Short Paper: Transfer algorithm for on-chain one-hop routing

Ruichao Jiang[1], Zirong Li[1], and Long Wen[1,2] * †

[1] Derivation Technology Ltd., Hong Kong, China
{ruichao.jiang,attens.li,long.wen}@derivation.info
[2] The Hong Kong Polytechnic University, Hong Kong, China
long1wen@polyu.edu.hk

Abstract. We propose a routing algorithm that finds the optimal one-hop swapping between two tokens. The gas consumption is optimized so that it can be run fully on-chain and supports large query amount. Our algorithm is adopted by the Monday Trade protocol.

Keywords: Automated market maker · Decentralized finance · Routing algorithm.

1 Introduction

Routing algorithms examine fragmented liquidities across different exchanges for the best swap path. [3] gives a satisfactory modeling of this problem and solves it using Cvxpy [4], a domain-specific optimization language. However, the computational power on blockchain is limited and expensive optimization procedure may not be feasible to run on-chain. The hybrid off-chain routing and on-chain execution model introduces an inherent risk: on-chain data may have changed by the time of execution. Moreover, the modeling in [3] assumes that the explicit forms of the trading functions (Definition 2) of the automated market makers (AMMs) are known, which is not always the case on-chain, especially for concentrated liquidity AMMs like Uniswap v3 [2]. We only assume the query access to the trading function.

1inch, an aggregator protocol, publishes an on-chain routing algorithm [1], which models the routing problem as a knapsack problem and solves it by dynamic programming (DP). To our best knowledge, it is the only on-chain routing algorithm published so far. In this article, we propose an on-chain routing algorithm (Algorithm 1) and show that it outperforms the DP-based algorithm.

The organization of the article is as follows. We formalize the routing problem and prove an optimality condition (Proposition 1) in Section 2. We propose our transfer algorithm in Section 3 and prove its convergence in Section 4. In Section 6, we present the experiment result to show the gas saving achieved by our algorithm.

---

*Corresponding author

†This work was fulfilled when Long Wen was with the Hong Kong Polytechnic University

2 Problem formulation

Denote by X/Y the token that we sell/buy.

Definition 1 (AMM). *Let $C : \mathbb{R}^2 \to \mathbb{R}$ be a function, called the AMM curve. An AMM of curve C quotes the price of X in Y by implicit differentiation*

$$p_{quote} = -\frac{dy}{dx} = \frac{\partial C/\partial x}{\partial C/\partial y},$$

where x and y are the inventory of X and Y in the AMM.

Definition 2 (Trading function). *The AMM trading function $E_{x,y} : x_{in} \mapsto E_{x,y}(x_{in})$ is the implicit function defined by the following equation.*

$$C(x + x_{in}, y - E_{x,y}(x_{in})) = k,$$

where $C(x, y)$ is the AMM curve, x and y the AMM inventory of X and Y, k some liquidity parameter.

Remark 1. $E_{x,y}(x_{in})$ is the amount of Y obtained when selling x_{in} amount of X to the AMM with inventory level (x, y).

For non-negative, monotonically decreasing, and convex C, $E_{x,y}$ is well-defined, non-negative, monotonically increasing, and concave [6]. Almost all AMMs in practice satisfy these conditions.

Definition 3 (Post-trade price). *The post-trade price of X in Y is defined as follows.*

$$E'_{x,y}(x_{in}) \coloneqq \lim_{\Delta x \to 0} \frac{E_{x,y}(x_{in} + \Delta x) - E_{x,y}(x_{in})}{\Delta x}.$$

Remark 2. The post-trade price of X in Y is the price quoted by the AMM in the state $(x + x_{in}, y - E_{x,y}(x_{in}))$.

Example 1. Consider the Uniswap v2 AMM with curve $xy = k$. The trading function is given by

$$E_{x,y}(x_{in}) = y - \frac{k}{x + x_{in}}.$$

The post-trade price

$$E'_{x,y}(x_{in}) = \frac{k}{(x + x_{in})^2} = \frac{y - E_{x,y}(x_{in})}{x + x_{in}}$$

coincides with the price of X in Y quoted by the Uniswap v2 AMM with inventory level $(x + x_{in}, y - E_{x,y}(x_{in}))$.

From now, we omit the subscripts of the AMM inventory in the trading function and write x for x_{in}. The problem is formulated as follows.

Given x amount of token X and N AMMs represented by their trading function E_i, $1 \leq i \leq n$,

$$\begin{aligned} \text{maximize: } & \sum_{i=1}^{N} E_i(w_i x), \\ \text{subject to: } & \omega_1, \omega_2, \cdots, \omega_n > 0, \\ & w_1 + w_2 + \cdots + \omega_n = 1. \end{aligned}$$

The scheme is illustrated in Figure 1 for $N = 3$.

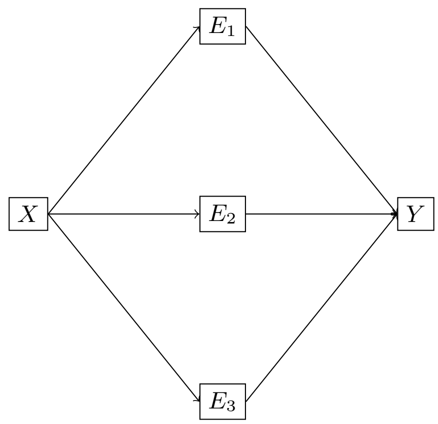


Fig. 1. Multi exchanges one hop

Since E_i's are concave, the problem is convex and numerous numerical algorithms exist. However, it is infeasible to run generic optimization solvers on-chain due to the gas constraint. Therefore, we exploit the structure of the problem.

Proposition 1 (Optimality condition). *A first order condition for the problem is: All exchanges with allocated X must share the same post-allocation price.*

Proof. Consider two exchanges E_i and E_j and assign $w_i x$ to E_i and $w_j x$ to E_j. Transfer an infinitesimal amount xdw from E_j to E_i and the change of the amount of Y, when expanded to the first order, is

$$\begin{aligned} & E_i((w_i + dw)x) - E_i(w_i x) + E_j((w_j - dw)x) - E_j(w_j x) \\ = & \left[E_i'(w_i x) - E_j'(w_j x) \right] x dw. \end{aligned}$$

If $E_i'(w_i x) > E_j'(w_i x)$, then the transfer from E_j to E_i increases the amount of Y and vice versa. Hence, at the optimal allocation, we must have

$$E_i'(w_i x) = E_j'(w_i x).$$

The following example is a demonstration of Proposition 1.

Example 2. Consider two exchanges

$$E_1(x) = 2\sqrt{x},$$
$$E_2(x) = 3\ln(x+1).$$

Even both E_1 and E_2 are monotonic and concave, they still intersect in 2 points, instead of 1 suggested by intuition. Using the Karush-Kuhn-Tucker (KKT) condition, the solution is

1. If $0 < x \leq \frac{1}{9}$,

$$w_1 = 1,$$
$$w_2 = 0.$$

   In this case, E_2 is not allocated because even after allocating all X to E_1, $E_1'(x)$ is still larger than $E_2'(0)$ (E' is the price of X in Y, so a larger E' means a cheaper Y).

2. If $x > \frac{1}{9}$,

$$w_1 x = \left(\frac{\sqrt{4x+13}-3}{2}\right)^2,$$
$$w_2 x = 3\left(\frac{\sqrt{4x+13}-3}{2}\right) - 1,$$

   and the post-trade prices are

$$E_1'(w_1 x) = \frac{1}{\sqrt{w_1 x}} = \frac{2}{\sqrt{4x+13}-3},$$
$$E_2'(w_2 x) = \frac{3}{w_2 x + 1} = \frac{2}{\sqrt{4x+13}-3},$$

   in agreement with Proposition 1.

3 Algorithm design

Proposition 1 says that we can aggressively allocate X to the exchange with the best post-allocate price and adjust the allocation according to the new best post-allocation price. This gives rise to the transfer-based Algorithm 1.

The notations are as follows.

Definition 4 (Allocation lists). *The allocation list is defined as*

$$\mathcal{A} : i \mapsto \left(\frac{1}{E_i'(x_i)}, x_i\right),$$

where x_i is the amount of X allocated to E_i.

Algorithm 1: Transfer algorithm for one-hop routing

```
Data: E: list of exchanges, x: total amount to be allocated, ε: tolerance
A ← initialize(E, x) ;                                   // Definition 4
D, R ← findDonor(A), findReceiver(A) ;                   // Definition 5
while e(A) ≥ εA[D][0] do                                 // Definition 6
    legitTransfer ← findLegitTransfer(A, D, R) ;         // Definition 7
    p_R ← getPrice(R, A[R][1] + legitTransfer) ;
    p_D ← getPrice(D, A[D][1] − legitTransfer) ;
    A[R] ← (p_R, A[R][1] + legitTransfer) ;
    A[D] ← (p_D, A[D][1] − legitTransfer) ;
    D, R ← findDonor(A), findReceiver(A) ;
return A ;
```

Remark 3. $\frac{1}{E_i'(x_i)}$ is the price of Y in X. So the larger the first entry in the allocation list, the more expensive the Y. We then should transfer X out of it.

Definition 5 (Donor and receiver). *The donor/receiver is defined as*

$$D := \arg\max_i \{\mathcal{A}[i][0] \mid \mathcal{A}[i][1] > 0\},$$
$$R := \arg\min_i \{\mathcal{A}[i][0]\},$$

i.e. donor is the allocated exchange with the most expensive Y and receiver is the exchange with the cheapest Y.

Definition 6 (Global error). *The global error of an allocation $\mathcal{A}$ is defined as*

$$e(\mathcal{A}) := \mathcal{A}[D][0] - \mathcal{A}[R][0].$$

Definition 7 (Legitimate transfer). *A transfer of amount d from the donor D to the receiver R is called legitimate if after the transfer,*

$$\mathcal{A}[R][0] \leq \mathcal{A}[D][0],$$

where $\mathcal{A}$ is the allocation list after the transfer.

We restrict to legitimate transfer to prevent from the scenario where one exchange E_i transfer too much to another exchange E_j so that E_j becomes the new donor and E_i becomes the new receiver. A naive approach like every time transfer half of the donor's amount to the receiver may lead to endless back-and-forth transfer between these two.

We know give an example of runtime of Algorithm 1.

Example 3. Consider two Univ v2 style AMM, E_1 with 1000 token X and 1000 token Y and E_2 with 10000 token X and 9999 token Y. We want to trade in 1000 token X. We calculate the theoretical maximum first.

$$(1000 + \Delta x)(1000 - \Delta y) = 1000^2,$$

$$(10000 + 1000 - \Delta x)(9999 - \Delta y') = 99990000,$$
$$\frac{1000 + \Delta x}{1000 - \Delta y} = \frac{10000 + 1000 - \Delta x}{9999 - \Delta y'}.$$

First two equations are AMM curve equations and the last equation is the optimality condition (equal post-trade prices). This solves to $\Delta x \approx 90.959, \Delta y \approx 83.375, \Delta y' \approx 833.208$. Then we show the runtime of Algorithm 1 on this with $\varepsilon = 0.01$.

1. Allocate 1000 token X to E_1, $\Delta x = 1000$, $\Delta y = 500$. Post-trade price $p_1 = 4$.
2. Transfer 500 X to E_2. $\Delta x = 500$, $\Delta y = \frac{1000}{3}$, $\Delta x' = 500$, $\Delta y' = \frac{3333}{7}$. $p_1 = 2.25$, $p_2 = \frac{1225}{1111}$.
3. Transfer 250 to E_2. $\Delta x = 250$, $\Delta y = 200$, $\Delta x' = 750$, $\Delta y' = \frac{29997}{43}$. $p_1 = \frac{25}{16}$, $p_2 \approx 1.16$.
4. Transfer 125 to E_2. $\Delta x = 125$, $\Delta y = \frac{1000}{9}$, $\Delta x' = 875$, $\Delta y' \approx 804.52$. $p_1 = \frac{81}{64}$, $p_2 \approx 1.18$.
5. (a) Try transferring 62.5 to E_2. $\Delta x = 62.5$, $\Delta y \approx 58.82$, $\Delta x' = 937.5$, $\Delta y' \approx 857.06$. $p_1 \approx 1.129$, $p_2 \approx 1.196$. The transfer is not legitimate per Definition 7.
   (b) Transfer 31.25 to E_2. $\Delta x = 93.75$, $\Delta y \approx 85.71$, $\Delta x' = 906.25$, $\Delta y' \approx 830.96$. $p_1 \approx 1.19628$, $p_2 \approx 1.18959$.
6. Since $\frac{p_1 - p_2}{p_1} \approx 0.006 < \varepsilon$, the algorithm terminates.

The algorithm obtained $85.714 + 830.862 = 916.576$ token Y, worse than the theoretical value $83.375 + 833.208 = 916.583$ by less than 1 cent. The result is better than, for example, allocating all token X to E_2, which gives 909 token Y.

4 Convergence of algorithm

Proposition 2. *After a legitimate transfer, the global error $e(\mathcal{A})$ decreases.*

Proof. Denote by $\mathcal{A}'$ the allocation list after a legitimate transfer.

Case 1: the donor remains the same. We have

$$\max_i\{\mathcal{A}'[i][0]\} = \mathcal{A}'[D][0] = \frac{1}{E'_D(x_D - d)} < \frac{1}{E'_D(x_D)} = \mathcal{A}[D][0],$$

where the inequality is by the concavity of E_D.

Case 2: a change of donor occurred and the new donor becomes $D' \neq D$. Since the transfer was legitimate, we have $D' \neq R$. Hence, D' corresponds to the second most expensive Y in $\mathcal{A}$. Hence,

$$\max_i\{\mathcal{A}'[i][0]\} = \mathcal{A}'[D'][0] = \mathcal{A}[D'][0] \leq \mathcal{A}[D][0].$$

Similarly,

$$\min_i\{\mathcal{A}'[i][0]\} \geq \mathcal{A}[R][0].$$

Hence, $e(\mathcal{A}') \leq e(\mathcal{A})$.

Proposition 2 implies the convergence of the transfer algorithm.

5 Complexity

The complexity of Algorithm 1 depends on the forms of trading functions and the liquidity distribution. So there is no simple characterization of the speed of convergence. Consider the following example.

Example 4. Consider N exchanges $E_1, E_2, \cdots, E_N$.

1. E_1 is allocated 1 and subsequently transfers $\frac{1}{2}, \frac{1}{4}, \cdots, \frac{1}{2^{N-1}}$ to $E_2, E_3, \cdots, E_N$. After this E_2 becomes the donor.
2. E_2 transfers $\frac{1}{4}, \frac{1}{8}, \cdots, \frac{1}{2^{N-1}}$ to $E_3, E_4, \cdots, E_N$.
3. Continue until E_{N-1} becomes the donor and it transfers $\frac{1}{4}$ to E_N.
4. At the end, E_1 has $\frac{1}{2^{N-1}}$, E_i has $\frac{1}{2^{N-i+1}}$ for $2 \le i \le N$.

To construct trading functions compatible with the above, we list all the constraints for $N = 4$.

- • To choose E_1 as the donor, we need $p_1(1) \ge p_2(0), p_3(0), p_4(0)$, and $p_1\left(\frac{1}{2}\right) \ge p_2\left(\frac{1}{2}\right), p_3(0), p_4(0)$, and $p_1\left(\frac{1}{4}\right) \ge p_2\left(\frac{1}{2}\right), p_3\left(\frac{1}{4}\right), p_4(0)$.
  • To choose E_2 as the receiver, we need $p_2(0) \le p_3(0), p_4(0)$. To choose E_3 as the receiver, we need $p_3(0) \le p_2\left(\frac{1}{2}\right), p_4(0)$. To choose E_4 as the receiver, we need $p_4(0) \le p_2\left(\frac{1}{2}\right), p_3\left(\frac{1}{4}\right)$.
  • For transfers from E_1 being legitimate, we need $p_1\left(\frac{1}{2}\right) \ge p_2\left(\frac{1}{2}\right)$, $p_1\left(\frac{1}{4}\right) \ge p_3\left(\frac{1}{4}\right)$, and $p_1\left(\frac{1}{8}\right) \ge p_4\left(\frac{1}{8}\right)$.
- • To choose E_2 as donor, we need $p_2\left(\frac{1}{2}\right) \ge p_1\left(\frac{1}{8}\right), p_3\left(\frac{1}{4}\right), p_4\left(\frac{1}{8}\right)$, and $p_2\left(\frac{1}{4}\right) \ge p_1\left(\frac{1}{8}\right), p_3\left(\frac{1}{2}\right), p_4\left(\frac{1}{8}\right)$.
  • To choose E_3 as receiver, we need $p_3\left(\frac{1}{4}\right) \le p_1\left(\frac{1}{8}\right), p_4\left(\frac{1}{8}\right)$. To choose E_4 as the receiver, we need $p_4\left(\frac{1}{8}\right) \le p_1\left(\frac{1}{8}\right), p_3\left(\frac{1}{2}\right)$.
  • For transfers from E_2 being legitimate, we need $p_2\left(\frac{1}{4}\right) \ge p_3\left(\frac{1}{2}\right)$, $p_2\left(\frac{1}{8}\right) > p_4\left(\frac{1}{4}\right)$.
- • To choose the E_3 as donor, we need $p_3\left(\frac{1}{2}\right) \ge p_1\left(\frac{1}{8}\right), p_2\left(\frac{1}{8}\right), p_4\left(\frac{1}{4}\right)$.
  • To choose E_4 as receiver, we need $p_4\left(\frac{1}{4}\right) \le p_1\left(\frac{1}{8}\right), p_2\left(\frac{1}{8}\right)$.
  • For the transfer from E_3 being legitimate, we need $p_3\left(\frac{1}{4}\right) \ge p_4\left(\frac{1}{2}\right)$.

The above constraints can be satisfied by

- $p_1(1) = p_1\left(\frac{1}{2}\right) = p_1\left(\frac{1}{4}\right) = 1$, $p_1\left(\frac{1}{8}\right) = 0.5$, $p_1(0) = 0.1$.
- $p_2\left(\frac{1}{2}\right) = p_2\left(\frac{1}{4}\right) = 0.9$, $p_2\left(\frac{1}{8}\right) = 0.4$, $p_2(0) = 0.1$.
- $p_3\left(\frac{1}{2}\right) = 0.8$, $p_3\left(\frac{1}{4}\right) = 0.3$, $p_3(0) = 0.1$.
- $p_4\left(\frac{1}{2}\right) = p_4\left(\frac{1}{4}\right) = p_4\left(\frac{1}{8}\right) = 0.3$, $p_4(0) = 0.1$.

If the numeric in Example 4 can be generalized from 4 to generic N, then we have found a scenario where the runtime is $\mathcal{O}(N^2)$. And this is not guaranteed to be the worst case. Note that in step 1, the minimum percentage of transfer is $m = \frac{1}{2^{N-1}}$. So $N = \mathcal{O}\left(\log \frac{1}{m}\right)$. And the complexity can be written as $\mathcal{O}\left(N \log \frac{1}{m}\right)$. It seems difficult to come up with an $\mathcal{O}\left(N^2 \log \frac{1}{m}\right)$ complexity example. If E_1 transfer m to each E_i, then E_i cannot transfer any token out as it reaches the precision limit.

In real blockchain environment, it's a common practice to cap all loops. So after certain rounds, Algorithm 1 is forced to terminate. The result may not reach the specified price tolerance.

6 Experiment

We compare the performance of the transfer algorithm implemented by [5] with DP. The experiment was performed on the Base blockchain at block 26325854. The liquidity sources used were 12 WETH/USDC pools across 7 AMMs.

1. Two pools in Pancake V3
   (a) 0x72AB388E2E2F6FaceF59E3C3FA2C4E29011c2D38
   (b) 0xB775272E537cc670C65DC852908aD47015244EaF
2. Four pools in Uniswap V3
   (a) 0xb4CB800910B228ED3d0834cF79D697127BBB00e5
   (b) 0xd0b53D9277642d899DF5C87A3966A349A798F224
   (c) 0x6c561B446416E1A00E8E93E221854d6eA4171372
   (d) 0x0b1C2DCbBfA744ebD3fC17fF1A96A1E1Eb4B2d69
3. One pool in Uniswap V2
   – 0x88A43bbDF9D098eEC7bCEda4e2494615dfD9bB9C
4. One pool in Aerodrome SlipStream
   – 0xb2cc224c1c9feE385f8ad6a55b4d94E92359DC59
5. One pool in Aerodrome
   – 0xcDAC0d6c6C59727a65F871236188350531885C43
6. Two pools in Sushiswap v3
   (a) 0x482Fe995c4a52bc79271aB29A53591363Ee30a89
   (b) 0x57713F7716e0b0F65ec116912F834E49805480d2
7. One pool in Alien Base
   – 0xB27f110571c96B8271d91ad42D33A391A75E6030

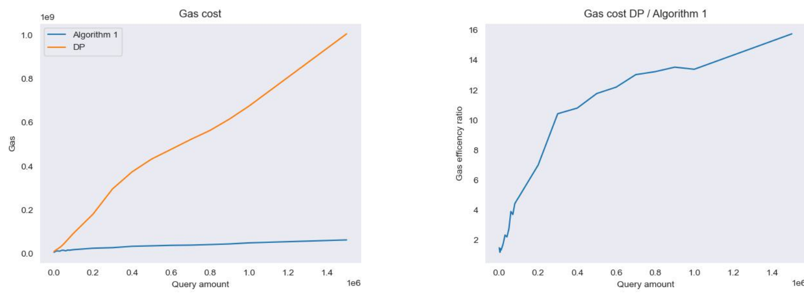
The statistics of these pools are as follows.

Pool	WETH	USDC (k)	Bid	Ask	Fee (%)
1(a)	517.18	432.85	2682.08	2682.62	0.01
1(b)	67.62	129.50	2681.11	2683.79	0.05
2(a)	86.42	54.11	2682.45	2682.99	0.01
2(b)	67.62	7480.20	2680.90	2683.58	0.05
2(c)	6329.37	1872.12	2671.36	2687.44	0.3
2(d)	948.72	41.00	2659.48	2713.20	1
3	2085.02	5578.46	2667.46	2683.51	0.3
4	10571.21	11005.10	2681.48	2683.62	0.04
5	4289.67	11492.50	2671.07	2687.14	0.3
6(a)	259.18	561.27	2682.12	2682.66	0.01
6(b)	797.65	1469.24	2680.91	2683.59	0.05
7	81.55	216.98	2680.39	2684.41	0.075

For DP, the input was divided into 10 parts and each part was queried against each pool, in total 120 queries. The query result became the input of the 0-1 knapsack problem and was solved by DP. This is the algorithm used in [1].

The result is in Figures 2. Our transfer algorithm outperforms DP in gas consumption up to 16 times (Figure 2b). Also, the maximal supported query amount is 20 times larger (31,000,000 vs 1,500,000 USDC).

Figure 2a shows that the gas cost of DP increases with query amount, despite DP makes a fixed amount of queries. This is because in concentrated liquidity AMMs, a large query may cross numerous ticks, which leads to higher gas cost.



(a) Gas consumptions by DP and Algorithm 1

(b) Gas consumption by DP / gas consumption by Algorithm 1

Fig. 2. Caption for this figure with two images

7 Discussion

In this article, we proposed a transfer algorithm for one-hop routing and proved its convergence. We showed empirically that it outperforms the DP-based solution. Further research includes detailed analysis of complexity and the generalization of the current one-hop version transfer algorithm to multi-hops case. The key to handling multi-hops case is to ensure a local property: After a transfer from one path to another, at most these two paths can change price. Otherwise we will be forced to query the prices of all paths to ensure correctness.

References

1. 1inch: 1inchprotocol (2020), https://github.com/1inch/1inchProtocol/blob/master/contracts/OneSplitBase.sol
2. Adams, H., Zinsmeister, N., Salem, M., Keefer, R., Robinson, D.: Uniswap v3 core. Tech. rep., Uniswap (2021), https://app.uniswap.org/whitepaper-v3.pdf
3. Angeris, G., Evans, A., Chitra, T., Boyd, S.: Optimal routing for constant function market makers. In: Proceedings of the 23rd ACM Conference on Economics and Computation. p. 115–128. EC '22, Association for Computing Machinery (2022). https://doi.org/10.1145/3490486.3538336
4. Diamond, S., Boyd, S.: Cvxpy: A python-embedded modeling language for convex optimization. The Journal of Machine Learning Research **17**(1), 2909–2913 (2016)
5. Monday Trade: https://monday.trade/
6. Schlegel, J., Kwaśnicki, M., Mamageishvili, A.: Axioms for constant function market makers. In: Proceedings of the 24th ACM Conference on Economics and Computation. p. 1079. Association for Computing Machinery (2023). https://doi.org/10.1145/3580507.3597720