On-chain optimal aggregation of Uniswap v3 clones

Vincent Danos
CNRS, ENS

Hamza El Khalloufi
University Paris 1 Panthéon-Sorbonne & Mangrove DAO

Leo Murao Watson
University of Toronto

Santiago Valencia
Giry SAS & Mangrove DAO

December 16, 2024

Abstract

In this paper we define a simple and efficient "push-and-solve" algorithm to compute the best execution or "optimal split" of a market order given a finite set of AMMs. Each AMM has to be decomposable into basic building blocks which we call price-parametrised AMMs. This has a practical application to the optimal splitting of orders among Uniswap v3 clones, as the algorithm is query-optimal (information on sources is queried on a call-by-need basis) and therefore of low enough complexity to be implemented as a smart contract. We also find a sufficient condition for AMMs based on price parametrisations to be aggregatable which is of independent interest as it allows one to build novel AMMs with concentrated liquidity, families of which can also be optimally executed on-chain.

1 Introduction

Automated market makers (AMMs) occupy a central place in decentralised finance (DeFi) as the primary means for on-chain trading. An AMM is a two-sided platform where traders (aka takers) consume the liquidity provided by liquidity providers (LPs) and which operates entirely on-chain. There exists a wide variety of types of AMMs among which Uniswap v2 and v3 [3, 4] stand out for their simplicity and their market share. Uniswap v3 improves on v2 by letting LPs decide in which price range their liquidity is deployed.

The low costs of reproducing DeFi protocols has led to a proliferation of clones and variants of Uniswap v2 and v3, all competing for the same liquidity. This has resulted in substantial fragmentation of capital across multiple similar AMMs. In turn, this capital fragmentation has prompted the emergence of aggregators who aim to consolidate the multiple offerings that takers face, relieve them of time-consuming searches, and optimise the execution of their market orders.

To name a few: Paraswap, 1inch, Odos.

Some level of re-intermediation seems a necessary evil. However, because of the latency of an aggregator's execution planning algorithm and that of the blockchain on which the plan is to be executed, there must be a discrepancy between the plan and its actual on-chain execution. What one might call a *temporal slippage* (not to be confused with slippage understood as market impact).[1] Ideally, aggregation should be executed *and* planned on-chain. There would be no latency, no induced slippage, so no loss of value to the intermediary (and as a bonus, a complete transparency).

[1]Typically, aggregators unload negative slippage on the taker, and keep a share of the positive one.

This paper offers progress in this direction, with an algorithm for *optimal splitting* which is simple enough for an on-chain implementation, and can be effectively used to reduce temporal slippage.

The optimal splitting problem is as follows:

> *We are given a finite family $\mathcal{F}$ of AMMs defined on the same pair X/Y, and a market order (say taker wants to buy X with an amount Δy of Y), and the problem is to split Δy among the AMMs in $\mathcal{F}$ so as to maximise the total amount of X obtained.*

Our algorithm assumes that the AMMs in $\mathcal{F}$ use *price-parametrised AMMs* (PPMs) as building blocks. (Think of a PPM as a single Uniswap v3 LP position.) Every AMM in $\mathcal{F}$ is represented as a concatenation of such blocks defined on disjoint price intervals.

As we show below, this is a good model of the structure of an AMM of the Uniswap v3 type. Indeed, Uniswap v3 itself can be seen as an on-chain protocol to aggregate optimally and efficiently its LP positions. It combines cleverly LP positions and keeps a low execution complexity (which is essentially independent of the number of LP positions).

It follows that our algorithm is suitable for the on-chain aggregation of families of AMMs of the Uniswap v3 (and v2) type (as the title of the paper indicates). We describe other types of AMMs with aggregatable LP positions (hitherto undescribed), to which it would apply as well, but the Uniswap case is by far the most important case.

Efficiency questions Not only is the order splitting algorithm optimal, but it is also *query-optimal* in the sense that it only queries the various components of members of $\mathcal{F}$ by need. This is key to obtaining an on-chain implementation with reasonable gas costs. (Taker, by going on-chain, no longer pays intermediation costs, but instead pays gas ie the the blockchain's infrastructural costs.)

Gas questions aside, is using an on-chain optimal splitter worth the bother? There is always the decentralisation advantage, and the fact that it is all clear what the optimal splitter contract does. But what about efficiency? We show (§5) that the gain, measured in price impact reduction when going from a mono-source swap to a multi-source one can be substantial if liquidity is evenly distributed among the sources.

One should also compare our on-chain optimal splitter with off-chain aggregators. It may be that in some conditions our on-chain aggregator, despite being limited to the very special case of optimal splits, can perform better than a general aggregator which can build complex multi-hop execution plans (Fig. 1) and has vastly more compute power. But also, the on-chain splitter and off-chain aggregator can work together. Indeed, in whichever market conditions, when the execution plan computed by the off-chain aggregator contains splits between Uniswap v3 AMM types as it often does (see Fig. 1 for examples), using the on-chain splitter at execution time will unconditionally improve the execution of the plan. Hence an optimal splitter should be part of the execution engine of any off-chain execution planner, at least on chains where the gas costs are negligible compared to expected gains (such as Arbitrum).

Limitations This paper contains no empirical analysis of how well the optimality advantage translates in concrete cases, nor does it study the resilience of the on-chain optimal splitter to price-bending attacks on its sources, such as sandwich attacks.

The algorithm itself also has limitations. If one is serious about applications, popular types of AMMs such as the Curve ones [19, 20] should be integrated.

Finally, in contexts where gas costs are of the same order of magnitude as potential improvements (small orders with a large number of sources with un-even liquidity distributions), the gains of optimality may not be worth the additional gas costs.

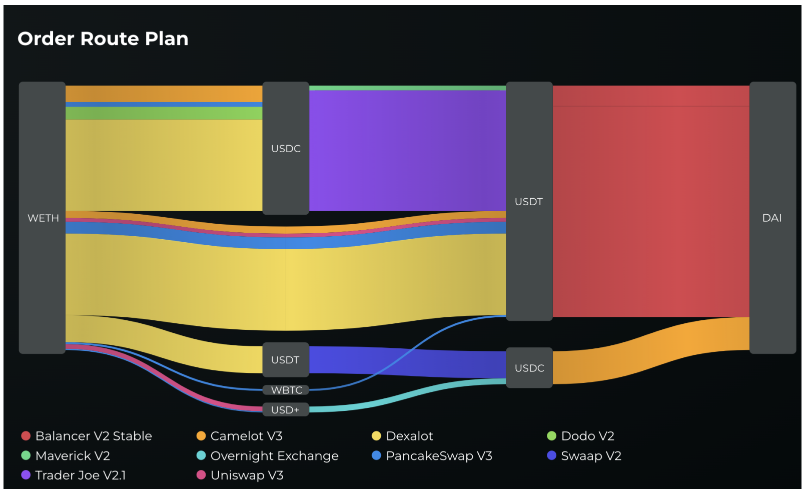

Figure 1: *An execution plan computed by the Odos routing algorithm for an ETH-to-USDC swap of* 10 *ETH on Arbitrum (October 2024). One sees that the Odos aggregator computes a plan which includes several splits between Uniswap v3 clones.*

1.1 Related work

Improving execution performance is, of course, a concern in traditional finance. Ref. [5] emphasizes the necessity of order splitting to mitigate the impact of large orders. Refs. [23, 10] extend this work to multi-venue routing, focusing on optimising execution cost across varied liquidity profiles.

The structure of AMMs inherently adjust asset prices based on trade sizes. In that respect it is analogous to some traditional models of price impact, such as those incorporating marginal supply-demand curves [11, 22, 1]. Because of this response of price to order size, optimised executions need to split orders across multiple AMMs. Ref. [7] analyses specific class of constant function AMMs from this point of view. Further, Ref. [14] frames optimal routing and arbitrage as global convex optimisation problems on arbitrary networks of AMMs. Later, Ref. [8] formulates optimal routing for trades across multiple AMMs, incorporating multiple tokens as inputs within a single optimisation framework.

More recently, Ref. [18] presents an efficient algorithm for optimal routing through constant function market makers using a dual decomposition approach. This comes close to the algorithm which we propose in this paper. However, our algorithm introduces two important innovations. First, we generalise beyond CFMMs and handle a broader class of AMMs defined piece-wise by price parametrisations. This allows us to address more complex and more realistic exchange mechanisms of the Uniswap v3 type (which we generalise introducing the notion of convenient price parametrisations). Second, our "push-and-solve" method only queries the parameters of its sources when it needs them which is crucial for an on-chain implementation.

1.2 Outline

The first part of the paper is dedicated to the notion of price-parameterised AMM (PPM) and various attached constructions. Notably we define the notion of convenient PPMs which will be the basis for the model of AMMs which our algorithm targets.

Once this is in place, we turn to a general formulation of the optimal splitting of an order among a family of source AMMs as a best execution problem (Def. 5). When this problem is convex (which we assume) it falls in the category of so-called *resource allocation problems* for which well-known algorithms exist. We then exploit the specific fact that PPMs are parameterised by price to build an equivalent formulation (Def. 7). This second form suggests a simple "push-and-solve" algorithm similar to an ascending auction between the various sources which we present in §4. Finally, §5 discusses the significance of the gains of using an optimal splitter.

2 Price parametrised AMMs

We start with the definition of our main building block the notion of *price-parametrised AMM*, PPM for short.

We write $\mathbb{R}_+$ for the non-negative reals, and $C^1([a,b])$ for the set of functions f that are continuously differentiable on (a,b) and such that f and f' are continuous on $[a,b]$.

Definition 1 *A price-parametrised AMM (PPM for short) is a pair of real-valued functions (x,y) defined on a closed interval $[a,b]$ with $0 < a < b$ positive real numbers, such that:*

1. *x, y are $C^1([a,b])$*
2. *x is decreasing*
3. *y is increasing*
4. *for $p \in [a,b]$:*

$$y'(p) = -px'(p) \tag{1}$$

The above implies that $x > 0$ on $[a,b)$ and $y > 0$ on $(a,b]$.

The functions x and y are called the components of the PPM.

Once says a PPM is *normalised* if $y(a) = x(b) = 0$.

See Fig. 2 for a graphical illustration.

The intuition (made precise in §2.2), is that a PPM describes a continuous automated market maker (AMM) that is to say a deterministic trading rule defined on a pair of assets X/Y where X is the base asset, and Y is the quote asset. The domain of x and y, namely $[a,b]$, is the price range within which the AMM is ready to trade. The component functions x and y determine the size of the AMM's reserves in X and Y as (bijective) functions of the AMM's current (marginal) price p.

The monotonicity conditions amounts to saying that the AMM sells (buys) when the price goes up (down), as expected from a market-making algorithm. Specifically, in a trade, the amounts Δx and Δy by which the reserves in X and Y change are such that $\Delta x \leq 0$ iff $\Delta y \geq 0$ iff $\Delta p \geq 0$.

The differential condition (1) expresses the fact that for an infinitesimal trade where Δy, Δx are very small, the execution price $\Delta y/\Delta x$ is p (up to sign), which is exactly saying that p is the marginal price of the asset X counted in units of Y (and therefore of dimension Y/X).

Condition (1) distinguishes (uniquely) the price parametrisation among all possible ones with the same image in $\mathbb{R}^2_+$.

As x is continuous decreasing, and y is continuous increasing it must be that x, y are bijections with inverse functions (which retrieve the current price from the reserves in X or Y):

$$x^{-1} : [x(b), x(a)] \to [a,b]$$
$$y^{-1} : [y(a), y(b)] \to [a,b]$$

Therefore the price is sufficient to describe the state of a PPM.[2]

Any PPM (x_f, y_f) has a natural implicit equation for its image (as a subset of $\mathbb{R}^2_+$) namely: $\Psi(x, y) = y_f^{-1}(y) - x_f^{-1}(x)$ which is increasing in both arguments. The AMM's trading rule will be constrained to stay on that invariant. This is a common approach to defining and reasoning about AMMs invariants [3, 4, 24, 16].

The notion of PPM is asymmetric in that the price is defined as $-y'/x'$ (and not the inverse). This means that we think of X as the base asset which one buys (sells), and of Y as the quote asset which one spends (receives as payment).

There is a symmetry operating on PPMs. Namely, one can precompose f by $g(x) = 1/x$ to generate another dual PP. Specifically, if (x, y) is a PPM on $[a, b]$, it is easy to see that so is the pair $(y \circ g, x \circ g)$ defined on $[1/b, 1/a]$. This dual PPM describes the same AMM under the dual convention that Y is base, X is quote, and accordingly the price is now $1/p$ of dimension X/Y.

2.1 Examples

Let's consider two simple examples of normalised PPMs to illustrate the above definition.

2.1.1 The linear case

Define the X-component of $f_L(a, b, C)$ as a linear schedule for selling X:

$$x_L(p) \quad = \quad C \cdot (b - p)$$

Parameters a, b, C can be interpreted as follows:
- a is the minimum price at which one is willing to sell
- $b > a$ is the price at which one wishes to have sold the entire initial inventory $x_L(a) = C(b - a)$
- C has dimension X^2Y^{-1} and is the amount of inventory sold per unit of price

The expression for $y_L(p)$ can be derive uniquely from (1) seen as an ODE defining y, namely: $y'(p) = pC$. Equation (1) is then true by construction. We get:

$$y_L(p) \quad = \quad C \cdot (p^2 - a^2)/2$$

where the integration constant is chosen so that the normalisation condition $y(a) = 0$ holds.

Fig. 2 plots both components of $f_L(a, b, x_0)$ with $a = 50$, $b = 150$.

This PPM is normalised. As $x_L(a) = C(b - a)$, we have $y_L(b)/x_L(a) = (a + b)/2$. This means that the execution price for buying the entire AMM's inventory is $(a + b)/2$.

A representation equivalent to $x(p)$ is that of a supply curve $S_x(p) = x(a) - x(p)$ with the supply curve defined on the same domain as $x(p)$ and representing the point of view of a buyer.

There are closed forms for the inverses of the component functions:

$$\begin{aligned} x_L^{-1}(x) &= b - x/C \\ y_L^{-1}(y) &= \sqrt{2y/C + a^2} \end{aligned}$$

It follows than an associated invariant can be defined as:

$$\psi_L(x, y) \quad = \quad x/C + \sqrt{2y/C + a^2} - b$$

Note that f_L is a natural way for a liquidity provider to express its buying/selling intentions. Empirical studies of the shape of order books in traditional finance have shown that linear supply maps are a powerful pattern to study the statistical properties of order books [9], as well as for the formulation of efficient market-making strategies [12].

[2]The PPM formalism is not suitable for AMMs which are piecewise price constant.

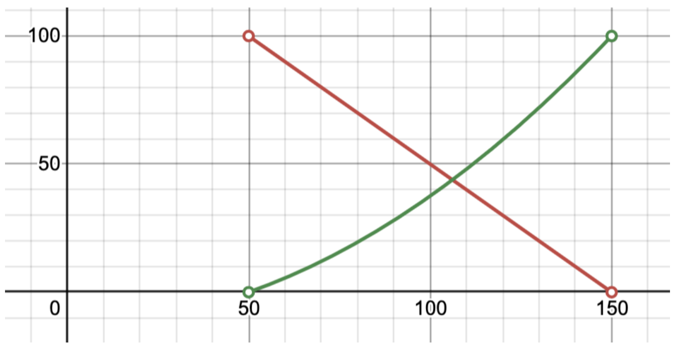

Figure 2: *Plot of $f_L(50, 150, 100)$ with minimum price $a = 50$, maximum price, $b = 150$, and starting inventory $x_0 = 100$; $x_L(p)$ (in red) decreases linearly from $x_0 = 100$ to 0, while $y_L(p)$ (in green) increases from 0 to $x_0(a + b)/2$. (NB: $y(p)$ is rescaled by $1/100$ to get both plots on the same scale.)*

2.1.2 The Uniswap v3 case

The Uniswap v3 [4] PPM $f_U(a, b, L)$ is:

$$\begin{aligned} x_U(p) &= L(1/\sqrt{p} - 1/\sqrt{b}) \\ y_U(p) &= L(\sqrt{p} - \sqrt{a}) \end{aligned}$$

Note that $y_U(a) = x_U(b) = 0$ so this PPM is normalised.

It is also a bona fide PPM since $x'_U(p) = -1/2Lp^{-3/2}$, $y'_U(p) = 1/2Lp^{-1/2}$, hence (1) is satisfied.

Fig. 3 plots both components of $f_U(a, b, L)$ with $a = 50$, $b = 150$ (same range as Fig. 2).

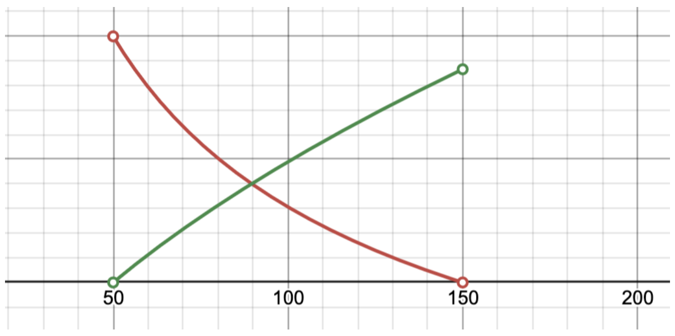

Figure 3: *Plot of the Uniswap v3 price parametrisation $f_U(50, 150, L)$ with minimum price $a = 50$, maximum price, $b = 150$, and $L \sim 1673$ chosen so that $x_U(a) = 100$ as in Fig. 2; $x_U(p)$ (in red) decreases non-linearly from $x_U(a) = 100$ to 0, while $y_U(p)$ (in green) increases from 0 to ~ 86. As in Fig. 2, $y(p)$ is rescaled by $1/100$ to get both plots on the same scale.*

One can compare visually the f_U and f_L parametrisations. To do so, in Fig. 2, and Fig. 3, we pick the same range $[a, b]$ and set the C and L constants to align the max inventories, ie $C(b - a) = x_L(a) = x_U(a) = L(1/\sqrt{a} - 1/\sqrt{b})$. One can also compare the respective execution price for consuming the entire inventory: as said $y_L(b)/x_L(a) = (a + b)/2$ the arithmetic mean, while $y_U(b)/x_U(a) = \sqrt{ab}$ the smaller geometric one. This reflects the fact that the selling schedule is more aggressive (ie $x_U \leq x_L$).

Our two examples offer different selling schedules: f_L is a linear selling schedule, while f_U is another novel way to shape one's intentions to buy and sell. Note that $\psi_U(x,y) := (x+L/\sqrt{b})(y+L\sqrt{a}) = L^2$ holds. This is the usual constant product invariant of a Uniswap v3 LP position [4].

We can specialise f_U by letting a, b go to 0 and $+\infty$ respectively and obtain a PPM corresponding to the plain constant product invariant $xy = L^2$ from Uniswap v2 [3].

We can also generalise f_U to a "concentrated" form of the Balancer PPM [24], $f_B(a,b,L,c_X,c_Y)$, with component maps:

$$\begin{array}{rcl} x_B(p) & = & L(c_Y/c_X)^{-c_Y}(p^{-c_Y} - b^{-c_Y}) \\ y_B(p) & = & L(c_Y/c_X)^{c_X}(p^{c_X} - a^{c_X}) \end{array}$$

where the weights c_X, c_Y and positive and such that $c_X + c_Y = 1$. As in the non-concentrated case, we find f_U as the special case where $c_X = c_Y = 1/2$.[3]

2.2 PPMs as AMMs

We build now a precise definition of the AMM, or the trading rule, associated to a PPM.

Recall that x measure the current amount of base X in the AMM's reserves, while y measures the amount of quote Y.

Given a PPM $f = (x_f, y_f)$ defined on $[a,b]$, we define a trading function $\delta_f : [a,b] \times \mathbb{R} \to [a,b]$ which takes as input 1) the current price p, and 2) either a Δx or a Δy. The inputs Δx or a Δy can be positive (received by the AMM) or negative (paid out by the AMM). In all cases δ_f returns the new current price q, which determines uniquely the new reserves.

An input $\Delta x \geq 0$ corresponds to a *sell* market order where: (i) taker gives Δx and (ii) $q = \delta_f(p, \Delta x) \leq p$ decreases. Likewise, an input $\Delta y \geq 0$ corresponds to a *buy* market order buy where: (i) taker gives Δy and (ii) $q = \delta_f(p, \Delta y) \geq p$ increases.

Definition 2 (buy order) *Consider first a market order with input Δy. There are three cases depending on the position of $\Delta y + y(p)$ relative to f's domain $[a,b]$:*

$$\delta_f(p,\Delta y) \quad := \quad \begin{cases} a & \textit{if } y_f(p) + \Delta y < y_f(a) & \textit{underflow} \\ y_f^{-1}(y_f(p) + \Delta y) & \textit{if } y_f(a) \leq y_f(p) + \Delta y \leq y_f(b) & \textit{in range} \\ b & \textit{if } y_f(b) < y_f(p) + \Delta y & \textit{overflow} \end{cases}$$

Let us write $q = \delta_f(p, \Delta y)$ for the post-transition price.

Formally, the definition applies whether the input Δy is positive or not.

In general, a positive input is similar to a forward request for quote: "If I pay you Δx (Δy), how much Y (X) do I receive from you?", while a negative input is like a backward request for quote: "If I want to receive Δx (Δy) from you, how much Y (X) do I have to pay you?".

In all cases the next price q is higher if $\Delta y \geq 0$ (forward buy order) lower if $\Delta y \leq 0$ (backward sell order), because y_f and therefore y_f^{-1} is increasing.

In the middle clause, we require that $y_f(p) + \Delta y$ falls within the image of $[a,b]$ under y_f, hence $y_f(q) = y_f(p) + \Delta y$ for some q, and this q is given by $y_f^{-1}(y_f(p) + \Delta y)$. This also means that the input Δy is received fully (if positive) or paid out fully (if negative) by the AMM.

In the underflow clause, it must be that $\Delta y \leq 0$. In addition $q = a$, hence the the effective amount $(\Delta y)_e = y(a) - y(p) \leq 0$ of Y paid out by the AMM is smaller in absolute value than the original Δy. The AMM gave all its Ys down to the minimum allowed $y(a)$ (which is 0 if f is normalised).

[3]Balancer's AMMs allow for more than two assets, which brings the interesting question of whether price parametrisations can be generalised. One would need a price vector of size $n-1$ to parametrise an n-asset market, and a PPM would therefore become a particular type of parametrised hypersurface.

Dually, in the overflow clause, it must be that $\Delta y \leq 0$. In addition $q = b$, and the effective amount of Y $(\Delta y)_e = y(b) - y(p) =:\geq 0$ received by the PPM is smaller than Δy. The gave all its Xs, down to the minimum allowed $x(b)$ (again 0 if f is normalised).

The case of a market order with (possibly negative) input Δx is entirely similar.

Definition 3 (sell order)

$$\delta_f(p, \Delta x) \quad := \quad \begin{cases} b & \textit{if } x_f(p) + \Delta x < x_f(b) & \textit{underflow} \\ x_f^{-1}(x_f(p) + \Delta x) & \textit{if } x_f(b) \leq x_f(p) + \Delta x \leq x_f(a) & \textit{in range} \\ a & \textit{if } x_f(a) < x_f(p) + \Delta x & \textit{overflow} \end{cases}$$

When the current price is at one end of the range, ie when $p = a$ or $p = b$, the AMM trades only in one direction. If $p = b$ ($p = a$) taker can only sell (buy): $\Delta y \leq 0$ (≥ 0), $\Delta x \geq 0$ (≤ 0).

2.2.1 Remarks

Suppose PPMs f and g differ only by a constant $C = (C_x, C_y)$, ie $g(p) = f(p) + C$. By the definition above, $\delta_g = \delta_f$. This is obvious in the under- and overflow clauses, and in the 'in range' one, one has $y_g^{-1}(y_f(p) + C_y + \Delta y) = y_f^{-1}(y_f(p) + \Delta y)$ since by definition $y_g^{-1}(Y + C_y) = y_f^{-1}(Y)$. Thus PPMs are only defined up to a translation. Hence, one can always normalise a PPM so that $y(a) = x(b) = 0$.

For clarity, we can make the trading rule fully explicit. Consider for instance the case $\Delta y \geq 0$, ie taker is buying some X. and its Δy is in range, the full state change is as follows:

$$\begin{array}{lll} \text{old amount of } Y\text{:} & y_f(p) & & \text{new amount of } Y\text{:} & y_f(p) + \Delta y \\ \text{old current price:} & p & \longrightarrow & \text{new current price:} & q = y_f^{-1}(y(p) + \Delta y) \\ \text{old amount of } X\text{:} & x_f(p) & & \text{new amount of } X\text{:} & (x_f \circ y_f^{-1})(y(p) + \Delta y) \end{array}$$

We see that, computationally, a determinant of the complexity of a transition is the complexity of computing x_f or y_f and their inverses.

It does not matter for the definition of δ_f that f satisfies (1). That the parameter can be interpreted as a price becomes important later (in the proof of Prop. 3).

2.2.2 Volume-volume maps are concave

We return to our examples f_U, f_L and illustrate the definition above by computing their *volume-volume* maps and verify that they are concave (which becomes important next Section). That is to say the maps that given a current price p, takes as input an amount δy or δx (given to the PPM), and returns the amount δx or δy paid by the PPM.

We only derive the Y-to-X volume-volume function, that is to say the case of a $\Delta y \geq 0$ injection, ie a buy market order. A similar calculation would give the X-to-Y one.

Consider the case of $f_U(a, b, L)$ with current price p.

Suppose we add $\delta y \geq 0$, with no overflow (ie $y_U(p) + \delta y \leq y_U(b)$).

Let $q \geq p$ be the new current price.

By definition $y_U(p) + \delta y = y_U(q)$, and $x_U(p) + \delta x = x_U(q)$. Hence:

$$\begin{array}{rcl} \delta y/L & = & \sqrt{q} - \sqrt{p} \\ \sqrt{q} & = & \delta y/L + \sqrt{p} \\ \delta x/L & = & 1/\sqrt{q} - 1/\sqrt{p} \\ & = & 1/(\delta y/L + \sqrt{p})) - 1/\sqrt{p} \\ & = & -1/\sqrt{p}\,(\delta y/L)/(\delta y/L + \sqrt{p}) \quad \leq \quad 0 \end{array}$$

So δx as a function of δy (L, p are constants here) is an expression of the form $-\delta y / b(a\delta y + b)$ with a, $b > 0$. The payout received by taker $-\delta x \geq 0$ is 0 for $\delta y = 0$, increases with δy (the more the AMM receives, the more it has to pay out), and is concave (decreasingly increasing so to speak).

Consider the case of $f_L(a, b, L)$ with current price p and an in-range addition of $\delta y > 0$:

$$\begin{array}{lcl} \delta y / C & = & 1/2(q^2 - p^2) \\ q^2 & = & 2\delta y / C + p^2 \\ \delta x / C & = & p - q \\ & = & p - \sqrt{2\delta y / C + p^2} \quad \leq \quad 0 \end{array}$$

Here also the payout received by taker $-\delta x \geq 0$ is positive, 0 for $\delta y = 0$, increasing and concave.

2.3 Splits and Sums of PPMs

We define below a split and a sum operation. The idea is that the split operation allows one to decompose PPMs horizontally, while the sum one allows one to compose them vertically.

Summation of a finite family of PPMs $\mathcal{F} = (f_i(a, b))$ assumes the f_is are defined on the same domain $[a, b]$. It is defined point-wise:

$$\begin{array}{lcl} (\sum_i x_i)(p) & = & \sum_i x_i(p) \\ (\sum_i y_i)(p) & = & \sum_i y_i(p) \end{array}$$

Clearly $\sum f_i$ is also a PPM, defined on $[a, b]$, normalised if all f_is are. More generally, the set of PPMs defined on $[a, b]$ is closed under linear combinations with positive coefficients.

What if we have a more general family $\mathcal{F} = (f_i)$ where the domains of definition of the f_is are possibly distinct? This is where the splitting operation becomes useful.

Given a PPM $f(a, b)$ defined on $[a, b]$ with components (x_f, y_f), and an interior price point $p_0 \in (a, b)$, one defines PPMs $f_1(a, p_0)$ and $f_2(p_0, b)$ with respective domains $[a, p_0]$ and $[p_0, b]$ and component maps:

$$\begin{array}{lcl} x_1(p) & = & x_f(p) - x_f(p_0) \\ y_1(p) & = & y_f(p) \end{array} \qquad \begin{array}{lcl} x_2(p) & = & x_f(p) \\ y_2(p) & = & y_f(p) - y_f(p_0) \end{array}$$

We will write $f = f_1 \oplus f_2$. It is easy to see that the splits f_1, f_2 are still PPMs, which are normalised if (x, y) is. Splitting generalises evidently to finite sets of interior price points.

Suppose given a general family $\mathcal{F} = (f_i(a_i, b_i);\ i \in I)$ where $[a_i, b_i]$ is the domain of f_i.

Consider a minimal increasing sequence $\mathbf{s} = (s_k;\ 1 \leq k \leq N)$ which contains all (and only) the a_is and b_is.

Every f_i uniquely splits on the set S_i of split points interior to its domain $[a_i, b_i]$.

This gives a decomposition:

$$f_i \quad = \quad \oplus_{a_i \leq s_k < b_i} f_{ik}(s_k, s_{k+1})$$

where f_{ik} has domain $[s_k, s_{k+1}]$.

We can now sum this doubly indexed family separately on each split interval $[s_k, s_{k+1}]$ and obtain a new family $\mathcal{G} = (g_k(s_k, s_{k+1}) := \sum_i f_{ik};\ 1 \leq k < N)$, where g_k is defined on $[s_k, s_{k+1}]$ by construction. We call this new family $\mathcal{G}$ the split-sum decomposition of $\mathcal{F}$. It is piecewise defined on each split interval, and describes the exact same offer of liquidity as $\mathcal{F}$.

2.3.1 An example

As an illustration, we can detail the case where the initial family $\mathcal{F}$ consists of $f_U(a,b,L)$, $f_U(a',b',L')$ with four distinct end points. Up to re-labeling we can assume that $a < a'$.

If $b < a'$, the domains do not overlap and $\mathcal{F}$ is already decomposed.

If $a' < b$ domains do overlap and there are two cases.

(1) Either $b < b'$, and the increasing split sequence is $s = (a, a', b, b')$. First we split each PPM along s:

$$\begin{array}{rcl} f_U(a,b,L) & = & f_U(a,a',L) \oplus f_U(a',b,L) \\ f_U(a',b',L') & = & f_U(a',b,L') \oplus f_U(b,b',L') \end{array}$$

Then we build $\mathcal{G}$ by summing vertically on each split interval:

$$\begin{array}{rclcl} g_1 & = & f_U(a,a',L) & & \\ g_2 & = & f_U(a',b,L) + f_U(a',b,L') & = & f_U(a',b,L+L') \\ g_3 & = & f_U(b,b',L') & & \end{array}$$

where we have used additivity of the f_U format, and we obtain the split-sum decomposition $f_U(a,a',L) \oplus f_U(a',b,L+L') \oplus f_U(b,b',L')$.

(2) Or $b' < b$, in which case $s = (a, a', b', b)$ and the decomposition is $f_U(a,a',L) \oplus f_U(a',b',L+L') \oplus f_U(b',b,L)$.

While the formal decomposition of $\mathcal{F}$ into $\mathcal{G}$ is always well-defined, because the f_U format is stable under splits and sums, in this special case the number of components of $\mathcal{G}$ grows as $|\mathbf{s}|$ the size of the price sequence $\mathbf{s}$, and not as $|\mathcal{F}||\mathbf{s}|$.

We now introduce formats which generalise f_U and f_L and ensure that $\mathcal{G}$ has a compact representation if $\mathcal{F}$ obeys the format.

2.3.2 Convenient PPMs

Definition 4 *A PPM (x,y) defined on $[a,b]$ is said to be convenient if it can be written:*

$$\begin{array}{rcl} x(p) & = & C \cdot (\phi(b) - \phi(p)) \\ y(p) & = & D \cdot (\psi(p) - \psi(a)) \end{array}$$

for some positive constants C, D, and some maps ϕ, ψ defined on $\mathbb{R}_+$.

From the definition one sees that $pC\phi'(p) = D\psi'(p)$, and that both ϕ and ψ are increasing, and bijections.

The 'liquidity' constants $C, D > 0$ somehow measure initial inventory or capital.

Our examples f_U, f_B and f_L are convenient PPMs: for f_L, $\phi(p) = p$, $\psi(p) = p^2/2$; for f_U, $\phi(p) = -1/\sqrt{p}$, $\psi(p) = \sqrt{p}$.

These formats are closed under summations and splits.

That is to say, given a family of convenient PPMs with *same* ϕ, ψ we have:

$$\textstyle\sum_i f_{\phi,\psi,C_i,D_i} \quad = \quad f_{\phi,\psi,\sum_i C_i,\sum_i D_i}$$

And given a convenient PPM f, we have:

$$f(a,b,C,D,\phi,\psi) \quad \simeq \quad f(a,p_0,C,D,\phi,\psi) \oplus f(p_0,b,C,D,\phi,\psi)$$

The latter can be readily verified ((ϕ,ψ) not shown for lighter notations):

$$\begin{array}{l} x^1_{a,b,C}(p) = C \cdot (\phi(b) - \phi(p)) - C(\phi(b) - \phi(p_0)) = C \cdot (\phi(p_0) - \phi(p)) = x_{a,p_0,C}(p) \\ y^1_{a,b,D}(p) = D \cdot (\psi(p) - \psi(a)) = y_{a,p_0,C}(p) \\ x^2_{a,b,C}(p) = C \cdot (\phi(b) - \phi(p)) = x_{p_0,b,C}(p) \\ y^2_{a,b,C}(p) = D \cdot (\psi(p) - \psi(a)) - D \cdot (\psi(p_0) - \psi(a)) = D \cdot (\psi(p) - \psi(p_0)) = y_{p_0,b,C}(p) \end{array}$$

We have proved that:

Proposition 1 *Suppose all the f_is in $\mathcal{F} = (f_i(a_i, b_i);\ i \in I)$ obey the same (ϕ, ψ) format, so do the g_ks in the split-sum decomposition $\mathcal{G}$.*

Split-and-sum decompositions of families of PPMs obeying a given convenient format will be our model of compact representations of AMMs in the next sections.

Such representations are suitable for a decentralised implementation (as a smart contract) because: during the execution of a market order every g_k is visited in turn until the order is filled, and because of the trivial summability of the format, each g_k can be represented as one aggregated PPM. On the other hand, when an LP mints a new position, it can be readily split-and-summed and added to the current representation $\mathcal{G}$.

In the case of Uniswap v3, ie a family following the f_U format, $\mathcal{G}$ is a pretty accurate description of the Uniswap internal data structure used to represent a collection of LP positions [4].

3 Best execution of a family of PPMs

Now we have all in place to state and solve our problem.

First, we define the optimisation programme which captures the notion of best execution of a family of PPMs. This is well defined but leaves one with no specific way to solve the said problem. Leaning on the specific properties of price parametrisations, we reformulate the problem into an equivalent problem which has a pleasingly simple solution. We conclude with a description of the query-minimal algorithm that naturally follows.

Throughout this section, $\mathcal{F}$ denotes a finite family of PPMs *with price*, that is to say $\mathcal{F} = (f_i : [a_i, b_i], p_i; 1 \leq i \leq n)$ where each f_i has current price p_i.

Definition 5 (best execution) *Fix a budget $\Delta y \geq 0$ (buy market order).*

Consider a split of Δy, that is to say a non-negative input family $\Delta y_i \geq 0$ with $\Delta y_i \leq y(b_i) - y(p_i)$ (no overflow).

Write $\Delta x_i := x_i(\delta_{f_i}(p_i, \Delta y_i)) - x_i(p_i) \leq 0$ for the output generated by f_i given Δy_i.[4]

The sequence $(\Delta y_i;\ 1 \leq i \leq n)$ is said to be a best split of Δy over $\mathcal{F}$ if it maximises the total output:

$$- \textstyle\sum_i \Delta x_i \leq \sum_i x(p_i) - x(b_i)$$

under the budget constraint $\sum_i \Delta y_i \leq \Delta y$.

Each output $-\Delta x_i$ is bounded above by the reserve available namely $x(p_i) - x(b_i)$. This justifies the inequality in the definition.

We cannot ask that the budget is entirely spent, ie $\sum_i \Delta y_i = \Delta y$, because $\mathcal{F}$ may not have enough X to consume entirely Δy. Indeed the maximum possible input is $K_Y = \sum_i (y_i(b_i) - y_i(p_i))$, regardless of the price taker is willing to pay.

If we were to lift the condition $\Delta y_i \geq 0$ the best split actually could sell on some of the AMMs and hence include some form of arbitrage. This would make the problem far more complex. The constraint seems reasonable in practice (see also discussion in our optimal routing paper [15]).

There is a symmetric Δx version in the case of a sell market order which we omit.

Proposition 2 *Using the notations of Def. 5, assume Δx_i is a convex function of Δy_i, there exists a unique best split $\Delta y_i^\star$.*

[4] Δx_i is non-positive because it is paid out by the AMM.

The problem belongs to the class of nonlinear resource allocation problem (with linear constraints) as defined in the convex optimisation literature [25], [21, p. 23]. It is known to have a unique solution as soon as the feasible set is bounded, which is indeed the case here as $0 \leq \Delta y_i \leq \Delta y$. (This problem is also a particular case of optimal routing with concave volume-volume functions [15].)

Henceforth we assume that each f_i in $\mathcal{F}$ has convex (or concave if we take the absolute value of the output) volume-volume maps. The explicit calculations of §2.2.2 have shown that this is indeed the case for our lead examples f_U, f_L.

Definition 6 *Let a price q be given.*

It induces a partition of $\mathcal{F}$ into three classes:

$$\begin{array}{ll} \textit{`spent'} & b_i \leq q \\ \textit{`on'} & q \in (a_i, b_i) \\ \textit{`idle'} & q \leq a_i \end{array}$$

The price q determines the inputs $\Delta y_i(q) \geq 0$ and outputs $\Delta x_i(q) \leq 0$ of each member of $\mathcal{F}$:

$$\begin{array}{lll} \textit{if } f_i \textit{ is `on'} & \Delta y_i(q) = y_i(q) - y_i(p_i) & \Delta x_i(q) = x_i(q) - x_i(p_i) \\ \textit{if } f_i \textit{ is `spent'} & \Delta y_i(q) = y_i(b_i) - y_i(p_i) & \Delta x_i(q) = x_i(b_i) - x_i(p_i) \\ \textit{if } f_i \textit{ is `idle'} & \Delta y_i(q) = 0 & \Delta x_i(q) = 0 \end{array}$$

The key insight is:

Proposition 3 *The best split $\Delta y_i^\star$ of Δy over $\mathcal{F}$ is characterised by a unique $q^\star$ (and associated partition) such $\Delta y_i^\star = \Delta y_i(q^\star)$.*

To see this, consider two members f_1, f_2 of the family with respective allocations Δy_1, Δy_2. Write q_1 and q_2 for their post-trade current prices (see Def. 2):

$$\begin{array}{rcl} q_1 & = & \delta_{f_1}(p_1, \Delta y_1) \\ q_2 & = & \delta_{f_2}(p_2, \Delta y_2) \end{array}$$

Suppose without loss of generality $p_1 \leq p_2$.

In general, the output for input δy at price p (aka the volume-volume map) for a PPM (x, y) is given by:

$$\delta x(\delta y) \quad = \quad (x \circ y^{-1})(y(p) + \delta y) - x(p)$$

If $p < b$ and $\epsilon > 0$ small enough so that input $\delta y + \epsilon$ stays in range, the difference in outputs for inputs δy, $\delta y + \epsilon$ is given by:

$$\begin{array}{rcl} \delta\delta x & := & ((x \circ y^{-1})(y(p) + \delta y + \epsilon) - x(p)) - ((x \circ y^{-1})(y(p) + \delta y) - x(p)) \\ & = & (x \circ y^{-1})(y(p) + \delta y + \epsilon) - (x \circ y^{-1})(y(p) + \delta y) \\ & = & (x \circ y^{-1})'(y(p) + \delta y)\epsilon + o(\epsilon) \\ & = & (x'(q)/y'(q))\epsilon + o(\epsilon) \end{array}$$

where $q = y(y(p) + \delta y)$.

By condition (1) we have:

$$\delta\delta x \quad = \quad -\epsilon/q + o(\epsilon) \tag{2}$$

Suppose f_1 is not spent, and f_2 is not idle. This means $q_1 < b_1$, $q_2 > a_2$.

By (2), The modified inputs lead to a change in outputs which at first order is given by:

$$\begin{array}{rcl}\delta\delta x_1 & = & -\epsilon/q_1 \\ \delta\delta x_2 & = & +\epsilon/q_2\end{array}$$

Suppose $q_1 < q_2$, we have $|\delta\delta x_1| > |\delta\delta x_2|$. (We get more more output from f_1, than we get less from f_2, so to speak.) And as the changes in inputs is done at constant budget, it cannot be that Δy_i is the optimal split.

Hence either $q_1 = q_2$ and both f_1, f_2 are 'on', or $p_1 < p_2$. In the latter case, either f_1 is spent (there is no higher price at which it is ready to sell X) or f_2 is idle (there is no lower price at which it is ready buy X), or both, which is what we wanted to prove. □

By the earlier Proposition, the best split $\Delta y_i^\star$ of $\mathcal{F}$ is unique, by the one right above, this unique best split is determined by a unique optimal price $q^\star$.

We can combine both propositions to obtain a dual equivalent problem.[5]

Definition 7 (sweepline) *For $q \in \mathbb{R}_+$ we define the function:*

$$Y(q) \quad = \quad \sum_i \Delta y_i(q)$$

It is easy to see that:

Lemma 1 *$Y(q)$ is well-defined, increasing, continuous, bijective, $Y(q) = 0$ iff $q \leq \inf p_i$, and $(\Delta y_i(q);\ 1 \leq i \leq n)$ is the best split of its sum $\sum \Delta y_i(q)$ over $\mathcal{F}$.*

From which we can deduce:

Proposition 4 (sweepline execution) *As in Def. 5, pick a family $\mathcal{F}$, and fix a budget $\Delta y \geq 0$ for $\mathcal{F}$. The optimal price $q^\star$ associated to the best split of Δy over $\mathcal{F}$ is the unique solution to $Y(q^\star) = \min(\Delta y, K_Y)$, with $K_Y = \sum_i (y_i(b_i) - y_i(p_i))$ the maximal input of $\mathcal{F}$.*

Hence, our two optimisation problems are equivalent. The price-based form of Prop. 4 suggests an algorithm that drives a sweepline in price space, starting from $\inf p_i$ (the best buying price) and finishing at $\sup b_i$ (the worst buying price) or earlier if the order is filled. As $Y(q)$ is continuous increasing, there is either a unique price q at which our budget is spent, and there we stop, or else, $q = \sup b_i$, at which point we also stop, but all members of $\mathcal{F}$ are 'spent', and the market order is only partially filled.

The intuition of the algorithm below proceeds directly from Prop. 4. Instead of asking for quotes, it queries the set of current active sources by asking: *"if I want to push your marginal price p to a new value q, how much Δy do I need to give you?"*[6]

In the next Section, we fully define the algorithm and show that it is well suited to our model of families of convenient PPMs (defined in §2.3).

4 Push and Solve algorithm

The algorithm assumes the following initial data:
- a family $\mathcal{F} = (f_i(a_i, b_i), p_i;\ 1 \leq i \leq n)$ where f_i is defined on $[a_i, b_i]$ and p_i is f_i's current price
- $\mathbf{p} = (p_i;\ 1 \leq i \leq n)$ is assumed wlog to be sorted in non-decreasing order

[5]This characteristic price can be seen as the optimal value of the Lagrange multiplier for the budget constraint.
[6]Precisely the type of question that a price bending attacker would ask!

- $\mathbf{s} = (s_k;\ 1 \leq k \leq N)$ is the minimal increasing sequence which contains the a_is, and b_is[7]
- each f_i is split over $\mathbf{s}$, ie is of the form:

$$f_i \quad = \quad \oplus_{a_i \leq s_k < b_i} f_{ik}(s_k, s_{k+1})$$

The f_is are called the *sources*.

The sequence $\mathbf{s}$ is called the *price grid*, its members are called the *price points*.

We sometimes write $|\mathcal{F}| = n$ for the family 'height' n (number of PPMs), and $|\mathbf{s}|$ for its 'width' (number of price points).

To simplify notations we assume that $\mathbf{p}$ is increasing (no ties), and that $\mathbf{p}$ and $\mathbf{s}$ are disjoint.[8]

The algorithm takes as an input $\mathcal{F}$ as above and a budget Δy_0; it returns the best split of Δy_0 over $\mathcal{F}$.

Throughout its execution of a buy order, the algorithm maintains the following data:
- a current price level p (represented as a thick line in Fig. 4)
- a vector of splits $(\Delta y_i;\ 1 \leq i \leq n)$ which represents the current allocation
- a current remaining budget $\Delta y > 0$
- a list of the current parameters of the active sources (f_i is *active* if the current price p is in $[a_i, b_i)$).

As the name suggests it has two distinct phases:
- push: a search phase where we 'push' iteratively prices as in an ascending auction, and activate sources successively.
- solve: which is done (at most) once.

Fig. 4 illustrates the various steps with a family of three PPMs.

Caveat: We only describe the algorithm in the case of a buy market order with budget $\Delta y \geq 0$. The case of a sell market order with budget $\Delta x \geq 0$ is similar.

Initialisation At the start:
- the current price p is set to $p_1 = \min \mathbf{p}$ (this is the smallest price at which there is something to buy; recall that we suppose $\mathbf{p}$ is increasing)
- the vector of splits is set to 0 (nothing has been allocated yet)
- the remaining budget is set to its initial value Δy_0
- the list of active sources is set to f_1 with its parameters C_1

Push A push step is as follows:

1. Determination of next price q:

Let p_i be the smallest current price such that $p < p_i$, and let s_k be the smallest price point such that $p < s_k$; the next price is $\min(p_i, s_k)$ if defined.

If there is no such next price, $p \geq b_i$ for all i, hence the f_is are 'spent', and the initial market order cannot be fully filled. The algorithm terminates (and returns the vector of splits).

If there is a next price, we call it q.

2. Determination of cost of pushing to next price q:

The total push cost is given by $\Delta(q) := \sum_i (y_i(q) - y_i(p))$ where the summation is over the currently active sources. There are two possible outcomes.

2.1 If $\Delta y < \Delta(q)$

the remaining budget cannot pay for the push up to q, and the algorithm enters the solve phase (see below).

2.2 If $\Delta y \geq \Delta(q)$

[7] In practice, a most refined price grid will be defined once and for all.

[8] There are ties in $\mathbf{p}$ in the very unlikely case where several f_is have the same current price.

the remaining budget can pay for the push up to q:
- (i) the current price is set to q
- (ii) the vector of splits is incremented: $\Delta y_i \mathrel{+}= (y_i(q) - y_i(p))$
- (iii) the remaining budget is decremented: $\Delta y \mathrel{-}= \Delta(q)$
- (iv) if q is a price point s_k, the list of active sources and their parameters is updated

If at step (iii) the remaining budget reaches zero, the algorithm stops, returns the vector of splits which can fill the order.

Solve In this phase, the next price q is such that $\Delta y < \Delta(q)$. Hence $q^\star$ the optimal price is in (p, q) and is the unique solution to:

$$\Delta y \quad = \quad \textstyle\sum_i (y_i(q^\star) - y_i(p))$$

where the summation is over the currently active sources. The algorithm computes $q^\star$, increments accordingly the vector of splits $\Delta y_i \mathrel{+}= (y_i(q^\star) - y_i(p))$, and terminates by returning it.

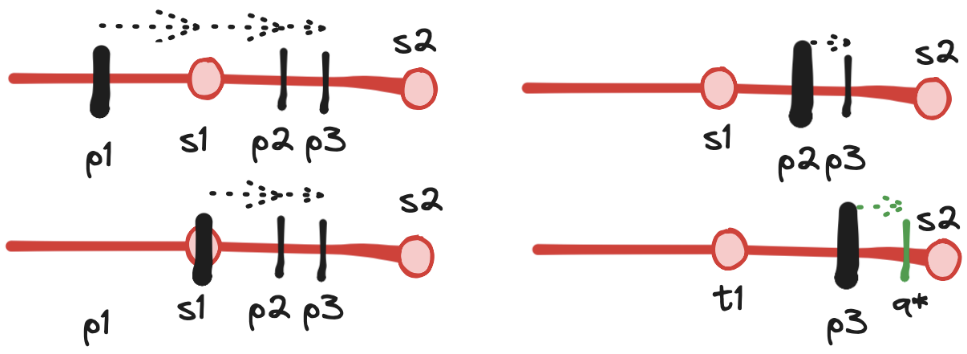

Figure 4: *Illustration of the push-and-solve procedure with three sources f_1, f_2, f_3 with respective marginal prices (at the start) $p_1 < p_2 < p_3$ with interleaving price points $s_1 < s_2$. There are three successive pushes respectively to s_1 (updating f_1's parameters), p_2 (activating f_2), and p_3 (activating f_3). The last step is a solve step (in green) which splits optimally the remaining budget between f_1, f_2, and f_3 all at a current price of p_3, to reach the final price $q^\star$.*

At each step (push or solve), the budget decreases (down to 0 if the order is fully filled), the vector of splits increases, $\sum_i \Delta y_i = \Delta y_0 - \Delta y$ stays invariant, and the vector of splits Δy_i is the best split for what of the original budget is already committed namely $\Delta y_0 - \Delta y$.

To see that the algorithm is *query-minimal* as claimed in the Introduction, we can track the number of queries made during a run. At initialisation, the algorithm queries the marginal prices of all members of $\mathcal{F}$ (n queries). During the run itself, f_i in $\mathcal{F}$ is queried at each new visited price point s_k in order to update the list of active sources and their parameters (Push, step 2.2.(iv)). Importantly, these queries are on a call-by-need basis. An upper bound is $|\mathcal{F}| \times |\mathbf{s}| = n \times N$ (the height of the family times its width), but the query cost of a typical run will be far lower.

Many algorithms can solve the problem of Prop. 4. For instance, one could proceed to a search of $q^\star$ by dichotomy. However: 1) our algorithm is query-minimal which is key if the algorithm is to be implemented on-chain where reading storage is typically costly (especially so as the family width $|\mathbf{s}|$ can be very large); 2) because of arbitrage we expect the marginal prices of the sources $\mathbf{p}$ to cluster near some driving price, and, on liquid markets at least, we expect $q^\star$ to not jump over multiple price points. Both considerations will tend to drive down the average number of queries to a much lower number than the worst case, namely $n \times N$, and reinforce the advantages of query-minimality.

4.1 Application to optimal splitting of convenient families of PPMs

The push-and-solve algorithm can be seen as a multi-source generalisation of the Uniswap v3 execution algorithm (modulo internal manipulations of state) which is essentially the case of one source of the convenient f_U format.

The multiple sources stack up as new active sources are activated when the current price up-crosses their marginal price (Push, step 2.2.(iv)). In the mono-source case, there is only ever one active source and one current price to track.

Interestingly, our algorithm is agnostic to the format used by the various sources. Ie one can mix sources following different formats, eg f_L and f_U. So it is both multi-source and multi-format.

However, if we suppose suppose that all sources in $\mathcal{F}$ obey the same convenient format with functions ϕ, ψ, the equation in the solve step takes the following simpler form:

$$\Delta y \quad = \quad \textstyle\sum_i (y_i(q^\star) - y_i(p)) \quad = \quad (\sum_i D_{ik})(\psi(q^\star) - \psi(p))$$

where the D_{ik} are the parameters of the current vertical stack of active sources ($f_{ik};\ 1 \leq i \leq |\mathcal{F}|$). Thus the solution can be written explicitly as:

$$q^\star \quad = \quad \psi^{-1}(\Delta y / (\textstyle\sum_i D_{ik}) + \psi(p))$$

In addition, in the case the format is f_U, or f_L, the corresponding ϕ, ψ functions have a simple closed form, eg $\psi_U^{-1}(z) = z^2$, and $\psi_L^{-1}(z) = \sqrt{2z}$, and can be readily computed.

From the application point of view this means that the optimal splitting problem for sources of the Uniswap v3 type can be solved in a query-minimal fashion (so soberly memory-wise), and completely symbolically (so soberly math-wise).

The same applies for the linear format f_L, but this is of lesser immediate importance as this type of AMM, although very simple, is not implemented yet (to our knowledge), and certainly not as wide-spread as the Uniswap v3 type of which there are many clones competing for liquidity (as said in the Introduction).

4.2 Comments

Note that a run is not actually executing the order, it is a simulation (which queries information from the sources on a need-to-know basis). Instead, it builds the best split to be executed once fully computed. One could execute the partial splits on-the-fly on their respective sources, since the algorithm never backtracks. But this would not be efficient in practice, because each partial execution would trigger repeated internal updates in the sources, which are best shared in a global order.

In concrete applications, taker can send a buy order with a maximal price (or maximal splippage). This just is another stopping condition for the algorithm which one can add when computing the next price (Push, step 2).

As explained in the split-and-sum subsection (§2.3), not only is every source split along **s**, but we also think of its components f_{ik}s has implicitly summed. (This plays no role in the algorithm.)

5 Slippage calculations

Because our push-and-solve algorithm has low computational cost (especially in the mono-format case) it is realistic. Because it is optimal it improves execution in principle. In this section we would like get some idea of the benefit we can expect in practice.

To do so we compute the price impact or slippage induced by a swap on a single source $f_U(a, b, p)$. Then we compute the same in the multi-source mono-format case, with the much simplifying, but not unrealistic assumption that all marginal prices of our sources are equal. In this special case, we can easily measure the improvement (or slippage reduction) brought about by our algorithm.

So, suppose we swap in Δy. Write p, q for the marginal prices before and after swap.

We have seen (§2.1.2) that the execution price is the geometric mean of p and q:

$$p_{ex} \quad := \quad -\Delta y/\Delta x \quad = \quad \sqrt{qp}$$

Define slippage as the relative change of price $S := p_{ex}/p - 1$:

$$\begin{aligned} S &= \sqrt{qp}/p - 1 \\ &= (\sqrt{q} - \sqrt{p})/\sqrt{p} \\ &= (L\sqrt{p})^{-1} \cdot \Delta y \end{aligned}$$

We see that the slippage of market orders that do not traverse a (possibly L-changing) price point is linear in the swap input Δy. The term $L\sqrt{p}$ can be construed as the inertia or depth of the market.

Now consider a family $(f_U(a, b, L_i);\ 1 \leq i \leq n)$ of n sources of the f_U type with the same domain. If we assume they have the same current price, they behave as one source $f_U(a, b, \sum_i L_i)$. Hence the ratio of the multi-source slippage to the mono-source one is

$$S_n/S_1 \quad = \quad \max(L_i)/\sum_i L_i \tag{3}$$

where we have picked the best mono source. This ratio is independent of the common marginal price p and input Δy (conditioned on the fact that Δy does not overflow).

There are two extreme cases:

(i) if there is one dominant pool $S_n \sim S_1$ the optimal splitting will not be substantially better than choosing the best source (which is easy to do right after initialisation)

(ii) if all pools are equally liquid $S_n \sim S_1/n$ the slippage improvement is measured by $1/n$ where n is the number of sources.

In other words, unsurprisingly, the improvement brought about by the optimal splitting algorithm is all the more interesting when the liquidity around the current price is more fragmented (more sources) and more evenly so (near equal liquidities).

6 Conclusion

We have shown a simple algorithm to execute a swap across multiple AMMs defined on the same pair so as the maximise the swap's output. For the technique to apply, those AMMs have to be built using a specific building block called a price-parametrised AMM (PPM). Importantly, the algorithm is simple enough that one can implement it readily as a smart contract.

In that respect, it is very different from off-chain aggregating algorithms, and can be used to unconditionally improve their on-chain execution (as explained in the Introduction), to the extent that its gas expenses are negligible compared to the improvement it brings about.

The algorithm can also be used directly to combat the negative consequences of the fragmentation of liquidity. An application which is particularly timely with the imminent introduction of Uniswap v4 markets [2]. Indeed the novel Uniswap v4 protocol introduces yet another level of flexibility, with the so-called "hooks". The market engine inside stays the same (ie an AMM of type Uniswap v3) but hooks allow, among other things, to source the LPs' liquidity at the time of execution and thus to collect additional yield LPs. The idea is very similar to Mangrove's "offer-is-code" approach to Decentralised Exchanges [26]. If many hooks become popular, this will unavoidably lead to more fragmentation of markets, to the detriment of takers. There our algorithm could be implemented as an on-chain front-end to any collection of Uniswap v4 markets (on a given pair) and protect takers while letting LPs enjoy their hooks.

Another contribution of this paper is the introduction of general format which is sufficient for developing concentrated AMMs with the same low complexity aggregability of LP positions as Uniswap v3. Our algorithm applies equally to those, and could even combine sources with different such formats.

References

[1] Carlo Acerbi and Giacomo Scandolo. Liquidity risk theory and coherent measures of risk. *Quantitative Finance*, 8(7):681–692, 2008.

[2] Hayden Adams, Moody Salem, Noah Zinsmeister, Sara Reynolds, Austin Adams, Will Pote, Mark Toda, Alice Henshaw, Emily Williams, and Dan Robinson. Uniswap v4 core [draft], 2023.

[3] Hayden Adams, Noah Zinsmeister, and Dan Robinson. Uniswap v2 core, March 2020.

[4] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core, March 2021.

[5] Rossella Agliardi and Ramazan Gençay. Optimal high-frequency trading with limit and market orders. *Quantitative Finance*, 17(10):1565–1584, 2017.

[6] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3:5–40, 2001.

[7] Guillermo Angeris and Tarun Chitra. Improved price oracles: Constant function market makers. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 80–91, 2020.

[8] Guillermo Angeris, Alex Evans, Tarun Chitra, and Stephen Boyd. Optimal routing for constant function market makers. In *Proceedings of the 23rd ACM Conference on Economics and Computation*, pages 115–128, 2022.

[9] Marcel Blais and Philip Protter. An analysis of the supply curve for liquidity risk through book data. *International Journal of Theoretical and Applied Finance*, 13(06):821–838, 2010.

[10] Nicolas Bundi and Charles-Albert Lehalle. Optimal execution across multiple venues in limit order markets. *Quantitative Finance*, 23(1):105–122, 2023.

[11] Umut Çetin, Robert A Jarrow, and Philip Protter. Liquidity risk and arbitrage pricing theory. *Finance and Stochastics*, 8(3):311–341, 2004.

[12] Jonathan Chávez-Casillas, José E. Figueroa-López, Chuyi Yu, and Yi Zhang. Adaptive optimal market making strategies with inventory liquidation cost, 2024.

[13] Rama Cont and Arseniy Kukanov. Optimal order placement in limit order markets. *Quantitative Finance*, 17(1):21–39, 2017.

[14] Vincent Danos, Hamza El Khalloufi, and Julien Prat. Global order routing on exchange networks. In *Financial Cryptography and Data Security*, pages 207–226. Springer, 2021.

[15] Vincent Danos, Hamza El Khalloufi, and Julien Prat. Global order routing on exchange networks. In *Financial Cryptography and Data Security. FC 2021 International Workshops*, pages 207–226, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

[16] Vincent Danos and Weijia Wang. Consistency of Automated Market Makers. In Yackolley Amoussou-Guenou, Aggelos Kiayias, and Marianne Verdier, editors, *4th International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2022)*, volume 110 of *Open Access Series in Informatics (OASIcs)*, pages 4:1–4:12, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[17] Theo Diamandis and Guillermo Angeris. The convex geometry of network flows. *arXiv preprint arXiv:2408.12761*, 2024.

[18] Theo Diamandis, Max Resnick, Tarun Chitra, and Guillermo Angeris. An efficient algorithm for optimal routing through constant function market makers. *arXiv preprint arXiv:2302.04938*, 2023.

[19] Michael Egorov. Stableswap-efficient mechanism for stablecoin liquidity. *Retrieved Feb*, 24:2021, 2019.

[20] Michael Egorov. Automatic market-making with dynamic peg. *Retrieved Dec 2021*, June 2021.

[21] Toshihide Ibaraki and Naoki Katoh. *Resource allocation problems - algorithmic approaches.* MIT Press series in the foundations of computing. MIT Press, 1988.

[22] Robert A Jarrow and Philip Protter. Liquidity risk and risk measure computation. *Review of Futures Markets*, 11(1):27–39, 2005.

[23] Shingo Kuno and Masamitsu Ohnishi. Optimal execution in the multi-venue market with dark pools. *Asia-Pacific Financial Markets*, 24:95–116, 2017.

[24] Fernando Martinelli and Nikolai Mushegian. A non-custodial portfolio manager, liquidity provider, and price sensor, 2020. https://balancer.finance, v2019-09-19.

[25] Michael Patriksson. A survey on the continuous nonlinear resource allocation problem. *European Journal of Operational Research*, 185(1):1–46, 2008.

[26] The Mangrove project. The "offer-is-code" approach to decentralised exchanges. 2021.