PAPER_Smart_Routing_Multi_Swap_Optimization

January 4, 2022

1 Problem Description

Ref Finance is the first and premier DEX on the Near protocol. The Ref team is continually working to add features and to improve the User Experience for Ref users. [1]

One key feature of Ref Finance has been the ability to create multiple liquidity pools for any given token pair. This feature brings a lot of opportunities as well as some interesting challenges. For example, suppose you want to swap Token A for Token B, and that there are two A-B pools available for the swap. That is, there are two pools, each of which has reserves of Token A and Token B to facilitate trades at a price determined by the constant-product market maker (CPMM) formula. Depending on the composition of each pool and the amount of the swap, you may find a better price in one pool or another, or by splitting the swap between the pools. The Ref UI has recently implemented an optimal "parallel swap" that determines the best allocation between parallel pools. [2] This parallel swap is applied automatically where it is applicable, in order to give the user the best value for the transaction (that is - to maximize the amount of Token B to be received). However, the current state of the Ref UI requires at least one pool to exist with reserves of Token A and Token B in order to facilitate direct trades between the tokens.

This leads us to define the two primary problems we address in this white paper, along with steps towards solving them:

1. Enable users to trade directly among more token assets in the Ref network.
2. Ensure a good price for the trades performed.

1.1 Motivation

At present, the only way to trade a given Token for another token is if a particular pool exists with reserves of each token. However, in the absence of a pool to facilitate a direct trade between a given pair of tokens, there might be a series of trades (multiple swaps through the pool network) to allow an effective trade between the two desired tokens.

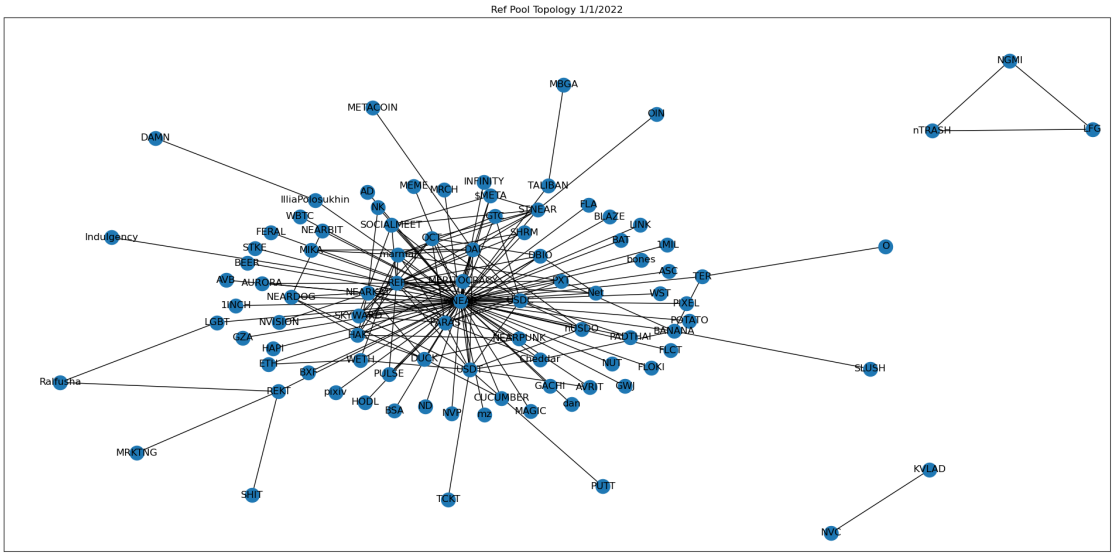
It is possible, through a series of manual trades, to perform multiple swaps, beginning with one pool, then taking the resulting output from the first pool to trade into the second pool, and so on, until a pool is reached with the desired output token. However, this process adds friction to the user experience, and thus could be taking away from the potential trades.

A more streamlined approach would allow the user to trade between any two tokens on the Ref Finance network, without the user having to worry about which pools to use or the underlying implementation of the series of transactions. Under the hood, this would transform the problem of having a pool for a token pair into a network-connectedness problem, which, given the net-

work topology of the Ref Finance pools we will show here, allows for a much simpler and reliable experience for the user.

1.2 The Ref Finance Network as a Graph

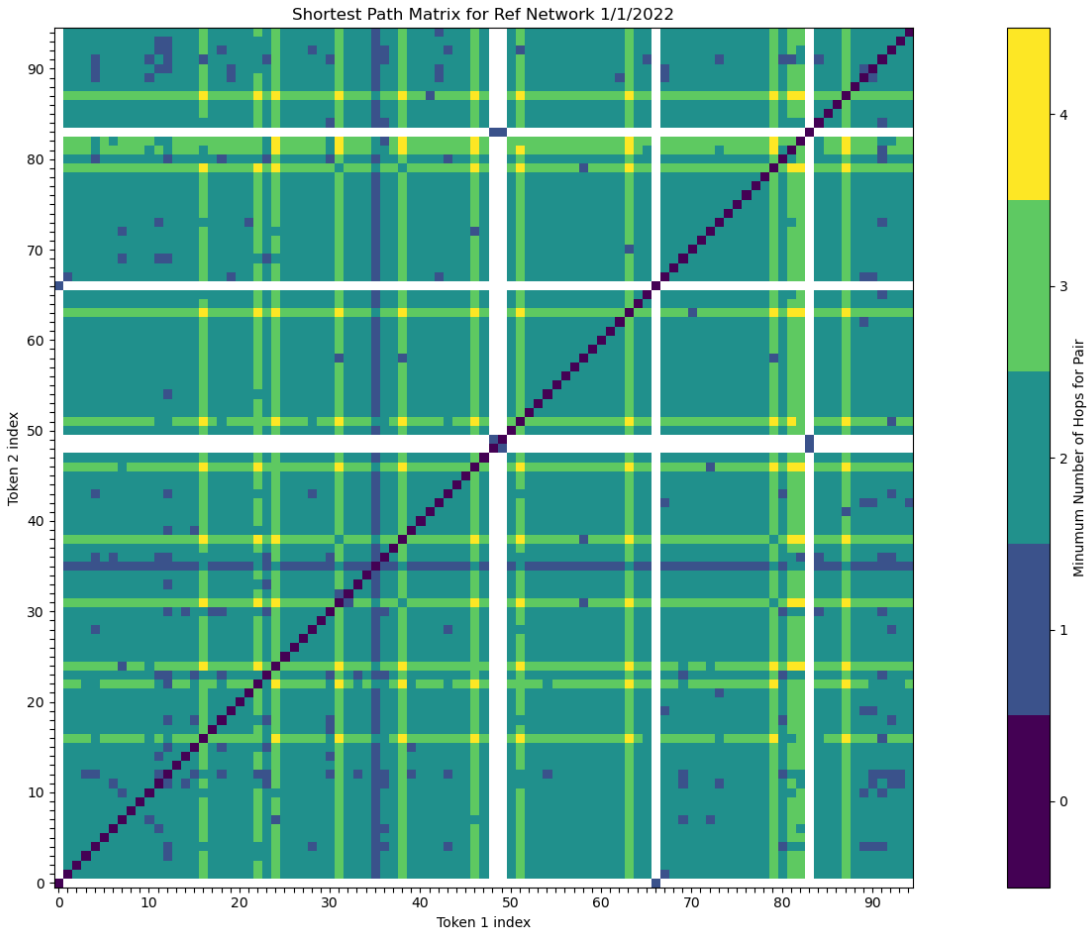
One way to model the collection of Ref Finance liquidity pools is as a mathematical graph. Graphs in mathematics consist of nodes and edges, and can be used to represent networks ranging from internet traffic and routing to supply chain logistics.



Above is a depiction of the topology of the non-zero-liquidity swap pools for Ref Finance, as of 1/1/2022. Each node (circle) in the graph represents a token, and each edge (line) in the graph represents one or more pools that facilitate swapping between the connected pair of tokens. That is, if a line exists between two tokens, there is a direct swapping pool between them. If there is not a line between the two tokens, then there is currently no pool to facilitate a direct swap between the two assets. Not that this visualization could provide an incentive for liquidity providers to instantiate their own pools between pairs.

By performing multiple swaps (or multiple hops in the graph network), we can facilitate trades between a much large collection of tokens programatically, without requiring the user to manually perform a series of swaps themselves. Note, on the right side of the image are a few tokens that form their own sub-graphs, but are disconnected from the main graph. For these tokens, unless a new connector pool is added, there is no way to perform multi-hops to facilitate trades with the rest of the network. They are essentially isolated from the rest of the network and therefore unreachable.

One way to think about the network graph is in terms of the minimum "distance", or minimum number of network hops, required to trade an arbitrary pair of assets. We capture this data below in a matrix of the minimum number of hops between a given pair of tokens:



The matrix is symmetric about the diagonal (the dark blue diagonal going from bottom left to top right in the figure), which is to be expected, since the ability to swap a between a pair of tokens in a pool is bi-directional. That is, if there exists a series of swaps among, say, pools $1 \to 2 \to 3$, then there also must exist a possible series of swaps going the other way (from pools $3 \to 2 \to 1$). The blue cross at token index 35 near the center of the matrix (which implies a certain token has 1-hop connection to most other tokens in the network) represents the *wrap.near* token. The series of white crosses on the figure represent the sub-networks mentioned above. These tokens are not connected by any number of hops to the main network nodes, and are therefore only reachable from within their own sub-network nodes.

The predominance of the dark-green color (corresponding to connectedness of 2 hops) implies qualitatively that most of the network is reachable through only 2 hops. We will show the exact statistics below.

The following is a table of summary statistsics for how many pairs in the network are connected by a certain minimum number of hops. Other than the few tokens mentioned above that are disconnected from the primary connected network of pools, which we will ignore for now, all other tokens are reachable by 4 swaps (network hops) or less.

Minimum # Hops	Frequency in Network	Percentage of Pairs in Network
1	177	4.4 %
2	2916	72.7 %
3	862	21.5 %
4	54	1.3 %

Currently, there are only 177 single-hop pairs in the Ref Finance network. This represents the current state of the Ref network in terms of possible direct swaps the user can currently make. However, note the substantial number (2916) of token pairs connected by 2 hops.

> Stated another way, by introducing only a double-hop, we will increase the number of direct swaps accessible by the Ref users by more than 1600 %!

From the above table, we can see that the connectedness of the network is such that about 77% of the network pairs are connected by 2 hops or less.

If you extend the consideration to allow for 3 hops, then that covers almost 99% of the network pairs.

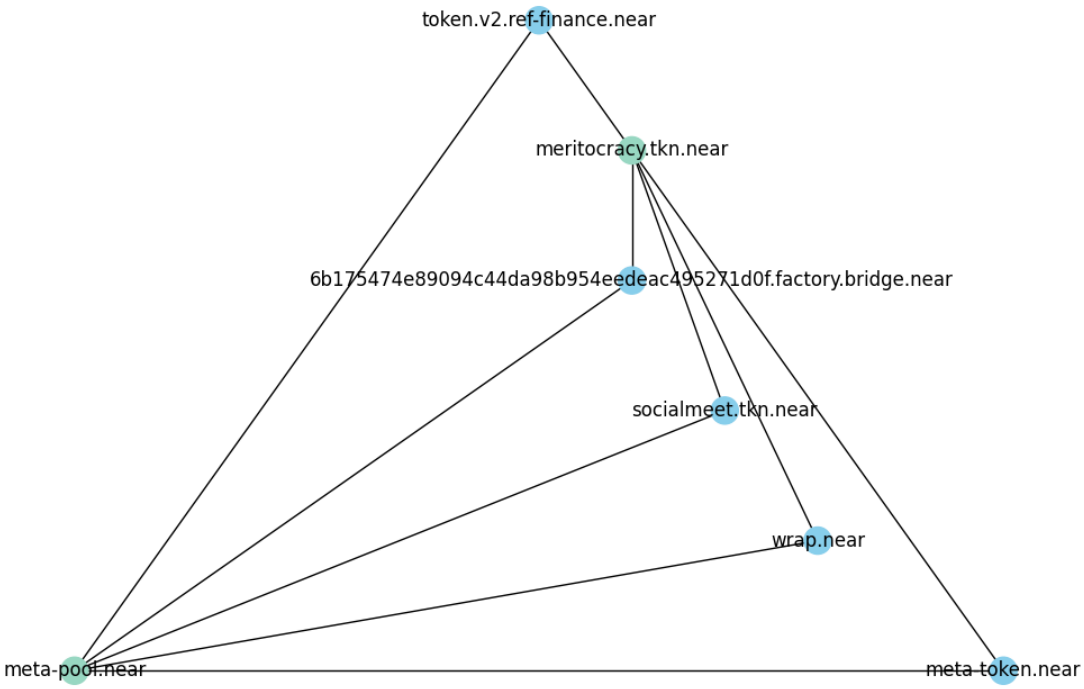
With more hops comes more algorithmic complexity and more computation time. However, substantial progress towards solving problem (1) of the paper can be achieved with only slightly more complexity than the parallel swap algorithm.

As such, this case (2 hops or less) will be the primary focus of this white paper.

1.3 Graph Topology for Single-Hop and Double-hops

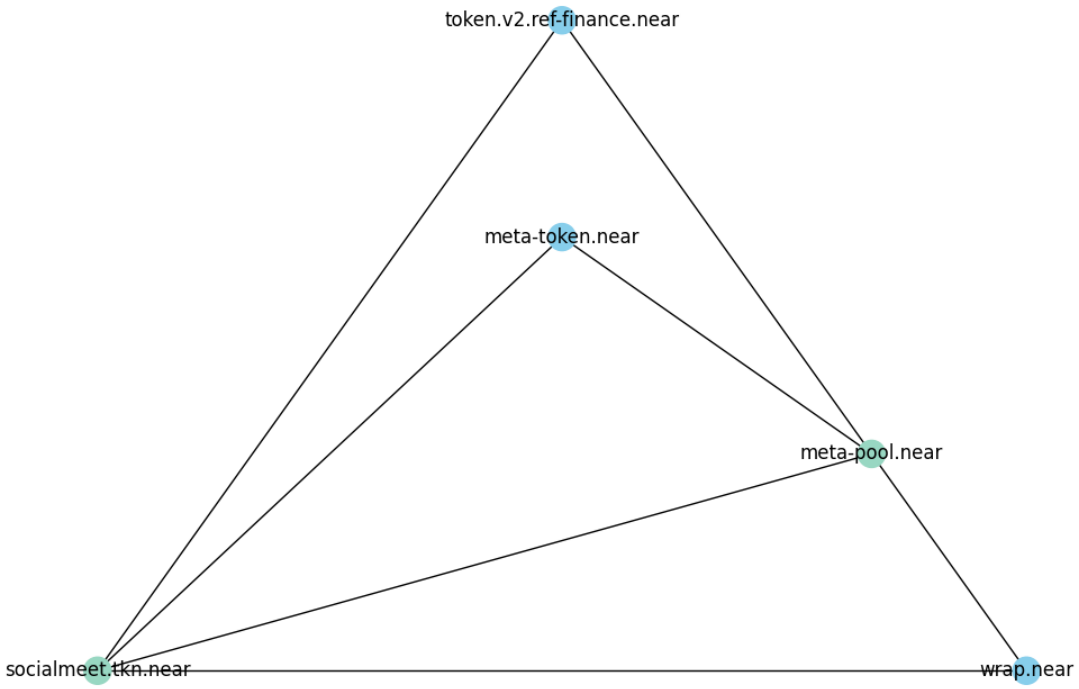
Consider the set of tokens taken from the Ref Finance network of pools: * `meta-pool.near` * `meritocracty.tkn.near` * `socialmeet.tkn.near`

If we consider two tokens from this set, we can determine all of the 1-hop and 2-hop path connections between them. For example, in the figure below, we consider the pair of `meta-pool.near` and `meritocracy.tkn.near`, which have double-hops, but no single-hops (that is, no direct-swap pools):



Here, ther are 5 intermediary tokens shared as common nearest-neighbors to both `meta-pool.near` and `meritocracy.tkn.near`.

Next, we consider below the pair of tokens `meta-pool.near` and `socialmeet.tkn.near`. Here, there are both single-hop paths and double-hop paths between the tokens.



But for all of these cases, what routes give the user better value? At first, it would seem that a single-hop would possibly provide better value, since there is only one fee application in this case (rather than doubling up on fees if two hops are taken). But, there can be cases where the value mismatch between the pools is such that a double-hop could provide better value than a single-hop.

To help in solving problem (2) of the paper, we propose an extension of the parallel swap algorithm, which makes use of the mathematical technique of Lagrange Multipliers, to determine the optimal allocation of trades among these paths. But first, we'd like to give an overview of the concept of optimization in general, and lay out a few of the possible methods of tackling the problem of giving value to users through a good token price.

2 Introduction to Optimization

2.1 Calculus

The mathematical field of optimization is dominated by calculus. Students who have completed Calculus 1 have received the training needed for most basic optimization problems. The goal is to take the derivative of the desired function (which we call the objective function) and set it to zero. Everywhere the derivative is zero represents a maximum or a minimum of the function. We use the term extreme in place of maximum or minimum, and it is usually straightforward to determine if an extreme is a maximum or a minimum.

2.2 Constraints and Lagrange Multipliers

Sometimes there are constrained optimization problems. The calculus of taking the derivative and setting it to zero still applies, but with a notable modification. Since the constraint must be accounted for, and it must be a part of the derivative, it is necessary to write the constraint as an expression that should equal zero. We can add the expression multiplied by a term λ known as a Lagrange Multiplier and then take our derivative, setting it to zero. Almost magically, the optimization is solved, including correct values of all the constraints. The Lagrange Multiplier is used to enable the Parallel Swap feature on Ref Finance. [2]

2.3 Convex Optimization

In general, optimization can become very complicated for a large number of constraints and for complicated objective functions. If constraints or the objective function are complicated, then the path to the solution will be complicated as well. In that case, the time and resources required to solve the problem grow rapidly as the number of variables increases. It is better for hand calculations and for computers to solve a special type of optimization, namely a convex optimization problem. This type of problem requires certain restrictions on the objective function and the constraints. Convex problems can be solved quickly and easily by a computer, even with many variables and constraints. Optimizing DeFi seems to be a convex problem; it certainly is for the task at hand. Therefore, we can confidently develop an algorithm and trust that the optimal solution can be found quickly and efficiently. Recently, Angeris *et al.* at Stanford wrote a paper that uses a standard Python convex optimization library, `cvxpy`, to find an optimal net set of trades among a family of liquidity pools of various types. [3] Because their code is open source, we were able to adapt it to model the same parallel swap problem we solved before, and found that their method gives identical results to ours.

2.4 Linear Programming

If the objective function and the constraints are all linear functions, then the optimization is linear. Linear optimization is a special case of convex optimization, for a faster solution. Constant Product Market Makers are definitely not linear; however, users generally want to trade within an approximately linear regime (as little slippage as possible), and it may be possible to use linear approximations within limitations to get a "good enough" solution in a much more reasonable amount of time. As our algorithms are implemented, it may be desirable to employ linear approximations. This will help the Ref team include reasonable constraints directly into the optimization, such as

- relative liquidity
- number of intermediate swaps (currently limited to 1, but could potentially be higher)
- maximum total number of transactions in the overall swap
- single algorithm to handle parallel swap and multi-swap
- single algorithm capable of CPMM and stable swap

Constraints such as these can be input with hard cutoffs, or as a secondary optimization problem.

2.4.1 But what is a "good enough" solution?

Some optimization problems have an obvious solution. For example, in a Decentralized Exchange with parallel swaps, one pool might have the best price and most liquidity, and the solution will be 100% of the swap in that pool. However, we rely on the computer to find the optimum for the cases

that are not obvious. Even the computer might take some amount of time to crunch the numbers and find a solution. In that period of time, the prices will potentially change due to other investors trading. We recognize for any trade that a slippage tolerance is allowed, of 0.1%, 0.5%, or even 1%. This betrays our knowledge that even after we initiate a trade, the prices can change by that much in essentially no time.

Any trade may fail due to slippage, as discussed above. As the number of trades increases, the probability of failure also increases. Spending more time to get a tiny increase in the outcome might be the factor of whether the entire trade fails or is completed. This factor is not based on any kind of profit percentage, but it should be noted that the amount of time spent to solve the problem should be proportional to the desired outcome.

Gas fees are not included in the optimization algorithm. While gas fees are small on the Near protocol, they are nonzero, and will start to add up, especially for a trade split into many (say, 20-30) swaps. These gas fees comprise a small but nonzero percentage of the swaps.

Consider the complex problem of numerous liquidity pools. Some may have an unfavorable price and/or very little liquidity. The algorithm must still consider all these pools. Imagine that using the single largest pool would yield a return of \$1000, but that a second transaction with a smaller pool could yield \$1000.01 and be selected as the best solution. This would actually be a worse solution, since the gas fees would be higher, and the risk of a failed transaction would be much higher.

The need for a "good enough" solution considers that the true optimum will always have error bars of around 1%. Therefore, if we can reach a solution within 1% below the maximum but in less time (really, 99% of the time would work, but we might be significantly faster) then we can have the best overall outcome and better user experience.

In the field of regression, more variables always gives a better pure result. However, this better numerical result is not always better in real life, and is not always helpful. Several techniques employed in regression are used to get the best result considering the number of variables. The ideas behind these techniques translate into optimization, such that it will be possible to get a good enough result using the smallest number of trades practical, and to improve the outcomes and the experience of Ref users.

2.5 Examples

2.5.1 Optimizing the number of trades

The idea of smart routing (including or in addition to parallel swap) is to allow trading from multiple pools simultaneously to get the best overall price for a trade. This is obviously good if I can swap equal amounts in two pools to reduce the slippage. However, if I have pools with very different liquidity, I might optimize the swap and get only 0.1% better outcome. This is a tiny improvement, especially considering gas fees, increased failure risk due to slippage, and the potential of a transaction succeeding with a slippage tolerance of 0.5% or 1% which might actually be less optimal.

Besides including a constraint for the maximum number of simultaneous swaps, it is very helpful to get the best outcome with the fewest simultaneous swaps. Linear optimization allows for regularization, which is similar to Lagrange Multipliers. Regularization allows for additional optimization by applying a penalty for the number of swaps, meaning a swap would only be added if it contributes

meaningfully to the outcome.

2.5.2 Optimizing the serial and parallel swapping

Similarly, a parallel swap might give a pretty good price while trading in 5 pools. However, a "serial" smart routing swap might actually yield a better price, even going through 2 or more intermediate pools. This would be very tedious to determine manually, but if the equations are set up correctly, it would be a straightforward outcome of the optimization algorithm.

3 Optimizing DeFi

3.1 General Equations

The task is to maximize ΔB_{total} as a function of ΔA_{total}, subject to the constraints,

$$\Delta A_{total} = \sum_i \Delta a_i$$
$$\Delta a_i \geq 0$$

A single pool is characterized by three terms:

- x: the amount of Token X in the pool
- y: the amount of Token Y in the pool
- ρ: the fee for trades in the pool (ie. 0.003 for a 0.3% fee)

Defining $\gamma \equiv 1 - \rho$, we see that

$$\Delta y = \frac{\Delta x \cdot y \cdot \gamma}{x + \Delta x \cdot \gamma}$$

Now we assume pool 1 contains Tokens A and C, and pool 2 contains Tokens B and C. I want to trade Token A for Token B, but I'll have to go indirectly from A to C in pool 1, and then from C to B in pool 2.

First, I'll swap from A to C. I'll define Δc as the amount of C I gain in this first swap, and the same amount I swap into pool 2 afterwards.

$$\Delta c = \frac{\Delta a \cdot c_1 \cdot \gamma_1}{a + \Delta a \cdot \gamma_1}$$

$$\Delta b = \frac{\Delta c \cdot b \cdot \gamma_2}{c_2 + \Delta c \cdot \gamma_2}$$

This is simple enough if there is exactly one path from A to C, and one from C to B. However, if there are multiple pools from A to C, then Δc will be a separate optimal solution with some parallel combination of the pools. Likewise, multiple pools from C to B will require a parallel solution there.

We can extend the parallel swap expression to a more general one that captures both single hops and double-hops, all in terms of the initial amount of Token A traded in, and of the reserve and fee parameters of each pool.

The total amount of ΔB, the amount of Token B that we want to maximize, is given by:

$$\Delta B_{total} = \sum_i \Delta b_i = \sum_{singlehops} \Delta b_i + \sum_{doublehops} \Delta b_i$$

The benefit of this solution is that it would not require the heavy machinery of a convex optimization solver. These solvers are plentiful in python, but are much more scarce in Javascript.

3.2 Algorithm

3.2.1 Interface

We'll start with the needed interface with the Ref Finance front end. To begin, the existing algorithm has a getPools call, with a filter for pools that contain Token A and Token B. Since we are limiting our algorithm to a single intermediate pool, we can simply change the filter to pools that contain Token A **or** Token B.

The current parallel swap algorithm returns a list of transactions, so we want to return the same type of list. If this interface is successful, then no additional front end work will be needed.

3.2.2 Coding

The cvxpy module in python is well-known and widely used for this sort of convex optimization problem. For implementing in JavaScript, the closest package might be one that is limited to linear problems…

- https://github.com/JWally/jsLPSolver
- https://www.npmjs.com/package/javascript-lp-solver

There is another package that might work but it seems to have less functionality???

- https://www.npmjs.com/package/optimization-js
- https://github.com/optimization-js/optimization-js

There is an industry standard non-linear solver, `nlopt`, which has implmentations in multiple languages, including C++, Python, Rust, and a Javascript wrapper:

- https://github.com/freethenation/node-nlopt

4 Path Forward

- Select Optimization Package
- Code Ref Finance problem in whichever package
- Integrate solution into Ref Finance
- Unit Test

5 References

1. https://app.ref.finance
2. P. Gartland & J.D. Jackson. "Parallel Swap Optimization". Sept. 2021.

3. G. Angeris, T. Chitra, A. Evans, & S. Boyd. “Optimal Routing for Constant Function Market Makers”. Dec. 2021.

6 Sample Code

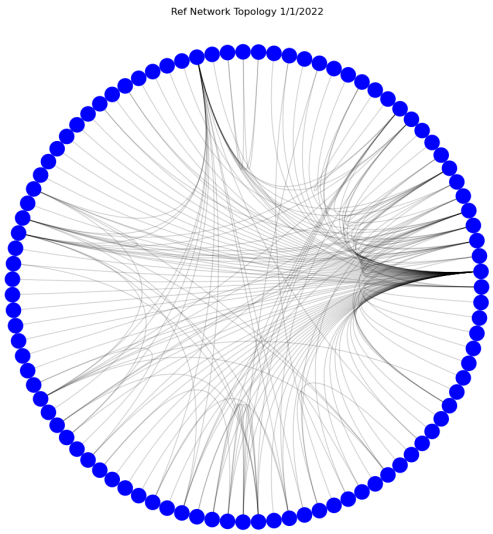
6.1 Sample 1

https://github.com/angeris/cfmm-routing-code/blob/master/arbitrage.py

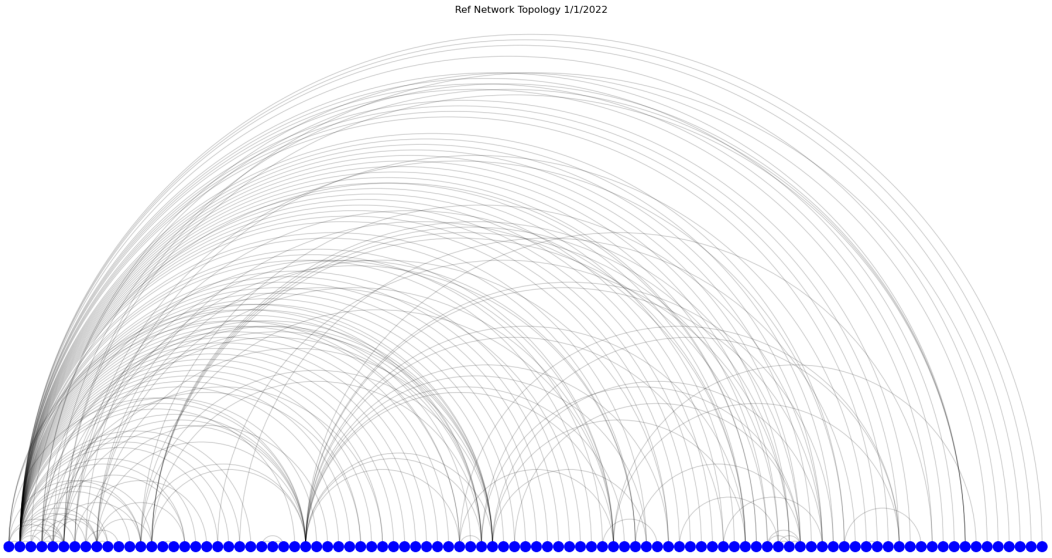
6.2 BACKUP: Extra Plots for Ref Finance Token/Pool Network Topology

Here are a few other graph representations of the network, to give a visual feel for how the nodes are connected and therefore what pools are available.

First, a Circos Plot displays a non-labeled picture of the connectedness of the network. The dense series of branches coming from the right side of the plot stem from the *wrap.near* token.



Next, we display an Arc Plot. This is just another way to see how connected the nodes of the network are.



And finally, we display a circular graph with token symbol labels to further plot the connectedness of the network.