Master's Thesis

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

Axel Perschmann

June 26, 2017

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

Writing period

12. 01. 2016 – 07. 31. 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, June 26, 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 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. [1] 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 [1] 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

bla

2.1 Exchange Markets, Bitcoins and Trading Basics

An exchange is a market, where financial instruments are sold and bought. It is typically organized by a broker, which can be both, an individual or a firm, executing buy and sell orders on behalf of dealers for a certain fee or commission. The respective prices are determined by the current market situations, in particular by supply and demand.

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.

Bitcoin exchange market x
E. g. Poloniex, ...

Most modern markets are usually fully electronic.....

2.1.1 Ask and Bid

Most exchange markets function after the so called auction market model, 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 ist called the *ask price*. Together they represent the current price at which an instrument is traded.

2.1.2 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.

	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.

1 Acme Internet Widget Company

2.1.3 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 liked 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.

2.1.4 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.

2.1.5 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 [2] Nevmyvaka et al. introduce a *Submit & Leave* strategy, 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 [1], where the order limit may be revised at discrete time steps, depending on trade progress and market changes.

2.2 Bitcoins

2.2.1 Marktreaktionen

"Bitcoin ist eine Währung, die äußerst sensibel auf Nachrichten reagiert. Begründet wird dies vor allem durch die Möglichkeit, ständig am Markt teilnehmen zu können: Es gibt keine zentrale Ausgabestelle mit geregelten Handelszeiten, an die man gebunden ist."

"Auch die Tatsache, dass viele Anfänger in Bitcoins investieren, führte bereits in der Vergangenheit zu den ein oder anderen Panikverkäufen. Wer sein Geld in Bitcoins investieren möchte, kann die meist lukrative Möglichkeit nutzen, sollte sich jedoch regelmäßig über Marktveränderungen informieren.

Da viele Investoren schnell auf Meldungen reagieren, kann es innerhalb von Stunden zu großen Kursverlusten oder Gewinnen kommen." https://www.btc-echo.de/bitcoin-trading-tipps-prinzipien-des-bicoin-handels_2015022502/

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 vectors $x_i \in X$ 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 $g_w()$ (or it's parameters w), such that $g_w(x_i) \approx y_i | i \leq N$. The learned function should ideally avoid overfitting by finding a generalization to previously unseen data.

Markow Decision Process

Value Function and Bellmann Equation

Value Iteration

Q-Learning

2.3.1 Logistic Regression

bla

2.3.2 Random Forest

bla

2.4 Reinforcement Learning

Reinforcement learning is a subdomain of machine learning, where strategies are learned by an *agent* interacting with its environment. Rather than learning from labeled training data, the agent applies a *trial and error* pattern and exploits external rewards to find actions, maximizing his expected future reward.

2.4.1 Dynamic backward programming

bla

2.4.2 Tree-Based Batch Mode Reinforcement Learning

bla

2.4.3 Neural Fitted Q-learning

bla

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 2.1.2) 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 [3] 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 5000 from Poloniex via HTTP GET requests. The volume of recorded orderbook snapshots for nine distinct currency pairs[2] 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}
```

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

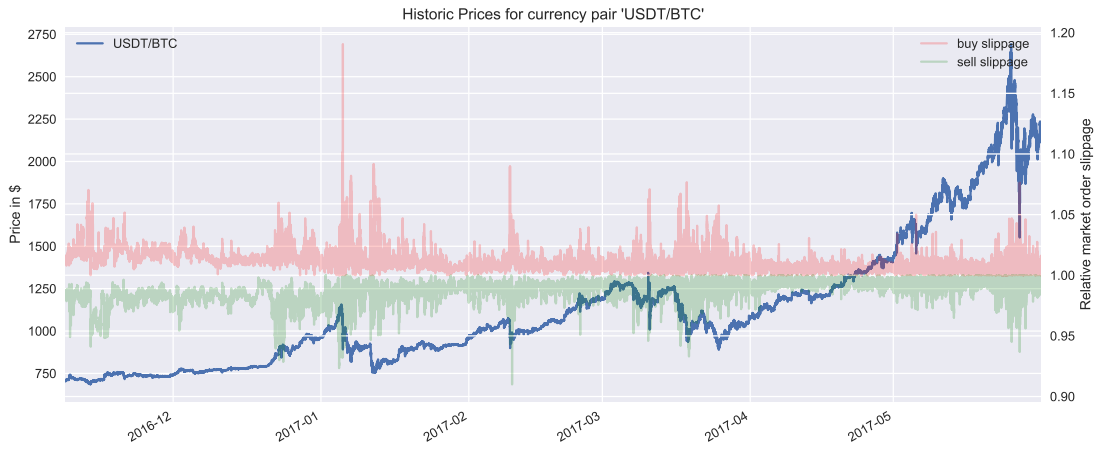

Figure 1: Historic center prices between Nov, 10th 2016 and Mar, 31 2017, as fetched from Poloniex.

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.

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

- 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 2 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()
```

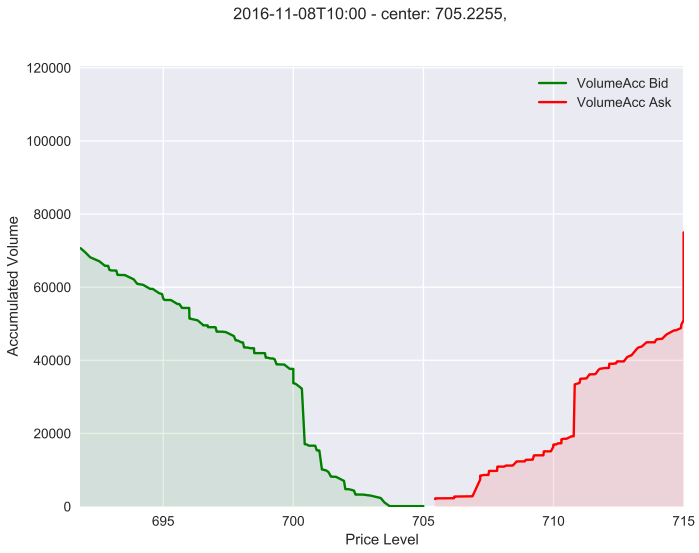

Figure 2: A simple visualization of an limit orderbook.

3.3 Simulator

The OTS framework serves as basis 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 visa 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. 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 3 visualizes a 60 minutes long orderbook window, where the solid red lines mark the *average* (a) and *worst* (b) 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.

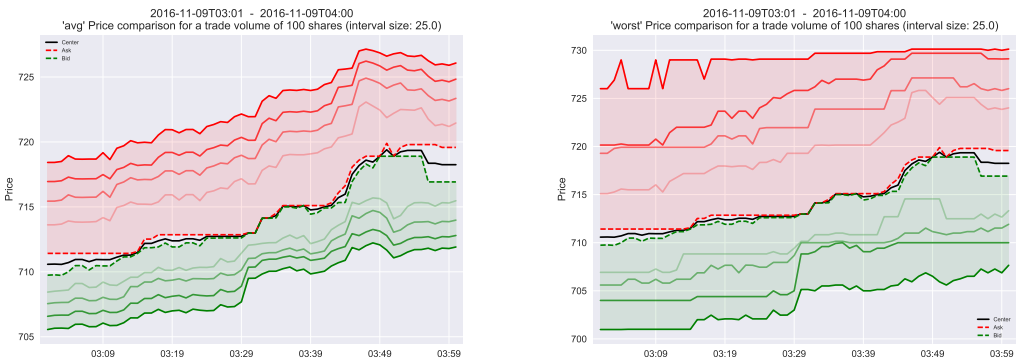

Figure 3: An orderbook window over a period of 60 minutes.

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

Figure 4: 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 and cash 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 5. 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.

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

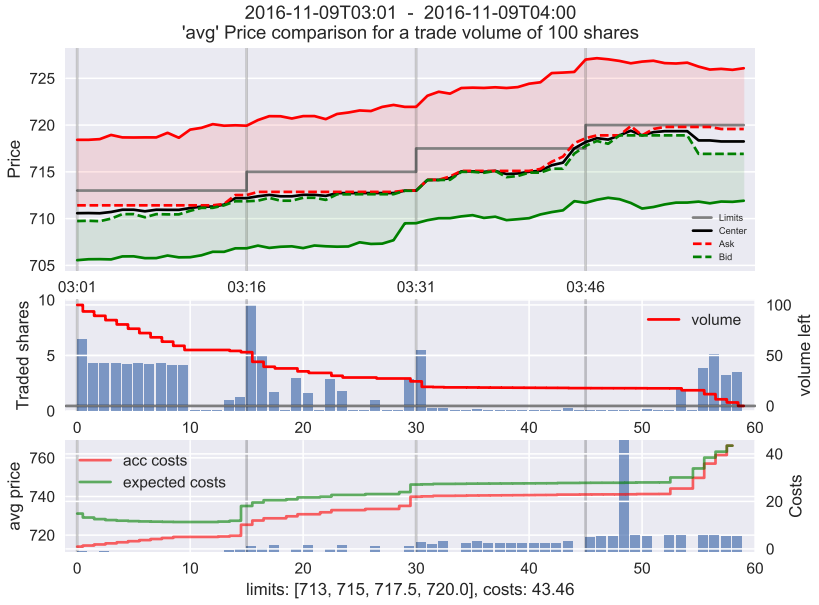

Figure 5: Visualization of an exemplary trading strategy.

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 6: 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 2.1.2) 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 2.1.4, 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

This chapter describes an Reinforcement Learning (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. [1], but modified in detail.

4.1 Original Algorithm

The original algorithm claims to find optimal limits from a discretized state space, representing trade progress (i. e. *remaining time* and *remaining inventory*) and additional market variables. While they achieved an impressive 50% gain over more simple trading strategies, they left some room for 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.

Nevmyvaka et al. [1] fused Q-learning and dynamic programming to learn a state-based strategy over the first year of their data in a brute force manner.

4.1.1 State space

The state space consists of various discrete variables, describing the current trade progress (*private variables*) and the current market situation as observable from orderbook data (*market variables*). The two private variables `time` and `volume` make the base for all subsequent experiments, while various additional market variables were enclosed to examine their impact on a valuable decision making.

As such, each `state` $s \in < time, volume, o_1, o_2, ... >$ forms a vector of at least two 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.

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.

All market variables were discretized into 0 (low), 1 (medium) and 2 (high), while the concrete category mapping process was not further described.

4.1.2 Action space

Actions define the level of trading aggression to be performed. An 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).

In case of the market situation as shown in Table 3, a buy order with an aggressive action $a = 1.4$ would translate into $limit = 28.7 + 1.4 = 30.1$. 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 is not further specified.

4.1.3 Costs

Costs are defined as the slippage induced from the previously chosen action. 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:

$$cost = (avg\_paid - initial\_center) * volume\_traded \tag{1}$$

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

Since the complete trade execution happens within a finite time horizon and full execution of the `volume` is mandatory, partial costs, as observed after the individual `trading_periods`, can simply be summed up without any discounting.

4.1.4 Learning Algorithm

In order to learn the optimal limit for each situation, orderbook windows are examined in a backward, brute-force manner as described in Listing 3. 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.

Listing 3: Brute-Force strategy learning approach as described in [1].

```
Optimal_strategy(V, H, T, I, L)
    For t=T to 0
        While(not end of data)
            Transform (orderbook) -> o_1 ... o_R
            For i =0 to I
                For a = 0 to L
                    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)
    Select the highest-payout action argmax cost(y, p) in every state y to output ↩
        optimal policy
```

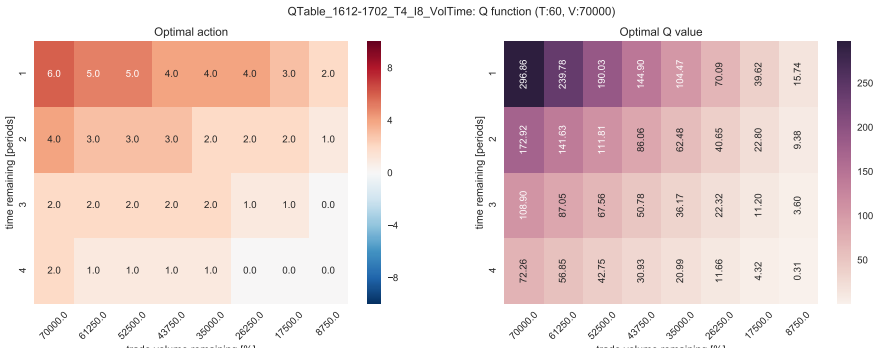

Figure 7: Heatmap.

While the running time depends only on the resolution of the private variables and the chosen action space, it is approximately independent of the number of market variables chosen. The cost update rule is given below:

$$cost(x,a) = \frac{n}{n+1} cost(x,a) + \frac{1}{n+1}[cost_{im}(x,a) + \arg\min_{p} cost(y,p)] \tag{3}$$

4.1.5 Discussion

- Discrete Actions / Lookup-table
- Vulnerable to seldom observed market states
- Why crossing the spread?! Start from ask, rather than bid!
  Influence/dependence on Spread (+8% claimed!) as MarketVariable seems obvious here.

4.1.6 Improvements tried

- actions represent deviation from ask, not bid

- Forward Sampling
  - Markov Property violated
  - Realistic samples, no rounding.
- Function Approximation
  - RandomForest (BatchTree Agent)
  - NN Agent
- Different Market Variables:
  - Immediate Slippage/MarketPrice Imbalance
  - MarketPrice Spread (buy vs. sell)
- Cost function
  - Slippage based on initial_center
  - Improvement over MarketPrice??

4.2 RL Agents

Various type

4.2.1 QTable Agent

bla

4.2.2 BatchTree Agent

bla

4.2.3 NN Agent

bla

5 Experiments

This chapter is about a concrete, real world example.

5.1 Backward approach

5.1.1 QTable Agent + additional orderbook features

bla

5.1.2 QTable Agent + additional technical indicators

5.1.3 QTable Agent vs. BatchTree Agent vs. NN Agent

bla

5.2 Forward approach

bla

5.2.1 Motivation

Markov Assumption is wrong: States do depend on path chosen before.
Data efficency. Learning with fewer samples
No need for discretization of volume.

5.2.2 Results

bla

6 Conclusion

Due to the volume of data, the real world example shown in **??** would have been a tough job on any single local workstation or students notebook. When analyzing big data, it is a great relief or even an inevitable thing to use a cluster of computer for distributed storage and data processing.

Hadoop is a great and powerful cluster framework and R is a highly popular and well-advanced programming language for statisticians. In a world of ever growing data, Hadoop and R make a perfect fit. Both combined, the mightful analytic capabilities of R can be applied to big data.

A Glossary

OTS Orderbook Trading Simulator

RL Reinforcement Learning

B References

[1] 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.

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

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

C List of Figures

D List of Tables

E Listings