On-chain routing

Synfutures team

March 27, 2025

Abstract

In this article, we propose a routing algorithm that finds the optimal direct swapping between two tokens. Then we optimize the gas consumption so that it can be run fully on-chain and supports large query amount.

1 Introduction

Routing algorithms examine fragmented liquidities across different exchanges to search for the best swap path. Angeris et al. (2022) gives a satisfactory modeling of this problem and uses a domain-specific optimization language Cvxpy (Diamond and Boyd, 2016) to solve the model. However, the computing resource on blockchain are limited and expensive optimization procedure may not be feasible to run on-chain. Moreover, the modeling and solution in Angeris et al. (2022) use the specific function forms of the trading functions of the automated market makers (AMMs), which are not directly available on-chain. Off-chain routing and on-chain execution introduce an inherent risk: The on-chain data may have changed by the time of execution. Moreover, off-chain algorithms are not trustless.

1inch, a leading DeFi aggregator protocol, published in its Github repository (1inch, 2020) an on-chain routing algorithm, which models the routing problem as a knapsack problem solved by dynamic programming (DP). To our best knowledge, it is the only on-chain routing algorithm in the public domain. In this article, we propose an on-chain routing algorithm (Algorithm 1) and show that it outperforms the knapsack-based algorithm.

In Section 2, we formalize the routing problem and derive an optimality condition (Proposition 1). In Section 3, we propose our algorithm and in Section 4, we prove the convergence and give two examples of its diverse run time behavior. In Section 5, we perform an experiment to show the gas saving achieved by Algorithm 1.

2 Modeling

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

Definition 1 (Trading function)**.** *Consider an AMM with the curve $C(x, y) = k$, where k is some liquidity parameter. Define the AMM trading function $E_{x,y} : x_{in} \mapsto E_{x,y}(x_{in})$ as follows.*

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

for $x > 0$.

$E_{x,y}(x_{in})$ is the amount of Y obtained when selling x_{in} amount of X in the AMM with inventory level (x, y). For AMMs satisfying some mild conditions (Schlegel et al., 2023), $E_{x,y}$ are well-defined, non-negative, monotonically increasing, and concave.

Definition 2 (Post-trade price)**.** *The post-trade price of X in Y is defined as the derivative of the trading function, i.e.*

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

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

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

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

We omit the subscripts of the AMM state in the trading function and write x_{in} just as x. The problem is formulated as follows.

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

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

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

Since E_i's are concave, the above problem is convex and therefore numerous numerical algorithms exist. However, it is infeasible to run optimization solvers on-chain due to the gas constraint. Therefore, we explore the special sturcture of the problem given by its constraints, which is summarized in the following proposition.

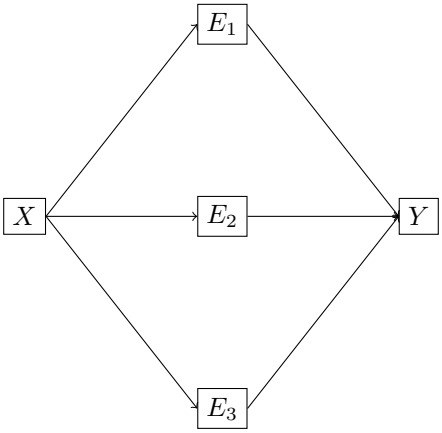

Figure 1: Multi exchanges one hop

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 we assign $w_i f$ to E_i and $w_j x$ to E_j. Divert a small portion xdw from E_j to E_i and the change in the amount of Y we obtain, when expanded to the first order, is

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

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

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

□

Below is an example showing that Proposition 1 indeed holds.

Example 2. *Consider two exchanges*

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

Even both E_1 and E_2 are monotonic and concave, they still intersect in a non-trivial way (Figure 2). Using the Karush-Kuhn-Tucker (KKT) condition, the solution is

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

$$\begin{aligned} w_1 &= 1, \\ w_2 &= 0. \end{aligned}$$

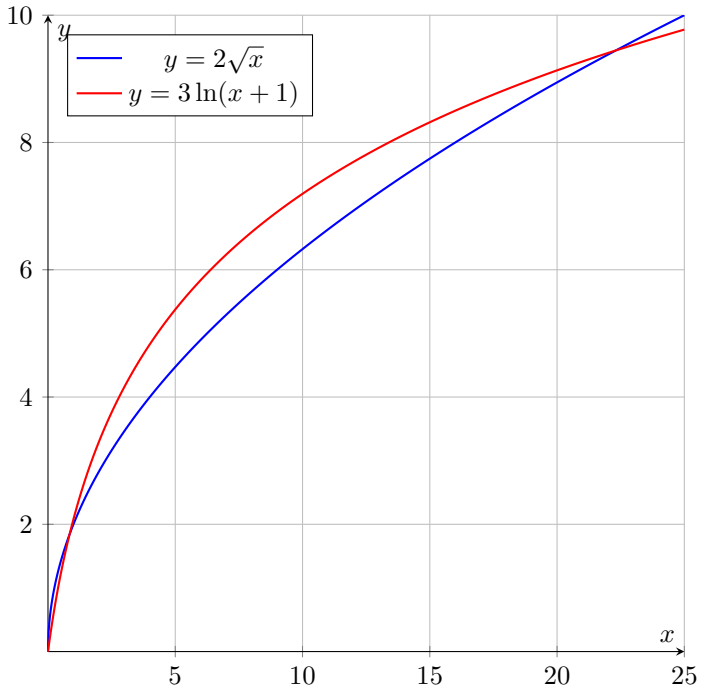

Figure 2: Trading functions for two exchanges

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[1] and adjust the allocation according to the new best post-allocation price.

We define some useful items.

Definition 3 (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.

Definition 4 (Donor and receiver)**.** *The donor/receiver is defined as*

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

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

Definition 5 (Global error)**.** *The global error of an allocation $\mathcal{A}$ is defined as*

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

Definition 6 (Legitimate diversion)**.** *A diversion of amount d from the donor D to the receiver R is called legitimate if after diversion,*

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

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

Remark 1. *We restrict to legitimate diversion to prevent from the scenario where one exchange E_i diverts too much to another exchange E_j so that E_j becomes the new donor and E_i becomes the new receiver. In the worst case, this may lead to endless back-and-forth diversion between these two.*

The result is given in Algorithm 1.

4 Analysis of algorithm

The convergence result is given by the following proposition.

Proposition 2. *After a legitimate diversion, the global error decreases.*

[1] If an exchange E_i is not allocated, its post-allocation price is $E_i'(0)$.

Algorithm 1: Single-hop allocation

Data: E: list of exchanges, x:total amount to be allocated,ε:global error tolerance

$\mathcal{A} \leftarrow \text{initialize}(E, x)$;

$D, R \leftarrow \text{findDonor}(\mathcal{A}), \text{findReceiver}(\mathcal{A})$; `// Definition 4`

while $e(\mathcal{A}) \geq \varepsilon \mathcal{A}[D][0]$ **do**

  $\text{legitDiversion} \leftarrow \text{findLegitDiversion}(\mathcal{A}, D, R)$;

  $p_R \leftarrow \text{getTradePrice}(R, \mathcal{A}[R][1] + \text{legitDiversion})$;

  $p_D \leftarrow \text{getTradePrice}(D, \mathcal{A}[D][1] - \text{legitDiversion})$;

  $\mathcal{A}[R] \leftarrow (p_R, \mathcal{A}[R][1] + \text{legitDiversion})$;

  $\mathcal{A}[D] \leftarrow (p_D, \mathcal{A}[D][1] - \text{legitDiversion})$;

  $D, R \leftarrow \text{findDonor}(\mathcal{A}), \text{findReceiver}(\mathcal{A})$;

end

return $\mathcal{A}$;

Proof. If the donor remains the same after a legitimate diversion, 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.

If a change of donor happened and the new donor becomes $D' \neq D$, since the diversion 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, we can show that

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

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

Remark 2. *If there are two exchanges, one having the same price as the donor and the other having the same price as the receiver, then after a legitimate diversion, the global error will stay the same. But even in this case, the allocation result becomes better.*

Proposition 2 implies that Algorithm 1 converges and each iteration improves the result.

The complexity analysis of Algorithm 1 is difficult. We consider an example.

Example 3. *Consider N exchanges, N being even. Suppose $x = 1$. Let $m \ll 1$ be a precision parameter so that mx is the smallest size of a diversion. The first $\frac{N}{2}$ exchanges are exactly the same and the last $\frac{N}{2}$ exchanges are exactly the same. And the last $\frac{N}{2}$ exchanges have worse price than the first $\frac{N}{2}$.*

1. *After* $\frac{N}{2}$ *iterations, the first* $\frac{N}{2}$ *exchanges are allocated* $\frac{2}{N}$ *amount of* X *each.*
2. *After* $\left\lfloor \log \frac{2}{mN} \right\rfloor$ *times of halving the diversion,* E_1 *diverts* m *amount of* X *to* $E_{\frac{N}{2}+1}$*.* E_i *repeats this for* $E_{\frac{N}{2}+i}$*,* $2 \leq i \leq \frac{N}{2}$*. This amounts to* $\frac{N}{2}\left\lfloor \log \frac{2}{mN} \right\rfloor$ *operations.*
3. *At this moment,* E_1 *becomes the donor and* $E_{\frac{N}{2}+1}$ *becomes the receiver again. However,* E_1 *is not able to find the amount of diversion. If* E_1 *were to divert* m *amount of* X*,* E_1 *would have diverted* $2m$ *amount of* X *in Step 2. Hence, Algorithm 1 halts.*

In this case, the query complexity of Algorithm 1 is $\Omega(N \log \frac{1}{mN})$*. The above requires* $m < \frac{2}{N}$*. Hence, if* $\frac{1}{m} = \mathrm{poly}(N)$*, the query complexity will be* $\Omega(N \log N)$*.*

Example 4. *Same as Example 3.*

1. *After* $\frac{N}{2}$ *iterations, the first* $\frac{N}{2}$ *exchanges are allocated* $\frac{2}{N}$ *amount of* X *each.*
2. *For* $1 \leq i \leq \frac{N}{2}$*, after* $\left\lfloor \log \frac{2}{2^{\frac{N}{2}-1}mN} \right\rfloor$ *times of halving the diversion,* E_i *diverts* $2^{\frac{N}{2}-1}m$ *amount of* X *to* $E_{\frac{N}{2}+i}$*.*
3. *Repeat 2 with halving times* $\left\lfloor \log \frac{2}{2^{\frac{N}{2}-j}mN} \right\rfloor$ *and the diversion amount* 2*,* $2 \leq j \leq \frac{N}{2}$*. Since a change of donor/receiver happens after each diversion, the cache is always cleared. In total,* E_i *diverted* $\left(2^{\frac{N}{2}}-1\right)m$ *amount of* X *to* $E_{\frac{N}{2}+i}$*. We assume* $\frac{1}{m} = \exp(N)$ *so the allocated amount of* E_i *is always approximately* $\frac{2}{N}$*.*

For each E_i*, the steps of operation is*

$$\sum_{j=1}^{\frac{N}{2}} \log \frac{2}{2^{\frac{N}{2}-j}mN} = \frac{3}{4}N - \frac{N^2}{8} + \frac{N}{2}\log\frac{1}{mN} = \Omega\left(N \log \frac{1}{mN}\right).$$

Therefore the query complexity in this case is $\Omega(N^2 \log \frac{1}{mN})$*. If* $m = \frac{1}{N2^{\frac{N}{2}}}$*, the complexity will be* $\Omega(N^3)$*. Note that it is almost necessary that* $m = \frac{1}{N2^{\frac{N}{2}}}$ *or smaller because we need*

$$\frac{2}{N} - \left(2^{\frac{N}{2}}-1\right)m > m,$$

i.e. after diverting X*, each donor has at least the minimal amount.*

It is unclear if Example 3 is the worst case scenario when the precision parameter satisfies $\frac{1}{m} = \mathrm{poly}(N)$ due to the complex behavior of Algorithm 1. For example, it is possible that a donor becomes a receiver in some future iteration.

Therefore, the allocated amount for that donor is not monotonically decreasing. However, both Examples 3 and 4 are extreme in the sense that to actually encounter these cases, exchanges $E_{\frac{N}{2}+i}$ have to suffer from large price impact. Example 4 is even more extreme because m has to be exponentially small in N. If, instead of having $O(N)$ donors, there are only $O(1)$ donors and each diversion changes allocation meaningfully, the complexity will be $O\left(\log \frac{1}{m}\right)$.

5 Experiment

We compare the performance of our implementation of Algorithm 1 with our implementation of the knapsack problem formulation, which is solved by dynamic programming (DP).

The experiment was performed on the Base blockchain at block 26325854. The liquidity sources used were 12 WETH/USDC pools across 7 AMMs (Appendix A).

In the DP approach, the input amount was divided into 10 parts each part was queried against each pool, in total 120 queries. Then query result was the input of the 0-1 knapsack problem and was solved by DP. This is the algorithm in 1inch (2020).

The detailed experiment result is in Appendix B. According to Table 1, Algorithm 1 outperforms DP in both WETH amount and gas consumption. Moreover, the maximal supported query amount is 20 times larger (31,000,000 vs 1,500,000).

The gas consumption comparison is as follows.

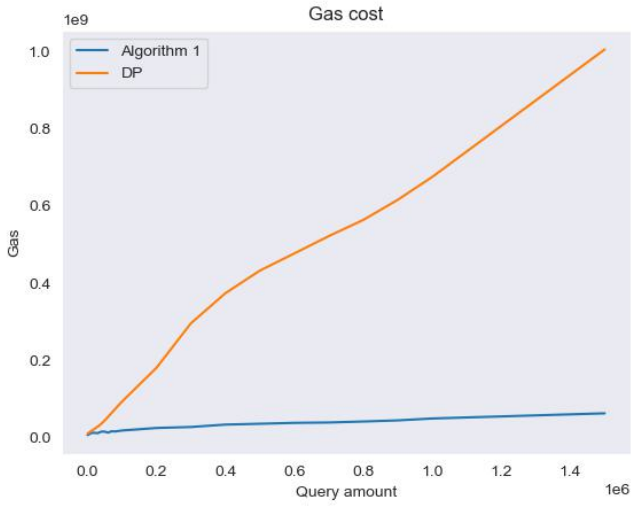

Figure 3: Gas consumptions by DP and Algorithm 1

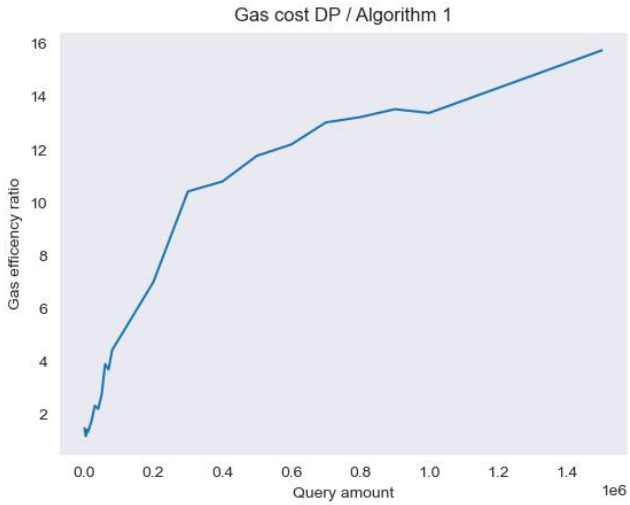

Figure 4: Gas consumption by DP / gas consumption by Algorithm 1

Figure 4 shows that Algorithm 1 saves gas up to 16 times compared to the knapsack DP approach and the gas saving effect is more pronounced when query amount is large. Since DP makes a fixed amount (120) of queries, it is interesting that queries of large amount consume significantly more gas. This is because in AMMs like Uniswap V3, a large query/trade may cause numerous tick crossings and that contributes to higher gas cost.

6 Discussion

In this article, we proposed an on-chain routing algorithm (Algorithm 1) and proved its convergence. We showed empirically that it outperforms the knapsack-based solution with the maximal 16 times save on gas and 20 times larger in maximum query amount. Further research on the complexity of Algorithm 1 is needed since it was not clear to us what counts as the worst scenario.

References

[1] 1inch. *1inchProtocol*. 2020. URL: `https://github.com/1inch/1inchProtocol/blob/master/contracts/OneSplitBase.sol`.

[2] G. Angeris, A. Evans, T. Chitra, and S. Boyd. "Optimal Routing for Constant Function Market Makers". In: *Proceedings of the 23rd ACM Conference on Economics and Computation*. EC '22. Association for Computing Machinery, 2022, pp. 115–128. DOI: `10.1145/3490486.3538336`.

[3] S. Diamond and S. Boyd. "Cvxpy: A python-embedded modeling language for convex optimization". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 2909–2913.

[4] J. Schlegel, M. Kwaśnicki, and A. Mamageishvili. "Axioms for Constant Function Market Makers". In: *Proceedings of the 24th ACM Conference on Economics and Computation*. Association for Computing Machinery, 2023, p. 1079. DOI: 10.1145/3580507.3597720.

Appendices

A Pool information

1. Two pools in Pancake V3
   - 0x72AB388E2E2F6FaceF59E3C3FA2C4E29011c2D38
   - 0xB775272E537cc670C65DC852908aD47015244EaF
2. Four pools in Uniswap V3
   - 0xb4CB800910B228ED3d0834cF79D697127BBB00e5
   - 0xd0b53D9277642d899DF5C87A3966A349A798F224
   - 0x6c561B446416E1A00E8E93E221854d6eA4171372
   - 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
   - 0x482Fe995c4a52bc79271aB29A53591363Ee30a89
   - 0x57713F7716e0b0F65ec116912F834E49805480d2
7. One pool in Alien Base
   - 0xB27f110571c96B8271d91ad42D33A391A75E6030

B Experiment result

The output WETH amount is kept to 4 digits. The WETH and Gas columns are obtained by Algorithm 1. N/A means out of gas.

USDC	WETH	WETH (DP)	Gas	Gas (DP)
32,000,000	N/A	N/A	N/A	N/A
31,000,000	11,150.5202	N/A	1,011,625,713	N/A
10,000,000	3,712.3658	N/A	310,765,982	N/A
9,000,000	3,342.4950	N/A	291,044,347	N/A
8,000,000	2,972.1640	N/A	264,565,520	N/A
7,000,000	2,601.5820	N/A	238,866,283	N/A
6,000,000	2,230.8192	N/A	213,991,116	N/A
5,000,000	1,859.7030	N/A	187,516,078	N/A
4,000,000	1,488.3185	N/A	149,340,518	N/A
3,000,000	1,116.6788	N/A	105,014,675	N/A
2,000,000	744.7289	N/A	83,254,978	N/A
1,600,000	595.8684	N/A	67,267,041	N/A
1,500,000	558.6465	558.6442	63,793,646	1,005,224,073
1,000,000	372.4976	372.4965	50,409,369	675,409,165
900,000	335.2598	335.2590	45,521,069	616,303,183
800,000	298.0194	298.0187	42,614,341	564,175,856
700,000	260.7763	260.7758	40,055,093	522,305,778
600,000	223.5305	223.5301	39,105,293	477,426,565
500,000	186.2819	186.2818	36,758,045	433,002,172
400,000	149.0313	149.0308	34,640,377	374,450,387
300,000	111.7777	111.7772	28,425,718	296,627,662
200,000	74.5211	74.5208	25,834,998	181,366,174
100,000	37.2620	37.2617	19,299,257	93,909,435
90,000	33.5359	33.5357	18,098,562	84,348,647
80,000	29.8098	29.8096	16,786,530	74,456,942
70,000	26.0837	26.0835	17,401,768	64,650,531
60,000	22.3576	22.3574	13,998,851	54,876,890
50,000	18.6314	18.6312	16,382,517	45,312,204
40,000	14.9052	14.9051	16,420,519	36,588,106
30,000	11.1790	11.1790	12,482,296	29,335,793
20,000	7.4528	7.4527	13,391,454	23,288,343
10,000	3.7265	3.7265	12,833,739	17,363,590
9,000	3.3539	3.3538	12,498,283	16,888,291
8,000	2.9812	2.9812	11,136,417	16,072,725
7,000	2.6086	2.6086	10,921,554	15,524,926
6,000	2.2360	2.2360	10,290,490	14,817,897
5,000	1.8634	1.8634	11,200,736	14,067,069
4,000	1.4907	1.4907	11,037,951	13,271,391
3,000	1.1181	1.1181	9,502,404	12,679,135
2,000	0.7454	0.7454	9,048,126	12,224,625

USDC	**WETH**	**WETH (DP)**	**Gas**	**Gas (DP)**
1,000	0.3727	0.3727	7,966,390	11,658,197
500	0.1864	0.1864	7,430,601	11,159,433

Table 1: Experiment result