Wombat — An Efficient Stableswap Algorithm

Wombat Team

Wombat Exchange

Abstract. Curve Finance invented the first stableswap-focused algorithm. However, its algorithm involves (1) solving complex polynomials and (2) requiring assets in the pool to have the same size of liquidity. This paper introduces a new stableswap algorithm - Wombat, to address these issues. Wombat uses a closed-form solution, so it is more gas efficient and adds the concept of asset-liability management to enable single-side liquidity provision, which increases capital efficiency. Furthermore, we derive efficient algorithms from calculating withdrawal or deposit fees as an arbitrage block. Wombat is named after the short-legged, muscular quadrupedal marsupials native to Australia. As Wombats are adaptable and habitat-tolerant animals, the invariant created is also adaptable and tolerant to liquidity changes.

Keywords: Decentralized Finance · Automated Market Maker · Stableswap · Asset Liability Management · Smart Contract.

1 Introduction

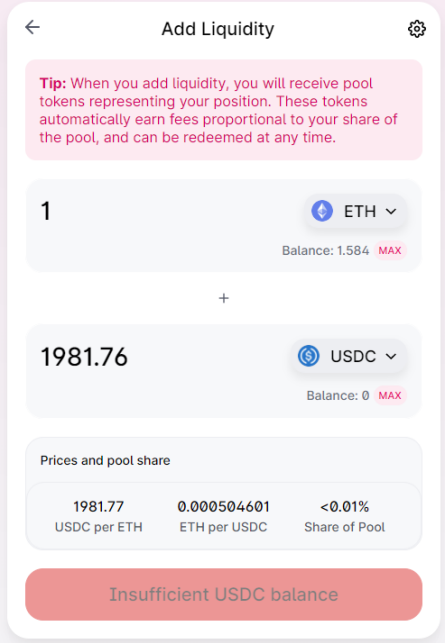
Automated Market Maker (AMM) is essential for Decentralized Finance (DeFi) traders to swap tokens on-chain. As one of the essential foundations of DeFi, AMM is integral to the success of any blockchain, whether it be the past, present, or future. An efficient AMM will have powerful effects on the ecosystem providing decentralized apps with an efficient platform to build upon and thus, fueling the next evolution of finance. Efforts from Uniswap, Curve, and Balancer have contributed to developing a quick, efficient, and effective AMM design. Most of the existing AMM designs use an invariant curve to create an efficient swap with reasonable slippage directed at assets with a range of volatility characteristics. Uniswap, one of the pioneers of AMM design, designed an AMM that focused on more volatile assets such as ETH and BTC. The invariant curve was designed to be more parabolic to ensure that price change follows demand shock [2]. Noting the characteristics of pegged assets, Curve proposed an invariant curve that was more linear in the interior part of the curve but more parabolic when x or y tended to infinity [3]. Such a design allows a more efficient swap between pegged assets such as stablecoins.

As DeFi pushes toward higher efficiency, one of the main issues with the current designs is their to scale while simultaneously remaining capital efficient. These scaling issues are bounded by a protocol's inherent designs and results in complicated token-pair compositions. It must be noted that computational inefficiencies affect the end-users in terms of fees forfeited as the cost is proportional to complexity. On top of the cost, users are forced to deal with complex user interfaces due to the inflexible algorithm, something that Wombat can both simplify and make cost-effective. The development of Wombat aims at resolving such deficiencies.

The invariant curve is a relationship between two or more token amounts. Mathematically speaking, it is a level set of a function that maps token amounts to a real number [1]. To simplify the discussion, we focus on the two-token case first. For example, consider $x + y = k$ for some constant k. That means, whenever a swap of Δx to Δy happens on this invariant curve, we know that $x_0 + \Delta x + y_0 + \Delta y = k$, which implies $\Delta x = -\Delta y$. On the xy-plane, the invariant curve is a linear line. The invariant function $x + y = k$ is called the Constant Sum Market Maker (CSMM). Under CSMM, x could be swapped for the same amount of y unless y runs out of liquidity. Another example of an invariant curve would be $xy = k$ for some constant k, also known as Constant Product Market Maker (CPMM) used by Balancer and Uniswap [2]. The invariant curve on xy-plane is more hyperbolic compared to CSMM.

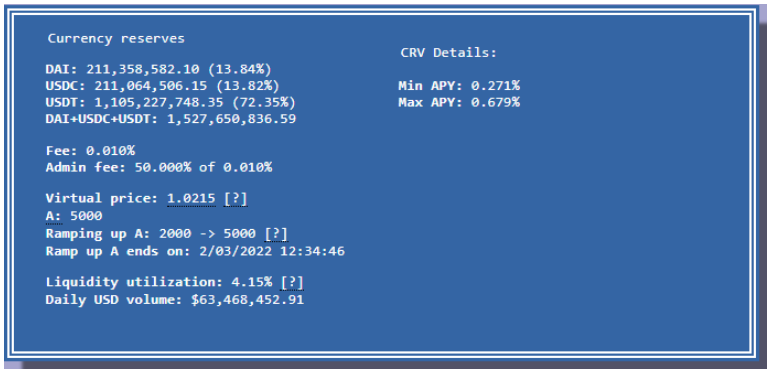
One main problem of many existing AMMs is that they require injecting tokens in pairs [5]. The ratio of these two tokens would, in turn, represent the price ratio, and the functional behavior of the CPMM would control the price movement. The screenshot below from Uniswap illustrates that a user needs to add ETH and USDC to the pool simultaneously.

Fig. 1: Uniswap Screenshot



In the case of Curve Finance, the equilibrium is achieved when the assets inside the pool have the same liquidity amount. Otherwise, the invariant curve will become more parabolic and inherently imply different trading prices for assets within the pool. The screenshot below shows that USDT is trading at a discount compared to USDC and DAI due to the pool imbalance.

Fig. 2: Curve Screenshot



In the world of cryptocurrencies, there is a bias towards specific stablecoins. At the time of writing, the market capitalization for USDT, USDC, and DAI are 80B, 53B, and 6.6B, respectively. With Curve's setup, the pool size will be constrained by the asset with the lowest supply, i.e., DAI, since it is uneconomic to deposit USDT and USDC further when the DAI supply is saturated. For this reason, it will be beneficial to remove the liquidity constraint to increase the system's scalability. Following Wombat's solution, we can have a more efficient algorithm that is more capital and gas efficient while removing scalability issues.

Lastly, Curve's stableswap algorithm is computationally inefficient when more assets are in the pool. The required gas is summarized below in section 2.1.

2 Wombat's Design

In light of the problems above, Wombat's design aims to:

1. Make the algorithm more gas efficient;
2. Allow single-sided liquidity provision;
3. Get rid of the same-liquidity constraint for assets in the same pool.

2.1 Wombat's Invariant Curve

Stableswap concerns the swap of pegged tokens and assets. Denote the set of tokens with the same peg as

$$\mathcal{T} := \{\text{token } k : V(\text{token } k) = C\},$$

where C is some fixed real number and $V(\cdot)$ is a value function that maps a token to its value metric, usually in USD.

The main goal of a stableswap invariant is to create a function to mimic CSMM when x_k are close to each other. Revisit the Curve's stableswap invariant:

$$An^n \sum_{k\in\mathcal{T}} x_k + D = ADn^n + \frac{D^{n+1}}{n^n \prod_{k\in\mathcal{T}} x_k},$$

where $A > 0$ denotes the *amplification factor* and D is the level preset by the state of token pools, which is a constant with respect to swap.

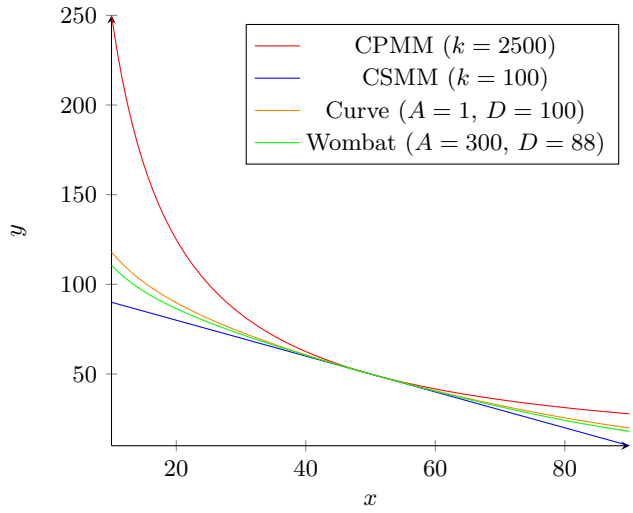
Here, Wombat introduces a new algorithm that can achieve a similar result as Curve's stableswap invariant, yet much easier to solve:

$$\sum_{k \in \mathcal{T}} \left(x_k - \frac{A}{x_k} \right) = D. \tag{1}$$

The intuition is that when x_k are close to each other, the reciprocal terms will be minimized, making the function closely mimic the linear term, i.e., the CSMM. However, if x_k are apart from each other, the reciprocal terms will grow faster and create a steeper slope for the curve.

For Curve Finance, solving the invariant constant D involves solving a high-degree polynomial. As for Wombat, solving D is plain simple by calculating the summation. For the case of a swap, Curve would involve solving a function with product terms, while Wombat can be simply solved with a quadratic equation.

Fig. 3: CFMM Comparisons



Wombat's simplicity enables it to be significantly more gas efficient since the cost of smart contract development is based on the amount of calculations. As shown in Table 1, the maximum gas usage by Curve is more than eight times the maximum displayed by Wombat.

Wombat Invariant Curve	Curve Finance Invariant Curve
Minimum Gas: 140,000 Maximum Gas: 180,000	Minimum Gas: 130,000 Maximum Gas: 1,500,000

Table 1: Comparison of Algorithm Gas Costs

As illustrated in the comparison table of the algorithm gas costs, Wombat's invariant curve behaves very similarly to Curve, but the algorithm is improved.

2.2 Enhanced Wombat's Invariant Curve

While equation (1) has presented a better alternative to Curve's stableswap invariant, Wombat further adopts the asset-liability management concept proposed by Platypus Finance [5] to get rid of liquidity constraints. The coverage ratio is used as the input for the invariant curve, which allows single-sided liquidity injection and arbitrary exchange across tokens within the preset pool.

To achieve so, we first forgo the ability to represent token price ratios since they are exogenously set to be 1. The moving parameters of the CFMM need not be the token's liquidity in the pool; instead, we change them into the coverage ratios r of the tokens, that is,

$$r_k = \frac{A_k}{L_k},$$

where r_k is the *coverage ratio* of token k pool, A_k is the *asset* available for withdrawal and swap in token k pool, and L_k is the *liability* in token k pool. We can also understand the liability L_k as the amount of token k injected in the pool as a deposit, and they are subject to withdrawal at any given time.

Now, we define our modified CFMM as follows: the invariant curve is given by

$$\sum_{k \in \mathcal{T}} L_k \left(r_k - \frac{A}{r_k} \right) = D.$$

Since the equation is now revolving around coverage ratio of tokens instead of liquidity, the system can get rid of liquidity constraint to enable features such as single-sided deposit and heterogeneous liquidity for equilibrium.

2.3 Swap Mechanism

For simplicity, we will focus on the two-token case, which is going to be the primary action performed by the traders. However, it is also worth-mentioning that our design allows a series of swaps simultaneously (i.e., from n tokens to m tokens) as long as it fits the CFMM.

Consider two tokens, i and j. We look into the case when a trader swaps from token i to token j. Let the initial coverage ratios of the two pools be $r_{i,0}$ and $r_{j,0}$, respectively. With the swap, the coverage ratios of the two pools change from $(r_{i,0}, r_{j,0})$ to (r_i, r_j), where

$$r_i = r_{i,0} + \frac{\Delta_i}{L_i} \quad \text{and} \quad r_j = r_{j,0} + \frac{\Delta_j}{L_j}. \tag{2}$$

Here, $\Delta_i > 0$ is specified by the trader, while an unknown Δ_j is to be determined by the CFMM. Since the level set specified by the CFMM remains unchanged before and after the swap, we have

$$\sum_{k\in\mathcal{T}\setminus\{i,j\}} L_k\left(r_k - \frac{A}{r_k}\right) + L_i\left(r_{i,0} - \frac{A}{r_{i,0}}\right) + L_j\left(r_{j,0} - \frac{A}{r_{j,0}}\right) = D = \sum_{k\in\mathcal{T}} L_k\left(r_k - \frac{A}{r_k}\right).$$

If we define the constant $D_{i,j} = L_i\left(r_{i,0} - \frac{A}{r_{i,0}}\right) + L_j\left(r_{j,0} - \frac{A}{r_{j,0}}\right)$, then

$$L_i\left(r_i - \frac{A}{r_i}\right) + L_j\left(r_j - \frac{A}{r_j}\right) = D_{i,j} \tag{3}$$

After a series of algebraic operations, we obtain the function

$$r_j = \frac{-b + \sqrt{b^2 + 4A}}{2},$$

where

$$b = \frac{L_i}{L_j}\left(r_i - \frac{A}{r_i}\right) - \frac{D_{i,j}}{L_j}.$$

Note that we reject the solution that $r_j < 0$. Knowing $r_{j,0}$ and r_j yields us knowledge on Δ_j.

Next, we focus on the properties of the invariant curves. Using implicit differentiation on equation (3), we can obtain the derivative

$$\frac{dr_j}{dr_i} = -\frac{L_i\left(1+\frac{A}{r_i^2}\right)}{L_j\left(1+\frac{A}{r_j^2}\right)} < 0.$$

The negative derivative tells us that given the same invariant curve, any swap action that increases r_i would reduce r_j. In other words, the more token i is injected into the system, the more token j the trader receives. Without explicitly calculating the second-order derivative, we can study the behavior of the first-order derivative by noticing that an increase in r_i leads to a decrease in $L_i\left(1+\frac{A}{r_i^2}\right)$, and a decrease in r_j leads to an increase in $L_j\left(1+\frac{A}{r_j^2}\right)$. Hence, $\frac{dr_j}{dr_i}$ would become less negative when r_i increases. This implies that $\frac{d^2r_j}{dr_i^2} > 0$. A convex invariant curve indicates that continuous injection of token i would yield the trader fewer and fewer token j in return per unit of token i injected.

2.4 Slippage Analysis

In the remainder of this section, we take a closer look at the slippage behavior of the invariant curve. Recalling that r_i and r_j relate to Δ_i and Δ_j as described in equation (2), we have

$$\frac{d\Delta_j}{d\Delta_i} = \frac{d\Delta_j}{dr_j}\cdot\frac{dr_j}{dr_i}\cdot\frac{dr_i}{d\Delta_i} = L_j\cdot\left(-\frac{L_i\left(1+\frac{A}{r_i^2}\right)}{L_j\left(1+\frac{A}{r_j^2}\right)}\right)\cdot\frac{1}{L_i} = -\frac{1+\frac{A}{r_i^2}}{1+\frac{A}{r_j^2}}. \quad (4)$$

If $r_i = r_j$, then

$$\left.\frac{d\Delta_j}{d\Delta_i}\right|_{r_i=r_j} = -1,$$

which indicates that every token i injected would yield one token j at this particular state. If $r_i < r_j$, then the derivative in equation (4) is less than -1, so every token i injected yields more than one token j. On the other hand, if $r_i > r_j$, then every token i injected yields less than one token j. Notice that swapping token i to token j increases r_i and lowers r_j. Hence, when $r_i < r_j$, the arbitrage opportunity encourages traders to swap token i to token j, thus reducing the gap between the two coverage ratios. Conversely, if $r_i > r_j$, swapping token i to token j widens the gap between the two coverage ratios and is discouraged with a loss for the traders.

Theorem 1. *The equilibrium state of the AMM is that for all* $i, j \in \mathcal{T}$, $r_i = r_j$.

Proof. Assuming the contrary, there exist $i, j \in \mathcal{T}$ such that $r_i < r_j$. Then one can find an arbitrage by swapping token i to token j until $r_i = r_j$.

3 Desirable Properties of AMMs

There are three desirable properties of AMMs, including path independence, liquidity sensitivity, and no-arbitrage [4]. Wombat AMM is capable to achieve path independence and is liquidity sensitive by design.

3.1 Path Independence

Path independence is crucial for AMMs because there should always exist a path to return from any state to the equilibrium in the system. The importance lies in the reality that if the system cannot regain equilibrium, then the system is out of balance and there may be arbitrage opportunities that will negatively impact the system's stability. Path independence ensures that a trader cannot place a series of trades and profits without assuming a potential risk. Furthermore, it helps provide a minimum representation of the current state in which we are only required to know the quantity vector. Finally, in a path independent system, traders do not to need to discover a strategy on how they trade.

Any swap action is performed along the same invariant curve, and the liabilities of all tokens remain unchanged throughout the process. Hence, after the coverage ratios change from $(r_{i,0}, r_{j,0})$ to (r_i, r_j) as described in equation (2), the coverage ratios can be returned to the status quo by swapping $-\Delta_j$ token j to token i. Therefore, the Wombat AMM is path independent.

3.2 Liquidity Sensitivity

Liquidity sensitivity is imperative because it makes the protocol desirable in its function, where a trade moves prices less in a liquid market than in an illiquid one. This is important in any trading market, where small trades should not drastically affect the market where sufficient liquidity is present. With this design, our AMM is able to mimic and emulate highly liquid traditional markets where prices can be held stable and more resistant to small trades, enabling a fairer and more transparent platform.

Recall that r_j can be written explicitly as a function of r_i, namely

$$r_j = \frac{-b + \sqrt{b^2 + 4A}}{2},$$

where

$$b = \frac{L_i}{L_j}\left(r_i - \frac{A}{r_i}\right) - \frac{D_{i,j}}{L_j}$$

and

$$D_{i,j} = L_i r_i + L_j r_j - A\left(\frac{L_i}{r_i} + \frac{L_j}{r_j}\right) = L_i r_{i,0} + L_j r_{j,0} - A\left(\frac{L_i}{r_{i,0}} + \frac{L_j}{r_{j,0}}\right).$$

Note that A, r_i, $r_{i,0}$, L_i, and $r_{j,0}$ are independent of L_j while r_j and $D_{i,j}$ are dependent of L_j. Also, recall from equation (2) that $\Delta_j = (r_j - r_{j,0})L_j < 0$ in the swap of token i to token j. Now we consider the partial derivative of Δ_j with respect to L_j.

$$\begin{aligned}
\frac{\partial \Delta_j}{\partial L_j} &= \frac{\partial r_j}{\partial L_j} L_j + r_j - r_{j,0} \\
&< \frac{1}{2}\frac{\partial b}{\partial L_j}\left(\frac{b}{\sqrt{b^2 + 4A}} - 1\right) L_j \\
&= \frac{1}{2}\left(-\frac{1}{L_j^2}\left(L_i\left(r_i - \frac{A}{r_i}\right) - D_{i,j}\right) + \frac{1}{L_j}\left(r_{j,0} - \frac{A}{r_{j,0}}\right)\right)\left(\frac{b}{\sqrt{b^2 + 4A}} - 1\right) L_j \\
&= \frac{1}{2}\left((r_{j,0} - r_j) - A\left(\frac{1}{r_{j,0}} - \frac{1}{r_j}\right)\right)\left(\frac{b}{\sqrt{b^2 + 4A}} - 1\right) < 0.
\end{aligned}$$

The last inequality holds since $r_j < r_{j,0}$ and $\dfrac{b}{\sqrt{b^2 + 4A}} < 1$. Hence, if we keep r_i, $r_{i,0}$, L_i, and $r_{j,0}$ unchanged, then Δ_j becomes more negative when L_j grows, meaning that the yield of token j increases.

4 Arbitrage Block

Wombat's design suffers from arbitrage issues when a withdrawal or deposit is made. The arbitrage issue is partly due to the asset-liability management design, as there are opportunities for the coverage ratios to be manipulated by rogue traders. A simple numerical example is illustrated below (assume A = 0.01).

There are two tokens, X and Y, whose assets and liabilities start at 100. The attacker first swaps 50 units of token X for 49.36 units of token Y, and the coverage ratio of token Y drops from 1 to 0.51.

Asset	Liability
Token X: 100	Token X: 100
Token Y: 100	Token Y: 100

Asset	Liability
Token X: 150	Token X: 100
Token Y: 50.64	Token Y: 100

Table 2: Swap X for Y

The attacker is also a major liquidity provider for token Y. He withdraws 50 unit of token Y, and the coverage ratio of token Y drops from 0.51 to 0.013.

Asset	Liability
Token X: 150	Token X: 100
Token Y: 50.64	Token Y: 100

Asset	Liability
Token X: 150	Token X: 100
Token Y: 0.64	Token Y: 50

Table 3: Withdraw Y

Finally, the attacker swaps the 49.36 units of token Y he obtained from Step 1 back to token X. Note that the attacker gets a better exchange rate in Step 3 than Step 1 because the coverage ratio is being manipulated. The attacker earns $(49.36 - 50) - (49.36 - 86.91) = \$36.91$ from the whole operation.

Asset	Liability
Token X: 150	Token X: 100
Token Y: 0.64	Token Y: 50

Asset	Liability
Token X: 63.09	Token X: 100
Token Y: 50	Token Y: 50

Table 4: Swap Y for X

Therefore, the goal for this section is to provide an algorithm to block the arbitrage and solve it mathematically.

4.1 Changes in the Asset and Liability

Recall once again that our modified CFMM is

$$\sum_{k\in\mathcal{T}} L_k \left(r_k - \frac{A}{r_k} \right) = D.$$

If the system returns to the equilibrium state as described by Theorem 1, then the global equilibrium coverage ratio r^* satisfies

$$\begin{aligned} \sum_{k\in\mathcal{T}} L_k \left(r^* - \frac{A}{r^*} \right) &= D \qquad (5) \\ \left(\sum_{k\in\mathcal{T}} L_k \right) (r^*)^2 - Dr^* - A \left(\sum_{k\in\mathcal{T}} L_k \right) &= 0 \\ (r^*)^2 - \frac{D}{\sum_{k\in\mathcal{T}} L_k} r^* - A &= 0. \end{aligned}$$

Solving the quadratic equation, we obtain $r^* = \dfrac{-b + \sqrt{b^2 + 4A}}{2}$, where $b = -\dfrac{D}{\sum_{k\in\mathcal{T}} L_k}$.

In a withdrawal or deposit of token i, let δ_i^A and δ_i^L denote the change in the asset and the change in the liability, respectively. Here, $\delta_i^L \geq -L_i$ is specified by the trader, where $\delta_i^L < 0$ denotes a withdrawal and $\delta_i^L > 0$ denotes a deposit. The quantity δ_i^A is to be solved as a function of δ_i^L to block the arbitrage. The new global equilibrium coverage ratio $r^{*\prime}$ after the withdrawal or deposit is given by

$$r^{*\prime} = \frac{\delta_i^L + \left(\sum_{k\in\mathcal{T}} L_k \right) r^*}{\delta_i^L + \sum_{k\in\mathcal{T}} L_k}. \qquad (6)$$

In this equation, $\left(\sum_{k\in\mathcal{T}} L_k \right) r^*$ denotes the total asset if the system returns to the equilibrium state. By adding δ_i^L, the numerator expresses the total asset after withdrawal or deposit if the system first returns to the equilibrium state, while the denominator denotes that total liability after withdrawal or deposit. From $r^{*\prime}$, we can solve for the new constant D' such that

$$\left(\delta_i^L + \sum_{k\in\mathcal{T}} L_k \right) \left(r^{*\prime} - \frac{A}{r^{*\prime}} \right) = D'. \qquad (7)$$

With the new constant D', we can backward deduce the coverage ratio r_i' needed to maintain the equilibrium of the system:

$$(L_i + \delta_i^L)\left(r_i' - \frac{A}{r_i'}\right) + \sum_{k\in\mathcal{T}\setminus\{i\}} L_k\left(r_k - \frac{A}{r_k}\right) = D' \tag{8}$$

$$(L_i + \delta_i^L)(r_i')^2 + \left(\sum_{k\in\mathcal{T}\setminus\{i\}} L_k\left(r_k - \frac{A}{r_k}\right) - D'\right) r_i' - A(L_i + \delta_i^L) = 0$$

$$(r_i')^2 + \frac{1}{L_i + \delta_i^L}\left(\sum_{k\in\mathcal{T}\setminus\{i\}} L_k\left(r_k - \frac{A}{r_k}\right) - D'\right) r_i' - A = 0,$$

so

$$r_i' = \frac{-b' + \sqrt{(b')^2 + 4A}}{2},$$

where

$$\begin{aligned} b' &= \frac{1}{L_i + \delta_i^L}\left(\sum_{k\in\mathcal{T}\setminus\{i\}} L_k\left(r_k - \frac{A}{r_k}\right) - D'\right) \\ &= \frac{1}{L_i + \delta_i^L}\left(\sum_{k\in\mathcal{T}} L_k\left(r_k - \frac{A}{r_k}\right) - L_i\left(r_i - \frac{A}{r_i}\right) - D'\right) \\ &= \frac{1}{L_i + \delta_i^L}\left(D - L_i\left(r_i - \frac{A}{r_i}\right) - D'\right). \end{aligned} \tag{9}$$

Finally, the corresponding change δ_i^A in the asset of token i is given by

$$\begin{aligned} \delta_i^A &= (L_i + \delta_i^L)r_i' - L_i r_i \\ &= \frac{D' + L_i\left(r_i - \frac{A}{r_i}\right) - D + \sqrt{\left(D' + L_i\left(r_i - \frac{A}{r_i}\right) - D\right)^2 + 4A(L_i + \Delta_i)^2}}{2} - L_i r_i. \end{aligned} \tag{10}$$

4.2 Maintain Global Equilibrium with $r^* = 1$

The essence of our modified CFMM is that the system is the most stable when all coverage ratios are 1. Swap, withdrawal, and deposit will change the coverage

ratios of each individual pool, but our design guarantees that we can always maintain the global equilibrium coverage ratio as $r^* = 1$, which is explained as follows.

As seen in Section 2.3, the liabilities L_k and the constant D stay unchanged in our swap mechanism, so the global equilibrium coverage ratio r^* in

$$\left(\sum_{k\in\mathcal{T}} L_k\right)\left(r^* - \frac{A}{r^*}\right) = D$$

is always preserved. As for withdrawal or deposits, if the initial global equilibrium coverage ratio is $r^* = 1$, then the new global equilibrium coverage ratio $r^{*\prime}$, defined in equation (6), is also 1.

Maintaining the global equilibrium coverage ratio to be 1 has another significant implication, given by the following theorem.

Theorem 2. *Assume that $r^* = 1$. If $\delta_i^L < 0$, then $\delta_i^L \le \delta_i^A < 0$; if $\delta_i^L > 0$, then $0 < \delta_i^A \le \delta_i^L$. Furthermore, in both cases, $\delta_i^L = \delta_i^A$ if and only if $r_i = 1$.*

Proof. If $r^* = 1$, then $D = \left(\sum_{k\in\mathcal{T}} L_k\right)(1-A)$ by equation (5), $r^{*\prime} = 1$ by equation (6), and $D' = \left(\delta_i^L + \sum_{k\in\mathcal{T}} L_k\right)(1-A)$ by equation (7). Hence, equation (8) becomes

$$\begin{aligned}
(L_i+\delta_i^L)\left(\frac{A_i+\delta_i^A}{L_i+\delta_i^L} - \frac{A(L_i+\delta_i^L)}{A_i+\delta_i^A}\right) + \sum_{k\in\mathcal{T}\setminus\{i\}} L_k\left(r_k - \frac{A}{r_k}\right) &= \left(\delta_i^L + \sum_{k\in\mathcal{T}} L_k\right)(1-A) \\
A_i+\delta_i^A - \frac{A(L_i+\delta_i^L)^2}{A_i+\delta_i^A} + D - L_i\left(r_i - \frac{A}{r_i}\right) &= \delta_i^L(1-A) + D \\
\delta_i^A - \frac{A(L_i+\delta_i^L)^2}{A_i+\delta_i^A} + \frac{AL_i^2}{A_i} &= \delta_i^L(1-A). \qquad (11)
\end{aligned}$$

If $\delta_i^L < 0$ and $\delta_i^A \ge 0$, then the left hand side (LHS) of equation (11) is positive while the right hand side (RHS) is negative, a contradiction. Similarly, if $\delta_i^L > 0$ and $\delta_i^A \le 0$, then the LHS is negative while the RHS is positive, a contradiction again. Hence, δ_i^L and δ_i^A always share the same sign.

To compare δ_i^L and δ_i^A, we first rewrite equation (9) as

$$
\begin{aligned}
b' &= \frac{1}{L_i + \delta_i^L}\left(\left(\sum_{k\in\mathcal{T}} L_k\right)(1-A) - L_i\left(r_i - \frac{A}{r_i}\right) - \left(\delta_i^L + \sum_{k\in\mathcal{T}} L_k\right)(1-A)\right) \\
&= -\frac{1}{L_i + \delta_i^L}\left(L_i\left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1-A)\right).
\end{aligned}
$$

Therefore, equation (10) yields

$$
\begin{aligned}
&\delta_i^A - \delta_i^L \\
&= \frac{-(L_i+\delta_i^L)b' + \sqrt{\left((L_i+\delta_i^L)b'\right)^2 + 4A(L_i+\delta_i^L)^2}}{2} - L_i r_i - \delta_i^L \\
&= \frac{-L_i\left(r_i + \frac{A}{r_i}\right) - \delta_i^L(1+A) + \sqrt{\left(L_i\left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1-A)\right)^2 + 4A(L_i+\delta_i^L)^2}}{2}.
\end{aligned}
$$

Note that

$$
\left(r_i - \frac{A}{r_i}\right)^2 + 4A = r_i^2 - 2A + \frac{A^2}{r_i^2} + 4A = \left(r_i + \frac{A}{r_i}\right)^2
$$

and

$$
(1-A)^2 + 4A = 1 - 2A + A^2 + 4A = (1+A)^2.
$$

Furthermore, by the AM-GM inequality, we have $r_i + \dfrac{1}{r_i} \geq 2$, so

$$
\begin{aligned}
\left(r_i - \frac{A}{r_i}\right)(1-A) + 4A &= r_i + \frac{A^2}{r_i} - A\left(r_i + \frac{1}{r_i}\right) + 4A \\
&\leq r_i + \frac{A^2}{r_i} - A\left(r_i + \frac{1}{r_i}\right) + 2A\left(r_i + \frac{1}{r_i}\right) \\
&= \left(r_i + \frac{A}{r_i}\right)(1+A),
\end{aligned}
$$

where the equality holds if and only if $r_i = 1$. As a result,

$$\begin{aligned}
&\left(L_i\left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1-A)\right)^2 + 4A(L_i + \delta_i^L)^2 \\
&= L_i^2\left(r_i - \frac{A}{r_i}\right)^2 + 4AL_i^2 + 2L_i\left(r_i - \frac{A}{r_i}\right)\delta_i^L(1-A) + 8AL_i\delta_i^L \\
&\quad + (\delta_i^L)^2(1-A)^2 + 4A(\delta_i^L)^2 \\
&= L_i^2\left(r_i + \frac{A}{r_i}\right)^2 + 2L_i\delta_i^L\left(\left(r_i - \frac{A}{r_i}\right)(1-A) + 4A\right) + (\delta_i^L)^2(1+A)^2.
\end{aligned}$$

If $\delta_i^L < 0$, then

$$\begin{aligned}
&\sqrt{\left(L_i\left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1-A)\right)^2 + 4A(L_i + \delta_i^L)^2} \\
&\geq \sqrt{L_i^2\left(r_i + \frac{A}{r_i}\right)^2 + 2L_i\delta_i^L\left(r_i + \frac{A}{r_i}\right)(1+A) + (\delta_i^L)^2(1+A)^2} \\
&= \left|L_i\left(r_i + \frac{A}{r_i}\right) + \delta_i^L(1+A)\right|,
\end{aligned}$$

so $\delta_i^A - \delta_i^L \geq 0$, with the equality holds if and only if $r_i = 1$; if $\delta_i^L > 0$, then

$$\begin{aligned}
&\sqrt{\left(L_i\left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1-A)\right)^2 + 4A(L_i + \delta_i^L)^2} \\
&\leq \sqrt{L_i^2\left(r_i + \frac{A}{r_i}\right)^2 + 2L_i\delta_i^L\left(r_i + \frac{A}{r_i}\right)(1+A) + (\delta_i^L)^2(1+A)^2} \\
&= \left|L_i\left(r_i + \frac{A}{r_i}\right) + \delta_i^L(1+A)\right|,
\end{aligned}$$

so $\delta_i^A - \delta_i^L \leq 0$, again with the equality holds if and only if $r_i = 1$.

4.3 Withdrawal Fees and Deposit Gains

Now, we are ready to describe our algorithm to block the arbitrage. When a withdrawal is made, $\Delta_i < 0$ is specified by the trader. We define $\delta_i^L = \Delta_i$ as the change in the liability of token i, and δ_i^A is given by

$$\delta_i^A = \frac{L_i\left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1-A) + \sqrt{\left(L_i\left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1-A)\right)^2 + 4A(L_i+\delta_i^L)^2}}{2} - L_i r_i. \tag{12}$$

As proved in Theorem 2, $|\delta_i^A| \leq |\delta_i^L| = |\Delta_i|$, so from the trader's perspective, there is always a withdrawal fee unless $r_i = 1$. Furthermore, it is apparent from equation (2) that $\delta_i^A \geq -L_i r_i = -A_i$, so the final amount of token i that the trader receives is bounded above by the amount of token i available in the system.

On the other hand, when a deposit is made, $\Delta_i > 0$ is specified by the trader. We define $\delta_i^A = \Delta_i$ as the change in the asset of token i, and we solve for δ_i^L in equation (12). Rearranging the terms and letting $A_i' = \delta_i^A + L_i r_i$ be the asset of token i after the deposit, we have

$$\begin{aligned} &2A_i' - \left(L_i\left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1-A)\right) \\ &= \sqrt{\left(L_i\left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1-A)\right)^2 + 4A(L_i+\delta_i^L)^2}. \end{aligned}$$

If we square the equation, cancel the identical terms on both sides, and divide the equation by 4, we get

$$(A_i')^2 - A_i'\left(L_i\left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1-A)\right) = A(L_i+\delta_i^L)^2.$$

This is a quadratic equation in δ_i^L, namely

$$A(\delta_i^L)^2 + b\delta_i^L + c = 0,$$

where $b = A_i'(1-A) + 2AL_i$ and $c = A_i' L_i\left(r_i - \frac{A}{r_i}\right) - (A_i')^2 + AL_i^2$. The discriminant of the quadratic equation is given by

$$\begin{aligned} b^2 - 4Ac &= (A_i')^2(1-A)^2 + 4A_i'(1-A)AL_i + 4A^2L_i^2 \\ &\quad - 4AA_i'L_i\left(r_i - \frac{A}{r_i}\right) + 4A(A_i')^2 - 4A^2L_i^2 \\ &= (A_i')^2(1+A)^2 + 4AA_i'L_i\left(1 - A - r_i + \frac{A}{r_i}\right), \end{aligned}$$

thus the change in the liability δ_i^L is

$$\delta_i^L = \frac{-(A_i'(1-A)+2AL_i) + \sqrt{(A_i')^2(1+A)^2 + 4AA_i'L_i\left(1-A-r_i+\frac{A}{r_i}\right)}}{2A}. \quad (13)$$

The negative branch is rejected since it is less than $-L_i r_i$. As proved in Theorem 2, $\Delta_i = \delta_i^A \leq \delta_i^L$, so from the trader's perspective, there is always a deposit gain unless $r_i = 1$.

5 Swap with Haircut Fees

As a profit for the protocol, we charge a certain percentage for each swap as a *haircut fee*, denoted as h. Part of this fee will be shared with the liquidity provider and will enter the token pool as a deposit; the rest is going to be retained by Wombat. Let ρ denote the *LP dividend ratio*, which is the ratio of the haircut fee that is shared with the liquidity provider. The portion retained by Wombat will not be included as part of the token assets for accounting purpose to maintain $r^* = 1$. In the following, we will describe how swap is performed with haircut fees.

During the first stage, token i is swapped to token j as described in Section 2.3. Here, we provide an explicit expression for Δ_j.

$$\begin{aligned}\Delta_j &= L_j r_j - L_j r_{j,0} \\ &= \frac{D_{i,j} - L_i\left(r_i - \frac{A}{r_i}\right) + \sqrt{\left(D_{i,j} - L_i\left(r_i - \frac{A}{r_i}\right)\right)^2 + 4AL_j^2}}{2} - L_j r_{j,0},\end{aligned}$$

where r_i and $D_{i,j}$ are given by equations (2) and (3), respectively. After the swap, the trader receives $(1-h)|\Delta_j|$ token j, with $h|\Delta_j|$ token j deducted as the haircut fee. Before the haircut fee is redeposited into the system, the amount of assets in token i and token j are respectively

$$L_i r_{i,0} + \Delta_i \quad \text{and} \quad L_j r_{j,0} + \Delta_j,$$

while the liabilities of token i and token j maintain at L_i and L_j, respectively.

Next, $\Delta_j' = h\rho|\Delta_j|$ token j is deposited into the system since it is shared with the liquidity provider. Using the token j version of equation (13) and setting $\delta_j^A = \Delta_j'$, we can compute the corresponding δ_j^L. Therefore, the summary of the final amounts is given by the following table.

Final asset of token i	$L_i r_{i,0} + \Delta_i$
Final asset of token j	$L_j r_{j,0} + (1 - h\rho)\Delta_j$
Final liability of token i	L_i
Final liability of token j	$L_j + \delta_j^L$

Table 5: Final asset and liability of tokens i and j after a swap with haircut fees

6 Exact Swap, Withdraw, and Deposit when $r^* = 1$

The previous sections show that swap, withdraw, and deposit almost always come with fees or gains. Sometimes, it is desirable to deduce the appropriate amount a trader needs to initially swap, withdraw, or deposit to attain the exact target amount.

In a swap, if the trader specifies the exact amount $|d_j|$ of token j that they would like to receive with $d_j < 0$, then $\Delta_j = \dfrac{d_j}{1-h}$, and r_j is defined as in equation (2). After solving for r_i in equation (3), we can deduce that

$$\begin{aligned}\Delta_i &= L_i r_i - L_i r_{i,0} \\ &= \frac{D_{i,j} - L_j\left(r_j - \frac{A}{r_j}\right) + \sqrt{\left(D_{i,j} - L_j\left(r_j - \frac{A}{r_j}\right)\right)^2 + 4AL_i^2}}{2} - L_i r_{i,0},\end{aligned}$$

which is the amount of token i that the trader should swap.

In withdrawal or deposits, as shown in equations (12) and (13), δ_i^A and δ_i^L can be expressed as a function of each other. Hence, if the trader wants to receive exactly $|d_i|$ token i in a withdrawal with $d_i < 0$, then letting $\delta_i^A = d_i$ and solving for δ_i^L using equation (13), we obtain $\Delta_i = \delta_i^L$ as the initial parameter for the withdrawal. Similarly, if the trader wants to have exactly d_i token i stored in the system with $d_i > 0$, then letting $\delta_i^L = d_i$ and solving for δ_i^A using equation (12), we obtain $\Delta_i = \delta_i^A$ as the initial parameter for the deposit.

7 Conclusions

We have introduced an innovative algorithm that resolves many issues of the current AMM environment. The existing solutions in the market are computationally inefficient, and these protocols lack the ability to scale while keeping an

intuitive design both technically and visually. Additional protocols have been built on these platforms to address some of these problems, but they inevitably add an extra layer of complexity for end-users. In the world of blockchain smart-contract development, unnecessary complexity is a burden to the end-users due to high gas fees and disincentivizes users from interacting with the protocol.

The design of the Wombat algorithm allows for maximum capital efficiency and scalability, which helps promote the growth of decentralized finance. Our work is centered on a CSMM that shows positive homogeneity since the response of relative price is identical at various levels of liquidity. Wombat's algorithm provides a solution to the status quo, which results in a price-sensitive and path-independent solution while being computationally efficient. We have proved that our arbitrage block can withstand manipulation of the coverage ratio. The arbitrage block can protect the system from malicious attacks, shielding Wombat from attack vectors that would hurt the system.

References

1. Angeris, G., & Chitra, T. (2020). Improved Price Oracles: Constant Function Market Makers. SSRN Electronic Journal.
2. Hayden Adams. (2019). Uniswap birthday blog - v0.
3. Michael Egorov. (2019). stableswap - efficient mechanism for Stablecoin liquidity.
4. Othman, A., Sandholm, T., Pennock, D. M., & Reeves, D. M. (2013). A practical liquidity-sensitive automated market maker. ACM Transactions on Economics and Computation (TEAC), 1(3):1-25.
5. Platypus Team. (2021). Platypus AMM Technical Specification.