# Optimal Trade Execution in Automated Market Makers

Project Report

Optimization Methods

Jaupi, Megi
jaupimeg@mit.edu

Theron, Paul
paulth@mit.edu

January 20, 2023

## 1 Introduction: Automated Market Makers

Traditional financial markets rely on market makers to provide enough liquidity in the markets. Financial institutions or individual traders place multiple bid ask orders in the market to ensure that buying and selling orders from different counterparties are met. The difference between the bid and ask price, namely the spread, represents the profit of this agent for the provided service. In the realm of decentralized finance (DeFi), cryptocurrency exchanges aim to reduce the need for extra agents in the market.

Automated market makers (AMMs) are systems that automatically determine the price of the assets in consideration and operate on the basis of smart contracts in order to execute the buy and sell orders. In this model, counterparties trade directly with the AMM instead of a centralised agent. Decentralized Exchanges (DEXes) and AMMs have gained a considerable amount of popularity. As of the 8th of December 2022, 100 tracked AMMs have a reported 24h trading volume of USD 1.41 Billion. Uniswap is reported to be the largest with a trading volume of around 500 USD Million.

Continuing the analogy, in centralised markets, market makers operate with an order book - a list of bid and ask orders. In contrast, AMMs rely on cryptocurrency reserves called liquidity pools, which in general comprise of one or multiple pairs of cryptocurrencies. There are several price generation mechanisms but the most common ones are Constant Product Market Makers. In these AMMs, pricing is based on the supply and demand in the liquidity pool such that the ratio of assets in any liquidity pool remains balanced, i.e. $xy = k$, where $k$ is a constant defined by the liquidity provided to a pool.

# 2 Problem Formulation

In a traditional order-book based market, large orders are often executed at suboptimal prices. A large order can run the book and execute at limit-sell orders of higher prices. Similarly, in AMMs trades affect market prices as it continuously shifts along the constant product curve. There are several liquidity pools in the decentralized exchanges per cryptocurrency pair. Big shifts in prices can occur if we place a large order in one AMM, leading to increased costs of trades and perhaps an arbitrage opportunity due to the illiquid nature of crypto exchanges.

The goal of this project is to make use of the structure of AMMs to derive an optimization based strategy of splitting large orders. Our objectives will be multiple. Intuitively, we may be interested in solely minimizing the total costs of the trade (purchasing the same amount of tokens with less costs). On the other hand, we may be interested in minimizing the price impact of the trade (minimizing disbalances between liquidity pools to not allow for arbitrage after the trade).

## 2.1 Single period convex optimization

### 2.1.1 Single Period with no trading Fees

We first model the problem of splitting an order across n liquidity pools, without trading fees, over a single period. In this context, we would have for example 3 pools coming from the Uniswap DEX, 3 coming from the DexGuru DEX of ETH and USDT token. These pools are defined by different liquidity (the value of $k$), as well as the same price (the ratio USDT/ETH). We want to buy a large amount of ETH tokens, and therefore split an order on all these pools. We will assume that the orders are instantaneous on all the pools.

Let us introduce some notation:

- $n$ is the number of pools
- $x_j, \forall j \in [1, n]$ is the amount of tokens $x$ in pool $j$ that we want to buy (ETH in the example)
- $y_j, \forall j \in [1, n]$ is the amount of tokens $y$ in pool $j$ (USDT in the example)
- $k_j, \forall j \in [1, n]$ is the liquidity of pool $j$, constant and equal to $x_j y_j$
- $\Delta x$ is the total amount of token x I want to get with my trade.

In the following, some relationships from $xy = k$, useful for our problem definition. Let's take a single pool $j$ for simplicity.

Simply formulated, if one wants to get $\Delta x$ of token x from the pool, which amount of token y, $\Delta y$ should one put into it:

$$(x - \Delta x)(y + \Delta y) = k = xy \tag{1}$$

Thereafter:

$$\Delta y = \frac{y\Delta x}{x - \Delta x} \tag{2}$$

$$\tag{3}$$

We define the order execution price of our trade, which is equal to:

$$p_{exec} = \frac{\Delta y}{\Delta x} = \frac{y}{x - \Delta x} \tag{4}$$

The price post trade in the pool will be equal to:

$$p_{post} = \frac{y + \Delta y}{x - \Delta x} \tag{5}$$

NB: If we split the order over all the pool, the overall execution price would be the weighted average of our execution price over the different pools. Subsequently, we derive multiple optimization problems. Due to the restriction on the length of this document, all the calculations are made in the appendix 5.

**Minimizing the market impact**

$$\min \sum_{j=1}^{n} p_j^{post} - p_j^{pre} = \min \sum_{j=1}^{n} \frac{y_j \Delta x_j (2x_j - \Delta x_j)}{x_j (x_j - \Delta x_j)^2} \tag{6}$$

$$\text{s.t} \sum_{j=1}^{n} \Delta x_j = \Delta x \tag{7}$$

$$\Delta x_j \geq 0 \tag{8}$$

**Minimizing the total execution costs**

$$\min \sum_{j=1}^{n} \frac{y_j}{x_j - \Delta x_j} \frac{\Delta x_j}{\Delta x} \tag{9}$$

$$\text{s.t} \sum_{j=1}^{n} \Delta x_j = \Delta x \tag{10}$$

$$\Delta x_j \geq 0 \tag{11}$$

### 2.1.2 Single Period with trading Fees

In the previous formulation, we did not model the trading fees when executing a trade over a specific pool. This approach is not realistic, as the trading fees are a significant part of the total cost of a trade for large trades. Moreover, in practice the pools on DEX are created by liquidity providers, who are incentivized to charge fees to the traders (in return for providing liquidity). In this section, we model the trading fees as a percentage of the trade size.

Let $f$ be the trading fee for a pool. Let's reuse the notations from the previous part. I want to buy $\Delta x$ ETH for $\Delta y$ USDT. If I send $\Delta y$ USDT to the pool, $f$ of it will be charged as a fee, so I will make the trade with $\Delta y(1-f)$. The pool will send me $\Delta x$ ETH, and will receive $\Delta y(1-f)$ USDT. Then, the invariant $k$ will be updated (fees will be added post trade).

Therefore we have by the laws of automated market makers:

$$(x-\Delta x)(y+\Delta y(1-f)) = k_{before} = xy \tag{12}$$

$$\tag{13}$$

$$\Delta y = \frac{y\Delta x}{(1-f)(x-\Delta x)} \tag{14}$$

$$\tag{15}$$

After that, the fee is added the the USDT pool, which makes the invariant change.

$$x_{post} = x - \Delta x \tag{16}$$

$$y_{post} = y + \Delta y \tag{17}$$

$$k_{post} = x_{post} y_{post} \tag{18}$$

$$p_{post} = \frac{y_{post}}{x_{post}} \tag{19}$$

So the quantity is bigger and the liquidity provider will receive some money afterwards when they decide to take off their liquidity. We derive new optimization problems as:

**Minimizing the market impact**:

$$\min \sum_{j=1}^{n} y_j [\frac{\Delta x_j (2x_j - \Delta x_j)}{x_j (x_j - \Delta x_j)^2} + \frac{f\Delta x_j}{(1-f)(x_j - \Delta x_j)^2}] \tag{20}$$

$$\text{s.t} \sum_{j=1}^{n} \Delta x_j = \Delta x \tag{21}$$

$$\Delta x_j \geq 0 \tag{22}$$

**Minimizing total execution costs**

$$\min \sum_{j=1}^{n} \frac{y_j}{(1-f)(x_j - \Delta x_j)} \frac{\Delta x_j}{\Delta x} \tag{23}$$

$$\text{s.t} \sum_{j=1}^{n} \Delta x_j = \Delta x \tag{24}$$

$$\Delta x_j \geq 0 \tag{25}$$

Note that we do not restrict $\Delta x_j$ to be integer. While usually in the financial markets one would buy integer amount of assets as it does not make sense to buy let's say 550.2 stocks, one other flexibility of cryptocurrency markets is the ability to purchase decimal units of tokens. Hence we can relax the integrality constraint without loss of practical relevance.

## 2.2 A stochastic dynamic programming approach

Splitting a trade across multiple AMMs during one time period is indeed interesting as an academic exercise but stands far from reality. As an extension to the problem we want to model the problem of splitting across multiple AMMs in multiple time periods. The natural mathematical translation of this problem would indeed be a stochastic dynamic optimization problem. Hereafter we make the important assumption that in time there is no new liquidity added in the liquidity pool, i.e. $k$ remains constant.

The fascinating part about decentralised exchanges is that we do not need to model or make any assumptions regarding the market impact of our trade. That is given to us by the dynamics of the market structure. What we need to model, however, is how the market price dynamics are. In the following we derive two formulations for two different price dynamics.

**Random walk approach**

For ease of notation let $s_t = \Delta x$. We use the following modelisation for the price over time $t$, $\epsilon$ is a normally distributed random variable. In this exercise we do not differentiate between the pre and post trade price.

$$p_t = f(p_{t-1}, s_t) + \epsilon_t \tag{26}$$

Where $f$ is the function that defines the price variation in the pool with our trade.

$f$ is defined as:

$$f(p_{t-1}, s_t) = \frac{y_{i,t} + \Delta y_{i,t}}{x_{i,t} - \Delta x_{i,t}} \tag{27}$$

**Binomial approach**

We will differentiate price post and pre trade, they are moving with the following evolution, where $r$ is a random variable taking values $u$ with probability $p$ and $d$ with probability $1 - p$.

$$\begin{aligned} p_t^{pre} &= r p_{t-1}^{post} \\ p_t^{post} &= f(p_t^{pre}, s_t) \end{aligned}$$

As this part is outside the scope of our project we provide more details on modeling, approaches to solve optimization problems by Bellman equations, as well as results on this topic in the appendix 5.2.

# 3 Experiments and results

We implement the optimization problem formulations provided at the previous section in several scenarios. Since we are rather interested in being close to reality, we focus on one of the most

| Experiment | Parameters | Optimizing total costs | Optimizing market impact | No-Split | Average Split |
|---|---|---|---|---|---|
| Experiment 1 Same $k$ same $p_i^{pre}$ | $p^{pre}$ | 1210 | 1210 | 1210 | 1210 |
| | $p^{post}$ | 1280 | 1280 | 2722 | 1280 |
| | $p^{exec}$ | 1244 | 1244 | 1815 | 1244 |
| | Total costs | 1'244'571 | 1'244'571 | 1'815'000 | 1'244'571 |
| Experiment 2 Diff $k$ same $p_i^{pre}$ | $p^{pre}$ | 1210 | 1210 | 1210 | 1210 |
| | $p^{post}$ | 1289 | 1266 | 1405 | 1305 |
| | $p^{exec}$ | 1249 | 1297 | 1374 | 1256 |
| | Total costs | 3'748'001 | 3'892'184 | 4'122'000 | 3'770'367 |
| Experiment 3 Diff $k$ diff $p_i^{pre}$ | $p^{pre}$ | 1205 | 1205 | 1205 | 1205 |
| | $p^{post}$ | 1250 | 1246 | 1387 | 1272 |
| | $p^{exec}$ | 1214 | 1240 | 1945 | 1237 |
| | Total costs | 2'429'174 | 2'492'764 | 2'774'940 | 2'474'819 |

Table 1: Experiment results for the scenario of 12 AMMs without transaction fees

widely traded cryptocurrencies, Ethereum (ETH) against the USD token (USDT). Overall in this section we optimize across 12 liquidity pools. To construct the liquidity pools we make use of market price and liquidity of Uniswap ETH:USDT pools. For the experiments that involve trading fees, we use the range of fees extracted from the same Uniswap pools.

We first start with the scenario where all 12 liquidity pools have the same constant product $k$ and are currently at the same market price. The second experiment involves liquidity pools of different constant products $k_i$ and currently at the same market price. For the case of no transaction costs, we also explore the scenario where liquidity pools are imbalanced, i.e they stand at different market prices. In all experiments we compare our optimization approach against trading all tokens in one AMM and against splitting the trade into the same portions across all AMMs.

## 3.1 Single Period with no transaction fees

Table 1 demonstrates the results for the scenario of no transaction fees. As we would have expected, in the absence of transaction costs, when the AMMs start at the same market price and when they operate with the same constant product formualtion, then the optimal strategy would be to split the order evenly across all AMMs. In this case, we invest 83.3 tokens of ETH in each of the liquidity pools for a total order of 1000 tokens.

In Experiment 2, we trade 3000 tokens of ETH in pools that start at the same market price but have different liquidity. Observe that we gain around USD 1M when we split the trade as opposed to no when we optimize for total costs, we save on the order of USD 230k with any splitting strategy. Optimizing for total costs provides us with savings of USD 22k as opposed to the average split and we end up with a post trade market price that is lower than the average split. On the other hand, optimizing for market impact, gives us higher total costs of trade but we end up with a post trade market price of much lower than the average split and the previous optimization.

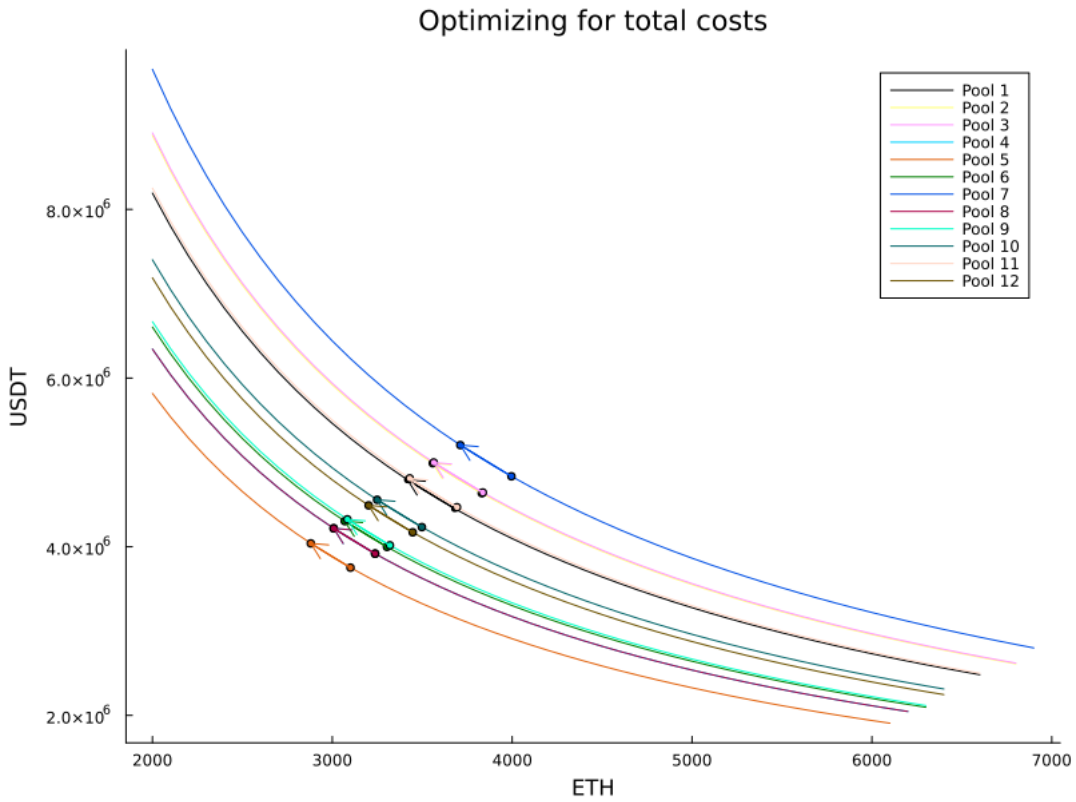



Figure 1: Optimizing for total costs the execution of 3000 ETH tokens in 12 AMMs starting at the same initial price levels and having different liquidities.

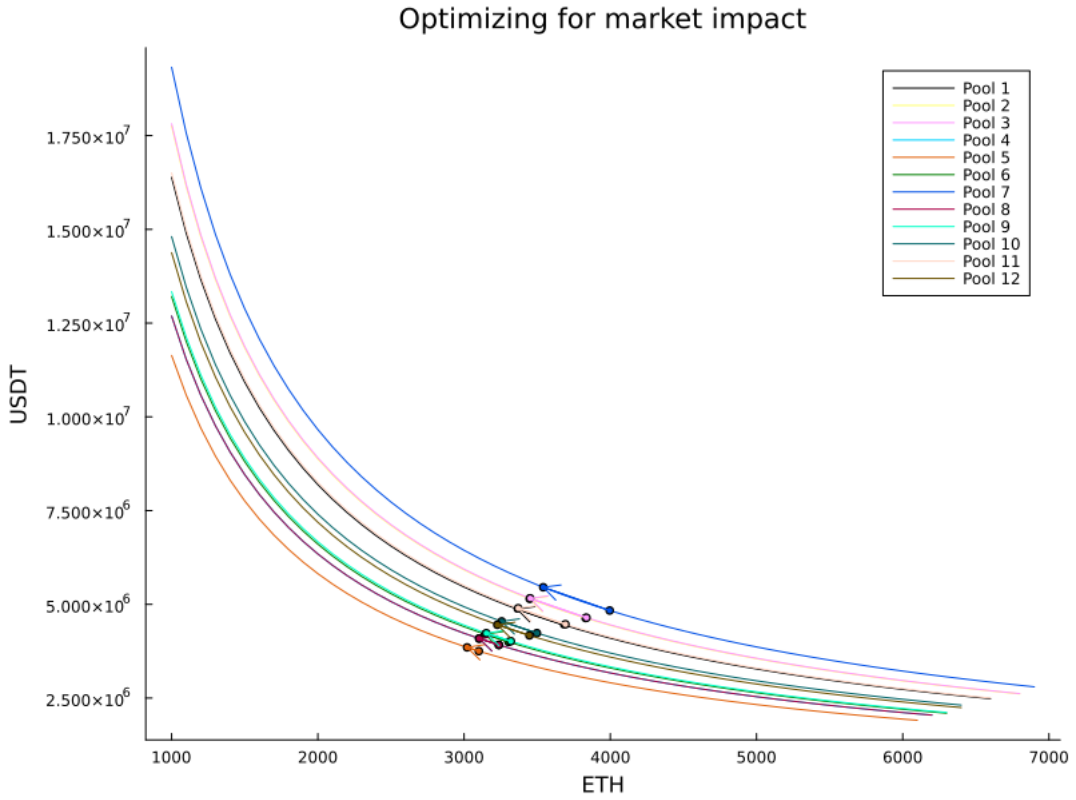



Figure 2: Optimizing for market impact the execution of 3000 ETH tokens in 12 AMMs starting at the same initial price levels and having different liquidities.

| Experiment | Parameters | Optimizing Total Costs | Optimizing market impact | No-Split | Average Split |
|---|---|---|---|---|---|
| Experiment 1 | $p^{pre}$ | 1210 | 1210 | 1210 | 1210 |
| Same $k$ same $p_i^{pre}$ | $p^{\bar{p}ost}$ | 1282 | 1280 | 2722.5 | 1280 |
| | $p^{exec}$ | 1304 | 1317 | 1833 | 1396 |
| | Total costs | 1'304'104 | 1'317'190 | 1'833'333 | 1'396'323 |
| Experiment 2 | $p^{pre}$ | 1210 | 1210 | 1210 | 1210 |
| Diff $k$ same $p_i^{pre}$ | $p^{\bar{p}ost}$ | 1294 | 1294 | 1268 | 1305 |
| | $p^{exec}$ | 1327 | 1431 | 1614 | 1420 |
| | Total costs | 3'983'901 | 4'294'688 | 4'843'892 | 4'260'538 |

Table 2: Experiment results for the scenario of 12 AMMs with transaction fees

There are also some interesting results in Experiment 3, when AMMs start on different price levels, with an average price of 1205 USD, and we yet again trade 3000 ETH tokes. Indeed, optimizing for total costs, will exploit the arbitrage opportunity and provide a very low execution price. It is interesting to see, however that this strategy will bring the liquidity pools in the same post-trade price of 1250 USD, whereas the other strategies have a similar post-trade average prices, but still maintain disbalances in the individual liquidity pools.

### 3.2 Single Period with transaction fees

Table 2 demonstrates the results for the scenario with transaction fees. We run the same Experiment 1 as in the previous section, with the formulation that includes liquidity pools. Again 12 pools, 1000 ETH tokens traded. As compared to the case without transaction fees, Optimizing either for total costs or market impact does not result in an evenly spread split among AMMs. This is intuitive, since it is more beneficial to evenly spread part of the trades in the AMMs with the lowest transaction fees. Then when the market moves enough to outweigh the benefit of splitting, it will continue to split the remaining part of the order in the next AMMs with the next lowest transaction costs.

Similarly to the previous section, we run the second experiment on trading 3000 ETH tokens, for liquidity pools that start at the same price level with different liquidities. We observe that the edge of optimization is even higher in the presence of transaction costs. Optimizing for total costs provides USD 280k savings over average split strategy or 6% and even results in a lower average post trade price by a factor of 1.7. The improvements are even higher when compared to the no-split strategy, namely a 17% improvement in the total costs and reduction of market impact by a factor of 3.5.

## 4 Conclusions and Further work

In this project we sought out to solve the problem of splitting a large order in Automated Market Makers in order to minimize the costs of the order or the market impact. First of all, the inner structure of the AMMs allowed us to formulate two realistic optimization problems that are

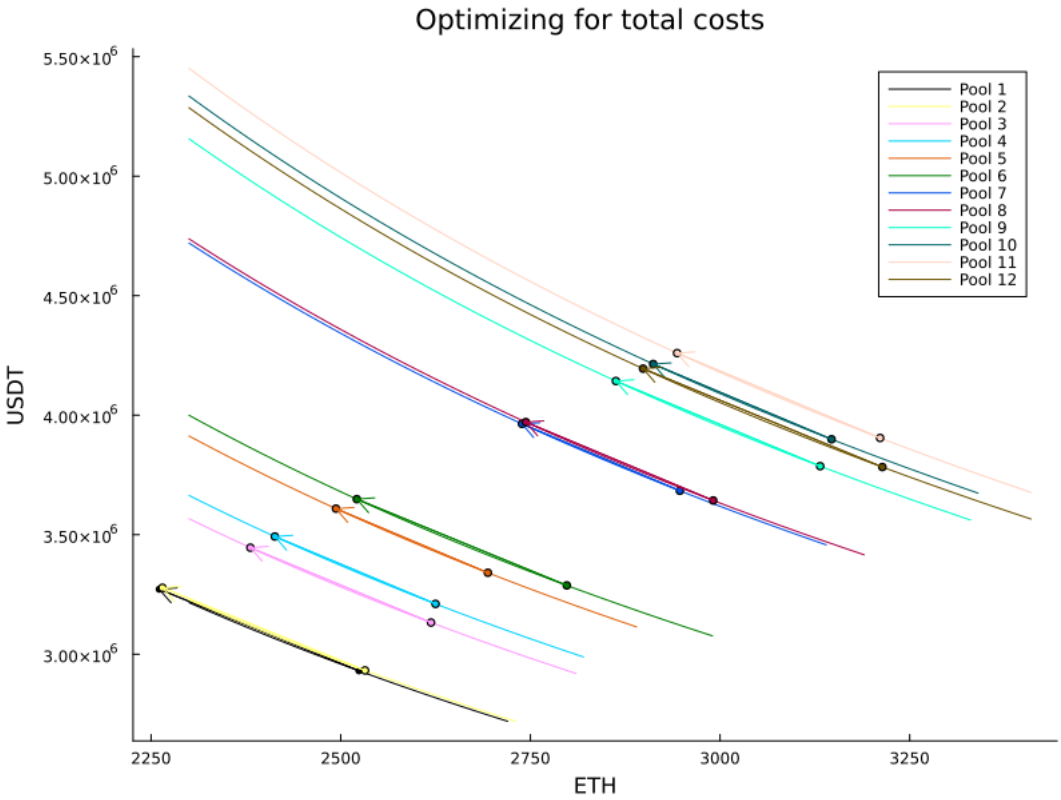



Figure 3: Optimizing for total costs the execution of 3000 ETH tokens in 12 AMMs starting at different initial price levels and having different liquidities. Optimizing for total costs result in the same post trade price across all liquidity pools.

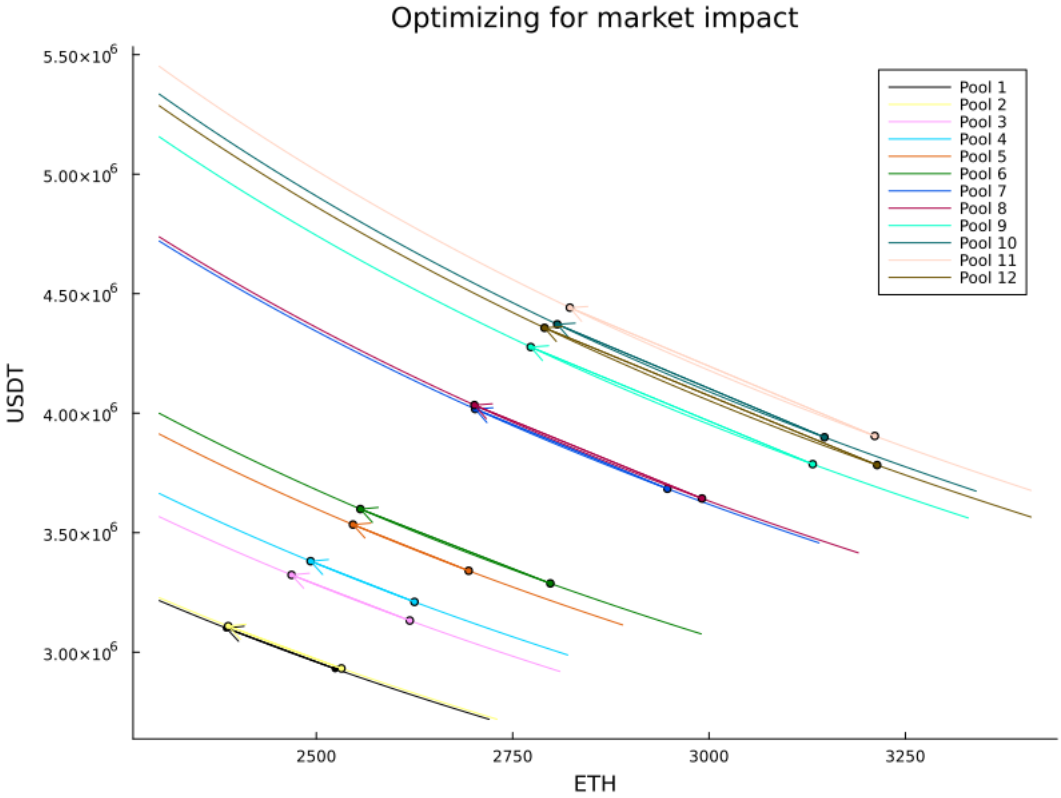



Figure 4: Optimizing for market impact the execution of 3000 ETH tokens in 12 AMMs starting at different initial price levels price levels and having different liquidities.

tractable computationally due to the convexity of the objective function. We formulated and solved the problems in the absence and presence of transaction costs. We concluded that the optimization approach has a significant edge over not splitting or uniformly splitting an order across AMMs. Further, the presence of transaction costs increases the necessity of optimized strategies.

The experiments conducted in this paper involved single-step optimization methods. A natural extension of the problem is multi-step stochastic optimization. In the appendix, we exploit the nice structure of AMMs to derive (under certain simplifications) a closed form solution for order splitting in case of random walk price dynamics. In addition, we explore binomial price dynamic formulations and provide solutions for the two-period binomial model.

As per the extensions to this project, we are certainly full of ideas. Expanding and formalizing the two approaches in the Appendix would be the first step. Also, the stochastic optimization method is not the only way to model multi-step order splitting. One other approach we could follow is that of a Robust Optimization approach. Assume we have a good predictor for future step crypto prices with a Machine Learning model. We could introduce an uncertainty set for these point predictions and perform multi-step robust optimization across the future prices.

# 5 Appendix

The appendix contains detailed calculus for the reader's convenience.

## 5.1 Single period order split

### 5.1.1 Without trading fees

We are reusing the notation of 2.1.1

Reformulation of minimization of market impact on a specific pool $j$:

$$\min \sum_{j=1}^{n} p_j^{post} - p_j^{pre} \tag{28}$$

$$\text{s.t} \sum_{j=1}^{n} \Delta x_j = \Delta x \tag{29}$$

$$\Delta x_j \geq 0 \tag{30}$$

### 5.1.2 With trading fees

Reformulation of minimization of market impact on a specific pool $j$:

$$p_j^{post} - p_j^{pre} = \frac{y_j + \Delta y_j}{x_j - \Delta x_j} - \frac{y_i}{x_j} \tag{31}$$

$$= y_j[\frac{(1-f)x_j + f\Delta x_j}{(1-f)(x_j - \Delta x_j)^2} - \frac{1}{x_j}] \tag{32}$$

$$= y_j[\frac{\Delta x_j(2x_j - \Delta x_j)}{x_j(x_j - \Delta x_j)^2} + \frac{f\Delta x_j}{(1-f)(x_j - \Delta x_j)^2}] \tag{33}$$

## 5.2 Stochastic modeling order split

### 5.2.1 Random walk price dynamics

Without loss of generality, we can make all the analysis over one pool and sum afterwards. So we define the variables:

- $w_t$: amount of tokens I still want to get at time t
- $s_t$: amount of tokens I will get at time t
- $p_t$: price of token x at time t

- $\epsilon_t$: random variable of mean 0 and variance 0.05
- $k_i$ is the liquidity of pool $i$ and is constant over time

T is the number of periods we want to optimize on. Our objective is to **minimize the expected market price**. *Nb*: We make a simplifying assumption here, because here we say that we get the token at price $p_t$ however, we have a closed form for execution price as described in the first part. We do not take it into account for this modelisation, for simplicity

Our variable are binded by the following constraints:

$$w_t = w_{t-1} - s_t \tag{34}$$
$$w_{T+1} = 0 \tag{35}$$
$$w_0 = S \tag{36}$$
$$p_t = f(p_{t-1}, s_t) + \epsilon_t \tag{37}$$

Where $f$ is the function that defines the price variation in the pool with our trade, it is closed form.

$f$ is defined as:

$$f(p_{t-1}, s_t) = \frac{y_{i,t} + \Delta y_{i,t}}{x_{i,t} - \Delta x_{i,t}} \tag{38}$$

where

$$\Delta x_{i,t} = s_t \tag{39}$$
$$\Delta y_{i,t} = \frac{y_{i,t}\Delta x_{i,t}}{x_{i,t} - s_t} \tag{40}$$
$$\tag{41}$$

and

$$x_{i,t} = \sqrt{\frac{k_i}{p_{t-1}}} \tag{42}$$
$$y_{i,t} = \sqrt{k_i p_{t-1}} \tag{43}$$
$$\tag{44}$$

So we have the function $f$:

$$
\begin{aligned}
xf(p_{t-1}, s_t) &= \frac{y_{i,t} + \frac{y_{i,t}s_t}{x_{i,t}-s_t}}{x_{i,t} - s_t} && (45)\\
&= \frac{y_{i,t}(x_{i,t} - s_t) + (y_{i,t}s_t)}{(x_{i,t} - s_t)^2} && (46)\\
&= \frac{k_i}{(x_{i,t} - s_t)^2} && (47)\\
&= \frac{k_i}{(\sqrt{\frac{k_i}{p_{t-1}}} - s_t)^2} && (48)\\
&= \frac{k_i p_{t-1}}{(\sqrt{k_i} - s_t\sqrt{p_{t-1}})^2} && (49)
\end{aligned}
$$

Our objective is:

$$
\begin{aligned}
\min_{s_t} \mathbb{E}_0 \left[ \sum_{t=0}^{T} p_t s_t \right] && (50)\\
\text{s.t} \sum_{t=0}^{T} s_t = \bar{S} && (51)\\
s_t \geq 0 && (52)
\end{aligned}
$$

By linearity of expectations:

$$
\min_{s_t} \sum_{t=0}^{T} \mathbb{E}_0[p_t s_t] \tag{53}
$$

Now we have the Bellman equation:

$$
\begin{aligned}
J_t(p_{t-1}, w_t) &= min_{s_t}\mathbb{E}[p_t s_t + J_{t+1}(p_t, w_{t+1})] && (54)\\
&= min_{s_t}\mathbb{E}[p_t s_t + J_{t+1}(p_t, w_t - s_t))] && (55)\\
&= min_{s_t}\mathbb{E}[(f(p_{t-1}, s_t) + \epsilon_t)s_t + J_{t+1}(f(p_{t-1}, s_t) + \epsilon_t), w_t - s_t)] && (56)\\
& && (57)
\end{aligned}
$$

At time T we have: $w_{T+1} = 0$ so $s_T = w_T$ we sell everything at the last period and we have:

$$
\begin{aligned}
J_T(p_{T-1}, w_T) = f(p_{T-1}, w_T)w_T && (58)\\
& && (59)
\end{aligned}
$$

So we have the following recursion at time T-1

$$
\begin{aligned}
J_{T-1}(p_{T-2}, w_{T-1}) &= min_{s_{T-1}<w_{T-1}} \mathbb{E}[p_{T-1}s_{T-1} + J_T(p_{T-1}, w_T)] && (60)\\
&= min_{s_{T-1}<w_{T-1}} \mathbb{E}[p_{T-1}s_{T-1} + f(p_{T-1}, w_T)w_T] && (61)\\
&= min_{s_{T-1}<w_{T-1}} \mathbb{E}[p_{T-1}s_{T-1} + f(p_{T-1}, w_{T-1} - s_{T-1})(w_{T-1} - s_{T-1})] && (62)\\
&= min_{s_{T-1}<w_{T-1}} \mathbb{E}[p_{T-1}s_{T-1} + \frac{k_i p_{T-1}(w_{T-1} - s_{T-1})}{(\sqrt{k_i} - (w_{T-1} - s_{T-1})\sqrt{p_{T-1}})^2}] && (63)\\
& && (64)
\end{aligned}
$$

We want to minimize the function g, defined as: (we factorize by p and do the substitution $s_{T-1} = w_{T-1} + z$ so that the variations of z and s are monotonic)

$$
\begin{aligned}
g(z) &= z - \frac{kz}{(\sqrt{k} + z\sqrt{p})^2} + A && (65)\\
&= z - \frac{z}{(1 + z\sqrt{\frac{p}{k}})^2} + A && (66)\\
& && (67)
\end{aligned}
$$

Now we want to derive the first order condition of g(z) with respect to z, so that we can find the optimal z. (with our substitution z is negative)

$$
\begin{aligned}
\frac{dg(z)}{dz} &= 1 - \frac{1 - z\sqrt{\frac{p}{k}}}{(1 + z\sqrt{\frac{p}{k}})^3} && (68)\\
& && (69)
\end{aligned}
$$

Therefore, if $0 > z > -\sqrt{\frac{k}{p}}$ we have: $g'(z) < 0$ and thus the minimim is at $z = 0$ therefore $s_{T-1} = w_{T-1}$

The condition $z > -\sqrt{\frac{k}{p}}$ is always true because it equal to the number of token x in the pool.

This function is minimized when $s_{T-1} = w_{T-1}$ and the value of the function is:

$$
\begin{aligned}
J_{T-1}(p_{T-2}, w_{T-1}) &= \mathbb{E}[p_{T-1}s_{T-1}] && (70)\\
&= f(p_{T-2}, w_{T-1})w_{T-1} && (71)
\end{aligned}
$$

By immediate recursion, this is true for all t and it means that we better place the from the first period (or anytime) and then be null.

Indeed, This results makes sense because with the way we modeled price dynamics, it is either going to change with a trade of ours, or be noisy around the initial value given the epsilon distribution.

### 5.2.2 Binomial price dynamics

For this new approach, we start by making the same assumptions as before, (one pool, $T$, and no liquidity is added so $k$ remains constant)

Notation:

- $x_0$ and $y_0$ are liquidity at time 0.
- $\bar{S}$ is the total amount of tokens we want to trade.
- $r$ represent the one-period returns in the market prices taking values of $u$ with probability $q$ and $d$ with probability $1-q$.
- $s_t$ is the total amount of tokens x traded at time t
- $w_t$ the total amount of tokens left at time t

We have: $w_0 = \bar{S}$ and $w_{T+1} = 0$ since we have to trade everything within T time periods.

Price notation:

- $p_t^{pre}$ pre-trade price at time t
- $p_t^{post}$ post-trade price at time t, if we trade we move the market
- $p_t^{exec}$ the price with which we have executed the trade, used to calculate the costs of the trade at time t.

Price dynamics at time 0:

$$p_0^{pre} = \frac{y_0}{x_0} \tag{72}$$

$$p_0^{exec} = \frac{y_0}{x_0 - s_0} \tag{73}$$

$$p_0^{post} = \frac{k}{(x_0 - s_0)^2} \tag{74}$$

Price dynamics at time t:

$$p_t^{pre} = r p_{t-1}^{post} \tag{75}$$

$$x_t = \sqrt{\frac{k}{p_t^{pre}}} \tag{76}$$

$$y_t = \sqrt{k p_t^{pre}} \tag{77}$$

$$p_t^{exec} = \frac{y_t}{x_t - s_t} \tag{78}$$

$$p_t^{post} = \frac{k p_t^{pre}}{(\sqrt{k} - s_t \sqrt{p_t^{pre}})^2} \tag{79}$$

Or more compactly:

$$p_t^{pre} = r\frac{kp_{t-1}^{post}}{(\sqrt{k} - s_{t-1}\sqrt{p_{t-1}^{post}})^2} \tag{80}$$

$$p_t^{exec} = \frac{\sqrt{kp_t^{pre}}}{\sqrt{\frac{k}{p_t^{pre}}} - s_t} \tag{81}$$

$$p_t^{post} = \frac{kp_t^{pre}}{(\sqrt{k} - s_t\sqrt{p_t^{pre}})^2} \tag{82}$$

Our objective now becomes:

$$\min_{s_t} \mathbb{E}_0\left[\sum_{t=0}^{T} p_t^{exec} s_t\right] \tag{83}$$

$$\text{s.t} \sum_{t=0}^{T} s_t = \bar{S} \tag{84}$$

$$s_t \geq 0 \tag{85}$$

By linearity of expectations:

$$\min_{s_t} \sum_{t=0}^{T} \mathbb{E}_0[p_t^{exec} s_t] \tag{86}$$

$$\text{s.t} \sum_{t=0}^{T} s_t = \bar{S} \tag{87}$$

$$s_t \geq 0 \tag{88}$$

**Two time periods**

To simplify the model, we assume $T = 1$ and we only operate in two time periods as shown in Figure 5. Then the optimization formulation is:

$$\min_{s_0, s_1} p_0^{exec} s_0 + q p_1^{exec,u} s_1 + (1-q) p_1^{exec,d} s_1$$

$$\text{s.t} \quad s_0 + s_1 = \bar{S}$$

$$s_0,\ s_1 \geq 0$$

Basically to solve this we can express everything in terms of $s_0$. The form of the cost function then will depend on the parameters.

**Solution - Experiment 1** $x_0 = 5000, y_0 = 100000, S = 2000, q = 0.5, u = 1.5, d = 0.9$. The cost is a strictly decreasing function of $s_0$, so the solving the optimization problem yields $s_0 = 2000$, i.e buy everything at the first period

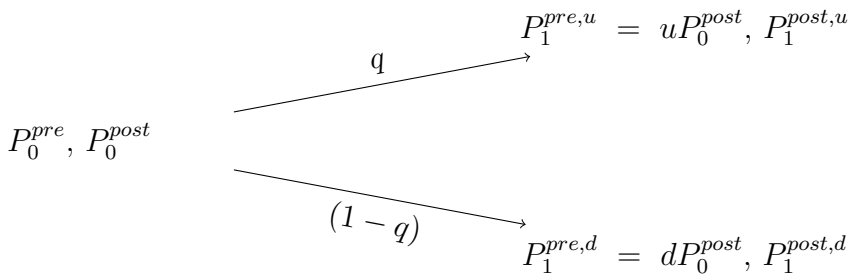
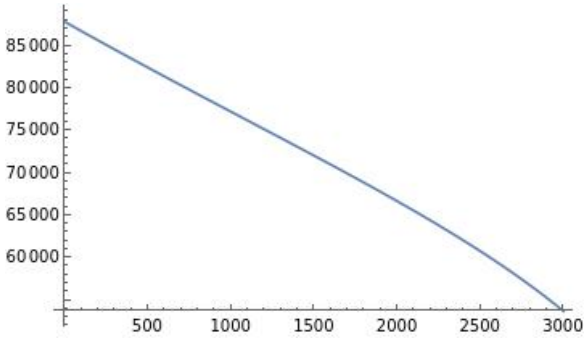
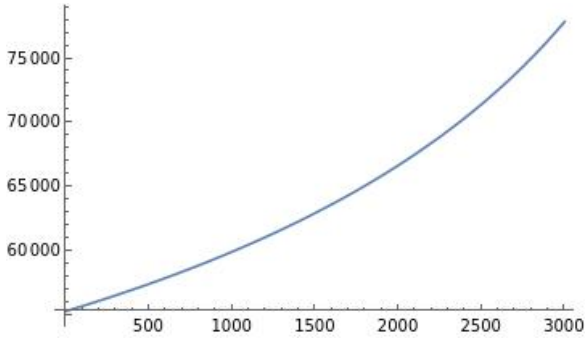



Figure 5: Two period binomial model for the evolution of the Market Price (Pre-trade and Post-trade)




(a) Experiment 1: $x_0 = 5000, y_0 = 100000, S = 2000, q = 0.5, u = 1.5, d = 0.9$. The cost is a strictly decreasing function of s0




(b) Experiment 2:$x_0 = 5000, y_0 = 100000, S = 2000, q = 0.5, u = 1.1, d = 0.6$. The cost is a strictly increasing function of s0

**Solution - Experiment 2** $x_0 = 5000, y_0 = 100000, S = 2000, q = 0.5, u = 1.1, d = 0.6$. The cost is a strictly increasing function of $s_0$, so the solving the optimization problem yields $s_0 = 0$, i.e buy everything at the second period

These two experiments are obvious because we have probability 0.5, and we expect the price to increase much more than it decreases in the first experiment, so we buy now. Contrary, we expect the price to decrease much more than it increases in the second experiment so we might as well wait to buy.

### N time periods

The Binomial Case with multiple time periods becomes more complex to solve because we cannot express conveniently everything in terms of $s_0$ as in the two periods. An approach that further study can explore is to solve the Bellman equation on that modelisation for price dynamics like we do in the random walk case.