

# Uniswap Arbitrage Analysis

## 0.前言

Uniswap的AMM (auto market maker) 范式无疑开启了一个新的时代, 该设想最早见于Vitalik的一篇文章, Bancor中也有类似的设计, 最终Uniswap把它发扬光大。

截止目前 (2020-09-27), uniswap V2上已经有10800多个交易对, 有市场的地方就有套利, uniswap这类去中心化交易所其实对套利是非常友好的, 整个套利过程可以在一笔交易中完成, 相当于一个原子操作, 不存在单腿风险, 外加AAVE、DyDx的闪电贷, 套利连本金都不需要了, 自己只付出手续费即可, Defi世界简直是套利者的天堂。

很多人说Defi是泡沫, 套利不创造价值, 这里扯一点自己的看法, 关于泡沫, 肯定是有, 但是泡沫过后终究会沉淀下一些有价值的东西, 2000年的时候大家也大喊互联网是泡沫, 再看看现在, 当然Defi能不能比肩互联网还不好说, 但是说它毫无价值肯定是错的。套利则是联系市场的纽带, 维系着整个市场的平衡稳定, 没有套利者, 那就都变成一个个单机游戏了。

本文主要分析一下Uniswap不同交易对之间的套利问题, 具体一点就是A->B->C->...->A, 通过这样的一系列交易用A赚到更多的A。

## 1.套利问题分析

本质上来看, 这个套利问题由两个子问题构成:

- 如何确定套利路径: 从A币种出发, 中间经过哪些币种, 最后能得到最多的A?
- 如何确定套利数量: 用多少A进行交易, 最终利润最多?

### 1.1 套利路径

我们把每个币种看做一个顶点, 每个交易对就是一条边, 问题就转化成了如何在一个有向图里面寻找环状路径的问题, 这是很经典的图问题了, 比较简单, 这里用DFS可以比较好地控制路径长度, uniswap交易时路径每增加一步就要消耗更多gas, 因此需要控制长度(maxHops), 参考实现:

```
def findArb(pairs, tokenIn, tokenOut, maxHops, currentPairs, path, circles):
    for i in range(len(pairs)):
        newPath = path.copy()
        pair = pairs[i]
        if not pair['token0']['address'] == tokenIn['address'] and not
pair['token1']['address'] == tokenIn['address']:
            continue
        if pair['reserve0']/pow(10, pair['token0']['decimal']) < 1 or
pair['reserve1']/pow(10, pair['token1']['decimal']) < 1:
            continue
        if tokenIn['address'] == pair['token0']['address']:
            tempOut = pair['token1']
        else:
            tempOut = pair['token0']
```

```

newPath.append(tempOut)
if tempOut['address'] == tokenOut['address'] and len(path) > 2:
    c = { 'route': currentPairs, 'path': newPath }
    circles.append(c)
elif maxHops > 1 and len(pairs) > 1:
    pairsExcludingThisPair = pairs[:i] + pairs[i+1:]
    circles = findArb(pairsExcludingThisPair, tempOut, tokenOut,
maxHops-1, currentPairs + [pair], newPath, circles)
return circles

```

## 1.2 套利数量

我们先来回顾一下Uniswap的CFMM(constant function market maker)模型，假设现在A, B币种有一个交易池，池中A的数量为 $R_0$ ，B的数量为 $R_1$ ，现在用 $\Delta_a$ 的A币种去兑换 $\Delta_b$ 的B币种，交易手续费为 $1 - r$ ，则必须满足：

$$(R_0 + r\Delta_a)(R_1 - \Delta_b) = R_0 R_1$$

即兑换前后 $R_0 R_1$ 保持为一个不变的常数，这也是CFMM名称的由来。

那假设我们现在找到了一条环状路径A->B->C->A，如何确定用多少A来套利能获得最大利润呢？这其实是下面这样一个优化问题：

$$\max(\Delta_a' - \Delta_a)$$

$$s.t.$$

$$R_n > 0, \Delta_n > 0$$

$$(R_0 + r\Delta_a)(R_1 - \Delta_b) = R_0 R_1 \quad (1)$$

$$(R_1' + r\Delta_b)(R_2 - \Delta_c) = R_1' R_2 \quad (2)$$

$$(R_2' + r\Delta_c)(R_3 - \Delta_a') = R_2' R_1' \quad (3)$$

其中(1)(2)(3)分别对应着从A->B, B->C, C->A的交易过程。现在这个问题看起来也挺容易的，可以直接把 $\Delta_a'$ 用 $\Delta_a$ 表示出来，代入 $\max(\Delta_a' - \Delta_a)$ ，然后求导即可。但是如果路径更长，那这个计算就很复杂了，为了解决任意长度路径下确定最优套利数量的问题，我们需要一个通用的方法。

考虑A->B->C兑换的情况，本来没有A->C的交易池，但是通过B这个桥梁，我们可以进行A->C的兑换，相当于A->C之间有一个虚拟交易池，我们能不能求得A->C这个虚拟交易池的等价参数呢？

假设A->C之间有个池子，参数为 $E_0, E_1$ ，我们现在要做的就是用A->B, B->C两个池子的参数表示出 $E_0, E_1$ 。

根据上面(1)(2)两式，可得：

$$\Delta_b = \frac{R_1 r \Delta_a}{R_0 + r \Delta_a} \quad (4)$$

$$\Delta_c = \frac{R_2 r \Delta_b}{R_1' + r \Delta_b} \quad (5)$$

将 (4) 代入 (5) 得到：

$$\Delta_c = \frac{\frac{R_0 R_1'}{R_1' + R_1 r} r \Delta_a}{\frac{r R_1 R_2}{R_1' + R_1 r} + r \Delta_a} \quad (6)$$

将 (6) 和 (4) 或者 (5) 进行形式上的对比就能得到：

$$E_0 = \frac{R_0 R_1'}{R_1' + R_1 r}$$

$$E_1 = \frac{r R_1 R_2}{R_1' + R_1 r}$$

这样我们就有了A->C的参数 $E_0, E_1$ ，考虑A->B->C->A的套利路径，我们可以结合A->C的参数 $E_0, E_1$ 和C->A的池子参数 $R_2', R_1'$ 按照上述同样的方法计算得到A->A的参数，假设为 $E_a, E_b$ ，如果 $E_a < E_b$ ，那就说明有套利空间，否则没有。这个过程可以应用到任意长度的套利路径，对每个路径，我们都能一步一步迭代求出等效的 $E_a, E_b$ ，并根据其大小关系判断是否有套利空间。

对于A->B->C->...->A这样一条路径，我们求出了等效的 $E_a, E_b$ ，如果存在套利空间，下一步就是计算最佳套利数量：

$$\Delta_a' = \frac{E_a r \Delta_a}{E_0 + r \Delta_a}$$

$$f = \Delta_a' - \Delta_a$$

这里的 $f$ 就是我们的利润函数，直接求导即可，最后得到最优套利数量为：

$$\Delta_a = \frac{\sqrt{E_a E_b r} - E_a}{r}$$

我们可以直接把计算最优套利数量的过程结合到上面的dfs中，得到利润最高的路径，参考代码：

```
def findArb(pairs, tokenIn, tokenOut, maxHops, currentPairs, path, bestTrades, count=5):
    for i in range(len(pairs)):
        newPath = path.copy()
        pair = pairs[i]
        if not pair['token0']['address'] == tokenIn['address'] and not pair['token1']['address'] == tokenIn['address']:
            continue
        if pair['reserve0']/pow(10, pair['token0']['decimal']) < 1 or pair['reserve1']/pow(10, pair['token1']['decimal']) < 1:
            continue
        if tokenIn['address'] == pair['token0']['address']:
            tempOut = pair['token1']
        else:
            tempOut = pair['token0']
        newPath.append(tempOut)
        if tempOut['address'] == tokenOut['address'] and len(path) > 2:
            Ea, Eb = getEaEb(tokenOut, currentPairs + [pair])
            newTrade = { 'route': currentPairs + [pair], 'path': newPath, 'Ea': Ea, 'Eb': Eb }
            if Ea and Eb and Ea < Eb:
```

```

        newTrade['optimalAmount'] = getOptimalAmount(Ea, Eb)
        if newTrade['optimalAmount'] > 0:
            newTrade['outputAmount'] =
getAmountOut(newTrade['optimalAmount'], Ea, Eb)
            newTrade['profit'] = newTrade['outputAmount'] -
newTrade['optimalAmount']
            newTrade['p'] = int(newTrade['profit']) / pow(10,
tokenOut['decimal'])
        else:
            continue

        bestTrades = sortTrades(bestTrades, newTrade)
        bestTrades.reverse()
        bestTrades = bestTrades[:count]
    elif maxHops > 1 and len(pairs) > 1:
        pairsExcludingThisPair = pairs[:i] + pairs[i+1:]
        bestTrades = findArb(pairsExcludingThisPair, tempOut, tokenOut,
maxHops-1, currentPairs + [pair], newPath, bestTrades, count)
    return bestTrades

```

## 2.工程实现

想光靠推推公式就能赚钱是不太行滴，还需要好的实现，这里实现的总体原则就是要快，以太坊区块间隔15s，你必须在15s内完成三件事情：

- 完成每个交易对reserves的更新
- 计算出最佳套利路径和数量
- 发交易

对于第一点，几个思路：

1. 交易对数量较少：直接batch request请求所有交易对的最新状态
2. 交易对数量多：先batch request获取一个全局状态，然后解析之后出现的每一个区块中所有交易的logs，如果涉及到对某一个交易对的更新（Swap, Sync, Mint, Burn），就更新本地对应交易对的reserves信息

















第二点：

- 想办法优化寻找环的速度，可以优化dfs，也可以尝试用bellman-ford之类的其他图算法

第三点：

- 发交易前可以调用uniswap的getAmountsOut函数检查一下是否真的能够获利，能获利且能覆盖手续费才发交易，避免手续费浪费

我自己实现了一下，其中一笔交易：

▶ Swap 0.001837877803868271 Ether For 0.029600841309442325  TRB On  Uniswap  
 ▶ Swap 0.029600841309442325  TRB For 0.796430316556100961  DAI On  Uniswap  
 ▶ Swap 0.796430316556100961  DAI For 0.346048644105336869  RPL On  Uniswap  
 ▶ Swap 0.346048644105336869  RPL For 36.504804784313389539  BUIDL On  Uniswap  
 ▶ Swap 36.504804784313389539  BUIDL For 20.691932331434012957  OXT On  Uniswap  
 ▶ Swap 20.691932331434012957  OXT For 0.018473813918786526 Ether On  Uniswap

▶ From   To Uniswap V2: TRB 3 For 0.001837877803868271 (\$0.65)  Wrapped Ethe... (WETH)  
 ▶ From Uniswap V2: TRB 3 To Uniswap V2: TRB-D... For 0.029600841309442325 (\$0.69)  Tellor Tribu... (TRB)  
 ▶ From Uniswap V2: TRB-D... To Uniswap V2: DAI-R... For 0.796430316556100961 (\$0.80)  Dai Stableco... (DAI)  
 ▶ From Uniswap V2: DAI-R... To Uniswap V2: RPL-B... For 0.346048644105336869 (\$0.74)  Rocket Pool (RPL)  
 ▶ From Uniswap V2: RPL-B... To Uniswap V2: OXT-B... For 36.504804784313389539 (\$63.59)  DFOHub (BUIDL)  
 ▶ From Uniswap V2: OXT-B... To Uniswap V2: OXT 2 For 20.691932331434012957 (\$6.23)  Orchid (OXT)  
 ▶ From Uniswap V2: OXT 2 To   For 0.018473813918786526 (\$6.53)  Wrapped Ethe... (WETH)

从0.0018个ETH变成了0.018个，不过没能覆盖手续费:(

### 3.结语

Contact: [ccyanxyz@gmail.com](mailto:ccyanxyz@gmail.com)

Donation: 0x0af66d3641640755878F0d72A610FC2EEA856Cd6

最后，Uniswap套利目前其实已经竞争十分激烈了，但是还是有很多其他东西值得探索的，比如模型更复杂的curve.fi，balancer.exchange，结合flashloan的跨dex套利等等，happy hacking :)