Multi-token Batch Auctions with Uniform Clearing Prices

–

Features and Models

Tom Walther
tom@gnosis.pm

January 17, 2019

This document describes the problem of multi-token batch auctions with uniform clearing prices as a price-finding mechanism proposed for a decentralized token trading platform. Moreover, it contains solution approaches based on combinatorial optimization formulations, as well as some computational results.

Contents

1 Introduction

In continuous-time token exchange mechanisms, orders are typically collected in order books of two tokens that are traded against each other. A trade happens whenever a buy order of one token is matched by a sell order of the other, i.e., if there exists an exchange rate that satisfies the limit prices stated in the respective orders. In a setting of multiple tradeable tokens, one separate order book is required for every token pair combination. This may significantly limit liquidity for less frequently traded token pairs and lead to large bid-ask spreads and low trading volumes.

In our approach, we want to collect orders for a set of multiple tokens in a single joint order book and compute exchange prices for all token pairs simultaneously at the end of discrete time intervals (*multi-token batch auction*). Trades between the same token pairs are then all executed at the same exchange rate (*uniform clearing price*). Moreover, this mechanism enables so-called *ring trades*, where orders are matched along cycles of tokens. In order to exclude arbitrage opportunities, we require prices to be consistent along such cycles, i.e., we want the constraint

$$p_{j|k} \cdot p_{k|l} = p_{j|l} \tag{1}$$

to be satisfied for the exchange rates between all tokens τ_j, τ_k, τ_l.

The concept of frequent batch auctions aiming at reducing arbitrage and improving liquidity has been investigated extensively in [3]. Moreover, the advantages of uniform-price clearing have been discussed in [4]. In this document, we want to describe the problem of determining uniform clearing prices for all pairs of tokens involved in a multi-token batch auction process, and present a mixed-integer programming (MIP) solution approach.

2 Problem statement

In this section, we want to give a high-level description of the problem that we are investigating, thereby introducing some general notation and giving an overview of the constraints and properties that we will focus on.

Data Let $\mathcal{T} := \{\tau_1 \dots \tau_n\}$ denote the set of the n tokens that we want to consider. Additionally, we introduce an artificial token τ_0 that shall be referred to as *reference token*, which will become important in our modelling approaches later on in Section 3. For convenience of notation, we use $\mathcal{I}^t := \{1 \dots n\}$ to denote the indices of our token set.

Pairwise exchange rates between two tokens τ_j and τ_k will be denoted by $p_{j|k}$, meaning the price of one unit of τ_j measured in units of τ_k. As an example, if $p_{j|k} = 10$, one would need to pay an amount of 10 units of τ_k in order to purchase one unit of τ_j.

Let there be a set $\mathcal{O} = \{\omega_1 \dots \omega_N\}$ of N *limit sell orders* in the batch to be processed and let $\mathcal{I}^o := \{1 \dots N\}$ denote its set of indices. Every order is specified as a tuple $\omega = (j, k, \overline{y}, \pi)$ whose semantic meaning is

"Sell (at most) $\overline{y}$ units of token τ_j for token τ_k if the exchange rate $p_{j|k}$ is at least π".

More precisely, we define and understand the elements of every such tuple as follows:

$j \in \mathcal{I}^t$	...	τ_j is the token to be sold
$k \in \mathcal{I}^t$	...	τ_k is the token to be bought
$\overline{y} \in \mathbb{R}_{\geq 0}$	...	maximum amount of τ_j to be sold
$\pi > 0$	...	limit price for order execution, i.e., $p_{j\|k} \geq \pi$

Please note that, theoretically, limit buy orders could be expressed in a similar way as a tuple $(j, k, \overline{x}, \pi)$ that reads

"Buy (at most) $\overline{x}$ units of token τ_j for token τ_k if the exchange rate $p_{j|k}$ is at most π".

The difference between buy and sell orders is the following:

- In a buy order, a fixed (maximum) buy volume $\overline{x}$ is set, so the trader indicates a preference to buy $\overline{x}$ units of token τ_j. The amount of units of token τ_k to be sold is flexible, as long as the limit price is respected (the smaller, the better).
- In a sell order, a fixed (maximum) sell volume $\overline{y}$ is set, so the trader indicates a preference to sell $\overline{y}$ units of token τ_k. The amount of units of token τ_j to be bought is flexible, as long as the limit price is respected (the higher, the better).

However, for notational simplicity, we will restrict ourselves to only considering sell orders in this paper. In fact, assuming that the trader has constant utility for an arbitrary number of

buy-tokens, the following argument can be used to convert limit buy orders to sell orders: Consider a buy order $(j, k, \overline{x}, \pi)$, which reveals that the trader would be ready to sell at most $\pi \cdot \overline{x}$ tokens τ_k in exchange for tokens τ_j. Then, the trader would not object to receiving more than $\overline{x}$ tokens τ_j for a maximum of $\pi \cdot \overline{x}$ tokens τ_k, i.e., if the exchange rate $p_{j|k}$ is lower than π. Hence, the trader would only benefit from submitting a sell order $(k, j, \pi \cdot \overline{x}, \frac{1}{\pi})$.

Objectives Having collected a batch of buy and sell orders for our set of tokens, we ultimately want to compute exchange rates for all token pairs. Therefore, the following optimization criteria could be used:

- maximize the amount of trade that is enabled (with respect to some reference token)
- maximize *trader welfare*, e.g., defined as difference of paid price vs. limit price

Constraints The solution that we are aiming at needs to satisfy several requirements that can be stated on a high level as follows:

- *(limit price)*:
  for all orders $\omega = (j, k, \overline{y}, \pi) \in \mathcal{O}$: the order can only be executed (fully or fractionally) if the exchange rate satisfies the limit price, i.e., $p_{j|k} \geq \pi$.
- *(token balance)*:
  for every token $\tau \in \mathcal{T}$: the amount of tokens τ that were bought must equal the amount of tokens τ that were sold across all orders.
- *(price coherence)*:
  for all token pairs (τ_j, τ_k): $p_{j|k} \cdot p_{k|j} = 1$.
- *(arbitrage freeness)*:
  for all token triples (τ_j, τ_k, τ_l): $p_{j|k} \cdot p_{k|l} = p_{j|l}$.

Lemma 2.1. *(arbitrage freeness) ⇒ (price coherence).*

Proof. Consider three tokens $\{\tau_j, \tau_k, \tau_l\} \subset \mathcal{T}$. We apply the arbitrage-freeness condition twice:

$$\text{(i)}\ \ p_{j|k} \cdot p_{k|l} = p_{j|l}$$
$$\text{(ii)}\ \ p_{j|l} \cdot p_{l|k} = p_{j|k}$$

Inserting (i) into (ii) and assuming prices to be strictly positive yields

$$p_{j|k} \cdot p_{k|l} \cdot p_{l|k} = p_{j|k} \qquad \Leftrightarrow \qquad p_{k|l} \cdot p_{l|k} = 1$$

□

Notice that the above constraints also imply $p_{j|j} = 1$ for every token $\tau_j \in \mathcal{T}$.

Example A typical instance of our batch auction problem could look like shown in Figure 1. It is assumed that some token pairs will encounter high trading demand whereas other will only rarely be traded on.

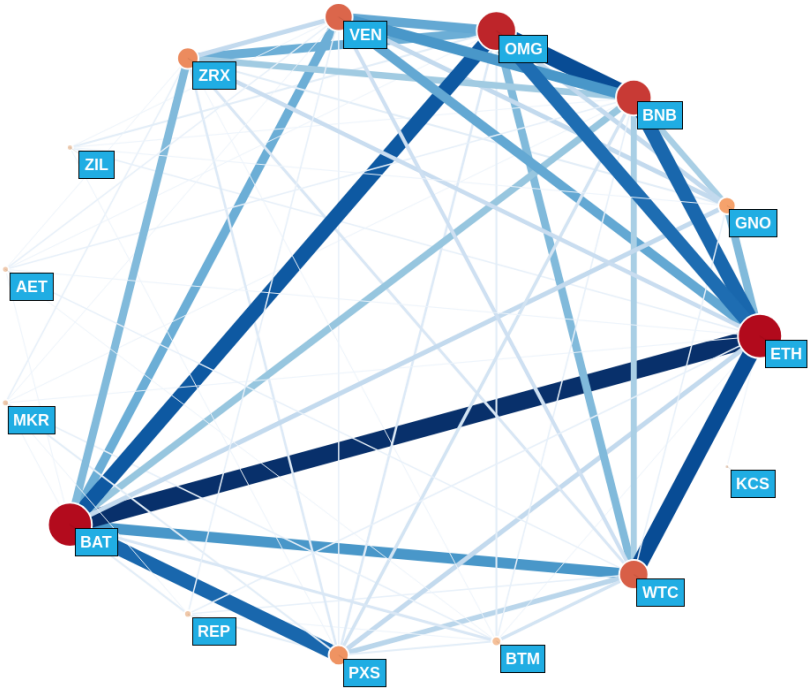

Figure 1: Token-order-graph representing a multi-token batch auction instance. Nodes correspond to tokens (with their size indicating the trading demand), the thickness of an edge represents the number of orders placed on the respective token pair.

3 Models and Formulations

In this section, we want to present and discuss several solution approaches for our batch auction problem in terms of mathematical optimization formulations.

Data At first, we will express all information given in the orders in terms of data matrices and vectors, aiming at being able to write down a complete optimization model. Recall that every limit sell order $\omega \in \mathcal{O}$ is defined by a tuple $(j, k, \overline{y}, \pi)$ specifying the sell- and buy-token, the maximum amount of tokens to be sold, as well as the limit price.

We introduce two data matrices

$$\mathbf{T}^b \in \{0,1\}^{N \times n} \quad \text{with} \quad \mathbf{T}^b \ni t^b_{i,j} = 1 \Leftrightarrow \text{token } \tau_j \text{ to be bought in order } \omega_i$$
$$\mathbf{T}^s \in \{0,1\}^{N \times n} \quad \text{with} \quad \mathbf{T}^s \ni t^s_{i,j} = 1 \Leftrightarrow \text{token } \tau_j \text{ to be sold in order } \omega_i$$

As for now, we are only considering orders of one token type against one other, so there must be exactly one entry equal to 1 per row (order) in both $\mathbf{T}^b$ and $\mathbf{T}^s$.

Let $(\overline{y}_i) =: \overline{\mathbf{y}} \in \mathbb{R}^N_{\geq 0}$ contain the maximum amounts of tokens to be sold in every order.

Finally, the limit prices of all orders shall be stored as vector $(\pi_i) =: \boldsymbol{\pi} \in \mathbb{R}^N_{\geq 0}$, where $\pi_i \in \boldsymbol{\pi}$ refers to the exchange rate between the respective buy and sell tokens at which order ω_i may be executed (according to the definition in Section 2).

Variables Primarily, we want to determine exchange rates between all token pairs. However, if we directly considered those exchange rates being variables, modelling arbitrage-freeness constraints of type (1) automatically leads to many multiplications being required. This may result in unnecessary limitations to the problem size that is tractable as well as numerical instability. In order to circumvent this issue, we can instead represent all token prices only with respect to the single (artificial) reference token τ_0. Therefore, let $p_j := p_{j|0}$ denote the price of token τ_j expressed in units of τ_0 (hence, $p_0 = 1$). Applying the arbitrage-freeness and price-coherence conditions directly, we can express the exchange rate between two tokens τ_j and τ_k as

$$p_{j|k} = p_{j|0} \cdot p_{0|k} = \frac{p_{j|0}}{p_{k|0}} = \frac{p_j}{p_k}. \tag{2}$$

Let $\mathbf{p} \in \mathbb{R}^n_{>0}$ be a vector containing all prices p_j.

We now have some freedom to decide what precisely the artificial reference token should represent. For example, we could select one particular $\tau_j \in \mathcal{T}$ to be the *numeraire*, i.e.,

$$p_0 = p_j. \tag{3a}$$

Then, all trading volume would be computed with respect to τ_j (which we will explain later in the model formulation sections), making this the principal token in the batch auction. Alternatively and more generally, we could also identify τ_0 with a weighted sum of all participating tokens in $\mathcal{T}$, i.e.,

$$p_0 = \sum_{j \in \mathcal{I}^t} \gamma_j \cdot p_j, \tag{3b}$$

with weights $\gamma_j > 0$. This way, it is possible to give similar importance to all tokens. In the following, we will use (3b) together with weights $\gamma_j = 1/(n \cdot p_j^{\text{old}})$, where p_j^{old} denotes the price of token τ_j found in the previous batch auction iteration.

Implications of unavailable p_j^{old}?!

Observation 3.1. *The choice of representation of the reference token has an influence on the solution. This can be seen by considering a single token pair (τ_j, τ_k) with only two buy orders $\omega_1 = (j, k, 1.0, +\infty, 2.0)$ and $\omega_2 = (k, j, 1.5, +\infty, 1.0)$. Both orders can be matched if $p_{j|k} \in [1, 2]$. However, if we choose token τ_j to be the reference token, a maximum of $1.0\,\tau_j$ can be transacted for prices $p_{j|k} \in [1, 1.5]$, whereas the maximum trading volume of $1.5\,\tau_k$ can be achieved when $p_{j|k} \in [1.5, 2]$.*

Additionally, we want to bound the deviation of the new exchange rates $p_{j|k}$ from the previously computed exchange rates $p_{j|k}^{\text{old}}$. Therefore, let p_j^{old} denote the price of token τ_j found in the previous batch auction iteration. Using a maximum fluctuation parameter $\delta > 0$, we impose the conditions

$$\begin{aligned} p_{j|k} \in \left[\left(\frac{1}{1+\delta}\right) p_{j|k}^{\text{old}}, (1+\delta)\, p_{j|k}^{\text{old}}\right] \quad &\Leftrightarrow \quad \frac{p_j}{p_k} \in \left[\left(\frac{1}{1+\delta}\right) \frac{p_j^{\text{old}}}{p_k^{\text{old}}}, (1+\delta)\, \frac{p_j^{\text{old}}}{p_k^{\text{old}}}\right] \\ &\Leftrightarrow \quad \begin{cases} p_j \geq \left(\frac{1}{1+\delta}\right) \frac{p_j^{\text{old}}}{p_k^{\text{old}}}\, p_k \\ p_j \leq (1+\delta)\, \frac{p_j^{\text{old}}}{p_k^{\text{old}}}\, p_k \end{cases} . \end{aligned} \tag{4}$$

As an example, $\delta = 1$ would set the bounds to half/twice the previous exchange rates. Note that, if no previous price exists for some token τ_j, the exchange rate bounds for token pairs involving τ_j could be determined on the basis of the limit prices given in the respective orders, or simply be set to $[0, \infty]$. In theory, the model permits a different fluctuation parameter on every token pair, which allows to capture real-world expections, e.g., , related to the liquidity that is available on each token pair.

Moreover, it is helpful to incorporate some explicit lower and upper bound for the price of every token τ_j, so let us require $p_j \in [\underline{p}_j, \overline{p}_j]$.

Lemma 3.2. *For the choice of weights* $\gamma_j = 1/(n \cdot p_j^{\text{old}})$ *in* (3b) *and a given maximum fluctuation parameter* δ*, we can impose bounds*

$$p_j \in \left[\left(\frac{1}{1+\delta}\right) p_j^{\text{old}}, (1+\delta)\, p_j^{\text{old}}\right] \forall\, j \in \mathcal{I}^t. \tag{5}$$

Proof. We will prove that the upper bound holds. Proving the lower bound works analogous. Assume $p_j > (1+\delta)\, p_j^{\text{old}}$. Then, (4) yields

$$(1+\delta)\, \frac{p_j^{\text{old}}}{p_k^{\text{old}}}\, p_k > (1+\delta)\, p_j^{\text{old}} \qquad \Leftrightarrow \qquad p_k > p_k^{\text{old}} \qquad \forall k \neq j.$$

Inserting into (3b) leads to a contradiction:

$$p_0 = 1 = \frac{1}{n \cdot p_j^{\text{old}}}\, p_j + \sum_{j \neq k \in \mathcal{I}^t} \frac{1}{n \cdot p_k^{\text{old}}} \cdot p_k > \frac{1+\delta}{n} + \sum_{j \neq k \in \mathcal{I}^t} \frac{1}{n} > 1.$$

□

All price bounds shall be stored in vectors $\underline{\mathbf{p}}$ and $\overline{\mathbf{p}}$, respectively.

In addition to computing optimal token prices $\mathbf{p}$, we also want to determine the status of execution for every order, i.e., the amount of tokens that were bought and sold. Therefore, the number of tokens bought in an order ω_i shall be denoted by x_i, with $(x_i) =: \mathbf{x} \in \mathbb{R}^N_{\geq 0}$. Conversely, $(y_i) =: \mathbf{y} \in \mathbb{R}^N_{\geq 0}$ shall represent the amounts of tokens sold.

We need to distinguish two different behaviours for every order: Either the exchange rate on the respective token pair satisfies its limit price (and thus the order may be executed), or it does not (thus the order needs to remain untouched). Optionally, we could incorporate a minimum fraction of execution, denoted by a parameter $r^{\min} \in [0,1]$, for the case that an order may be executed. This parameter can either be set globally for all orders, or for every order individually.

In general, for an order given as $\omega = (j, k, \overline{y}, \pi)$, this yields the following two sets of constraints:

$$\mathfrak{C}^1 := \left\{ \begin{array}{rcl} y & \in & [\, r^{\min}\, \overline{y},\, \overline{y}\,] \\ y \cdot p_j & = & x \cdot p_k \\ \frac{p_j}{p_k} & \geq & \pi \end{array} \right\} \quad \text{and} \quad \mathfrak{C}^0 := \left\{ \begin{array}{rcl} x & = & 0 \\ y & = & 0 \\ \frac{p_j}{p_k} & < & \pi \end{array} \right\} \tag{6}$$

Objective Our primary objective shall be to determine token prices/exchange rates such that the total feasible *trading volume* is maximized. Thereby, trading volume refers to the sum of all tokens traded across all orders, weighted by their respective price, i.e., it measures

the value of all transactions in terms of units of the reference token τ_0. Thus, for every order $\omega = (j, k, \overline{y}, \pi)$, we introduce an additional variable $v \geq 0$, for which we require

$$v = y \cdot p_j \qquad \text{and} \qquad v = x \cdot p_k \tag{7}$$

The trading volumes of all orders shall be stored in a vector $\mathbf{v} \in \mathbb{R}^N_{\geq 0}$. It is then our objective to maximize $\sum_{i \in \mathcal{I}^o} v_i$.

Alternatively, we want to consider optimizing the total *trading surplus* (or *traders' welfare*). If some sell order $\omega = (j, k, \overline{y}, \pi)$ is executed, i.e., a certain amount y of token τ_j sold, the trading surplus w measures the value of the difference between the amount x of tokens τ_k that is purchased and the minimum amount $y \cdot \pi$ that the trader would have accepted:

$$w = (x - y \cdot \pi)\, p_k \tag{8}$$

We store the trading surplus of every order in a vector $\mathbf{w} \in \mathbb{R}^N_{\geq 0}$.

More research on trading surplus (linear formulation, relation to volume, etc.)!

3.1 Nonlinear programming model

When looking at the sets of constraints (6), the most intuitive model seems to be a nonlinear program. As we will see, it is possible to avoid binary variables.

$$\text{maximize} \quad \sum_{i \in \mathcal{I}^o} v_i \tag{9a}$$

$$\text{subject to} \quad \sum_{i \in \mathcal{I}^o} t^b_{i,j}\, x_i = \sum_{i \in \mathcal{I}^o} t^s_{i,j}\, y_i \qquad \forall\, j \in \mathcal{I}^t \tag{9b}$$

$$v_i = x_i \sum_{j \in \mathcal{I}^t} t^b_{i,j}\, p_j \qquad \forall\, i \in \mathcal{I}^o \tag{9c}$$

$$v_i = y_i \sum_{j \in \mathcal{I}^t} t^s_{i,j}\, p_j \qquad \forall\, i \in \mathcal{I}^o \tag{9d}$$

$$y_i \leq \overline{y}_i \qquad \forall\, i \in \mathcal{I}^o \tag{9e}$$

$$x_i \geq y_i\, \pi_i \qquad \forall\, i \in \mathcal{I}^o \tag{9f}$$

$$n = \sum_{j \in \mathcal{I}^t} \frac{1}{p^{\text{old}}_j} \cdot p_j, \tag{9g}$$

$$p_j \leq (1+\delta)\, \frac{p^{\text{old}}_j}{p^{\text{old}}_k}\, p_k \qquad \forall j,k \in \mathcal{I}^t \times \mathcal{I}^t \tag{9h}$$

$$x_i, y_i, v_i \in \mathbb{R}_{\geq 0} \qquad \forall\, i \in \mathcal{I}^o \tag{9i}$$

$$p_j \in \left[\left(\frac{1}{1+\delta} \right)\, p^{\text{old}}_j, (1+\delta)\, p^{\text{old}}_j \right] \qquad \forall\, j \in \mathcal{I}^t \tag{9j}$$

The objective function (9a) maximizes the total volume in terms of units of the reference token τ_0 that is processed with all orders.

Constraint (9b) ensures that the total numbers of tokens bought and sold are equal for every token across all orders. The summations in this constraint are only responsible for selecting the correct tokens that are traded in the orders.

The constraints (9c) and (9d) compute the buy and sell trade volume for every order with respect to the reference token, and make sure these two are equal. This guarantees that the token prices are chosen such that they are consistent with the traded amounts of tokens. If the traded token amounts x_i and y_i are zero for some order ω_i, i.e., ω_i is not executed at

all, the corresponding trade volume v_i will be zero as well. However, this comes at the price of introducing nonlinearity (and even nonconvexity) into the model.

The maximum token amount to be sold as specified in every limit order is incorporated into the model via constraint (9e). Constraint (9f) ensures that the limit price is satisfied, if the order is to be executed, or that both x_i and y_i are set to zero otherwise. Altogether, the constraints (9c)–(9f) capture both cases of (6) simultaneously.

Finally, as described earlier, the constraints (9g) and (9h) specify the representation of the reference token and enforce a maximum fluctuation of exchange rates on all token pairs.

This model allows for orders to be left untouched even if the computed token prices satisfy the given limit price, so there is no way to express a value other than $r^{\min} = 0$ for the minimum order execution parameter. While this may a desirable property of the model, we believe that this can only be amended with the introduction of binary variables that indicate whether prices allow for an order to be executed, or not.

3.2 Mixed-integer linear programming model I

Since nonlinearity and nonconvexity usually pose strong restrictions on the tractable model size, we want to proceed with proposing a mixed-integer linear programming (MIP) formulation as an alternative to the NLP model presented above. Thereby, the challenge is to find an equivalent linear formulation for (6).

The key idea is to avoid explicitely computing the values x and y for every order $\omega = (j, k, \overline{y}, \pi)$, but only working with the trading volume v as defined in (7) instead. With uniform clearing prices, x and y can be computed unambiguously from the value of v later on. The order constraints (6) can be expressed as follows:

$$\tilde{\mathfrak{C}}^1 := \left\{ \begin{array}{rcl} v & \in & [\, r^{\min}\, \overline{y}\, p_j,\ \overline{y}\, p_j\,] \\ \frac{p_j}{p_k} & \geq & \pi \end{array} \right\} \quad \text{and} \quad \tilde{\mathfrak{C}}^0 := \left\{ \begin{array}{rcl} v & = & 0 \\ \frac{p_j}{p_k} & < & \pi \end{array} \right\} \tag{10}$$

In order to reformulate (10) in terms of a mixed-integer linear program, we first introduce a binary variable $z \in \{0, 1\}$ for every order and associate its states to the constraint sets as $\{z = 1\} \Leftrightarrow \tilde{\mathfrak{C}}^1$ and $\{z = 0\} \Leftrightarrow \tilde{\mathfrak{C}}^0$. For modelling these dependencies, there exist two major strategies that we will apply in the following: the *big-M* approach as, e.g., mentioned in [2], as well as *disjunctive programming*, introduced by Balas [1].

Big-M reformulation In the big-M approach, the constraints for both $\tilde{\mathfrak{C}}^0$ and $\tilde{\mathfrak{C}}^1$ are modified so that they are equivalent to the original ones if the binary variable takes the respective value, and are rendered redundant by large-enough constants in the opposite case. As for our constraint systems, this translates into:

$$p_j \geq \pi\, p_k + (1 - z)(\underline{p}_j - \pi \overline{p}_k) \tag{11a}$$

$$p_j \leq (\pi - \varepsilon)\, p_k + z\, (\overline{p}_j - (\pi - \varepsilon)\, \underline{p}_k) \tag{11b}$$

$$v \leq \overline{y}\, p_j \tag{11c}$$

$$v \leq \overline{y}\, \overline{p}_j\, z \tag{11d}$$

$$v \geq r^{\min}\, \overline{y}\, (p_j - (1 - z)\, \overline{p}_j) \tag{11e}$$

It can easily be verified that substituting $z = 1$ and $z = 0$ yields the desired constraints (10). Notice that the strict inequality $\frac{p_j}{p_k} < \pi$ in $\tilde{\mathfrak{C}}^0$ is approximated by an inequality $\frac{p_j}{p_k} \leq \pi - \varepsilon$, with some very small $\varepsilon > 0$.

Disjunctive programming reformulation The advantage of the big-M reformulation is its simplicity and compactness. However, its relaxation is not as tight as it can be, i.e., it

does not describe the convex hull of the feasible regions of $\tilde{\mathfrak{C}}^0$ and $\tilde{\mathfrak{C}}^1$. This property can be secured in the disjunctive programming approach through the addition of additional auxiliary variables.

For every order $\omega = (j, k, \overline{y}, \pi)$, we introduce two pairs of auxiliary non-negative price variables: $p^{s,0}, p^{s,1}$ – referring to the price p_j of the sell-token, as well as $p^{b,0}, p^{b,1}$ – referring to the price p_k of the buy-token. Then, a disjunctive programming formulation can be given as follows:

$$(1-z)\,\underline{p}_j \leq p^{s,0} \leq (1-z)\,\overline{p}_j \tag{12a}$$
$$(1-z)\,\underline{p}_k \leq p^{b,0} \leq (1-z)\,\overline{p}_k \tag{12b}$$
$$z\,\underline{p}_j \leq p^{s,1} \leq z\,\overline{p}_j \tag{12c}$$
$$z\,\underline{p}_k \leq p^{b,1} \leq z\,\overline{p}_k \tag{12d}$$
$$p_j = p^{s,0} + p^{s,1} \tag{12e}$$
$$p_k = p^{b,0} + p^{b,1} \tag{12f}$$
$$p^{s,1} \geq \pi\, p^{b,1} \tag{12g}$$
$$p^{s,0} \leq (\pi - \varepsilon)\, p^{b,0} \tag{12h}$$
$$v \leq \overline{y}\, p^{s,1} \tag{12i}$$
$$v \geq r^{\min}\, \overline{y}\, p^{s,1} \tag{12j}$$

The general idea is that both $p^{s,1}$ and $p^{b,1}$ shall take the true value of p_j and p_k within the respective bounds if $z = 1$ (i.e., the order can be executed), and be set to zero otherwise; and vice-versa for $p^{s,0}$ and $p^{b,0}$. This is modelled through constraints (12b)–(12c). The aggregation of the auxiliary price variables to the actual token prices is expressed in constraints (12f) and (12e). All the other constraints (12g)–(12j) represent the original model constraints from (10), now expressed in terms of the auxiliary variables.

Model Using either of the two reformulations of (10) for all orders as a substitution for (13c), the full MIP model can be stated as:

$$\text{maximize} \quad \sum_{i \in \mathcal{I}^o} v_i \tag{13a}$$

$$\text{subject to} \quad \sum_{i \in \mathcal{I}^o} t^b_{i,j}\, v_i = \sum_{i \in \mathcal{I}^o} t^s_{i,j}\, v_i \qquad \forall\, j \in \mathcal{I}^t \tag{13b}$$

$$\tilde{\mathfrak{C}}^1_i \vee \tilde{\mathfrak{C}}^0_i \qquad \forall\, i \in \mathcal{I}^o \tag{13c}$$

$$n = \sum_{j \in \mathcal{I}^t} \frac{1}{p^{\text{old}}_j} \cdot p_j, \tag{13d}$$

$$p_j \leq (1+\delta)\, \frac{p^{\text{old}}_j}{p^{\text{old}}_k}\, p_k \qquad \forall j,k \in \mathcal{I}^t \times \mathcal{I}^t \tag{13e}$$

$$v_i \in \mathbb{R}_{\geq 0} \qquad \forall\, i \in \mathcal{I}^o \tag{13f}$$

$$p_j \in \left[\frac{1}{1+\delta}\, p^{\text{old}}_j, (1+\delta)\, p^{\text{old}}_j\right] \qquad \forall\, j \in \mathcal{I}^t \tag{13g}$$

Additional inequalities The model does not take into account that there exist dependencies between orders on the same token pair as well as across adjacent token pairs. In the following, we will derive valid inequalities that capture some of these dependencies.

First, whenever a sell order with a certain limit price is executed (fully or partially) at some determined market price, all orders on the same token pair with a better (i.e., lower) limit price must also be executed. Conversely, if some order can not be executed, all orders with worse (i.e., higher) limit prices may not be executed as well. In particular, let $\omega_{i_1} = (j, k, \cdot, \pi_{i_1})$ and $\omega_{i_2} = (j, k, \cdot, \pi_{i_2})$ be two orders selling τ_j for τ_k with $\pi_{i_1} \leq \pi_{i_2}$. Then, we can relate the corresponding binary variables as $z_{i_1} \geq z_{i_2}$.

This idea induces a natural ordering of the orders on every token pair by their limit price. Let there be $N_{j,k}$ sell orders $(\omega_{i_1}, \omega_{i_2}, \ldots, \omega_{i_{N_{j,k}}})$ on token pair (τ_j, τ_k), i.e., orders that offer token τ_j for τ_k. Assuming that these orders are sorted increasingly by their limit prices

$$\pi_{i_1} \leq \pi_{i_2} \leq \ldots \leq \pi_{i_{N_{j,k}}},$$

this yields the chain of inequalities

$$z_{i_1} \geq z_{i_2} \geq \ldots \geq z_{i_{N_{j,k}}}. \tag{14}$$

Hence, there are only $N_{j,k} - 1$ non-redundant such inequalities per token pair and thus always less than N overall. Please note that if two orders ω_{i_1} and ω_{i_2} are given with equal limit prices $\pi_{i_1} = \pi_{i_2}$, the equality $z_{i_1} = z_{i_2}$ needs to hold.

Other types of valid inequalities can be derived when considering two orders going in the opposite direction, i.e., $\omega_{i_1} = (j, k, \cdot, \pi_{i_1})$ selling token τ_j for τ_k and $\omega_{i_2} = (k, j, \cdot, \pi_{i_2})$ vice-versa. Then, if $\pi_{i_1} \cdot \pi_{i_2} > 1$, we can conclude that ω_{i_1} and ω_{i_2} can not both be executed at same time, because their feasible price ranges are disjoint. Hence, we can impose

$$z_{i_1} + z_{i_2} \leq 1.$$

Conversely, $\pi_{i_1} \cdot \pi_{i_2} \leq 1$ would imply that every exchange rate between τ_j and τ_k allows executing at least one of the two orders, which yields

$$z_{i_1} + z_{i_2} \geq 1.$$

These two types of inequalities can also be generalized to chains of orders along cycles of tokens. Let $(\tau_{j_1}, \tau_{j_2}, \ldots, \tau_{j_m}, \tau_{j_1})$ be such a token cycle of length m, and let there be a set of orders $(\omega_{i_1} = (j_1, j_2, \cdot, \cdot, \pi_{i_1}), \omega_{i_2} = (j_2, j_3, \cdot, \cdot, \pi_{i_2}), \ldots, \omega_{i_m} = (j_m, j_1, \cdot, \cdot, \pi_{i_m}))$. Now, if $\prod_{l=1}^{m} \pi_{i_l} > 1$, our arbitrage-freeness requirement again forbids that all orders can be executed at the same time. Hence, we can write

$$\sum_{l=1}^{m} z_{i_l} \leq m - 1. \tag{15}$$

In the other case where $\prod_{l=1}^{m} \pi_{i_l} \leq 1$, we know that at least one of the orders needs to be allowed being executed, which yields

$$\sum_{l=1}^{m} z_{i_l} \geq 1. \tag{16}$$

In both cases (15) and (16), we can use the inequalities (14) in order to detect redundancies. However, even the numbers of non-redundant inequalities of these types grow significantly with the length of the token cycles that are being considered. For our practical purposes, we have incorporated inequalities for cycles of length $m = \{2, 3\}$ and observed remarkable performance gains.

Describe how exactly non-redundant inequalities are determined!?

Computational results!

3.3 Mixed-integer linear programming model II

Adjust to only use sell orders!

In the previous MIP model (13), we have modelled the constraints of every order separately, i.e., we have used one binary disjunctive formulation per order that controls its behavior in case it may or may not be executed. On top of that, we have used the equations (**??**) to indicate the dependencies between orders on the same token pair. However, we can also encode these dependencies deeper into the model by considering order volumes within certain price ranges on a token pair in a somewhat aggregated fashion. This approach gives rise to an alternative MIP formulation that we will present in the following.

As before, let $\mathcal{I}^p := \{(j,k) \in \mathcal{I}^t \times \mathcal{I}^t, j < k\}$ denote the index set of (undirected) pairs of tokens. We want to aggregate orders on every token pair, so we collect all buy and sell orders on (j,k), i.e., all $\omega = (j,k,\cdot,\cdot) \in \mathcal{O}$ and $\omega = (k,j,\cdot,\cdot) \in \mathcal{O}$, and sort them in an increasing order of their limit prices. Assuming that there are m orders for token pair (j,k), we get

$$\underline{p}_{j|k} =: \pi_{j,k}^{(0)} < \pi_{j,k}^{(1)} < \pi_{j,k}^{(2)} < \ldots < \pi_{j,k}^{(m)} < \pi_{j,k}^{(m+1)} := \overline{p}_{j|k}, \tag{17}$$

where $\underline{p}_{j|k}$ and $\overline{p}_{j|k}$ are explicit but somewhat arbitrary bounds on the exchange rate (e.g., half and double the previous rate). The above ordering (17) defines regions $r_l := [\pi^{(l)}, \pi^{(l+1)}]$, $l \in \{0 \ldots m\}$, for the exchange rate $p_{j|k}$. Notice that, without loss of generality and for simplicity of notation, we assume the limit prices of all orders to be pairwise different, so that the strict inequalities in (17) hold. If some limit prices were equal, we would only end up with less exchange rate regions. Let $\mathcal{I}^r_{j,k} := \{0 \ldots m\}$ denote the set of all such regions for every token pair (j,k), i.e., $l \in \mathcal{I}^r_{j,k}$ refers to the interval r_l of that token pair.

Now, let $\tilde{x}^{(1)}_{j,k,l}$ denote to the cumulated amount of tokens τ_j that could be bought across all buy orders $\omega = (j,k,\cdot,\cdot,\cdot) \in \mathcal{O}^b$ if the exchange rate $p_{j|k}$ were in r_l. Conversely, let $\tilde{x}^{(2)}_{j,k,l}$ denote the cumulated number of available tokens τ_k in buy orders $\omega = (k,j,\cdot,\cdot,\cdot) \in \mathcal{O}^b$. Both shall be stored in vectors $\tilde{\mathbf{x}}^{(1)} := (\tilde{x}^{(1)}_{j,k,l})$ and $\tilde{\mathbf{x}}^{(2)} := (\tilde{x}_{j,k,l})^{(2)}$. Analogously, we aggregate available sell tokens for sell orders $\omega = (j,k,\cdot,\cdot,\cdot) \in \mathcal{O}^s$ and $\omega = (k,j,\cdot,\cdot,\cdot) \in \mathcal{O}^s$, and store them in $\tilde{\mathbf{y}}^{(1)} := (\tilde{y}^{(1)}_{j,k,l})$ and $\tilde{\mathbf{y}}^{(2)} := (\tilde{y}^{(2)}_{j,k,l})$, respectively. From here on, we are only working with this aggregated order representation.

In terms of variables, we will use the same price variables $\mathbf{p} \in \mathbb{R}^n_{\geq 0}$ as previously. In addition, we will use variables $\mathbf{v}^A := (v^A_{j,k,l})$ and $\mathbf{v}^B := (v^B_{j,k,l})$ to represent trading volumes for every price interval r_l on every token pair (j,k). Moreover, we represent the total absolute and net trading volume on each token pair by $(v^{\text{abs}}_{j,k}) =: \mathbf{v}^{\text{abs}} \in \mathbb{R}^{|\mathcal{I}^p|}_{\geq 0}$ and $(v^{\text{net}}_{j,k}) =: \mathbf{v}^{\text{net}} \in \mathbb{R}^{|\mathcal{I}^p|}_{\geq 0}$.

If the exchange rate on a token pair (j,k) falls into region r_l, the following set of constraints needs to hold:

$$\tilde{\mathfrak{C}}_{j,k}^{(l)} := \left\{ \begin{array}{rcl} v_{j,k,l}^{A} & \geq & r^{\min}\left(\tilde{x}_{j,k,l}^{(1)}\, p_j + \tilde{y}_{j,k,l}^{(2)}\, p_k\right) \\ v_{j,k,l}^{A} & \leq & \tilde{x}_{j,k,l}^{(1)}\, p_j + \tilde{y}_{j,k,l}^{(2)}\, p_k \\ v_{j,k,l}^{B} & \geq & r^{\min}\left(\tilde{y}_{j,k,l}^{(1)}\, p_j + \tilde{x}_{j,k,l}^{(2)}\, p_k\right) \\ v_{j,k,l}^{B} & \leq & \tilde{y}_{j,k,l}^{(1)}\, p_j + \tilde{x}_{j,k,l}^{(2)}\, p_k \\ p_j & \geq & \pi_{j,k}^{(l)}\, p_k \\ p_j & \leq & \pi_{j,k}^{(l+1)} p_k \end{array} \right\} \tag{18}$$

On a token pair (j,k) with m regions r_l, the disjunction $\bigvee_{l\in\{0\ldots m\}} \tilde{\mathfrak{C}}_{j,k}^{(l)}$ needs to be satisfied.

Model Similarly as described in Section 3.2 for the first MIP model, (18) can be reformulated using either a big-M or a disjunctive programming approach. Without going into detail about this, the overall model reads as in (19).

The objective function (19a) maximizes the sum of the trading volumes over all trading pairs and is supposed to yield the same value as the objective (13a) of the previous MIP model.

The first constraint (19b) secures, again, the token balance for every token by requiring the net traded volumes to be balanced over all token pairs in which the respective token is involved.

$$\text{maximize} \quad \sum_{(j,k)\in\mathcal{I}^p} v_{j,k}^{\text{abs}} \tag{19a}$$

$$\text{subject to} \quad \sum_{\substack{k\in\mathcal{I}^t \\ k\neq j}} v_{k,j}^{\text{net}} = \sum_{\substack{k\in\mathcal{I}^t \\ k\neq j}} v_{j,k}^{\text{net}} \qquad \forall\, j \in \mathcal{I}^t \tag{19b}$$

$$v_{j,k}^{\text{abs}} = \sum_{l\in\tilde{\mathcal{I}}_{j,k}^r} \left(v_{j,k,l}^A + v_{j,k,l}^B\right) \qquad \forall\, (j,k) \in \mathcal{I}^p \tag{19c}$$

$$v_{j,k}^{\text{net}} = \sum_{l\in\tilde{\mathcal{I}}_{j,k}^r} \left(v_{j,k,l}^A - v_{j,k,l}^B\right) \qquad \forall\, (j,k) \in \mathcal{I}^p \tag{19d}$$

$$\bigvee_{l\in\{0\ldots m\}} \tilde{\mathfrak{C}}_{j,k}^{(l)} \qquad \forall\, (j,k) \in \mathcal{I}^p \tag{19e}$$

$$n = \sum_{j\in\mathcal{I}^t} \frac{1}{p_j^{\text{old}}}\, p_j, \tag{19f}$$

$$p_j \leq (1+\delta)\, \frac{p_j^{\text{old}}}{p_k^{\text{old}}}\, p_k \qquad \forall j,k \in \mathcal{I}^t \times \mathcal{I}^t \tag{19g}$$

$$p_j \in \left[\frac{1}{1+\delta}\, p_j^{\text{old}}, (1+\delta)\, p_j^{\text{old}}\right] \qquad \forall\, j \in \mathcal{I}^t \tag{19h}$$

$$v_{j,k}^{\text{abs}}, v_{j,k}^{\text{net}} \in \mathbb{R}_{\geq 0} \qquad \forall\, (j,k) \in \mathcal{I}^p \tag{19i}$$

$$v_{j,k,l}^A, v_{j,k,l}^B \in \mathbb{R}_{\geq 0} \qquad \forall\, (j,k) \in \mathcal{I}^p, l \in \tilde{\mathcal{I}}_{j,k}^r \tag{19j}$$

3.4 Computational comparison

In order to investigate the performance of our MIP models, we have conducted computational experiments for different numbers of tokens ($n \in \{5, 10, 20, 50\}$) and orders ($N \in \{100, 200, 500\}$). For every combination of n and N, we have generated 20 random instances, whereby the randomness reflects our expectation of somewhat realistic situations. In particular, we expect not all tokens to be equally important in terms of trading volume and, hence, to have varying numbers of orders on different token pairs. We used Gurobi 8.0.0 as MIP solver on an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz machine with 16Gb RAM and using 4 threads, and a timelimit set to 1800 seconds for every instance.

The value of the maximum fluctuation parameter has been set to $\delta = 0.1$, meaning that prices are allowed to change by at most 10% from the (randomly set) previous prices. Moreover, we used $r^{\min} = 0$, hence orders can be left untouched even if their limit price complies with the computed exchange rate. This guarantees that all our test instances are feasible.

The (geometric) means of the runtimes have been computed only with respect to the instances that could be solved to optimality before the timelimit. Conversely, the average optimality gap does not take solved instances into account.

In general, the results of our computational experiments as in Table 1 and Table 2 show that runtimes of the two MIP formulations sharply increase both with the number of tokens and the number of orders that are being considered. While all instances with no more than 200 orders could be solved relatively fast by all models, instances with 500 orders are much harder to solve, with some (or even most) of the instances running into our set timelimit. The MIP model I (13) generally performs better on our testsets. Interestingly, for this model the big-M reformulation seems to be superior, whereas for the MIP model II, running times are lower using the Disjunctive Programming reformulation.

Without further model improvements, instances with 50 tokens and 500 orders already seem to impose a limit to the tractable problem size. While this limit may be pushed towards somewhat bigger instances by making use of additional valid inequalities such as (14)–(16), it appears unlikely that the tractable problem size can be increased by some orders of magnitude. If larger problems are to be considered, we can think of the following heuristics/approximations:

- Aggregate orders with similar limit prices on every asset pair
- Optimize over subsets of assets separately and fix prices in overall problem

# orders		# assets 5	10	20	50
100	∅ runtime	0.15	0.20	0.24	0.50
	# timeouts	0	0	0	0
	– ∅ gap	-	-	-	-
200	∅ runtime	0.42	1.78	2.26	6.67
	# timeouts	0	0	0	0
	– ∅ gap	-	-	-	-
500	∅ runtime	3.46	158.66	190.70	896.63
	# timeouts	0	0	0	7
	– ∅ gap	-	-	-	4.23%

(a) Big-M reformulation (11).

# orders		# assets 5	10	20	50
100	∅ runtime	0.25	0.38	0.48	1.09
	# timeouts	0	0	0	0
	– ∅ gap	-	-	-	-
200	∅ runtime	0.92	3.34	4.08	12.28
	# timeouts	0	0	0	0
	– ∅ gap	-	-	-	-
500	∅ runtime	6.96	184.60	229.98	1398.84
	# timeouts	0	1	2	17
	– ∅ gap	-	3.25%	2.27%	4.85%

(b) Disjunctive programming reformulation (12).

Table 1: Computational results for MIP model I (13).

# orders		# assets 5	10	20	50
100	∅ runtime	0.44	0.65	0.54	0.57
	# timeouts	0	0	0	0
	– ∅ gap	-	-	-	-
200	∅ runtime	4.14	18.40	14.27	20.18
	# timeouts	0	0	0	0
	– ∅ gap	-	-	-	-
500	∅ runtime	647.98	646.57	857.77	-
	# timeouts	0	19	19	20
	– ∅ gap	-	8.19%	6.85%	8.12%

(a) Big-M reformulation.

# orders		# assets 5	10	20	50
100	∅ runtime	0.34	0.85	1.12	1.14
	# timeouts	0	0	0	0
	– ∅ gap	-	-	-	-
200	∅ runtime	1.18	8.81	10.64	25.50
	# timeouts	0	0	0	0
	– ∅ gap	-	-	-	-
500	∅ runtime	24.70	288.50	315.62	-
	# timeouts	0	3	6	20
	– ∅ gap	-	2.31%	2.66%	5.39%

(b) Disjunctive programming reformulation.

Table 2: Computational results for MIP model II (19).

4 Extensions

The problem can possibly be extended in various directions:

- Add accounts/balances to the model.
- Basket orders: Buy/sell a set of tokens for a set of other tokens at some limit price.
- Automated market makers: Instead of signaling discrete demand at a specific price with one order, those would allow to express continues demand function over a price range.
- if it becomes a problem to find a valid solution *at all*, the optimization problem can be broadened to allow violations of the current constraints but measure the violation and include this to the objective function.

References

[1] Egon Balas. Disjunctive Programming. *Annals of Discrete Mathematics*, 5:3–51, 1979.

[2] Pierre Bonami, Andrea Lodi, Andrea Tramontani, and Sven Wiese. On mathematical programming with indicator constraints. *Mathematical Programming*, 151(1):191–223, 2015.

[3] Eric Budish, Peter Cramton, and John Shim. The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response. *The Quarterly Journal of Economics*, 130(4):1547–1621, 2015.

[4] Richard Engelbrecht-Wiggans and Charles M. Kahn. Multi-Unit Auctions with Uniform Prices. *Economic Theory*, 12(2):227–258, 1998.