Optimal stable coin folding strategy

Stefan Duprey, Frederic Fayes, Tom Suter
SwissBorg
stefan@swissborg.com, frederic@swissborg.com, tom@swissborg.com

***Abstract*—A basic 'Lend/Borrow' strategy on the Aave protocol is one of the best and safest way to generate yield on your stable coins.**
Those strategies are perfect fits for multiple 'Lend/Borrow' iterations or 'Folding' due to the inherent pegging of stable coins and the low liquidation risk incurred.
The question of the optimal AUM allocation between multiple pools is a complex one.
APY dilution generate non linear effects, which can be solved through advanced non linear optimization algorithms/solvers, but those solvers are computationally intensive and won't transport easily into solidity code and limited computations due to high gas prices.
We here propose a linearization which give a simpler constrained quadratic optimization problem, whose solution can be expressed analytically through Lagrange multiplier.
That analytical solution can be seamlessly computed in solildity and live on a blockchain even with high gas prices.

I. Folding strategy

Iteration	Supply	Borrow
0	A	A*X
1	A*X	A*X*X
...		
n	A*X^n	A*X^{n+1}

The total amount supplied after n iterations is:

$$S = \sum_{i=0}^{n} A * X^i = A\frac{1-X^{n+1}}{1-X} \tag{1}$$

$$S \approx \sum_{i=0}^{+\infty} A * X^n = A\frac{1}{1-X}$$

The total amount borrowed after n iterations is:

$$B = \sum_{i=0}^{n} A * X^{i+1} = A * X\frac{1-X^{n+1}}{1-X} \tag{2}$$

$$B \approx \sum_{i=0}^{+\infty} A * X^{n+1} = A\frac{X}{1-X}$$

By defining the total APY on the supply side as the sum of the base APY and the liquidity incentive APY, here comes :

$$\text{sAPY} = \text{supply APY} + \text{incentive supply APY} \tag{3}$$

$$\text{bAPY} = \text{borrow APY} - \text{incentive borrow APY} \tag{4}$$

$$\text{total APY} = \frac{A}{1-X} * sAPY - \frac{A*X}{1-X} * bAPY \tag{5}$$

This total APY is obviously strictly increasing as a function of the percentage ratio X.

$$\frac{d}{dX}\left(\text{total APY}\right) = \frac{(sAPY - bAPY)}{(1-X)^2} > 0 \tag{6}$$

The upper bound for X is only set by type of pools according to the incurred liquidation threshold.
For stable coin pools with historicaly prooved pegging, X can be chosen as high as 72%.

II. Allocating between pools : an optimization problem

A. Aave protocol : a trade off to maximize utilization/assured withdrawal

Aave protocol implements at its core a regulating mechanism which aims at maximizing profits for liquidity supplier while keeping them safe and always able to withdraw their liquidity.
The utilization rate, which is just the ratio of the total borrowed divided by the total supplied:

$$U = \frac{B_t}{L_t} \tag{7}$$

is algorithmically targeted to stay close to an optimal $U_{optimal}$ fixed by Aave governance per different pools.
The targeting mechanism can be summed up to the definition of two different borrow rate growth regime according to the value of U under or above U$_{optimal}$.

$$R_b = \begin{cases} R_{b_0} + \frac{U}{U_{optimal}} * R_{slope1}, \text{ if } U \leq U_{optimal} \\ R_{b_0} + R_{slope1} + \frac{U-U_{optimal}}{1-U_{optimal}} * R_{slope2},\ U \geq U_{optimal} \end{cases}$$

R$_b$ is the borrow rate, which will rise sharply after $U > U_{optimal}$ to prevent the ratio from being too close to one, which could keep suppliers from withdrawing their fund.
R_{b0} is the constant base variable borrow rate.
The supply rate is then defined as the borrow rate multiplied by the utilization ratio.

$$R_s = R_b * U \tag{8}$$

The greater the utilization ratio, the more yield suppliers will get.

$$R_s - R_b = R_b * (U-1) < 0 \tag{9}$$

B. Stable coin pool parameters

Aave protocol governance has assigned the following parameters all common across stable pool reserves :

Coin	U$_{optimal}$	base	slope1	slope2
DAI	90%	0%	4%	60%
USDC	90%	0%	4%	60%
USDT	90%	0%	4%	60%

C. Liquidity incentives APR

1) Definition: The definition of the liquidity incentives can be found in Aave V2 protocol white paper :

$$isAPR = 100 * \frac{aEmissionPerYear * rewardPrice}{totalATokenSupply * tokenPrice} \quad (10)$$

where the parameters match the following in the smart contract are detailed in the solidity section

$$ibAPR = 100 * \frac{vEmissionPerYear * rewardPrice}{totalCurrentVariableDebt * tokenPrice} \quad (11)$$

where the parameters match the following in the smart contract are detailed in the solidity section

We see that from the assumption of a fixed reward price and emission rate, the borrow APR depends on the total current debt :

$$APR_b = \frac{cte_b}{B_t} \quad (12)$$

And the supply APR depends on the total liquidity:

$$APR_s = \frac{cte_s}{L_t} \quad (13)$$

2) Compounding: We will here assume that those incentive annual APR are indeed harvested and reinvested in the strategy leading to an annualized compounded APY:

$$APY = (1 + \frac{APR}{secondsPerYear})^{secondsPerYear} - 1 \quad (14)$$

The reverse equation is :

$$APR = ((1 + APY)^{\frac{1}{secondsPerYear}} - 1) * secondsPerYear \quad (15)$$

D. Dilution effect

When a proportion X of an additional amount AUM is dispatched to a specific liquidity pool. This pool will incur a dilution effect on its liquidity.
$L_t \leftarrow L_t + X * AUM$

E. First order approximation

We can simplify the non linear term under the assumption:

$$\frac{1}{L_t + X * AUM} = \frac{1}{L_t} * \frac{1}{1 + \frac{X}{\frac{L_t}{AUM}}} = \frac{1}{L_t} * \sum_{n=0}^{n=+\infty} (-1)^n \frac{X^n}{\frac{L_t}{AUM}^n}$$

If $X << \frac{L_t}{AUM}$, as a first order approximation:

$$\frac{1}{L_t + X * AUM} = \frac{1}{L_t} - \frac{AUM}{L_t^2} * X$$

Here comes the new diluted utilization ratio :

$$U(X) = \frac{B_t}{L_t + X * AUM} = \frac{B_t}{L_t} * \frac{1}{1 + \frac{X}{\frac{L_t}{AUM}}} \approx \frac{B_t}{L_t} * (1 - \frac{X}{\frac{L_t}{AUM}}) \quad (16)$$

Here comes the new diluted incentive APYs:

$$APR_s(X) = \frac{cte_s}{L_t + X * AUM} \approx cte_s * (\frac{1}{L_t} - \frac{AUM}{L_t^2} * X) \quad (17)$$

After multiple folding and , we then linearize as in 17
It comes after linearization :

$$APR(X) = APR_s(X) + APR_s(X)$$
$$= \frac{cte_s}{L_t + \frac{X * AUM}{1 - X_{leverage}}} + \frac{cte_b}{B_t + \frac{X_{leverage} * X * AUM}{1 - X_{leverage}}}$$
$$\approx cte_s * (\frac{1}{L_t} - \frac{AUM}{L_t^2} * X)$$
$$APR(X) \approx A_0 + A_1 * X \quad (18)$$

where

$$A_0 = \frac{cte_s}{L_t} + \frac{cte_b}{B_t} \quad (19)$$

$$A_1 = -\frac{cte_s * AUM}{L_t^2 * (1 - X_{leverage})} - \frac{cte_b * AUM * X_{leverage}}{B_t^2 * (1 - X_{leverage})} \quad (20)$$

F. Optimization problem

When having to allocate an amount AUM between n pools, one will look at optimizing the total generated yield :

$$(\omega_1, ..., \omega_n) = \arg \max_{\sum_{i=1}^{i=n} \omega_i = 1, 0 \le \omega_i \le 1} \sum_{i=1}^{i=n} \omega_i \times tAPY_i (\omega_i * AUM) \quad (21)$$

where the total APYs APY_i for each pool i are defined in equations 3 and 4.
The 14 equation shows that the liquidity incentive APYs on Aave are actually the main prominent reason for choosing a specific pool.
We can in a first approximation try to solve the simpler problem :

$$(\omega_1, ..., \omega_n) = \arg \max_{\sum_{i=1}^{i=n} \omega_i = 1, 0 \le \omega_i \le 1} \sum_{i=1}^{n} \omega_i \times APR_i (\omega_i * AUM) \quad (22)$$

G. Constrained quadratic optimization for the linearized problem

Under the conditions $X << \frac{L_t}{AUM}$ and $U < U_{optimal}$, by using the first order approximations 18 :

$$(\omega_1, ..., \omega_n) = \arg \max_{\sum_{i=1}^{i=n} \omega_i = 1, 0 \leq \omega_i \leq 1} \sum_{i=1}^{n} \omega_i \times (A_{0i} + A_{1i} * \omega_i)$$

where A_{0i}, A_{1i} are the constants for the pool i.
By calling

$$J(x) = -\sum_{i=1}^{n} x_i \times (A_{0i} + A_{1i} * x_i)$$

The optimization problem becomes a constrained quadratic minimization problem

$$\min_{\sum x_i = 1, 0 \leq x_i \leq 1} J(x)$$

H. Analytical solution from Lagrange multiplier

By rewriting the constraint as :

$$g(x) = \sum_{i=1}^{n} x_i$$

We can compute the optimal solution using the Lagrange multiplier and the modified functional taking into the equality constraints and the inequality constraints using KKT conditions and slack variables:

$$J'(x) = J(x) + \lambda (g(x) - 1)$$
$$+ \sum_i \theta_i * (x_i - s_i^2) + \sum_i \gamma_i * (1 - t_i^2 - x_i)$$

We solve the Lagrangian equations :

$$\begin{cases} \nabla_x J'(x) = 0 \\ \nabla_\lambda J'(x) = 0 \\ \nabla_\theta J'(x) = 0 \end{cases} \tag{23}$$

The ∇_x conditions give the following :

$$\begin{pmatrix} A_{01} + 2 * A_{11} * x_1 + \lambda + \theta_1 - \gamma_1 \\ \vdots \\ A_{0i} + 2 * A_{1i} * x_i + \lambda + \theta_i - \gamma_i \\ \vdots \\ A_{0n} + 2 * A_{1n} * x_n + \lambda + \theta_n - \gamma_n \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

First case, if all θ_i and γ_i are null, it means that all slack variables s_i^2 and t_i^2 are positive and the constraints are not active.
The x solution reads :

$$\forall i \in [1, \ldots, n], x_i = -\frac{\lambda + A_{0i}}{2 * A_{1i}} \tag{24}$$

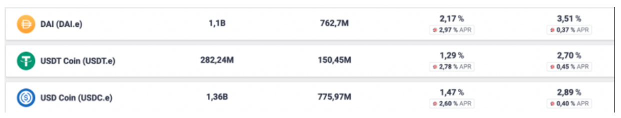
and the Lagrange multiplier can then be quantified through the constraint :

$$\lambda = \frac{-1 - \sum_{i=1}^{n} \frac{A_{0i}}{2 * A_{1i}}}{\sum_{i=1}^{n} \frac{1}{2 * A_{1i}}} \tag{25}$$

In the case where on one x_i does not respect the inequalities, it means that either $x_i = 1 or 0$ depending on if θ_i or γ_i is not null. The solution is then obtained recursively by reiterating the 24 and 25 with the remaining x_is until all are found respecting the inequalities.

III. Allocating between pools : an optimization problem

We here present optimization results for the following snapshot parameter for a three pools (DAI, USDC, USDT) allocation on Aave Avalanche.



DAI (DAI.e)	1,1B	762,7M	2,17 % 2,97 % APR	3,51 % 0,37 % APR
USDT Coin (USDT.e)	282,24M	150,45M	1,29 % 2,78 % APR	2,70 % 0,45 % APR
USD Coin (USDC.e)	1,36B	775,97M	1,47 % 2,60 % APR	2,89 % 0,40 % APR

Fig. 1. Optimization parameters

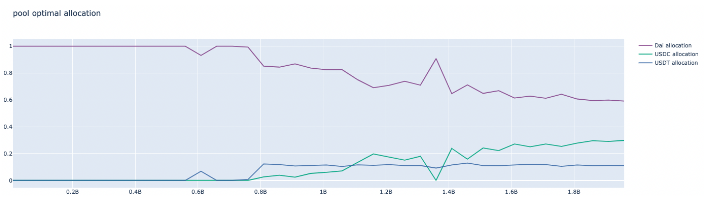


Fig. 2. Non linearized exact solution

A.

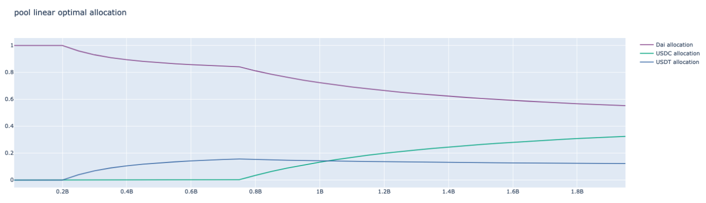


Fig. 3. Approximative linearized solution

IV. Solidity code for optimization

The new diluted incentive APYs expression 17 and both equations 27 and 10 give us the expression for our A_{0i} and A_{1i}

$$isAPR = 100 * \frac{a * r * w}{L_t * t * d} = \frac{cte_s}{L_t} \tag{26}$$

where $cte_s = 100 * \frac{a * r * w}{t * d}$ the parameters match the followings in the smart contract :

$$\begin{cases} isAPR = incentiveDepositAPRPercent \\ a = aEmissionPerYear \\ r = REWARDPRICEETH \\ w = WEIDECIMALS \\ t = TOKENPRICEETH \\ d = TOKENDECIMALS \\ L_t = totalATokenSupply \end{cases}$$

$$ibAPR = 100 * \frac{v * r * w}{B_t * t * d} = \frac{cte_b}{B_t} \quad (27)$$

where $cte_b = 100 * \frac{v*r*w}{t*d}$ the parameters match the followings in the smart contract :

$$\begin{cases} ibAPR = incentiveBorrowAPRPercent \\ v = vEmissionPerYear \\ r = REWARDPRICEETH \\ w = WEIDECIMALS \\ B_t = totalCurrentVariableDebt \\ t = TOKENPRICEETH \\ d = TOKENDECIMALS \end{cases}$$

By using equations 20 and 19, we can compute A_{0i} and A_{1i} for each pool.
Equations 25 and **??** give us then the optimal pool allocation.

V. Conclusion

We have managed to simplify the optimization problem and transport it to EVM like blockchain languages.
The solution can scale up to at least ten pools without being too computationally intensive.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[2] V. Buterin *et al.*, "Ethereum white paper," *GitHub repository*, vol. 1, pp. 22–23, 2013.

[3] "Aave white paper," *https://github.com/aave/protocol-v2/blob/master/aave-v2-whitepaper.pdf*.

[4] "Aave documentation," *https://docs.aave.com/portal/*.