2023-06-09

Optimal Ordering to Maximize MEV Arbitrage

Granton Michael White
Brigham Young University

Optimal Ordering to Maximize MEV Arbitrage

Granton Michael White

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Zach Boyd, Chair
Ben Webb
Mark Kempton

Department of Mathematics

Brigham Young University

ABSTRACT

Optimal Ordering to Maximize MEV Arbitrage

Granton Michael White
Department of Mathematics, BYU
Master of Science

The rise of cryptocurrencies has brought with it new math problems with new sets of constraints. The MEV problem entails solving for the ordering of pending trades that maximizes a block creator's profit. In decentralized finance, time is a big constraint, so an exhaustive search of all possible orderings is impossible. I propose a solution to the MEV problem that gives a near optimal result that can be solved in a reasonable amount of time. I layout the method and the formulas required for my solution. Additionally, I test my solution on synthesized data to show that it works as desired.

Keywords: arbitrage, cryptocurrency, blockchain, mev, maximal extractable value, miner extractable value, front-running, decentralized finance, defi, optimization, ordering

Acknowledgements

I would like to thank Zach Boyd, who allowed me to research topics of more interest to me, introduced me to Chris Hair, and took time out of his schedule to help me complete the requirements for my degree.

I would also like to thank Chris Hair who has been a wealth of knowledge in cryptocurrencies and decentralized finance. He was a constant help throughout this process and this thesis never could have been written without him.

And finally, I would like to thank my wife, Carly White. She took a break from her career and focused on raising our children to allow me to complete my master's program. Thank you.

Contents

Contents **iv**

1 Introduction **1**

2 The MEV Problem **3**

2.1 Non-cryptocurrency Example 3

2.2 Financial Background 4

2.3 Previous Work 8

2.4 The MEV Problem 10

2.5 Abstraction/Assumptions 10

3 Method **12**

3.1 Two Forms of Profit 12

3.2 Setting-up the Knapsack Problem 13

3.3 Solving the Dynamic Knapsack Problem 16

3.4 Additional Speed-ups 19

4 Formulas **23**

4.1 Profit from a Sandwich 23

4.2 Average Profit-per-trade 23

4.3 Adjusting the Average Profit-per-trade of Supersets 23

4.4 Allocation Between Sandwiches 24

4.5 Adjusting Average Profit-per-trade for Second Sandwiches 24

5 Tests **26**

5.1 Data 26

5.2 Testing 26

5.3 Results 27

A Derivations **29**

A.1 Profit Function 29

A.2 Allocation Between Sandwiches 30

A.3 Adjusting the Average Profit-per-trade of Supersets 31

B Profit Function Characteristics **33**

B.1 Near-linear with respect to a_0 33

B.2 Near-linear with respect to N 33

B.3 Monotonically increasing with respect to N 37

B.4 Monotonically increasing with respect to a_0 37

B.5 Concave with respect to a_0 37

Bibliography **38**

Chapter 1. Introduction

With the introduction of blockchain technologies and decentralized finance (DeFi) came a new set of math problems. Decentralized exchanges (DEXs) were created with automated market makers (AMMs) which set exchange rates algorithmically instead of by a bid-ask system like most centralized exchanges (CEXs). This algorithmic setting of exchange rates opens up opportunity to manipulate the exchange rates. Further, with Ethereum switching from a proof-of-work (PoW) system to a proof-of-stake (PoS) system, these exchange rates can be manipulated and profited on with very little risk. *Arbitrage* is taking advantage of the inefficiencies in a market to make money with little-to-no risk. Manipulating these exchange rates for a profit is a form of arbitrage.

Currently, there are over 24 thousand different cryptocurrencies totaling over one trillion USD across the world. With over 28 billion USD in cryptocurrency traded across 625 different exchanges every day, there is a large monetary incentive to manipulate exchange rates for profit [1].

Traditionally, block creators (called miners in a PoW system and called validators in a PoS system) make money from the block reward, transaction fees, and tips (also called priority fees). By taking advantage of the arbitrage opportunity created by DEXs, validators can increase their profit. The maximal extractable value (MEV), formally called miner extractable value, is the maximum amount a block creator can profit from organizing a block. Solving for the MEV is call *the MEV problem*.

Time is the largest constraint when solving for the MEV. Validators have just 12 seconds to propose a block and get it verified on Ethereum's blockchain, so an exhaustive search of all possible orderings of trades is not possible. A validator needs an efficient way to solve for order of trades if they seek to get the MEV from the block.

The MEV problem is a very interesting math problem. It is a combinatorial optimization problem framed in the context of cryptocurrency markets. My solution to the MEV problem

reframes it as a special case of the knapsack problem. My solution to the MEV problem, therefore, can be applied to all similar knapsack problems, not only to problems in DeFi.

The solution I propose, though does not guarantee the MEV, gives a solution that should be very close to the MEV. It does this in an efficient manner that can be solved in a reasonable amount of time. Testing on synthesized data shows that my method generates substantial profits over other strategies.

I make assumptions and abstractions to simplify the problem. Most of these assumptions are purely for convenience and would not change the solution by relaxing them. Removing other assumptions I make would add additional constraints unaccounted for in my solution. Because of the assumptions I make, my solution does not solve the MEV problem as posed in the real world. Nonetheless, my solution solves a big piece of the MEV puzzle and is (to the best of my knowledge) the first solution to the constrained combinatorial optimization problem posed.

Chapter 2. The MEV Problem

2.1 Non-cryptocurrency Example

To further clarify the problem for those who don't have a background in DeFi or cryptocurrency, and to show applications to a broader group of math problems, I will give an example of a similar problem that does not have anything to do with cryptocurrency.

Imagine you are packing to go survive in the wilderness. There are a bunch of tools and survival equipment in front of you. You can choose whatever items you want, but the total combined weight of the items you choose has to be less than some fixed amount. This is a classic example of a knapsack problem.

You may think picking the items that give you the most value relative to their weight would be the best thing to do, but in the real-world, however, there are more things to consider. The utility of the items, or value they give you in the wilderness, could be dependent on one another.

For example, each piece of a tent by itself does not have much value, but when the pieces are combined they have a lot of value. You wouldn't take the tent poles without taking the other pieces of the tent too. Their value is dependent on whether or not you have the other pieces. Perhaps there are even two different tents to choose from. Each tent independently gives you a lot of value, but if you already have a tent, an additional tent doesn't benefit you as much. The second tent, therefore, drops in value as soon as you choose the first tent.

By adding complexities, we are changing the problem and making it a lot harder to solve for the optimal solution. This problem is almost identical to the example I am trying to solve, just with a different set of constraints and complexities. Each choice gives you value, but there are also additive and antagonistic effects with the other choices.

2.2 Financial Background

Since very few people are familiar with DeFi, I have tried to give the necessary background required to fully understand the problem and its application. I have simplified a lot of the more complex topics since a full understanding of each topic is not required for this paper. The following background should be sufficient.

Arbitrage. *Arbitrage* is taking advantage of the inefficiencies in a market to make money with little-to-no risk. This is most commonly done by capitalizing on a difference between prices in a market or between markets.

A simple example would be profiting from the difference between the prices of meals and the individual items at McDonald's. If a burger costs \$3, fries cost \$2, and a soda costs \$1, but the meal (which contains all three) only costs \$5, there is an arbitrage opportunity. You could buy a meal and then sell the individual items at their market value. You'd profit \$1 for each meal you bought/sold. Though this is an example of an arbitrage opportunity, I do not recommend using this as a profit generating strategy.

Currency arbitrage is taking advantage of different exchange rates between currencies to make money. For example, if the exchange rate between the USD and EUR is 1:1, USD and GBP is 3:1, and GBP and EUR is 1:2, you could profit from just exchanging currencies. You could exchange 100 USD for 100 EUR, then the 100 EUR for 50 GBP, then the 50 GBP for 150 USD for a profit of 50 USD. If all of this is done simultaneously, this can be done with no risk.

The arbitrage opportunity created by AMMs on DEXs is similar, but not the exact same. The idea is that since the exchange rate is set algorithmically, if you have the right to order trades how you want, you can purposefully change the exchange rate in one direction. By placing your own trades, you can profit from this change in the exchange rate.

Blockchain and Block Creators. The exact ins and outs of blockchain are not necessary to understand for this paper, so I will cover just what is necessary. *Blockchain* is basically

like a ledger of all the past trades of a given cryptocurrency. New "blocks" of trades are added periodically by block creators, also called validators in a PoS system. Block creators pick trades from a large list of pending trades and organize them into a list or "block".

The size of a block differs between cryptocurrencies. Bitcoin limits its block size by memory while Ethereum limits by computation. Each trade on the block takes up a different amount of memory/compute. For the purpose of this paper, I assume each trade takes up the same amount of space on the block. This is equivalent to assuming there are a fixed number of trades allowed per block.

Validators make money through the block reward, transaction fees, and tips (also called priority fees). I choose to ignore block rewards since they have no effect on the solution to the MEV problem. I also group the transaction fees and the tips into one value called tips. The tip is chosen by the person who submitted the trade. The idea is that a higher tip for the validator will give your trade priority and your trade is more likely to be executed in the next block.

Decentralized Exchanges. Most centralized exchanges (CEXs) work almost like a live auction. For every asset there is a bid and an ask. The bid is the most someone is willing to buy the asset for while the ask is the lowest someone is willing to sell the asset for. Generally, the ask is greater than the bid. When these two prices cross, a trade is made. The people or organizations providing the bids and asks are called *market makers*.

Decentralized exchanges (DEXs) work differently. Instead of a bid-ask system, they generally use automated market makers (AMMs) operating over a pool of assets called the liquidity pool. An AMM uses an algorithm to determine the price of the given assets. The most common of these is a constant product formula. This means that there is a certain amount of two assets in a pool and at any given time their product must be equal to a constant ($A \times B = C$). So if you put in some of one asset, you only get out the amount of the other asset that doesn't change the constant. The assets in liquidity pools are tokens.

Even though there is a difference between tokens and coins, I use the term interchangeably throughout this paper.

For example, if there is a pool with 300 of token A and 100 of token B, we have $300 \times 100 = 30,000$, so the constant associated with this pool would be 30,000. If someone wanted to exchange 50 of token B for token A, they would receive 100 of token A since $(300 - 100) \times (100 + 50) = 200 \times 150 = 30,000$. If a second person then wanted to exchange 50 of token B for A, they would only receive 50 of token A since $(200-50) \times (150+50) = 150 \times 200 = 30,000$. In practice, the liquidity pools are much larger than this, so the magnitude of changes in the exchange rates are much smaller.

The most popular DEX is Uniswap. Uniswap uses an automated liquidity protocol powered by a constant product formula like in the example above. The AMM in different exchanges have different liquidity protocols; even Uniswap has different versions with different fee structures. For this paper, I'm assuming every trade goes through one, decentralized exchange whose pools use a constant product formula, similar to Uniswap V2.

Sandwich Arbitrage. Since exchange rates are set algorithmically in DEXs, each trade has a known impact on the exchange rate before being placed. Block creators can organize trades in such a way that by including their own trades they profit from this manipulation of exchange rates. Additionally, there are different DEXs and CEXs one can arbitrage between. This makes it so there are many different ways to arbitrage in cryptocurrency markets. This includes but is not limited to sandwiches, triangular arbitrage, spacial arbitrage, reverse sandwiches, cross-protocol arbitrage, time-bandit attacks, and forking attacks.

In this paper, I am only considering the arbitrage created by sandwiches. This fits well with my assumption that every trade is through one DEX. A *sandwich* is where the block creator "sandwiches" other people's trades between two of their own. Since the other people's trades will change the exchange rate in a known way, the arbitrageur can trade with one exchange rate and then trade back with a different exchange rate, profiting from the difference between the two.

For example, if the exchange rate between token A and token B is 1:1, the arbitrageur can trade 1 token A for 1 token B. If they then order a bunch of other people's trades from A to B, they can move the exchange rate from 1:1 to 1.2:1. The arbitrageur can then exchange their B token for 1.2 A tokens, profiting 0.2 of token A.

Since the trades are processed in blocks, it makes the risks associated with performing a sandwich very low. That's what makes it a good arbitrage opportunity. Sandwiches are very similar to front-running in CEXs. Though front-running in CEXs is illegal, sandwiches in DEXs are legal.

Note that some trades have a minimum quantity associated with them to prevent others from moving the exchange rates against them too much. This is called a *slippage tolerance*. There are ways to order trades so they are less of an issue, but I have chosen to just ignore these for now.

The Knapsack Problem. Though not a financial topic, many people may not be familiar with the knapsack problem. Since my solution transforms the MEV problem to a knapsack problem, a basic knowledge of the knapsack problem is helpful.

The knapsack problem is a combinatorial optimization problem. Given a set of items with corresponding weights and a values, the solution to the knapsack problem is the combination of items with total combined weight less than a given constant that maximize the total combined value. In the case that you can't repeat or divide items, it is known as the 0-1 knapsack problem.

The knapsack problem is an NP-complete problem meaning there is no known solution that can be solved in polynomial time. The greedy solution would be to repeatedly pick whichever item has the greatest value relative to its weight that doesn't put you over the allowed weight. You continue this until there are no remaining items that you can add (given your weight constraint). Though this tends to do well, it does not guarantee an optimal solution. The following example helps clarify why. Consider the following objects:

item	weight	value
A	6	8
B	5	5
C	5	5
D	4	1

Consider a bag that can hold a weight of 10. The relative value given the weight of each item ($\frac{\text{value}}{\text{weight}}$) is $\frac{4}{3}$, 1, 1, $\frac{1}{4}$ respectively. The greedy solution would tell you to choose item A first. The only item remaining you could take (since you are only allowed to take a total weight of 10) would be D. Together that gives a total value of 9. If you would have chosen item B and C, you would've had a value of 10. Thus the greedy solution didn't give you the optimal solution.

Though the greedy solution doesn't guarantee you the optimal solution, it does tend to result in a pretty good solution. Since blocks hold a large amount of trades with each trade taking up a small amount of space, our problem is well suited for the greedy solution giving a near-optimal solution. Plus, finding the actual solution requires dynamic programming which would take too long given our time constraint. Thus, the greedy solution is our best option.

2.3 Previous Work

Much of the current literature analyzes the existence and impact of arbitrage in cryptocurrency markets [2] [3] [4]. Kaihua Qin, Liyi Zhou, and Arthur Gervais also look at how to detect sandwich arbitrage in DEXs [3]. They show that MEV arbitrage yielded 540.54 million USD in profit over a 32 month period. Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais quantify the probability of being able to perform a sandwich based on the relative positioning of a transaction within a block [5].

Since MEV arbitrage is seen as an invisible tax on other traders since it moves exchange rates against them, it is generally regarded as a "bad" thing. Because of that, most of the research is about how to stop or prevent MEV attacks instead of how to best execute them (as I do in my paper). Most research focuses on changes that could be made to DEXs or AMMs to stop the front-running of other's trades [6] [7] [8]. Lioba Heimbach and Roger Wattenhofer propose a protection against sandwiches using an optimal slippage tolerance [9].

There is very little research on how to best arbitrage cryptocurrency markets. Kaihua Qin, Liyi Zhou, and Arthur Gervais are the first, to my knowledge, to propose an algorithm that claims to get the MEV from a block [3]. The issue with their solution is that they view each trade independently and look at the sandwich arbitrage potential of a single trade. They ignore the possibility of grouping similar trades together. This would move the exchange rate more, generating more profit. It would also limit the amount of personal trades the arbitrageur would need to include in the block, allowing more room for other trades to profit from. Because of this, their solution is far from the MEV. In fact, the profit they estimate that their algorithm would have generated is less than 7% of the amount of profit that they claim was actually generated through sandwich arbitrage over the same time period (which would have to be less than the optimal MEV). Sandwiching each trade independently does not result in a solution anywhere near the MEV of a block.

Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente also pose a solution to the MEV problem [10]. In their solution, they do not account for limited space in a block. The solution they propose orders all the trades, with the arbitrageur's mixed in, that maximizes profit. This solution assumes that all available trades and all trades the arbitrageur submits fit on the block which is generally not true for any cryptocurrency. This is not the same problem I seek to solve. I am answering the problem of picking which trades to put on the block, in what order, and what personal trades to include to maximize profits.

2.4 The MEV Problem

The question I seek to answer is *Given a set of trades, how can a validator organize 100 of either those trades or their own to get as close to the MEV as possible and how can this be solved quickly?* This is done by maximizing the profit from the tips of each trade while simultaneously maximizing the profit generated from sandwich arbitrage.

2.5 Abstraction/Assumptions

In order to make this problem more tractable, I made some assumptions and abstracted from the real world. Even after these abstractions, the solution can still be applied to the real world: adjustments would just have to be made. The method would be the same.

I am assuming all trades flow through one, decentralized exchange whose pools use a constant product formula. An example would be if all trades flowed through Uniswap V2. I am doing this to avoid the complexity of arbitrage between exchanges, which though could increase profits, makes the problem a lot harder.

I am also ignoring block rewards. These do not change the optimal ordering so it does not change the solution to the MEV problem. I also group the transaction fees and the tips into one value called tips. This doesn't change the problem at all, but simplifies notation quite a bit. I also ignore any transaction costs.

There technically are other types of trades that are rare that I am ignoring. This include deposits and withdrawls. This is an unneeded complication that doesn't need to be considered.

I'm assuming there are a fixed number of trades allowed per block. As mentioned in 2.2, the exact number of trades-per-block changes based on a couple of factors. This is equivalent to assuming each trade takes up the same amount of space on a block. Though this is not a realistic assumption, it is a very reasonable assumption.

I am also choosing to ignore the minimum quantity or slippage tolerance some trades have. The purpose of these is to stop the exchange rate from changing too much before the trade is executed. Though I am ignoring these, they can be included without too much alteration. This would include ordering trades based on their minimum quantities or potentially moving the exchange rate in the opposite direction before beginning a sandwich.

I am assuming all the arbitrage opportunity is generated through sandwiches. As mentioned 2.2, there are other ways to arbitrage, but they either not considered or not relevant given the other assumptions made. In order to perform a sandwich, the arbitrageur needs some of the token to place their first trade. I am assuming the arbitrageur has some fixed amount of each coin in order to perform sandwiches.

Chapter 3. Method

In this chapter, I will walk through my method of reframing the MEV problem as a knapsack problem. I will then walk through the necessary updates to the trades as they are added to the block. Please refer to figure 3.1 for a diagram of the steps.

3.1 Two Forms of Profit

There are two ways for a validator to profit from including a trade on the block. The first is the tip and the second is by using that trade in an arbitrage attempt. Since the only form of arbitrage we are considering is sandwiches, trades provide value to validators through the tip and by sandwiching the trade.

A validator sandwiches other's trades because those trades have a known impact on the exchange rate. The validator can profit from the change in exchange rate. By sandwiching multiple trades together, the validator can move the exchange rate more, profiting more. Note that all the trades sandwiched would need to be through the same liquidity pool meaning all the trades are between the same two coins (one directional).

Let $\mathfrak{C}$ be the set of all available tokens and let $\mathbb{A}, \mathbb{B}, \mathbb{C} \in \mathfrak{C}$ be three arbitrary tokens. In order to perform a sandwich between token $\mathbb{A}$ and $\mathbb{B}$, the arbitrageur would first need to trade some amount of token $\mathbb{A}$ for token $\mathbb{B}$. Call this number a_0. They would receive some amount of token $\mathbb{B}$ for the a_0 of token $\mathbb{A}$ they sold. Call this number b. The arbitrageur would then order other trades from token $\mathbb{A}$ to token $\mathbb{B}$. This changes the exchange rate between token $\mathbb{A}$ and $\mathbb{B}$. They then sell the b token $\mathbb{B}$ they received from the initial trade back to token $\mathbb{A}$. Since the exchange rate has changed, the amount of $\mathbb{A}$ they receive should be more than the a_0 they started with. Let a_f be the amount of token $\mathbb{A}$ they receive from selling b token $\mathbb{B}$. The profit the arbitrageur gets from that sandwich would be $a_f - a_0$. The total profit of the block would be the profit from arbitrage plus all the tips of the trades included in the block.

Throwing additional trades in the sandwich (such as trades between token $\mathbb{A}$ and $\mathbb{C}$) does not increase the arbitrage profit of the sandwich since those trades would be executed in a different liquidity pool; they would have no effect on the exchange rate between token $\mathbb{A}$ and $\mathbb{B}$. Including trades from token $\mathbb{B}$ to token $\mathbb{A}$ in the sandwich would decrease the sandwich's profit since they would push the exchange rate in the opposite direction.

3.2 Setting-up the Knapsack Problem

Let $\mathcal{T} = \{T_i\}$ (red in figure 3.1) be the set of all pending trades where each trade $T_i = (q_i, c_i^s, c_i^b, t_i)$ is a 4-tuple of the quantity being traded, the token being sold, the token being bought, and the tip (or priority fee).

1. Group all the trades in $\mathcal{T}$ by their (c_i^s, c_i^b) pairs (blue in figure 3.1). Since it is not beneficial to include trades between multiple different tokens in a single sandwich, we only need to consider the possible sandwiching of trades between the same two coins. Let G^{AB} be the group of all trades between token $\mathbb{A}$ and $\mathbb{B}$.

2. Take the power set of each $G^{c^s c^b}$ (green in figure 3.1). Each possible subset of G^{AB} is a possible sandwich candidate. Let $\{g_i^{AB}\} = \mathcal{P}(G^{AB})$. I call these groups of pending trades *sandwich groups*.

3. Calculate the profit generated by sandwiching each $g_i^{c^s c^b}$. Due to the AMM, each trade has a known impact on the exchange rate. We can use this to calculate the total profit the arbitrageur would receive from sandwiching g_i^{AB} given a starting trade of a_0.

4. Calculate the Average Profit-per-trade for each $g_i^{c^s c^b}$. The total profit generated by g_i^{AB} is the arbitrage profit plus the sum of the tips in g_i^{AB}. The average profit-per-trade of g_i^{AB} would be that number divided by two plus the number of trades in g_i^{AB}. We need

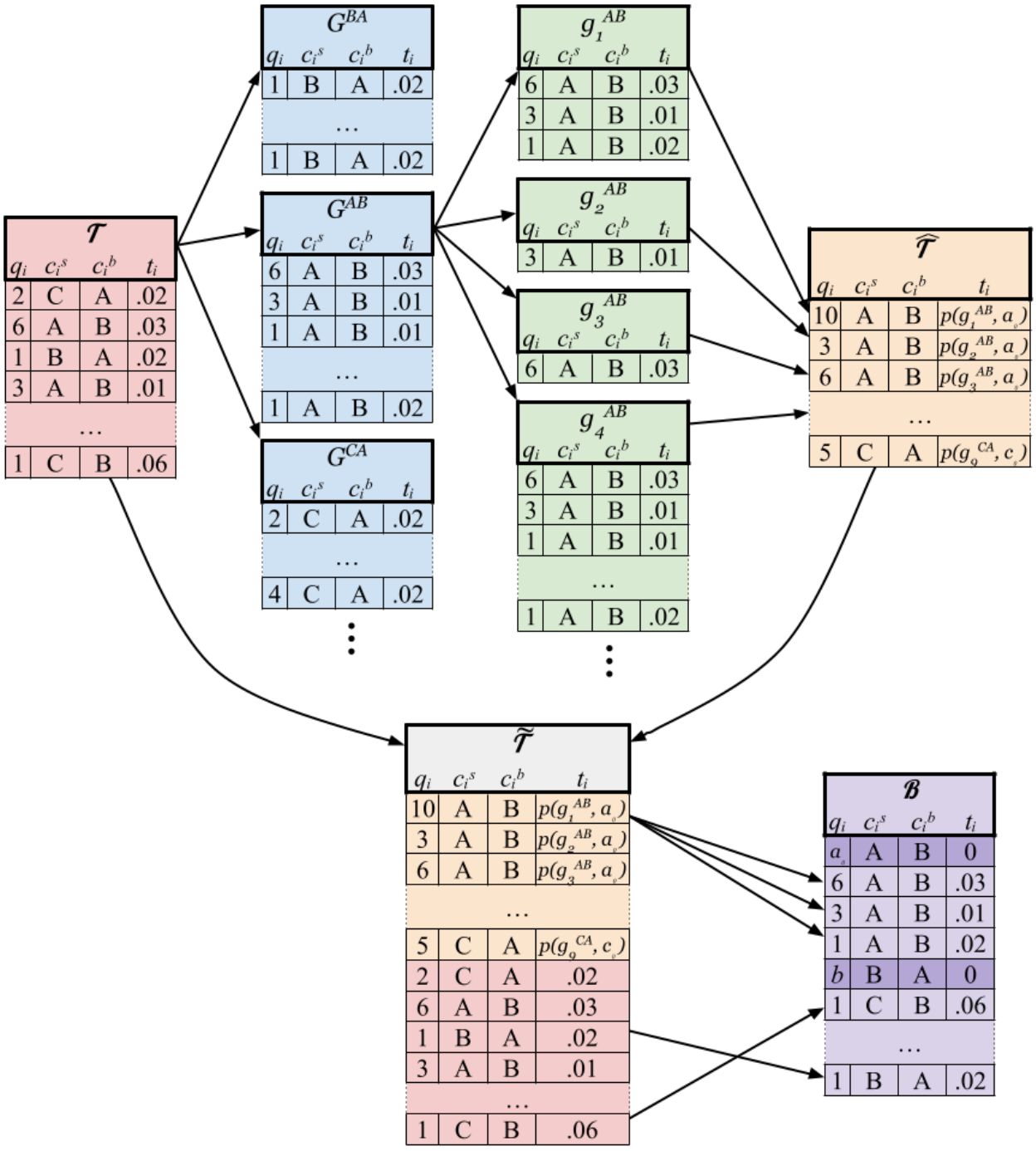

Figure 3.1: a diagram of turning the MEV problem into a knapsack problem

to add two since, in order to sandwich trades, the arbitrageur would need to include two of their own trades.

Referring back to the knapsack problem in 2.2, the total profit generated by g_i^{AB} is the "value" and the number of trades in the sandwich is the "weight" since only a fixed number of trades can fit on the block. The average profit-per-trade is the $\frac{\text{value}}{\text{weight}}$.

5. Organize all $g_i^{c^s c^b}$ into a set $\hat{\mathcal{T}}$ (orange in figure 3.1). Define $\hat{\mathcal{T}} = \{\hat{T}_j\}$ where $\hat{T}_j = \left(\sum q_i,\ c_j^s,\ c_j^b,\ p(g_j^{c^s c^b},\ c_{j,0}^s)\right)$ for all $g_j^{c^s c^b} \in \mathcal{P}(G^{c^s c^b})$ and all $c^s, c^b \in \mathfrak{C}$. $\hat{\mathcal{T}}$ gives us a list of all of the possible sandwiches and their respective average profit-per-trade. The tips associated with single trades are very comparable to the average profit-per-trade of the sandwich groups. Both are the amount of profit the validator brings in for each trade it takes up on the block. In figure 3.1, the average profit-per-trades are stored in the tips column. You can think of the tip column here as the average profit-per-trade.

6. Combine single trades $\mathcal{T}$ and sandwich groups $\hat{\mathcal{T}}$ into a single set $\widetilde{\mathcal{T}}$ (orange and red in figure 3.1). Define $\widetilde{\mathcal{T}} = \mathcal{T} \cup \hat{\mathcal{T}}$. This gives us a single set of either single trades or groups of trades (organized in a sandwich) and their respective average profit-per-trade. For single trades, the average profit-per-trade is just the tip. For a sandwich, the average profit-per-trade is the sum of the tips plus the profit generated by the arbitrage, all divided by the total number of trades in the set plus two. Note that both are stored in the tips column in figure 3.1.

Each time the validator adds to the block, they are either adding a single trade (profiting from the tip alone) or a group of trades in a sandwich (profiting from the sum of the tips and arbitrage profit). $\widetilde{\mathcal{T}}$ is then a list of all the things the validator can add to the block.

7. Add trades/groups with the highest average profit-per-trade to the block $\mathcal{B}$ (Purple in figure 3.1). We have now set up the MEV problem in a way that is very similar to a classic knapsack problem: $\widetilde{\mathcal{T}}$ is our set of "objects", each item's profit is its "value",

and the space it takes up on the block is the "weight". If a group is added to the block, the validator needs to first add their initial trade, followed by each of the individual trades in the group, and finishing with their trade back. The arbitrageur's trades are highlighted in dark purple in figure 3.1.

There is a big difference, however, between how we've set up the MEV problem and the knapsack problem. After adding an "item" (which in this case is either a trade or a sandwich group) to the block it effects all the other trades. Many of the sandwich groups contain the same trade, so as one group or trade is added to the block, the "weight" and " value" of any other groups containing those trades need to be adjusted. Many trades need to be removed. The arbitrageur also only has a fixed amount of each token to start a sandwich with, so including one sandwich affects the profit generated by sandwiches between different tokens. This is what makes this problem a *dynamic* version of the knapsack problem and makes solving for the solution a little harder.

3.3 Solving the Dynamic Knapsack Problem

The greedy solution to the knapsack problem would be to pick the trade or sandwich group with the largest average profit-per-trade to the block. Note that you would only consider groups containing a total number of trades less than or equal to the space remaining on the block. As mentioned in 3.2, our problem is dynamic and needs to be updated continuously. These are the steps that need to be taken each time the validator wants to add to the block.

1. Limit $\widetilde{\mathcal{T}}$ to items that fit in $\mathcal{B}$. Since we are assuming only a fixed number of trades can fit on the block, we must restrict $\widetilde{\mathcal{T}}$ to only consider groups that can fit on $\mathcal{B}$. As more trades are added to the block, $\widetilde{\mathcal{T}}$ will need to be restricted more and more.

2. Add the item with the highest average profit-per-trade. Following the greedy solution to the knapsack problem, we would add the item with the highest $\frac{\text{value}}{\text{weight}}$ ratio.

This corresponds with the trade or sandwich group with the largest average profit-per-trade. Without loss of generality, call this item $\tilde{T_k}$.

3. Remove subsets of $\tilde{T_k}$ from $\mathcal{B}$. There may be subsets of $\tilde{T_k}$ already on the block. This could include single trades or sandwich groups. We cannot double-count trades, so they need to be removed.

Let $\tilde{T_b}$ be some trade or group on the block $\mathcal{B}$ such that $\tilde{T_b} \subset \tilde{T_k}$. Though $\tilde{T_b}$ has a higher average profit-per-trade than $\tilde{T_k}$ (hence why it got added to the block first), $\tilde{T_k}$ has a higher total profit. This is because all of the tips included in $\tilde{T_b}$ are also included in $\tilde{T_k}$ and all the arbitrage profit generated by $\tilde{T_b}$ is also generated by $\tilde{T_k}$. Since our goal is to maximize total profit, we want to replace $\tilde{T_b}$ with $\tilde{T_k}$.

4. Adjust supersets of $\tilde{T_k}$ in $\widetilde{\mathcal{T}}$. Many of the sandwich groups in $\widetilde{\mathcal{T}}$ may contain $\tilde{T_k}$. These groups may still be added to $\mathcal{B}$ later, but their average profit-per-trade needs to be adjusted. Let $\tilde{T_t} \in \widetilde{\mathcal{T}}$ be some superset of $\tilde{T_k}$, i.e. $\tilde{T_k} \subset \tilde{T_t}$.

The amount of profit $\tilde{T_t}$ generates needs to be adjusted to be only the amount of additional profit above $\tilde{T_k}$. This is done by subtracting the profit of $\tilde{T_k}$ from the profit of $\tilde{T_t}$. This removes all the tips of common trades between $\tilde{T_k}$ and $\tilde{T_t}$. This also removes the effect the trades in $\tilde{T_k}$ have on the arbitrage profit of $\tilde{T_t}$.

The space $\tilde{T_t}$ takes up on the block also needs to be adjusted because if $\tilde{T_t}$ is added to the block, $\tilde{T_k}$ needs to be removed from the block (see step 3). The additional space $\tilde{T_t}$ takes up on the block is then just the difference between the two.

5. Remove non-supersets of $\tilde{T_k}$ from $\widetilde{\mathcal{T}}$. Any trade in $\widetilde{\mathcal{T}}$ that is between the same two coins as $\tilde{T_k}$ that is not a superset of $\tilde{T_k}$ can be removed. This includes subsets, disjoint sets, and non-containing overlapping sets. Though this step is not required, it limits the size of $\widetilde{\mathcal{T}}$ which may speed up the process a bit. A smaller $\widetilde{\mathcal{T}}$ results in less items to search through and also results in less groups that need to be updated.

Subsets of $\tilde{T_k}$ can be removed since all the trades in them are already on the block. If we were to try and update their average profit-per-trade, we would see that the additional profit they provide above $\tilde{T_k}$ would be zero. We would also see that the additional space they would take up on the block would be zero. In other words, they are basically already on the block since they are contained in $\tilde{T_k}$ and do not need to be considered.

Disjoint sets can also be removed. These are single trades or sandwich groups that are between the same two coins as $\tilde{T_k}$ but don't share any common trades with $\tilde{T_k}$. Disjoint single trades can be removed since equation (4.1) is monotonically increasing with respect to N (see Appendix B.3). This means it would be more profitable to include it in the sandwich than to have it on the block separately. Disjoint sandwich groups can be removed since it would be better to combine the sandwiches into one instead of putting them on the block separately. It would take up less space on the block because it would only require two additional trades from the arbitrageur instead of four. In either case, if we wanted to add a single trade or sandwich group that is disjoint to $\tilde{T_k}$, we would just remove $\tilde{T_k}$ from the block and add the union of the two sets instead (see step 3). Note that the union already exists in $\widetilde{\mathcal{T}}$ and could replace $\tilde{T_k}$ later.

We can also remove any non-containing, overlapping sets. These are sandwich groups who share some trades with $\tilde{T_k}$, but $\tilde{T_k}$ contains trades they don't have and they contain trades $\tilde{T_k}$ doesn't have. Once again, it would be better to combine these two sandwiches on the block than to include them separately. Combining them takes up less space on the block leaving more free space to add more trades, increasing profits. If we wanted to add a non-containing overlapping set, we would just remove $\tilde{T_k}$ from the block and add the union of the two sets instead.

6. Adjust groups with same starting coin from $\widetilde{\mathcal{T}}$. The arbitrageur only has a certain amount of each token they can use for a sandwich. Let a_0 represent the amount of token $\mathbb{A}$ they have. Since equation (4.1) is monotonically increasing with respect to a_0 (see Appendix B.4), the arbitrageur should allocate all a_0 of their token $\mathbb{A}$ to the sandwich in order to

maximize profits. This changes as soon as we include the opportunity to perform a second sandwich, such as one between token $\mathbb{A}$ and $\mathbb{C}$.

Since equation (4.1) is concave with respect to a_0 (see Appendix B.5), each additional token the arbitrageur puts towards a sandwich gets less additional return than the previous one. Therefore, if there are two potential sandwiches (token $\mathbb{A}$ to $\mathbb{B}$ and token $\mathbb{A}$ to $\mathbb{C}$), and assuming the arbitrageur only has a certain amount of token $\mathbb{A}$ to allocate toward a sandwich, there is an optimal allocation between the two sandwiches that maximizes profits.

In order to account for the effect adding a sandwich to the block has on other sandwiches already on the block that start with the same token, the average profit-per-trade of sandwiches in $\widetilde{\mathcal{T}}$ need to be adjusted each time a sandwich is added to $\mathcal{B}$. This is done by first calculating the optimal allocation between the two sandwiches. Then, the profit of the groups needs to be changed to be the profit of the group given the new allocation minus the decrease in profit the new allocation would have on the group just added to $\mathcal{B}$.

If one of the second sandwiches gets added to the block and there are still sandwich groups off the block that start with the same token, we must recalculate the optimal allocation between the three potential sandwiches. The same would be done when considering a fourth sandwich and so on.

7. Repeat until $\mathcal{B}$ is full. Steps 1-6 can be repeated until the block $\mathcal{B}$ is full or until there are no more pending trades in $\widetilde{\mathcal{T}}$ left to add to $\mathcal{B}$.

3.4 Additional Speed-ups

There are additional speed-ups that can be made. They are optional, but may result in a faster solution. The first speed-up will still yield the same result while the second may have a small, negative impact on the profit the arbitrageur receives from the block.

1. Limit the power set calculation. The main computation cost in our solution is computing the power sets of $G^{c^s c^b}$ for all $c^s, c^b \in \mathfrak{C}$ and their respective profits. If we

can reduce the number of sets to consider, we can drastically limit our computation costs. Consider the following two trades:

$$(2,\ \mathbb{A},\ \mathbb{B},\ 0.05)$$

$$(1,\ \mathbb{A},\ \mathbb{B},\ 0.02)$$

The first trade is better for a block creator in every way. Since it has a higher quantity, it has more arbitrage potential. Additionally, it has a higher tip. There is no case that a validator would include the second trade without the first. When computing our power sets, we only need to compute sets that include only the first trade, both trades, or neither trade. We can ignore the ones that include the second without the first. This can speed up our solution by reducing the number of calculations we do.

2. Assume near-linearity. Since the arbitrageur will most likely have a small number of a given token relative to the amount in the pool, the profit function (4.1) will be almost linear (shown in Appendix B.1). This means the optimal thing to do in most cases will be to put all the tokens into one sandwich and avoid the extra cost of the two additional trades required to perform the second sandwich.

Figure 3.2 is a plot of the optimal allocation between two sandwiches. It assumes the two sandwiches have the same number of tokens in the liquidity pools (A_0^B) and the same number of trades used to push the exchange rate (N). In other words, the two sandwiches are identical in the sense that they provide the same arbitrage opportunity.

This shows that even though the two sandwiches give the same arbitrage opportunity and equation (4.1) is concave with respect to a_0 (Appendix B.5), it is optimal to allocate all the funds to one sandwich in liquid markets. Once the pool hits a point of illiquidity (meaning the sandwich contains a large enough number of coins relative to the liquidity pool) it is better to split your starting tokens evenly between the two sandwiches. In this case, that point is where the sum of a_0 and N is about 0.002% of the liquidity pool. These numbers do not remain constant for all cases. In the case where the sandwiches do not have the same

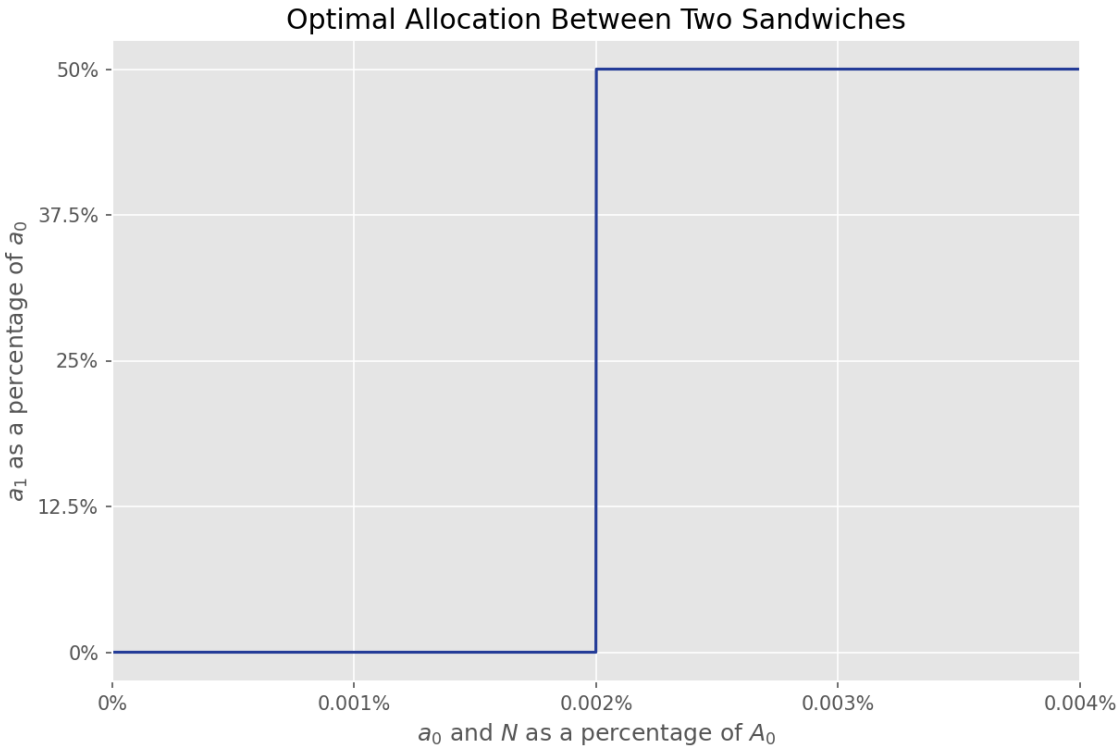

Figure 3.2: the optimal allocation between two identical sandwiches.

arbitrage opportunity, both the point where it becomes optimal to allocate between both sandwiches and the allocation percentage will change.

We can assume the amount the arbitrageur is trading (a_0) is very small relative to the amount in the liquidity pool (A_0^B) and that it would be optimal for the arbitrageur to allocate all of their funds to the one sandwich and ignore the other. Though this isn't always the best thing to do, in most cases it would be. Removing all those other groups from consideration will decrease computation time drastically since we will not need to calculate the optimal allocations between sandwiches. This will also reduce the number of items in $\widetilde{\mathcal{T}}$ we are considering, reducing computation time.

Chapter 4. Formulas

4.1 Profit from a Sandwich

Let g_k^{AB} be an arbitrary group of trades between token $\mathbb{A}$ and $\mathbb{B}$. Define $N = \sum \left(q_i \mid T_i \in g_k^{AB}\right)$ as the total amount of token $\mathbb{A}$ being traded for token $\mathbb{B}$. Let a_0 represent the amount the arbitrageur allocates to their initial trade in the sandwich. Let A_0^B be the starting amount of token $\mathbb{A}$ in the liquidity pool. The profit the arbitrageur receives from the sandwich is

$$P\left(g_k^{AB}, a_0\right) = \frac{a_0 N(a_0 + 2A_0^B + N)}{a_0^2 + 2a_0 A_0^B + a_0 N + (A_0^B)^2}. \tag{4.1}$$

The derivation can be found in Appendix A.1.

4.2 Average Profit-per-trade

Let g_k^{AB} be an arbitrary group of trades between token $\mathbb{A}$ and $\mathbb{B}$. Define $L = |g_k^{AB}|$ as the number of trades in g_k^{AB}. Let a_0 represent the amount the arbitrageur allocates to their initial trade in the sandwich. The average profit-per-trade of sandwiching g_k^{AB} is

$$p(g_k^{AB}, a_0) = \frac{P(g_k^{AB}, a_0) + \sum_{i=1}^{L} t_i}{L + 2}. \tag{4.2}$$

Note that we must add the 2 in the denominator to account for both of the arbitrageur's own trades.

4.3 Adjusting the Average Profit-per-trade of Supersets

Let $\tilde{T_k}$ be either a group or a single trade that was just added to the block. Let $\tilde{T_l} \supset \tilde{T_k} \in \widetilde{\mathcal{T}}$ be a sandwich group that contains all of the trades in $\tilde{T_k}$. Let L_k, L_l be the number of trades in $\tilde{T_k}, \tilde{T_l}$ respectively. Note that this equals 1 for a single trade and $|g_k^{AB}| + 2$ for a group of trades. The new average profit-per-trade of $\tilde{T_l}$ is

$$p'(\tilde{T_l}, a_0) = \frac{P(\tilde{T_l}, a_0) + \sum_{i=1}^{L_l} t_i - P(\tilde{T_k}, a_0) - \sum_{j=1}^{L_k} t_j}{L_l - L_k} \tag{4.3}$$

where a_0 represents the amount used in the initial trade of the sandwich. Note that in the case that $\tilde{T_k}$ is a single trade, $L_k = 1$ and $P(\tilde{T_k}, a_0) = 0$. The derivation can be found in Appendix A.3.

4.4 Allocation Between Sandwiches

Let $\mathbb{A}, \mathbb{B}, \mathbb{C} \in \mathfrak{C}$ be arbitrary. Without loss of generality, let g_k^{AB} represent a group between token $\mathbb{A}$ and $\mathbb{B}$ on the block and let g_l^{AC} represent a group between token $\mathbb{A}$ and $\mathbb{C}$ not yet on the block. Let a_0 represent the amount of token $\mathbb{A}$ the arbitrageur has. The allocation to the second sandwich that maximizes profit is

$$a_1 = \underset{a_1}{\operatorname{argmax}}\ P(g_k^{AB}, a_0 - a_1) + P(g_l^{AC}, a_1). \tag{4.4}$$

The derivation can be found in Appendix A.2.

In the case that more than one sandwich starting from a given coin has been added to the block, we must solve for the allocation between all the sandwiches. As shown in B.1, equation (4.1) is not linear with respect to a_0, so we cannot just keep the a_1 solved for above and repeat the process for the next sandwich. They must all be recalculated and optimized together.

Let $\{g_{k_i}^{Ac_i^b}\}$ be an arbitrary set of groups of trades all from token $\mathbb{A}$ to some other token so that the c_i^b are all unique. Let $n = |\{g_{k_i}^{Ac_i^b}\}|$. Let a_0 be the amount of token $\mathbb{A}$ the arbitrageur has. The optimal allocation between the sandwich opportunities is

$$\vec{a} = \underset{\vec{a}}{\operatorname{argmax}}\ \sum_{i=1}^{n} P\Big(g_{k_i}^{Ac_i^b}, a_0 - \sum_{j=i}^{n-1} a_j\Big) \tag{4.5}$$

where $|\vec{a}| = n - 1$.

4.5 Adjusting Average Profit-per-trade for Second Sandwiches

Assume g_k^{AB} was just added to the block. We must adjust all $g_i^{Ac_i^b}$ for all i and $c_i^b \neq \mathbb{B}$. Let g_l^{AC} be one of those groups. Let a_1 be the optimal allocation between groups. The average

profit-per-trade of g_l^{AC} needs to be updated to

$$p'(g_l^{AC}, a_0, a_1) = \frac{P(g_l^{AC}, a_1) + \sum_{i=1}^{L} t_i - \Big(P(g_k^{AB}, a_0) - P(g_k^{AB}, a_0 - a_1)\Big)}{L+2}. \tag{4.6}$$

Chapter 5. Tests

In this chapter, I test my solution to the MEV problem with synthesized data. Since my solution relies on assumptions and is abstracted from the real world, the data I will be using will be random samples from distributions fitted to actual data. This allows me to do many tests with realistic samples.

Since there are over 24 thousand different cryptocurrencies and 625 different exchanges, a simulation of trades across all liquidity pools is nearly impossible [1]. Instead, I have focused on a few, highly liquid pools. This increases the chance for profitable sandwiches since there are more pending trades for each liquidity pool. This provides a better environment to test my solution since the effect of including sandwiches is magnified.

5.1 Data

All blockchain data was pulled from Etherium's blockchain through Etherscan [11]. The three tokens I considered were *ETH*, *BTC*, and *USDC*. These are the three most traded tokens on Etherium's blockchain. I pulled data from the pools with the highest liquidity between each of these tokens.

Once I had the data, I fit a log-normal distribution to the quantity of tokens sold (in USD) for each of the liquidity pools. I also fit a log-normal distribution to the tips associated with the trades. To generate the set of pending transactions, I chose two tokens at random, sampled the quantity from the associated distribution, and then sampled a tip.

5.2 Testing

I tested three strategies against each other. The first does not consider any arbitrage opportunities: it just maximizes the tips received from the block. The second strategy picks the most profitable sandwich opportunity and then fills in the rest of the block with the highest

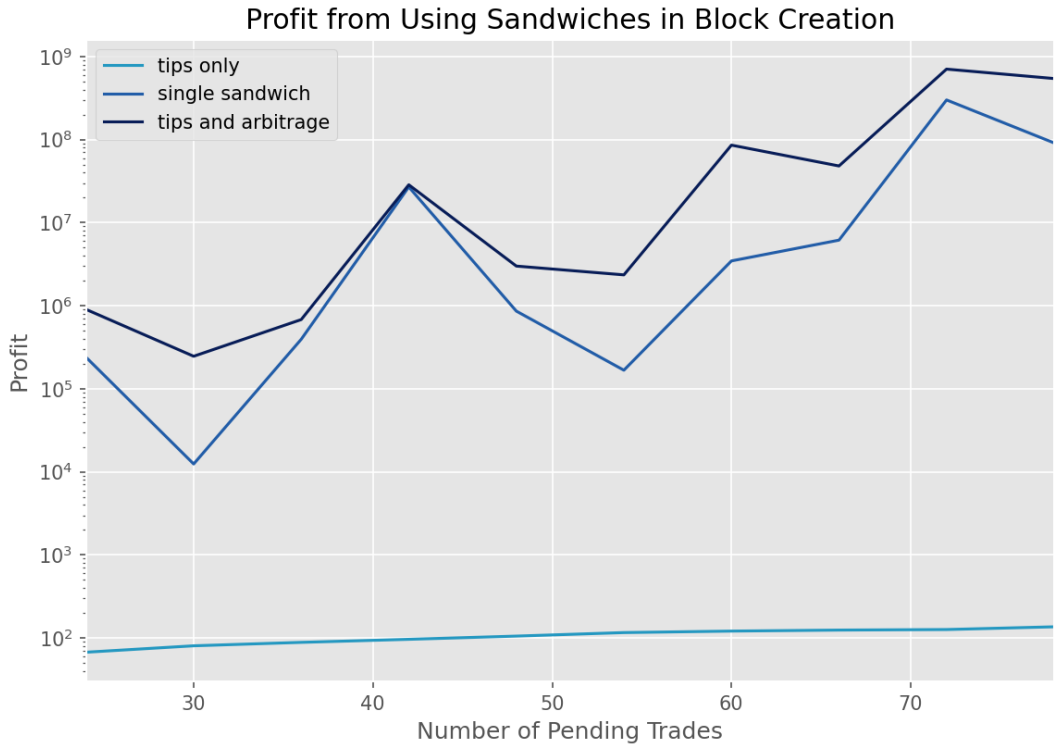

Figure 5.1: a plot of the median of 50 tests of the profit generated by three different block ordering strategies.

tip. The third strategy is my solution to the MEV problem and should result in the most profit.

I tested my strategy with different amounts of pending trades ($\mathcal{T}$). The more trades there are to pick from, the better sandwich opportunities there will be. I ran 50 tests for each of the 3 strategies for 10 different amounts of pending trades. Figure 5.1 plots the median profit for each of the strategies across the different amounts of pending trades.

5.3 Results

It is clear in figure 5.1 that utilizing sandwich arbitrage increases profit. As expected, my ordering generated the most profit followed closely by the single sandwich results. The results of the single sandwich strategy are so close to my MEV ordering because we are

only considering 3 tokens. If more tokens were considered but the number of pending trades remained constant, a single sandwich would take up less of the total block. To maximize profit, a smarter ordering of multiple sandwiches would be needed.

In the real world there are a lot more than three tokens being traded. Even though the number of pending transactions would be a lot larger than what I tested, the number traded in each pool takes up a smaller amount of the block than what I tested. These tests set up a scenario where my MEV strategy could be stress tested with large amounts of trades between each token. This tested my strategy's ability to adjust the average profit-per-trade and continue to pick the best trades when building a block.

Though my solution to the MEV problem relies on a simplified version of reality, my method and solution to the problem posed provides a big insight into solving the MEV problem. My solution can be applied and extended to any similar ordering problem, not only to the MEV problem in decentralized finance.

Appendix A. Derivations

A.1 Profit Function

Let $\mathbb{A}$ and $\mathbb{B}$ be two arbitrary tokens in $\mathfrak{C}$. Assume the arbitrageur is trying to profit from a sandwich between token $\mathbb{A}$ and $\mathbb{B}$ and has a_0 of token $\mathbb{A}$ for the initial trade. A sandwich is performed by selling a_0 of token $\mathbb{A}$ for token $\mathbb{B}$, ordering other trades from token $\mathbb{A}$ to token $\mathbb{B}$, then selling the token $\mathbb{B}$ received from the initial trade back to token $\mathbb{A}$.

Assume there is a total of N token $\mathbb{A}$ being traded for token $\mathbb{B}$ in the trades being sandwiched by the arbitrageur. Assume the liquidity pool starts with A_0^B of token $\mathbb{A}$ and B_0^A of token $\mathbb{B}$ with $A_0^B B_0^A = C$, some constant. If the arbitrageur puts a_0 of token $\mathbb{A}$ into the liquidity pool, we now have

$$A_1^B = A_0^B + a_0$$
$$B_1^A = \frac{A_0^B B_0^A}{A_0^B + a_0} = B_0^A - b$$

where b is the number of token $\mathbb{B}$ the arbitrageur receives for their initial trade. Solving for b gives us

$$b = B_0^A - \frac{A_0^B B_0^A}{A_0^B + a_0}.$$

By then ordering trades to sell N more of token $\mathbb{A}$ for token $\mathbb{B}$, the liquidity pool changes so we have

$$A_{N+1}^B = A_0^B + a_0 + N$$
$$B_{N+1}^A = \frac{A_0^B B_0^A}{A_0^B + a_0 + N}.$$

The arbitrageur then trades their b of token $\mathbb{B}$ for token $\mathbb{A}$. The number of each token in the pool is now

$$B_{N+2}^A = \frac{A_0^B B_0^A}{A_0^B + a_0 + N} + b$$
$$A_{N+2}^B = \frac{A_0^B B_0^A}{\frac{A_0^B B_0^A}{A_0^B + a_0 + N} + b} = A_0^B + a_0 + N - a_f$$

where a_f is the total number of token $\mathbb{A}$ the arbitrageur receives for their b token $\mathbb{B}$. Solving for a_f gives us

$$\begin{aligned} a_f &= A_0^B + a_0 + N - \frac{A_0^B B_0^A}{\frac{A_0^B B_0^A}{A_0^B + a_0 + N} + b} \\ &= A_0^B + a_0 + N - \frac{A_0^B B_0^A}{\frac{A_0^B B_0^A}{A_0^B + a_0 + N} + B_0^A - \frac{A_0^B B_0^A}{A_0^B + a_0}} \end{aligned}$$

which tells us that the profit the arbitrageur receives is

$$\begin{aligned} a_f - a_0 &= A_0^B + a_0 + N - \frac{A_0^B B_0^A}{\frac{A_0^B B_0^A}{A_0^B + a_0 + N} + B_0^A - \frac{A_0^B B_0^A}{A_0^B + a_0}} - a_0 \\ &= A_0^B + N - \frac{A_0^B B_0^A}{\frac{A_0^B B_0^A}{A_0^B + a_0 + N} + B_0^A - \frac{A_0^B B_0^A}{A_0^B + a_0}} \\ &= \frac{a_0 N (a_0 + 2A_0^B + N)}{a_0^2 + 2a_0 A_0^B + a_0 N + (A_0^B)^2} \end{aligned} \tag{A.1}$$

which holds for an arbitrary group of trades.

A.2 Allocation Between Sandwiches

In order to perform a sandwich between token $\mathbb{A}$ and $\mathbb{B}$, the arbitrageur needs some quantity of token $\mathbb{A}$. We called this a_0. Since equation (4.1) is monotonically increasing with respect to a_0 (see Appendix B.1), the arbitrageur should allocate all a_0 of their token $\mathbb{A}$ in order to maximize profits. This changes as soon as we include the opportunity to perform a second sandwich, this one between token $\mathbb{A}$ and $\mathbb{C}$.

Note that equation (4.1) is concave with respect to a_0. This means that each additional token the arbitrageur puts towards a sandwich gets less additional return than the previous one. Therefore, if there are two potential sandwiches (token $\mathbb{A}$ to $\mathbb{B}$ and token $\mathbb{A}$ to $\mathbb{C}$), and assuming the arbitrageur only has a certain amount of token $\mathbb{A}$ to allocate toward a sandwich, there is an optimal allocation between the two sandwiches that maximizes profits.

Let $g_{k_1}^{AB}$ and $g_{k_1}^{AC}$ be two different sets of trades. These are each a different sandwich opportunity: one between token $\mathbb{A}$ and $\mathbb{B}$ and the other between token $\mathbb{A}$ and $\mathbb{C}$. The

average profit-per-trade (4.2) if the arbitrageur allocated a_0 to each sandwich would be

$$p(g_{k_1}^{AB}, a_0) = \frac{P(g_{k_1}^{AB}, a_0) + \sum_{i=1}^{|g_{k_1}^{AB}|} t_i}{|g_{k_1}^{AB}| + 2} = \frac{P(g_{k_1}^{AB}, a_0) + \sum_{i=1}^{L_1} t_i}{L_1 + 2}$$

$$p(g_{k_1}^{AC}, a_0) = \frac{P(g_{k_2}^{AC}, a_0) + \sum_{i=1}^{|g_{k_2}^{AC}|} t_i}{|g_{k_2}^{AC}| + 2} = \frac{P(g_{k_2}^{AC}, a_0) + \sum_{i=1}^{L_2} t_i}{L_2 + 2}$$

for each group. Since the arbitrageur only has a_0 of token $\mathbb{A}$, they cannot allocate a_0 to both sandwiches. Thus we wish to find $a_1 \in [0, a_0]$ such that it maximizes the average profit-per-trade between the two sandwiches. Thus we set

$$\begin{aligned} a_1 &= \underset{a_1}{\text{argmax}} \frac{P(g_{k_1}^{AB}, a_0 - a_1) + \sum_{i=1}^{L_1} t_i + P(g_{k_2}^{AC}, a_1) + \sum_{i=1}^{L_2} t_i}{L_1 + 2 + L_2 + 2} \\ &= \underset{a_1}{\text{argmax}}\ P(g_{k_1}^{AB}, a_0 - a_1) + P(g_{k_2}^{AC}, a_1). \end{aligned}$$

This allocation maximizes the average profit-per-trade between two sandwiches.

A.3 Adjusting the Average Profit-per-trade of Supersets

Recall that the average profit-per-trade of a group of coins is the total profit generated by the group divided by the total number of trades needed to execute the sandwich. Since many of the groups contain the same trades, including one group or trade on the block changes the average profit-per-trade of other groups off the block. This is because both the additional profit the group generates and the additional space the group takes up on the block change.

If we add a single trade to the block, we must remove the tip associated with that trade from the average profit-per-trade of every group (not on the block) that included that trade (there shouldn't be any groups on the block that included that trade). We also need to adjust the average profit-per-trade of those groups up since the additional "weight" it has on the block (meaning space it takes up on the block) is one less than before. Thus, the

average profit-per-trade for every group that included that trade changes to

$$p'(g_l^{AB}, a_0) = \frac{P\big(g_l^{AB}, a_0\big) + \sum_{i=1}^{|g_l^{AB}|} t_i - t_j}{|g_l^{AB}| + 1} = \big(p(g_l^{AB}, a_0) - t_j\big)\frac{|g_l^{AB}| + 2}{|g_l^{AB}| + 1}$$

where g_l^{AB} is the group whose average profit-per-trade is being adjusted and t_j is the tip associated with the trade that was added.

If instead of a single trade we add a group to the block, we must adjust the average profit-per-trade of every group (not on the block) that is a superset of the group we added (there shouldn't be any supersets on the block). We must lower the arbitrage profit to just the increase in arbitrage profit, we can only include tips that aren't part of the added group, and the weight associated with the block is now the number of additional trades the new group would add. The average profit-per-trade for every group that was a superset of the group changes to

$$p'(g_l^{AB}, a_0) = \frac{P(g_l^{AB}, a_0) - P(g_k^{AB}, a_0) + \sum_{i=1}^{|g_l^{AB}|} t_i - \sum_{j=1}^{|g_k^{AB}|} t_j}{|g_l^{AB}| - |g_k^{AB}|}$$

where g_l^{AB} is the group being adjusted and g_k^{AB} is the group that was added.

We can combine both of these into one formula. Let $\tilde{T_k}, \tilde{T_l} \in \widetilde{\mathcal{T}}$ be either groups or single trades both from token $\mathbb{A}$ to $\mathbb{B}$. Assume $\tilde{T_k}$ was just added to the block and $\tilde{T_l}$ is off the block and needs to be adjusted. Let L_k, L_l be the number of trades in $\tilde{T_k}, \tilde{T_l}$ respectively. Note that this is 1 for a single trade and $|g_k^{AB}| + 2$ for a group of trades. The new average profit-per-trade of $\tilde{T_k}$ is

$$p'(\tilde{T_l}, a_0) = \frac{P(\tilde{T_l}, a_0) + \sum_{i=1}^{L_l} t_i - P(\tilde{T_k}, a_0) - \sum_{j=1}^{L_k} t_j}{L_l - L_k}. \tag{A.2}$$

Note that in the case that $\tilde{T_k}$ is a single trade, $L_k = 1$ and $P(\tilde{T_k}, a_0) = 0$.

Appendix B. Profit Function Characteristics

B.1 Near-linear with respect to a_0

It is clear that equation (4.1) is not linear in a_0. With sufficiently large A_0^B relative to a_0, however, it gives us near-linear behavior. This can be seen both mathematically and graphically.

B.1.1 Graphical visualization. Figure B.1 is a plot of the profit generated from the arbitrage (P) with respect to the number of tokens used in the arbitrageurs initial trade (a_0). Figure B.2 is the return (P/a_0) on the arbitrageurs initial trade (a_0). The different lines represent larger or smaller values of A_0^B with the darker lines corresponding to the larger values. We can see that the larger A_0^B is relative to a_0, the more linear equation (4.1) becomes.

B.1.2 Mathematical proof. Note that

$$\begin{aligned}\lim_{a_0\to\infty}\frac{\partial^2 P}{\partial a_0^2} &= \lim_{a_0\to\infty}\frac{-2\big(A_0^B\big)^2\Big(3a_0^2+3a_0\big(2A_0^B+N\big)+3\big(A_0^B\big)^2+4A_0^BN+N^2\Big)}{\Big(a_0^2+a_0\big(2A_0^B+N\big)+\big(A_0^B\big)^2\Big)^3}\\ &= \lim_{a_0\to\infty}\frac{-6\big(A_0^B\big)^2a_0^2+\mathcal{O}\big(a_0\big)}{a_0^6+\mathcal{O}\big(a_0^5\big)}\\ &= 0.\end{aligned}$$

Thus equation (4.1) gets more and more linear as a_0 increases.

B.2 Near-linear with respect to N

It is clear that equation (4.1) is not linear in N. With sufficiently large A_0^B relative to N, however, it gives us near-linear behavior. This can be seen both mathematically and graphically.

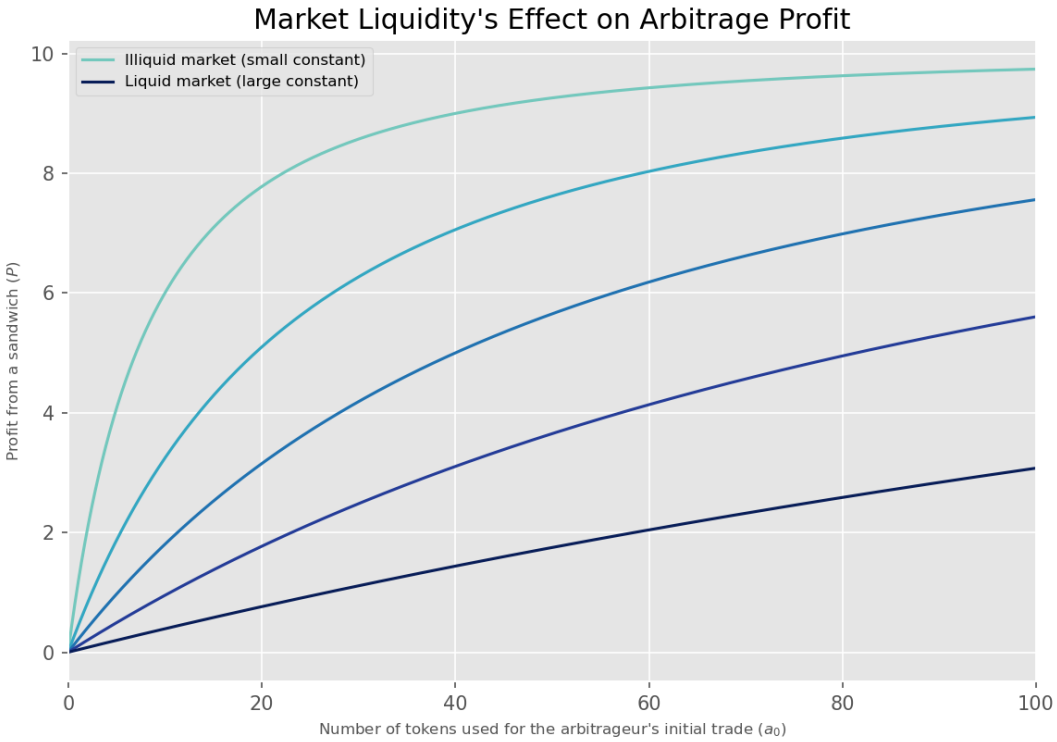

Figure B.1: the effect the liquidity pool's size relative to the arbitrageur's initial trade has on the profit of a sandwich.

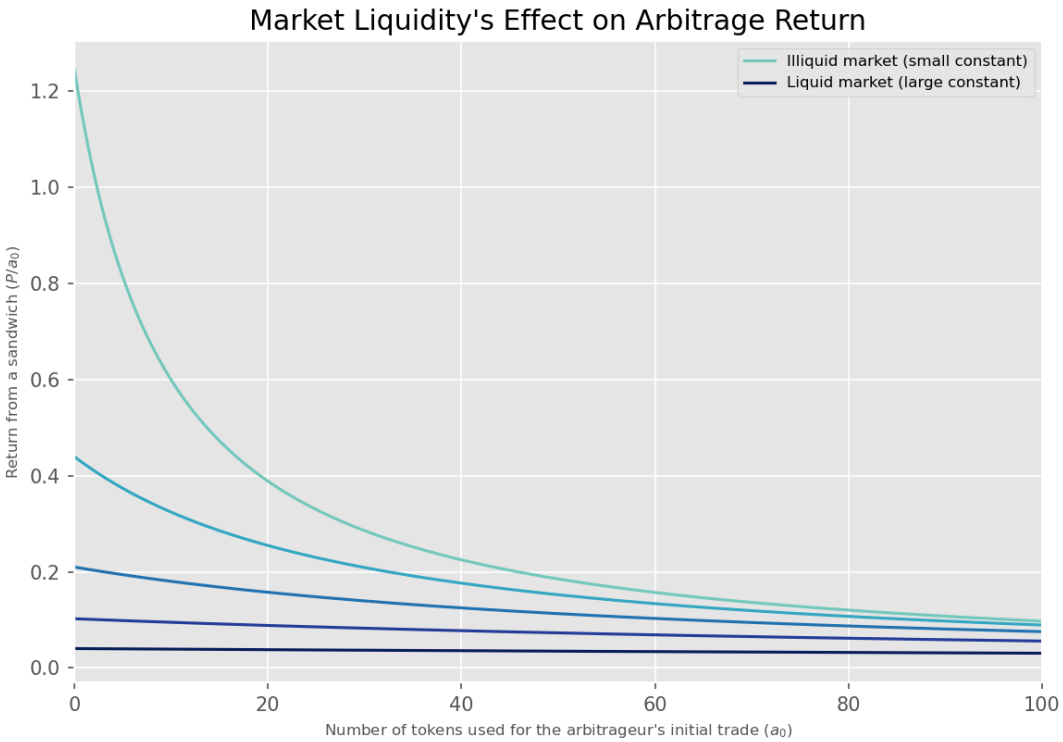

Figure B.2: the effect the liquidity pool's size relative to the arbitrageur's initial trade has on the return of a sandwich.

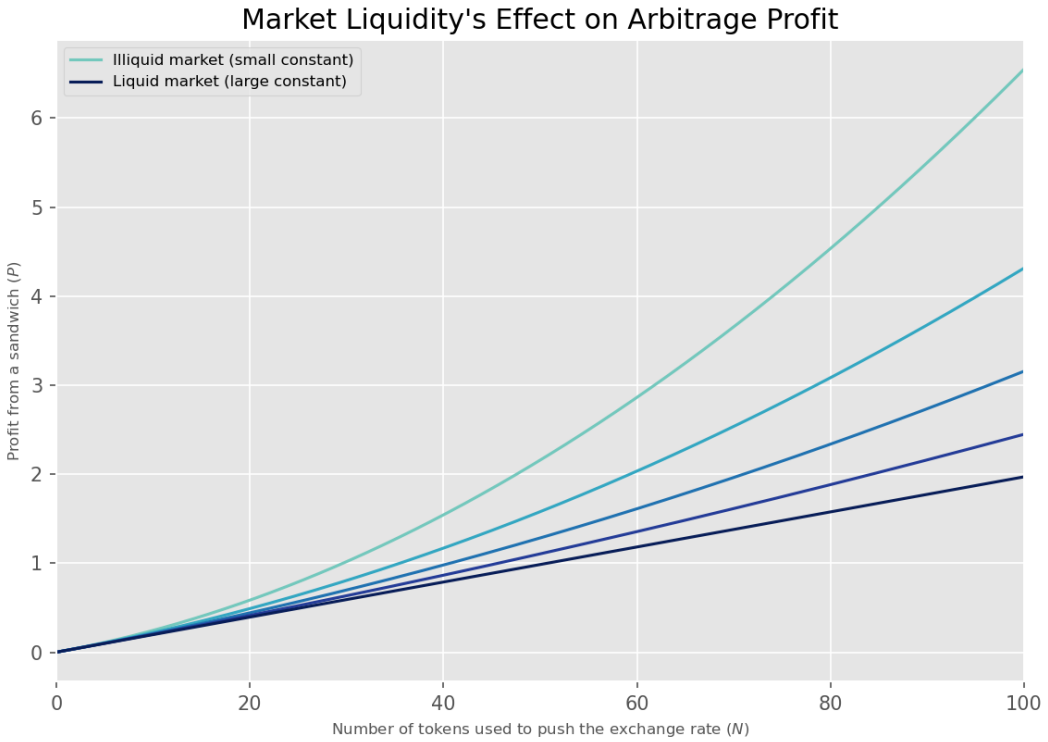

Figure B.3: the effect the liquidity pool's size relative to the number of tokens used to move the exchange rate has on the profit of a sandwich.

B.2.1 Graphical visualization. Figure B.3 is a plot of the profit generated from the arbitrage (P) with respect to the number of tokens used to change the exchange rate (N). The different lines represent larger or smaller values of A_0^B with the darker lines corresponding to the larger values. We can see that the larger A_0^B is relative to N, the more linear equation (4.1) becomes.

B.2.2 Mathematical proof. Note that

$$\begin{aligned}
\lim_{N\to\infty} \frac{\partial^2 P}{\partial N^2} &= \lim_{N\to\infty} \frac{-2a_0^2\left(A_0^B\right)^2}{\left(a_0^2 + a_0\left(2A_0^B + N\right) + \left(A_0^B\right)^2\right)^3} \\
&= \lim_{N\to\infty} \frac{-2\left(A_0^B\right)^2 a_0^2}{a_0^6 + \mathcal{O}\left(a_0^5\right)} \\
&= 0.
\end{aligned}$$

Thus equation (4.1) gets more and more linear as N increases.

B.3 Monotonically increasing with respect to N

Note that

$$\frac{\partial P}{\partial N} = \frac{a_0\big(A_0^B\big)^2}{\Big(a_0^2 + a_0\big(2A_0^B + N\big) + \big(A_0^B\big)^2\Big)^2} > 0$$

since $a_0 > 0$. Thus the profit function (4.1) is monotonically increasing with respect to N.

B.4 Monotonically increasing with respect to a_0

Note that

$$\frac{\partial P}{\partial a_0} = \frac{\big(A_0^B\big)^2\big(2a_0 + 2A_0^B + N\big)}{\Big(a_0^2 + a_0\big(2A_0^B + N\big) + \big(A_0^B\big)^2\Big)^2} > 0$$

since $a_0, N, A_0^B > 0$. Thus the profit function (4.1) is monotonically increasing with respect to a_0.

B.5 Concave with respect to a_0

Note that

$$\frac{\partial^2 P}{\partial a_0^2} = \frac{-2\big(A_0^B\big)^2\Big(3a_0^2 + 3a_0\big(2A_0^B + N\big) + 3\big(A_0^B\big)^2 + 4A_0^B N + N^2\Big)}{\Big(a_0^2 + a_0\big(2A_0^B + N\big) + \big(A_0^B\big)^2\Big)^3} < 0$$

since $a_0, N, A_0^B > 0$. Thus the profit function (4.1) is concave with respect to a_0.

Bibliography

[1] CoinMarketCap today's cryptocurrency prices by market cap. https://coinmarketcap.com/. Accessed: 2023-05-16.

[2] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges, 2019.

[3] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest?, 2021.

[4] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain, 2019.

[5] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges, 2020.

[6] Ben Weintraub, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State. A flash(bot) in the pan. In *Proceedings of the 22nd ACM Internet Measurement Conference*. ACM, oct 2022.

[7] Carsten Baum, Bernardo David, and Tore Kasper Frederiksen. P2dex: Privacy-preserving decentralized cryptocurrency exchange. In *Applied Cryptography and Network Security: 19th International Conference, ACNS 2021, Kamakura, Japan, June 21–24, 2021, Proceedings, Part I*, page 163–194, Berlin, Heidelberg, 2021. Springer-Verlag.

[8] Michele Ciampi, Muhammad Ishaq, Malik Magdon-Ismail, Rafail Ostrovsky, and Vassilis Zikas. Fairmm: A fast and frontrunning-resistant crypto market-maker. Cryptology ePrint Archive, Paper 2021/609, 2021. https://eprint.iacr.org/2021/609.

[9] Lioba Heimbach and Roger Wattenhofer. Eliminating sandwich attacks with the help of game theory. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. ACM, may 2022.

[10] Massimo Bartoletti, James Hsin yu Chiang, and Alberto Lluch-Lafuente. Maximizing extractable value from automated market makers, 2022.

[11] Etherscan. https://etherscan.io/. Accessed: 2023-05-23.