Modified Bellman-Ford algorithm for arbitrage searching

Péter Gyimesi

December 2025

1 Overview

The Tycho Searcher processes real-time blockchain data to identify arbitrage opportunities. It receives block updates from a data feed, maintains an up-to-date graph of token exchanges, and applies a modified Bellman-Ford algorithm to detect negative cycles, which correspond to profitable arbitrage paths.

2 Searcher Workflow

Initialization: The searcher initializes its graph data structure and waits for block updates.
Block Update Handling: Upon receiving a new block, the searcher updates the graph with the latest exchange rates and liquidity data.
Arbitrage Detection: The searcher executes the modified Bellman-Ford algorithm to find negative cycles, which indicate arbitrage opportunities.
Result Export: The searcher processes the detected opportunities and can export or log them for further action.

3 Problem

Currently, the algorithm works for the entire graph, and does not handle the updates separately.

We consider a directed graph in which nodes represent tokens or currencies, and each directed edge represents an exchange between two tokens. The program may query the exchange rate of an edge for a fixed amount. The goal is to produce an arbitrage starting from a fixed node and a starting amount.
Input: A directed graph, a starting node, the functions on the edges (which can be treated as a black box), the ratio of the gas cost compared to one token.
Output: A simple cycle, from the starting node, and the initial value.

Each edge is associated with a function, and we can query a point of those functions.
Function input: The ID of the edge, and the amount.
Function output: The amount we get from the other currency, and the cost for that transaction (gas cost) - which is typically a constant.

4 Algorithm

The gas cost is typically a relatively small constant. Therefore, we first ignore it and incorporate it later. The currency-exchange functions f are non-negative, monotone, and concave. Formally, $f(0) = 0$, for $i < j$ we know that $f(i) \leq f(j)$ and $\frac{f(j)}{j} \leq \frac{f(i)}{i}$.

First, let us assume that the currency-exchange functions are linear. Suppose we find a cycle $v_1, v_2, \dots v_n$, with exchange ratios $r_1, r_2 \dots r_n$. This cycle represents an arbitrage opportunity if and only if $r_1 \cdot r_2 \cdot \dots r_n > 1$ or equivalently $ln(r_1) + ln(r_2) \cdots + ln(r_n) > 0$.

The latter problem after multiplying each value by -1 reduces to finding a negative cycle, which can be solved by the Bellman-Ford algorithm.

Empirically, the exchange functions are close to linear, and we have to find a cycle starting from a fixed node. We therefore fix an initial amount and, for each other currency, compute the maximum amount we can get by optimally choosing the money changers.

We propose the following heuristic algorithm: (for simplicity, we can assume that the graph is simple (there is at most one currency exchanger between any two currencies)):

Algorithm 1 Algorithm for finding the best path for each currency

Input: Starting node, starting amount, the data for each currency-exchanger
Output: The highest value and best path for every node
$Best_{\text{Starting Node}}$ = Starting Amount
List of vectors $Path$
$Path_{\text{Starting Node}}$ = (Starting Node)
Queue **Active** = (Starting Node)
while Active is not empty **do**
  $u :=$ the first active node
  remove u from **Active**
  for each changing option (u, v) with function f **do**
    if $f(Best_u) > Best_v$ **then**
      $Best_v = f(Best_u)$
      $Path_v = Path_u \circ v$
      if v is not in **Active then**
        add v to **Active**
      end if
    end if
  end for
end while
return $(Best, Path)$

The gas cost is neither negligible nor dominant, so a shorter path with a slightly lower gross return might be better than a longer one.

For this reason, before comparing the options at each node, we subtract the gas cost associated with the corresponding exchange.

If there is no arbitrage, the algorithm is deterministic and finds the optimal value and path for each node. The proof follows the same structure as the correctness proof of the Bellman-Ford algorithm.

If there is an arbitrage, the algorithm may fail to terminate. To avoid that, we can modify the algorithm and considering an edge u, v only if v does not appear on the path to u.

In that case ordering the edges differently might lead to a different result.

We have to find a cycle through the starting node. With the previously calculated values and paths, it is straightforward to select a reasonably good candidate cycle.

Algorithm 2 Algorithm for finding a good cycle

Input: Starting Node, The highest value and best path for every node
Output: The best cycle through the starting node
for $(u, v, f), v = StartingNode$: The changing options ending in the first node **do**
  if $f(Best_u) > Best_v$ **then**
    $Best_v = f(Best_u)$
    $Path_v = Path_u + v$
  end if
end for
return $Best_{StartingNode}, Path_{StartingNode}$

If, after this procedure the best value obtained for the $StartingNode$ is still the $StartingAmount$, then the algorithm has not identified any arbitrage opportunity.

Otherwise, an arbitrage cycle has been found. Next we have to find the best starting value, and to evaluate the exchanges along each edge of the cycle accordingly. For larger starting values the effective exchange ratios may become suboptimal due to the concavity of the exchange functions, making traversal of the entire cycle disadvantageous. On the other hand with a relatively small starting amount the accumulated gas costs may outweigh the gross profit of the cycle.

In most practical cases, the final profit as a function of the starting amount is a convex function. In that case we can apply the following approach to find the best amount to start with.

Algorithm 3 Algorithm for finding the best starting amount

Input: The cycle found by the previous algorithm
Output: The best starting amount
$lo = 0$
$hi = inf$
while $hi - lo > eps$ **do**
  $mid_1 = \frac{2*lo+hi}{3}$
  $mid_2 = \frac{lo+2*hi}{3}$
  if profit with mid_1 < profit with mid_2 **then**
    $lo = mid_1$
  else
    $hi = mid_2$
  end if
end while

Combining all three ideas we get the final algorithm.

5 Further Possibilities

The algorithm identifies a single cycle together with an optimal amount to send through it. Using similar ideas, it is possible to detect multiple edge-disjoint cycles, and each of which can be exploited independently. If there are several beneficial but non edge-disjoint cycles, it may also be beneficial to use them jointly. In that case, the effective exchange ratios on the shared edges may deteriorate, but the gas cost will decrease because there are fewer changes.

The optimal tree and best cycle depend on the starting value, which may change during the third algorithmic phase. Consequently, rerunning the procedure with that new value, it is possible to get a better tree and cycle after that. This approach is conceptually related to the max-cost circulation problem.

6 Running time

The primary bottleneck of the algorithm is the first phase. Although it always terminates, its theoretical worst-case running time may be exponential. In practice, after a number of iterations, the optimal values stabilize of the spanning tree the algorithm can be terminated early. In our experiments we had to check the neighbors of each node only a few times.

The remaining components execute efficiently. In the second phase each option needs to be scanned only once and in the third phase the profit of that found cycle is evaluated relatively few times.

Due to the complex interactions involved the subroutine of calculating the value of a change is significantly more expensive than a constant-time operation.

It is possible to optimize that. Each function will be nearly linear, so after knowing some points, the remaining values can be accurately predicted. By extrapolating from these known points that prediction will be pretty accurate. In the first phase, if the predicted value satisfies $f(Best_u) < Best_v$, then evaluating the exact function value is unnecessary. This optimization significantly reduces the number of expensive subroutine calls, since most invocations occur while the spanning tree is still evolving.

7 Results

In local experiments, the algorithm performed efficiently, even on graphs with more than 1,000 nodes and edges.

The running time was also satisfactory in real-world tests. Here the communication overhead—specifically, querying the exchange functions—was significantly slower. On the other hand, with the current parameters arbitrage opportunities are rarely detected. When the gas cost is reduced substantially, the algorithm performs markedly better, which indicating potential for further improvement.

References

https://arxiv.org/html/2406.16573v1
https://www.ultipa.com/docs/graph-analytics-algorithms/spfa
https://cp-algorithms.com/graph/bellman_ford.html
https://github.com/jtapolcai/tycho-searcher-cpp