university of groningen

faculty of science and engineering

Buy-or-Rent Decisions in a Two-Token Blockchain Network

Boris Winter

university of groningen / faculty of science and engineering

University of Groningen

Buy-or-Rent Decisions in a Two-Token Blockchain Network

Master's Thesis

To fulfill the requirements for the degree of
Master of Science in Artificial Intelligence
at the University of Groningen under the supervision of
Prof. dr. D. Grossi (Artificial Intelligence, University of Groningen / University of Amsterdam)
and
Dr. H.A. de Weerd (Artificial Intelligence, University of Groningen)

Boris Winter (s2774291)

August 31, 2022

Contents

Abstract

Blockchain technology has seen an incredible surge in popularity in recent years, and many different blockchain networks exist at present. VeChainThor is a blockchain network that uses a two-token model in which the main token generates gas tokens, and the gas tokens are used to pay for network use. This two-token model yields what we call the VeChain Decision Problem: users of the network need to decide whether they want to buy gas tokens directly, or if they want to buy main tokens to generate the gas tokens. In the field of online algorithms, the Ski Rental Problem refers to a well-studied collection of problems in which one needs to decide between renting or buying. In this exploratory research, we map the VeChain Decision Problem to the Ski Rental Problem while exploring the features that make the VeChain Decision Problem unique and more challenging to solve. Most importantly, we identify and model the uniquely direct relationship between user decisions and buy/rent prices that order-driven cryptocurrency exchanges provide through their limit order books. We consider deterministic and randomized online algorithms that optimally solve Ski Rental Problems, and we analyze their performance in the VeChain Decision Problem. Moreover, we explore the influence of the usage of these algorithms on the network adoption ratio and -speed of network users. Results suggest that the consistency in performance of deterministic algorithms makes them more appropriate for use in the VeChain Decision Problem than randomized algorithms. Results also show that trends in the buy-to-rent price ratio influence user performance and adoption behavior. Finally, simulations in a multi-agent setting suggest that user performance is affected by the strategies applied by other users. Future studies are encouraged to advance multi-agent research of this problem, and to improve the existing deterministic algorithms by incorporating price predictions.

1 Introduction

In 2008, an entity going by the name of Satoshi Nakamoto published the idea for Bitcoin [1]: a decentralized digital currency that exists on a peer-to-peer network, and works through what is now commonly referred to as blockchain technology [2]. Since the inception of Bitcoin in 2008, both Bitcoin and blockchain technology have seen an incredible surge in popularity. As a result of this popularity, many variations to Bitcoin have been created, and the current range of blockchain applications goes far beyond the digital money idea that popularized the technology. Collectively, Bitcoin and its variations are known as cryptocurrencies, which refers to the tokens that are used to execute payments on the blockchain. One of the most significant advances in the blockchain space occurred when blockchain was combined with smart contracts. The general notion of smart contracts was proposed by Nick Szabo in 1994, who defined a smart contract as "a computerized transaction protocol that executes the terms of a contract" [3]. The programmable payments that blockchain technology provides makes the technology ideal for implementing smart contracts on a larger scale [2]. The combination of blockchain and smart contracts has opened up a wide range of possibilities, and the development of blockchain applications has increased exponentially in recent years [4]. Many different fields are currently interested in utilising the benefits of blockchain networks. In the agriculture sector, blockchain technology has been proposed as a way to make the food supply chain more effective and less prone to fraudulent behavior [5]. The "Big Four" auditing firms are very involved with blockchain and aim to use the technology to streamline and secure their services [6]. Researchers studying blockchain technology envision large-scale applications, stating for example that "We can envision putting proof of existence of all legal documents, health records, and loyalty payments in the music industry, notary, private securities and marriage licenses in the blockchain." [2].

One specific blockchain network that has gotten a lot of interest from companies is the VeChainThor network, or simply VeChain [7]. VeChain is currently being used by Walmart China, who built their Walmart China Blockchain Traceability Platform on the VeChainThor network [8]. Another company that uses VeChain is PwC, who recently unveiled their Air Trace solution [9]. Governments have started to notice VeChain as well, with the San Marino government currently using the VeChainThor network in the certification method of their Digital Covid Certificates [10]. Although blockchain technology is still in its infancy, it is reasonable to assume that VeChain will remain a relevant blockchain network for the foreseeable future.

An interesting and quite unique feature of the VeChainThor network is that it uses a two-token model as opposed to the much more common models that use a single cryptocurrency token. In this two-token model, an energy token (called VTHO) is used to pay for network use. The main token (VET) is used as a value-transfer medium on the network and automatically generates VTHO [7]. This two-token design of the VeChainThor network yields what this study dubs the VeChain Decision Problem. The VeChain Decision Problem states that users of the VeChainThor network need to make a decision: do they buy VTHO every time they want to use the network? Or do they buy VET in order to generate the VTHO that they require to use the network?

In the field of online problems, the Ski Rental Problem refers to a well-studied collection of problems in which one needs to decide between renting or buying. The first example of such a problem in literature comes from the context of snoopy caching [11]. A lot of variations to the Ski Rental Problem exist, but the original formulation of the hypothetical problem can be summarized as follows: A skier wants to go skiing, but they do not know how many days they will ski. They need to decide between renting the skis for \$1 per day or buying the skis for \$$B$ [12]. The skier wants to minimize their cost, but making the correct decision is difficult because of the unknown amount of skiing days.

In the Ski Rental Problem, the amount of skiing days can be seen as the input of the problem. How-

ever, the issue with this input is that the information is incomplete. You only learn that there is another skiing day when that day arrives. Problems like this, in which there is incomplete information and input is received sequentially, are called online problems [13]. The most common way of solving online problems is through the use of online algorithms. Online algorithms are specifically designed to deal with the lack of information and the sequentially arriving input. Online algorithms can be deterministic or randomized. Deterministic online algorithms always produce the same output for the same input. A simple explanation of randomized algorithms is that they are probability density functions over deterministic algorithms [14]. Randomized algorithms therefore do not always produce the same output for the same input.

The performance of online algorithms is usually determined using the framework of competitive analysis. Competitive analysis compares the online algorithm against the optimal offline algorithm that does know the entire input in advance [13]. This comparison yields the performance metric used to compare online algorithms, which is called the competitive ratio (CR). When the Ski Rental Problem was introduced, a deterministic online algorithm was proposed along with it that was proven to achieve optimal performance in the domain of competitive analysis [11]. Since the introduction of the original Ski Rental Problem, many variations to the problem, as well as solutions to those variations have been proposed.

Looking back at the VeChain Decision Problem, it is clear that the decision of the network user about buying VTHO directly or buying VET to keep generating VTHO mirrors the buy-or-rent problems studied under the name of the Ski Rental Problem. This resemblance raises the question whether there is a need to study the VeChain Decision Problem as a variation of the Ski Rental Problem.

The Vechain Decision Problem has been discussed informally (though not under that name) by both users of the VeChainThor network and by people that speculate on the price of the VET and VTHO tokens. Their interest makes sense, since both of these parties stand to gain from understanding the relationship between the buy-or-rent-like decision and the token prices. Users of the VeChainThor network could benefit from implementing the best available solution to the VeChain Decision Problem in their decision making. In turn, it would be interesting for both parties to examine how the application of these solutions affects the commitment behavior of network users. Solutions might influence the point in time that users decide to adopt the network for the long-term. From the perspective of the VeChain community, there definitely exists a demand for more understanding about, and optimal solutions to, the VeChain Decision Problem.

Yet, studying the Vechain Decision Problem also makes a lot of sense from the perspective of the field of online problems. If the VeChain Decision Problem can be mapped successfully to the Ski Rental Problem, the VeChain Decision Problem can be seen as an important extension of the entire set of Ski Rental Problems. What makes this extension meaningful is the relationship that the buy and rent price would have with the result of the decision. In the cryptocurrency world, there exists a uniquely direct relationship between the decisions of blockchain network users and the price of cryptocurrency tokens. Users of a network need to acquire the tokens, and generally do so through the limit order books of order-driven cryptocurrency exchanges. This means that any time that a network user buys or sells tokens, they immediately and directly influence the price of that token. As a result, the price of the token will have changed for the next user that decides whether or not to buy the token. The straightforward relationship described here opens up the possibility of studying the bi-directional influence between usage and buy/rent prices in Ski Rental Problems, under reasonable assumptions. To recognize how uniquely direct this relationship is, one only needs to try to unravel the entire relationship between a single skier's decision to buy or rent, and the price of a pair of ski's.

Besides studying the relationship described in the previous section, the direct relationship between usage and prices opens up another possibility. As far as we are aware, the Ski Rental Problem has

so far only been considered in a single-agent setting. This makes sense, considering the extent of the assumptions about price influences that would be required for most real-world multi-agent scenarios. Again, the direct relationship between usage and prices that is innate to the VeChain Decision Problem changes this. The VeChain Decision Problem provides a reason to study, under reasonable assumptions, a multi-agent environment in which buy-or-rent decisions are made sequentially by the different users, and in which each decision affects all future decisions of all users.
To summarize, studying the VeChain Decision Problem is interesting both from the perspective of the field of online problems and from that of the VeChain community. Users of the VeChainThor blockchain require the solutions that online algorithms might offer them, and the field of online problems can use the VeChain Decision Problem as a reason to start multi-agent research on Ski Rental Problems under reasonable assumptions.

1.1 Research Questions

All of the above leads to the research questions that this thesis aims to investigate:

Q1. Can we create a mapping from the VeChain Decision Problem to the original Ski Rental Problem?

Q2. How do users of the VeChainThor network perform when they use solutions to existing Ski Rental Problems to guide their buy-or-rent decision making process?

Q3. What does the adoption behavior of network users look like when they adopt these solutions?

1.2 Thesis Outline

The remainder of the thesis is set up as follows. Section 2 goes over the background knowledge required to understand the methods. Section 3 defines the VeChain Decision Problem, and proceeds to explain the model and experiments that are used to study the problem. Section 4 presents the results of the experiments. Finally, Section 5 discusses the main contributions and limitations of this paper, as well as potential future work.

2 Preliminaries

2.1 Online Problems

Before attempting to map the VeChain Decision Problem to the Ski Rental Problem, it is important to understand the general notion of online problems. Online problems are optimization problems where there is incomplete information, meaning that the complete input is not known in advance [13]. Instead, the input is received sequentially while the program runs, and needs to be dealt with once it arrives. Many real-world examples of online problems can be found in technological fields, such as the list accessing problem [13] snoopy caching [11], and cloud cost optimization [12].

2.1.1 Online algorithms

While online problems can be solved by offline algorithms, the lack of complete information can affect the performance of these algorithms significantly. Online algorithms are a set of algorithms that are designed specifically for online problems, which means that they are able to make a sequence of decisions without complete knowledge of the future inputs [13]. Online algorithms themselves can be divided into *deterministic* and *randomized* online algorithms. Deterministic online algorithms are algorithms that always return the same output for the same input sequence. While deterministic algorithms are a valid solution for online problems, the game-theoretic nature of online problem analysis makes it so that randomization is important to achieve good performance [13]. This is where randomized online algorithms come in. Randomized online algorithms are basically probability distributions over deterministic online algorithms [14]. This means that the algorithm does not have a predetermined condition that the input is tested against. Instead, the condition is randomly selected at the start of the run, according to a probability density function.

2.1.2 Adversaries

Online problems are often modelled as a zero-sum game in which an online player uses the online algorithm to play against an adversary [13]. Here, the goal of the adversary is to — either with or without knowledge of the player's algorithm — generate input sequences that minimize the player's performance. There exist different adversaries for online algorithms [15]. However, in the case of the Ski Rental Problem, the oblivious adversary is often considered. The oblivious adversary must always construct the input before the game begins, and does not know the actions that the player will take. In some cases, the oblivious adversary is allowed to know the probability distribution used by the randomized algorithm [14]. In this study, we consider an oblivious adversary that has no knowledge about the player at all.

2.1.3 Competitive analysis

In the field of online algorithms, competitive analysis refers to the framework that is most commonly used to analyze the performance of the algorithms. Put simply, competitive analysis assesses online algorithms by comparing their performance to the performance of an optimal offline algorithm that has complete knowledge of the future [13]. The main performance measure used in competitive analysis is called the competitive ratio (CR). This is the ratio between the cost that is incurred when an online algorithm is used, and the cost that is incurred when the optimal offline algorithm is used [13]. The competitive ratio is a measure in the framework of worst-case performance [13]. This means that the competitive ratio is not concerned with the average ratio over all inputs, but with the supremum. In

other words, the competitive ratio is the ratio, over all inputs, where the online algorithm performs the worst compared to the optimal offline algorithm. If an algorithm attains a competitive ratio of c, it is called c-competitive [13]. The equation used for calculating the competitive ratio of an algorithm A (CR_A) is given in Equation 1 [14].

$$CR_A = \sup_{i \in I} \frac{Cost_A(i)}{Cost_{OPT}(i)} \tag{1}$$

Here, $Cost_A(i)$ is the cost of algorithm A for input i. Furthermore, $Cost_{OPT}(i)$ is the cost of the optimal deterministic algorithm for input $i \in I$. In the case of randomized algorithms, the fact that the algorithm uses randomization needs to be taken into account when determining the competitive ratio. The competitive ratio of randomized algorithms is therefore defined as the worst-case *expected* competitive ratio [13]. Equation 2 gives a formal definition of the worst-case expected competitive ratio. All definitions in the equation are equal to those of Equation 1, and $\mathbb{E}$ refers to the expected value.

$$CR_A = \sup_{i \in I} \frac{\mathbb{E}[Cost_A(i)]}{Cost_{OPT}(i)} \tag{2}$$

2.2 The Ski Rental Problem

The Ski Rental Problem is a classical problem in computer science, and online algorithms in particular. The original Ski-Rental Problem is defined as follows [12]: A skier has to choose between buying skis for \$$B$, or renting skis for \$1 per day. However, the skier does not know for how many days they will ski. The goal of the skier is to minimize their costs. If the price of buying would be equal to or less than the rent, buying would obviously be the better option. However, when buying is more expensive than renting, the decision becomes more complex. The Ski Rental Problem is defined so that \$$B$ is always larger than or equal to \$1.

Variations to the classical Ski Rental Problem have been suggested, and most of those have been solved optimally. These include: MultiSlope Ski Rental [16], Constrained Ski Rental [12], Dynamic Price Ski Rental [17], and Multi-Shop Ski Rental [18]. Like with many online problems, the Ski Rental Problem and its variations are usually modelled as a zero-sum game. In this game, the adversary determines how many days the skier will ski.

2.2.1 Optimal deterministic algorithm

For many of the proposed Ski-Rental Problem variations, optimal deterministic and randomized online algorithms have been found. These algorithms generally use the ratio between the total amount of money spent on rent and the cost of buying to inform their decision. In the case of the original Ski Rental Problem, the optimal deterministic solution is to rent until you have paid the buy price in rent, then buy [11]. In this study, we refer to this algorithm as DET. The pseudocode for the algorithm can be found in Algorithm 1. The worst-case competitive ratio of the optimal deterministic algorithm is proven to be 2 [11].

Algorithm 1 Optimal deterministic algorithm (DET)

```
1: state ← "RENTING"
2: c ← 0                                              ▷ Total money spent
3: while new ski day do
4:     if state == "RENTING" then
5:         if c == B then
6:             c ← c + B                               ▷ Buy skis
7:             state ← "BOUGHT"
8:         else
9:             c ← c + 1                               ▷ Rent skis
10:
```

2.2.2 Optimal randomized algorithm

The pseudocode for the optimal randomized online algorithm for the original Ski-Rental Problem is given in Algorithm 2. We refer to this algorithm as RAND. The RAND algorithm randomly selects a time $t \in [0,1]$ using the probability density function given in Equation 3, and buys if that time point is reached while they are still skiing [19]. The worst-case expected competitive ratio of the optimal randomized algorithm is $\frac{e}{e-1} \approx 1.58$ [19].

$$f(t) = e^t/(e-1) \tag{3}$$

Algorithm 2 Optimal randomized algorithm (RAND)

```
1: n ← maximum length of the game in days
2: state ← "RENTING"
3: c ← 0                                              ▷ Total money spent
4: pdf ← f(t) = e^t/(e−1)
5: Randomly select day t ∈ [0,n] according to pdf
6: while new ski day do
7:     if state == "RENTING" then
8:         if c == t then
9:             c ← c + B                               ▷ Buy skis
10:            state ← "BOUGHT"
11:        else
12:            c ← c + 1                               ▷ Rent skis
```

2.2.3 The Ski Rental Problem with price fluctuations

The Ski Rental Problem has also been studied in environments with fluctuating prices. In [20], the Ski Rental Problem was studied in the context of a fluctuating buy price. In that study, the rent price is dependent on the buy price and a buy-to-rent ratio μ, meaning that the ratio between the buy price and the rent price always remains the same. The authors from [20] allow themselves to limit the price fluctuations between two days based on the Consumer Price Index (CPI). This is achieved by setting a maximum fluctuation ratio $\alpha \geq 1$, which establishes the maximum price change between two time points. The actual price difference for each step p_{t+1}/p_t is chosen randomly from a uniform distribution where $\frac{1}{\alpha} \leq p_{t+1}/p_t \leq \alpha$ [20]. Note that authors of this study talk about the Online Leasing

Problem rather than the Ski Rental Problem. As these names refer to the same general problem, we refer to the problem presented in [20] as the CPI Ski Rental Problem to avoid confusion. After introducing the CPI Ski Rental Problem, the authors present an optimal online strategy for the problem, which they call Strategy A. Strategy A is given in Algorithm 3 [20].

Algorithm 3 Strategy A

Keep renting from the original time and buy at time t, where t satisfies the condition $R_t + r_t \geq \gamma P(t), \gamma \leq 1$.

Here, R_t refers to the total amount of money spent on rent at time t, r_t is the rent cost at time t, and $P(t)$ is the cost of buying at time t. The value of the γ parameter is set based on the problem setting that originates from the values of μ and α, so as to arrive at the optimal solution for those conditions. It was determined that the minimum competitive ratio (and therefore the optimal solution) for the CPI Ski Rental Problem was found when γ was set according to Equation 4, where T is the maximum time that satisfies the inequality $1 + \alpha + ... + \alpha^{T-1} \leq \mu$ [20].

$$\gamma = \frac{1 + \alpha + ... + \alpha^{T-1}}{\mu} \tag{4}$$

We note that Strategy A is equal to DET when μ goes to infinity. This is because this makes the difference between the sum in the numerator and μ go to zero. In turn, this means that γ goes to 1 and that the difference between R_t and $R_t + r_t$ becomes negligible. We also note that γ can never exceed 1 due to the inequality. Therefore, Strategy A buys either quicker than DET or at the same time. How much quicker Strategy A decides to buy depends on the values of α and μ. All else being equal, higher values of α result in the difference between the numerator and denominator in Equation 4 either increasing or staying the same (with the numerator always being less than the denominator, by definition). Consequently, conditions with higher maximum fluctuation ratios generally result in faster buying while conditions with lower maximum fluctuation ratios trend the behavior to that of DET. The opposite is true for the buy-to-rent ratio. All else being equal, higher values of μ generally make γ approach a value of 1 because the sum of α values can get closer to μ. Lower μ values generally result in smaller γ values because the α sum cannot get as close. Note that by definition of the (CPI) Ski Rental Problem, we always have $\mu \geq 1$.

2.3 Blockchain

Blockchain is a relatively new technology in which a distributed ledger is used in order to store and transact data. The specific workings of blockchain technology are not relevant to this study. What is relevant, is the following. Pretty much every blockchain network has a monetary token associated with it. This token is referred to as a cryptocurrency token, and is used as a form of payment on the network. Transacting on a blockchain network can simply mean transferring a token from one place to another. However, in most non-financial use of blockchain networks, transactions usually contain other types of data. Making transactions on a blockchain network is not free, so users of blockchain networks need to pay for the transactions that they make. Generally, this payment is in the form of the cryptocurrency token of the network that they use. For example, if you wish to send data (Bitcoin tokens or other) through the Bitcoin network, you will have to pay a transaction fee in Bitcoin to make that transaction.

2.4 VeChain

The VeChainThor network [7] (in this study also referred to as VeChain or the VeChain network) is a blockchain network that is designed with the goal of solving the problem of expensive and volatile transaction fees that popular blockchain networks such as Ethereum encounter [21]. VeChain determined that the volatile and unpredictable transaction costs were the main reason that larger enterprises were still unwilling to move their data to a blockchain network [7]. In an attempt to solve this issue, the VeChainThor network contains a two-token model. In this model, there is a "gas" token called VeThor (VTHO), and a main VeChain Token (VET). VeChain Tokens continuously generate VTHO, and VTHO tokens can be used to pay for transactions. Buying and holding VET gives additional benefits besides VTHO generation, in the form of voting rights. According to VeChain's tokenomics (the economics behind blockchain network tokens), there are two votes through which the cost of making transactions on the network can be influenced [7]. Entities that hold certain amounts of VET get a proportionate voting weight in these and other votes. There exists a fixed supply of 86,712,634,466 VET tokens, while there is no limit to the amount of VTHO tokens. The amount of VTHO tokens in circulation started at zero when the VeChainThor network was launched on 30-06-2018, but has increased steadily ever since. Each VET token generates 0.000432 VTHO per day, which means that a total of 37,459,858 VTHO are added to the network each day. At present, the cost of a simple token transfer is set to 0.21 VTHO, but sending data is generally more expensive (all depending on the amount and type of data transacted). When a user uses the VeChainThor network, 70% of the VTHO that they pay is destroyed, and the other 30% of the VTHO is given to the nodes that run the network. The VeChainThor network generates a new blockchain block every 10 seconds, which means that the state of the network is updated (for example by adding new transactions to the distributed ledger) every 10 seconds.

2.5 Cryptocurrency Markets

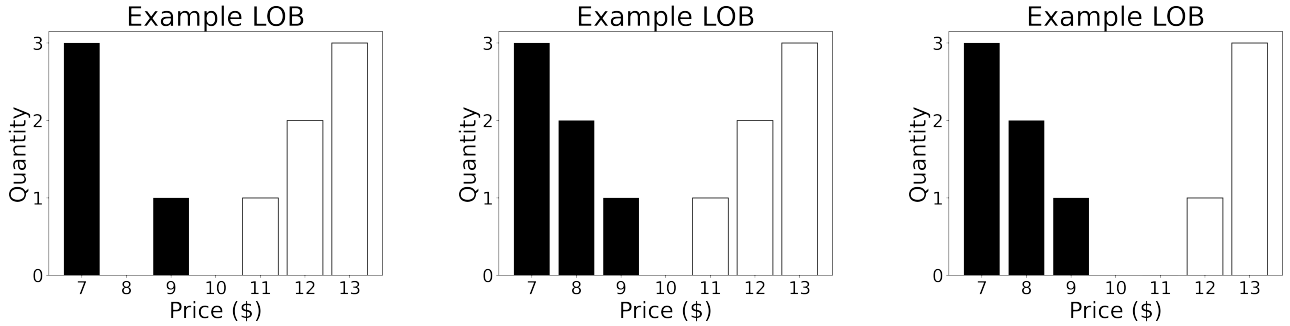
Before users of a blockchain network can make transactions, they obviously first need to acquire the tokens required to make transactions. Nowadays, cryptocurrency tokens like VeChain's VET and VTHO are very actively traded on online marketplaces. These marketplaces are known as cryptocurrency exchanges, and are the most common way of acquiring cryptocurrency tokens. One of the most popular cryptocurrency exchanges is called Binance [22]. The primary way in which exchanges like Binance make money is through transaction fees that users pay when they buy or sell tokens on the exchange. However, fee-less trading has become more common in recent years. All main cryptocurrency exchanges operate as order-driven markets. Order-driven markets are markets where all participants have direct access to the market [23], often through limit order books (Section 2.6). Because of the direct access, everyone buying or selling in an order-driven market influences the market directly. Order-driven markets stand in contrast to the more traditional quote-driven markets in which only designated entities present their buy and sell offers [23]. In cryptocurrency markets, tokens are often traded against other cryptocurrencies that are pegged to a real-world currency. The most well-known of these pegged cryptocurrencies is Tether [24], which is also known as USDT. Sometimes, it is easier to obtain data about cryptocurrencies in terms of USDT rather than U.S. Dollars. Moreover, order pairs using USDT (like for example the VET / USDT coupling) are among the most traded order pairs and can therefore be more representative of the entire market than other order pairs. For these reasons, this study sometimes uses prices expressed in USDT. Given that USDT is pegged to the U.S. Dollar, all mentions of USDT, dollar, and $ in this study may be seen as referring to the same thing.

2.6 Limit Order Books (LOB's)

In this section, we describe the general workings of limit order books. While doing so, we use and explain multiple relevant terms. All these terms are well established in literature and their definitions can be found in all basic textbooks that cover the subject. In order-driven markets, participants that want to buy or sell something can choose between two main order types: *market orders* and *limit orders*. When making a limit order, the participant indicates exactly how much of a product they want to buy/sell and for what price. A *limit order book (LOB)* can then be created by gathering all outstanding limit orders. This limit order book can be used as a representation of the market at that point in time. Collectively, limit orders might be referred to as *quotes*. However, a limit order book usually differentiates between limit buy orders and limit sell orders. Limit buy orders are referred to as *bids*, and the highest buy order is called the *best bid*. Limit sell orders are called *asks* and the *best ask* is the lowest sell order. The *mid price* is the average of the best bid and the best ask. Participants can also make market orders. When a participant makes a market order, they choose to buy or sell directly into the limit order book. This means that they accept to trade with the n most profitable limit orders that are available. Here, n is determined by how many of the limit orders are needed to buy/sell the required amount of product. The *liquidity* of a limit order book refers to the amount of product that is present in the order book. As one might imagine, liquidity is a very important factor in determining price movements. Quotes within a set price range will contain lower quantities when liquidity is low. This in turn means that lower quantities need to be bought/sold to move the price over that entire price range. The opposite is true when liquidity is high.

2.6.1 Example LOB

In order to illustrate the workings of a limit order book, some examples are provided here. Consider the example limit order book of the VTHO/USDT pair shown in Figure 1a. On the left side of the figure, the bids are shown in black. There is an outstanding bid to buy three tokens at a price of \$7 per token, and a bid to buy one token at \$9. Currently, nobody wants to buy tokens at a price of \$8. We also take note of the asks at prices \$11-\$13 (shown in white). Using the best bid (\$9) and the best ask (\$11), we determine the current mid price to be $(\$9 + \$11)/2 = \$10$.



(a) The initial limit order book (LOB).

(b) After Alice places a bid for VTHO at \$8.

(c) After Bob market buys 2 VTHO.

Figure 1: Three stages of an example limit order book (LOB). As is common for LOB's, the bid side is shown to the left of the LOB (in black), and the ask side is shown to the right (in white).

Alice is a user of the VeChainThor network, and they decide that they want to buy 2 VTHO tokens when the LOB looks like Figure 1a. However, they do not want to place a market order as they find

the lowest ask at \$11 too expensive. Therefore, Alice places a limit order to buy 2 VTHO at a price of \$8 per token. The LOB now looks like the image in Figure 1b.
Next, Bob wants to buy 2 VTHO as well. However, they need it more urgently than Alice and therefore decide to place a market order for 2 VTHO. The market order is immediately executed against the asks of the order book. The best ask is at \$11, but they are only selling one VTHO. After buying this VTHO token from the best ask, the market order that Bob executed needs to buy one more VTHO token. Therefore, the order also buys one VTHO from the next best ask at a price of \$12. Because of the size of the market order and the limited liquidity in the order book, Bob had to buy parts of their order at different prices. They eventually paid $\$11 + \$12 = \$23$ for two VTHO tokens, at an average price of \$11.50 per token. The LOB now looks like the image in Figure 1c. Because the limit sell order at \$11 has filled completely, the mid price has now moved as well. The new best ask is at \$12, so the new mid price is $(\$9 + \$12)/2 = \$10.50$.

2.7 Modelling Limit Order Books

The financial benefits that accurate order book representations can yield make the modelling of order books a popular field of research. Here, we focus on the stochastic model for order book dynamics proposed in [25]. This is an interesting model because it is has been shown to work for data sets other than the one used in the initial paper [26]. Moreover, the model seems to generate a specific hump in the order book that is often found in empirical analyses of stock markets [25], [27]. A final advantage of this model is that the authors supply an estimation procedure that can be used to apply the model to different Level II order book data sets. Level II order book data is simply data that contain the offers at all prices of a limit order book.
The model from [25] simulates order books by executing orders on a discrete price grid $\{x, ..., y\}$ with a set step size π between each price point. The model considers market orders, limit orders, and cancellation orders. Cancellation orders are simply cancellations of limit orders. In the model, limit- and cancellation orders refer to an order at a specific price point on the price grid. Market orders have a set rate at which they arrive, and the limit- and cancellation order arrival rates are set for each price point i separately. The model generates the orders independently from each other using Poisson processes. In practice, this means that the model randomly selects an order to execute, where the probability of an order being selected is equal to the order rate of that order, divided over the total order rate $r(t)$. The total order rate is calculated using Equation 5 [26].

$$r(t) = 2m + \sum_{i=1}^{a(t)-x} \lambda(i) + \sum_{i=1}^{y-b(t)} \lambda(i) + \sum_{i=1}^{max(i)} \theta(i) N_l(i,t) \tag{5}$$

Here, m refers to the market order rate, $\lambda(i)$ to the limit order rate at price point i, and $\theta(i)$ to the cancellation order rate at price point i. Moreover, $a(t)$ is the price point of the best ask and $b(t)$ is the price point of the best bid. Lastly, $N_l(i,t)$ refers to the number of outstanding limit orders at i price points from the opposite best quote (this means that the position of bid limit orders is measured through their distance i from the best ask, and vice versa). The total order rate is recalculated after an order is executed because it depends on the state of the order book. After an order is executed and the total order rate has been recalculated, the model determines the time until the next order. Equation 6 shows how the time between orders is calculated [26]. In this equation, p is a random number between 0 and 1, and $\tau(t)$ is the time until the next order from time t.

$$\tau(t) = -\frac{log(p)}{r(t)} \tag{6}$$

2.7.1 Model estimation procedure

Here, we describe the estimation procedure from [25] that can be used to find the model parameters for the LOB model, for any Level II order book data set. In order to initialize the model with new data, an estimate is needed for the market order rate m, the limit order rates $\lambda(i)$, and the cancellation order rates $\theta(i)$. Equation 7 describes how the market order rate is estimated. Here, N_m is the number of market orders that are present in the data, and T stands for the total time that the data encompasses. Also, S_m is the average size of a market order and S_l is the average size of a limit order. The estimate for m is referred to as $\hat{m}$, and m can simply be set to $\hat{m}$ when creating the model.

$$\hat{m} = \frac{N_m}{T}\frac{S_m}{S_l} \tag{7}$$

Equation 8 gives the calculation that should be performed to get an estimate of the limit order rates. The estimate from the data is referred to as $\hat{\lambda}(i)$. Furthermore, $N_l(i)$ refers to the total number of limit orders that arrived at a distance i from the opposite best quote. Recall that for bids, the opposite best quote refers to the best ask, and vice versa. In practice, this means that we count, for each step i from the opposite best quote, the number of times that the amount of tokens offered at that price increases. It is suggested to only use the total number of limit orders for $1 \leq i \leq 5$ in the estimation procedure [25], but the range $1 \leq i \leq 10$ has been shown to work as well [26].

$$\hat{\lambda}(i) = \frac{N_l(i)}{T} \tag{8}$$

In order to obtain $\lambda(i)$ from $\hat{\lambda}(i)$, one additional step is needed. In accordance with [28], $\lambda(i)$ is found by extrapolating a power law function to the estimate using a least-squares fit [25]. Equation 9 shows this function, in which i still refers to the distance in steps, and k and a are the parameters of the power law function [25].

$$\lambda(i) = \frac{k}{i^a} \tag{9}$$

The estimation method for the cancellation rates is given by Equation 10 [25]. Like with the estimation of the limit order arrival rates, the estimate of the cancellation rates uses a step distance $1 \leq i \leq 5$. It determines the estimate for these distances and repeats the value of distance $i = 5$ for all larger distances. In the equation, $N_c(i)$ stands for the total number of cancelled orders at distance i from the opposite best quote. Similarly to the limit orders, the cancelled orders are counted by checking how often the amount of tokens offered at a price decreases. When doing so, it is important to disregard any decreases due to market orders [25]. Also in Equation 10, Q_i refers to the value of the steady state shape Q at distance i. The estimation method for the steady state shape is given by Equation 11. Here, S_c refers to the average size of the cancelled orders, and S_l refers to the average size of the limit orders. Like in the other equations, T stands for the total time that the data encompasses. When creating the model, $\theta(i)$ can simply be set equal to $\hat{\theta}(i)$.

$$\hat{\theta}(i) = \frac{N_c(i)}{TQ_i}\frac{S_c}{S_l} \text{ for } i \leq 5 \text{ and } \hat{\theta}(i) = \hat{\theta}(5) \text{ for } i > 5. \tag{10}$$

The steady state shape Q_i is the average number of limit orders at a distance i from the opposite best quote [25]. The steady state shape is determined as an average of the steady state shape of the bid side Q_i^B and the ask side Q_i^A [25]. Equation 11 shows how to calculate the steady state shape of the bid side. The ask side is calculated in the same way. In Equation 11, M refers to the number of data

points in the Level II order book data set. Also, $S_i^B(j)$ is the quantity of shares/products/tokens bid at a distance i from the opposite best quote in data point j [25].

$$Q_i^B = \frac{1}{S_l}\frac{1}{M}\sum_{j=1}^{M} S_i^B(j) \tag{11}$$

3 Methods

3.1 The VeChain Decision Problem

For entities that wish to make transactions on the VeChain network, its design creates the following decision: do they buy VTHO for each transaction that they wish to make, or do they buy a large amount of VET to continuously generate the required VTHO? It is clear that this question mirrors the buy-or-rent question of the Ski Rental Problem. In this section, we start by mapping features of VeChain's buy-or-rent problem to features of the original Ski Rental Problem. Next, we identify the differences between the two problems, and explain how we handle these differences. Finally, we use these findings to create a definition for the buy-or-rent problem that users of the VeChain network encounter. We refer to this problem as the VeChain Decision Problem.

3.1.1 Similarities between the VeChain Decision Problem and the Ski Rental Problem

In order to implement solutions to the Ski Rental Problem in the setting of the VeChain network, it is important to accurately map the VeChain Decision Problem to the original Ski Rental Problem. A summary of the mapping that we use can be found in table 1.

Ski Rental Problem	**VeChain Decision Problem**
Skiing	Making transactions on the VeChain network
Ski's	VTHO tokens
A day of skiing by the skier	A day of network usage by the user
A skier renting ski's for a day	A user buying their daily required amount of VTHO
A skier buying ski's	A user buying enough VET to generate the required VTHO on a daily basis
Circumstances that determine the amount of skiing days	Circumstances that determine the length of the user's project

Table 1: Mapping between the original Ski Rental Problem and the VeChain Decision Problem.

Firstly, skiing is the activity for which you wish to rent or buy something. This corresponds to the activity of making transactions on the VeChain network. The ski's are the product that you wish to acquire and are what is used for the activity. Therefore, the ski's map to the VTHO tokens, as these are used to make the transactions. We map a day of skiing by the skier to a day of network usage by the user, which means that we use a time system with daily increments. An alternative to this would be to map a day of skiing to making transactions in a blockchain block. However, since VeChain's blocks occur every ten seconds, this would make it necessary to consider the absence of transactions in certain blocks (we cannot reasonably expect users to make a transaction every ten seconds). With daily usage, we can more reasonably assume that the user makes a certain number of transactions in that time period. In addition to this, it also makes more sense to assume that users of the network make their buy-or-rent decision on a daily basis rather than on a ten-second basis. Because we consider daily increments, the decision of renting ski's for a day is mapped to the decision to buy enough VTHO to fuel one day of transactions. Furthermore, the decision to buy ski's maps to the decision to buy enough VET to generate the required amount of VTHO each day, as both remove any further costs. Lastly,

we map the length of the skiing trip to the length of the network user's project. In both cases, there are unpredictable circumstances that determine this length. For the Ski Rental Problem, these are the circumstances that determine the amount of skiing days, like the weather. In the VeChain Decision Problem, they are all circumstances that determine the length of the user's project. Examples of such circumstances are the project being scrapped, delayed, or finish more quickly than thought. It could also be that the entire VeChain network fails, which leads to the project failing as well.

3.1.2 Special features of the VeChain Decision Problem

While the VeChain Decision Problem clearly mirrors the Ski Rental Problem, there are some important differences that together make the problem studied here different from problems in existing research.

First of all, in the original Ski Rental Problem, the skier's decisions do not influence the buy and rent prices. In the VeChain Decision Problem, on the other hand, the relationship between user decisions and rent/buy prices is both more relevant and more easily modelled. Let us first analyze why the original problem and many of its real-world examples do not have this feature. Consider the real-world example of cloud cost optimization, in which the user needs to decide between storing answers in cache (buying) or recalculating them on demand (renting) [12]. The average 'cloud-optimizer' has to accept the price packages offered by companies like AWS [29]. While it is true that extreme use of the AWS services probably affects their price, modelling this relationship seems a futile effort. One would not only need to know details about the prices that Amazon pays for building/destroying data centers, but also what decisions Amazon would make when AWS usage goes up or down. Simply put, the relationship between user decisions and rent/buy prices in this example is not direct enough to make modelling it worthwhile. Acknowledging the above, the main distinguishing feature of the VeChain Decision Problem stems from the fact that the products to rent or buy are cryptocurrency tokens. As explained in Section 2.5, cryptocurrency tokens are very actively traded on order-driven exchanges. Because these exchanges are the main way of acquiring cryptocurrencies, their order books act as a uniquely straightforward bridge between user decisions and usage costs. Furthermore, this relationship makes it so that buy and rent prices in the VeChain Decision Problem can also vary due to market actors speculating on the cryptocurrency tokens. All in all, the straightforward relationship between user decisions and token prices makes it both more important and more feasible to model the influences of user decisions on rent/buy prices and vice versa.

The presence of price fluctuations can also be expected to affect the performance of the solutions to the Ski Rental Problem when they are used to solve the VeChain Decision Problem. Recall from Section 2.2 that the algorithms used for solving the Ski Rental Problem (partly) base their buy-or-rent decision on the ratio between the total rent that they have paid and the current cost of buying. Because of this, the price fluctuations in the VeChain Decision Problem continuously affect the time point at which these strategies decide to buy. Consider for example a scenario where the rent price steadily increases. This means that the amount of money spent on rent will increase more quickly over time. In turn, this will make the time point where the algorithm decides to buy arrive more quickly than initially determined.

Another difference between the original Ski Rental Problem and the VeChain Decision Problem is that in the original problem, a skier can only buy or rent an entire set of ski's. It generally does not make sense to rent or buy only a part of a ski. In the VeChain Decision Problem, partly buying might be a valid option. For example, it would be possible to buy only 50% of the required VET, and to keep renting for the other 50%. Moreover, while renting ski's for multiple days at once does not make sense with constant prices, the variable prices in the VeChain Decision Problem might make it

interesting to buy more than the daily required amount of VTHO when its price is low. While these are all valid considerations when designing an optimal strategy for the VeChain Decision Problem, this study is more concerned with the performance of solutions to the original Ski Rental Problem. Therefore, the problem presented here does not consider the possibilities of partly buying or renting for multiple days.

Lastly, we know from Section 2.4 that buying and holding VET gives additional benefits in the form of voting rights. Since these votes do not occur often and would complicate the problem setting, these votes are not considered in this study.

3.1.3 The VeChain Decision Problem definition

Considering the similarities discussed in Section 3.1.1 and the differences discussed in Section 3.1.2, we define the VeChain Decision Problem as follows:

> *You wish to use the VeChain network for a project, but you do not know the duration of this project or if the network will cease to exist in the future. You have to make a set number of N transactions each day, for which you require exactly M VTHO. Your first option is to rent. This means that you buy the M VTHO that you require on a daily basis. Your second option is to buy. This means that you make a single purchase of P VET, where P is the exact amount of VET needed to generate M VTHO per day. Each day, you can decide whether you want to rent or buy, based on the buy and rent prices that you encounter that day. Both the buy and the rent price fluctuate based on the effects of users buying through limit order books. Prices can also fluctuate based on unknown price trends. Once you have bought you cannot switch back to renting. Your goal is to minimize the cost that you pay over the entire duration of the project.*

3.2 Model

In order to gather the data that is required to study the research questions of this study, we create a model that can simulate the VeChain Decision Problem and analyze the performance of different algorithms. The model consists of three main components: the VeChainThor network, the users of the network, and the economy. In this section, we will give a rough overview of how these three components are related. The sections that follow will go into more detail about each component.

An overview of the components is given in Figure 2. The blue parts refer to the VeChainThor model component. This is the component that is responsible for keeping track of the simulation and making sure that everything happens in the right order. While doing so, this component also models all relevant aspects of the VeChain network. The yellow component represents the users of the network. The goal of each user is to solve the VeChain Decision Problem as optimally as possible, using the available data from the network and the economy. As can be seen in Figure 2, the actions of users influence both the network and the economy, which in turn changes the data that the users will use in the future. The red component represents the economy. The economy contains a cryptocurrency exchange through which the users can buy VET and VTHO tokens. The state of the economy can also be affected by realistic price trends that follow from speculation rather than network use.

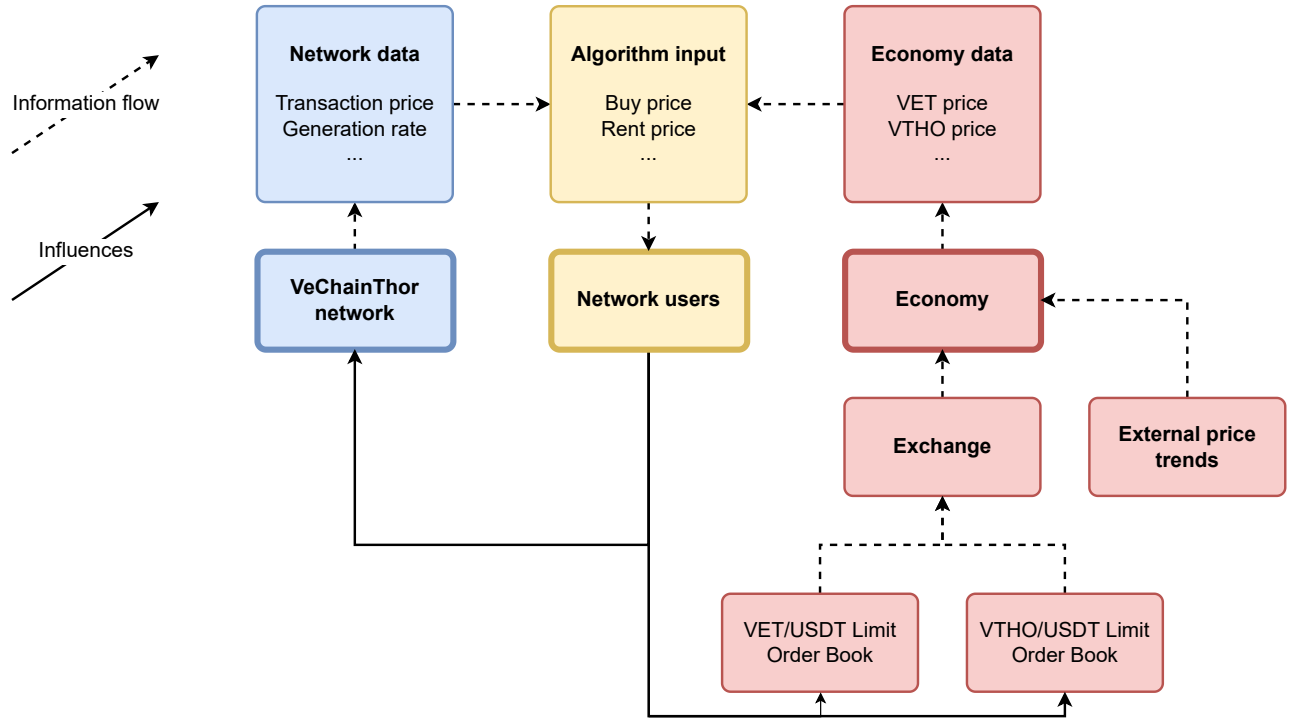


Figure 2: General overview of the model used to simulate the VeChain Decision Problem. The VeChain network component is shown in blue, the users are shown in yellow, and everything related to the economy is shown in red.

3.3 VeChainThor Model Component

The VeChainThor model component is the component of the model that represents the VeChainThor blockchain. This component handles the evolution of time in the blockchain network, and calls on the economy and users when it is their turn to perform an action. A general overview of the daily update sequence that this component executes is shown in Figure 3.

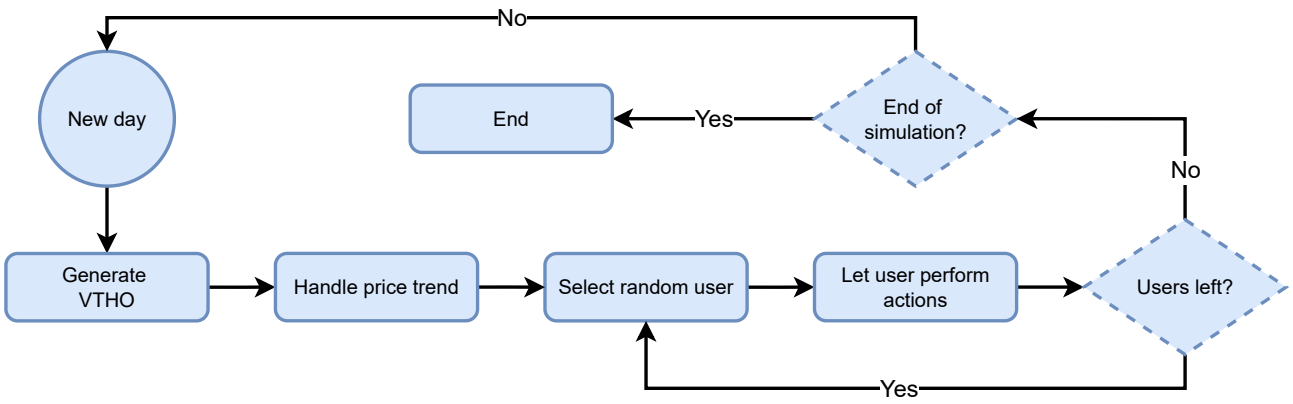


Figure 3: Flowchart containing the daily update sequence that the VechainThor model component executes on each day of the simulation.

Each new day, the network starts by generating the amount of VTHO that has been generated by the entire VET supply in the previous day. Recall from Section 2.4 that with the default amount of

VET, this means that 37,459,858 VTHO is generated. The most important part of this step is that the network calls on the economy to update the amount of circulating VTHO, and to use the new amount of circulating VTHO to update the liquidity of the limit order books of the exchange. Next, the network calls on the economy to handle any price trends that are not the result of network use. If these trends are present (see Section 3.4.2 for an explanation of these trends), the economy changes the price of VET and/or VTHO based on the trends. Now that the conditions of the new day have been set, it is time for the users to use the network. The network component selects the users at random until each user has used the network once. At that point in time, the day has ended and the network checks whether the simulation is done. If this is the case, the network will stop. If not, the network will move on to the next day.

3.3.1 Default parameters of the VeChainThor model component

The VeChainThor model component needs to be initialized with a couple of parameters. All parameters relevant to the experiments along with their default values are given in Table 2.

Parameter	**Description**
Economy	The economy to use. The economy component contains its own parameters that are explained in Section 3.4.3.
Simulation length	The length of the simulation in days. By default set to 3650 days (10 years).
Generation rate	The amount of VTHO that a single VET token generates per day. As mentioned in Section 2.4, the default value is 0.000432 VTHO per VET per day.

Table 2: Default parameters of the VeChainThor model component that are relevant to the experiments.

3.4 Economy

An important part of the model is the economy. The economic component of the model provides an economy for the VET and VTHO tokens. The economy contains a simple version of a cryptocurrency exchange and keeps track of the amount of circulating tokens and their prices. The economy only updates when the VeChainThor model component calls on it or when a user interacts with the exchange. This section gives a general overview of the economy. The limit order books of the exchange are discussed separately in Section 3.5, as their implementation demands special attention.

3.4.1 Exchange

Arguably the most important part of the entire model is the cryptocurrency exchange where users buy and sell their tokens. This exchange serves as a bridge between the decisions of users and the economic impact of those decisions. Accurately modelling price movements based on user behavior and market conditions is a science on its own, so obtaining a perfect model is not expected. Luckily, this is also not required as the study is mostly concerned with user performance and -behavior relative to the different conditions that we model. However, in order to understand the interplay between the user's decisions, and the VET and VTHO markets, it is important that we model the price fluctuations that result from user decisions as accurately as possible. We want to capture the real-world differences

between how buying VET influences the VET price and how buying VTHO influences the VTHO price. Therefore, an attempt is made to mirror the real world limit order books of VET and VTHO as well as possible, while keeping the effects computationally tractable.
To this end, the exchange is represented by two limit order books (LOB's). One of the LOB's contains the VET / USDT pair and the other contains the VTHO / USDT pair. Every time that a user wants to interact with the exchange, they are presented with a LOB for each pair. We only allow users to place market orders on the exchange. When a user places a market order, the (relative) price fluctuation is determined based on the shape of the LOB. This relative price fluctuation is then used to determine the new price of the token. As described in Section 2.6, LOB shapes are always changing and their shape can greatly influence the total price of a market order. To realistically model the randomness of the LOB shape, users are presented with a different instance of the LOB every time they interact with the exchange. Each presented LOB instance is selected from a set of pre-generated LOB instances that corresponds to the chosen exchange pair. The instances are generated using a process that ensures that they reflect the properties of the order books of the exchange pair in question. The LOB generation process is described in Section 3.5.4.
The LOB instances that represent the current state of the exchange can be used by users to calculate the exact costs of their current buy and rent price. Once a user has made a decision, they execute the buy or sell order on the chosen LOB. If the order that the user made results in the quantity of one or more of the price points going to zero, the user will have affected the mid price. The relative change in mid price that results from an order is used to change the price of the corresponding token. Obviously, this means that the precision of the price points (also referred to as ticks) of the LOB model is very important in determining when users affect the price. Moreover, the shape of a LOB can be heavily affected when a user executes market orders on it. While it is possible to model the behavior of the LOB after this happens, doing so is unnecessarily complicated for our purposes. Therefore, we make the assumption that there is sufficient time between the users' actions for the order book to reset to a shape that is more or less in equilibrium. Doing so means that we can present each user with a new LOB, instead of having to model the LOB behavior after a user has executed an order.

3.4.2 Speculative price trends

As mentioned in Section 2.5, cryptocurrency markets are notoriously volatile and speculative. This means that the price of tokens can be affected a lot by factors that are unrelated to token usage. In order to investigate how these factors influence the VeChain Decision Problem, different price trends are introduced in some of the experiments. These price trends are meant to simulate a long-term uptrend and downtrend that occur as a result of trading by speculative market actors. The implementation of the speculative price trends can be done more simplistically than the usage-dependent price influences, as we only require the daily price change that results from the speculation. There is no need to simulate how this price change came to be.
Because VET and VTHO are actively traded tokens, the history of their prices is easily accessible. It is therefore quite straightforward to base the speculative price trends on real-world data. Doing so will allow for a more realistic price sequence than, for example, randomly selecting the change in price from a distribution as was done in [**?**]. In order to create the sequences, we obtain the historical daily closing price data of VET[1] and VTHO[2] from Yahoo Finance. From this data, two subsets with the duration of one year (365 closing prices) are selected. The first subset contains closing prices from 19-04-2020 to 18-04-2021, and shows an uptrend in which the price of VET rose from $0.003936 to

[1] `https://finance.yahoo.com/quote/VET-USD/`
[2] `https://finance.yahoo.com/quote/VTHO-USD/`

$0.254632 while the price of VTHO rose from $0.000294 to $0.022546. The second subset contains closing prices from 20-04-2021 to 19-04-2022, and contains a clear downtrend in which the price of VET decreased from $0.251997 to $0.061677 while the price of VTHO decreased from $0.021500 to $0.004323. Figure 4 shows the price of VET and VTHO during the uptrend and Figure 5 shows the same for the downtrend. Although both price trends have the same duration, it should be noted that the absolute price change over the entire period is not equal for the uptrend and the downtrend. Specifically, we note that the absolute price change in the downtrend is less extreme than that of the uptrend. While this does not matter for our model, it is important to note as this difference in severity could cause a similar difference in the severity of the effects of adding the price trends in our experiments.

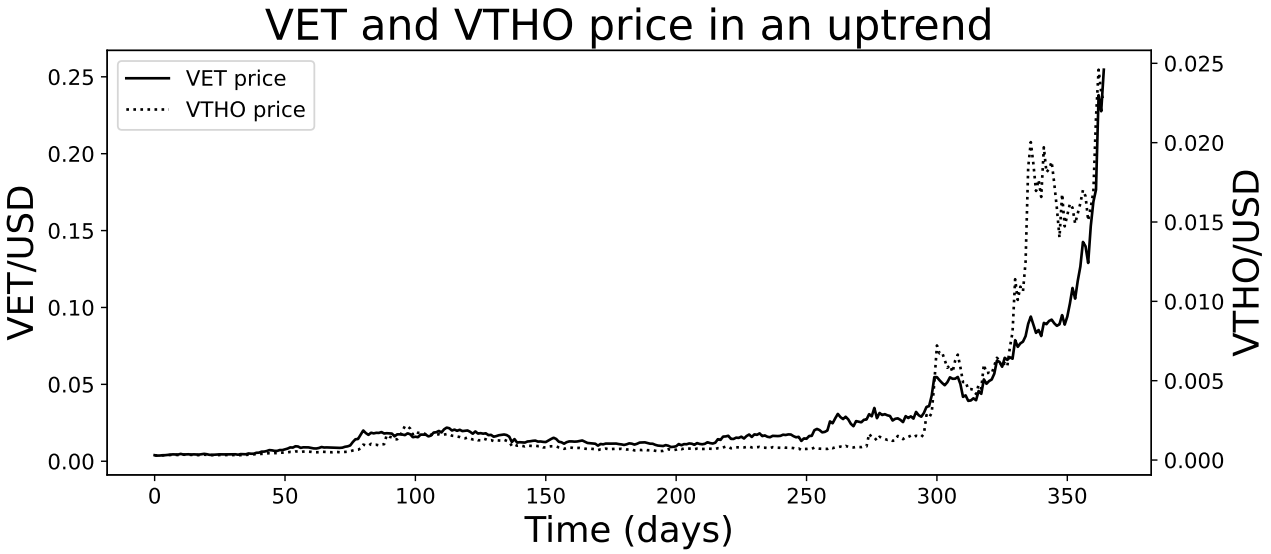


Figure 4: A year of upward price movement for the VET/USD and VTHO/USD price pairs.

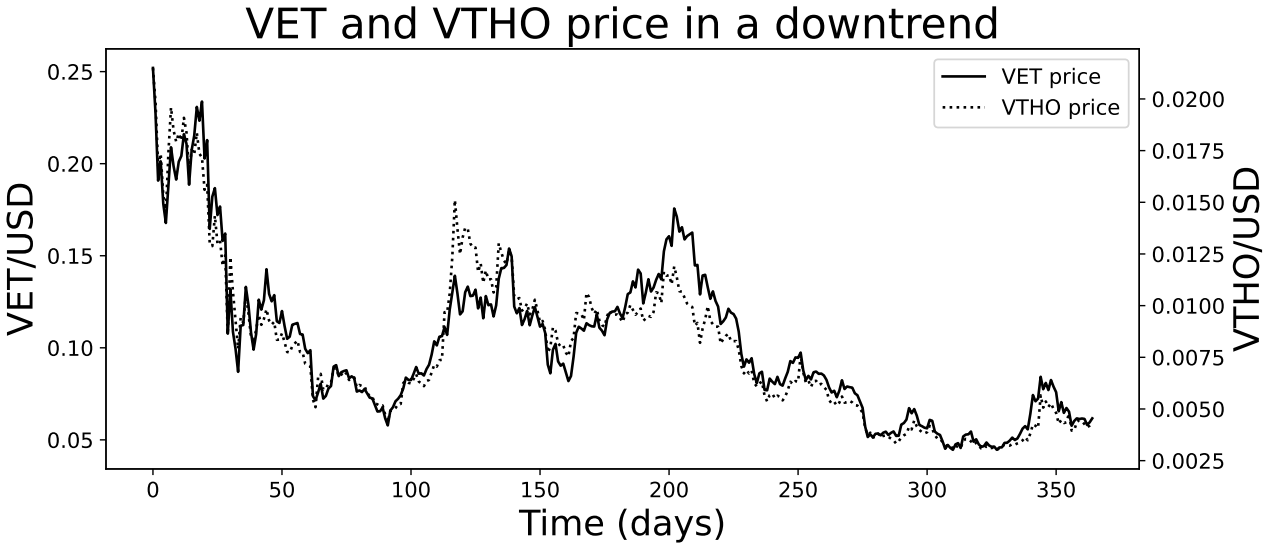


Figure 5: A year of downward price movement for the VET/USD and VTHO/USD price pairs.

Looking at Figures 4 and 5, we note that the two trends show different behavior. The uptrend shows

a relatively slow price increase followed by an intense spike. This upward behavior is often found in (cryptocurency) markets and a good example of why this data-driven approach results in a more realistic price sequence than simply selecting a random percentage increase. The downtrend is much more gradual than the uptrend and even contains some significant upward movement. The graphs also show us that, apart from a few deviations, the prices changes of VET and VTHO more or less mirror each other.
Since the algorithms that solve Ski Rental Problems mostly base their decisions on the buy-to-rent ratio (the ratio between the buy price and the rent price), only price trends that change this ratio are interesting when examining the effects of speculation. Looking at the data, it would not make much sense to model a speculative price trend in which both tokens are affected as the buy-to-rent ratio would remain approximately equal. In the experiments, we therefore only apply the effects of the speculative price trends on the VET token. It should be noted that doing so discounts any relationship between the price of VET and VTHO. However, it does allow us to use this data-driven approach to investigate the influences of a buy-to-rent ratio that varies due to speculation.

3.4.3 Default parameters of the economy

Parameter	**Description**
VET price	The price of a single VET token. By default initialized to 0.0235 USDT.
VTHO price	The price of a single VTHO token. By default initialized to 0.0015 USDT.
Circulating VET	The amount of VET available to the economy. By default initialized to 86,712,634,466 VET.
Circulating VTHO	The amount of VTHO available to the economy. By default initialized to 38,396,354,542 VTHO.
VET liquidity ratio	The proportion of circulating VET that exists in the limit order book of the exchange. By default initialized to 0.00674.
VTHO liquidity ratio	The proportion of circulating VTHO that exists in the limit order book of the exchange. By default initialized to 0.01226.
Speculative price trend	The price trend that results from speculation rather than network use. Can be used to add a trend to the price of VET or VTHO. By default initialized to none.

Table 3: Parameters of the economy that are relevant to the experiments, along with their default initial values.

In order to create simulations that mirror the real-world as much as possible, we initialize the economy using the parameters summarized in Table 3. Firstly, we initialize the price of a VET token to 0.0235 USDT and the price of a VTHO token to 0.0015 USDT. These values are obtained by taking the mean of the mid price from the data set that is used to generate the limit order books. This ensures that the two values relate to each other because they come from the same period of time. Furthermore, doing so assures us that the initial token prices match with the shapes of the limit order books that we generate.

The circulating supply of the tokens needs to be initialized as well. The circulating supply refers to the supply that is theoretically available to the market. We assume that all VET in existence is part of the circulating supply, so this is initialized to 86,712,634,466 VET. As the amount of VET in existence cannot change, this value will remain the same for the entire simulation. Because VTHO is continuously generated, its supply is ever increasing. We initialize the amount of circulating VTHO to 38,396,354,542 VTHO. This equal to the total amount of VTHO that was generated from genesis on 30-06-2018 up until 19-04-2021. We use the amount of VTHO that existed on 19-04-2021 because this is the date that lies exactly in the middle of the date ranges from which the speculative price trends are obtained. Like with the initial token prices, the goal here is to make all initial data match as well as possible.

As mentioned in Section2.6, the liquidity of a limit order book is an extremely important factor in determining the effect of market orders on price. Instead of using absolute values for the liquidity of the order pairs, the economy uses liquidity ratio parameters. These ratios determine the proportion of the circulating supply that is available in the limit order book. We use a ratio as opposed to an absolute value so that the liquidity takes into account that the circulating supply of VTHO keeps increasing. Although the increasing supply might not have a large effect in the short-term, we should expect a larger circulating supply to result in higher liquidity in the long-term. Again, in order to make all initial values match as well as possible, we determine the liquidity ratio of the order pairs through the data set used for generating the limit order books. The liquidity ratio of the VET pair comes out to be 0.00674, and the liquidity ratio of the VTHO pair is determined to be 0.01226. For the exact calculations used to determine these values, please refer to Section 3.5.5.

Lastly, the economy can be initialized with one of the speculative price trends discussed in Section 3.4.2. The trends can be applied so that either one of the tokens increases/decreases in price, or both. The sequences have a length of 365 days but will be stretched to fit the length of the simulation. This means that in a simulation with a length of 3650 days, the speculative price trend affects the price once every 10 days. By default, the speculative price trend is not considered.

3.5 Limit Order Books

One of the main objectives of this study is to model the relationship between user decisions and token prices. Because we model this relationship exclusively through the LOB's of the exchange, the properties of the LOB's are a key factor in how users affect the economic conditions in the simulation. Inspired by [26], we use the model from [25] that we introduced in Section 2.7 to generate multiple instances of LOB's that accurately reflect the LOB shapes of the real-world VET / USDT and VTHO / USDT markets.

To this end, we determine the order rates of the LOB model using the estimation procedure from [25], as outlined in Section 2.7.1. To use this procedure, we require Level II order book data from the VET / USDT and VTHO / USDT price pairs. For both pairs, the data are collected using the API from the Binance cryptocurrency exchange [22]. Binance is chosen because it is one of the largest global cryptocurrency exchanges, which means that their order books can be expected to be a good representation of the markets in general. Data are gathered on one day, and are only used after verifying that no major events happened on that day. This way, we can assume that the data come from a day that is representative of a normal market day. For a specification of how the data are gathered and cleaned, please refer to Appendix A. We will refer to the data set resulting from that procedure as the LOB data set. Table 4 gives an overview of the data that each data point of the LOB data set contains.

Tick distance	**1**	**2**	**3**	**4**	**...**	**10**
Bids	(0.023, 100)	(0.022, 150)	(0.021, 0)	(0.020, 200)	...	(0.014, 100)
Asks	(0.024, 0)	(0.025, 150)	(0.026, 100)	(0.027, 300)	...	(0.033, 200)

Table 4: Example of all data that is contained in one data point of the data set created using the raw Binance API data.

For both the bid and the ask side of the order book, we obtain (price, quantity) pairs for the 10 price points closest to the opposite best quote. These pairs give the sum of tokens from all limit orders at each price point. The price point values are based on the tick step size. The tick step size determines the precision of the price points and its value is based on an analysis of the raw data and token prices. In [25], they used data from the stock market in which the price points contained three significant figures (74300, 74200, etc.). As specified in Appendix A, we round the VTHO data to five decimals (0.00148, 0.00147, etc.) and the VET data to four decimals (0.0233, 0.0232, etc.). Rounding the VET and VTHO data to a different precision makes it so that the shapes of the LOB's are more comparable to each other, when using the same amount of ticks. This is because the token prices also differ by an approximate factor of 10 when the data was gathered. If we would use the same precision, the VTHO LOB would only cover about one-tenth of the price range of the VET LOB. Because we do not limit the price fluctuations of the tokens in our model, it makes sense to have the LOB's cover approximately the same relative price range. Besides the tick step size that is used, there is another difference between the estimation method presented in [25] and the one applied here. This is that we use order data up to ten price ticks away from the opposite best quote for the estimation of the limit order rates. This is opposed to the five ticks used in [25]. The reason for using more ticks is simply that the data is available and allows for a more precise estimation.
The estimation procedure and all equations that are used are explained in Section 2.7.1. Here, we will continue by presenting the order rates that resulted from the estimation procedure. Sections 3.5.1 through 3.5.3 give the obtained market- limit- and cancellation order rates for the VET and VTHO price pairs.

3.5.1 Market order rates

The market order rate determines how often a market order is placed. The market order rate is estimated using equation 7 (Section 2.7.1). Applying the market order rate estimation procedure to the LOB data set yielded the following market order rates m for VET and VTHO:

$$m_{\text{VET}} = 2.04$$

$$m_{\text{VTHO}} = 0.62$$

3.5.2 Limit order rates

The limit order rate determines how often a market order is placed at each tick distance i away from the opposite quote. The limit order rate is estimated using equation 8 (Section 2.7.1). Fitting the power law function (Equation 9) to the estimated limit order rates yielded the following limit order rate functions $\lambda(i)$ for VET and VTHO:

$$\lambda_{\text{VET}}(i) = \frac{1.79}{i^{1.11}}$$

$$\lambda_{\text{VTHO}}(i) = \frac{2.52}{i^{0.58}}$$

Figures 6 and 7 show the raw limit order rates as well as the estimated rates from the power function for both pairs. By visually comparing the rates from the model and the data, it can be seen that the model's approximation of the data is fairly good.

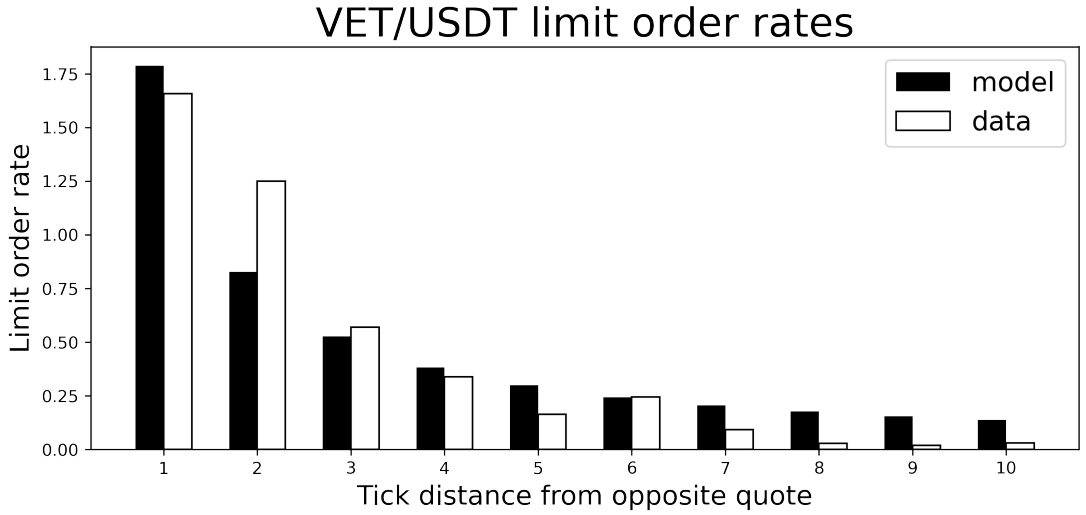


Figure 6: Limit order rates of the VET / USDT pair, as determined from the raw Binance API data and estimated using the model.

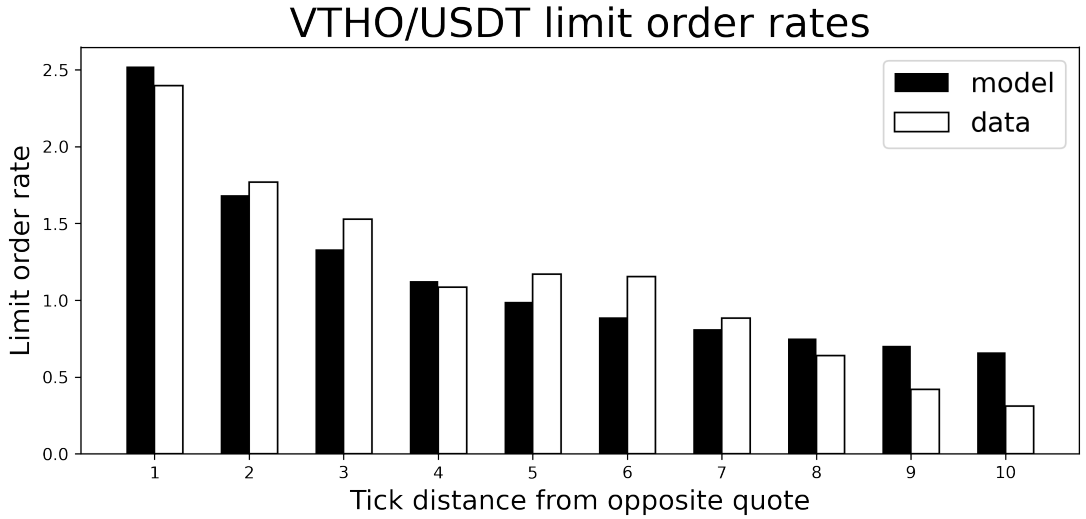


Figure 7: Limit order rates of the VTHO / USDT pair, as determined from the raw Binance API data and estimated using the model.

3.5.3 Cancellation order rates

The cancellation order rate $\theta(i)$ determines how often a limit order is cancelled at each tick distance i from the opposite quote. The cancellation order rate is estimated using Equation 10 (Section 2.7.1). Applying the cancellation order rate estimation procedure to the LOB data set yielded the cancellation order rates shown in Table 5.

Tick distance	**1**	**2**	**3**	**4**	**5+**
$\theta_{\mathbf{VET}}(\mathbf{i})$	0.591	0.285	0.364	0.085	0.084
$\theta_{\mathbf{VTHO}}(\mathbf{i})$	0.202	0.129	0.153	0.162	0.078

Table 5: Estimated cancellation order rates of the VET and VTHO LOB's, rounded to three decimals.

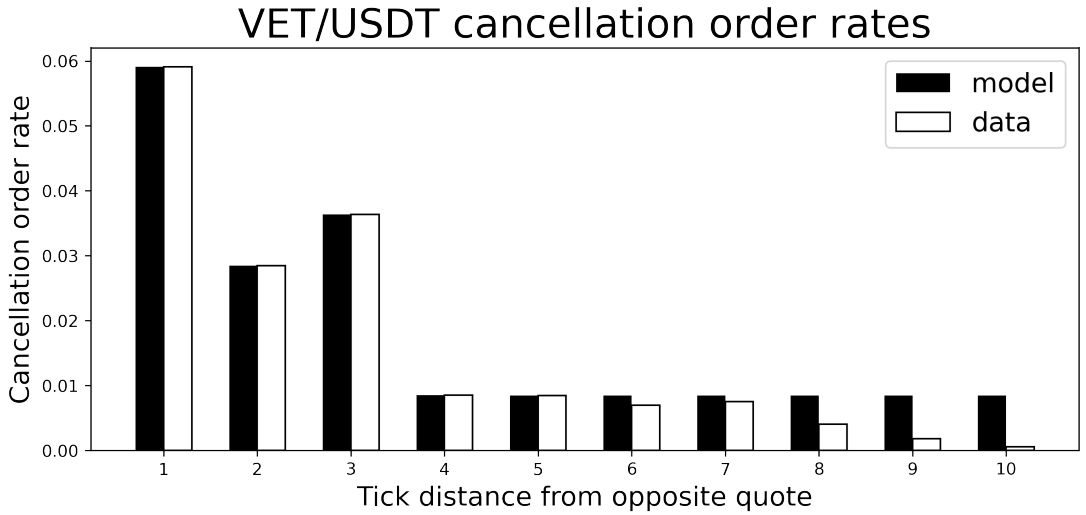


Figure 8: Cancellation order rates of the VET / USDT pair, as determined from the raw Binance API data and estimated using the model.

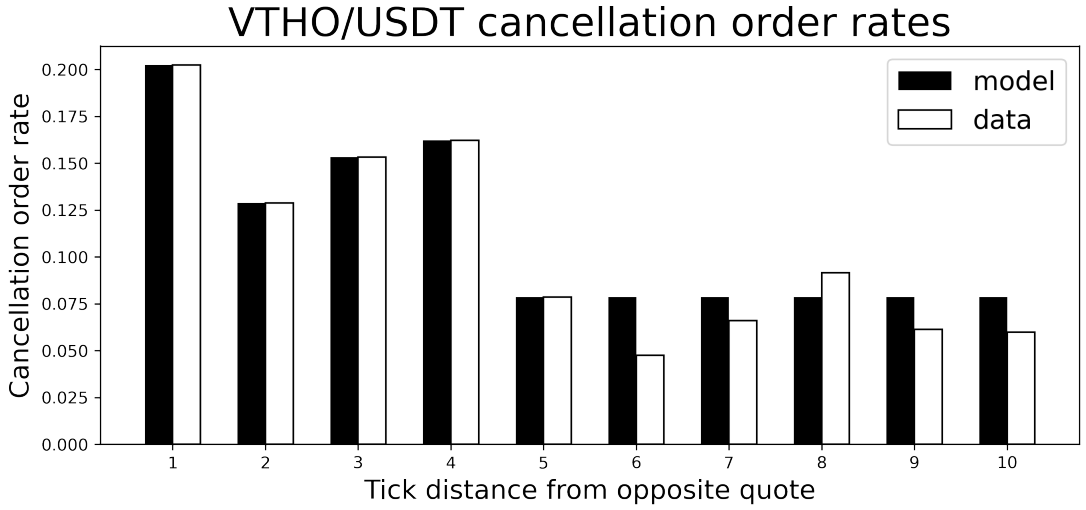


Figure 9: Cancellation order rates of the VTHO / USDT pair, as determined from the raw Binance API data and estimated using the model.

Figures 8 and 9 show the cancellation order rates $\theta(i)$ from the data as well as the estimated values. Due to the method used, the model's rates for the first five ticks equal those of the data. However, the model seems to be a good estimate for the data after five ticks as well.

3.5.4 Generating the limit order books

Now that the order rates have been estimated for both order pairs, the model from [25] can be used to generate LOB instances for both order pairs that are representative of the real-world. In total, we generate 100 LOB instances for each order pair. To generate the instances, the model is initialized with the order rates that have been determined in the previous section. The model is then run until it reaches the time point 1000, which gives enough time for the LOB to fill up (as determined through trial-and-error). At that point in time, the LOB is representative of an instance of a real-world LOB of the order pair. The generated LOB instance is saved and the model is reset before generating the next instance. To get an idea of how the LOB's of the VET and VTHO order pairs differ, we can average the 100 LOB's of each pair [26]. Figure 10 gives the average shape of the VET / USDT LOB's and Figure 10 shows the average shape of the VTHO / USDT LOB's. The difference in shapes between the two LOB's strengthens the case for why we need to use real-world data to generate the LOB's. Buying x VTHO clearly has a different influence on the VTHO price than buying x VET has on the VET price.

It is important to note that we require that the LOB's can be used in different economic conditions. The model from [25] generates LOB's with absolute values. We need to convert the absolute LOB's to relative LOB's, for two reasons. First of all, because the LOB's are pre-generated and the liquidity of the VTHO pair goes up when VTHO is generated (through on the liquidity ratio), an absolute depth profile can cease to be representative over time. Secondly, since the price can be expected to change a lot over time, it is important that the distance from the mid price is given in relative values rather than absolute ticks. To exemplify: the price moving 10 ticks of 0.0001 USDT is much less volatile when the price is at 2.00 USDT than when it is at 0.02 USDT. A LOB with absolute ticks that is generated from data when price is 0.02 USDT is therefore not representative anymore when price is 2.00 USDT.

Converting the depth profile of the LOB's is simple enough. In order to convert the absolute order quantity at each price point to a relative quantity, the quantity of each limit order is simply divided over the sum of outstanding limit order quantities in the LOB.

Converting the price points to relative values is a bit more work. In the absolute LOB's that are generated, the x-axis only contains the distance from the mid price. The mid price itself is therefore not present in the LOB. In order to determine the step size of one tick relative to the mid price, we need to know the mid price of the LOB. As there is no one mid price in the LOB data set, we analyze the data in order to determine the mean mid price of the entire data set. Specifically, we calculate the relative step size of a tick using Equation 12. Here, N is the total amount of data points in the LOB data set, $bid_{best}(i)$ is the highest bid at data point i, $ask_{best}(i)$ is the lowest ask at data point i, and $tick_{absolute}$ refers to the absolute step size of a tick. To clarify, the absolute step size of a tick in this data is the precision to which the order prices were rounded, so 0.0001 for VET and 0.00001 for VTHO. Simply put, Equation 12 divides the absolute tick step size over the mean mid price in the data. For the VET LOB, it is calculated that each tick of the VET LOB corresponds to a relative price movement of 0.00426, or 0.426%. For the VTHO LOB, the relative tick step size is determined to be 0.00684, or 0.684%.

$$tick_{relative} = tick_{absolute} / \frac{\sum_{i=1}^{N}((bid_{best}(i) + ask_{best}(i))/2)}{N} \tag{12}$$

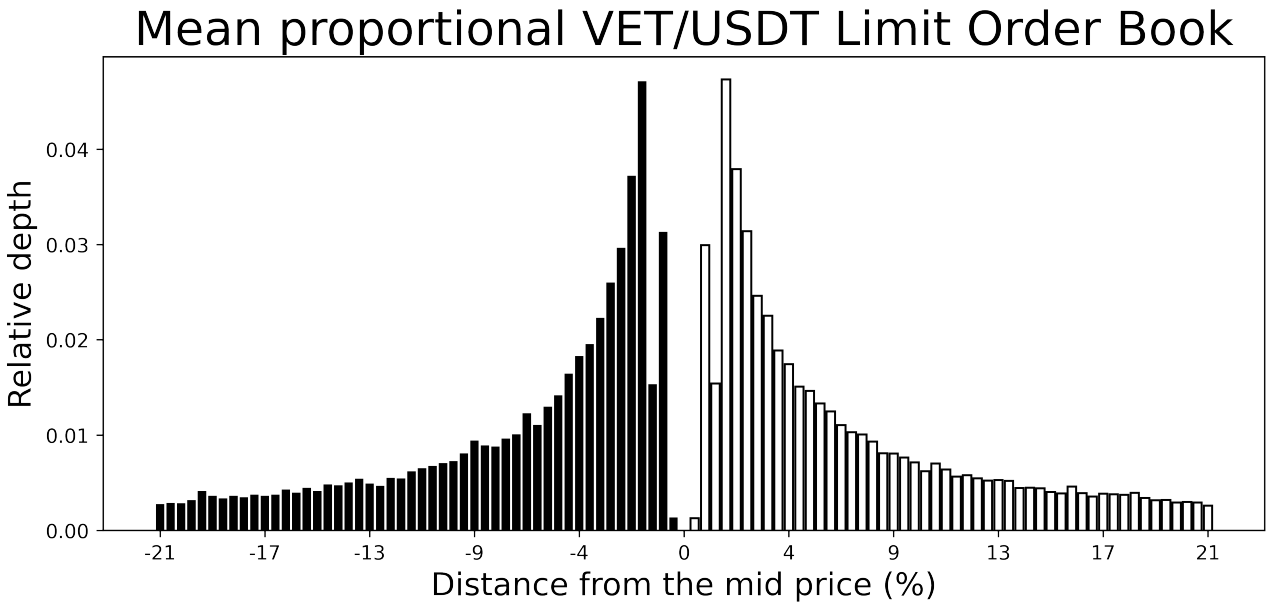


Figure 10: Mean shape of the 100 generated LOB's of the VET/USDT pair.

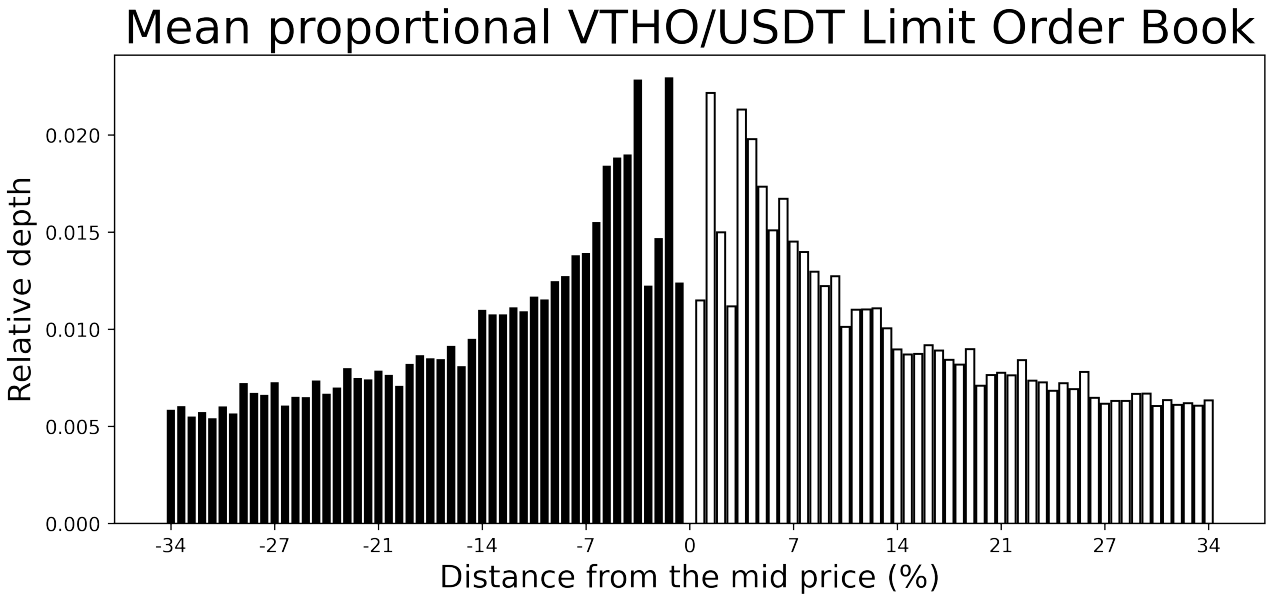


Figure 11: Mean shape of the 100 generated LOB's of the VTHO/USDT pair.

3.5.5 Liquidity

Section 3.5 introduced the concept of liquidity and why it is very important in order-driven markets. As mentioned in Section 3.4.3, we manage the liquidity of the order books through liquidity ratios that are set to constant values. This section explains how we obtained the liquidity ratios for the VET and VTHO LOB's.

Liquidity estimates are freely available for most cryptocurrency pairs. However, this data usually only concerns the liquidity within $\pm 2\%$ of the mid price. Since it should be possible for price to move more than 2%, it is important that a reasonable estimate of the liquidity is obtained for the entire scope of

the LOB's. To this end, the mean LOB's shown in Figures 10 and 11 are used to extrapolate the liquidity to the entire range of all LOB's. Firstly, as the exchange that is created represents the entire market instead of a single exchange, the total market liquidity (in dollars) within $\pm 2\%$ of the mid price is obtained from Live Coin Watch for the VET[3] and VTHO[4] pairs. The data was gathered on the same day as the Level II order book data used to generate the LOB's. The liquidity values should therefore approximately match to the LOB shapes. However, the calculations performed here should still be viewed as a very rough estimate, given that liquidity is notoriously volatile. To extrapolate the liquidity, it is first estimated how many ticks of the generated order books account for a price movement of 2%. We can do this using Equation 13, in which $tick_{fraction}$ is the relative tick size of the LOB, as calculated in Section 3.5.4.

$$ticks_{num} \approx 0.02/tick_{fraction} \tag{13}$$

For VET, this comes out to $0.02/0.00426 \approx 5$ ticks. Now, a cumulative sum of the relative quantities of the mean LOB can be used to estimate the proportion of the order book that the first n ticks contain. For both the bid and the ask side, the first 5 ticks contain 13.2% of the limit order amount. Moreover, the bid side contains 50.3% of the entire LOB and the ask side contains 49.7%. This means that for the mean VET LOB, the first 5 ticks contain

$$(0.132/0.503) = 0.262 \text{ of the bids.}$$

$$(0.132/0.497) = 0.266 \text{ of the asks.}$$

The first 2% of price movement on any side (bid or ask) therefore contains on average

$$(0.262 + 0.266)/2 = 0.264 \text{ of all orders on that side.}$$

According to the Live Coin Watch data, the average liquidity on one side of the VET order book is

$$(\$3,994,400 + \$3,099,500)/2 = \$3,546,950.$$

This means that the average total liquidity on one side of the mean VET LOB can be estimated to be

$$(1/0.264) * \$3,546,950 = \$13,435,416.67.$$

With a total circulating supply of 86,712,634,466 VET and the mean mid price of VET in the LOB data set being \$0.0235, this yields a VET liquidity ratio of

$$13,435,416.67/(86,712,634,466 * 0.0235) = 0.00674.$$

In the case of VTHO, a 2% price move is contained in $0.02/0.00684 \approx 3$ ticks. Again, a cumulative sum of the relative quantities of the mean LOB can be used to estimate the proportion of the order book that the first 3 ticks contain. For the bid side, the first 3 ticks contain 0.050% of the limit order amount. For the ask side, the first 3 ticks contain 0.049% of the limit order amount. The bid side contains 50.2% of the entire LOB and the ask side contains 49.8%. This means that for the mean VTHO LOB, the first 3 ticks contain

$$(0.050/0.502) = 0.100 \text{ of the bids.}$$

[3] https://www.livecoinwatch.com/price/VeChain-VET

[4] https://www.livecoinwatch.com/price/VeThorToken-VTHO

$$(0.049/0.498) = 0.098 \text{ of the asks.}$$

The first 2% of price movement on any side therefore contains on average

$$(0.100 + 0.098)/2 = 0.099 \text{ of all orders on that side.}$$

Using the Live Coin Watch data, the average $\pm 2\%$ liquidity on one side of the VTHO order book is determined to be

$$(\$99,931 + \$66,247)/2 = \$83,089.$$

This means that the average total liquidity on one side of the mean VTHO LOB can be estimated to be

$$((1/0.099) * \$83,089 = \$839,282.83.$$

The mean mid price of VTHO in the LOB data set is 0.0015. On the day that the LOB data was gathered, approximately 45,630,180,356 VTHO had been generated in total. Ignoring any VTHO burns for simplicity, this gives a VTHO liquidity ratio of

$$\$839,282.83/(45,630,180,356 * 0.0015) = 0.01226.$$

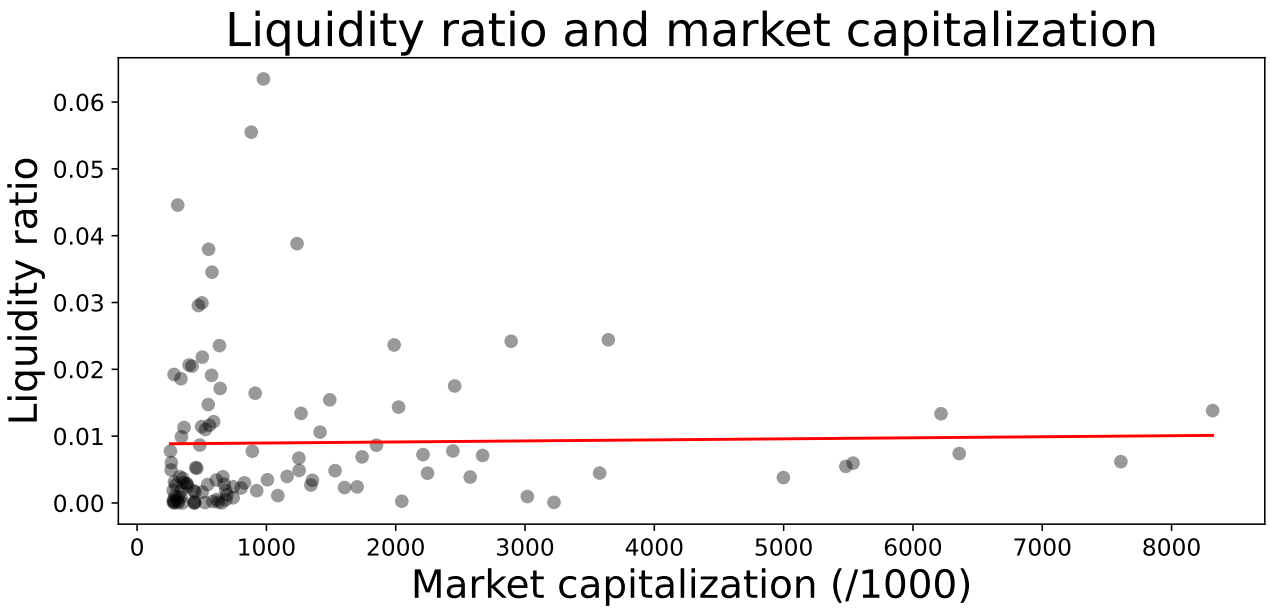


Figure 12: Liquidity ratio as a function of market capitalization (in $) for cryptocurrencies with Live Coin Watch market capitalization rank 10-120. The red line indicates a linear regression performed on the data.

The use of a liquidity ratio makes it easy to update the liquidity based on the amount of tokens in circulation. However, it is not unthinkable that there also exists a relationship between market capitalization and liquidity ratio. Tokens with a higher market capitalization can be seen as more popular, which in turn might lead to them being traded more actively. To investigate whether a linear relationship exists and should be implemented in the model, we study the cryptocurrency tokens that have a

Live Coin Watch market capitalization rank between 10 and 120. The top 10 coins in market capitalization are omitted from the data since these are difficult to compare to VET and VTHO. Examples of this are Bitcoin, which probably has a higher market cap by simply being the first cryptocurrency, and USDT, which is pegged to the dollar. For each token in the data set, we obtain the market capitalization in dollars, as well as the $\pm 2\%$ liquidity in dollars from Live Coin Watch. We obtain the liquidity ratio of each token by dividing the $\pm 2\%$ liquidity value by the market capitalization. Next, we fit a simple linear regression to the data. The result suggests that there is no significant linear relation between market capitalization and the liquidity ratio ($F(1,108) = 0.047$, $R^2 = 0.00$, $p = 0.83$). The data points as well as the fitted regression line can be found in Figure 12. As no linear relationship seems to exist between market capitalization and the liquidity ratio, we assume that the liquidity ratio is not influenced by the total market capitalization and can remain constant during the simulations.

3.6 Network Users

The network users are the entities that use the VeChainThor network. The goal of each user is to solve the VeChain Decision Problem as defined in Section 3.1. Users share most of their properties, and are primarily differentiated by the algorithm that they use to determine whether they want to buy or rent. Here, we first describe the common properties of all users. Next, Sections 3.6.3 through 3.6.8 describe each user according to the strategy that they apply.

Parameter	**Description**
ID	Integer used to identity the user.
Active	Determines whether the user is active on the VeChainThor network. Users always start out as active.
State	Keeps track of whether the user is renting or has bought. Users always start out as renting.
Maximum days	Determines the maximum amount of days that the user is active, as chosen by the adversary. Cannot be used to inform the user's decision making.
Algorithm	The algorithm that the user uses to determine when to buy.
User size	The size of the user, expressed in the amount of VTHO that they use per day. Defaults to 1 VTHO / day.
VET needed	The amount of VET that the user needs to buy if they wish to generate their daily required VTHO. Determined through a combination of the user size and VTHO generation rate.

Table 6: Important properties shared by all users.

Each user has an ID by which they can be identified. Users can either be active, which means that they are using the network, or inactive, which means that they are not using the network. All users start out as active. In Section 2.1.2, we introduced the role of the oblivious adversary in online problems. In the VeChain Decision Problem, the oblivious adversary represents the circumstances that determine the number of days that the user will use the VeChain network. For our model, we assume that these circumstances are completely random and not influenced by anything. This means that the oblivious adversary simply assigns a random day from a uniform distribution in the range $\{1,...,L\}$ to each

user, where L is the length of the simulation in days. Users do not know the day that the adversary assigns to them, and are forced to switch to inactive on that day. Each user also has a state that keeps track of whether they are renting or have bought. Users always start out in the renting state but can choose to buy instead of rent from the first day. Each user has a size parameter, which indicates how much VTHO they use per day. A user's size can be used to determine how much VET they need to buy in order to generate the amount of VTHO that they need on a daily basis. This calculation is done using Equation 14, in which $user_{size}$ refers to the size of the user in VTHO/day, and G_{rate} refers to the amount of VTHO that is generated by a single VET in a single day. A summary of the important user parameters is given in Table 6.

$$VET_{needed} = user_{size}/G_{rate} \tag{14}$$

3.6.1 Daily process of a user

Each simulated day, all users are called on by the VeChainThor network model in random order. Once a user is called, they perform a sequence of actions that represent their actions on that day. A general overview of these actions is given in Figure 13. When a user is selected, they first check whether they are still active. If they are not, they will do nothing else. If the user is active, they will check on their state. A user that has already bought VET does not need to decide whether they want to buy. Therefore, these users can simply spent their generated VTHO by making transactions. Users that are still renting need to make the buy-or-rent decision and buy VET or VTHO based on that decision. After buying VET or VTHO, the user updates their expenses so that they can use their new expenditure data the following day. If a renting user decides to buy, they change their state to indicate that they have bought. After a user has spent the generated or bought VTHO, they check whether they require a different amount of VTHO in the future. If so, they calculate this amount based on the usage trend (set in the experiments) and update their size. Finally, active users check whether they have reached their final day of being active, by comparing the current day with their maximum number of active days. If a user has reached their final day, they deactivate themselves.

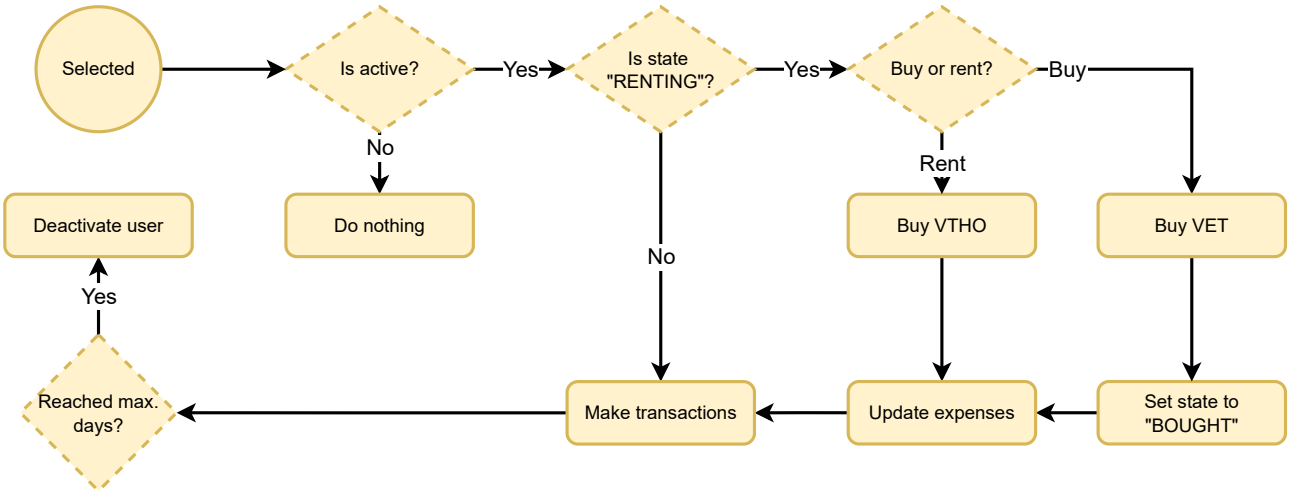


Figure 13: Actions of a user when selected by the VeChainThor network model.

3.6.2 Buy-or-rent decision

When making the buy-or-rent decision, a user uses their algorithm along with some of their personal metrics like the amount of money that they have spent on rent. Besides this, users can use the current buy and rent price in their decision making process. With constant token prices, the buy- and rent price can simply be obtained by multiplying the required amount of VET or VTHO with their respective prices. This is not the case when the user needs to acquire their tokens from an exchange. When the user needs to buy from an exchange, they require the current state of the exchange in order to determine what they will pay if they decide to rent or buy. Recall from Section 3.4.1 that the state of the exchange at each point in time is represented by a VET / USDT LOB and a VTHO / USDT LOB. The user can use these LOB's in order to calculate how much they will pay for each decision.

3.6.3 Keep-Renting user

In practice, it is possible that the VeChain network contains users that never want to make a long-term commitment to the network. In order to model users for which this is the case, we introduce the Keep-Renting user. The Keep-Renting user uses the KEEP-RENTING algorithm (A_{KR}), which says to keep renting forever.
As we introduce this user ourselves, it is interesting to determine the competitive ratio (CR) of A_{KR} (and therefore the Keep-Renting user) before analyzing its performance empirically. Take $R(t)$ to be the total amount of rent paid on day t and take $B(t)$ to be the buy price on day t. Take L to be the length of the simulation in days and let D be the final day $\leq L$ chosen by the adversary. Because we are concerned with simulations of limited length L, we want to define the CR of the Keep-Renting user for a simulation of that length. Recall that the optimal strategy is to buy on day 1 if $R(D) \geq B(1)$, and to keep renting otherwise. Defining the CR for a simulation of limited length means that we cannot ensure that it is possible that the adversary chooses a day that results in the total rent spent on that day being high enough such that the optimal solution is to buy on day 1. Simply put, we cannot guarantee that the length of the simulation L allows $R(D) \geq B(1)$. Therefore, we need to define the CR split over two scenarios: a scenario where $R(L) \geq B(1)$ (such that there exist D that make buying on day 1 the optimal solution), and a scenario where $R(L) < B(1)$ (such that the optimal solution is to keep renting for all possible D).
The worst-case scenario for the Keep-Renting user is when the adversary sets D equal to L. This is because this creates the maximum difference between the buy price at day 1, and the total rent paid at day D. The ratio between the cost of A_{KR} and the optimal strategy is either 1 (when the optimal strategy is to keep renting), or $\frac{R(D)}{B(1)}$ (when the optimal strategy is to buy on day 1). Combining those two facts, we find for the competitive ratio of A_{KR} in a simulation of the original Ski Rental Problem:

$$CR(A_{KR}) = \begin{cases} \frac{R(L)}{B(1)}, & \text{if } R(L) > B(1) \\ 1, & \text{otherwise} \end{cases}$$

3.6.4 Instant-Buy user

Representing the opposite of the Keep-Renting user, the Instant-Buy user models a user that instantly makes a long-term commitment. The Instant-Buy user uses the INSTANT-BUY algorithm (A_{IB}), which says to always buy on day 1. Recall that in the Ski Rental Problem and the VeChain Decision Problem, the buy price is higher than the rent price by definition. Clearly, this means that the worst-case scenario for the Instant-Buy user is when the adversary chooses day 1 as the final day. Keep the

definitions used in determining the CR of the Keep-Renting user. Given that the minimum length of a simulation is 1 day, the possibility of $D = 1$ always exists. Therefore, $R(1) < B(1)$ always holds as well. The above also means that the worst-case scenario for the Instant-Buy user can exist in every simulation. We therefore do not need to define it over multiple scenarios. In all simulations, the CR of the Instant-Buy user is given by Equation 15.

$$\frac{B(1)}{R(D)} \tag{15}$$

3.6.5 Random user

The Random user uses the RANDOM algorithm (A_{RANDOM}), and models a user that randomly chooses a day on which they will buy. The user selects a day in the range $\{1, ..., L\}$ from a uniform distribution, where L is the length of the simulation in days. The user starts out by renting and only buys if they are still active on the selected day.
Using the same definitions as in the previous two sections, we determine the CR of the Random user for a simulation of limited length L. No matter the optimal solution or the day that the adversary chooses, the highest cost of the Random user will always be obtained when they buy on their final day, since this results in the user paying the maximum amount of rent while also paying the buy price. The total cost of the Random user in the worst-case scenario is therefore $R(D-1)+B(D)$. This is always more expensive than to keep renting since the buy price is higher than the rent price by definition.
Similar to KEEP-RENTING and INSTANT-BUY, the RANDOM algorithm is an offline algorithm since it determines the day to buy on beforehand. However, because the algorithm does contain a random aspect, we use the worst-case *expected* ratio for the CR of the Random user just like for randomized online algorithms such as RAND.
Like with the Keep-Renting user, we need to consider that the limited simulation length might restrict us to a scenario where the optimal solution is to always keep renting. Therefore, we define the CR split over the two optimal scenarios like we did for the Keep-Renting user. Equation 16 shows the CR of the Random user when the optimal strategy is to keep renting, and Equation 17 shows the same for when the optimal strategy is to buy on day 1. In these equations, $\mathbb{E}$ refers to the expected value, and $\sup_{t \in L}$ refers to the supremum over the set of possible days L that the adversary can choose. All other variables are the same as defined in this or the previous two sections.

$$CR(A_{RANDOM}) = \sup_{t \in L} \frac{\mathbb{E}[R(t-1)+B(t)]}{R(t)} \tag{16}$$

$$CR(A_{RANDOM}) = \sup_{t \in L} \frac{\mathbb{E}[R(t-1)+B(t)]}{B(1)} \tag{17}$$

3.6.6 Deterministic user

The Deterministic user uses the optimal deterministic algorithm (DET) introduced in Section 2.2.

3.6.7 Rand user

The Rand user uses the optimal randomized algorithm (RAND) introduced in Section 2.2.

3.6.8 A-Adapted user

In an attempt to find an online algorithm that performs well under the fluctuating prices of the VeChain Decision Problem, we adapt Strategy A from [20] to our setting. We'll call the adapted algorithm A-ADAPTED and the user that implements it the A-Adapted user. Recall from Section 2.2.3 that the CPI Ski Rental Problem problem from [20] assumes a constant ratio μ between the buy price and the rent price, and that they limit the maximum price fluctuations by a maximum fluctuation ratio α. Both these assumptions do not apply in our setting. The ratio of the buy price and the rent price can freely change and price fluctuations are not limited given that any number of other users could potentially influence the price before a user's next decision. If we want to adapt Strategy A to the VeChain Decision Problem, we therefore need to determine how we set μ and α in A-ADAPTED. With regards to the buy-to-rent ratio μ, all that matters when deciding whether to buy on day t is the buy-to-rent ratio on that day (assuming that we cannot predict future ratios). Because of this, we can simply change μ to be a time-dependent variable $\mu(t)$ that gives the buy-to-rent ratio at time t.
Since a maximum fluctuation ratio is not defined in our problem, doing the same for α is impossible. Intuitively, the *achieved* maximum fluctuation ratio α_a seems like a good replacement since we could argue that the maximum of all achieved fluctuation ratios is a good estimate of the maximum fluctuation ratio. Of course, we cannot use the maximum of all achieved fluctuation ratios in the calculation of γ, as we can only determine this value at the end of the entire simulation. The next best thing is to make α_a time-dependent as well and to keep updating it during the simulation. Therefore, we replace the maximum fluctuation ratio α of Strategy A with $\alpha_a(t)$: the maximum achieved fluctuation ratio at time t. As no fluctuations have taken place on the first day, we use $\alpha_a(1) = 1$.
Having adapted μ and α to the VeChain Decision Problem, we have found a way to define γ for A-ADAPTED that comes as close as possible to the original definition of γ in [20]. The general strategy of A-ADAPTED remains equal to Strategy A. For clarity, we repeat this strategy in Algorithm 4. Equation 18 shows how γ is defined in the A-ADAPTED algorithm. Similar to Strategy A, T is the highest value of t for which the inequality $1 + \alpha_a(t) + ... + \alpha_a(t)^{T-1} \leq \mu(t)$ holds. It should be noted that like with Strategy A, the CR of the A-ADAPTED algorithm is equal to that of DET for the original Ski Rental Problem.

Algorithm 4 A-ADAPTED

Keep renting from the original time and buy at time t, where t satisfies the condition
$R_t + r_t \geq \gamma P(t), \gamma \leq 1$.

$$\gamma = \frac{1 + \alpha_a(t) + ... + \alpha_a(t)^{T-1}}{\mu(t)} \tag{18}$$

3.7 Metrics

In order to examine the performance and adoption behavior of the users in the VeChain Decision Problem, we need to define and quantify these terms. Section 3.7.1 specifies how we quantify and track user performance, and Section 3.7.2 does the same for adoption behavior.

3.7.1 User performance

Users need to keep track of a couple of metrics in order for us to determine their performance. Specifically, we need the cost of the user and the cost of the optimal offline algorithm for the same input of

that simulation. In order to determine the cost of the user, each user simply keeps track of the amount of money that they have spent on rent and on buying. The cost of the optimal offline solution is the minimum between the buy price on day 1, and the cost of renting for the entire time. In order to know the cost of the optimal offline solution for the exact same scenario that the user encounters, each user needs to record the price that they would have paid if they had bought on the first day. In addition, the user needs to keep track of the total money that they would have spent on rent if they kept renting for the entire period that they are active. This means that they need to record the price of renting, even if they have already bought.

In order to start answering the questions that this study asks, it is important to quantify user performance and adoption behavior. The most straightforward measure for user performance would be the competitive ratio, as this is the established metric for online algorithm performance. However, recall from Section 2.1.3 that the competitive ratio is defined as the maximum difference between the cost of an algorithm and the cost of the optimal offline algorithm, *over all inputs*. Due to the large feature space of the VeChain Decision Problem, we cannot expect to encounter each possible scenario in the experiments that we perform. This makes it difficult to empirically find the CR of the users in the VeChain Decision Problem. To separate a user's theoretical performance (CR) and their performance in the experiments, we judge the performance of the users by their achieved competitive ratio (ACR). We define ACR as the ratio between the total cost paid by the user and the total cost that the optimal user would have paid in that exact same scenario. Equation 19 shows how to calculate the ACR of algorithm A, in which $Cost_A(i)$ is the cost of algorithm A for input i, and $Cost_{OPT}(i)$ is the cost of the optimal offline algorithm for that same input. This definition means that like with CR, optimal performance yields an ACR of 1.0, and higher ACR values indicate worse performance. It also means that we can calculate an ACR value for each simulation separately. To compare the performance of the users in an experiment, we consider both the mean ACR and the maximum ACR over all simulations. Doing so allows us to compare the users' performance on two levels. The maximum ACR is most like the CR definition for deterministic algorithms, as its value will approach the theoretical maximum with more simulations. However, its value is very dependent on whether any existing local maxima are encountered in the simulations. The mean ACR, on the other hand, gives us a more general measure of performance. In practice, the maximum ACR could be seen as a more useful metric for risk-averse users, while the mean ACR might be more indicative for users that are willing to take a little more risk.

$$ACR(A) = \frac{Cost_A(i)}{Cost_{OPT}(i)} \tag{19}$$

3.7.2 Adoption behavior

In order to say something about the adoption behavior of network users, it is first necessary to define adoption. A user is said to adopt the network when they switch from renting to buying. This makes sense, because buying VET suggests a long-term commitment. If you do not know the algorithm that a user uses, you have to assume that them buying VET means that they expect to use the network for a longer time period. You would expect them to keep renting otherwise. Based on the chosen definition of adoption, the most interesting adoption metrics are deemed to be the network adoption ratio and -speed. We define the network adoption ratio as the fraction of network users that has switched from renting to buying at any point in time. Equation 20 shows the formal definition, in which $N_{bought}(t)$ refers to the number of users that have decided to buy at time t, and N is the total number of users that have ever been active during the simulation. To get a measure of adoption speed, we can simply look

at the day on which a user decides to buy. Here, buying earlier indicates faster adoption and buying later indicates slower adoption. As users that keep renting do not have a buying day, we only consider the adoption speed of users that actually decide to buy.

$$r_{adoption}(t) = \frac{N_{bought}(t)}{N} \tag{20}$$

3.8 Experiments

In order to thoroughly investigate the research questions, multiple experiments are performed. We start by recreating the original Ski Rental Problem with our model, and we build up to a multi-agent simulation of the VeChain Decision Problem by systematically enabling features of the model. By not adding all features at once, we can more precisely study the effects of each individual feature. All experiments use the default settings summarized in Table 7, unless otherwise indicated.

Parameter	Default value
Simulation length	3650 days
Generation rate	0.000432 VTHO per VET per day
User size	1 VTHO per day
VET starting price	0.0235 USDT
VTHO starting price	0.0015 USDT
Circulating VET	86,712,634,466 VET
Circulating VTHO	38,396,354,542 VTHO
VET liquidity ratio	0.00674
VTHO liquidity ratio	0.01226
Speculative price trend	Stable

Table 7: Default parameters used in the experiments when a different value is not specified.

We now first give an overview of all experiments, along with their goal. Next, sections 3.8.1 through 3.8.5 detail the experimental set-up of the experiments.

3.8.1. **Original Ski Rental Problem**
For starters, the original Ski Rental problem is replicated using our model. This experiment is first of all used to validate the correct implementation of the problem and the online algorithms. In addition, it allows us to examine the effect that a finite simulation has on the performance of the algorithms.

3.8.2. **Single-agent**
Next, we study the most basic form of the VeChain Decision Problem by setting the model to its default settings and determining the performance of each user separately. The goal here is to establish a baseline for the users' ACR in the VeChain Decision Problem, when economic influences are not considered.

3.8.3. **Single-agent + exchange**
In this experiment, the influence of the limit order books of the exchange are investigated.

The goal of this experiment is to establish the competitive ratios that are achieved when prices fluctuate due to a user's interactions with the exchange. Moreover, we examine the effect of the user's size on their performance in simulations with limited length.

3.8.4. **Single-agent + exchange + speculation**
Here, the effects of speculative price trends are investigated in the single-user setting. Competitive ratios are established for different buy-to-rent price trends. Furthermore, this experiment focuses on the effect that the price trends have on the mean adoption ratio and -speed.

3.8.5. **Multi-agent**
Lastly, this experiment focuses on user performance in a multi-agent setting. The goal is to identify how the users perform when other users of the VeChainThor network apply the same or different algorithms.

3.8.1 Original Ski Rental Problem

In order to ensure a correct implementation of the problem and the algorithms, the original Ski Rental Problem is simulated using our model of the VeChain network. To recreate the original problem, we initialize a single user and we set some of the model parameters different to their default settings. The parameters that are changed from their default value are summarized in Table 8. By setting the generation rate to 1, the user needs to either buy 1 VTHO per day or 1 VET once. The price of VET is set to 10 USDT and the price of VTHO is set to 1 USDT, as these are the most commonly used example values in previous research of the original Ski Rental Problem. The simulation length is set to 50 days because this is long enough to study the different behavior of the algorithms. The exchange and the speculative price trends of our economy are not yet considered, which means that the user can always buy/rent for the set prices.

Parameter	**New value**
Simulation length	50 days
Generation rate	1 VTHO per VET per day
VET starting price	10 USDT
VTHO starting price	1 USDT

Table 8: Model parameters that are changed from their default values, in order to recreate the original Ski Rental Problem.

We evaluate the performance of each user (Random, Keep-Renting, Instant-Buy, Deterministic, Rand, and A-Adapted) on the Ski Rental Problem separately. To this end, we run the same 10,000 simulations of the experiment for each user, and we calculate their mean and maximum ACR over the entire set of simulations.

3.8.2 Single-user

In this experiment, we start setting up the VeChain Decision Problem. To this end, all model parameters are set to their default values. This means that the initial settings of the VeChainThor network and the token prices are as close to the real-world settings of the VeChain Decision Problem as possible. We investigate the performance and behavior of a single user that uses 1 VTHO / day. Like with

the original Ski Rental Problem, the user can rent and buy for constant prices as we do not enable the exchange or price trends. As in the previous experiment, we evaluate the performance for each user individually. We again run the same 10,000 simulations of the experiment for each user, and we calculate their mean and maximum ACR over the entire set of simulations.

3.8.3 Single-user + exchange

This experiment extends the model used in the previous experiment by enabling the exchange introduced in Section 3.4.1. This means that the user now has to buy their tokens through the limit order books of the exchange, rather than being able to acquire them for a set price. As mentioned in Section 3.4.1, the exchange only considers market orders. Because of this assumption, user size (the amount of VTHO that a user uses on a daily basis) becomes a relevant factor to performance. Due to the nature of limit order books, larger users will move the price more (and thereby pay more) than smaller users when placing a market order.
For the user to be representative of common real-world users, we would ideally like the user size to be as small as possible. However, this might not be possible due to the minimal size that is required to move the price through the LOB's (price only moves if the user buys the entire quantity present in a tick). If the user is too small to move the price through the LOB's, the exchange would not be relevant as the user could keep buying tokens for the same constant price as in the previous experiment. Furthermore, the limited simulation length can make it so that smaller users move the price so little that they are better off by renting even if the adversary chooses the last day. In this scenario, the combination between user size and simulation length causes the results to not be representative of the real world with a theoretically unlimited length. Because of these limitations of our model, we cannot set the user size to be as small as we want and expect the results to be representative. The experiment from this section can help us choose a user size that makes the results representative while using a simulation length that keeps the simulations computationally tractable.
In order to investigate the effect of user size on performance in this setting, we study the performance of each user as a function of their size. We consider a user size of $x \cdot G$, where G is the amount of daily generated VTHO (37,459,858 VTHO), and x is varied in the range $\{0.05, 0.1 \ldots, 1.0\}$. This means that we consider a user size of 5% to 100% of the daily generated VTHO. It would theoretically be possible that a user uses more than 100% of the daily generated VTHO. However, this would mean that they cannot buy enough VET to supply themselves with the required amount of VTHO (they would need more than all VET in existence). If the user is unable to buy enough VET, studying the buy-or-rent decision makes little sense. Besides, remember from Section 2.4 that the VTHO generation rate can be increased through a vote. The long-term impracticality of more VTHO being used than generated means that we can safely assume that such an increase happens before or shortly after VTHO use exceeds VTHO generation. To ensure an honest comparison, we run the same 10,000 simulations for each combination of user and user size. The mean and maximum ACR is then calculated for each combination separately.

3.8.4 Single-user + exchange + speculation

For this experiment, the model is extended further by including the speculative price trends introduced in Section 3.4.2. As the online algorithms mostly base their actions on some form of the buy-to-rent ratio, it is worthwhile to investigate the effects of trends in the buy-to-rent ratio on user performance and -behavior. To this end, we compare the performance and behavior of the users as a function of three different speculative price trends: a stable VET price, an increasing VET price, and a decreasing

VET price. This corresponds to a stable buy-to-rent ratio, an increasing buy-to-rent ratio, and a decreasing buy-to-rent ratio. Because the exchange is also enabled, the effects from the speculative price trends are an addition to any usage-related price trends.
This experiment uses all default values except for the user size. As explained in the previous section, the combination between user size and simulation length is an important factor when including the exchange in our simulations. The user size is therefore set based on the exchange experiments from the previous section. We run 10,000 simulations for each combination of user and speculative price trend, and combine the results.

3.8.5 Multi-agent

Having established the performance and behavior of the users in different single-user scenarios of the VeChain Decision Problem, we perform a simple multi-agent experiment as an initial exploration into multi-agent buy-or-rent decisions. To this end, we examine the performance of the A-Adapted user for different scenarios in which three other agents try to solve the VeChain Decision Problem as well. We consider three different agent compositions: Random, Uniform, and A-Adapted. In the Random composition, all other agents are Random users. In the Uniform composition, the three other agents are the Random user, the Deterministic user, and the Rand user. In the A-Adapted composition, all four agents are A-Adapted users. We test each composition setting under the same buy-to-rent price trends that we considered in the previous section (stable, increasing, and decreasing).
We use a simulation length of 7300 days (20 years) and set the size of each user to 0.20 times the VTHO generation rate. These changed settings are also specified in Table 9. We run 10,000 simulations for each combination of the compositions and buy-to-rent price trends. For each combination, we calculate the mean and maximum ACR of the same A-Adapted user.

Parameter	**New value**
Simulation length	7300 days
User size	$0.20 * G$ VTHO per day

Table 9: Model parameters that are changed from their default values for the multi-agent experiment. Here, G is the daily VTHO generation rate of 37,459,858 VTHO / day.

3.9 Implementation

The entire model is implemented using the Mesa framework (v. 0.9.0) [30] on Python 3.9.7. Mesa is a framework that aids in developing agent-based models. The most important features of Mesa that this study uses are the data collection module, the scheduler, and the batch run feature. The data collection module allows us to keep track of agent- and model data at any frequency that we require. The scheduler is used to handle the (in our case random) sequence in which the agents act. Lastly, the batch run feature allows for an easy way of running sets of experiments over multiple iterations.
All experiments performed in this study are run on a Lenovo Thinkbook 15-IIL with 16GB RAM and an Intel Core i7-1065G7 processor at 1.30GHz. The experiments are run using the batch run feature of the Mesa framework. Experiments are generally run through Jupyter Notebooks, but some of the longer experiments are run using separate Python files as the Notebooks do not seem to work well with the batch run feature of Mesa. The entire code base for this study can be found on GitHub [31].

4 Results

4.1 Original Ski Rental Problem

The original Ski Rental Problem was simulated using our model of the VeChain Decision Problem, as outlined in Section 3.8.1. This experiment had three main objectives. The first objective was to validate that the problem as well as the existing algorithms were implemented correctly. Secondly, this experiment allowed us to empirically determine the performance of the users that were introduced in this study (the Random, Keep-Renting, Instant-Buy, and A-Adapted user), for the original Ski Rental Problem. The third objective of this experiment was to obtain performance data from a simulation in order to understand the differences between the theoretical CR of the algorithms and the maximum- and mean ACR achieved by the users.
This last point is important, since the differences between a simulation with limited length and infinite theoretical inputs might make it so that the achieved competitive ratios of the users deviate from their theoretical values. We focus the analysis of the results in this section on identifying these differences. The main results of the experiment can be found in Table 10, which contains the mean- and maximum ACR of each of the users. Additionally, to get a better idea of the distribution of the ACR values, Figure 14 shows a violin plot of the achieved competitive ratio by user. Lastly, Figure 15 shows the mean ACR of the different algorithms, as a function of the number of days chosen by the adversary.

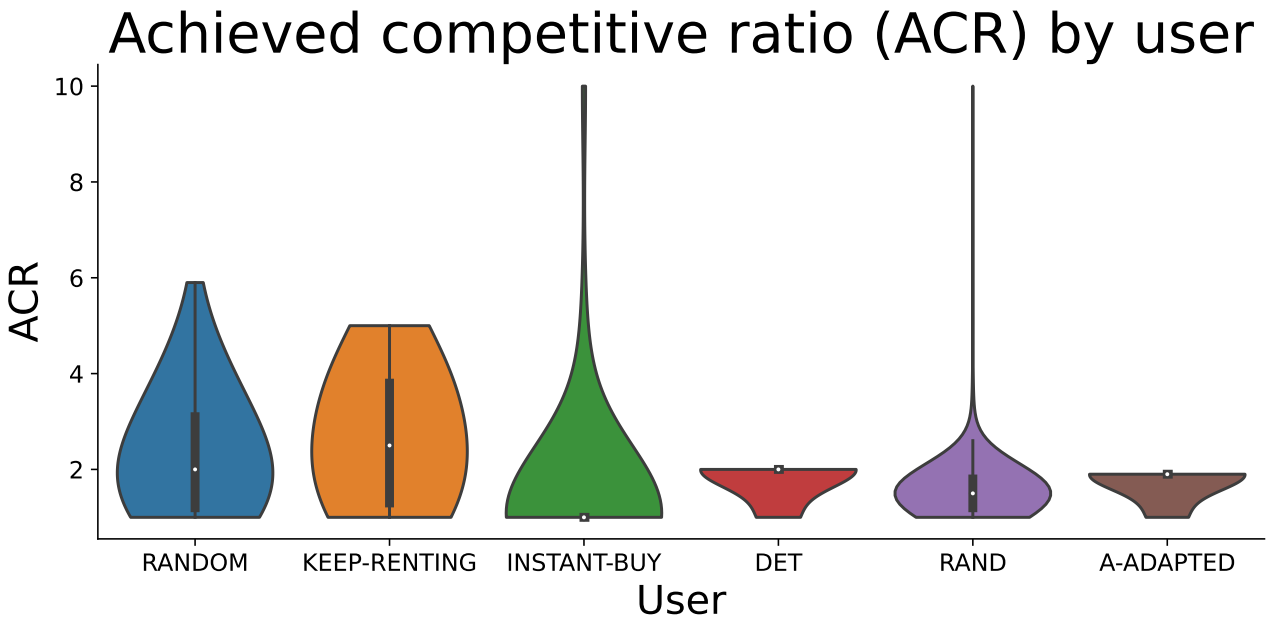


Figure 14: Violin plot of the achieved competitive ratios per user. All violins were scaled to the same width and the white dots indicate the median ACR.

	RANDOM	**K-RENTING**	**I-BUY**	**DET**	**RAND**	**A-ADAPTED**
Mean ACR	2.28	2.63	1.39	1.80	1.53	1.73
Max. ACR	10.00	5.00	10.00	2.00	10.00	1.90

Table 10: Mean- and maximum ACR achieved by the users in the original Ski Rental Problem.

We note from Table 10 that the Random user achieved the second worst mean ACR of 2.28, and shared the worst maximum ACR of 10.0. Moreover, we see in Figure 15 that the Random user achieved their worst-case performance when they are only active for one day. This is however also when they achieved their best mean ACR. As can be seen from Figure 15, the mean ACR of the Random user increased when they were active for more days. This can be explained by the workings of the RANDOM algorithm. Because the Random user buys on a random day from the entire simulation, the probability of having bought on one of the first days is low. Consequently, the probability of making the optimal decision when the adversary chooses one of the first days is high and the mean achieved CR will be low. This probability will increase for longer simulations, as this makes it so that the probability of deciding to buy on a single day decreases.

Table 10 shows us that the Keep-Renting user achieved a mean ACR of of 2.63 and a maximum ACR of 5.00. We find using the cases outlined in Section 3.6.3 that this maximum ACR is equal to the CR of the Keep-Renting user, for this scenario of length 50 and with constant buy and rent prices of \$10 and \$1: $\frac{50}{10} = 5.00$. As expected, Figure 15 shows that the Keep-Renting user performed optimally when $R(D) \leq B(1)$ (as renting is equal to the optimal strategy in those scenarios). However, once $D > B$, the mean ACR increased linearly with $R(D)$. Because of this linear increase, the mean ACR of the Keep-Renting user is highly dependent on the simulation length. The mean ACR of 2.63 is already the highest out of all users, but will also keep increasing when simulation length increases.

Also from Table 10, we note that the maximum ACR of the Instant-Buy user was 10.0. This again is in line with the expectations for that user in the current settings. We know from Section 3.6.4 that the worst-case scenario for the Instant-Buy user was when the adversary chose day 1 as the final day. With $B(1)$ being \$10, we find using Equation 15: $\frac{10}{1} = 10.0$. Although the Instant-Buy user shared in the worst maximum ACR, they also achieved a mean ACR of 1.39, which is the best performance of all users. Again, the reason for this performance stems from the settings of the experiment. As can be seen in Figure 15, the Instant-Buy user performed (as expected) much worse when the adversary chose a low number of days, and optimally when the adversary chose a number of days $\geq B(1)$. The setting of $B(t)$ in combination with the length of the simulation makes it so that the latter scenario is more likely, which in turn leads to the Instant-Buy user outperforming the other users on average. For the same price settings, we would expect the Instant-Buy user to have its mean ACR trend towards optimal performance with increasing L.

We now shift our attention to the users that use existing solutions to the Ski Rental Problem. From Table 10, we note that the Deterministic user achieved a maximum ACR of 2.00, which is in line with the theoretical CR of the DET algorithm. Figure 15 shows us that for each day, the Deterministic user performed as we would expect from the DET algorithm: they achieved a mean ACR of 1.00 when $R(D) \leq B(1)$ and and a mean ACR of 2.00 when $R(D) > B(1)$. Table 10 shows us that the Deterministic user achieved a mean ACR of 1.80. It should again be noted that this mean ACR value is highly dependent on the simulation length. Given that the Deterministic user achieves an ACR of 2.00 when $R(D) > B(1)$, we would expect the mean ACR of the Deterministic user to trend towards 2.00 when L increases.

Table 10 indicates that the Rand user achieved a maximum ACR of 10.00 and a mean ACR of 1.53. The maximum ACR of 10.00 is in line with the theory, as this can occur when the randomization leads to the user choosing to buy on the first day while the adversary picks that day as well (like with the Instant-Buy user, the ACR is then determined as $\frac{10}{1} = 10.0$). However, recall from Section 2.1.3 that the CR of a randomized algorithm is defined as the worst-case *expected* ratio. This means that for this user, the CR of the RAND algorithm cannot be compared with the maximum ACR. Instead, we need to compare it to the worst case mean ACR over all days that the adversary could choose. The results indicate that in the simulations, this worst-case mean ACR was approximately 1.63, and was achieved

when the adversary chose day 6. Both the mean ACR and the worst-case mean ACR approximated the CR of the RAND algorithm at 1.58. We expect the worst-case mean ACR to trend towards this value with increased simulation length and iterations (it would not necessarily be achieved at day 6).

Lastly, Table 10 shows us that the A-Adapted user achieved a mean ACR of 1.73 and a maximum ACR of 1.90. This maximum ACR is another clear example of how the ACR values from our simulations with limited length can differ from the theoretical CR values. Recall that the CR of the A-Adapted user is equal to that of the Deterministic user for the original Ski Rental Problem. Because the A-ADAPTED algorithm is deterministic, we would expect the maximum ACR that is achieved in the simulation to equal the CR of the A-ADAPTED algorithm (we did observe this for the Deterministic user with the DET algorithm). We also know that DET was proven to be the optimal deterministic algorithm for the original Ski Rental Problem. At first sight, it should therefore not be possible for the A-ADAPTED algorithm to achieve a better maximum ACR than DET as we see here. The reason that this does happen can be explained by the limited length of the simulation and the workings of the algorithms. Take $r(t)$ to be the rent price and take $R(t)$ to be the rent spent at time t. Recall that DET bought when $R(t) \geq B(t)$. Recall also that for the A-ADAPTED algorithm, we have $\gamma = 1$ for the original Ski Rental Problem. Therefore, A-ADAPTED always bought when $R(t) + r(t) \geq B(t)$. Because of this, A-ADAPTED always bought one day sooner than DET. Therefore, it only spent $R(t) - r(t) + B(t)$ in the worst case scenario while DET spent $R(t) + B(t)$. This difference translates itself to the maximum ACR. With a buy price of \$10 and the rent price still at \$1, we find for A-ADAPTED: $\frac{10-1+10}{10} = 1.90$. For DET we find the expected maximum ACR of $\frac{10+10}{10} = 2.00$. The fact that this difference occurs is due to the ratio between the buy price $B(t)$ and the rent $r(t-1)$. When this ratio increases, the ratio between $R(t)$ and $r(t-1)$ increases as well. This means that the difference between $R(t) + r(t-1)$ and $R(t)$ becomes negligible and the cost of the A-ADAPTED algorithm trends towards that of DET.

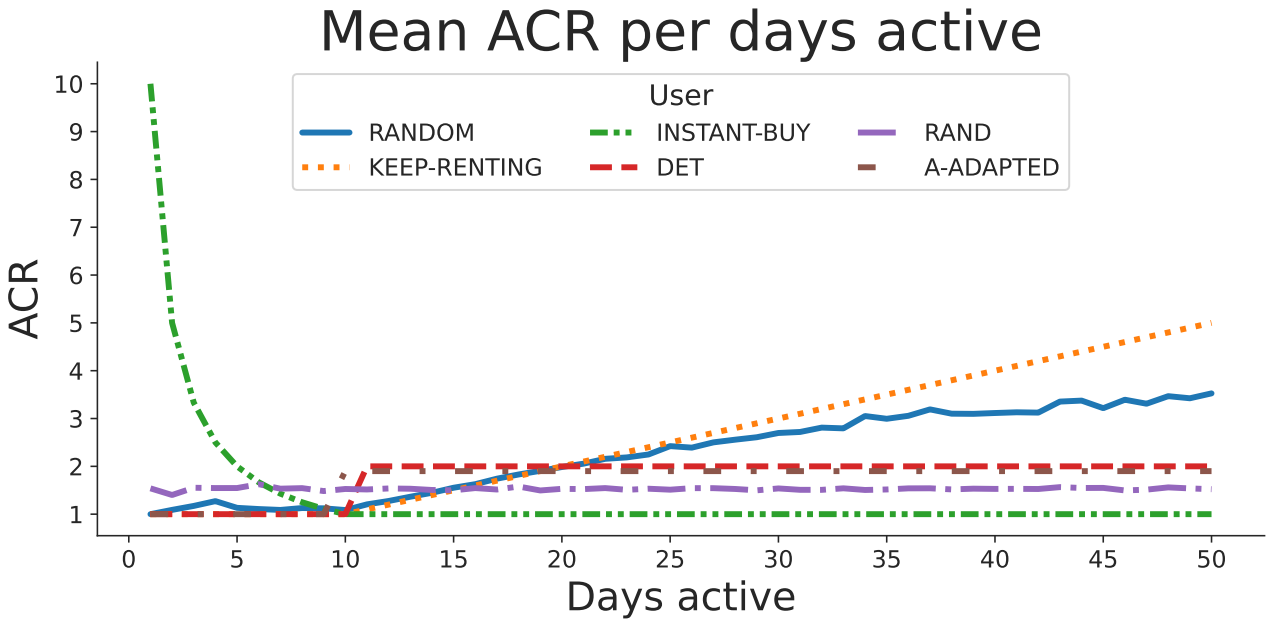


Figure 15: Mean ACR as a function of the number of days that the user was active (as selected by the adversary), for each of the users.

4.2 Single-user

Here, the results of the experiment introduced in Section 3.8.2 are discussed. Recall that this experiment is equal to the original Ski Rental Problem, just with the generation rate and prices from the VeChain Decision Problem. Table 11 summarizes the performance of the different users for this experiment. We can see that the Keep-Renting, Deterministic, and A-Adapted users all achieved a mean and maximum ACR of 1.00. We can discover the reason for this optimal performance by looking at the buying behaviour of those users. Because the Keep-Renting user achieved a mean ACR of 1.00, we conclude that the optimal solution was to keep renting in all simulations. This conclusion is strengthened by the optimal performance of the Deterministic and A-Adapted users, as this indicates that the total rent never exceeded the buy price. The performance of the Random and Rand users suggests that their randomization made it so that they did buy sometimes. Given that we have established that the total amount spent on rent never exceeded the buy price, we conclude that purely the randomisation feature of these users was the reason for their higher ACR values. Recall that these simulations had a length of 10 years. The fact that the strategy to keep renting was optimal for all simulations suggests that the VeChain Decision Problem is irrelevant if we assume a maximum usage length of 10 years and no price influences. Thus, the results of this experiment emphasize the importance of modelling the economic influences if we want to investigate the VeChain Decision Problem and answer the research questions of this study.

	RANDOM	**K-RENTING**	**I-BUY**	**DET**	**RAND**	**A-ADAPTED**
Mean ACR	10.51	1.00	100.77	1.00	1.51	1.00
Max. ACR	772.58	1.00	36265.43	1.00	271.53	1.00

Table 11: Mean- and maximum ACR achieved by the algorithms in the VeChain Problem with constant prices.

4.3 Single-user + exchange

The experiment described in Section 3.8.3 was performed in order to examine the performance of the different users when they affect their own future buy and rent prices by having to buy the tokens from an exchange. Figure 21 shows the mean achieved ACR as a function of the user size, for each of the different users. The maximum ACR values are not too noteworthy and can be found in Appendix B.

An important initial observation from Figure 21 is the fact that the mean ACR of the Deterministic and A-Adapted algorithms remained constant at a value of 1.0 for user sizes 0.05 to 0.25. Examining Table 17, we note that this is also the maximum ACR that the users achieved for these sizes. Based on these values, we conclude that the optimal strategy was to always keep renting for those user sizes. This means that for user sizes 0.05 to 0.25, the effects of the single user buying from an exchange did not affect the buy and rent prices enough to change the results observed in the results of the previous section. The mean ACR of the Deterministic and the A-Adapted users did start increasing around a user size of 0.25, and started to plateau around a user size of 0.60.

In line with these findings, we note that a user size of 0.60 was the size where the mean ACR of the Instant-Buy user started to flatten out a bit. It is also the size where the mean ACR of the Keep-Renting user started increasing heavily. For the Random user, we observe that the mean ACR reached its lowest point around a user size of 0.55, and that its performance was worse when the user size was to the lower- or higher end of the range.

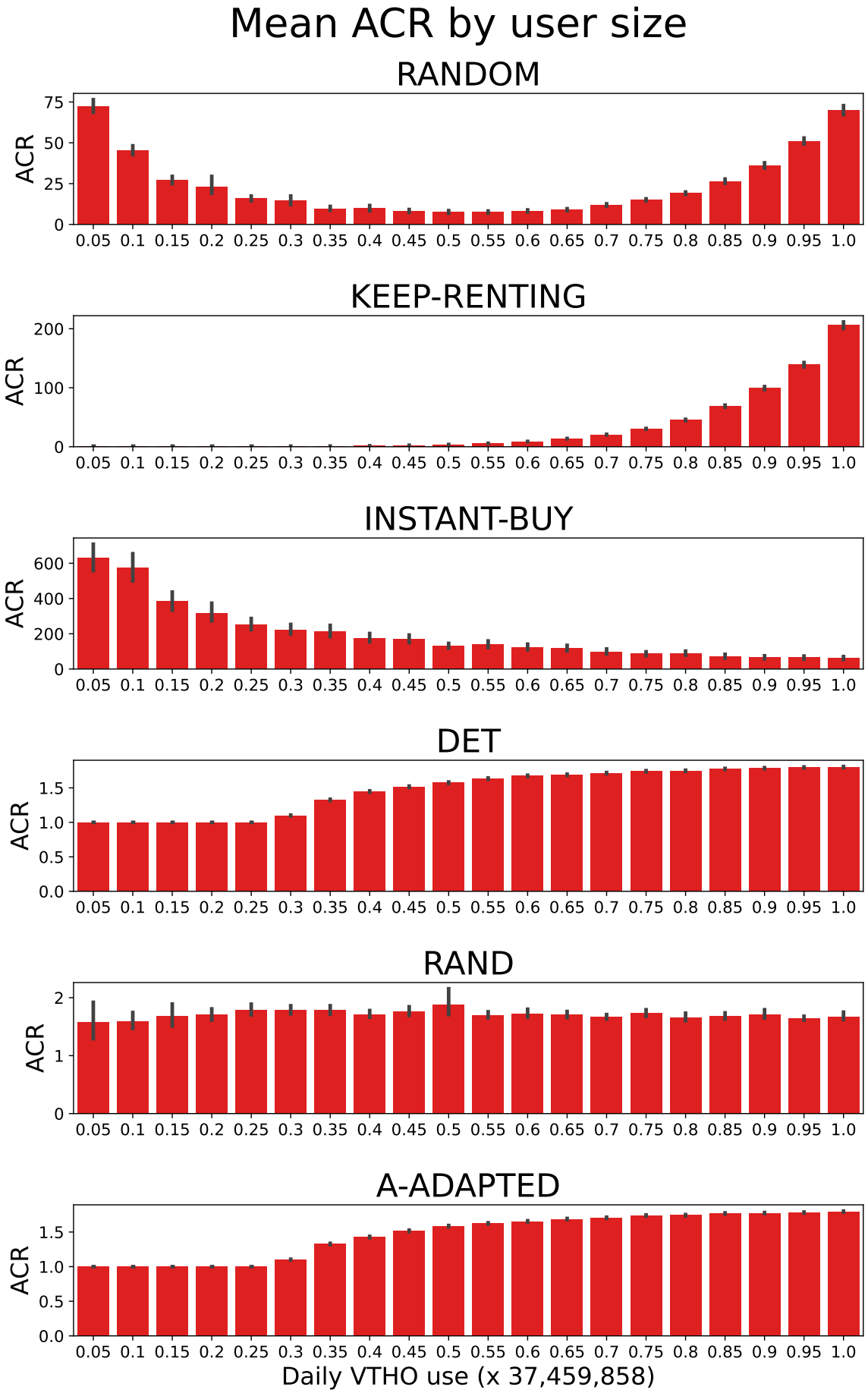


Figure 16: Achieved competitive ratio (ACR) by user size for the different users.

Recall that one of the objectives of this experiment was to find the best user size to use in further simulations of the VeChain Decision Problem with a single user. There were two criteria for the best user size. First of all, we wanted to minimize the user size to make it as realistic as possible. Secondly, we wanted the user size to be large enough so that it was not the case that the option to keep renting was optimal for most of the days chosen by the adversary. Based on the performance of the users around a user size of 0.60, we determined that this was a decent user size for which single-user experiments would start producing representative results. Further single-user experiments were therefore performed with a user size that set to 0.60 times the VTHO generation rate.

4.4 Single-user + exchange + speculation

In this section, we discuss the results from the experiment introduced in Section 3.8.4. One goal of this experiment was to investigate the influence of speculative price trends on the user's adoption ratios and -speeds. However, we first examined the effect of these trends on the performance of the users. Based on the results from the previous section, a user size of $0.60 \cdot G$ was used in the simulations, where G is the daily VTHO generation rate of $37,459,858$ VTHO / day. The mean ACR values of the users are shown in Table 12, and the maximum ACR values can be found in Table 13.

	RANDOM	**K-RENTING**	**I-BUY**	**DET**	**RAND**	**A-ADAPTED**
No trend	9.18	8.85	111.58	1.67	1.70	1.66
Uptrend	7.75	9.06	119.93	2.06	2.04	2.04
Downtrend	7.62	8.84	119.96	1.61	1.65	1.60

Table 12: Mean ACR achieved by the users in different buy-to-rent price trends.

An interesting observation from Table 12 is that all users achieved their highest mean ACR in the buy-to-rent uptrend condition, and their lowest mean ACR in the buy-to-rent downtrend condition. The same is not true for the maximum ACR of all users. Most interesting about the maximum ACR values is the fact that the Deterministic and A-Adapted users were able to maintain their theoretical CR values for the Ski Rental Problem of 2.00 when there was no trend or when the buy-to-rent ratio was in a downtrend. With a stable buy-to-rent price trend, the settings of this experiment are equal to the experiment with just the exchange, so it makes sense that the users achieved the same result in that case. In addition, we see that an uptrend in the buy-to-rent ratio made it so that the Deterministic and A-Adapted users' maximum ACR exceeded their Ski Rental Problem CR of 2.00. One reason why we observe this worse maximum ACR for the uptrend but not for the downtrend might be because the speculative downtrend was less severe than the uptrend, as mentioned in Section 3.4.2.

	RANDOM	**K-RENTING**	**I-BUY**	**DET**	**RAND**	**A-ADAPTED**
No trend	2418.67	38.82	39783.07	2.00	159.74	2.00
Uptrend	2697.20	39.63	26227.17	2.73	69.04	2.77
Downtrend	903.02	39.08	29595.93	1.99	283.22	2.00

Table 13: Maximum achieved competitive ratios of the algorithms for the different buy-to-rent price trends.

One last observation from the user performance data is that the Deterministic and the A-Adapted user

performed nearly identical. Since the A-ADAPTED algorithm is designed in a way that should make it perform well in an environment with fluctuating prices, we would have expected for the A-Adapted user to significantly outperform the Deterministic user. In order to investigate this finding further, we created a violin plot of the γ-values of the A-Adapted user on the day that they decided to buy (if they did decide to buy). This plot can be found in Figure 17. From the plot, we notice that the γ-value was very close to 1 when the user bought in any of the simulations. Recall that a γ-value close to 1 means that the A-Adapted user acted more or less like the Deterministic user in all of these simulations. Because the two users acted similarly, their performance was similar as well.

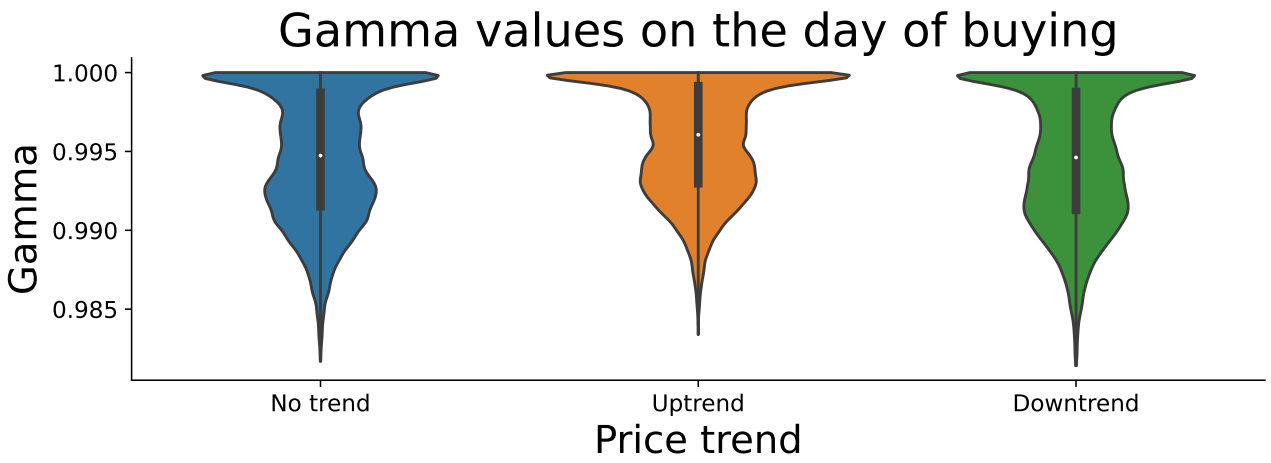


Figure 17: Gamma values of the A-Adapted user on the day that they decided to buy, for different buy-to-rent price trends.

Based on the workings of the A-ADAPTED algorithm, γ-values close to 1 can occur for two reasons: high buy-to-rent ratios or low fluctuation ratios. In order to investigate why the γ-values were close to 1 when the A-Adapted user decided to buy, we plotted the maximum fluctuation ratio and the buy-to-rent ratio over time for the experiment with the A-Adapted user and no speculative buy-to-rent price trend. This plot can be found in Figure 18. From the plot, we see that the buy-to-rent ratio started around 35000. This value is expected and and is due to the initial prices and default settings of the VeChain network. The ratio decreased quickly over time as users increased the VTHO price by buying it. The ratio settled around a value of 1000. The fact that the ratio settled can be either because users were not buying VTHO anymore, or because the effect of their buying leveled off. We also note from the plot that the maximum achieved fluctuation ratio started at 1 but quickly shot up to its maximum value of around 3.5. Comparing the maximum $\alpha_a(t)$ value of approximately 3.5 and the minimum $\mu(t)$ value of around 1000, we can see why the γ values remained close to 1. At no point in the simulation did the difference between the two values become small enough for there to exist a significant difference between the numerator and the denominator in Equation 18. Because of this, γ had no real impact on the buying behavior of the A-Adapted user. These results indicate that the A-Adapted user performed not much better than the Deterministic user in the VeChain Decision Problem because of the problem setting itself. The fluctuation ratios were simply too low and the difference between the buy- and rent price was too low for the A-ADAPTED algorithm to make a real difference.

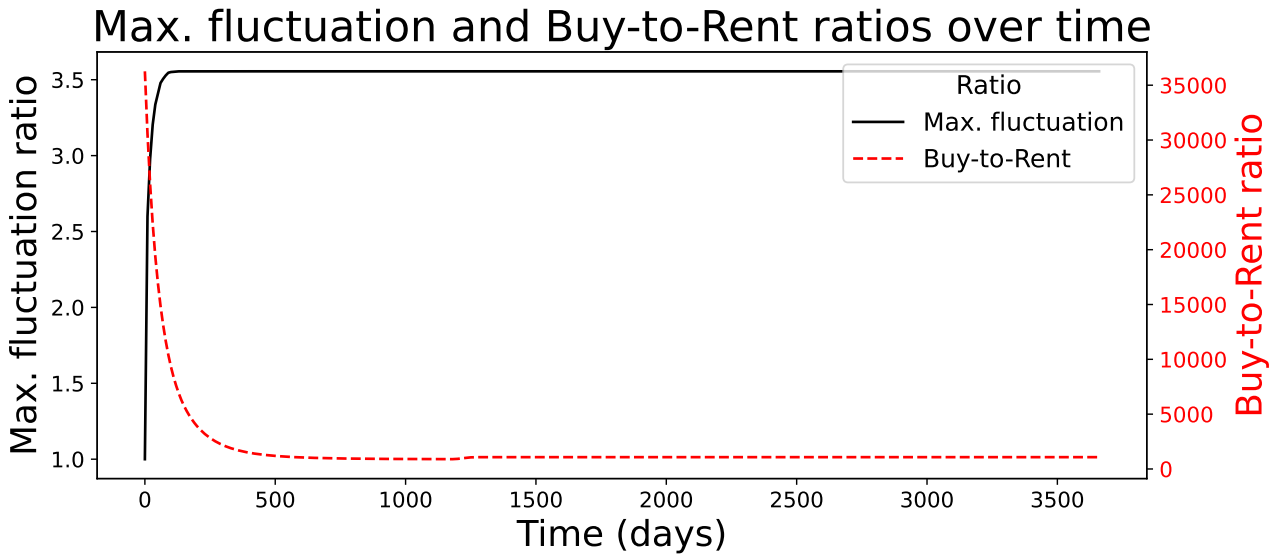


Figure 18: Maximum achieved fluctuation ratio and buy-to-rent ratio over time for the scenario with the A-Adapted user and no speculative buy-to-rent price trend. Values are averaged over 1000 simulations.

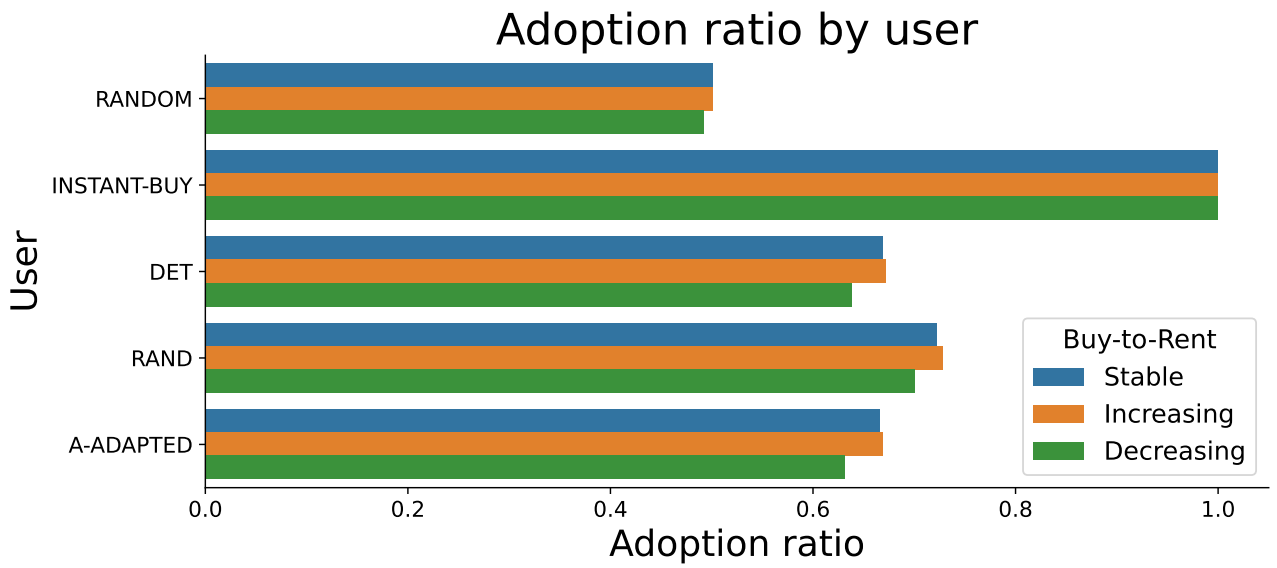


Figure 19: Mean adoption ratio of the different users for three different buy-to-rent price trend conditions: no trend, uptrend, and downtrend.

We now turn to the adoption behavior of the users in these different buy-to-rent price trends. Figure 19 shows the mean adoption ratio of the different users for the various buy-to-rent price trends. From it, we see that the Random user decided to buy approximately 50% of the time. Obviously, the Instant-Buy user always bought and therefore achieves a mean adoption ratio of 1.0. Focusing on the users that apply an online algorithm, we see that the Deterministic, Rand, and A-Adapted users all achieved a mean adoption ratio of between 0.60 and 0.75. Although the Rand user bought a bit more often than the other two algorithms. This last finding is probably due to the fact that the Rand user randomly buys more quickly at times. Focusing on these last three users, we see that an uptrend in the buy-to-

rent ratio resulted in the highest adoption ratio, while a decreasing buy-to-rent ratio resulted in the lowest.

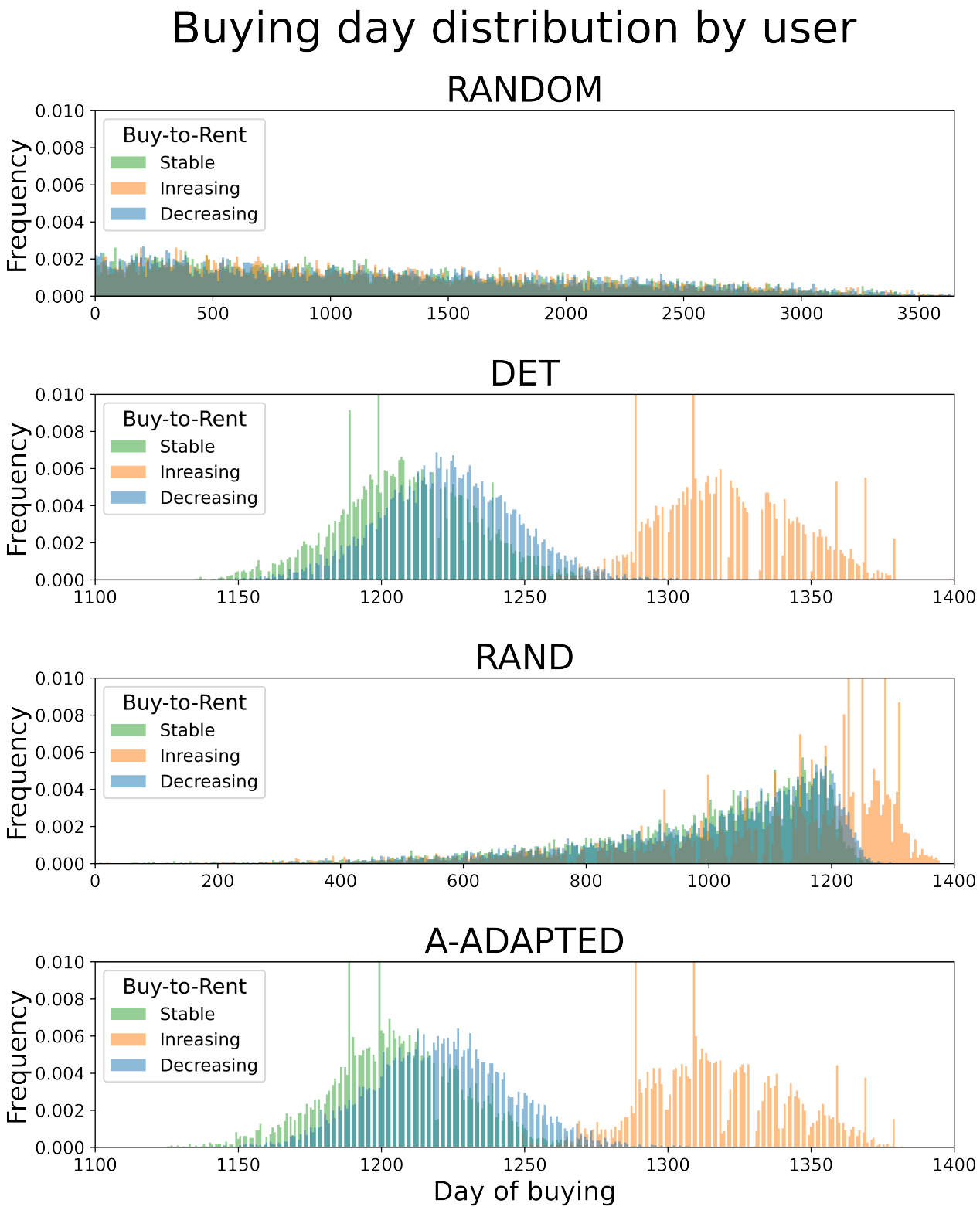


Figure 20: Buying day distribution of the Random, Deterministic, Rand, and A-Adapted user, for different buy-to-rent price trends. Note the varying ranges of the x-axes.

Besides the adoption ratio, the other adoption metric that we are interested in is the adoption speed, as measured by the days on which the users decided to buy. To examine the influence of the different buy-to-rent price trends on the adoption speed, we plotted a histogram of the buying days for each of the users except for the Keep-Renting and Instant-Buy user. These last two users were omitted since their buying behavior does not change. The plot is shown in Figure 20. The Random user does not seem to have been affected by the buy-to-rent price trends. This makes sense, since the RANDOM algorithm does not take the buy-to-rent ratio into account. We can see that the Random user bought on pretty much any day, but that they bought more often on earlier days than on later days. This is expected behavior, as the probability that the simulation ends before the user buys is higher for the later days.
Like with their performance, the adoption speeds of the Deterministic and the A-Adapted algorithms were similar. Both users bought at approximately the same days, and the range of their buying days was affected similarly by the buy-to-rent price trends. We note that the increasing buy-to-rent price trend shifted the buying days backwards in time by approximately 150 days. On the other hand, a downtrend in the buy-to-rent ratio shifted the buying days forward by approximately 25 days. Like with the ACR measures, this difference in severity of the effects might be explained by the difference in severity of the two price trends that we modelled.
The Rand user was affected by the different price trends as well, but this is only obvious for the uptrend and not for the downtrend. The latest buying day in the range of the Rand user was approximately similar to that of the Deterministic and A-Adapted user. However, the Rand user sometimes bought a lot earlier as well. This finding can be explained by the randomization factor of the RAND algorithm. Finally, we note that the distribution of buying days of the Rand user is left-skewed, which is what we would expect from the probability density function that the algorithm uses.

4.5 Multi-agent

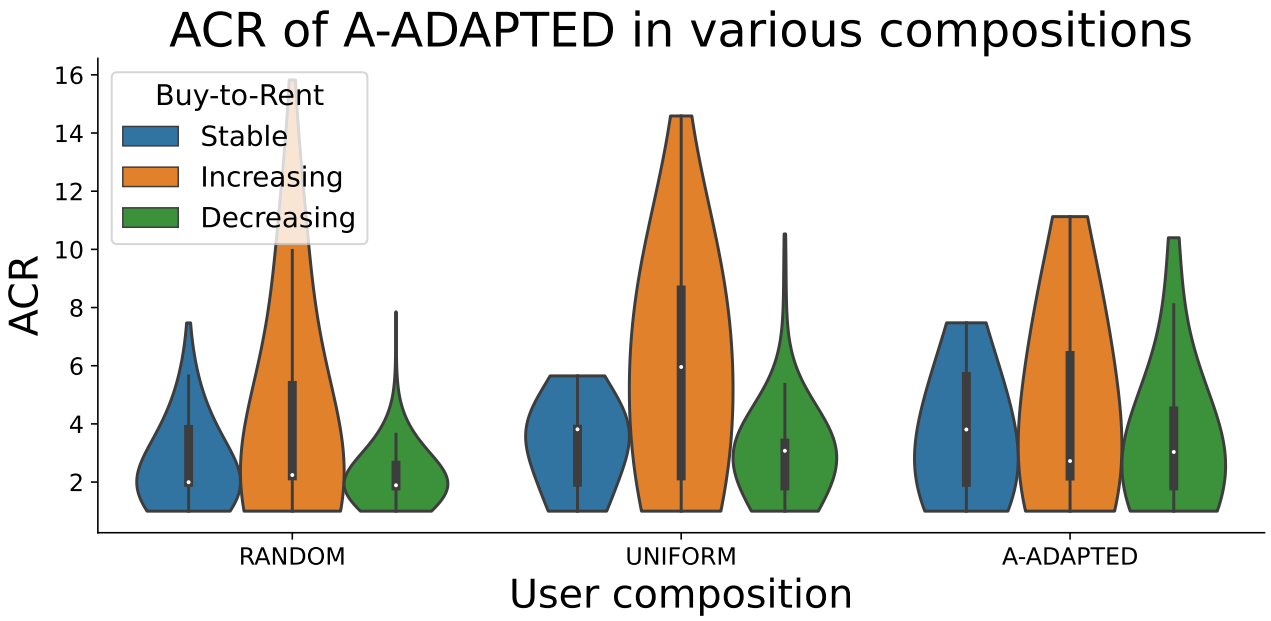


Figure 21: ACR of the A-Adapted user in multi-agent scenarios with different user compositions and buy-to-rent price trends. Violins are scaled to the same width and the white dots indicate the median values.

The experiment described in Section 3.8.5 was performed as an initial exploration into a multi-agent simulation of the VeChain Decision Problem. The performance of the A-Adapted user was determined in a multi-agent setting with multiple compositions and for different buy-to-rent price trends. Figure 21 shows a violin plot of the ACR values achieved by the A-Adapted user for each scenario. The mean ACR values for the same experiment are given in Table 14, and the maximum ACR values can be found in Table 15.
The results show that for all price trends, the A-Adapted user achieved its best mean ACR when all other users used the RANDOM algorithm. The worst mean ACR in the uptrend condition was achieved in the Uniform composition. The worst mean ACR for the other two price trend conditions was achieved when all users used the A-ADAPTED algorithm. In line with these mean ACR values, Figure 21 shows us that the violins of the Random composition are very bottom-heavy compared to those of the other compositions. The other compositions also resulted in higher median ACR values. The results show that for all compositions, the best mean performance was achieved in the buy-to-rent downtrend condition and the worst mean performance was achieved in the uptrend condition. From Table 15, We note that the A-Adapted user reached its highest maximum ACR value in the Random composition. We cannot be sure that this means that the Random composition results in the highest maximum ACR value overall, given that we cannot be sure that we encounter the input that results in the global maximum. However, it is an interesting thought that other users acting randomly might create an environment in which reaching a higher maximum ACR is more likely.

Buy-to-Rent	**RANDOM**	**UNIFORM**	**A-ADAPTED**
No trend	2.50	3.21	3.39
Uptrend	3.81	5.50	4.50
Downtrend	2.12	2.87	3.21

Table 14: Mean ACR of the A-Adapted user for the different compositions and price trends.

Buy-to-Rent	**RANDOM**	**UNIFORM**	**A-ADAPTED**
No trend	7.48	5.65	7.48
Uptrend	17.92	14.62	11.93
Downtrend	9.67	10.75	10.48

Table 15: Maximum ACR of the A-Adapted user for the different compositions and price trends.

We can more easily analyze the effects of the composition on the mean ACR. Because the other users in the composition influence the environment for the A-Adapted user, it might be that the difference in mean ACR between the compositions is due to the composition instead of the performance of the A-Adapted user. We wanted to assert that the reason for the best mean performance in the Random composition was (at least partly) due to the A-Adapted user performing better in that composition. Therefore, we ran the same 10,000 simulations for the same price trends, but with only Random users. We calculated the mean ACR for one of the Random users over all simulations. The results of this experiment can be found in Table 16. We note that the mean ACR of the Random user was higher than that of the A-Adapted user in all buy-to-rent price trend conditions. These results suggest that the A-ADAPTED algorithm was at least partially responsible for the better performance of the A-Adapted user in the Random composition. The results from the experiments in this section indicate

that users trying to solve the VeChain Decision Problem can benefit from choosing the best algorithm to apply in a multi-agent setting.

Buy-to-Rent	**RANDOM**
No trend	3.63
Uptrend	4.14
Downtrend	3.89

Table 16: Mean ACR of the Random user when everyone uses the RANDOM algorithm, for different buy-to-rent price trends.

5 Discussion

In this Section, we discuss the the methods and findings of this study. We summarize the main contributions of the study in Section 5.1, we go over the limitations of the study in Section 5.2, and we discuss potential future work in Section 5.3.

5.1 Summary of Main Contributions

This study asked three main questions. Firstly, we wanted to know whether it was possible to create a mapping from the VeChain Decision Problem to the Ski Rental Problem. To investigate this question, we identified the similarities and differences between the two problems. We created a definition for the VeChain Decision Problem that mirrored the Ski Rental Problem while still considering the special features of the VeChain Decision Problem.
The second question that this study asked was how users solving the VeChain Decision Problem would perform when they used solutions to existing Ski Rental Problems. To answer this question, we created a model that was able to simulate the VeChain Decision Problem for a single or multiple users at the same time. Most importantly, the model considered the effects that user decisions had on the buy and rent prices by modelling these effects through limit order books.
We initially determined that the model was able to simulate the original Ski Rental Problem. While doing so, we investigated the performance of users implementing the existing solutions as well as the solutions introduced by us. We identified the differences between the theoretical competitive ratios of all users and the mean- and maximum achieved competitive ratios (ACR) that they achieved in our simulations of the original Ski Rental Problem.
Next, we studied the effect of user size on user performance. While doing so, we also investigated when results from simulations with limited length became representative of theoretical performance in unlimited length. To do so, we enabled the exchange of the model, which meant that buying and renting by users affected the corresponding prices. The results showed that users that were too small did not affect the buy and rent prices enough during the simulation. This resulted in the keep-renting option being optimal in all simulations, which meant that smaller user sizes would not not lead to a good measure of real-world performance. In order to get results that were representative of the real world, we needed to find a user size that was large enough so that the performance of the user approximated its performance for a setting with unlimited length. We found that with the used tick size of our limit order books and a simulation length of 10 years, we needed to model a user with a size of at least 0.60 times the amount of daily generated VTHO. As we deemed smaller users to be more representative of the real world as well, we decided to use this minimal size as the user size for all further single-user simulations of the VeChain Decision Problem.
Having specified a user size, we tested the performance of different algorithms in a single-user version of the VeChain Decision Problem. We investigated three users with simple offline algorithms that we introduced ourselves: the Random user, the Keep-Renting user, and the Instant-Buy user. We also tested the performance of the optimal deterministic and randomized solutions to the original Ski Rental Problem. Lastly, we adapted an algorithm that performed optimally in an environment with certain limited price fluctuations to our problem setting. We called the user that applied this algorithm the A-Adapted user, and tested it's performance in the VeChain Decision Problem. It was found that out of all users, the A-Adapted user achieved the best average performance in all buy-to-rent price trend conditions. We found that the average performance of the optimal deterministic and optimal randomized algorithms was very close to that of A-Adapted. However, the optimal randomized algorithm could encounter situations in which it performed much worse compared to the

other two online algorithms. This finding was most likely due to the randomization factor of the algorithm, and indicated that it is unwise for risk-averse users to apply randomized algorithms when solving the VeChain Decision Problem.
We also studied the performance of the A-Adapted user in a multi-agent setting. This multi-agent setting considered three different compositions: one in which all other users were Random users, one where there was a single Random, Deterministic, and Randomized user, and a composition in which all users used the A-Adapted algorithm. The results showed that the composition affected the performance of the A-Adapted user and that this difference could not (completely) be explained by the different composition. All in all, the results of this experiment indicated that users trying to solve the VeChain Decision Problem are affected by how other users decide to buy or rent, and that users could benefit from considering this in their own decision making process.
The third question that this study asked was whether applying the existing solutions to Ski Rental Problems would affect the network adoption behavior of the users applying these solutions. We looked at both the mean adoption ratio of network users, and the network adoption speed. Because buying indicates a long-term commitment, we defined the adoption ratio as the mean proportion of users that decided to buy during the simulations. We defined the adoption speed by the day that buying users bought on. Both the algorithm used by the users as well as the buy-to-rent price trend condition had a slight effect on the adoption ratio. The buy-to-rent price trend condition had a pretty strong effect on the adoption speed. Compared to the stable buy-to-rent price trend condition, an increasing buy-to-rent ratio resulted in users buying later and a decreasing ratio resulted in users buying sooner.
All in all, this study created a definition for the VeChain Decision Problem that maps to the Ski Rental Problem. We showed that solutions to the Ski Rental Problem can be used to solve the VeChain Decision Problem, and that these solutions allow users to perform much better than when they make random decisions (by buying on a random day) or when they stick to the same plan (by always renting or instantly buying). We also showed that the uniquely direct relationship that is present in the VeChain Decision Problem between user decisions and buy/rent prices allows for the possibility of studying a variation to the Ski Rental Problem in a multi-agent setting, while requiring minimal assumptions about how these agents affect each other in the real world.

5.2 Limitations

This study knows multiple limitations. Firstly, it should be noted that simulating the VeChain Decision Problem is a balancing act between user size, simulation length, liquidity, and the tick size of the limit order books. The simulation length needs to be short enough in order to make it computationally tractable to perform enough simulations. However, shorter simulation lengths can make it so that the optimal solution remains to keep renting, even if the user is active until the end of the simulation. This scenario occurs more quickly if the user size is small, as a smaller user affects the buy and rent prices less per day than a larger user. Moreover, the tick size of the limit order books that is used in the simulation also influences this. If the tick size is large, smaller users might not affect the buy and rent prices at all because they do not buy the entire amount of tokens present in that tick. In turn, the amount of tokens present in that tick is also dependent on the liquidity of the order books. While we think that we have been able to strike a balance in these settings, we did need to use a very large user size in order to do so. In Section 5.3, we discuss a couple of ways in which the model could be improved with regards to this balancing act.
Another limitation of this study is that the model does not consider the effect that users have on the shape of the limit order books. When a user buys into a LOB, they change the price but the next user that has to decide will only notice the difference in price. The LOB presented to the second user does

not take into account that the buying of the first user removed limit orders. The second user simply sees a random instance of the limit order book in equilibrium. In line with this, another limitation arises from the fact that the order book is represented by LOB instances coming from real-world data in which the LOB was more or less in equilibrium. Because the evolution of the shape of the LOB is not modelled, we need to assume that the mean shape remains the same over time. There is however no reason to assume that this mean shape remains the same over the long period of time that we model. It is for example very well possible that the mean shape starts changing when a lot of users start buying into it because the VeChain network is being used much more.
The study also made a couple of assumptions with regards to VeChain that make the model less representative of the real world. First of all, while we did consider that only 70% of the VTHO gets burned, we did not consider that the nodes receiving the other 30% might provide a consistent selling pressure on VTHO. Moreover, we did not consider the fact that users buying VET get voting rights that could allow them to influence the amount of usage that they can get out of their VET. Furthermore, in determining the performance of the users, we did not consider that users buying VET might have the opportunity to sell their VET after their project has come to an end. This does not necessarily have to be possible, because the tokens might not be worth anything anymore. However, on the other hand, the user might actually make a profit when they sell their VET after use.
With regards to the results, it should be noted that the maximum ACR can only be thought of as an indication of worst-case performance. Since its value is very dependent on the scenarios that the user encounters, users that achieved a low maximum ACR in our simulations might still have a high global maximum ACR hidden somewhere in the input space.
Lastly, we acknowledge that the definition of the VeChain Decision Problem created in this study makes it so that users are presented with a very binary decision: each day, they can either buy or rent. Of course, this is not necessarily true in the real world. Users could decide to only buy a part of the amount of VET that they would need. Moreover, users could decide to buy more VTHO than they require on a single day, for example when they deem the VTHO price to be cheap at that point in time. Finally, in the real world, users also have the option of switching back and forward between renting and buying.

5.3 Future Work

The findings in this study are promising and could inspire a lot of future work. First of all, future research could investigate the performance of other solutions to existing Ski Rental Problems in the VeChain Decision Problem. Other existing algorithms could be implemented directly or adapted for use in the VeChain Decision Problem. In doing so, it will probably be best to focus on deterministic online algorithms, given the disappointing worst-case performance that the randomized algorithm achieved in our simulations.
Secondly, future studies applying the model presented here could look into generating limit order books that cover the same price fluctuation range, but with more precise price points (ticks). This increased precision would mean that the minimal amount that the price can move decreases. In turn, this means that the studies could test the performance of smaller users in the same simulation length, without encountering the issue that the optimal solution over the entire simulation is to keep renting.
Moreover, the ability to model smaller users would make it more feasible to implement the market orders from the users into the model from [25]. Doing so would mean that the decisions of the users not only affect the price that other users encounter, but also the shape of the limit order book that they need to buy or sell in to. Implementing the market order into the model from [25] would thereby result in a more accurate model of the effects between a user's decision and the decisions of other

users.
Future research could also investigate the possibility of implementing price predictions in the online algorithms. Solutions with decent price predictions could use this data in order to either postpone long-term commitments or to make the commitment sooner. For example, one could consider adding a parameter like the γ parameter in A-ADAPTED that not only allows for quicker buying (as is the case with A-ADAPTED) but also allows for later buying. This parameter could be set based on the future buy-to-rent ratios that the algorithm predicts.
Finally, having established that the VeChain Decision Problem provides a reason to study Ski Rental-like problems in a multi-agent setting, future research is encouraged to investigate the possibilities that this offers. More extensive research into the performance of the algorithms in a multi-agent setting would be a start. Furthermore, the direct relationships between the decisions of users allow for the implementation of game theoretic reasoning in the algorithms. The transparency of blockchain networks means that it is fairly easy to see the decisions of other users, which means that it is reasonable to assume this possibility. We know from our initial multi-agent simulations that the strategy applied by other users is relevant to a user's performance. Because of this, it makes sense to think that users could incorporate the decisions made by other users into their algorithms as an additional data point.

Bibliography

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[2] M. Crosby, P. Pattanayak, S. Verma, and V. Kalyanaraman, "Blockchain technology: Beyond bitcoin," *Applied Innovation*, vol. 2, no. 6-10, p. 71, 2016.

[3] N. Szabo, "No title," *Smart contracts*, 1994.

[4] L. W. Cong and Z. He, "Blockchain disruption and smart contracts," *The Review of Financial Studies*, vol. 32, no. 5, pp. 1754–1797, 2019. doi: 10.1093/rfs/hhz007.

[5] A. Kamilaris, A. Fonts, and F. X. Prenafeta-Bold, "The rise of blockchain technology in agriculture and food supply chains," *Trends in Food Science Technology*, vol. 91, pp. 640–652, 2019. doi: 10.1016/j.tifs.2019.07.034.

[6] B. L. Handoko, B. A. Gunawan, and M. F. P. Djati, "Importance of blockchain within the big 4 cpa firms: Cryptocurrency's existence," in *Proceedings of the 6th International Conference on E-Commerce, E-Business and E-Government*, pp. 163–169, 2022. doi: 10.1145/3537693.3537718.

[7] VeChain Foundation, "VeChain whitepaper." https://www.vechain.org/whitepaper/. Accessed: 2021-12-02.

[8] VeChain Foundation, "Walmart China Takes on Food Safety with VeChainThor Blockchain Technology." https://medium.com/vechain-foundation/walmart-china-takes-on-food-safety-with-vechainthor-blockchain-technology-b1443e0e079c/, 2019. Accessed: 2022-04-13.

[9] PwC, "PwC China release Air Trace, a traceability solution powered by blockchain, at 2021 CIFTIS, Beijing." https://www.pwchk.com/en/press-room/press-releases/pr-130921.html/. Accessed: 2022-04-13.

[10] VeChain Foundation, "San Marino Approves National Green Pass, Allowing Citizens And Residents To Move Freely." https://medium.com/vechain-foundation/san-marino-approves-national-green-pass-allowing-citizens-and-residents-to-move-freely-1e874f4ccd1b/, 2021. Accessed: 2022-04-13.

[11] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, "Competitive snoopy caching," *Algorithmica*, vol. 3, no. 1, pp. 79–119, 1988. doi: 10.1007/BF01762111.

[12] A. Khanafer, M. Kodialam, and K. P. Puttaswamy, "The constrained ski-rental problem and its application to online cloud cost optimization," in *2013 Proceedings IEEE INFOCOM*, pp. 1492–1500, IEEE, 2013. doi: 10.1109/INFCOM.2013.6566944.

[13] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. cambridge university press, 2005.

[14] X. Yang, W. Zhang, Y. Zhang, and W. Xu, "Optimal randomized algorithm for a generalized ski-rental with interest rate," *Information Processing Letters*, vol. 112, no. 13, pp. 548–551, 2012. doi: 10.1016/j.ipl.2012.04.006.

[15] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson, "On the power of randomization in on-line algorithms," *Algorithmica*, vol. 11, no. 1, pp. 2–14, 1994. doi: 10.1007/BF01294260.

[16] Z. Lotker, B. Patt-Shamir, and D. Rawitz, "Ski rental with two general options," *Information processing letters*, vol. 108, no. 6, pp. 365–368, 2008. doi: 10.1016/j.ipl.2008.07.009.

[17] M. Bienkowski, "Ski rental problem with dynamic pricing," 2008.

[18] L. Ai, X. Wu, L. Huang, L. Huang, P. Tang, and J. Li, "The multi-shop ski rental problem," in *The 2014 ACM international conference on Measurement and modeling of computer systems*, pp. 463–475, 2014. doi: 10.1145/2591971.2591984.

[19] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki, "Competitive randomized algorithms for non-uniform problems," in *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pp. 301–309, 1990.

[20] X. Feng, Y. Xu, G. Ni, and Y. Dai, "Online leasing problem with price fluctuations under the consumer price index," *Journal of Combinatorial Optimization*, vol. 36, no. 2, pp. 493–507, 2018. doi: 10.1007/s10878-018-0305-7.

[21] G. A. Pierro and H. Rocha, "The influence factors on ethereum transaction fees," in *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pp. 24–31, IEEE, 2019. doi: 10.1109/WETSEB.2019.00010.

[22] Binance, "Buy/Sell Bitcoin, Ether and Altcoins — Cryptocurrency Exchange — Binance." https://www.binance.com/. Accessed: 2022-04-17.

[23] P. Handa, R. A. Schwartz, and A. Tiwari, "The ecology of an order-driven market," *Journal of Portfolio Management*, vol. 24, no. 2, p. 47, 1998. doi: 10.3905/jpm.24.2.47.

[24] Tether, "Tether." https://tether.to/. Accessed: 2022-05-23.

[25] R. Cont, S. Stoikov, and R. Talreja, "A stochastic model for order book dynamics," *Operations research*, vol. 58, no. 3, pp. 549–563, 2010. doi: 10.1287/opre.1090.0780.

[26] S. Watts, "Simulating limit order book models," 2016.

[27] M. D. Gould, M. A. Porter, S. Williams, M. McDonald, D. J. Fenn, and S. D. Howison, "Limit order books," *Quantitative Finance*, vol. 13, no. 11, pp. 1709–1742, 2013. doi: 10.1080/14697688.2013.803148.

[28] J.-P. Bouchaud, M. Mézard, and M. Potters, "Statistical properties of stock order books: empirical results and models," *Quantitative finance*, vol. 2, no. 4, p. 251, 2002. doi: 10.1088/1469-7688/2/4/301.

[29] Amazon Web Services, "Cloud Computing Services - Amazon Web Services (AWS)." https://aws.amazon.com/. Accessed: 2022-06-04.

[30] J. Kazil, D. Masad, and A. Crooks, "Utilizing python for agent-based modeling: The mesa framework," in *Social, Cultural, and Behavioral Modeling* (R. Thomson, H. Bisgin, C. Dancy, A. Hyder, and M. Hussain, eds.), (Cham), pp. 308–317, Springer International Publishing, 2020. doi: 10.1007/978-3-030-61255-9_30.

[31] B. Winter, "Modelling the VeChain Decision Problem." https://github.com/BorisWinter/MasterThesis.

[32] Binance, "Web Socket Streams for binance (2022-06-17)." https://github.com/binance/binance-spot-api-docs/blob/master/web-socket-streams.md. Accessed: 2022-08-17.

Appendices

A Collecting and cleaning the LOB data set

In order to obtain the Level II order book data used to generate the limit order books, we accessed the VET / USDT and VTHO / USDT order books of the Binance cryptocurrency exchange through their API. Specifically, the method described in [32] was used. We repeat the main steps of the method described in [32] here, altered for the VET / USDT order book.

1. Open a stream to `wss://stream.binance.com:9443/stream?streams=vetusdt@trade`.
2. Buffer the events you receive from the stream.
3. Get a depth snapshot from `https://api.binance.com/api/v3/depth?symbol=VETUSDT&limit=1000` .
4. Drop any event where u is $\leq$ *lastUpdateId* in the snapshot.
5. The first processed event should have $U <= \textit{lastUpdateId} + 1$ AND $u >= \textit{lastUpdateId} + 1$.
6. While listening to the stream, each new event's U should be equal to the previous event's $u + 1$.
7. The data in each event is the absolute quantity for a price level.
8. If the quantity is 0, remove the price level.
9. Receiving an event that removes a price level that is not in your local order book can happen and is normal.

The API data was collected once per second on 17-06-2022 from 11:30 until approximately 20:00, which is when the connection was automatically closed by Binance. The resulting .txt files were converted to .csv files and data that was not required for further analysis was discarded. After this process, we had three files for each order pair:

- **Difference:** A file containing all changes in the depth profile during that time.
- **Snapshot:** A snapshot of the order book from just after we started listening to the depth profile changes.
- **Trades:** A file that contained all trades that occurred after we started listening to the depth profile changes.

We combined the difference and snapshot data in order to create a data set that contained, for each recorded second, a set of the ten best bids and asks along with the total quantities offered at those prices. Each bid and ask was presented as a (price, quantity) pair, in which we rounded the prices to a set precision. For each of the two order pairs, we identified the best precision (step size) to round the prices to when combining the files. For VET, this was determined to be 4 decimals, and for VTHO this was determined to be 5 decimals.

B Maximum ACR values from the user exchange experiment

User size	RANDOM	K-RENTING	I-BUY	DET	RAND	A-ADAPTED
0.05	15856.23	1.00	174747.79	1.00	1076.03	1.00
0.1	6630.43	1.00	138023.28	1.00	270.26	1.00
0.15	5442.18	1.00	119956.51	1.00	791.22	1.00
0.2	24417.69	1.00	164288.43	1.00	178.63	1.00
0.25	3555.23	1.00	89454.45	1.00	145.36	1.00
0.3	8634.87	1.43	48420.05	2.00	128.77	2.00
0.35	4450.71	2.77	97401.7	2.00	133.8	2.00
0.4	5901.73	5.00	63459.32	2.00	115.89	2.01
0.45	2401.80	8.61	43166.54	2.00	214.5	2.01
0.5	2117.94	14.91	34413.07	2.00	1127.63	2.00
0.55	1446.31	24.28	49122.81	2.00	122.63	2.01
0.6	1794.55	39.98	27571.58	2.00	264.49	2.00
0.65	1905.53	62.61	34097.97	2.00	158.07	2.00
0.7	1574.00	100.71	32414.43	2.01	81.35	2.01
0.75	1139.71	153.88	26252.37	2.01	185.89	2.00
0.8	1060.04	239.79	22184.41	2.01	300.99	2.01
0.85	4108.17	344.67	24243.41	2.01	137.54	2.01
0.9	4660.54	535.23	21589.44	2.01	267.29	2.01
0.95	1643.28	800.28	21934.5	2.01	84.1	2.01
1.00	1113.98	1180.47	17425.48	2.01	294.96	2.00

Table 17: Maximum ACR of the users for different user sizes. User size is given as a fraction of the daily amount of generated VTHO.