Tampere University of Applied Sciences

Graph-Based Circular and Exchange Arbitrage Detection

Eric Brown

BACHELOR'S THESIS
April 2023

Degree Programme in Software Engineering

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Software Engineering

ERIC BROWN:
Graph-Based Circular and Exchange Arbitrage Detection

Bachelor's thesis 53 pages, appendices 7 pages
April 2023

Although (circular) arbitrage is already a well-known concept in the trading of crypto-currency pairs on decentralized exchanges (DEXes) like Uniswap, this thesis explored the idea of combining circular arbitrage with exchange arbitrage. The strategy was to leverage graph-theoretic methods on multiple DEXes and centralized exchanges (CEXes) that are aggregated and sorted by their offered exchange rates for each crypto trade pair. By doing so, the aim is to find a circular trade where each trade occurs on the DEX/CEX with the best rate, allowing for more profitable opportunities than being confined to one DEX for all trades.

This thesis not only explores the combination of circular arbitrage with exchange arbitrage but also focuses on the creation of a program that implements these strategies. Then the methodology used to develop the program and the arbitrage results obtained from its implementation are presented. In addition, this thesis explores other ways to optimize profitable arbitrage opportunities and improve the program implementing this strategy. Examples include improving the speed and performance of the program, as well as maintaining profitable arbitrage even after accounting for the high gas fees required for each trade in the circular trade by factoring the ideal trade volume. Furthermore, other programmatic improvements are considered, such as changing programming languages and developing smart contracts with Solidity specifically for atomic transactions.

Key words: circular arbitrage, exchange arbitrage, Ethereum, graph-theory, depth-first search

CONTENTS

ABBREVIATIONS AND TERMS

DEX	decentralized exchange
CEX	centralized exchange
DeFi	decentralized finance
API	application programming interface
AMM	automated market maker
LOM	limit order market
DFS	depth-first search
BFS	breadth-first search
ABI	application binary interface
tx	transaction
JSON	JavaScript object notation
dApp	decentralized applications
MetaMask	A wallet to interact with Ethereum-based dApss and send transactions on the Ethereum network.
mainnet	The official Ethereum network where txs take place.
testnet	A parallel Ethereum network used for testing without spending real funds.

1 INTRODUCTION

Many software engineers and cryptocurrency traders create or implement a trading bot as a means of generating passive income. These bots range in levels of complexity and have different techniques and algorithms used to find profitable trades within the exchange they are built for. With multiple decentralized exchanges (DEXes) and centralized exchanges (CEXes) in cryptocurrency markets, it is possible to perform both circular and exchange arbitrage with a synchronous path of trades to make a profit. To accomplish this programmatically, the two methods of arbitrage need to be combined in the program to allow for finding a circular trade, where each crypto currency is exchanged on the platform with the best exchange rate.

The goal for the circular arbitrage implementation is to find profit using a graph-theoretic algorithm. Although graph-theoretic algorithms such as Bellman-Ford are commonly used for finding most profitable circular paths in a graph, of currencies as nodes and rates of exchange as weighted edges, the program implements a Depth-first search (DFS) method due to limitations applied from the method used for finding exchange arbitrage. Other works such as GitHub user ccyanxyz's repository "uniswap-arbitrage-analysis" validates profitable circular arbitrage is possible with a Python implementation of DFS on only the Uniswap DEX (ccyanxyz 2022); this related work will be explored later.

To fit exchange arbitrage into a program firstly focused on finding circular trades to execute for a profit, the exchange arbitrage has to occur at every trade the DFS algorithm is exploring, comparing all DEXes and CEXes for the best exchange rate. Fortunately, this can be relatively easy to implement using an exchange aggregation service such as SwapSpace. Integrating their API and data into the program allows for a more convenient method of finding the exchange platform offering the best exchange rate for each trade.

The purpose of this thesis is to develop a program able to find combined circular and exchange arbitrage; then examine the possible new opportunities for leveraging a two-dimensional price discrepancy. The program developed for

this thesis will attempt to find more profitable opportunities by not limiting the circular arbitrage to just one DEX, but allow each leg of the circular trade to occur on the exchange with the best rate. Therefore, producing an automated crypto trading bot capable of making a profit.

2 BACKGROUND

2.1 Arbitrage

Wherever there are open markets, there are arbitrage opportunities; and this holds true for the current growth in DEXes for trading crypto-currencies. Decentralized exchanges like Uniswap use an automated market maker (AMM) model to provide liquidity. In an AMM, agents supply and demand liquidity by adding or removing assets in proportion to the existing pool, with the price determined by a predetermined convex relationship. This model contrasts with centralized exchanges (CEXes) that use a limit order book, where liquidity suppliers compete with each other and can strategically post prices (Lehar, A. & Parlour, C. 2021). Arbitrage, as defined by Merriam-Webster (2023), is the practice of buying and selling assets in different markets to capitalize on price disparities. In the context of the cryptocurrency market, this can be achieved through the two methods of exchange arbitrage and circular arbitrage. While both methods offer the potential for profit, they also present unique challenges that will be explored in this thesis.

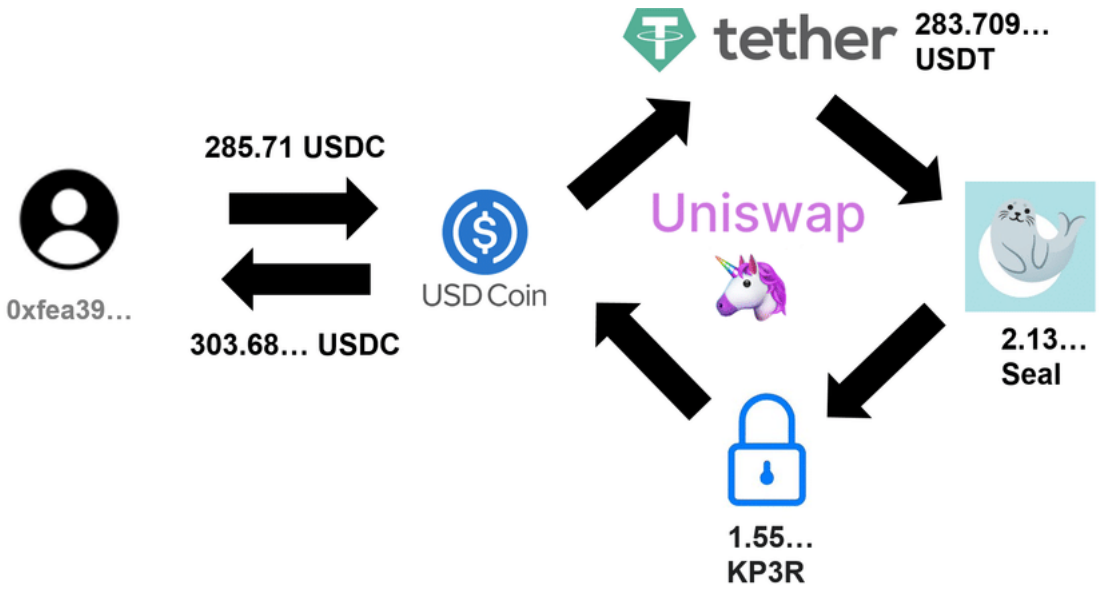


FIGURE 1. “An example [of circular] arbitrage on Uniswap V2: The trader traded 285.71 USDC through four different exchange markets across USDC, USDT, Seal, Kp3r, and finally received 303.68 USDC. The revenue of this cyclic arbitrage is 17.97 USDC (18 USD)” (Wang, Ye, et al. 2022, 2).

Circular (cyclic) arbitrage is a trading strategy that involves profiting from price discrepancies between multiple currencies in a circular path. In other terms, it is a method of trading, in which a trader takes advantage of price disparities between two or more currencies in a sequence of trades that returns to the initial currency, producing a net profit. On both DEXes and CEXes, it is possible to find profit with circular arbitrage, such as in Uniwap as shown in Figure 1. This shows a trader, identified by a wallet address, transfers USD into USDC on Uniswap, to then trade multiple cryptocurrencies, and then withdraw the USD and ultimately make a profit. These trades all happen instantaneously or at least immediately one after another when using a program. For programming this type of arbitrage, this requires some graph-theoretical algorithm, this will be expanded upon later.

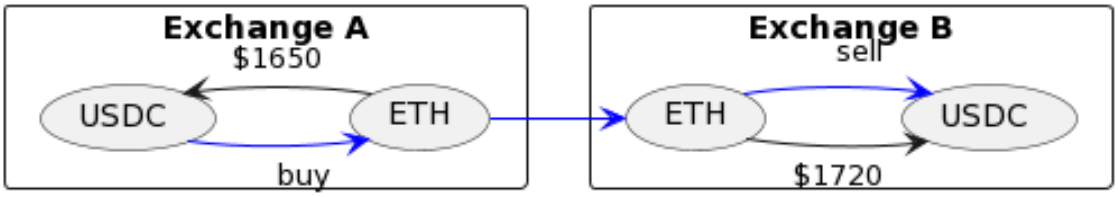


FIGURE 2. Simple exchange arbitrage example between buying then selling ETH on two different decentralized exchanges to profit off the price discrepancy.

Exchange arbitrage is another trading strategy which involves buying and selling the same asset, such as a cryptocurrency, on two or more different exchanges to take advantage of price discrepancies between the exchanges and generate profits. For example, in Figure 2, if the price of Ethereum (ETH) is $1,650 on Exchange A and $1,720 on Exchange B, a trader can buy Ethereum on Exchange A for $1,650 and sell it on Exchange B for $1,720, resulting in a profit of $70, or a profit of 4.24%. This strategy works by exploiting the inefficiencies, desynchronization, and time lags between different markets of high liquidable assets, allowing traders to profit from price discrepancies.

Circular arbitrage is an already well-known concept in the field of crypto trading bots, and is a crucial part of the current DEX ecosystem for balancing tokens trading pair price (Lo, Y. & Medda, F. 2021, 20); competition is saturated with

trading bots finding these imbalanced trading pair prices. Wang, Y. et al. found “traders have executed 292,606 cyclic arbitrages [on Uniswap] over eleven months and exploited more than 138 million USD in revenue” (Wang, Y. et al. 2022, 1). Furthermore, DEXes can experience desynchronization in token trading pair price across platforms, creating price disparities where “multiple DEXes must synchronize through [exchange] arbitrage“ (Zhou, L., Qin, K. & Gervais, A. 2021, 2). This thesis and implemented program demonstrate novel profitable arbitrage opportunities emerge when combining circular arbitrage with exchange arbitrage to find price disparities between tokens trading pair price and disparities of a tokens value across exchange platforms.

2.2 Ethereum

Ethereum (ETH) is the native cryptocurrency of the Ethereum blockchain, a decentralized public ledger for peer-to-peer transactions, which is used as a means of payment for transactions, fees, and as a store of value. It has its own blockchain network and is not reliant on any other tokens or smart contracts to function. Minors assemble the transactions into blocks and chain them together using a cryptographic hash (Wood, G. 2014). Each transaction is associated with an operation that modifies the state of corresponding accounts. A transaction can be verified as successful once it has been mined onto the blockchain, and as a public ledger, anyone can view the history of all transactions. There are many blockchains with their own crypto currency/currencies, however this thesis work will only explore arbitrage opportunities on the Ethereum blockchain.

2.2.1 Gas Fees

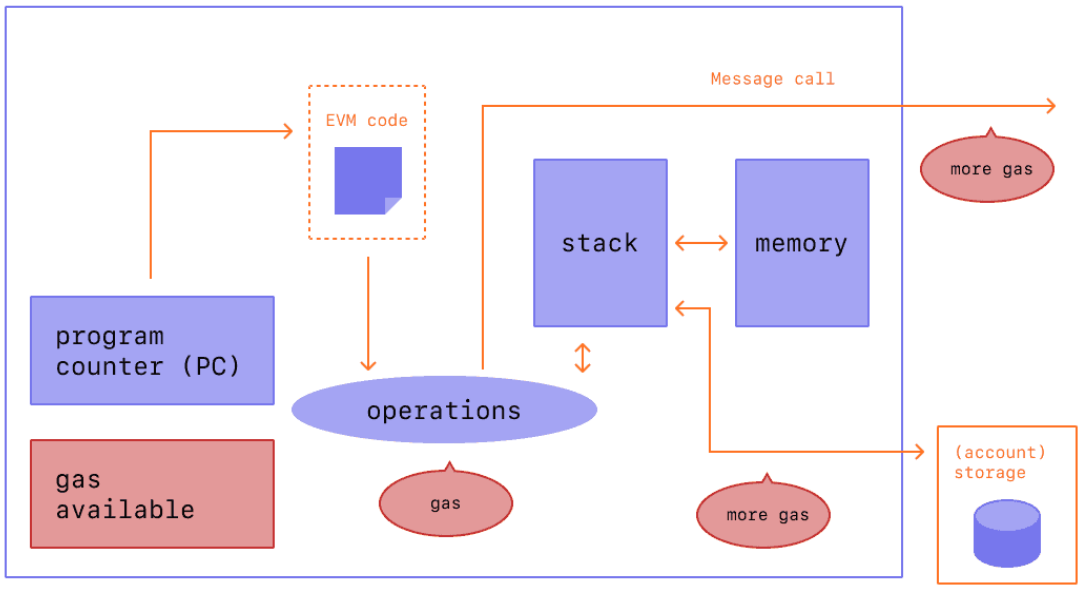


FIGURE 3. Illustration that any computational effort (blue) requires a proportional amount of gas (red) to be paid to perform the task in the transaction (Smith, C. 2023).

$$gas\ fee = gas\ price * gas\ consumption \quad (1)$$

In Ethereum, gas fees are similar to trading fees in that every transaction requires a fee to cover the cost of executing the transaction, however gas fees are more dynamic given its the product of gas consumption and gas price (1). The gas consumption of executing a transaction is proportional to the amount of computing resources used to complete the transaction and any functions associated with it, as illustrated in Figure 3. Therefore, the more complex the transaction, the more gas that will need to be consumed for completion. The gas price is chosen by transaction issuers who must factor the supply and demand between the amount of issued transactions and the available computing resources (such as miners) that are able to process those transactions (Zarir, A. et al. 2021, 32). The gas fee paid by the user to the miner for executing the transaction is determined by the product of the gas price and gas consumption, and the miner will most likely choose the most profitable transaction to mine. Gas prices have been inflating due to the imbalance of supply and demand for the overwhelming amount of transactions thus “market

size of exploited cyclic arbitrage is significantly smaller than the exploitable opportunities because of – – the high gas fees" (Wang, Y et al. 2022, 2). However, there are rumors in articles, such as by journalist Marcel Deer at Cointelegraph, stating Ethereum 2.0 (Eth2) will be released in a couple years, and with the upgrade gas fees will drop dramatically (Deer, M. 2022). Until the Ethereum blockchain updates its method miners have to use for approving transactions, gas fees will continue to remain high; disrupting opportunities for profitable arbitrage.

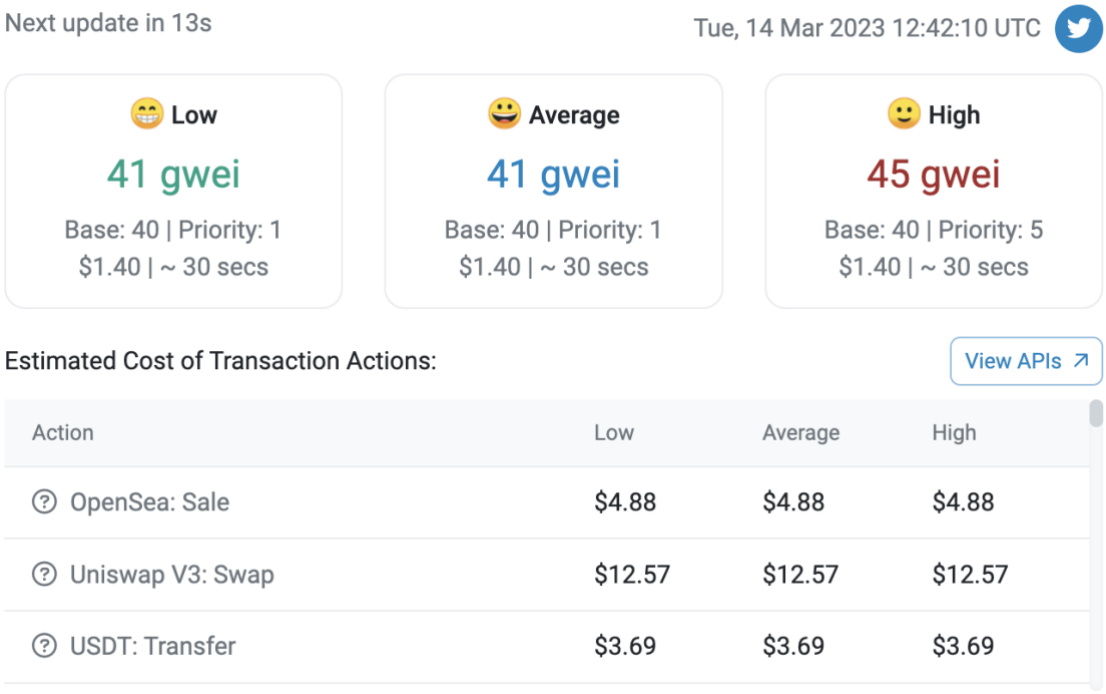


FIGURE 4. Timestamp of current gas fees on Ethereum network 12:42 UTC 14 March 2023 provided by Etherscan.io Gas Tracker.

Although gas prices are always fluctuating, they have been steadily rising as more people are sending transactions than before, and have to increase the gas price to get their transaction onto the blockchain first. For arbitrage in any crypto markets, trading in higher volume is always better for profitability to negate the cost of the gas fees. For example, Figure 4 is a snapshot of gas fees on the Ethereum network at that moment in 14 March 2023, if a profitable circular arbitrage opportunity was found given 4 crypto currencies, including the starting currency, 4 trades would have to occur. Therefore, the gas fee would have to be paid 4 times and at this moment, those 4 trades would result in a

minimum of 6 USD in estimated gas fees. If typical circular arbitrage opportunities have a profit of 1-3%, the trade volume would have to be large enough so the profit percentage can cover the gas fees, allowing the circular trade to still be profitable.

2.2.2 ERC20 Tokens

The ERC-20 standard stipulates that a smart contract must implement nine *functions*:

```
function name()
function symbol()
function decimals()
function totalSupply()
function balanceOf(address _owner)
function transfer(address _to, uint256 _value)
function transferFrom(address _from, address _to, uint256 _value)
function approve(address _spender, uint256 _value)
function allowance(address _owner, address _spender)
```

There are also two *events*:

```
event Transfer(address indexed _from, address indexed _to, uint256 _value)
event Approval(address indexed _owner, address indexed _spender, uint256 _value)
```

FIGURE 5. Required functions of a token smart contract following the ERC20 standard (Ethereum 2022).

Ethereum Request for Comment 20 (ERC20) tokens are tokens created and hosted on the Ethereum blockchain, which comply with the ERC20 token standard. They are fungible tokens interchangeable with each other; this exchange of Ethereum based tokens is where the circular arbitrage occurs. ERC20 is a set of rules and guidelines that a token must follow to be considered an ERC20 token; as shown in Figure 5, these are the required functions an ERC20 token must have available for its smart contract, according to the current documentation on GitHub. These tokens are programmable, meaning that developers can create smart contracts on the Ethereum network to control their distribution, use, and extended functionality. ERC20 tokens can be used for a variety of purposes, such as utility tokens to access services, security tokens to represent ownership in assets, or even as a means of payment (Ansari, K. & Kulkarn, U. 2020, 1). They rely on the Ethereum blockchain network to function and are typically traded on cryptocurrency exchanges. There are other types of ERC tokens i.e., ERC721 used for other cases such as NFTs.

The program resulted from this thesis work finds exchange and circular arbitrage using an Ethereum network Metamask crypto wallet to hold Ethereum blockchain funds used for the trades. Therefore, all trades must be made with tokens that are Ethereum or that are on the Ethereum network, that being the ERC20 tokens. For example, this Ethereum crypto (Metamask) wallet could not accept or hold Bitcoin (BTC), due to a plethora of reasons: different underlying technologies and protocols creating both blockchains, different cryptographic algorithms, and so on. In summary, ETH is the native cryptocurrency of the Ethereum blockchain, while ERC20 tokens are programmable tokens created on the Ethereum blockchain network that follow the ERC20 token standard.

2.2.3 Smart Contracts

While Ethereum as a currency can be executed as a simple transaction on its blockchain, any transaction involving ERC20 tokens or more complicated transfers requires a smart contract. Smart contracts “are computer programmes of arbitrary complexity which run persistently on the blockchain itself, where they will execute in an entirely trustless, decentralised and propably correct manner on a global network of independent validating computers“ (Cuffe, P. 2018, 1). Smart contracts typically just execute the “transfer” function required in the ERC20 token standard (Figure 5), sending funds from one wallet to another; however, they can be more complex. As noted in Ye Wang et al., traders commonly perform a “cyclic arbitrage within a single blockchain transaction. Such atomic implementations mitigate the financial loss of users in cyclic arbitrage” (Wang, Y et al. 2022, 2). Due to the volatile prices, executing each transaction independently sequentially might result in unexpected gains or loss from the estimated amount; or one of the trades might not have been successful, thus a failure in obtaining profit from a circular trade. Therefore, wrapping all transactions of a circular trade into one smart contract would mean all trades have to be successful, or else the smart contract will cancel and no trades will be completed.

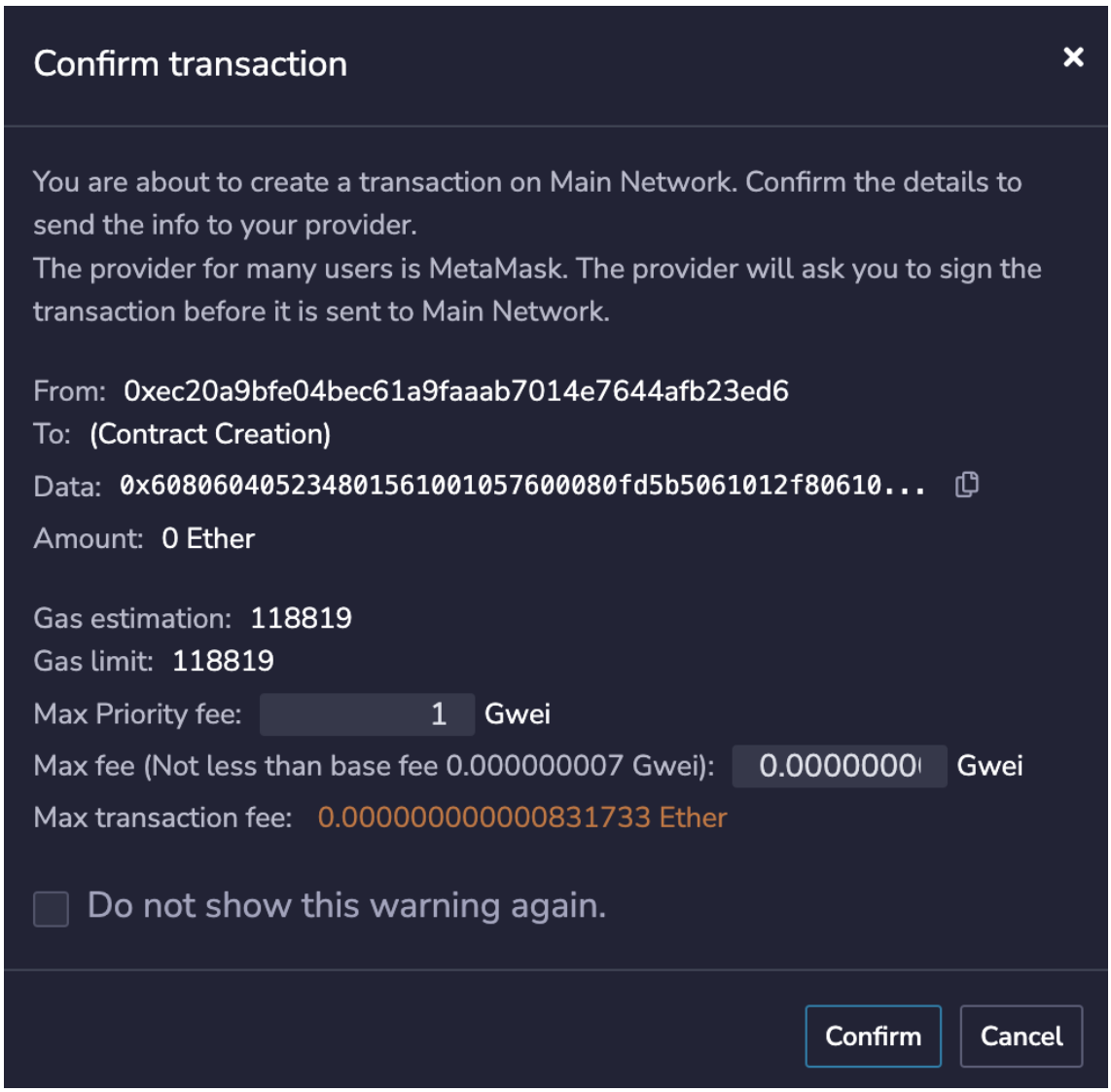


FIGURE 6. Remix web IDE alert before confirming the deployment of a smart contract on Ethereum mainnet.

Custom smart contracts can be made as well through creating contract code on solidity and compiling it for deployment. Figure 6 is an example of a deployment confirmation of a custom smart contract from Remix IDE, a web IDE platform for creating and deploying smart contracts, onto the Ethereum mainnet. Once a contract is deployed anyone can use that contract if they know the contract address. Each deployment creates a new instance of the contract with its own unique address on the blockchain. However, it is generally not recommended to deploy the same contract multiple times unless there is a specific need for it. As it can be more efficient and cost-effective to reuse an existing contract instance rather than to deploy a new one each time (Panda, S. & Satapathy, S. 2021, 554). Additionally, each deployment of a contract incurs a deployment (gas) fee,

which can be very expensive if done multiple times unnecessarily. Therefore, it is generally best to deploy the contract once and reuse the same instance for subsequent transactions that follow the same strategy.

2.3 SwapSpace

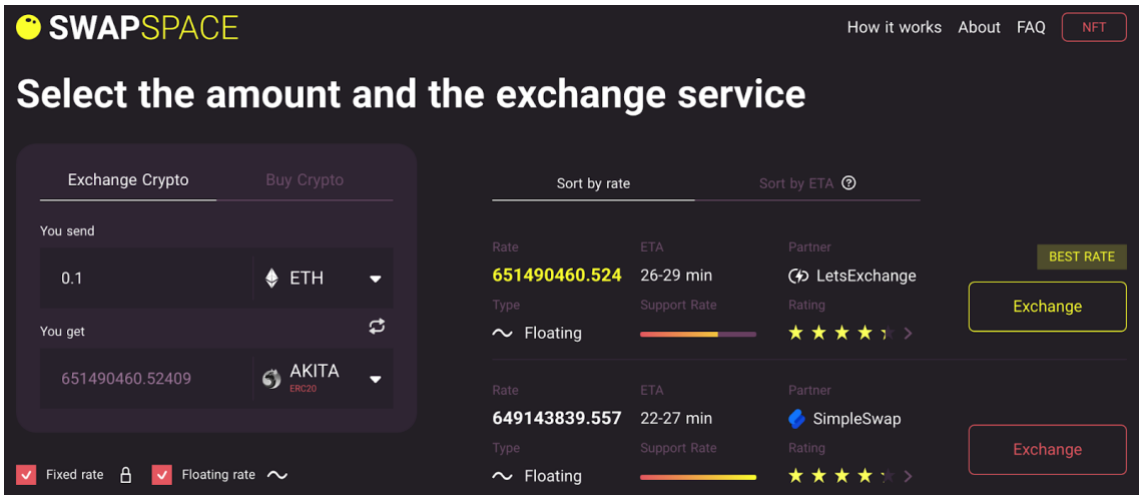


FIGURE 7. SwapSpace website aggregating ETH-AKITA exchange options through its API.

While the created program uses graph-theoretic methods for finding circular arbitrage of the tokens on the Ethereum network, it is the exchange aggregator service, SwapSpace, that is used for aggregating and sorting all available exchanges (Figure 7) for finding the best token trading pair price. SwapSpace is a non-custodial instant cryptocurrency exchange aggregator that offers over 2250 cryptocurrencies for exchange with fixed and floating rates, no registration or additional fees, and doesn't require personal data, ensuring security and transparency for users.

- Float rate: The quoted rate (“from amount” and “to amount”) between currencies can change during the exchange, therefore receiving more or less coins than expected.
- Fixed rate: The quoted rate (“from amount” and “to amount”) between currencies offered by the exchange will freeze the rate for approximately

15 to 120 minutes to allow time to transfer the base currency with the quoted rate guaranteed.

SwapSpace partners with “a large variety of reliable exchange services to provide customers with an opportunity to choose the best swap option” (SwapSpace 2023b), thereby capable of easily performing exchange arbitrage just with their API. Using this API in tandem with the steps required to make a functioning circular arbitrage program is simple relative to the power of leveraging both arbitrage strategies.

2.3.1 Structure of Fees and Service

Although SwapSpace profits from all trades it initiates through its website and API, it charges no additional fees to its customers. Rather, SwapSpace is “sharing the commissions with the [partnered] exchange providers instead” and claims paying through this service would never be more costly than paying “directly to the [partners] integrated services” (SwapSpace 2023b). Furthermore, using the API requires an API key obtained by joining as an affiliate. The advantage of executing trades through the created affiliate API key is that SwapSpace shares “50% of [the] revenue share.” Thus, any commission SwapSpace receives from the partnered exchange, half is shared with the affiliate, which for the thesis project is the author. Although the program does not attempt to factor affiliate revenue in profitable arbitrage, it is a small benefit that can be considered insurance for floating rates that can fluctuate.

2.3.2 API Endpoints

Functional details about the endpoints used and their implementation will be provided in the methods chapter, however a brief overview of SwapSpace API endpoint structure will be provided here. Overall, the integration flow from one endpoint to another is straightforward; as outlined from the SwapSpace documentation, the order and a brief description of each endpoint is given in Table 1. The partners endpoint is not used in the program created for this thesis

because any partner and rate type is allowed, therefore there is no reason to get the list of all available partners and filter them by what they provide. Additionally, the endpoint providing the list of estimated “amounts” does not give the implicit rate of exchange; instead as a parameter the base currency amount is given and this endpoint returns the destination currency amount. The array of available exchanges from all the partners and the fixed/float type is also returned, ultimately this array can be sorted by the destination currency amount to get the best returned amount. Once the exchange is created, the trade must be completed by sending the exact base currency amount to the provided wallet address given in the response from this API endpoint. Finally, the status can be checked by the exchange status endpoint requiring the ID of the created exchange.

TABLE 1. A brief summary of the SwapSpace API endpoints necessary for the program (SwapSpace 2023a).

API Endpoint	Response	Details
GET /v2/currencies	JSON object array of exchangeable currencies	Returns important information needed for estimated exchange “amounts" and create “exchange” endpoints
GET /v2/partners	JSON object array of partnered exchanges	Provides Booleans of whether each partner accepts fixed and/or float rates
GET /v2/amounts	JSON object array of available exchanges and their quoted amount returned in exchange	Does not give implicit exchange rate, rather the exact “to” and “from” amount for the exchange
POST /v2/exchange	Details about the created exchange	Initiates the transaction by creating the exchange though the transaction must still be sent to the SwapSpace wallet
GET /v2/exchange/{id}	Current status of created exchange (by id)	After sending funds to the SwapSpace wallet, it can take up to an hour to receive the swapped token.

2.3.3 API Call Limit

The API call limit SwapSpace enforces is fairly straightforward and generous when compared to the status quo; i.e., crypto CEXes such as Binance, Coinbase, etc.… As shown in Table 2, given a credit limit of 2500 per 5 minutes, with each API call costing only 10 credits except for the exchange creation call for swapping two currencies, it is unlikely that these limitations will pose a significant constraint to the program's operations.

TABLE 2. Summarizes the SwapSpace API call limit rules, showing credits per action and credits per every 5 minutes (SwapSpace 2023a).

Action	Credits per Action	Credits Limit (Every 5 minutes)
Creating an exchange (POST)	400	2500
Other requests	10	

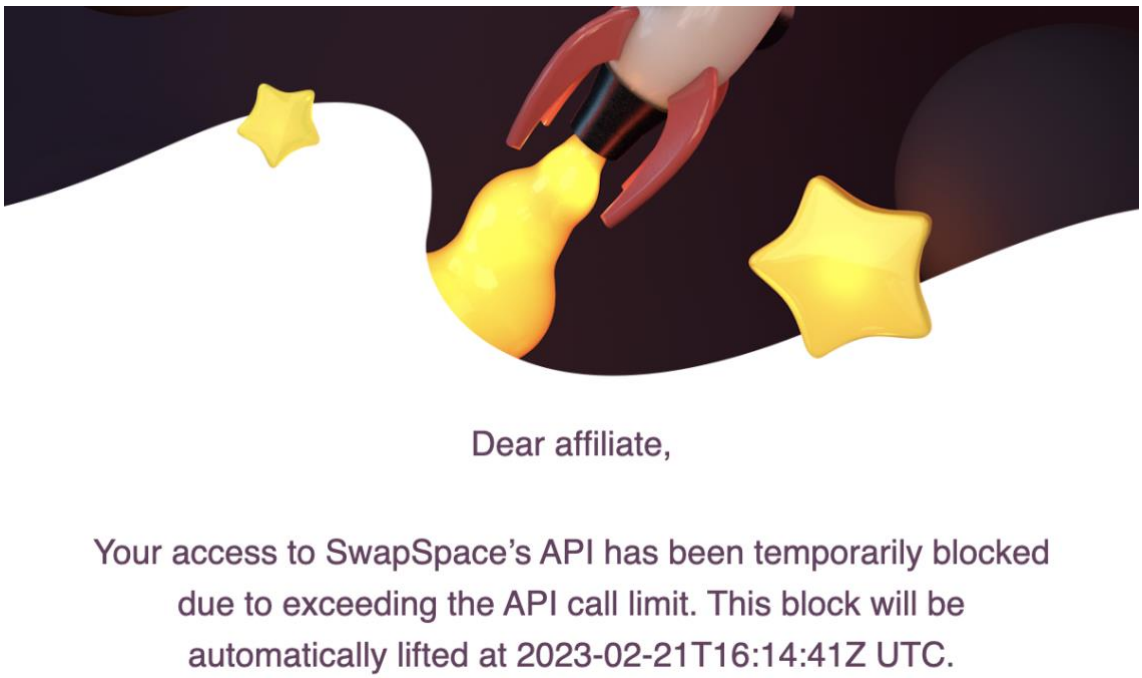


FIGURE 8. Example of an email from SwapSpace issuing a temporary API ban.

However, if the program surpasses the API call limit, a temporary ban will be given to the affiliate account linked to the API key. As shown in Figure 8, upon

getting banned, an email is sent to the affiliate giving the length of the ban. Based on the personal experience of exceeding the API call limit numerous times in a single day from testing, each ban given as a consequence of this violation consistently lasts 30 minutes, regardless of whether it occurs consecutively in the same day. Working with the SwapSpace API after testing Binance and other CEX endpoints showcased its permissiveness.

2.4 Slippage

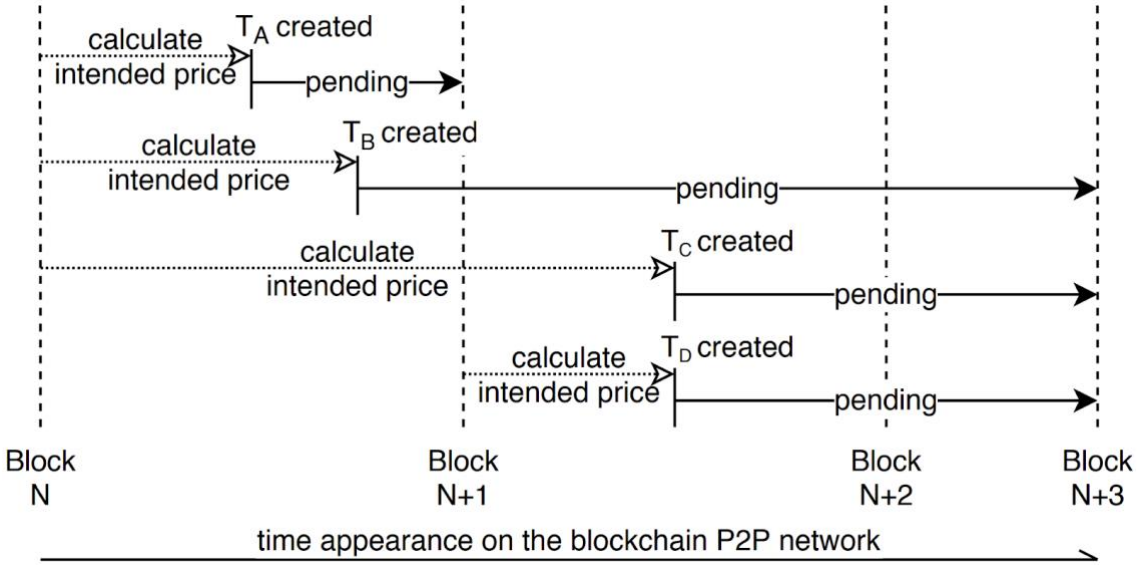


FIGURE 9. Slippage visualization in an Automated Market Maker (AMM) DEX by Zhou, L., Qin, K., Torres, C., Le, D. & Gervais, A. (2021, 4).

Price slippage is the change in an asset's price during a trade, including expected and unexpected changes. Figure 9 graphically visualizes slippage as the following happening: “[Expected price] of T_A is based on the AMM state of block N. T_A does not suffer from unexpected slippage, because no concurrent transactions exist. T_B executes in block N + 3. [Expected price] of T_C ’s is based on block N, as we assume network delays. If T_C and T_D change the state of the underlying market, those may induce unexpected slippage for T_B” (Zhou, L., Qin, K., Torres, C., Le, D. & Gervais, A. 2021, 4) or vice versa for all transactions in block N+3. Expected slippage is the anticipated price change based on volume and liquidity at the start of the trade, while unexpected slippage occurs during the period between trade commitment and execution.

Between getting an exchange rate while searching for profitable arbitrage and actually sending the transaction to swap currencies is expected to have some slippage. This is handled by only searching for circular arbitrage within a small amount of currencies at a time, therefore the length of time between the calculated price and sending any transaction will not be long.

2.5 Previous Implementation

Previous implementations of DFS on a graph of currencies as vertices and exchange rates as edges led the program to the current state it is and the area of crypto trading it is looking for arbitrage. The original project (evvic 2023b) was purely C++11 and attempted at finding profitable circular arbitrage within the Binance CEX convert API endpoints (Binance). This first implementation was heavily focused on speed and efficiency of finding a circular path and executing the trades as quickly as possible. While this original program was fully functional, it never successfully found profitable circular arbitrage. This led me to conclude that profitable arbitrage is not possible within a single CEX where quotes for conversion rates were manipulated. The focus then shifted away from efficiency and towards finding concrete verification of profitable arbitrage in some crypto markets. Further research in the field of crypto arbitrage revealed aggregated exchanges by the service SwapSpace and the web3.py library that simplified crypto transactions.

2.6 Related Works

The project discussed in this thesis has been refined and enhanced by related works in the field. While the project originally focused on only the CEX Binance, specifically its Convert API, further research about circular arbitrage within a single DEX seemed far more ideal than a CEX where the conversion rates appeared to be heavily controlled and confined by the exchange providing them. The git repository “uniswap-arbitrage-analysis” by user ccyanxyz, was enlightening on changing the area of searching for circular arbitrage by considering a DEX like Uniswap; the repository also gave validation for

implementing a recursive DFS function for finding profitable circular arbitrage (ccyanxyz 2022). However, in ccyanxyz's analysis of their projects results, finding profitable arbitrage on a single DEX like Uniswap is challenging due to how much competition for arbitrage already exists. Additionally, in their tests the found profitable arbitrage was no longer profitable after paying the gas fees for each part of the trade (ccyanxyz 2022). Combining all of the previous research and effort inspired change to the original state of the thesis project, to finding a new area for performing circular arbitrage, different from a single CEX or a single DEX such as ccyanxyz's Uniswap arbitrage project, that would potentially be less crowded with competition.

3 PROGRAM METHODS

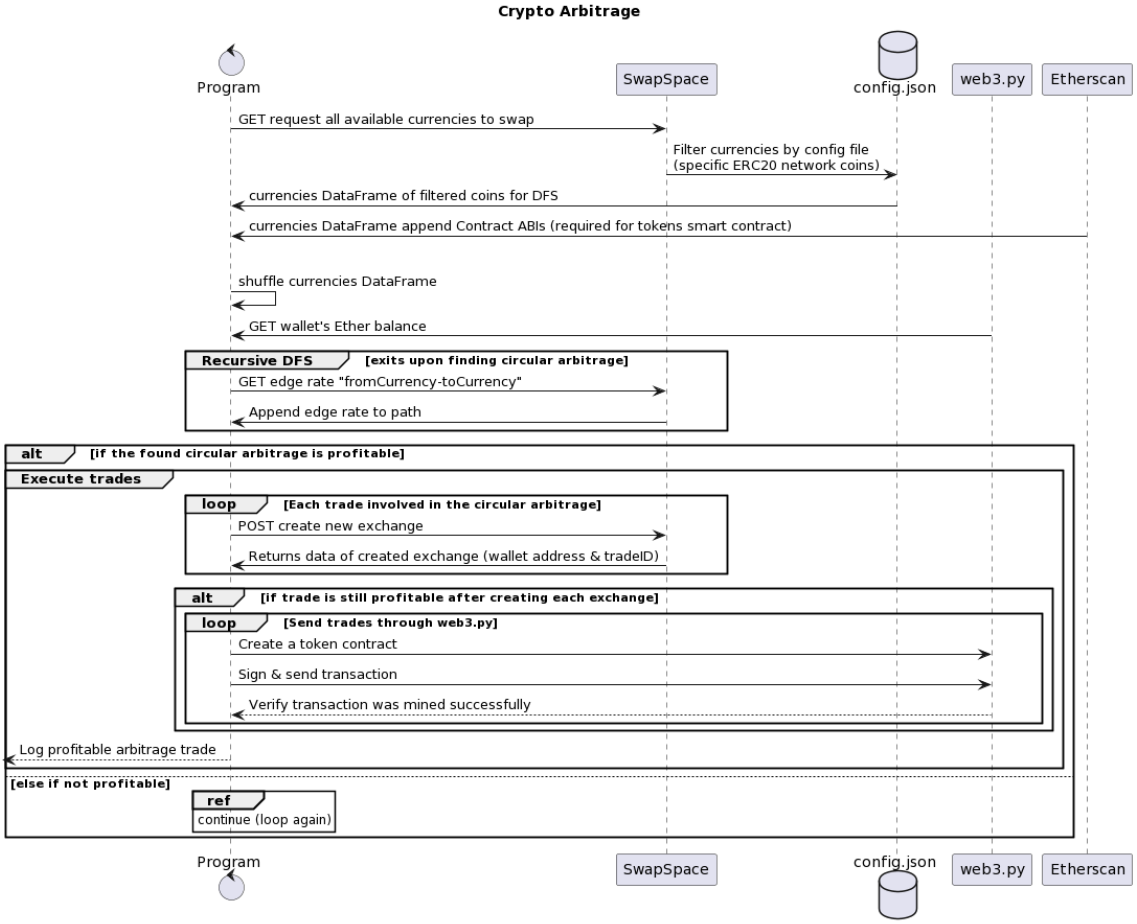


FIGURE 10. Sequence Diagram of the entire program's process.

The purpose of this program is to identify circular arbitrage opportunities across multiple cryptocurrency exchanges. The program “circular-arbitrage-lib” (evvic 2023a) was developed using Python 3.9, along with several external sources including Web3.py, SwapSpace API, and Etherscan API. Web3.py was used to interface with the Ethereum blockchain, while SwapSpace was used to aggregate and order all the exchanges by the best rate for the given coin trade pair. Ultimately, the main functions of the program can be grouped into three sections.

1. Gather the currencies to be used and preprocess their important data.
2. Perform the algorithmic depth-first search over the currencies until finding a profitable circular arbitrage.

3. Execute transactions necessary to perform trades when a profitable circular trade is found

The following sections in this chapter will be presented in the order of their execution (Figure 10) within the software program's methods.

3.1 Load and Pre-process Currencies

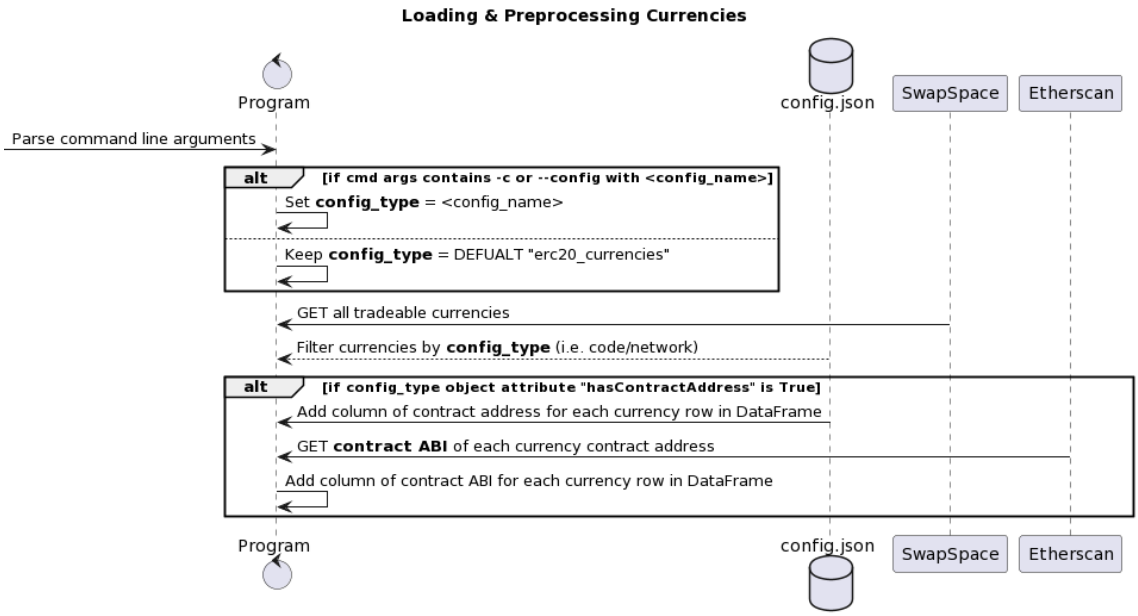


FIGURE 11. Sequence diagram for creating a data frame of currencies with necessary data.

While SwapSpace can exchange over 2250 crypto currencies, the program searching for arbitrage would be very slow and have high chances of slippage if using all available currencies on all possible networks. Therefore, it is important before searching for any arbitrage to filter the currencies to only those desired. Figure 11 is a sequence diagram giving an overview of how the currencies are filtered, and this will be expanded upon in the following subsections. This diagram of loading and preprocessing the currencies does not include error handling that occurs, however if an error does occur the program will abort because there is no purpose in attempting to continue onto searching for profitable arbitrage when missing crucial information.

3.1.1 Configure Network and Tokens

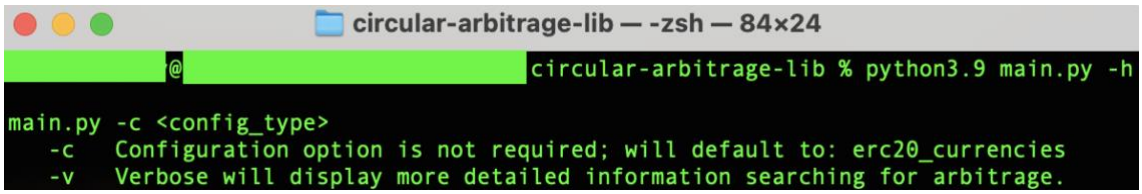


FIGURE 12. Running the program, arbitrage.py with the help flag: -h.

Before running the program, it is important to know the important optional parameter that changes the tokens used in finding circular arbitrage. Figure 12 displays an example of the program's output when the help (-h) option is passed, providing a succinct explanation of the -c option parameter, which enables the customization of the token set and trading network. The program offers the flexibility to search for arbitrage opportunities within a specified network and list of desirable tokens. This can be achieved by modifying the config.json file located in the project's root folder; by adding a list of desired tokens and the network on which they are traded.

```
{
   "erc20_currencies": {
       "network": ["erc20", "eth"],
       "currencies": [
          {"code": "eth",  "contract_address":
"0x0000000000000000000000000000000000000000"},
           {"code": "uni",  "contract_address":
"0x1f9840a85d5af5bf1d1762f925bdaddc4201f984"},
           {"code": "zrx",  "contract_address":
"0xe41d2489571d322189246dafa5ebde1f4699f498"},
           {"code": "usdt", "contract_address":
"0xdac17f958d2ee523a2206206994597c13d831ec7"},
           {"code": "shib", "contract_address":
"0x95ad61b0a150d79219dcf64e1e6cc01f0b64c4ce"},
           {"code": "link", "contract_address":
"0x514910771af9ca656af840dff83e8264ecf986ca"},
           {"code": "1INCH", "contract_address":
"0x111111111117dc0aa78b770fa6a738034120c302"},
           {"code": "lrc",  "contract_address":
"0xbbbbca6a901c926f240b89eacb641d8aec7aeafd"}
       ],
       "hasContractAddress": "True"
   },
   "test_main_tokens": {
       "network": ["eth", "btc", "doge", "dot", "ada", "xrp", "sol"],
       "currencies": [
           {"code": "eth"},
           {"code": "btc"},
           {"code": "doge"},
           {"code": "dot"},
           {"code": "ada"},
           {"code": "xrp"},
           {"code": "sol"},
```

```
            {"code": "usdc"}
        ],
        "hasContractAddress": "False"
    }, …
}
```

FIGURE 13. Showing two configurations in the config.json file: “erc20_currencies” and “test_main_tokens”.

By creating a custom configuration, users are able to tailor the circular arbitrage to the tokens and network the wallet is capable of interacting with. As shown in Figure 13, config.json is an array of config objects that should hold currency codes, the network(s) the currencies exist on, and a Boolean value of whether or not the currencies have contract addresses. The first object in the JSON array being the default option, “erc20_currencies”, is all Ethereum based and by SwapSpace standards is on the erc20 and eth networks. This data is crucial for filtering all currencies returned by the SwapSpace API call to get all available currencies to exchange. It is important to note that while any crypto blockchain/network and any coin can be used for searching for profitable circular arbitrage, the program only has the functionality of executing transactions for tokens on the Ethereum network.

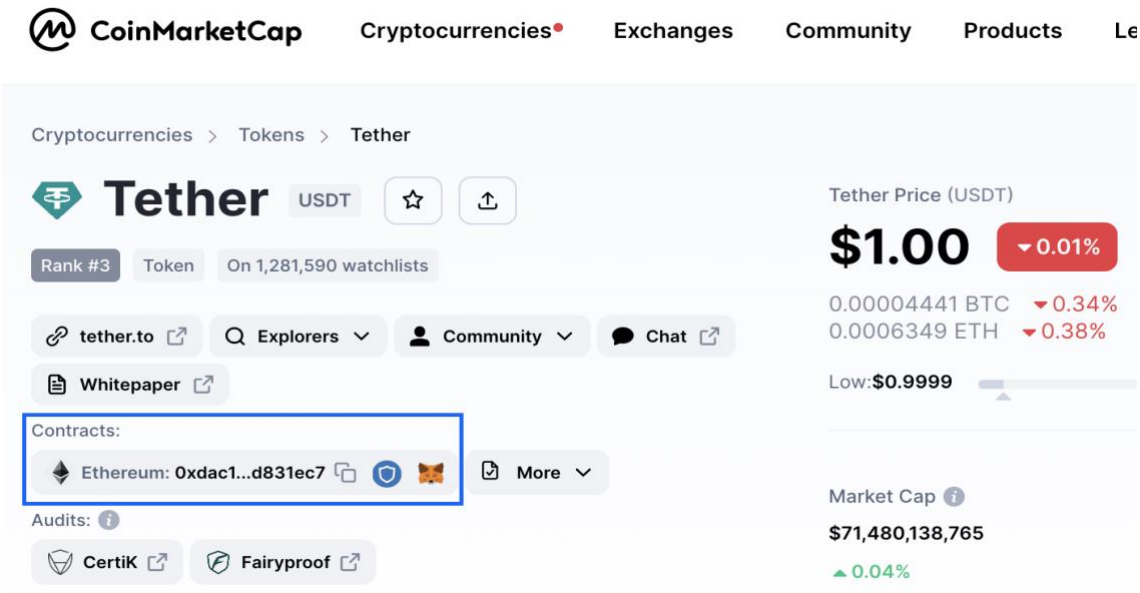


FIGURE 14. An example of getting an ERC20 tokens contract address using CoinMarketCap.

Also optionally, the contract address is supplied for each currency object in the config.json file (Figure 13) if the currencies require it; ERC20 tokens require a contract address for transactions because it builds the smart contract. Finding an Ethereum based tokens contract address can be deceptive because there is a different address for each currency for every net such as test net and proxy address. Therefore, it is important to search for the Ethereum mainnet contract address of the token; CoinMarketCap is a reliable price tracking website that easily gives the mainnet contract address for (ERC20) tokens (Figure 14).

3.1.2 Filter Currencies

FIGURE 15. Example of running the program without any option, defaults to “erc20_currencies” configuration option.

```
[
 {
    "name": "Ethereum",
    "extraIdName": "",
    "popular": false,
    "stable": false,
    "validationRegexp": "",
    "code": "eth",
    "network": "arbitrum",
    "hasExtraId": false,
    "id": "yIUgrQdpat"
  },
  {
    "name": "Binance Coin (ERC20)",
    "extraIdName": "",
    "popular": false,
    "stable": false,
    "validationRegexp": "",
    "code": "bnb",
    "network": "erc20",
    "hasExtraId": false,
    "id": "s8YX6aH1AX"
  },
…
]
```


FIGURE 16. Snippet of SwapSpace API response for getting list of tradable currencies.

Upon starting the program, it first checks if there was an optional name of a configuration object included. If none was passed, then the program defaults to the first object in the config.json file, “erc20_currencies” as shown in Figure 15. Then, performs the API call to SwapSpace, getting the list of all available currencies; this list will have thousands of currencies that SwapSpace has the ability to trade given the partnered exchanges. This JSON response of currencies contains important information (Figure 16) such as name, network, currency code, id, and “extraIdName”, if the specific currency requires it. All these currency objects from the response JSON list are filtered by their network and currency code to just the desired currencies for arbitrage.

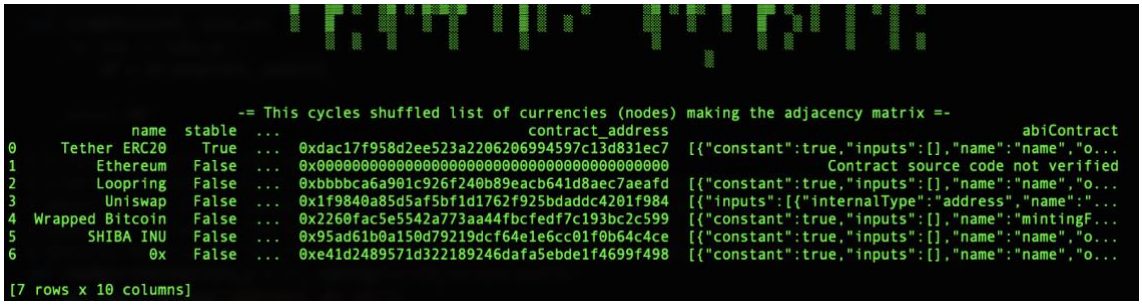


FIGURE 17. The program displays the Data frame of currencies information after being filtered.

Furthermore, if the currencies of the configuration object include a contract address, and the Boolean value “hasContractAddress” is assigned true, contract addresses of the currencies will be appended to the filtered currencies Data frame. Additionally, if the currencies have contract addresses, their ERC20 token contract ABI will be appended as well. The ABI (Abstract Binary Interface) provides rules for interacting with a program or contract at the binary level, identifying functions by their unique signatures based on their name and parameter types. By comparing incoming data to function signatures in the contract's bytecode, the ABI standard ensures compliance and standardized function calls (Di Angelo, M. & Salzer, G. 2020, 2). The contract ABI for each ERC20 token holds all the function declarations the token is capable of using in its smart contract; therefore, it is also needed with the contract address of the token for generating the smart contract.

In this program, the ABI contract for a token is obtained through the Etherscan API. Given the contract address and the Etherscan API key, the get request url would appear as follows:

```
https://api.etherscan.io/api?module=contract&action=getabi&address={}&apikey={}
```

The first {} placeholder was replaced with the actual contract address, and the second {} placeholder was replaced with the API key. This API call uses the “getabi” action of the contract module to retrieve the ABI, and requires an API

key for authorization. With the “abiContract " column appended to the Data frame of currencies, the order of the rows is then shuffled, so traversing the currencies will always be slightly different in its order even if the currencies chosen from config.json remain the same.

Once finished collecting this additional information needed for creating the transactions with the currencies, the program prints the data frame to the terminal as shown in Figure 17. Ethereum has an error with its ABI contract field but that is because it is not relevant or required for its transactions. An Ethereum transaction on the Ethereum network is set up differently from the ERC20 tokens on the ERC20 network in the web3.py library; Ethereum does not need to have a smart contract made manually.

3.2 Searching for Profitable Arbitrage

3.2.1 Graph Theory

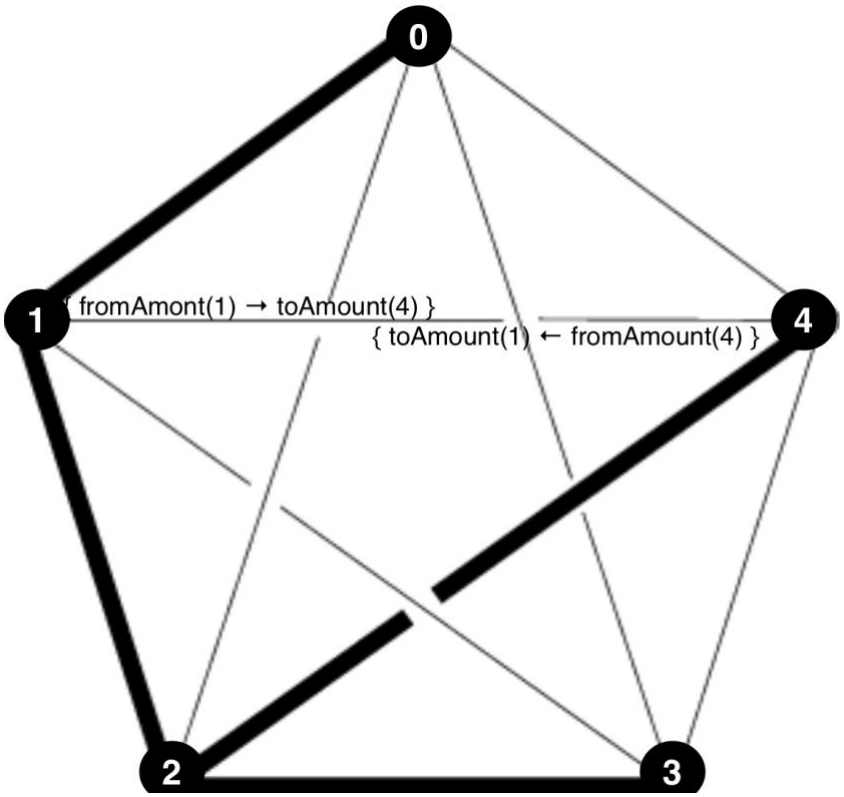


FIGURE 18. Example of a bidirectional graph of 5 nodes of all connected edges

The data structure used in this program represents a bidirectional graph where each node is a currency and the edge between two nodes represents a conversion between those two currencies; an example of this with five “currencies” is shown in Figure 18. Specifically, each edge is labeled with the “toAmount” and “fromAmount” values, which represent the amount of the base (from) currency converted to the destination (to) currency. These edge labels allow the graph to represent the specific conversion rates between different currencies, rather than just the exchange rates. This method of edge representation has its benefits and drawbacks, but ultimately cannot be changed due to the SwapSpace API and will be discussed in the next section.

TABLE 3. Currency object representing a vertice; values as a row in a Data frame of currencies.

Attribute Name	Data Type	Description
name	String	Official name of the token/currency
code	String	Unique code to identify token/currency
network	String	Network the token’s transaction occur on
hasExtraId	Boolean	SwapSpace conditional for additional id name
id	String	SwapSpace id for token
extraIdName	String	SwapSpace additional id name for token
contract_address	Ethereum address	(ERC20 only) Smart contract address for token
abiContract	Stringified JSON	(ERC20 only) ABI of smart contract for token

Typically, in graph theory, an adjacency matrix or adjacency list data structure is used to represent all available edges between nodes of a graph. However, the program only requires a one-dimensional array of the nodes (currencies), because it is implied, they are all connected to one another in both directions as any of the currencies used for finding arbitrage with the SwapSpace API can be traded. This one-dimensional array is represented in the program as a Pandas DataFrame (Table 3) of (shuffled) currencies in Figure 17. A Python list data

structure would also work but the DataFrame data structure can organize and represent data more efficiently than a list of Python dictionaries.

3.2.2 Getting Edge Values

An edge value in this bidirectional graph (Figure 18) is an object containing the amount of the base currency to trade, the amount of the destination currency to receive, the DEX/CEX partner the trade is occurring on, fixed or float exchange rate, and other data shared in Table 4. To get an edge object value (for one direction) between two nodes/currencies, the API call: get estimated exchange amounts to SwapSpace will receive all information for this edge. For example, to find edge options for 0.1 BTC-ETH:

```
https://api.swapspace.co/api/v2/amounts?amount=0.1&fromCurrency=btc&toCurrency=eth&fromNetwork=btc&toNetwork=eth
```

This API call returns a list of exchange options, from different DEX/CEX exchange partners and options of fixed or float exchange rate types (see Appendix 1). In the case of this program, the most important metric is the “toAmount” value, which is the amount received of the destination currency. The best exchange option (edge) is obtained by sorting the JSON array of exchange objects in ascending order by their “toAmount” value and returning the top one, because the “fromAmount” value for each option in the array is the same.

TABLE 4. Trade edge Data frame row representing an edge in the graph.

Attribute Name	Data Type	Description
partner	String	Name of the exchange the swap occurs through
fromAmount	Double	Amount of the base token being sent
fromCurrency	String	Currency code of the base token
fromNetwork	String	Network code of the base token
toAmount	Double	Amount of the destination token being received
toCurrency	String	Currency code of the destination token
toNetwork	String	Network code of the destination token
fixed	Boolean	Conditional whether edge is fixed or float
id	String	Optional id to reference the quote
partnerId	Unsigned Integer	Id of the partnered exchange the swap occurs through

Just this small part of the program is where the exchange arbitrage occurs; choosing the exchange with the best value for each part of the trade adds extra chances of profitability. Additionally, the program is capable of finding circular arbitrage with the count of trades being only two, therefore in that case it is no longer circular arbitrage, just exchange arbitrage. Table 5 is an example of a logged profitable exchange arbitrage due to it only being two trades: BTC-ETH on LetsExchange exchange and ETH-BTC on Changelly. This is just one example of edges in a profitable trade logged by the program. Ultimately, how an edge is used next will be discussed in the next section.

TABLE 5. Example of only exchange arbitrage (BTC-ETH-BTC) found for profitability in the program on 16 February 2023.

partnerId	partner	fromAmount	toAmount	fromCurrency	fromNetwork	toCurrency	toNetwork	fixed
23	letsexchange	0.5	7.2656	btc	btc	eth	eth	False
9	changelly	7.2656	0.5026	eth	eth	btc	btc	False

3.2.3 Depth-First Search

Although there are multiple graph-theoretic algorithms that can be implemented for finding circular arbitrage, Depth-first search (DFS) seemed most ideal with the constraints put on by the SwapSpace API and its speed for quickly finding a circular path regardless of profitability. DFS is a graph traversal algorithm that explores as far as possible along each branch before backtracking (Reif, J. 1985, 230). When applied to a bidirectional weighted graph, DFS uses a stack to keep track of nodes to visit, and the most recently explored node is chosen for expansion first. This is because DFS follows a last-in-first-out (LIFO) strategy for adding and removing nodes from the stack.

```
# Recursive DFS function exploring adjacent currencies (nodes) for SwapSpace
# @Requires const currencies: list of all coins & their (SwapSpace) data
# @Requires visited: list of booleans aligned with currencies flagging where DFS has
been
# @Requires const root: string name of root coin
# @Requires path: indexed path aligned with currencies and visited of current path
# @Requires trade_edges: path of current edges constructing where DFS is exploring to
(edges are data from SwapSpace)
# @Requires fromAmount: latest amount from last trade that will be the amount used
for next trade
# @Returns return_code, trade_edges, best:
def DFS(currencies, visited, root, path, trade_edges, fromAmount, best):
```

FIGURE 19. The program's declaration of the implemented DFS function.

The DFS implementation, which is illustrated in the activity diagram of Appendix 2 and the code snippet of the implemented recursive DFS function's parameters in Figure 19, will be explained in this section. The goal of the implementation is to find the root node, Ethereum (ETH), through forward exploration, beginning from the root node. To avoid redundant exploration, a boolean array named "visited" is used to keep track of the explored currencies, and when a currency is explored, its index in the "visited" array is switched to true. For the purpose of not just exploring, but also tracking the circular trades, a stack (a list in Python) named path holds the indexes of the currencies sequentially being explored through DFS. As a currency gets explored it is pushed onto the stack, then after exhausting the search of all nodes adjacent to it, that currency index is popped off the stack, plus the visited array for that currency index is switched back to false, and the DFS retracts to the previous currency before continuing exploration. For further data tracking, a Data frame “trade_edges”, acting like a stack of the edges data in parallel to the path stack also pushes and pops off

the exchange edge data obtained from the SwapSpace API call as the graph is traversed. The “fromAmount” is the amount of the currency available to exchange for the next currency it's exploring, this makes the API call of getting all exchange rate data between the two (nodes) currencies easier. Finally, a Pandas data frame “best” holds the current best-found circular trade so far for statistical purposes. Every time a (profit or non-profitable) circular arbitrage is found, the resulting amount of Ethereum from the current path of “trade_edges” is compared with the resulting Ethereum from “best”, and if the current is better than best, Data frame best copies the edges. To simplify the activity diagram (Appendix 2), the “best” Data frame is not mentioned, but does not affect how DFS functions to find profitable arbitrage.

```
# Run recursive DFS function
# @Returns return_code int: number to determine return status:
#                           (0 = profit, 1 = no profit, 2 = no circ arb)
# @Returns trade_edges Data frame: trade edges of last circular trade
# @Returns best Data frame: Last best found circular arbitrage (logging)
return_code, trade_edges, best = DFS(currencies, visited, root, path, trade_edges,
fromAmount, best)
```

FIGURE 20. The DFS entrypoint initial call of recursive DFS function and return values.

Upon the recursive DFS function returning back into the entrypoint function, it returns three values shown in Figure 20. The “return_code” is simply an integer used to ultimately determine and handle the DFS outcome:

0. Successfully found profitable arbitrage.
1. The program found arbitrage from DFS, but none profitable.
2. No circular arbitrage found at all in DFS.

If profitability was found then the next section of creating and sending the transactions would occur. Otherwise, this section about DFS would run again, checking again for arbitrage possibilities. The “trade_edges” value is the Data frame holding the stack of edges completing the circular arbitrage. This is necessary for creating the transactions so the exchange partner and amounts of the currencies are known. Finally, “best” is used simply for tracking the best circular arbitrage trade found from this DFS attempt, in most cases no profitable

arbitrage is found, therefore this is used for statistical tracking of the best non-profitable arbitrage found in the entire DFS attempt.

3.2.4 Time Complexity Constraints

```
# If not the root node and path length > 4: skip node
MAX_PATH_LENGTH = 4
# Because the API is slow, long arbitrage path lengths are bad for efficiency
if not currencies.iloc[i]["code"] == root and len(path) >= MAX_PATH_LENGTH:
    if DEBUG_PRINT:
        print("Skipping node because path length is already at", MAX_PATH_LENGTH)
    continue
```

FIGURE 21. Inside the *DFS* function, an imposed limit on max length of found circular arbitrage.

The ideal situation for finding the most profitable circular path in a directional-graph is to have the data structure representing the graph already contain all the edge weights. For example, if all the currencies (vertices) had the rate of exchange as the edge weight, then DFS could run through the list of currencies and their exchange rates to produce an ordered list of the most profitable circular trades in milliseconds.

$$O\ (V + E) \tag{2}$$

The O-time complexity for performing DFS on a graph of (V) vertices and (E) edges (2).

$$E = V\ (V - 1) \tag{3}$$

The number of edges (E) can be calculated when it is known the number of vertices in a bidirectional graph and all vertices (V) are connected (3).

$$O\ (V^2) \tag{4}$$

Therefore, O-time complexity for performing DFS on a bidirectional graph with all vertices (V) connected to one another in both directions is exponential (4), extrapolated from Equations 2 and 3. This time complexity would still be fast on

a graph where all vertices and edge weights are already known. But unfortunately, SwapSpace has no method of giving the exchange rate between two currencies (let alone multiple currency edges); rather, as mentioned in the previous section (Appendix 1), SwapSpace only gives the amount to receive of the destination currency when given the initial amount of the base currency in the swap. This means every iteration in DFS for exploring other nodes, requires an API call to get the amount of the destination currency, given the amount of the previously explored currency. This limitation of speed has a few major constraints to track:

1. Slippage: If the SwapSpace API for estimated exchange rate (Appendix 1) is listed as a float rate, it will likely encounter slippage (positive or negative); however, if it is a fixed rate, “the exchange service will freeze the rates for some time to help you to escape the rate fluctuations” (SwapSpace 2023b).
2. Size of graph: When the graph of currencies for finding circular arbitrage is bidirectional and all currencies are connected with edges in both directions, Equation 4 holds. Therefore, every additional currency added to the list of currencies to explore for finding profitable circular arbitrage exponentially lengthens the time to complete the DFS algorithm. The more time it takes to find circular arbitrage because of a larger graph means more price slippage (Figure 9).
3. Length of circular trade:

$$O\ (V^2 + n) \tag{5}$$

   Due to the many API calls required to get all data for a circular trade, every additional currency included in the circular trade adds an exponential amount of time searching vertices and adds a linear amount of time to execute the (n) number of transactions (5). Therefore, Figure 21 shows purposely limiting the length of found circular arbitrage in the program for efficiency. Finding smaller profitable circular trades is always preferable when considering other factors too, such as gas fees and SwapSpace API call limits (Table 2).

Fortunately exchange rates are not too volatile due to saturated arbitrage, therefore not much slippage would occur even if it took a minute between exploring nodes to executing the transactions. Additionally, before sending the transaction the program will do a final check of prices, which will be discussed more when executing the transactions.

3.3 Executing Transactions

3.3.1 Creating the Exchanges

Upon finding a profitable circular trade, the DFS algorithm recursively returns and brings back the Data frame of trade edges, the edge representing each trade with crucial information specified in Table 4. The “return_code” also returned from the DFS function would equal 0, meaning profit was detected. This calls the “executeTrades” function with the currencies and “trade_edges” Data frames as parameters. The “trade_edges” are then iterated through sequentially to create an exchange for each trade in the circular arbitrage. Creating an exchange on SwapSpace (see Appendix 3) returns crucial information such as the wallet address to transfer the base token to.

After creating all the exchanges from each trade edge, the important data collected from the API response is added to the trade edge data frame, modifying the columns and transforming the Data frame from trade edges to exchange edges (Table 6). Although most of the data remains the same as the two edges are similar, differentiating them by names and their columns helps simplify the transaction process and to focus on the necessary data. Creating the exchange for the transaction is different from getting the estimated amounts during DFS because in the former, SwapSpace communicates with the partnered DEX/CEX to officially establish the trade. As shown in Table 6, crucial data from creating the exchange is returned, such as the wallet address to transfer the tokens to for the CEX/DEX to swap with. Additionally, the ID of trade is important to have to check the status of the exchange after sending the funds.

TABLE 6. Exchange edge changed data from trade edge to relevant data captured from API

Attribute Name	Data Type	Description
partner	String	Name of the exchange the swap occurs through
toAddress	String	Ethereum wallet address of partnered exchange to transfer base currency funds to
fromAmount	Double	Amount of the base token being sent
fromCurrency	String	Currency code of the base token
fromNetwork	String	Network code of the base token
toAmount	Double	Amount of the destination token being received
toCurrency	String	Currency code of the destination token
toNetwork	String	Network code of the destination token
rate	Double	SwapSpace given rate of exchange between tokens
exchangeId	String	SwapSpace id of the created exchange
status	String	SwapSpace labeled current status of the exchange

Finally, to check if any slippage has occurred since the estimated amounts were given during the search for profitable arbitrage, the rate of exchange for each edge is multiplied together. If the product is greater than 1, the trade is still profitable before factoring gas fees and further unexpected slippage that could occur until all the received tokens are in the MetaMask wallet associated with the program. If it has changed and the circular trade is no longer profitable, the function returns and the program cycles back to searching for profitable arbitrage; otherwise, it continues with the transaction.

3.3.2 Sending the Transactions

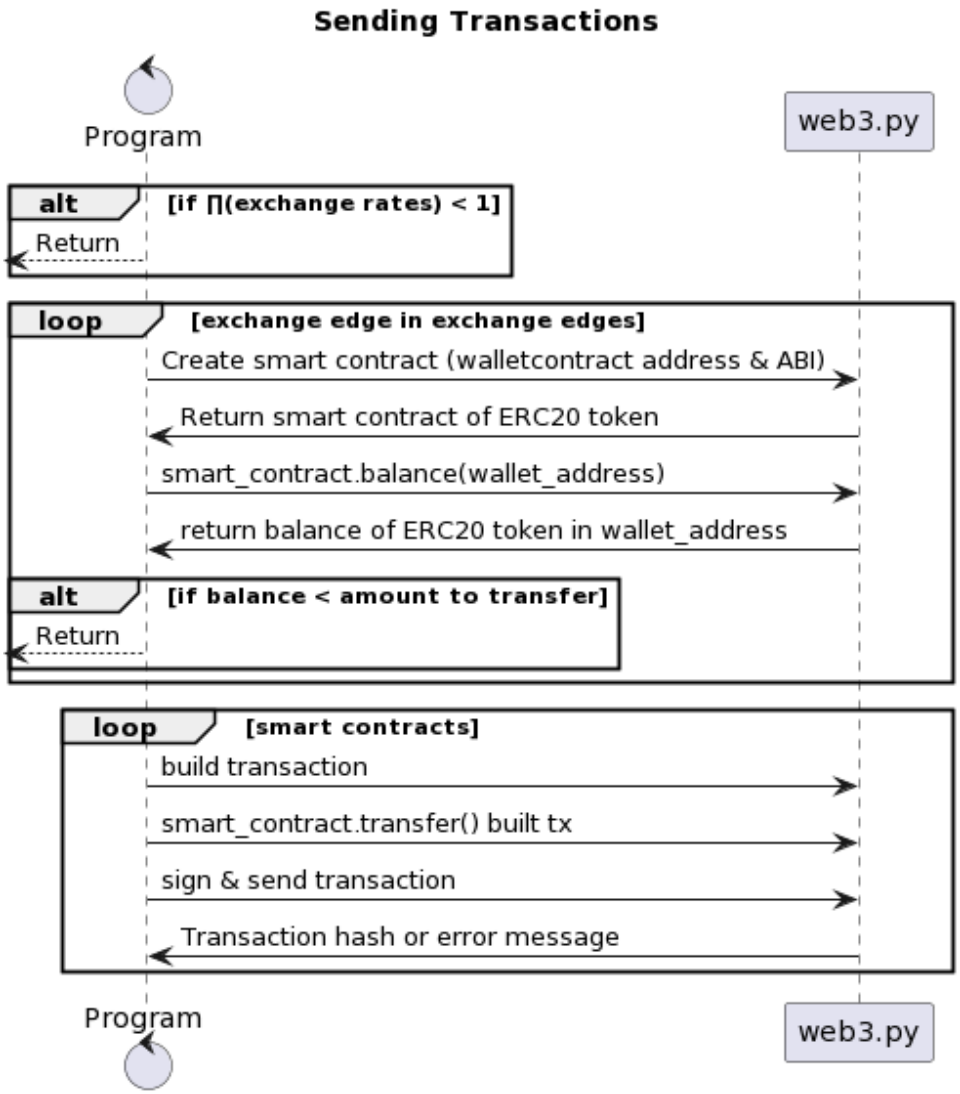


FIGURE 22. Sequence diagram of the steps in sending all transactions.

Since the exchange edges involved in circular arbitrage are still profitable after creating the exchanges, each token will now be sent sequentially to the responding wallet address SwapSpace created with the exchange (Figure 22). Every ERC20 token has a contract address and ABI included in the edge; the contract address is the address to find the specific smart contract (Figure 5) on the Ethereum network, and the ABI is basically a JSON-formatted dictionary of the specific smart contract's functions and their parameters and return type(s). The program functionality currently only supports Ethereum and ERC20 tokens for transactions. For each exchange edge which is an ERC20 token, the smart contract for the token is created through web3.py with the contract address and

ABI; with the smart contract created, all functionality defined in the ABI of the contract can be used. Before creating any transaction with the smart contract, the balance of each ERC20 token is checked, and if any token balance in the wallet associated with the project is less than the amount required to send to SwapSpace, the function returns and the search for profitable arbitrage continues.

```
tx = token_contract.functions.transfer(
   toAddress, int(amount)
).build_transaction(
   {
       "chainId": CHAIN_ID_ETH_MAINNET,
       "nonce": nonce,
       "maxFeePerGas": int(w3.eth.gas_price * 1.5),
       "maxPriorityFeePerGas": int(w3.eth.max_priority_fee * 1.5),
       "type": 2,
   }
)
```

FIGURE 23. Code snippet of building a transaction and using ERC20 transfer function.

Otherwise, the ERC20 token transaction is built with the basic information (Figure 23) of the amount, the wallet address to send to and other minor details. The transfer function is then called with the built transaction; every ERC20 token smart contract has a mandatory transfer function (Figure 5). Afterwards the transaction is signed and sent to the Ethereum network. If the transaction was successfully sent, it will return a transaction hash that can be used to reference the transaction and check its status. Although this transfer can be checked with the transaction hash, it is more beneficial to monitor the overall status of the exchange with the SwapSpace exchange ID.

3.3.3 Verify the Transactions Success

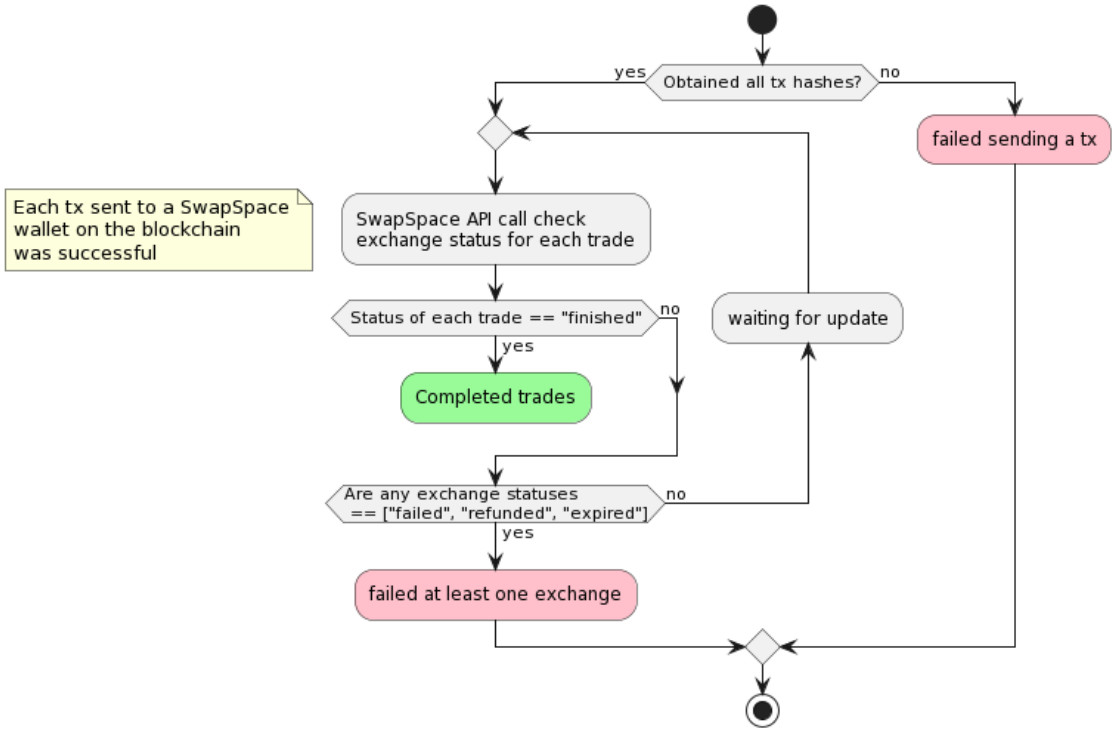


FIGURE 24. Activity diagram demonstrating logic flow of verifying all transactions success.

Verification must be performed because the transactions are sent to SwapSpace wallets through the blockchain, not through a SwapSpace API call returning a status of the transaction. SwapSpace does have an API endpoint for exchange status, this is useful for tracking all statuses after sending the transaction to their wallet address. Transaction verification of all trades involved in the detected profitable circular arbitrage involves two main checks.

1. The first check is whether all the trades were sent to the SwapSpace wallets through web3.py, represented in the first conditional block in Figure 24. Before SwapSpace makes the swap and the desired currency is transferred into the wallet associated with the project, SwapSpace must first detect the base token was sent to their wallet. If the trade was successfully sent to the blockchain, the program will receive a transaction hash, basically a receipt that can be used to check the status of the transaction through web3 or websites like Etherscan. Although a transaction hash does not confirm the currency made it to the

SwapSpace wallet, it at least confirms the transaction was sent which is enough for this first check.

2. The second check involved using the “exchangeId” value given from SwapSpace upon creating an exchange for a trade (Table 6). SwapSpace has an API endpoint to check the status of an exchange given its “exchangeId”, therefore this API call is performed for each leg of the circular trade. The status SwapSpace returns is one of the keywords in Table 7, these keywords are categorized for the program to understand when a trade is complete, failed, or still pending and needs time to be determined (Figure 24).

Whether the circular arbitrage was successful in all the transactions or not, the exchange edges data frame is logged so the “exchangeId” can be checked in the future in case there are still pending trades. Some exchanges can take nearly an hour to complete so when it takes too long the program will continue searching for arbitrage rather than keep waiting for an update.

TABLE 7. A list of all SwapSpace exchange statuses grouped into categories of pending, successful, and failed.

Group	Status	Description
pending	waiting	Waiting to receive funds into designated wallet
	confirming	Confirming the exchange/swap
	exchanging	Currently exchanging currencies
	sending	Sending exchanged currency to project wallet
	verifying	Verifying currency made it back to project wallet
successful	finished	Successfully completed exchange
failed	failed	Failed the exchange
	refunded	Initial currency being sent back to project wallet
	expired	Funds were not sent to SwapSpace wallet in time

4 RESULTS

The purpose of this thesis was to build and test a programmatic method of finding profitable circular arbitrage across multiple crypto DEXes and CEXes. The results of what the program was able to find after being allowed to run for lengthy windows of time, around 6-8 hours per session over a month, are presented in this report. The data collected and logged was the SwapSpace quoted exchange of two currencies that combined with others made a profitable circular trade. Ultimately, over the span of the month, the program ran for an estimated total of 96 hours.

Circular arbitrage is a competitive area saturated with bots and programs looking to find price discrepancies in the market and quickly make trades to profit off that discrepancy. This keeps the market balanced but makes arbitrage opportunities very low. Table 8 shows four logged profitable circular trades on February 27, 2023, however the profitability is extremely low and while most arbitrage opportunities will be like this, they most likely will be disregarded, unless the high volume of the trade allows its low profit percentage to still profit after covering the gas fees.

TABLE 8. Profitable circular/exchange arbitrage trades logged on 27 February 2023.

Logged trade ID	Profit (%)	Start amount (ETH)	End amount (ETH)	Number of transactions
00	0.01	1.0	1.000107	4
01	0.09	1.0	1.000856	3
02	0.11	1.0	1.001125	2
03	0.2	1.0	1.002021	4

Interestingly, a case of only exchange arbitrage was found on 27 February 2023 from trade-02 (Table 8). As shown in Table 9, the exchange involved swapping Ethereum for Loopring (LRC) and back to Ethereum; this did not involve circular

arbitrage as it only involved the base currency and one other currency. The swap from ETH to LRC occurs on SimpleSwap and the second trade of LRC to ETH occurs on Changelly, generating a profit of 0.11% or 0.001125 ETH. Therefore, showing profitable exchange arbitrage can be found on this program as well.

TABLE 9. Trade 02 from TABLE 8 more in depth with a profit of 0.11%, displaying exchange arbitrage only.

partner	From amount	From currency (code)	To amount	To currency (code)
simpleswap	1.0	eth	4499.232997	lrc
changelly	4499.232997	lrc	1.001125	eth

Arbitrage is a low risk, low reward field in finance, therefore arbitrageurs understand their profit margins will never typically be in the double digits. An ideal profitable circular trade is shown in Table 10, where the profit is 2.46% over a total of four trades to return back to the starting crypto currency, Ethereum (ETH). This logged trade is also an ideal example for showing the benefit of combining circular arbitrage with exchange arbitrage, as out of the four trades, there are three distinct partners used for swapping crypto currencies.

TABLE 10. A logged circular arbitrage with an ideal profit of 2.46% on 8 March 2023 consisting of 4 trades.

partner	From amount	From currency (code)	To amount	To currency (code)
changelly	1.0	eth	4811.943275	lrc
changelly	4811.943275	lrc	1516.336022	usdt
swapuz	1516.336022	usdt	0.072269	wbtc
letsexchange	0.072269	wbtc	1.024628	eth

So far for all circular trades logged the found profitability has been low but expected for arbitrage trading. However, the logged trade in Table 11 shows the program finding a circular trade of 4 currencies (inclusive) for a profit of 146.02%; a surreal four-part trade of starting with 1 ETH and ending with 2.46 ETH. Although this was originally disregarded as a bug, all JSON responses from creating each exchange with the partners were examined (see Appendix 4) and no inconsistencies were found. All four exchanges were created within a 10 second window shown by their timestamps, and the JSON data looks correct for any initiated exchange.

TABLE 11. A logged circular arbitrage with an inconceivable profit of 146.02% on 9 March 2023 consisting of 4 trades.

partner	From amount	From currency (code)	To amount	To currency (code)
changehero	1.0	eth	7117.041501	zrx
changelly	7117.041501	zrx	0.073341	wbtc
swapuz	0.073341	wbtc	646.590213	uni
changelly	646.590213	uni	2.460185	eth

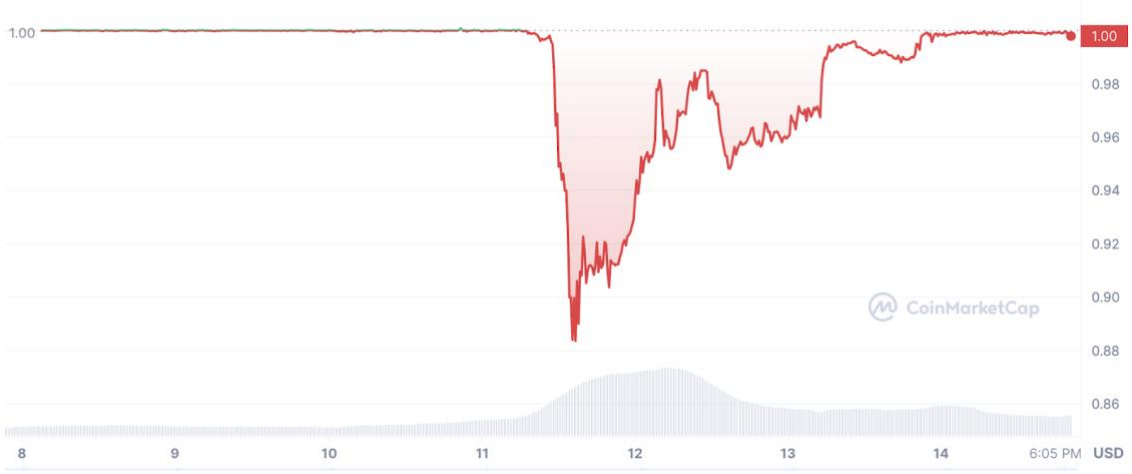


FIGURE 25. USDC price from 8 March 2023 to 15 March 2023 (CoinMarketCap).

After the collapse of a major US bank, Silicon Valley Bank (SVB) on Friday, March 9th 2023, the crypto market was disrupted. The stable coin USDC (USD Coin) meant to imitate USD, fell to $0.87 on Saturday (Figure 25) causing it to deviate from its 1:1 dollar peg (Helmore, E. 2023). This affected other crypto currencies and the market was in heavy turmoil, this disruption ultimately led to price disparities between crypto currencies and between DEXes thus allowing an extremely high profit (Table 11) to be possible in arbitrage. It can be expected that the end result from actually executing the transactions would not be close to the predicted profit percentage due to high slippage from all the market turmoil.

5 DISCUSSION

While this thesis accomplished its purpose of finding profitable circular and exchange arbitrage across multiple crypto CEXes and DEXes, the research thus far has unveiled further avenues to explore for improving the program's efficiency and search for profitable arbitrage. For example, creating a custom crypto exchange aggregator that would replace the functionality of SwapSpace. This would especially benefit the program by conceivably populating all edges in the graph of currencies so DFS would run faster, rather than be limited to one API call per iteration of the algorithm to get the next trade edge. Also, besides just optimizing the code for faster execution time, other methods mentioned below can be further researched and tested to drastically improve the goal of the program.

5.1 Funding

Transaction Hash	Method	Block	Age	From		To	Value	Txn Fee
0x84c206b275103443...	Transfer	16705794	39 days 18 hrs ago	0xEc20a9...AFb23ed6	OUT	Wrapped BTC: WBTC...	0 ETH	0.00044855
0xc1d6fde6809f44d37...	Transfer	16691485	41 days 18 hrs ago	0xEc20a9...AFb23ed6	OUT	Tether: USDT Stablec...	0 ETH	0.00060407
0x547c8df565a961766...	Transfer	16691135	41 days 19 hrs ago	0xEc20a9...AFb23ed6	OUT	0x: ZRX Token	0 ETH	0.00061258
0xc3fb155190989d9a2...	Transfer	16690899	41 days 20 hrs ago	0xEc20a9...AFb23ed6	OUT	Wrapped BTC: WBTC...	0 ETH	0.00086792
0x50fcd5d66e76df433...	Transfer	16685872	42 days 13 hrs ago	0xEc20a9...AFb23ed6	OUT	0x: ZRX Token	0 ETH	0.00091408
0xab0deee86b1fdb8d4...	Transfer	16685729	42 days 14 hrs ago	0xEc20a9...AFb23ed6	OUT	0x: ZRX Token	0 ETH	0.00091942
0xb331f2bcd22b83c63...	Transfer	16677300	43 days 18 hrs ago	0xEc20a9...AFb23ed6	OUT	0x9798d5...3cb956b5	0 ETH	0.00084
0x33b8def9c17032076...	Transfer	16669518	44 days 20 hrs ago	0x10788F...B8D1503d	IN	0xEc20a9...AFb23ed6	0.00584438 ETH	0.00051484

FIGURE 26. Transaction history of the Metamask wallet associated with the program provided by Etherscan.

Unfortunately, none of the detected circular trades could become successful transactions on the blockchain due to lack of funds in the wallet. Given the trade volume for each trade being around 1 Ethereum or $1450.00 USD, and the wallet having only at most 0.0058443 ETH, or $10.00, there was only enough Ether for gas fees. The program attempted to send transactions to the blockchain, but the miners were unable to execute them due to the absence of funds in the Metamask wallet associated with the program. As a result, the designated SwapSpace wallet could not receive the funds to initiate the swap.

Figure 26 shows failed transactions of sending ERC20 tokens out from the project's wallet; although they failed, the gas fee was still paid to the miner who attempted the transaction. When the wallet no longer had funds to cover the gas fee, the transaction would fail to send.

5.2 Expanded Graph

A current limitation is the number of currencies used in the DFS algorithm for finding arbitrage: typically, only 7 or 8 currencies were used (Figure 17) and were mostly the same currencies throughout testing. Every additional currency used adds an exponential amount of unique paths from the graph for traversal, thereby increasing arbitrage potential. Increasing the number of current currencies in the graph has the drawback of making the algorithm much slower because each one-directional edge between currencies requires an API call to SwapSpace to get the "to amount" and "from amount" of currencies in the exchange. Increasing the amount of currencies increases the graph size, thereby increasing total time for DFS graph traversal. Mathematically testing the optimized amount of currencies/nodes in the graph would give the most amount of arbitrage opportunities while ensuring it is not too slow for entire graph traversal. Along with lack of nodes used for arbitrage detection, all the currencies were limited to Ethereum based tokens due to the program only having functionality for sending transactions on the Ethereum network. Arbitrage detection in the program can work with any coins SwapSpace supports, but the program only has access to a Metamask wallet that only supports transactions on the Ethereum network. It would be beneficial for finding price discrepancies by expanding the network diversity through allowing currencies that function on other networks such as Bitcoin, Polygon, Binance Smart Chain and so on.

5.3 Expanded Networks

As previously touched on as a graphical limitation, the arbitrage possibilities could be increased by broadening the number of crypto currencies used by

allowing different crypto networks. The current state of the program can only handle Ethereum based transactions, ERC20, tokens. However, allowing other networks such as Bitcoin and Polygon would diversify the coins and markets further to potentially find even more price discrepancies in a circular trade between them all. It is possible to implement these other networks in the program as SwapSpace has these networks and more, already tradeable. The program would just need to be expanded upon to have functionality of sending transactions to the Bitcoin blockchain, Polygon blockchain, and so on. Additionally, each new supported network would need a wallet that supports that specific network; such as the Metamask wallet used with the program only functions with tokens on the Ethereum blockchain.

5.4 C++ PyBind

```
#include <pybind11/pybind11.h>
#include <pybind11/stl.h> // Needed for list -> vector conversion
#include <vector>
#include <iostream>

namespace py = pybind11;

PYBIND11_MODULE(arbitrage, m) {
   // C2CEdge struct (all product pairs as edges)
   pybind11::class_<C2CEdge>(m, "C2CEdge")
       .def(pybind11::init<unsigned short, unsigned short, double, std::string,
std::string>())
       .def(pybind11::init())
       .def_readwrite("from", &C2CEdge::from)
       .def_readwrite("to", &C2CEdge::to)
       .def_readwrite("rate", &C2CEdge::rate);
}
```

FIGURE 27. C++ class interface into Python library from circular-arbitrage-lib/pybind.cpp (evvic 2023a).

One drawback on performance with the program is it is entirely written in Python. This allowed the development to be much faster, but using a compiled language like C++ would improve speed, thereby decreasing the amount of uncalculated slippage between price calculation and transaction execution. Although it is too much to expect the whole program to be rewritten in a low-level language, considering the complicated Python libraries used such as web3.py. Instead, it is possible to have the benefit of using Python for making and sending the smart contracts but using C++ for DFS and finding circular

arbitrage through the PyBind11 package. A C++ class (Figure 27) can be imported into Python as a library using PyBind11 to reap the benefits of C++s efficiency and speed. The library can then replace the majority of the current Python program, and just have to return the information needed for web3.py to create the smart contracts and send them.

5.5 Solidity

FIGURE 28. Flowchart of creating and deploying a smart contract from Solidity.

Solidity is a programming language and most popular for writing smart contracts for Ethereum and other blockchains (Solidity 2023). While smart contracts can be created with web3.py, custom smart contracts have to be written in Solidity (Figure 28) and compiled so the Python script or any program can use the custom smart contract. Learning and implementing Solidity into the program involved with sending transactions to a blockchain would allow for new methods of sending the transactions that will be further expanded on below.

5.5.1 Atomic Transactions

One significant issue regarding how transactions are sent sequentially (Figure 22) in the created program is the possibility of one or more of the transactions to fail. Most likely this would ruin the chance of profitability in the circular trade. Traders can group all *n* trades of a circular arbitrage into a single blockchain transaction using smart contracts, which allows for atomic execution (Ye Wang et al. 2022, 7); any attempt to insert other transactions between these trades is prevented. Additionally, Ethereum allows for the cancellation and state-reversion of the smart contract if not all conditions are met, therefore, the entire

smart contract is cancelled and actions are reversed if one transaction within the entire circular trade is not executed. Although the gas fees paid with the smart contract are not recouped to the trader, this solves the issue of sequentially executing the transactions.

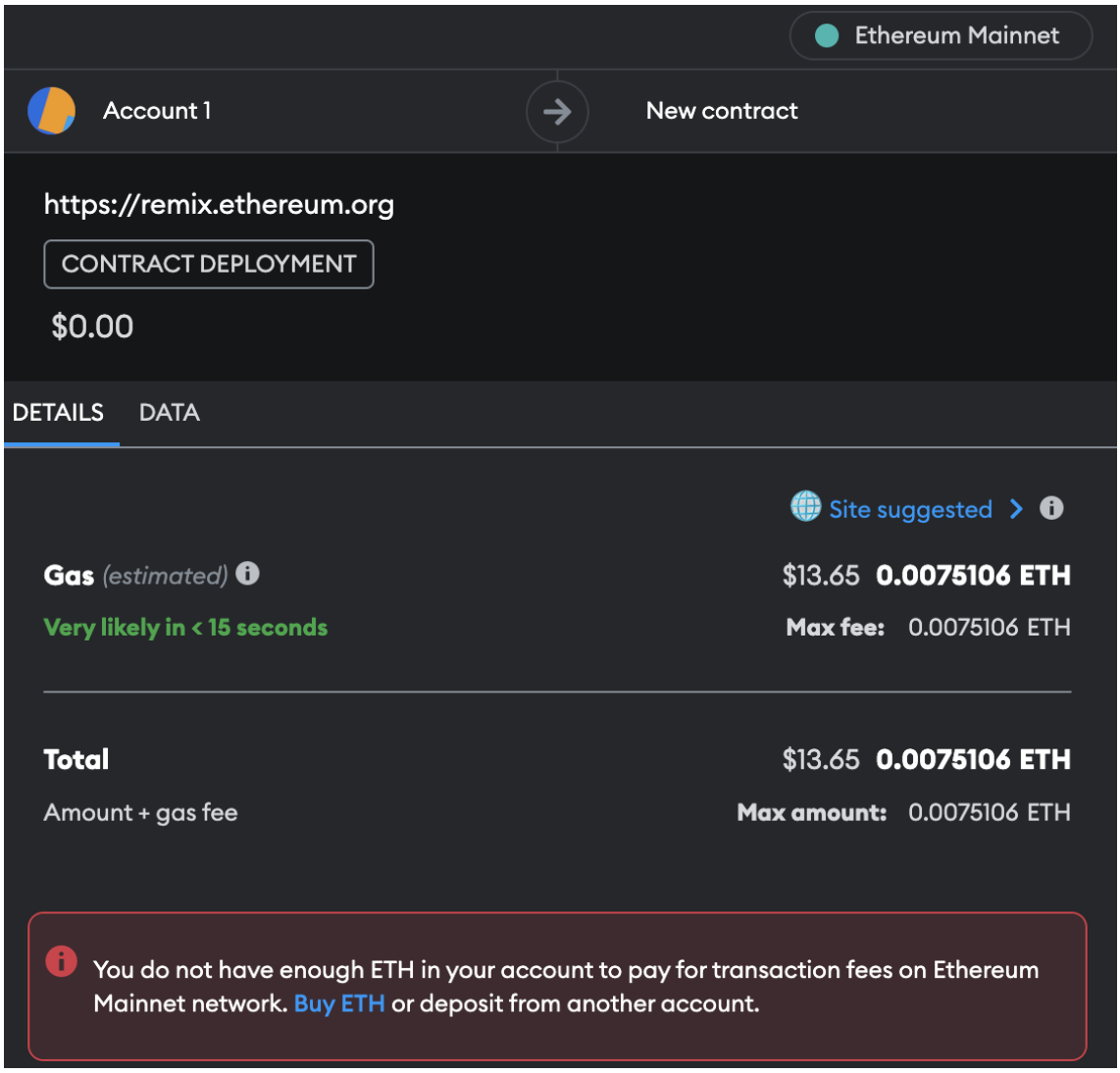


FIGURE 29. Deployment cost of a smart contract on Ethereum network 22 March 2023.

Deploying a custom smart contract to the Ethereum network can be very expensive; any changes or additions to the smart construct would have to be redeployed and the fees would need to be paid again. When attempting to implement atomic transactions in the program, the created smart contract on Remix IDE would cost 0.0075106 ETH, or $13.65 at the time of deployment (Figure 29). Deployment fees are one of the reasons the program for this thesis ultimately did not implement atomic transactions.

5.5.2 Flash Loans

For profitable arbitrage it is important to trade in high volumes to minimize the impact of gas fees; this can make it difficult to begin if it is not possible to supply a large sum of capital to the program to trade with. However, smart contracts allow transactions to be grouped into one atomic transaction on the Ethereum blockchain. Therefore, flash loans were introduced in Uniswap V2 to enable users to borrow funds up to the total liquidity available in a pool and repay the full sum in the same Ethereum transaction. Flash loans have no credit risk to the lenders because the loan is both originated and repaid in the same atomic transaction (Lehar, A. & Parlour, C. 2021, 13). Further research must be done to truly determine if implementation of flash loans into the Solidity-made smart contracts is viable with this program's approach for circular arbitrage across different DEXes/CEXes. If possible, it might be limited to parts of the trade occurring on the Uniswap DEX only.

6 CONCLUSIONS

In this paper the viability of combining circular arbitrage and exchange arbitrage is demonstrated to find new profitable trade opportunities. To find profitable arbitrage on the Ethereum network despite high gas fees, a high trade volume is necessary to offset the impact of subtracting gas fees from low percentage profitable trades. All instances of profitable exchange and circular arbitrage found in the results were not capable of being executed onto the blockchain due to lack of funds in the wallet (Figure 26), therefore it was ultimately not verifiable profit can be realized through the program, but can at the very least detect profitable trade paths across multiple DEXes/CEXes. Executing transactions on an Ethereum testnet would be ideal for testing the realized profit, but SwapSpace only supports Ethereum mainnet for swapping transactions, so testing the detected profitable circular trades requires funding.

This paper also supports the use of graph-theoretic algorithms for finding profitable trades. Although the program and thesis only used depth-first search for finding profitable circular trades, other graph-theoretic algorithms such as Bellman-Ford and breadth-first search (BFS) could easily be implemented instead. Any graph-theoretic algorithm used must be modified for the technical implementation because of the unconventional directional graph used in this program. A directional graph where edge weights were not the implicit rates between currencies rather, they are the exact amount of each currency being exchanged is an unconventional and limiting approach imposed by use of the SwapSpace API for aggregating exchanges to trade crypto currencies on. This limitation is ultimately valued as beneficial as the DFS algorithm can still find profitable trades, either as circular or exchange arbitrage.

While profitable circular arbitrage across DEXes was found occasionally, a higher quantity of profitable trades was expected to be found. Although there are numerous limitations to consider as to why the quantity of profitable trades found was underwhelming. The main and most likely reason for this is lack of program run time: over the span of March and late February 2023, the program was actively running for a total of 1 week and the windows of time it ran at were usually at the same time of the day. More runtime would allow for more

exposure, and running the program at different times of the day would add more variety to market fluctuations as different parts of the world are actively trading.

REFERENCES

Ansari, K. & Kulkarn, U. 2020. Implementation of Ethereum Request for Comment (ERC20) Token. Proceedings of the 3rd International Conference on Advances in Science & Technology (ICAST). Read on 04.03.2023. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3561395

ccyanxyz. 2022. uniswap-arbitrage-analysis. [Computer program]. Updated on 05.11.2022. Read on 18.01.2023. https://github.com/ccyanxyz/uniswap-arbitrage-analysis

Cuffe, P. 2018. The role of the erc-20 token standard in a financial revolution: the case of initial coin offerings. IEC-IEEE-KATS Academic Challenge, Busan, Korea, 22-23 October 2018. Read on 04.03.2023. https://researchrepository.ucd.ie/entities/publication/9fb39a34-fac8-47dc-af16-58f6332351c4/details

Deer, M. 2022. Will the Ethereum 2.0 update reduce high gas fees? [Article] Released on 12.03.2023. Read on 03.03.2023. https://cointelegraph.com/explained/will-the-ethereum-20-update-reduce-high-gas-fees

Di Angelo, M. & Salzer, G. 2020. Tokens, types, and standards: identification and utilization in Ethereum. 2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). IEEE. Read on 07.03.2023. https://publik.tuwien.ac.at/files/publik_287890.pdf

Ethereum. 2022. EIP-20 Token Standard. EIPs. [GitHub] Updated on 06.05.2022. Read on 04.03.2023. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

evvic. 2023a. circular-arbitrage-lib. [Computer program]. Updated on 06.04.2023. Read on 06.04.2023. https://github.com/evvic/circular-arbitrage-lib

evvic. 2023b. graph-theory. [Computer program]. Updated on 31.01.2023. Read on 31.01.2023. https://github.com/evvic/graph-theory

Helmore, E. 2023. USD Coin Value Falls after Revealing $3.3bn Held at Silicon Valley Bank. The Guardian. Release on 11.03.2023. Read on 14.03.2023. https://www.theguardian.com/technology/2023/mar/11/usd-coin-depeg-silicon-valley-bank-collapse

Lehar, A. & Parlour, C. 2021. Decentralized exchanges. Available at SSRN 3905316. Read on 03.03.2023. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3905316

Lo, Y. & Medda, F. 2021. Uniswap and the Emergence of the Decentralized Exchange. Journal of Financial Market Infrastructures 1-25. Read on 03.03.2023. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3715398

Merriam-Webster. 2023. Arbitrage. Merriam-Webster Dictionary. Read on 04.03.2023. https://www.merriam-webster.com/dictionary/arbitrage?src=search-dict-box

Panda, S. & Satapathy, S. 2021. An investigation into smart contract deployment on Ethereum platform using Web3. js and solidity using blockchain. 549-561. Data Engineering and Intelligent Computing: Proceedings of ICICC 2020. Read on 06.04.2023. https://link.springer.com/chapter/10.1007/978-981-16-0171-2_52

Reif, J. 1985. Depth-first search is inherently sequential. Information Processing Letters 20.5. 229-234. Read on 10.03.2023. https://users.cs.duke.edu/~reif/paper/dfs.ptime.pdf

Smith, C. 2023. Gas and Fees. Ethereum. Updated on 17.02.2023. Read on 03.03.2023. https://ethereum.org/en/developers/docs/gas/

Solidity. 2023. Solidity (Version 0.8.19). Updated on 22.02.2023. Read on 20.03.2023. https://docs.soliditylang.org/en/v0.8.19/

SwapSpace. 2023a. Docs. SwapSpace API (V2). Read on 29.01.2023. https://docs.swapspace.co/

SwapSpace. 2023b. F.A.Q. SwapSpace. Read on 29.01.2023. https://swapspace.co/faq

Wang, Y., Chen, Y., Wu, H., Zhou, L., Deng, S. & Wattenhofer, R. 2022. Cyclic arbitrage in decentralized exchanges. In Companion Proceedings of the Web Conference 2022. 12-19. Read on 26.02.2023. https://arxiv.org/pdf/2105.02784.pdf

Wood, G. 2014. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper. 1-34. Read on 12.03.2023. https://files.gitter.im/ethereum/yellowpaper/VIyt/Paper.pdf

Zarir, A., Oliva, G., Jiang, Z. & Hassan, A. 2021. Developing cost-effective blockchain-powered applications: A case study of the gas usage of smart contract transactions in the ethereum blockchain platform. ACM Transactions on Software Engineering and Methodology (TOSEM). 1-38. Read on 07.03.2023. http://www.cse.yorku.ca/~zmjiang/publications/tosem2020_zarir.pdf

Zhou, L., Qin, K. & Gervais, A. 2021. A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. Cornell University. Read on 03.03.2023.
https://arxiv.org/abs/2106.07371

Zhou, L., Qin, K., Torres, C., Le, D. & Gervais, A. 2021. High-frequency trading on decentralized on-chain exchanges. In 2021 IEEE Symposium on Security and Privacy (SP). Cornell University. 428-445. Read on 11.03.2023.
https://arxiv.org/pdf/2009.14021.pdf?utm_campaign=ConsenSys%20Diligence&utm_source=hs_email&utm_medium=email&_hsenc=p2ANqtz-9syjEr6eOfKaTzHLqmYJYnNrQbNAA4bYsCZamlMBjA8f65_pfvV8c35FsfKbtbtwByA6Ko

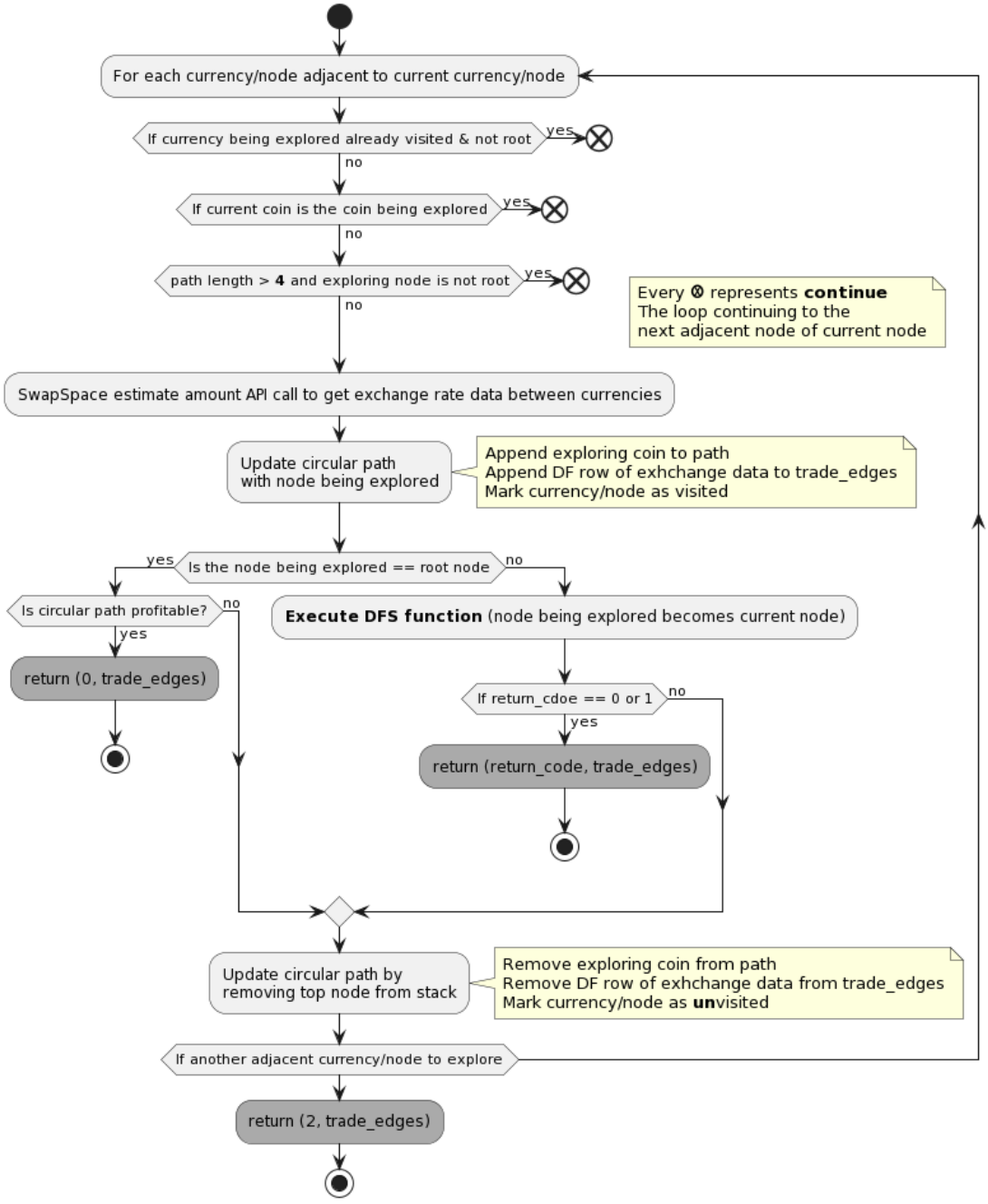
APPENDICES

Appendix 1. SwapSpace aggregating BTC-ETH exchange rates


```
[
  {
    "id": "cbc1b41d-da4b-3e39-b4ab-a56e4fa2b865",
    "supportRate": 2,
    "duration": "3",
    "min": 0.00777151,
    "max": 0.46875816,
    "fixed": true,
    "partner": "switchain",
    "exists": true,
    "fromAmount": 0.1,
    "fromCurrency": "btc",
    "fromNetwork": "btc",
    "toAmount": 1.3064042117483,
    "toCurrency": "eth",
    "toNetwork": "eth"
  },
  {
    "id": "",
    "supportRate": 3,
    "duration": "17-36",
    "min": 0.00045906,
    "max": 3.1861559,
    "fixed": false,
    "partner": "fixedfloat",
    "exists": true,
    "fromAmount": 0.1,
    "fromCurrency": "btc",
    "fromNetwork": "btc",
    "toAmount": 1.3408817,
    "toCurrency": "eth",
    "toNetwork": "eth"
  },
...
]
```


Snippet of SwapSpace API response returning an array of all exchange options (DEXes and CEXes) for exchanging BTC-ETH. Given the amount of Bitcoin being sent, the array gives all the partnered exchanges and the amount of Ethereum that would be received from each exchange.

Appendix 2. Activity Diagram of DFS implementation in the program



Appendix 3. SwapSpace response for creating a token exchange


```
{
    "id": "k93fHaj7glmT",
    "partner": "fixedfloat",
    "fixed": false,
    "timestamps": {
        "createdAt": "2023-02-20T17:23:07",
        "expiresAt": "2023-02-20T17:53:07"
    },
    "from": {
        "code": "eth",
        "network": "eth",
        "amount": 1,
        "address": "0xd64239c98dd0665f6cd6f11acd185ce23a4bb822",
        "extraId": "",
        "transactionHash": ""
    },
    "to": {
        "code": "shib",
        "network": "erc20",
        "amount": 125304110,
        "address": "0xEc20a9BFE04Bec61A9fAaaB7014E7644AFb23ed6",
        "extraId": "",
        "transactionHash": ""
    },
    "rate": 125304110,
    "status": "waiting",
    "confirmations": -1,
    "refundExtraId": "",
    "blockExplorerTransactionUrl": {
        "from": "",
        "to": ""
    },
    "blockExplorerAddressUrl": {
        "from":
"https://blockchair.com/ethereum/address/0xd64239c98dd0665f6cd6f11acd185ce23a
4bb822?from=swapspace",
        "to":
"https://blockchair.com/ethereum/address/0xEc20a9BFE04Bec61A9fAaaB7014E7644AF
b23ed6?from=swapspace"
    }
}
```


The SwapSpace API response for creating the exchange between ETH-SHIB token pair through the FixedFloat exchange platform. Upon creation of the exchange on SwapSpace, a wallet address is provided where the base currency (ETH) is required to be sent so the swap can occur.

Appendix 4. Responses for a detected 4-token abnormal arbitrage

page (1/4)


```
{
   "id": "vw-rohF_mIlH",
   "partner": "changehero",
   "fixed": false,
   "timestamps": {
       "createdAt": "2023-03-09T10:07:53",
       "expiresAt": "2023-03-09T11:07:53"
   },
   "from": {
       "code": "eth",
       "network": "eth",
       "amount": 1,
       "address": "0x3a3f64d58a80600f58d9b7117b6ed675c746154d",
       "extraId": "",
       "transactionHash": ""
   },
   "to": {
       "code": "zrx",
       "network": "erc20",
       "amount": 7117.041500984951,
       "address": "0xEc20a9BFE04Bec61A9fAaaB7014E7644AFb23ed6",
       "extraId": "",
       "transactionHash": ""
   },
   "rate": 7117.041500984951,
   "status": "waiting",
   "confirmations": -1,
   "refundExtraId": "",
   "blockExplorerTransactionUrl": {
       "from": "",
       "to": ""
   },
   "blockExplorerAddressUrl": {
       "from":
"https://blockchair.com/ethereum/address/0x3a3f64d58a80600f58d9b7117b6ed675c7
46154d?from=swapspace",
       "to":
"https://blockchair.com/ethereum/address/0xEc20a9BFE04Bec61A9fAaaB7014E7644AF
b23ed6?from=swapspace"
   }
}
```


Starting the 4-part exchange with 1 ETH for 7117.04 ZRX on ChangeHero.

(continues)

Appendix 4. Responses for a detected 4-token abnormal arbitrage

(page 2/4)


```
{
   "id": "9PVXadmlW2UC",
   "partner": "changelly",
   "fixed": false,
   "timestamps": {
       "createdAt": "2023-03-09T10:07:57",
       "expiresAt": "2023-03-09T11:37:57"
   },
   "from": {
       "code": "zrx",
       "network": "erc20",
       "amount": 7117.041500984951,
       "address": "0x160e2dcd3aa610e031bc795f26b8cea65f0bd247",
       "extraId": "",
       "transactionHash": ""
   },
   "to": {
       "code": "wbtc",
       "network": "erc20",
       "amount": 0.07334122,
       "address": "0xEc20a9BFE04Bec61A9fAaaB7014E7644AFb23ed6",
       "extraId": "",
       "transactionHash": ""
   },
   "rate": 0.000010305015081034734,
   "status": "waiting",
   "confirmations": -1,
   "refundExtraId": "",
   "blockExplorerTransactionUrl": {
       "from": "",
       "to": ""
   },
   "blockExplorerAddressUrl": {
       "from":
"https://blockchair.com/ethereum/address/0x160e2dcd3aa610e031bc795f26b8cea65f
0bd247?from=swapspace",
       "to":
"https://blockchair.com/ethereum/address/0xEc20a9BFE04Bec61A9fAaaB7014E7644AF
b23ed6?from=swapspace"
   }
}
```


Swapping 7117.04 ZRX for 0.0733 WBTC on Changelly.

Appendix 4. Responses for a detected 4-token abnormal arbitrage

(page 3/4)


```
{
   "id": "CVbrCgVvKvIg",
   "partner": "swapuz",
   "fixed": true,
   "timestamps": {
       "createdAt": "2023-03-09T10:08:00",
       "expiresAt": "2023-03-09T13:08:00"
   },
   "from": {
       "code": "wbtc",
       "network": "erc20",
       "amount": 0.07334122,
       "address": "0xded23c676BeF1Afd03923557019e214D235e9adF",
       "extraId": "",
       "transactionHash": ""
   },
   "to": {
       "code": "uni",
       "network": "erc20",
       "amount": 646.5902133616008,
       "address": "0xEc20a9BFE04Bec61A9fAaaB7014E7644AFb23ed6",
       "extraId": "",
       "transactionHash": ""
   },
   "rate": 8816.1911318301066,
   "status": "waiting",
   "confirmations": -1,
   "refundExtraId": "",
   "blockExplorerTransactionUrl": {
       "from": "",
       "to": ""
   },
   "blockExplorerAddressUrl": {
       "from":
"https://blockchair.com/ethereum/address/0xded23c676BeF1Afd03923557019e214D23
5e9adF?from=swapspace",
       "to":
"https://blockchair.com/ethereum/address/0xEc20a9BFE04Bec61A9fAaaB7014E7644AF
b23ed6?from=swapspace"
   }
}
```


Swapping 0.0733 WBTC for 646.59 UNI on Swapuz.

Appendix 4. Responses for a detected 4-token abnormal arbitrage

(page 4/4)


```
{
   "id": "TnOkUe3EjlWl",
   "partner": "changelly",
   "fixed": false,
   "timestamps": {
       "createdAt": "2023-03-09T10:08:03",
       "expiresAt": "2023-03-09T11:38:03"
   },
   "from": {
       "code": "uni",
       "network": "erc20",
       "amount": 646.5902133616008,
       "address": "0xfca87847c627e582dc20537e0b4b09a82fb7b320",
       "extraId": "",
       "transactionHash": ""
   },
   "to": {
       "code": "eth",
       "network": "eth",
       "amount": 2.46018496,
       "address": "0xEc20a9BFE04Bec61A9fAaaB7014E7644AFb23ed6",
       "extraId": "",
       "transactionHash": ""
   },
   "rate": 0.00380485956817933,
   "status": "waiting",
   "confirmations": -1,
   "refundExtraId": "",
   "blockExplorerTransactionUrl": {
       "from": "",
       "to": ""
   },
   "blockExplorerAddressUrl": {
       "from":
"https://blockchair.com/ethereum/address/0xfca87847c627e582dc20537e0b4b09a82f
b7b320?from=swapspace",
       "to":
"https://blockchair.com/ethereum/address/0xEc20a9BFE04Bec61A9fAaaB7014E7644AF
b23ed6?from=swapspace"
   }
}
```


Swapping UNI for ETH on Changelly, finishing the circular trade and ultimately showing the end amount of ETH. Initially starting the circular trade with 1 ETH and ending with 2.46 ETH.