Ref Finance Smart Routing V2 Test Plan

February 26, 2022

1 Ref Finance Smart Routing V2 Algorithm Test Plan

Giddy & Dave

1.1 Introduction

The number of possible pool topologies for the smart routing paradigm is more extensive than that of the parallel swap case.

Below, we outline a list of different pool topologies to test the smart routing algorithms in different ways.

Note, this would require creating at least 68 simple pools and at least 53 Fungible Tokens on the testnet. For the previous testing plan for parallel swap, I manually created the necessary pools and tokens in the CLI. However, such a manual process would be extremely time consuming and could be prone to human error. Therefore, we propose to set up a more automated testing framework to remove some of the manual and reduntant portions of the testing process.

We propose to proceed as follows:

1.1.1 Create the Fungible Token contracts

The first step will be to deploy fungible token contracts for each of the 53 necessary tokens. Upon deployment, a specified number will be transferred to the unit testing user account. The tokens will then be registered in the testnet ref finance contract, and all necessary storage deposits will be paid. While this step can be accomplished by a series of CLI arguments, these CLI arguments can be condensed into a for-loop script and run in a batch file.

1.1.2 Create the simple pools in the specified topology

Further below, we will outline the different pool topologies per token. But once the particular topology is known, then we need to create a simple pool connecting the desired tokens. Once each pool is created, it needs to have a specified ratio of liquidity added to it from the unit testing user account. Similar to the creation of the FT contracts above, this step can be coded into a for-loop script in a batch file.

1.1.3 Pre-Swap Snapshot

Before performing the desired swaps, we need to log the data of the current state of the set of pools we created, along with the FT holdings of the unit testing user account. We can run get_pools

on the ref contract, and we can run ft_balance_of for each of the tokens for the unit testing user account. This can also be coded in a command line script.

1.1.4 Perform the Swaps using the Smart Routing V2 algorithm

This is the most difficult step to automate. However, we have been looking into the Cypress testing framework, which is a unit testing framework that can perform UI testing. Happily, Cypress allows opening up the local Ref Finance server URL, and can programmatically click buttons and type data into the UI and capture outputs.

We have made a proof-of-principle script that loads the local Ref Finance site on localhost, logs into a testnet account, chooses tokens to swap, clicks the inputAmount section, types the input amount, clicks the swap button, and approves the transaction. It then captures the transaction hash URL from the transient pop-up notification and logs the transaction hash to a file. Using this template, we can programmatically performa all the test trades and then just look at the data and transaction hash information at the end. This will eliminate the tedious process of clicking through the trades by hand.

This might be helpful for the Ref team in the future as well, to help with more automated testing.

1.1.5 Post-swap Snapshot

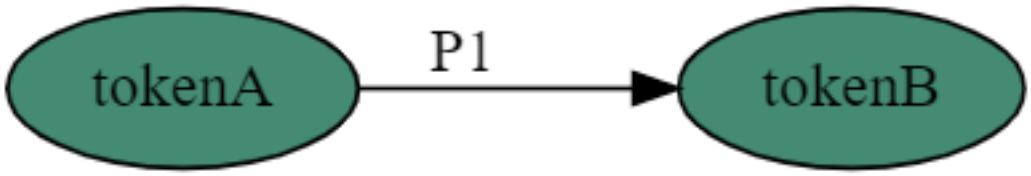
This is identical to the pre-swap snapshot, but will capture the overall state of the pools and the FT holding of the unit testing user account *after* the swaps have been performed.

1.2 Pool Configurations

For each of the configurations below, we associate one or more tests. The first 4 configurations are essentially re-testing the parallel swap functionality. For now, we assume there are limitations to possible solutions, including that there be a maximum of 4 actions across 4 different pools. For topologies that contain more paths/routes than this, we have updated the algorithm to choose only the routes that had the highest allocation in the initial calculation. That is, if there were 4 possible routes, and each route had allocations of 100, 100, 50, and 10, then the algorithm will re-calculate only using the first two 100-allocation routes. Also, we limit the cases where a pool is shared by multiple routes. (See configurations 6,7,8,14,15,16,17,18,19). In these cases, we only choose the highest-allocation route amount the possible shared-pool routes. For some general expectations, we would expect the smart routing V2 to at least match or outperform the direct swap (if it exists for the token pair) and the original parallel swap (if it exists for the token pair).

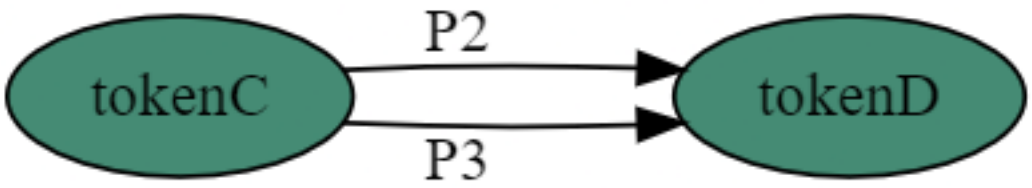
1.2.1 Configuration 1

Single pool, direct swap.



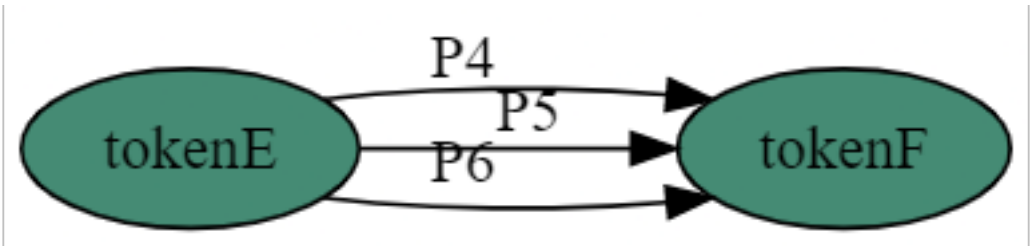
1.2.2 Configuration 2

Double pool, parallel swap.



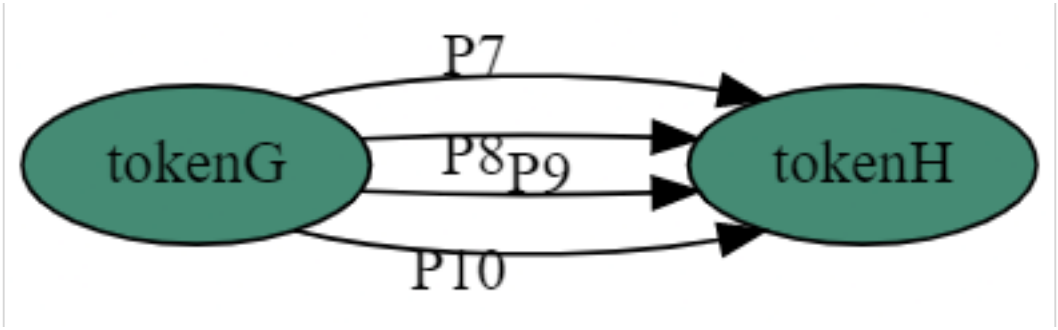
1.2.3 Configuration 3

Triple pool, parallel swap.



1.2.4 Configuration 4

Quadruple pool, parallel swap.



1.2.5 Configuration 5

One intermediate Token, one single route.

1.2.6 Configuration 6

One intermediate token, but two pools on first hop. This will test to ensure that the algorithm only chooses a single route – either Pool 13 –> Pool 15, or Pool 14 –> Pool 15.

1.2.7 Configuration 7

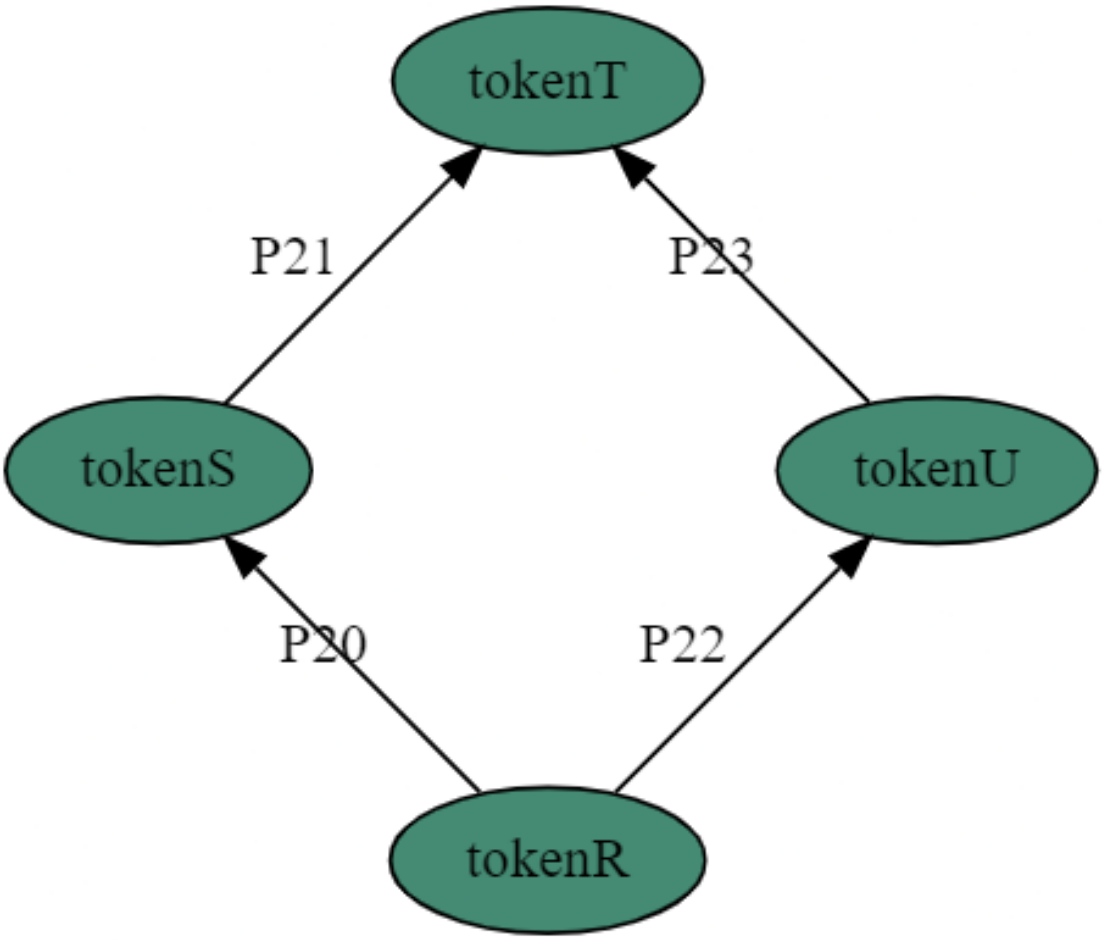
One intermediate token, the reverse of Configuration 6. That is, the second hop has two possible pools. We can use the same set of pools and tokens from Configuration 6 to perform these tests, but just switch the input/output tokens.

1.2.8 Configuration 8

One intermediate token, but two possible pools in first hop, two possible pools in second hop. We expect this configuration to test that the algorithm still only chooses a single route: either Pool 16 –> Pool 18, or Pool 16 –> Pool 19, or Pool 17 –> Pool 18, or Pool 17 –> Pool 19.

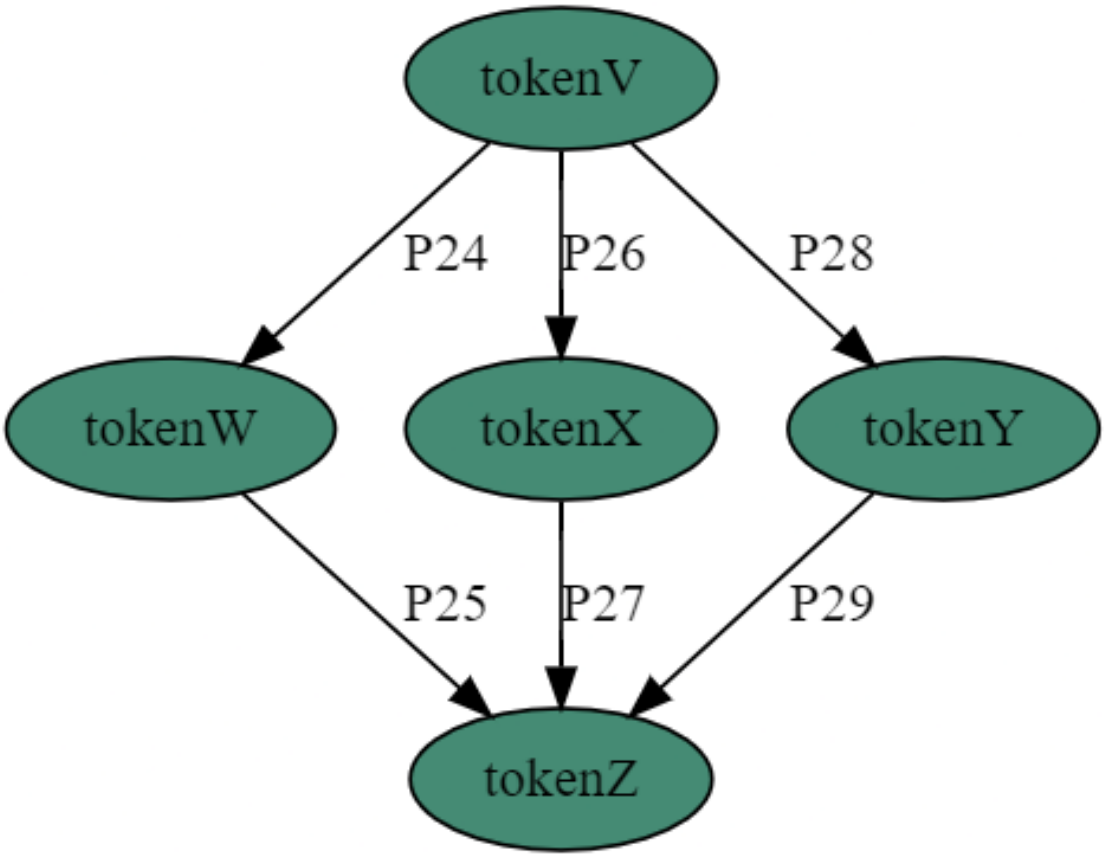
1.2.9 Configuration 9

Two separate intermediate tokens, allow for two independent routes. The algorithm should give the optimal allocation split between the two routes.



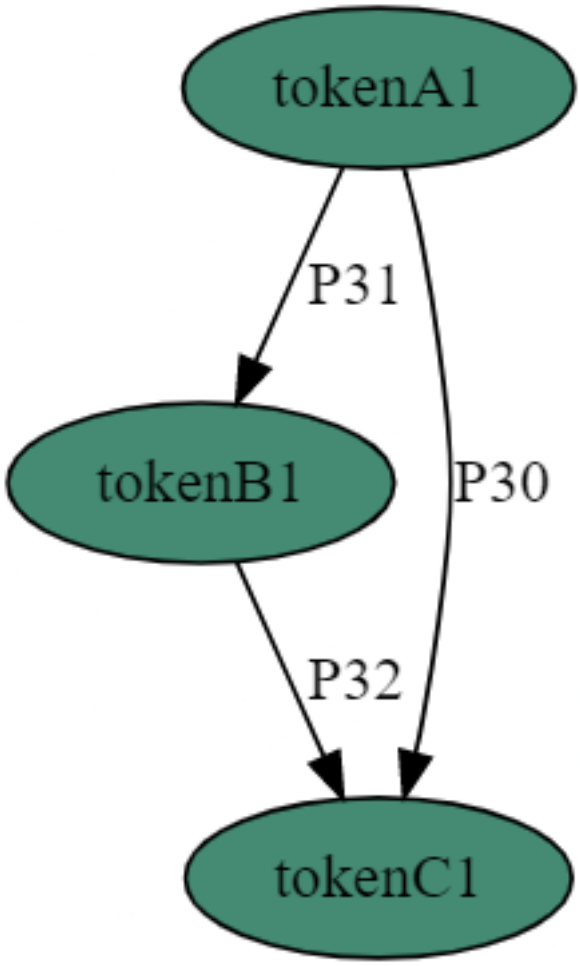
1.2.10 Configuration 10

Three separate intermediate tokens, allow for three independent routes. The algorithm should give the optimal allocation split between the two best routes.



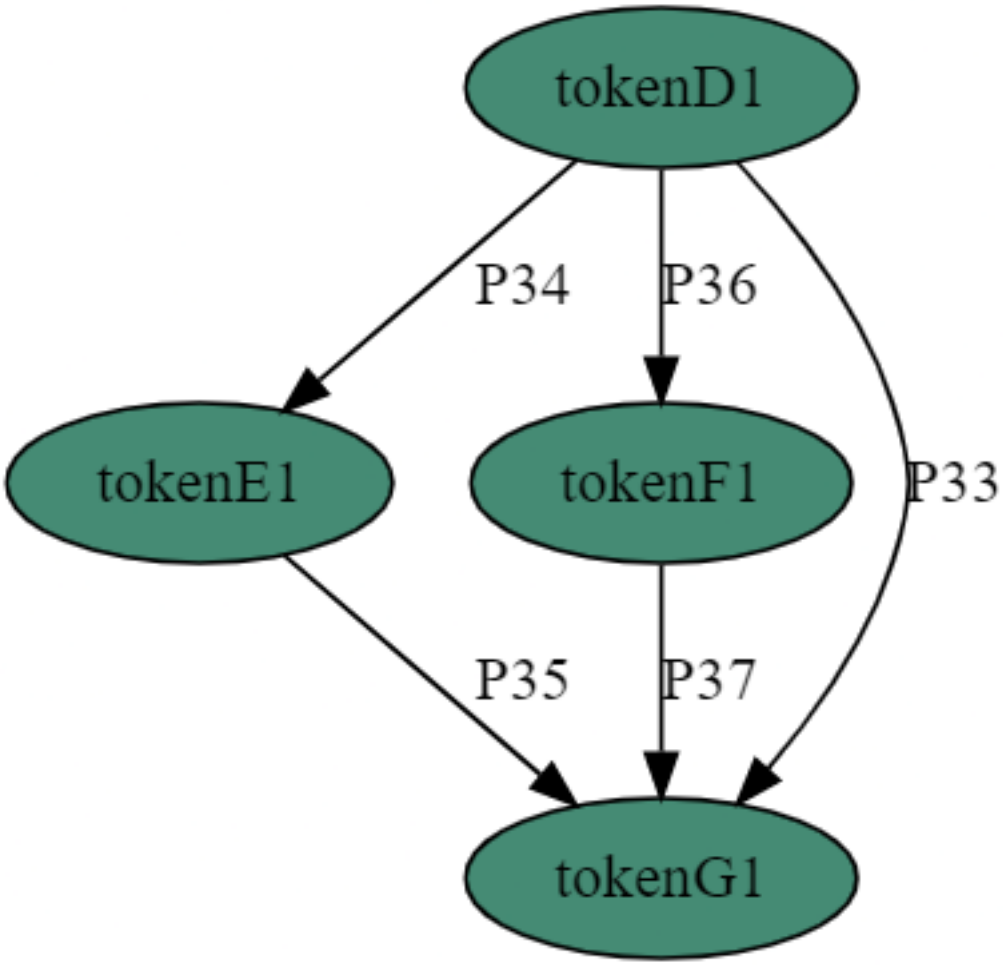
1.2.11 Configuration 11

A hybrid case where there is one direct hop possible, and one route with an intermediate token. The algorithm should find the optimal allocation split between the one-hop (direct) route and the two-hop (on intermediate token) route.



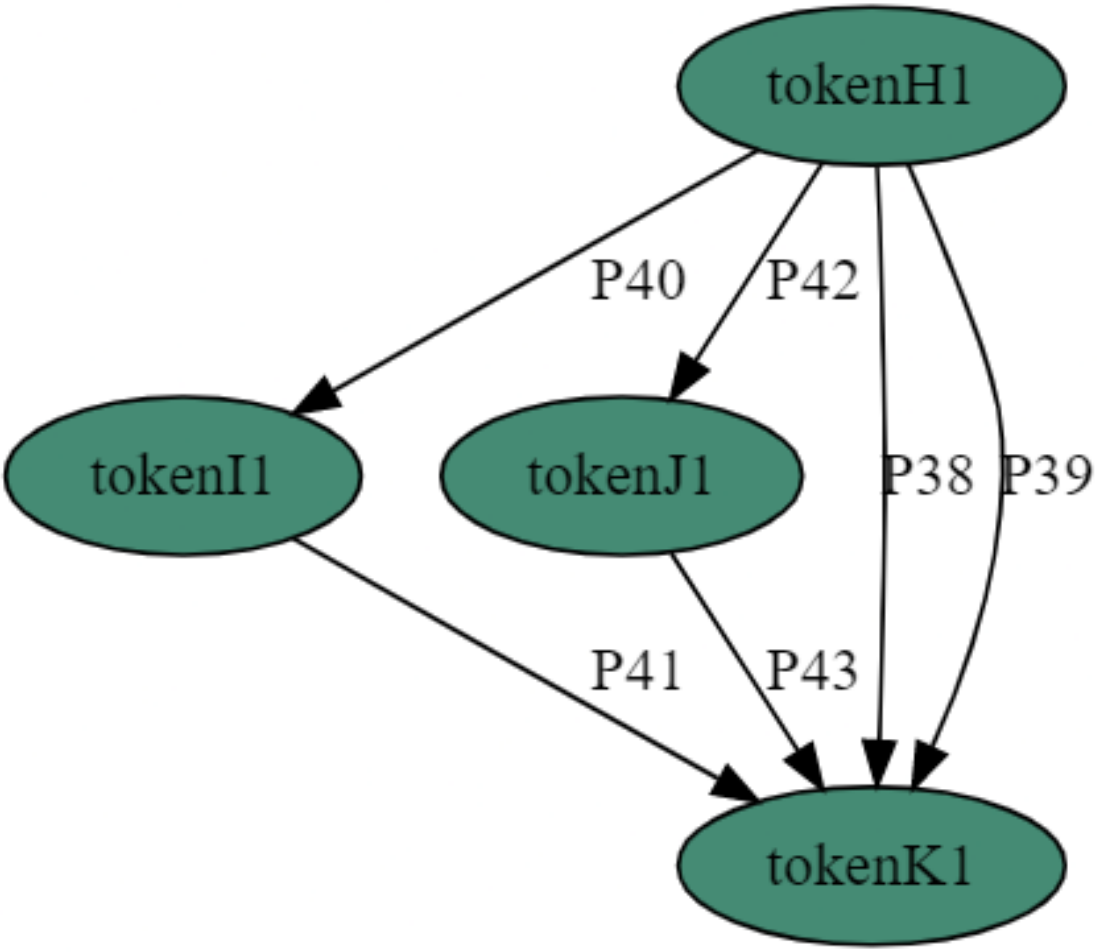
1.2.12 Configuration 12

Another hybrid case where there is one direct hop possible, and two independent routes with an two intermediate tokens. The algorithm should find the optimal allocation split between the following pairs of routes: either Pool 33 and Pool 34 –> Pool 35, or Pool 33 and Pool 36 –> Pool 37, or Pool 34 –> Pool 35 and Pool 36 –> Pool 37.



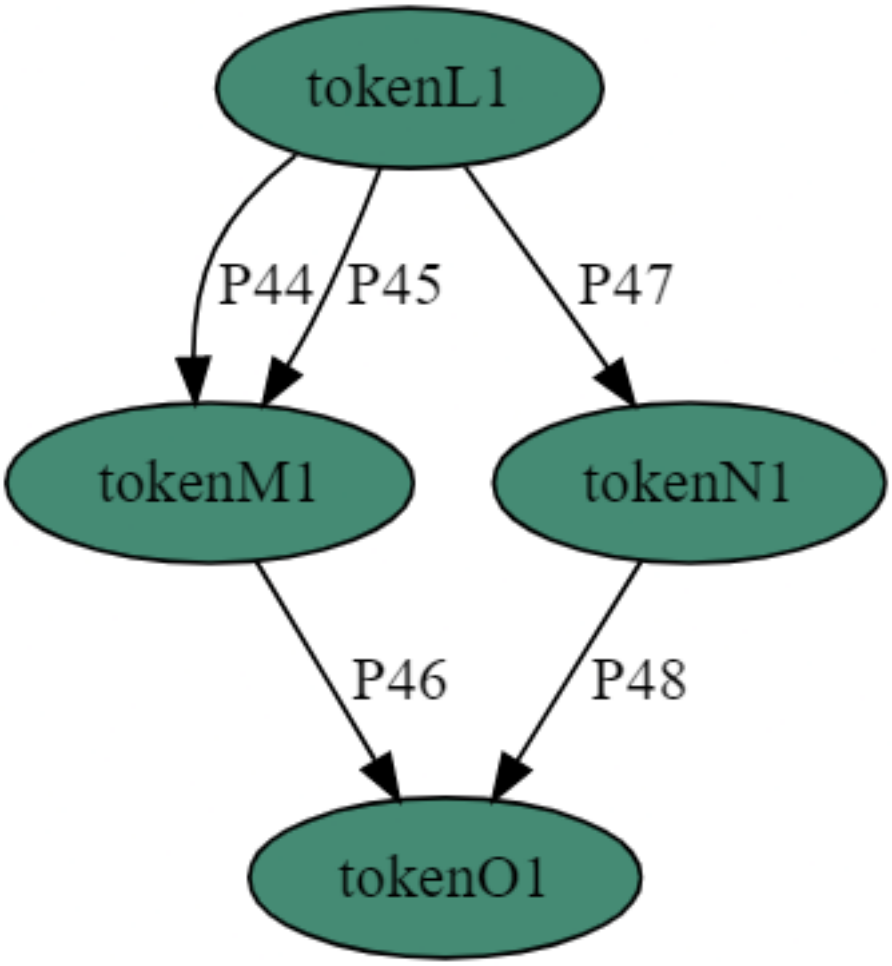
1.2.13 Configuration 13

Another hybrid case similar to configuration 12, where there are two direct hops possible, and two independent routes with an two intermediate tokens. The algorithm should find the optimal allocation split between the combinations of the routes.



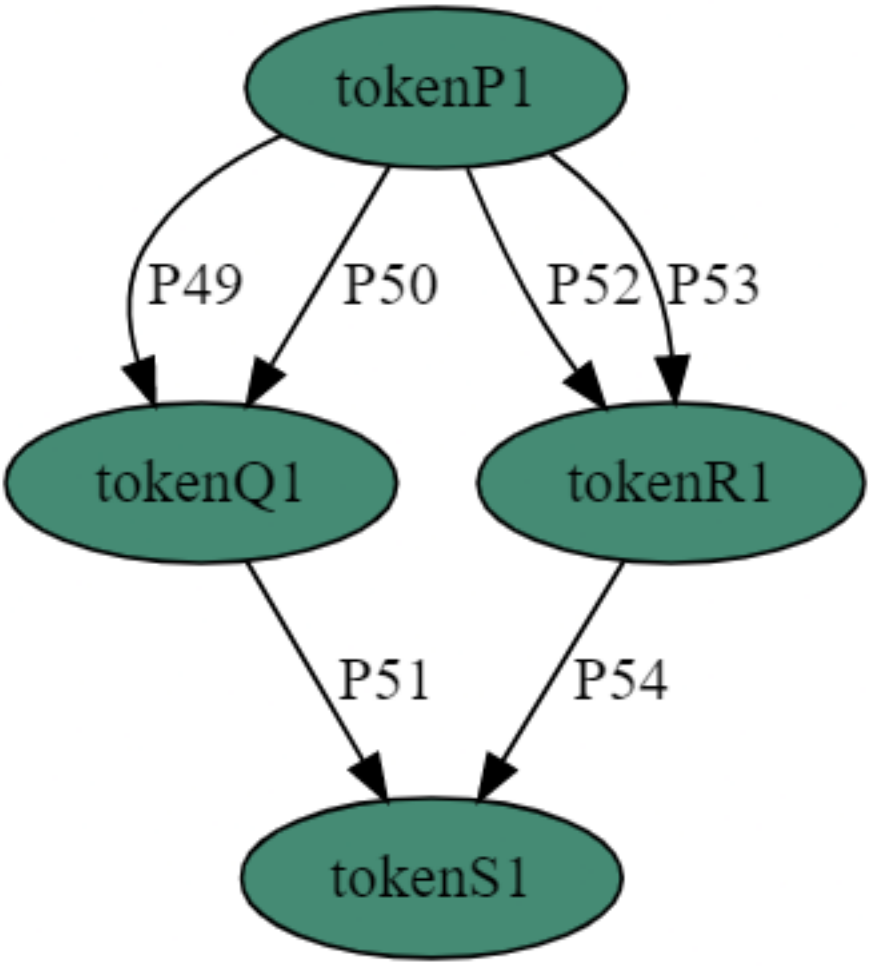
1.2.14 Configuration 14

A case with two intermediate tokens and no direct hops. For one of the intermediate tokens, there are two possible parallel pools. The algorithm should choose the best allocation split between two routes: either Pool 44 –> Pool 46 and Pool 47 –> Pool 48, or Pool 45 –> Pool 46 and Pool 47 –> Pool 48



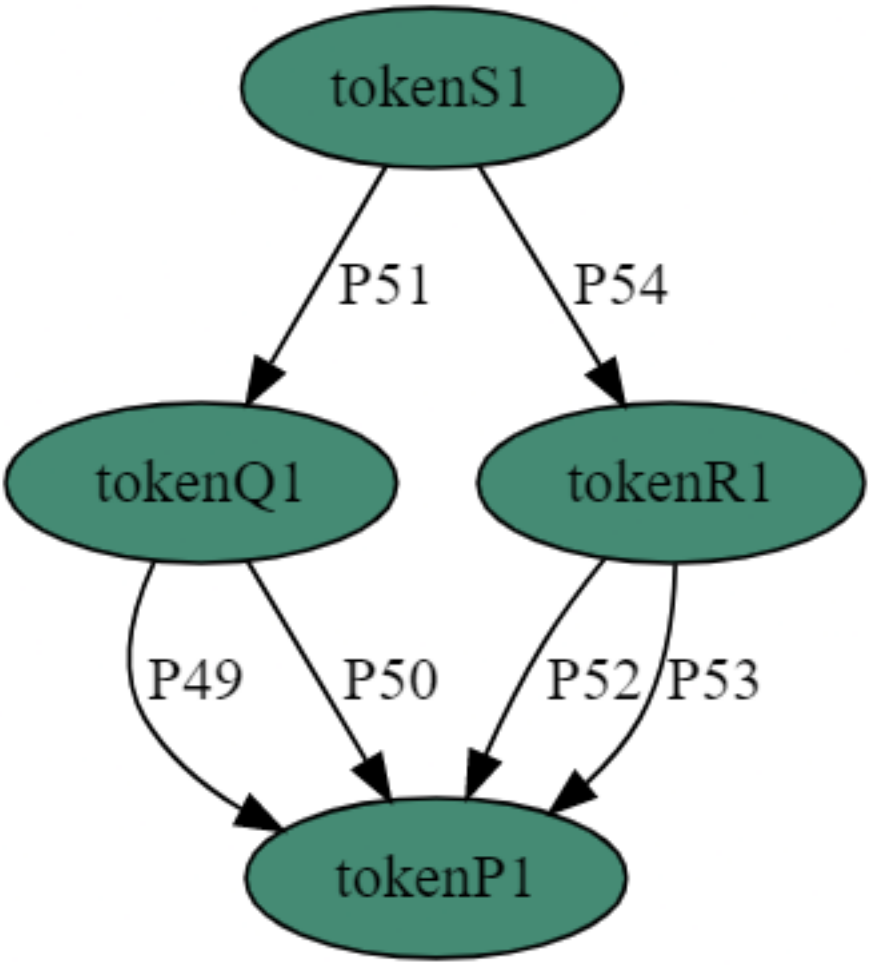
1.2.15 Configuration 15

A case with two intermediate tokens and no direct hops. But for the first hop, there are two possible parallel pools for each intermediate token. The algorithm should choose the best allocation for two routes, with only one of the parallel pools per intermediate token chosen for each of the first hops.



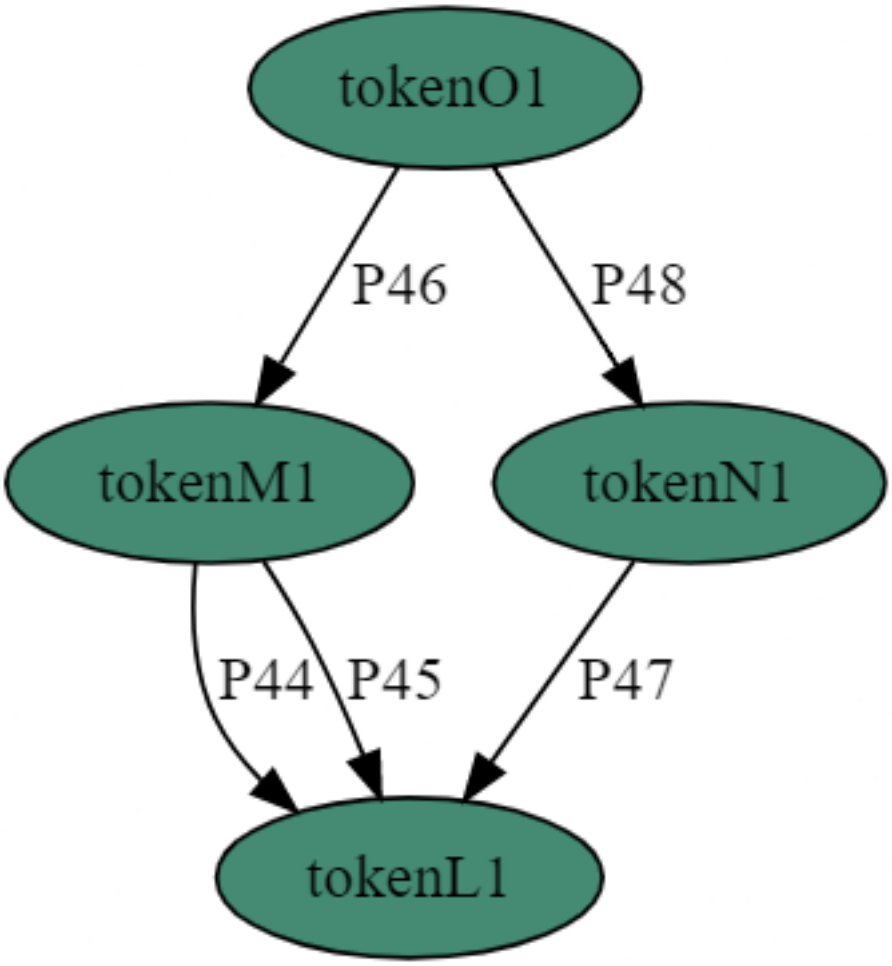
1.2.16 Configuration 16

The inverse of Configuration 15. Can just switch the input/output tokens from Configuration 15.



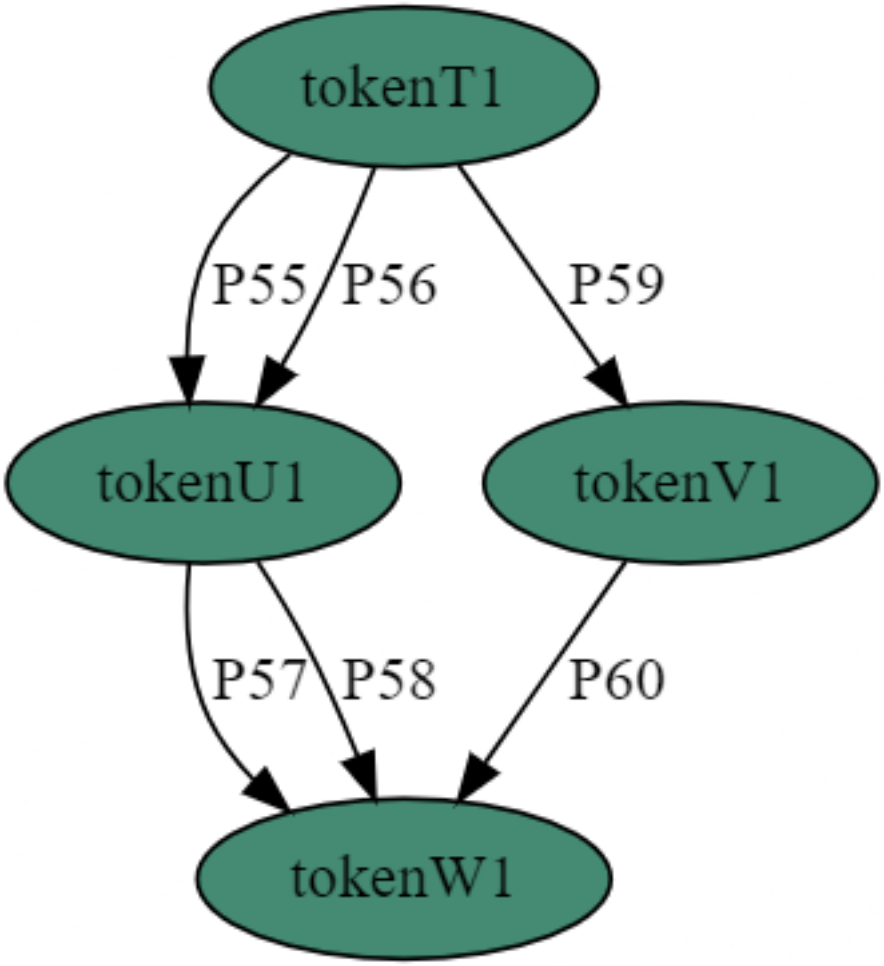
1.2.17 Configuration 17

The inverse case of Configuration 14. Can just switch input/output tokens from Configuration 14.



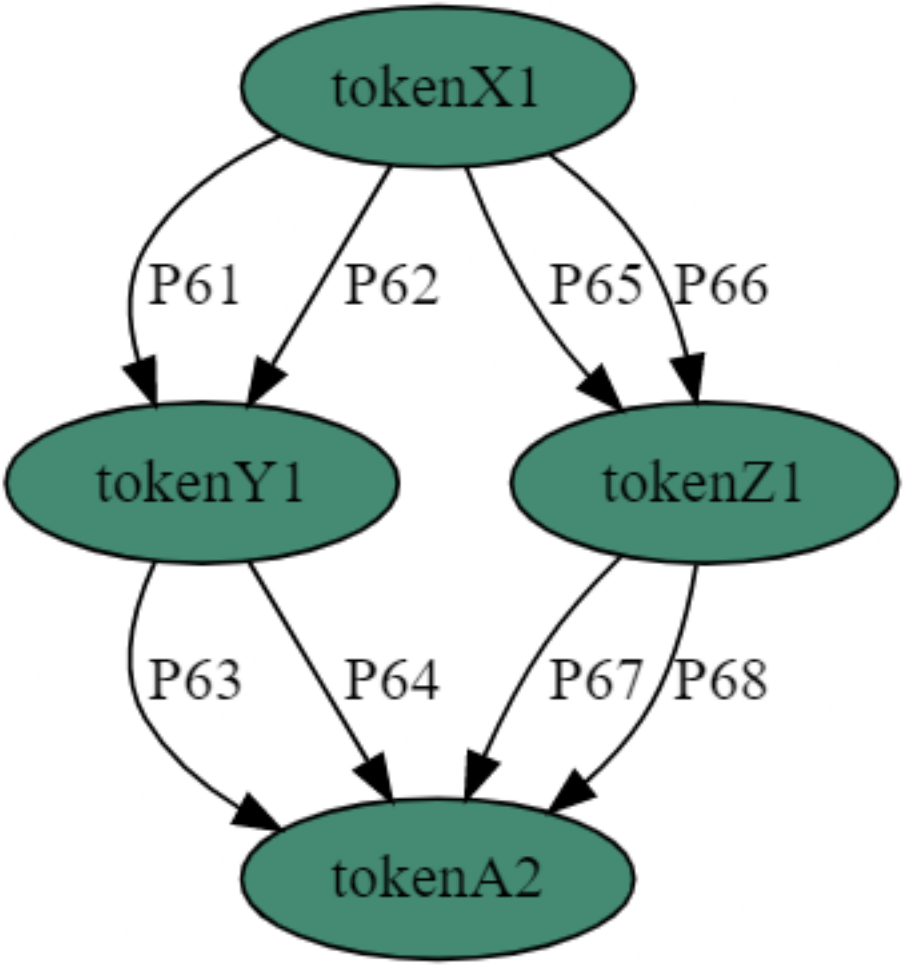
1.2.18 Configuration 18

Two intermediate tokens, no direct hops. Assume for one of the intermediate tokens, for both the first and the second hop, there are two possible parallel pools. The agorithm should choose the best allocation between the non-parallel route and a single route of the parallel case.



1.2.19 Configuration 19

Two intermediate tokens, no direct hops. Assume for each of the hops for each token, there are two possible parallel pools. The algorithm should choose the best allocation split between two routes including each intermediate token.


tokenX1
P61
P62
P65
P66
tokenY1
tokenZ1
P63
P64
P67
P68
tokenA2