presented backward algorithm exploits the available data profoundly in a brute force manner, the underlying assumptions deserve a short discussion. Alternative formulations are tested out, to potentially improve the algorithms cost saving capabilities.

4.4.1 Subject of Trade

Nevmyvaka et al. [10] tackled the problem of trading a certain amount of shares within a fixed time period. A more practicable problem definition must distinguish between the concrete trading direction. While their problem definition fits to the case of selling shares that are in the investors possessions, it does not mirror the opposite case properly. If the subject of trade is to obtain shares, it is more realistic to define a fixed amount of disposable *cash*, rather than the number of *shares*, that shall be acquired. Subsequently, all experiments in buy direction are instructed to commute a fixed amount of cash into as many shares as possible.

If cash is the subject of trade, the optional market variable *Immediate Market Order Cost* (e. g. 72.321$) becomes meaningless, and will be replaced by *Immediate Market Order Shares* (e. g. 92 BTC), displaying the number of shares acquirable with a immediately.

4.4.2 Discrete State Space

The algorithms most obvious weakness lies in its vulnerability to seldomly observed market situations. Since the state-action function is implemented as a simple lookup table without any generalization capabilities, it is strictly dependent on a thorough exploration of the underlying state space. As exhaustive exploration is enforced for the value range of private variables only, market variables are entrusted to chance. Especially when increasing the state space dimension by adding multiple market variables simultaneously, the explanatory power of a learned state-action mapping depends crucially on the number of underlying observations. There exists even the chance of certain states never been monitored at all during the training phase.

The concrete discretization process was not further described or questioned. Particularly it is not stated, how boundaries between 0 (low), 1 (medium) and 2 (high) have been chosen, and how the trading performance may be influenced by a higher market variable resolution.

Potential Improvements are described and tested in Section 5.2.

Firstly, the discretization is automatized, such that boundaries between bins are automatically derived from the training data. Varying discretization resolutions are tested. Secondly, the underlying lookup tables are replaced by function approximations, making discretization unnecessary.

Alternative Market Variables may be considered.

Except for *Immediate Market Order Cost*, the market variables exploited (see Section 4.1.1) basically came from market level 1 data only. Section 5.2.3 evaluates the impact of additional market level 2 features, describing potential imbalances between ask and bid book and forecasting estimated action consequences.

4.4.3 Cost scaling

While discretization of market variables mainly affects the strategies explanatory power, the resolution of private variable `volume` leads to considerable rounding issues in regards to the cost function employed. As observed successor states x_{t-1} must be discretized equally to allow looking up the corresponding minimal costs, the immediate costs, as computed in Equation (14), must be scaled accordingly. Simply replacing `volume_traded` with `round(volume_traded)` falsifies the actual costs, as they correlate to the accomplished trade volume in an unpredictable, non-linear manner as indicated in Figure 14.

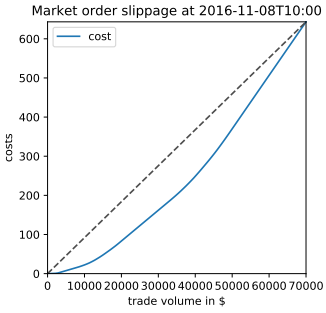


Figure 14: Non-linear slippage growth.

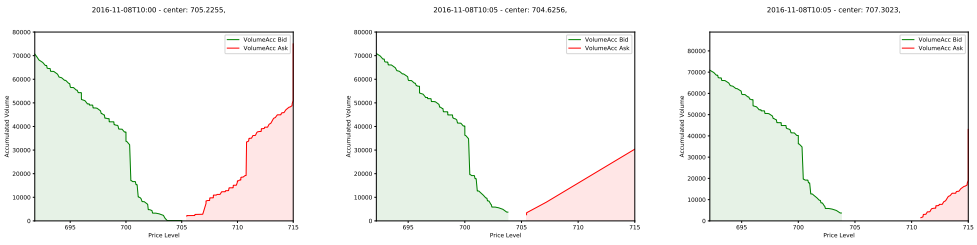
Nevmyvaka et al. [10] did not mention this problematic and presumably did not perform any cost scaling at all.

Potential Improvements A simple, but imprecise approach is to perform cost scaling, as described above. In general, this problem should be void, when function approximations are trained from the original float values, instead of rounded values.

4.4.4 Markov Property

The proposed backward algorithm assumes individual `trading_periods` to be of an (approximately) Markovian nature. This is in fact not true, as the OTS's internal masterbook shape depends drastically from the preceding trade history.

Figure 15 shows differing masterbook shapes, that can build the base of a simulated `trading_period`, e. g. if starting at $t = 45min$ and a remaining trade volume of 17.500\$. 15a shows the original orderbook, as if no orders had been matched by our strategy. This version of the masterbook is queried by the original backward algorithm, potentially resulting in more passive trading aggressions, as it does not account for the own impact on the market at all. 15b and 15c show two ideas, how the market impact can be incorporated. Both lead to different masterbook shapes and to different subsets of attainable prices.



(a) Original Orderbook. **(b)** Assuming 52.500 shares being matched at `t=0`, then no further matches. **(c)** Assuming 52.500 shares being matched evenly at 1.166 shares per minute.

Figure 15: Different shaped masterbooks at `t=45`.
The shapes differ, depending on the preceding trade history.

Potential Improvements Initializing the OTS outside the original start point ($t = 0, V = 100\%$), the supposedly traded volume must be incorporated, e. g. as done in Figure 15c. Other than that, a more realistic sampling method may be applied: The forward sampling process, described in Section 4.3, approaches sampling from the other side. Rather than evaluating actions on individual `trading_periods`, the OTS always starts with $V = 100\%$ and keeps going until the full trade is executed. As such, more realistic samples are generated. This does not give the problem a Markovian nature, but the internal masterbook is always of a realistic shape. In

contrast to the backward approach, exhaustive exploration of the state space is not self-evident and must be enforced. As a side benefit, the forward sampling approach generates more versatile samples for training the function approximates, as the generated samples do not necessarily start at *discrete* start points only.

4.4.5 Action Space

The mapping from actions to limits deserves some reflection as well. The proposed method (see Section 4.1.2) adds the value of the chosen action directly to the current best price of the opposing book, such that the bid-ask spread must be crossed before any orders may be matched.

On the one hand, it seems pointless to fix the origin at the best price of the *opposing* book. By doing so, it appears obvious, that decisions derived from state spaces including a variable for the current spread size, outperform decisions derived from state spaces not including this market variable. Indeed, the bid-ask spread was posed as the market variable causing the greatest individual impact on cost reduction, namely -7.97%, which is a major fraction of the maximum achievement of -12.85%[9].

On the other hand, the proposed mapping method does not necessarily fit to the Bitcoin data at hand. As is shown in Figure 5, Bitcoin prices have burst from roughly 700$ to more than 2.000$ in the period of recording, i. e. interpreting actions as the absolute difference to the current best prices, maps to significantly different levels of aggression as time passes.

Potential Improvements lay in alternative action-limit mappings.

On the one hand side, the limit base may be fixed to the other side of the bid-ask-spread, reducing pointless dependencies on the speads size. On the other hand, actions should be interpreted as factors, rather than summands to allow for a consistent aggression interpretation. The mapping functions are thus redefined as follows:

`limit_buy = ask * (1 + (a/1000))` instead of `limit_buy = bid + a`

`limit_sell = bid * (1 - (a/1000))` instead of `limit_sell = ask - a`.

Actions a now represent the deviation from the limit base in per mile. The influence of the chosen limit base is analyzed in Section 5.2.1.

9 Strategies derived from five dimensional state spaces including the market variables Spread, ImmCost & Signed Vol, as described in Section 4.1.3, outperformed strategies derived from two dimensional state spaces, containing the private variables only, by -12.85% in average.

5 Orderbook Agents

This chapter describes several *Orderbook Agents*, implemented to learn optimal *Submit & Revise* trading strategies from a given training data set.

5.1 Implementation

Each *Orderbook Agents* inherits some commonly shared functionality from the super class `class RL_Agent_Base`. This base class provides a consistent framework, allowing to swap the logic behind `learn_from_samples` and `predict` only, while general functionality like `load`, `save`, `plot_heatmap_Q` and `collect_samples` may be reused.

An *Orderbook Agent* holds the environment parameters, used to initialize the OTS. As such, different definitions for the optimal trading problem require different Orderbook Agents.

5.1.1 Backward Sampling & Learning

The process of sampling from training data and learning optimal state-action values is split into two independent phases, mostly due to performance[10] reasons. In the *sampling phase*, all possible combinations of actions and private variables are evaluated and stored as 5-tuples $sample = (state, action, cost, timestamp, new\_state)$. As market variables are supposedly not influenced by the agents behavior (see Section 4.1.1), it suffices to remember the orderbooks time point, to allow adjacent substitution of market variables.

Samples are stored as DataFrame and may be exported and reused by other agents[11], as wished. This allows to skip the time consuming process of backward sampling. Before the action learning phase starts, the collected samples may be enhanced by a variety of market variables, discretized or untouched. The helper function `addMarketFeatures_toSamples()` retrospectively adds market variables to the agents samples DataFrame.

If desired, features are discretized according to their individual value range.
E. g. a chosen resolution of 3, evenly transforms feature o_m into o_{m_disc3}, according to automatically determined boundaries:
0 (if $o_m < 1/3$ quantile), 1 (if $1/3 \leq o_m < 2/3$ quantile) and 2 (if $o_m \geq 2/3$ quantile).
Boundaries arise from the training data and are applied to test data unaltered.

10 The backward sampling phase for the example mentioned in Section 4.2, took roughly 10 hours, even though the work load was distributed among 24 CPU's.
11 Both agents should refer to the same environment settings, as samples collected from different environments (e. g. trading horizon of $H = 8min$ vs. $H = 60min$) may not be compatible.

The learning phase starts hereinafter, whereas two types of orderbook agents have been implemented and experimented with, each of which employ different techniques:

QTable_Agent : Dynamic Programming as described in Section 4.2.
Expected state-values are stored in a simple lookup table, hence discretization of the state space is required. For each state, a vector of length $L = num\_actions$ is maintained, of which the first argmin refers to the optimal action. The *QTable* does not generalize to previously unobserved states and returns the very first action (here -4) by default. For proper cost-updates an additional *NTable* is maintained, referencing the particular number of observations made.
QTable and NTable entries are computed in an iterative manner. In the first round, all samples with `time_left=1` are evaluated, in the second round all samples with `time_left=2`, etc. The private variable `volume` is discretized according to the specified resolution and linear cost scaling (see Section 4.4.3) is performed.

BatchTree_Agent : Tree-Based Batch Mode Reinforcement Learning [4].
An ensemble of 400 decision trees, with a maximum allowed depth of 15 is fed with the full batch of samples simultaneously, no discretization required. T learning rounds are performed, to completely allow expected costs to unfold over the given time horizon.
The state space is enlarged by an extra dimension for the chosen action. Predicting the optimal action to choose for a given state, the ensemble must predict $L = num\_actions$ q-values, from which the argmin refers to the optimal action.

5.2 Backward Sampling Experiments

Various experiments have been conducted to examine the agents ability to find optimal solutions to the problem of optimized trade execution. The recorded orderbook snapshots for currency pair USDT/BTC (see Section 3.1) have been split into training period (Nov, 10th 2016 - Apr, 30th 2017) and test period (May 2017).

The studied agents refer to the very same environment settings. Their common task is to buy Bitcoins worth of 70.000$ within a trading horizon of 60 minutes. The problem definition is opposed to the task evaluated by Nevmyvaka et al. [10], where the number of shares (respectively Bitcoins) was fixed, rather than the amount of cash.

The training set translates into 4.154 orderbook windows, while the test set gives 724 orderbook windows. A period_length of 15 minutes is assumed, such that the OTS expects up to 4 order limit prices. After an initial backward sampling phase, the obtained samples-DataFrame makes the base for varying learning phases.

5.2.1 Action-Limit Mapping

As mentioned in Section 4.4.5, it seems pointless to force actions to cross the bid-ask-spread before any orders can be matched. Table 4 shows exemplary for November 2016, how the choice of the limit base affects the agents trading performance. While buy orders, forcing agents to cross the bid-ask-spread (i. e. `currBid*`), typically benefit from the two market variables `spread` and `marketSpread`, this is not necessarily the case for agents which have the limit base fixed to the opposing best price (i. e. `currAsk*`). As the latter agents consistently showed better performance, the limit base is henceforth fixed to the best price of the corresponding trading direction.

	slippage	performance
currBid	220.19	92.6%
currBid_mSpread	218.71	91.9%
currBid_spread	215.76	90.7%
currAsk	214.00	90.0%
currAsk_mSpread	214.53	90.2%
currAsk_spread	215.38	90.6%
S&L: 5	237.75	100.0%
MarketOrder	737.58	310.2%

Table 4: Evaluating the impact of different limit base levels.
Results stem from applying the respective strategies on all 537 orderbook windows recorded between Nov, 10th 2016 and Nov, 30th 2016.

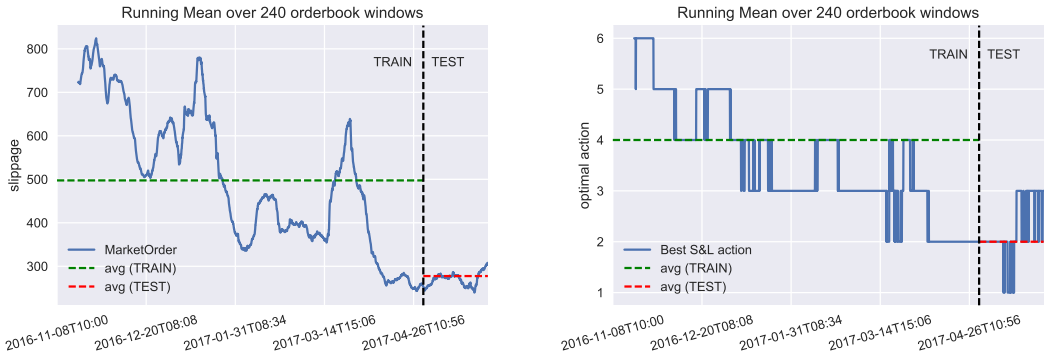
In order to produce comparable results, all subsequent experiments are based on the same set of 15 actions: $[-4,-3,...,8,9,10]$. In line with the formulas presented in Section 4.4.5, these actions translate into order limits deviating by -0.4% to $+1.0\%$ from the current best price.

5.2.2 Baseline

Simple S&L strategies (see Section 2.1.8) and immediate market order placements serve as benchmark for the performance measurement.

Due to bursting Bitcoin prices (see Figure 5), the investigated sum of 70.000$ constitutes a declining contingent of the total market volume. Figure 16a shows a running average over the amount of slippage, as induced by immediate market orders. Concurrent to declining slippage, the optimal S&L actions become less aggressive as time passes. The green, dashed lines show the respective average over the train period, which significantly differs from the red, dotted line referring to the average over the test period. Figure 17 shows the average costs, induced by varying S&L strategies within the training period.

In order to provide a more realistic and unexaggerated baseline, the optimal S&L action is estimated from the test period. As such, performance of subsequents experiments are



(a) Observed Market Order Slippage. **(b)** Best S&L action.

Figure 16: Concurrent to declining slippage, the optimal S&L actions become less aggressive as time passes.

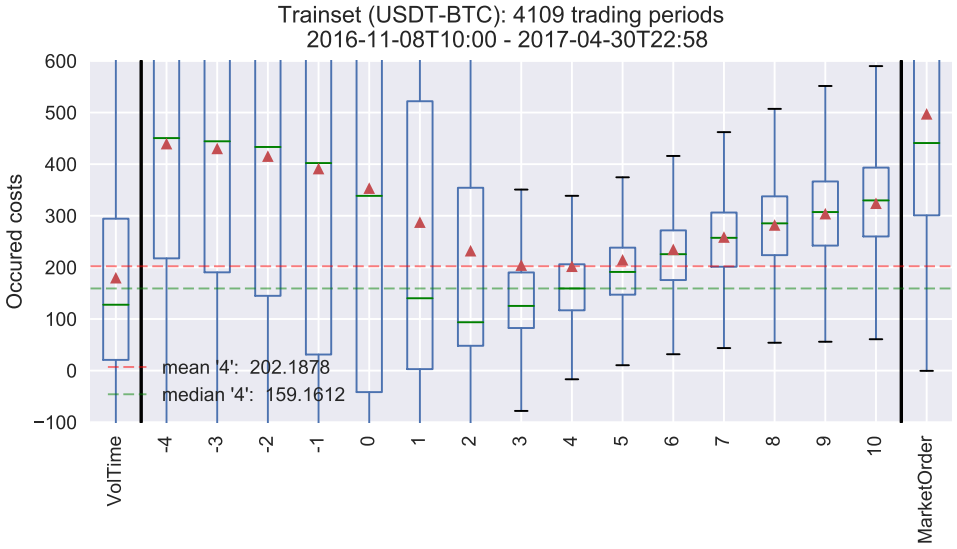


Figure 17: Average costs induced by the varying S&L actions over the full training period. In average, action 4 performed best.

compared to the performance of a simple market order and the S&L strategy "2", with the initial order limit fixed to 1.002 times the initial ask (i. e. +0.2%).

5.2.3 Additional Market Variables

In addition to the market variables proposed in Section 4.1.2, the following orderbook features have been examined:

marketPrice* describes the relative difference between current center price and the worst price that must be paid (or is received) in case of simple market orders.

sharecount_* quotes the number of Bitcoins, immediately available for 70.000$. `sharecount_buy` is tantamount to the originally proposed `ImmCost` (See Section 4.4.1).

center_price was added as a consequence of Figure 16: Apparently, as the ratio between current Bitcoin price and the investable 70.000$ increases, an optimal strategy may have to place less aggressive order limits.

_a*_effects quotes the minimum volume immediately obtainable by individual actions. The value is a lower bound of the truly obtainable volume, because only the current orderbook is retrievable. New opportunities, arising within the forthcoming `trading_period` are unacquainted and not included. Results are shown in Section 5.2.5.

Table 5 shows the performance of individual QTable agents trained on both private variables plus one discretized market variable attached at a time. While observed trading costs undercut those induced by simple market orders by -49.16%, the gain in comparison to the optimal S&L strategy lies at -8.93%.

Adding the market variable `spread` yields a notable improvement of -4.84% over the plain `VolTime`-agent. By increasing the discretization level to 5, the performance can be further improved to -6.34%, verifying the findings of Nevmyvaka et al. [10]. However, in contrast to their reported results, the market variable `sharecount_buy` falls flat with a worsening of $+6.70\%$ (compared to -4.26%).

Disappointing is the futility of `center_price`. For a discretization level of 5, the corresponding agent underperforms the plain `VolTime`-agent by $+7.35\%$. Potentially, this may be explained by a compulsorily underexploited training set: Due to generally rising Bitcoin prices, the majority of prices in the test period will map to the highest discretization value available, which, in a simple lookup-table, resembles only a fraction of the assessed training set.

The different discretization levels yield rather mixed results. While `spread` typically performed best in case of 5 levels, no clear statement can be made for the other variables.

The exact reason, why factually only adding `spread` causes a beneficial effect, is unclear, but one possible explanations lies in the data set employed. The original experiment assessed a different data quality. Due to the low resolution of the available orderbook

	slippage	med	std	perf_2	perf_4	perf_M	perf_VolTime
center_price_disc3	149.57	46.15	420.14	−3.44%	−15.19%	−46.10%	−0.70%
center_price_disc5	161.70	37.83	450.36	+4.39%	−8.32%	−41.73%	+7.35%
marketPrice_buy_worst_disc5	146.83	39.62	351.77	−5.21%	−16.75%	−47.08%	−2.52%
marketPrice_sell_worst_disc5	148.94	42.71	388.86	−3.85%	−15.55%	−46.33%	−1.12%
marketPrice_spread_disc5	150.46	44.15	371.32	−2.87%	−14.69%	−45.78%	−0.11%
marketPrice_imbalance_disc5	150.10	68.68	336.60	−3.10%	−14.90%	−45.91%	−0.35%
sharecount_buy_disc3	149.57	46.15	420.14	−3.44%	−15.19%	−46.10%	−0.70%
sharecount_buy_disc5	160.73	37.78	449.22	+3.76%	−8.87%	−42.08%	+6.70%
sharecount_imbalance_disc5	148.03	40.49	372.86	−4.44%	−16.07%	−46.65%	−1.73%
sharecount_sell_disc5	151.50	47.49	425.35	−2.20%	−14.10%	−45.40%	+0.58%
sharecount_spread_disc5	148.93	40.84	352.12	−3.86%	−15.56%	−46.33%	−1.13%
spread_disc3	147.41	35.48	370.96	−4.84%	−16.42%	−46.88%	−2.14%
spread_disc5	141.07	37.39	349.72	−8.93%	−20.02%	−49.16%	−6.34%
spread_disc9	142.80	36.69	364.76	−7.81%	−19.03%	−48.54%	−5.20%
VolTime	150.63	33.83	358.66	−2.76%	−14.60%	−45.72%	0.00%
2	154.90	68.62	389.15	0.00%	−12.17%	−44.18%	+2.84%
4	176.37	141.66	273.58	+13.86%	0.00%	−36.44%	+17.09%
MarketOrder	277.49	246.48	158.66	+79.14%	+57.33%	0.00%	+84.22%

Table 5: Evaluating the impact of additional market variables.
Average performance over the test period may 2017, currency pair USDT/BTC.
See Appendix A.1, Table 12 for complete results.

snapshots, a large fraction of the market activity remains inaccessible and consequently the majority of trading opportunities are missed by the agents. Furthermore, the minute time-scaled data inevitably requires the trading horizon to be rather long. Experiments with shorter time horizons nullified the achievable savings, while longer time horizons led to unacceptable computation times. Consequently, the agents where trained on 4.109 sixty-minute orderbook windows, while Nevmyvaka et al. invoked 45.000 two-minute orderbook windows.

Look-Ahead Features

In order to proof the algorithms general ability to find costs reducing strategies, look-ahead features[12] were added to the universe of market variables. `future_center*` quotes percentual changes between the current center price and the center price in 5, 15 and 60 minutes respectively. This hypothetical knowledge about future price trends reduces observed trading costs by −12.47% (i. e. −14.88% over simple S&L strategy).

	slippage	med	std	perf_2	perf_4	perf_M	perf_VolTime
ob_direction_disc5	61.99	97.52	319.84	−59.98%	−64.85%	−77.66%	−58.84%
future_center5_disc5	141.61	41.79	389.58	−8.58%	−19.71%	−48.97%	−5.98%
future_center15_disc5	133.78	40.90	405.88	−13.63%	−24.15%	−51.79%	−11.18%
future_center60_disc5	131.85	47.63	391.46	−14.88%	−25.25%	−52.49%	−12.47%
VolTime	150.63	33.83	358.66	−2.76%	−14.60%	−45.72%	0.00%
2	154.90	68.62	389.15	0.00%	−12.17%	−44.18%	+2.84%
4	176.37	141.66	273.58	+13.86%	0.00%	−36.44%	+17.09%
MarketOrder	277.49	246.48	158.66	+79.14%	+57.33%	0.00%	+84.22%

Table 6: Evaluating the impact of look-ahead features.
Average performance over the test period may 2017, currency pair USDT/BTC.

12 look-ahead features provide a glance into the future, and are thus equivalent to cheating.

A vastly larger impact (−58.84% respectively −59.98%) is caused by the look-ahead feature `ob_direction`, quoting the general price trend of the currently observed orderbook window. In contrast to the `future_center*` variables, it's value stays constant within individual orderbook windows: `ob_direction = orderbook[-1].get_center()/ orderbook[0].get_center()`, which seems to provoke more stable strategies.

Constant Market Variables

The findings from the look-ahead features encouraged for a supplementary experiment. Rather than considering the actual market situation, the market variables are observed once (at `t=0`), and kept frozen for the remaining trading horizon. Hope was, to provoke more stable strategies, as the agents could potentially decide on a *major* strategy and consequently only adapt the limits to the private variables representing the actual trade progress.

	slippage	med	std	perf_2	perf_4	perf_M	perf_VolTime
center_orig_disc3	158.96	48.93	431.17	2.62%	−9.87%	−42.71%	+5.54%
center_orig_disc5	150.37	47.01	422.14	−2.92%	−14.74%	−45.81%	−0.17%
marketPrice_buy_worst_disc5	150.74	50.15	402.13	−2.69%	−14.54%	−45.68%	+0.07%
marketPrice_sell_worst_disc5	157.56	43.95	379.33	+1.72%	−10.66%	−43.22%	+4.61%
marketPrice_spread_disc5	152.19	53.07	398.10	−1.75%	−13.71%	−45.15%	+1.04%
marketPrice_imbalance_disc5	147.16	47.03	383.19	−5.00%	−16.57%	−46.97%	−2.30%
sharecount_buy_disc3	149.57	46.15	420.14	−3.44%	−15.19%	−46.10%	−0.70%
sharecount_buy_disc5	150.37	47.01	422.14	−2.92%	−14.74%	−45.81%	−0.17%
sharecount_imbalance_disc5	152.44	53.41	400.89	−1.59%	−13.57%	−45.06%	+1.20%
sharecount_sell_disc5	150.37	47.01	422.14	−2.92%	−14.74%	−45.81%	−0.17%
sharecount_spread_disc5	149.23	53.41	402.65	−3.66%	−15.39%	−46.22%	−0.93%
spread_disc3	148.92	41.70	361.12	−3.86%	−15.57%	−46.33%	−1.14%
spread_disc5	143.56	40.78	350.74	−7.32%	−18.60%	−48.26%	−4.69%
spread_disc9	145.54	43.25	352.05	−6.04%	−17.48%	−47.55%	−3.38%
VolTime	150.63	33.83	358.66	−2.76%	−14.60%	−45.72%	0.00%
2	154.90	68.62	389.15	0.00%	−12.17%	−44.18%	+2.84%
4	176.37	141.66	273.58	+13.86%	0.00%	−36.44%	+17.09%
MarketOrder	277.49	246.48	158.66	+79.14%	+57.33%	0.00%	+84.22%

Table 7: Evaluating the impact of constant market variables.
Average performance over the test period may 2017, currency pair USDT/BTC.
See Appendix A.2, Table 13 for complete results.

While Table 7 shows improved performance for previous bad performers, the leading `VolTimeSpread`-agent dropped from −8.93% to −7.32%. Contrary to initial hopes, this approach did not lead to superior performance and was not further pursued.

5.2.4 Simulation of preceding trades

As described in Section 4.4.4, the OTS's internal masterbook shape depends drastically from the preceding trade history. In the following, the effect of incorporating the own impact on the market is investigated.

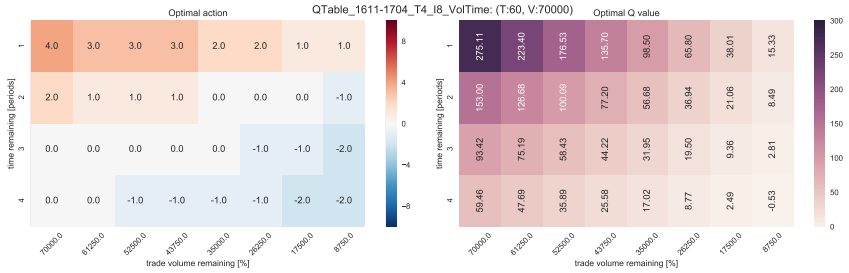
The backward sampling phase is modified, such that prior to the actual trade simulations, the supposedly consumed trading volume is removed from the masterbook. The

	slippage	med	std	perf_2	perf_4	perf_M	perf_VolTime	perf: Table 5
center_orig_disc3	158.97	48.93	431.16	+2.62%	−9.87%	−42.71%	+5.54%	−0.70%
center_orig_disc5	150.47	47.49	422.12	−2.86%	−14.69%	−45.78%	−0.11%	+7.35%
marketPrice_buy_worst_disc5	148.33	47.46	400.73	−4.24%	−15.90%	−46.55%	−1.53%	−2.52%
marketPrice_sell_worst_disc5	154.06	42.10	370.91	−0.54%	−12.65%	−44.48%	+2.28%	−1.12%
marketPrice_spread_disc5	151.12	53.07	395.68	−2.44%	−14.32%	−45.54%	+0.33%	−0.11%
marketPrice_imbalance_disc5	143.86	47.05	375.43	−7.13%	−18.44%	−48.16%	−4.49%	−0.35%
sharecount_buy_disc3	149.57	46.15	420.14	−3.44%	−15.19%	−46.10%	−0.70%	−0.70%
sharecount_buy_disc5	150.47	47.49	422.12	−2.86%	−14.69%	−45.78%	−0.11%	+6.70%
sharecount_imbalance_disc5	151.92	53.41	400.69	−1.92%	−13.86%	−45.25%	+0.86%	−1.73%
sharecount_sell_disc5	149.93	47.01	420.70	−3.21%	−14.99%	−45.97%	−0.47%	+0.58%
sharecount_spread_disc5	149.36	53.41	402.86	−3.58%	−15.31%	−46.17%	−0.84%	−1.13%
spread_disc3	145.63	41.89	346.00	−5.98%	−17.43%	−47.52%	−3.32%	−2.14%
spread_disc5	139.87	41.22	336.30	−9.70%	−20.69%	−49.59%	−7.14%	−6.34%
spread_disc9	141.30	42.80	336.88	−8.78%	−19.89%	−49.08%	−6.19%	−5.20%
ob_direction_disc5	62.03	97.52	320.76	−59.95%	−64.83%	−77.64%	−58.82%	−58.84%
VolTime_simulatedTrades	147.86	42.10	346.17	−4.55%	−16.17%	−46.72%	−1.84%	
VolTime	150.63	33.83	358.66	−2.76%	−14.60%	−45.72%	0.00%	
2	154.90	68.62	389.15	0.00%	−12.17%	−44.18%	+2.84%	
4	176.37	141.66	273.58	+13.86%	0.00%	−36.44%	+17.09%	
MarketOrder	277.49	246.48	158.66	+79.14%	+57.33%	0.00%	+84.22%	

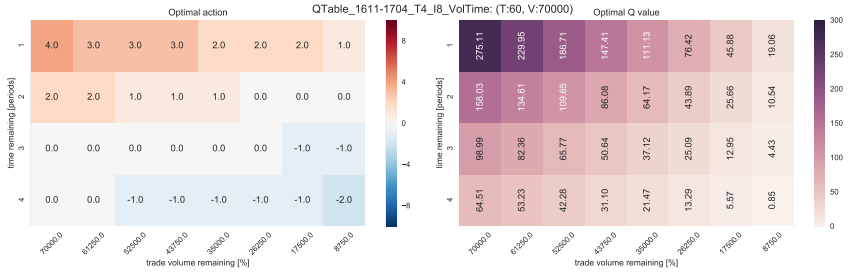
Table 8: Evaluating the impact of incorporating preceding trades.
Average performance over the test period may 2017, currency pair USDT/BTC.
See Appendix A.3, Table 14 for complete results.

trading volume affected is matched and removed from the masterbook evenly along the elapsed time, as is indicated in Figure 15c.

Table 8 attests a clear improvement over the original sampling method, if preceding trades are incorporated into the OTS masterbook. Figure 18 shows, how the plain `VolTime`-agent becomes slightly more aggressive, which improves the performance over the optimal S&L strategy from −2.76% to −4.55%.



(a) QTable of `VolTime`-agent.



(b) QTable of `VolTime_simulatedTrades`-agent.

Figure 18: A slight increase in aggression, if preceding trades are incorporated.
$T = 4$, $I = 8$, L=15

The best performing `VolTimeSpread_disc5`-agent now undercuts costs induced by simple market orders by −49.59% (before: −49.16%), the gain in comparison to the optimal S&L strategy enlarged to −9.70% (before: −8.93%).

5.2.5 Estimating Action Effects

Table 9 evaluates the impact of adding (lower bound) knowledge about consequences of individual actions to the state space. That way, `_a_4_` quotes the minimum volume immediately obtainable by a limit order placed +0.4% above the current ask price.

Potentially due to coarse discretization, these variables cause no major impact. All `_a*_`-agents perform worse than the underlying `VolTime_simulatedTrades`-agent.

	slippage	med	std	perf_2	perf_4	perf_M	perf_VolTime
_a_-4_disc5	149.84	40.54	368.50	−3.27%	−15.04%	−46.00%	−0.52%
_a_-3_disc5	150.70	37.95	372.15	−2.71%	−14.55%	−45.69%	+0.05%
_a_0_disc5	153.01	45.17	375.98	−1.22%	−13.25%	−44.86%	+1.58%
_a_1_disc5	148.56	41.60	385.79	−4.10%	−15.77%	−46.46%	−1.37%
_a_2_disc5	149.47	43.47	368.49	−3.50%	−15.25%	−46.13%	−0.77%
_a_3_disc5	145.04	42.42	365.27	−6.37%	−17.76%	−47.73%	−3.71%
_a_4_disc5	155.52	40.84	373.98	+0.40%	−11.82%	−43.95%	+3.25%
_a_5_disc5	152.05	38.53	372.03	−1.84%	−13.79%	−45.21%	+0.94%
_a_6_disc5	151.80	40.39	371.41	−2.00%	−13.93%	−45.29%	+0.78%
_a_7_disc5	148.61	40.39	363.67	−4.06%	−15.74%	−46.44%	−1.34%
_a_8_disc5	152.46	41.26	363.02	−1.57%	−13.56%	−45.06%	+1.22%
_a_9_disc5	151.59	40.84	363.68	−2.14%	−14.05%	−45.37%	+0.64%
_a_10_disc5	150.28	40.84	363.70	−2.99%	−14.80%	−45.84%	−0.23%
VolTime_simulatedTrades	147.86	42.10	346.17	−4.55%	−16.17%	−46.72%	−1.84%
VolTime	150.63	33.83	358.66	−2.76%	−14.60%	−45.72%	0.00%
2	154.90	68.62	389.15	0.00%	−12.17%	−44.18%	+2.84%
4	176.37	141.66	273.58	+13.86%	0.00%	−36.44%	+17.09%
MarketOrder	277.49	246.48	158.66	+79.14%	+57.33%	0.00%	+84.22%

Table 9: Evaluating the impact of action-consequence knowledge.
Average performance over the test period may 2017, currency pair USDT/BTC.

5.2.6 Function Approximation: BatchTree

In the next step the look-up table is replaced by an approximation thereof. This eliminates any need for discretization, avoids the cost scaling problem and as such, potentially helps to exploit the underlying sample transitions better. Various BT-Agents are trained on the sample transitions collected by the improved backward collection process (see Section 5.2.4). Results are shown in Table 10.

While the plain `BT_VolTime`-agent performs de facto worse than the original, discrete `VolTime`-agent (−2.47% vs. −4.55%), agents employing individual market variables perform terribly if fed with continuous values. The worst of all, `BT_VolTimeSpread`-agent, performs +60.79% worse than the optimal S&L strategy!

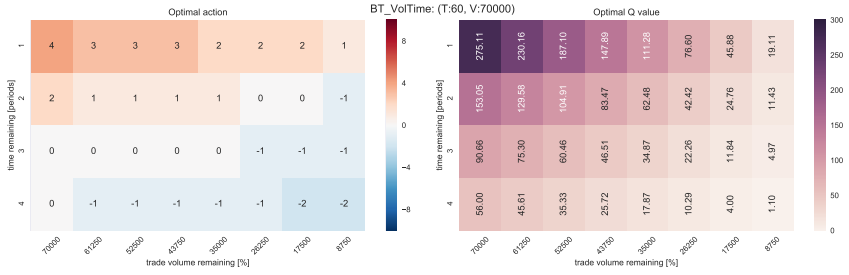
Conversely, a BT-agent trained on all fifteen `_a*_effects`-variables simultaneously lifts the BT performance from −2.47% to −7.53%, which is a good result. Unfortunately, it

	slippage	med	std	perf_2	perf_4	perf_M	perf_VolTime
BT_VolTime_a4_	163.21	25.48	466.45	+5.36%	−7.46%	−41.18%	+8.35%
BT_VolTime_a*_	143.24	38.53	373.35	−7.53%	−18.78%	−48.38%	−4.90%
BT_VolTime	151.08	47.39	391.02	−2.47%	−14.34%	−45.55%	+0.30%
BT_VolTimeSpread	249.07	237.29	142.11	+60.79%	+41.22%	−10.24%	+65.36%
VolTime_simulatedTrades	147.86	42.10	346.17	−4.55%	−16.17%	−46.72%	−1.84%
VolTime	150.63	33.83	358.66	−2.76%	−14.60%	−45.72%	0.00%
2	154.90	68.62	389.15	0.00%	−12.17%	−44.18%	+2.84%
4	176.37	141.66	273.58	+13.86%	0.00%	−36.44%	+17.09%
MarketOrder	277.49	246.48	158.66	+79.14%	+57.33%	0.00%	+84.22%

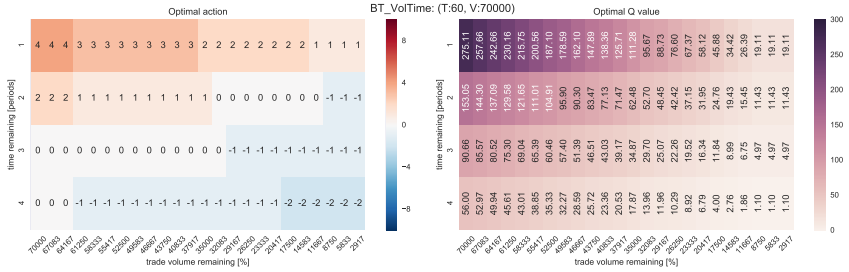
Table 10: Evaluating the performance of function approximators.
Average performance over the test period may 2017, currency pair USDT/BTC.

does not catch up with the previously attained −9.7% (see Section 5.2.4).

Figure 19 shows exemplary for the plain `VolTime`-agent, how the BT-agent is able to deliver a more accurate resolution of q-values and optimal actions.



(a) QTable by `VolTime`-agent.



(b) QTable approximation by `BT_VolTime`-agent (High resolution).

Figure 19: The BT-agent generalizes to continuous states.

5.3 Forward Sampling Experiments

Based on the same training data and simulator settings as used in the Backward Sampling Experiments, the novel forward sampling approach (see Section 4.3) is applied.

A BT-agent samples from the previously shuffled training data, following an ϵ-greedy strategy. In each exploration phase the respective orderbook window is run through up to 60 times, while only disparate paths are traced. After every 256 exploration phases, the underlying RandomForestRegressor is retrained from scratch, using all transition samples collected to this point. Intermediate models are stored, to document the learning

progress over time. The continuous state space consists of both private variables and fifteen minimum-consequence-variables (i. e. `_a*_`) as introduced in Section 5.2.3.

The performance of all sixteen intermediate models is shown in Table 11. Starting vaguely halfway through the training data, the `BT_Forward`-agent outperforms all previously examined agents, before the final model constitutes the overall winner with an improvement of −11.41% over the optimal S&L strategy.

	slippage	med	std	perf_2	perf_4	perf_M	perf_VolTime
BT_Forward_samples_025.078	143.72	74.69	300.93	−7.22%	−18.52%	−48.21%	−4.59%
BT_Forward_samples_043.334	147.92	82.24	310.49	−4.51%	−16.13%	−46.69%	−1.80%
BT_Forward_samples_061.103	149.56	77.33	288.56	−3.45%	−15.20%	−46.10%	−0.71%
BT_Forward_samples_079.809	146.67	77.20	277.43	−5.31%	−16.84%	−47.14%	−2.62%
BT_Forward_samples_097.615	143.45	73.18	281.71	−7.40%	−18.67%	−48.31%	−4.77%
BT_Forward_samples_116.008	144.26	77.94	285.80	−6.87%	−18.21%	−48.01%	−4.23%
BT_Forward_samples_134.011	138.22	73.30	269.71	−10.77%	−21.63%	−50.19%	−8.24%
BT_Forward_samples_152.352	136.90	69.59	264.95	−11.62%	−22.38%	−50.66%	−9.11%
BT_Forward_samples_170.605	137.75	77.50	261.17	−11.07%	−21.90%	−50.36%	−8.55%
BT_Forward_samples_188.224	137.64	73.71	253.37	−11.14%	−21.96%	−50.40%	−8.62%
BT_Forward_samples_206.409	137.10	74.22	257.54	−11.49%	−22.27%	−50.59%	−8.98%
BT_Forward_samples_224.083	142.90	75.43	265.78	−7.75%	−18.98%	−48.50%	−5.13%
BT_Forward_samples_242.342	129.85	74.43	239.96	−16.17%	−26.37%	−53.20%	−13.79%
BT_Forward_samples_261.018	137.98	75.69	248.07	−10.92%	−21.77%	−50.27%	−8.39%
BT_Forward_samples_279.347	138.79	75.28	258.49	−10.40%	−21.31%	−49.98%	−7.86%
BT_Forward_samples_298.020	137.23	78.71	242.88	−11.41%	−22.19%	−50.55%	−8.89%
VolTime	150.63	33.83	358.66	−2.76%	−14.60%	−45.72%	0.00%
2	154.90	68.62	389.15	0.00%	−12.17%	−44.18%	+2.84%
4	176.37	141.66	273.58	+13.86%	0.00%	−36.44%	+17.09%
MarketOrder	277.49	246.48	158.66	+79.14%	+57.33%	0.00%	+84.22%

Table 11: Performance growth of the final `BT_Forward`-agent.
Average performance over the test period may 2017, currency pair USDT/BTC.

Figure 20 visualized the `BT_Forward`-agents performance evolution over the training phase. The horizontal lines refer to baselines and previously achieved performances.

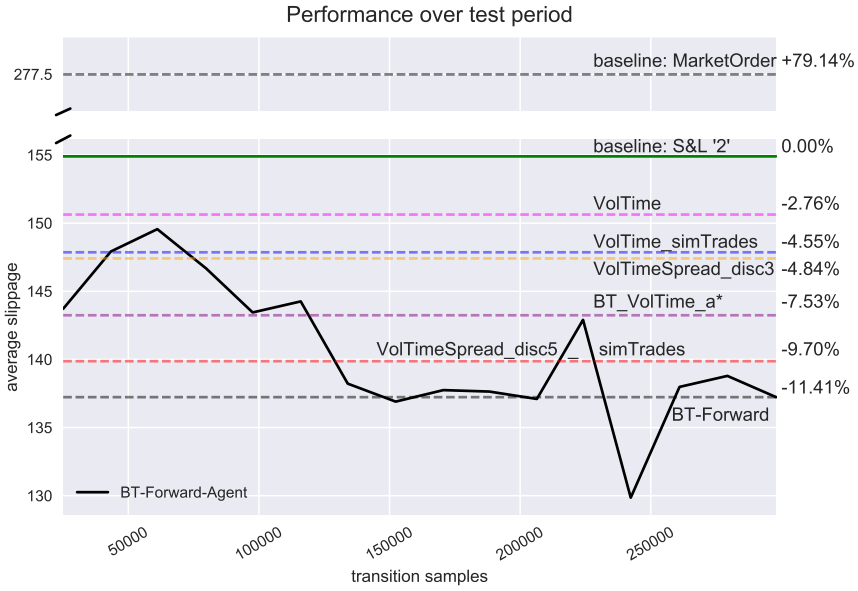


Figure 20: The `BT_Forward`-agent performance on the test period.

6 Conclusion

ToDo

A Appendix

A.1 Additional Market Variables

	slippage	med	std	perf_2	perf_4	perf_M	perf_VolTime
center_orig_disc3	149.57	46.15	420.14	−3.44%	−15.19%	−46.10%	−0.70%
center_orig_disc5	161.70	37.83	450.36	+4.39%	−8.32%	−41.73%	+7.35%
center_orig_disc9	161.65	38.83	450.40	+4.36%	−8.35%	−41.74%	+7.32%
marketPrice_buy_worst_disc3	148.35	41.26	361.17	−4.23%	−15.89%	−46.54%	−1.51%
marketPrice_buy_worst_disc5	146.83	39.62	351.77	−5.21%	−16.75%	−47.08%	−2.52%
marketPrice_buy_worst_disc9	150.53	39.66	368.95	−2.82%	−14.65%	−45.75%	−0.07%
marketPrice_sell_worst_disc3	146.72	41.91	385.13	−5.28%	−16.81%	−47.13%	−2.59%
marketPrice_sell_worst_disc5	148.94	42.71	388.86	−3.85%	−15.55%	−46.33%	−1.12%
marketPrice_sell_worst_disc9	149.28	45.05	390.19	−3.63%	−15.36%	−46.20%	−0.89%
marketPrice_spread_disc3	150.27	40.95	355.46	−2.99%	−14.80%	−45.84%	−0.23%
marketPrice_spread_disc5	150.46	44.15	371.32	−2.87%	−14.69%	−45.78%	−0.11%
marketPrice_spread_disc9	151.98	47.12	372.53	−1.89%	−13.83%	−45.23%	+0.90%
marketPrice_imbalance_disc3	148.42	50.44	358.49	−4.19%	−15.85%	−46.51%	−1.47%
marketPrice_imbalance_disc5	150.10	68.68	336.60	−3.10%	−14.90%	−45.91%	−0.35%
marketPrice_imbalance_disc9	150.50	64.44	357.22	−2.84%	−14.67%	−45.76%	−0.09%
sharecount_buy_disc3	149.57	46.15	420.14	−3.44%	−15.19%	−46.10%	−0.70%
sharecount_buy_disc5	160.73	37.78	449.22	+3.76%	−8.87%	−42.08%	+6.70%
sharecount_buy_disc9	161.65	38.83	450.40	+4.36%	−8.35%	−41.74%	+7.32%
sharecount_imbalance_disc3	148.57	40.68	353.91	−4.08%	−15.76%	−46.46%	−1.36%
sharecount_imbalance_disc5	148.03	40.49	372.86	−4.44%	−16.07%	−46.65%	−1.73%
sharecount_imbalance_disc9	153.27	47.28	380.38	−1.05%	−13.10%	−44.77%	+1.75%
sharecount_sell_disc3	149.57	46.15	420.14	−3.44%	−15.19%	−46.10%	−0.70%
sharecount_sell_disc5	151.50	47.49	425.35	−2.20%	−14.10%	−45.40%	+0.58%
sharecount_sell_disc9	150.37	46.15	424.22	−2.93%	−14.75%	−45.81%	−0.17%
sharecount_spread_disc3	147.57	40.95	350.33	−4.73%	−16.33%	−46.82%	−2.03%
sharecount_spread_disc5	148.93	40.84	352.12	−3.86%	−15.56%	−46.33%	−1.13%
sharecount_spread_disc9	146.61	39.62	373.00	−5.35%	−16.87%	−47.16%	−2.66%
spread_disc3	147.41	35.48	370.96	−4.84%	−16.42%	−46.88%	−2.14%
spread_disc5	141.07	37.39	349.72	−8.93%	−20.02%	−49.16%	−6.34%
spread_disc9	142.80	36.69	364.76	−7.81%	−19.03%	−48.54%	−5.20%
ob_direction_disc3	65.80	81.55	335.76	−57.52%	−62.69%	−76.29%	−56.32%
ob_direction_disc5	61.99	97.52	319.84	−59.98%	−64.85%	−77.66%	−58.84%
ob_direction_disc9	57.20	86.64	321.42	−63.07%	−67.57%	−79.38%	−62.02%
future_center5_disc3	133.58	40.40	377.88	−13.76%	−24.26%	−51.86%	−11.31%
future_center5_disc5	141.61	41.79	389.58	−8.58%	−19.71%	−48.97%	−5.98%
future_center15_disc3	107.74	31.91	364.92	−30.44%	−38.91%	−61.17%	−28.47%
future_center15_disc5	133.78	40.90	405.88	−13.63%	−24.15%	−51.79%	−11.18%
future_center15_disc9	121.11	47.41	411.10	−21.82%	−31.33%	−56.36%	−19.60%
future_center60_disc3	139.20	36.02	398.05	−10.14%	−21.07%	−49.83%	−7.58%
future_center60_disc5	131.85	47.63	391.46	−14.88%	−25.25%	−52.49%	−12.47%
future_center60_disc9	131.57	48.52	422.35	−15.06%	−25.40%	−52.59%	−12.65%
VolTime	150.63	33.83	358.66	−2.76%	−14.60%	−45.72%	0.00%
2	154.90	68.62	389.15	0.00%	−12.17%	−44.18%	+2.84%
4	176.37	141.66	273.58	+13.86%	0.00%	−36.44%	+17.09%
MarketOrder	277.49	246.48	158.66	+79.14%	+57.33%	0.00%	+84.22%

Table 12: Evaluating the impact of additional market variables.
Average performance over the test period may 2017, currency pair USDT/BTC.
See Section 5.2.3 for the actual experiment description. This is the full version of Table 5.

A.2 Constant Market Variables

	slippage	med	std	perf_2	perf_4	perf_M	perf_VolTime
center_orig_disc3	158.96	48.93	431.17	2.62%	−9.87%	−42.71%	+5.54%
center_orig_disc5	150.37	47.01	422.14	−2.92%	−14.74%	−45.81%	−0.17%
center_orig_disc9	149.76	46.15	420.67	−3.32%	−15.09%	−46.03%	−0.57%
marketPrice_buy_worst_disc3	150.27	44.67	398.78	−2.99%	−14.80%	−45.85%	−0.24%
marketPrice_buy_worst_disc5	150.74	50.15	402.13	−2.69%	−14.54%	−45.68%	+0.07%
marketPrice_buy_worst_disc9	155.99	44.67	416.56	+0.70%	−11.56%	−43.79%	+3.56%
marketPrice_sell_worst_disc3	150.13	41.26	352.48	−3.08%	−14.88%	−45.90%	−0.33%
marketPrice_sell_worst_disc5	157.56	43.95	379.33	+1.72%	−10.66%	−43.22%	+4.61%
marketPrice_sell_worst_disc9	147.90	48.20	357.93	−4.52%	−16.14%	−46.70%	−1.81%
marketPrice_spread_disc3	152.15	51.32	391.87	−1.78%	−13.74%	−45.17%	+1.01%
marketPrice_spread_disc5	152.19	53.07	398.10	−1.75%	−13.71%	−45.15%	+1.04%
marketPrice_spread_disc9	150.74	41.60	382.24	−2.69%	−14.53%	−45.68%	+0.08%
marketPrice_imbalance_disc3	148.04	51.06	379.25	−4.43%	−16.06%	−46.65%	−1.71%
marketPrice_imbalance_disc5	147.16	47.03	383.19	−5.00%	−16.57%	−46.97%	−2.30%
marketPrice_imbalance_disc9	146.85	54.71	375.38	−5.20%	−16.74%	−47.08%	−2.51%
sharecount_buy_disc3	149.57	46.15	420.14	−3.44%	−15.19%	−46.10%	−0.70%
sharecount_buy_disc5	150.37	47.01	422.14	−2.92%	−14.74%	−45.81%	−0.17%
sharecount_buy_disc9	149.76	46.15	420.67	−3.32%	−15.09%	−46.03%	−0.57%
sharecount_imbalance_disc3	151.20	52.13	395.90	−2.39%	−14.27%	−45.51%	+0.38%
sharecount_imbalance_disc5	152.44	53.41	400.89	−1.59%	−13.57%	−45.06%	+1.20%
sharecount_imbalance_disc9	154.90	57.56	395.50	−0.00%	−12.18%	−44.18%	+2.84%
sharecount_sell_disc3	149.57	46.15	420.14	−3.44%	−15.19%	−46.10%	−0.70%
sharecount_sell_disc5	150.37	47.01	422.14	−2.92%	−14.74%	−45.81%	−0.17%
sharecount_sell_disc9	149.53	46.15	420.50	−3.47%	−15.22%	−46.11%	−0.73%
sharecount_spread_disc3	150.78	48.23	419.23	−2.66%	−14.51%	−45.66%	+0.10%
sharecount_spread_disc5	149.23	53.41	402.65	−3.66%	−15.39%	−46.22%	−0.93%
sharecount_spread_disc9	148.65	52.13	408.66	−4.04%	−15.72%	−46.43%	−1.31%
spread_disc3	148.92	41.70	361.12	−3.86%	−15.57%	−46.33%	−1.14%
spread_disc5	143.56	40.78	350.74	−7.32%	−18.60%	−48.26%	−4.69%
spread_disc9	145.54	43.25	352.05	−6.04%	−17.48%	−47.55%	−3.38%
VolTime	150.63	33.83	358.66	−2.76%	−14.60%	−45.72%	0.00%
2	154.90	68.62	389.15	0.00%	−12.17%	−44.18%	+2.84%
4	176.37	141.66	273.58	+13.86%	0.00%	−36.44%	+17.09%
MarketOrder	277.49	246.48	158.66	+79.14%	+57.33%	0.00%	+84.22%

Table 13: Evaluating the impact of constant market variables.
Average performance over the test period may 2017, currency pair USDT/BTC.
See Section 5.2.3 for the actual experiment description. This is the full version of Table 7.

A.3 Simulation of preceding trades

	slippage	med	std	perf_2	perf_4	perf_M	perf_VolTime
center_orig_disc3	158.97	48.93	431.16	+2.62%	−9.87%	−42.71%	+5.54%
center_orig_disc5	150.47	47.49	422.12	−2.86%	−14.69%	−45.78%	−0.11%
center_orig_disc9	149.76	46.15	420.66	−3.32%	−15.09%	−46.03%	−0.57%
marketPrice_buy_worst_disc3	148.49	45.78	396.35	−4.14%	−15.81%	−46.49%	−1.42%
marketPrice_buy_worst_disc5	148.33	47.46	400.73	−4.24%	−15.90%	−46.55%	−1.53%
marketPrice_buy_worst_disc9	154.36	39.16	414.24	−0.35%	−12.48%	−44.37%	+2.48%
marketPrice_sell_worst_disc3	150.23	40.84	351.30	−3.02%	−14.82%	−45.86%	−0.26%
marketPrice_sell_worst_disc5	154.06	42.10	370.91	−0.54%	−12.65%	−44.48%	+2.28%
marketPrice_sell_worst_disc9	152.07	49.16	362.68	−1.83%	−13.78%	−45.20%	+0.96%
marketPrice_spread_disc3	151.10	52.13	390.08	−2.45%	−14.33%	−45.55%	+0.31%
marketPrice_spread_disc5	151.12	53.07	395.68	−2.44%	−14.32%	−45.54%	+0.33%
marketPrice_spread_disc9	148.45	41.60	380.40	−4.16%	−15.83%	−46.50%	−1.44%
marketPrice_imbalance_disc3	145.82	47.88	374.09	−5.86%	−17.32%	−47.45%	−3.19%
marketPrice_imbalance_disc5	143.86	47.05	375.43	−7.13%	−18.44%	−48.16%	−4.49%
marketPrice_imbalance_disc9	144.90	51.39	365.94	−6.45%	−17.84%	−47.78%	−3.80%
sharecount_buy_disc3	149.57	46.15	420.14	−3.44%	−15.19%	−46.10%	−0.70%
sharecount_buy_disc5	150.47	47.49	422.12	−2.86%	−14.69%	−45.78%	−0.11%
sharecount_buy_disc9	149.76	46.15	420.67	−3.32%	−15.09%	−46.03%	−0.57%
sharecount_imbalance_disc3	153.26	53.41	398.42	−1.06%	−13.10%	−44.77%	+1.75%
sharecount_imbalance_disc5	151.92	53.41	400.69	−1.92%	−13.86%	−45.25%	+0.86%
sharecount_imbalance_disc9	154.33	57.43	394.25	−0.37%	−12.49%	−44.38%	+2.46%
sharecount_sell_disc3	149.57	46.15	420.14	−3.44%	−15.19%	−46.10%	−0.70%
sharecount_sell_disc5	149.93	47.01	420.70	−3.21%	−14.99%	−45.97%	−0.47%
sharecount_sell_disc9	149.53	46.15	420.50	−3.47%	−15.22%	−46.11%	−0.73%
sharecount_spread_disc3	150.35	48.23	418.91	−2.94%	−14.76%	−45.82%	−0.19%
sharecount_spread_disc5	149.36	53.41	402.86	−3.58%	−15.31%	−46.17%	−0.84%
sharecount_spread_disc9	148.47	50.52	410.27	−4.15%	−15.82%	−46.50%	−1.43%
spread_disc3	145.63	41.89	346.00	−5.98%	−17.43%	−47.52%	−3.32%
spread_disc5	139.87	41.22	336.30	−9.70%	−20.69%	−49.59%	−7.14%
spread_disc9	141.30	42.80	336.88	−8.78%	−19.89%	−49.08%	−6.19%
ob_direction_disc3	69.01	101.49	323.00	−55.45%	−60.87%	−75.13%	−54.18%
ob_direction_disc5	62.03	97.52	320.76	−59.95%	−64.83%	−77.64%	−58.82%
ob_direction_disc9	57.74	87.92	321.20	−62.73%	−67.26%	−79.19%	−61.67%
VolTime_simulatedTrades	147.86	42.10	346.17	−4.55%	−16.17%	−46.72%	−1.84%
VolTime	150.63	33.83	358.66	−2.76%	−14.60%	−45.72%	0.00%
2	154.90	68.62	389.15	0.00%	−12.17%	−44.18%	+2.84%
4	176.37	141.66	273.58	+13.86%	0.00%	−36.44%	+17.09%
MarketOrder	277.49	246.48	158.66	+79.14%	+57.33%	0.00%	+84.22%

Table 14: Evaluating the impact of incorporating preceding trades.
Average performance over the test period may 2017, currency pair USDT/BTC.
See Section 5.2.4 for the actual experiment description. This is the full version of Table 8.

B Glossary

OTS Orderbook Trading Simulator

RL Reinforcement Learning

SL Supervised Learning

BT BatchTree

MDP Markov Decision Process

S&L Submit & Leave Strategy

S&R Submit & Revise Strategy

Eurex European Exchange

NASDAQ National Association of Securities Dealers Automated Quotations (American Stock Exchange))

C References

[1] R. Bellman. *A Markovian Decision Process*. In: *Indiana Univ. Math. J.* Vol. 6 (4 1957), pp. 679–684. ISSN: 0022-2518.

[2] R. Bellman. *Dynamic Programming*. 1957.

[3] J. Davis. *The Crypto-Currency: Bitcoin and its mysterious inventor*. http://www.newyorker.com/reporting/2011/10/10/111010fa_fact_davis. [Online; accessed 05-July-2017]. 2011.

[4] D. Ernst, P. Geurts, and L. Wehenkel. *Tree-Based Batch Mode Reinforcement Learning*. In: *J. Mach. Learn. Res.* Vol. 6 (Dec. 2005), pp. 503–556. ISSN: 1532-4435.

[5] P. Klemperer. *Auction Theory: A Guide to the Literature*. In: *JOURNAL OF ECONOMIC SURVEYS*, Vol. 13, No. 3 (1999).

[6] S. Lange, T. Gabel, and M. Riedmiller. *Batch Reinforcement Learning*.

[7] T. Lee. *Five surprising facts about Bitcoin*. https://www.washingtonpost.com/news/the-switch/wp/2013/08/21/five-surprising-facts-about-bitcoin-2. [Online; accessed 05-July-2017]. 2013.

[8] M. Minsky. *Steps Toward Artificial Intelligence*. 1960.

[9] S. Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. http://bitcoin.org/bitcoin.pdf. 2009.

[10] Y. Nevmyvaka, Y. Feng, and M. Kearns. *Reinforcement Learning for Optimized Trade Execution*. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: ACM, 2006, pp. 673–680. ISBN: 1-59593-383-2.

[11] Y. Nevmyvaka et al. *Electronic Trading in Order-Driven Markets: Efficient Execution*. In: *CEC*. 2005.

[12] Poloniex. *Poloniex Digital Asset Exchange*. http://www.poloniex.com. [Online; accessed 20-June-2017]. 2010.

[13] M. Riedmiller. *Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method*. In: *Proceedings of the 16th European Conference on Machine Learning*. ECML'05. Porto, Portugal: Springer-Verlag, 2005, pp. 317–328. ISBN: 3-540-29243-8, 978-3-540-29243-2.

[14] E. Staff. *Blockchains: The great chain of being sure about things*. https://www.economist.com/news/briefing/21677228-technology-behind-bitcoin-lets-people-who-do-not-know-or-trust-each-other-build-dependable. [Online; accessed 05-July-2017]. 2015.

[15] R. S. Sutton and A. G. Barto. *Reinforcement Learning I: Introduction*. 1998.

[16] C. J.C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis. Cambridge, UK: King's College, 1989.

[17] C. J.C. H. Watkins and P. Dayan. *Q-learning*. In: *Machine Learning*. 1992, pp. 279–292.

D List of Figures

1 BTC/USDT and USDT/Pound volatility compared. 8
2 The discrete RL scenario defined by Sutton and Barto[15] 9
3 Batch Reinforcement Learning 13
4 Growing Batch Reinforcement Learning 14
5 Historic center prices between Nov, 10th 2016 and May, 31 2017, as fetched from Poloniex. 16
6 A simple visualization of an limit orderbook. 17
7 An orderbook window over a period of 60 minutes. 19
8 Figure describing the masterbook adjustments as a graph?! 20
9 Visualization of an exemplary trading strategy. 21
10 Sample of a *curious masterbook shape*. 21
11 State-Action function, visualized after the first training round. 26
12 State-Action function, visualized after the second training round. 26
13 Final State-Action function. 27
14 Non-linear slippage growth. 30
15 Different shaped masterbooks at `t=45`. 31
16 Concurrent to declining slippage, the optimal S&L actions become less aggressive as time passes. 36
17 Average costs induced by the varying S&L actions over the full training period. In average, action 4 performed best. 36
18 A slight increase in aggression, if preceding trades are incorporated. 40
19 The BT-agent generalizes to continuous states. 42
20 The `BT_Forward`-agent performance on the test period. 43

E List of Tables

1 Exemplary snapshot of a limit orderbook 5
2 Trading history, as returned after four consecutive calls of `ots.trade()`. 20
3 Action $a = 1.4$ translates into $limit = 28.7 + 1.4 = 30.1$. 24
4 Evaluating the impact of different limit base levels. 35
5 Evaluating the impact of additional market variables. 38
6 Evaluating the impact of look-ahead features. 38
7 Evaluating the impact of constant market variables. 39
8 Evaluating the impact of incorporating preceding trades. 40
9 Evaluating the impact of action-consequence knowledge. 41
10 Evaluating the performance of function approximators. 42
11 Performance growth of the final `BT_Forward`-agent. 43
12 Full version of Table 5 46
13 Full version of Table 7 47

14 Full version of Table 8 48

F List of Code

1 Data fetched from Poloniex via HTTP GET request 15
2 OrderbookContainer 17

List of Pseudo Code

1 Value Iteration. 11
2 Q-learning [16]. 12
3 Fitted Q Iteration [4]. 14
4 Optimal_strategy, as described in [10]. 25
5 Forward sampling and learning approach. 29