MIP clearing engine overview

Maciej Reich

November 19, 2019

This document is the draft documentation of the MIP approach to clearing orders.

1 Overview

The MIP approach simultaneously looks for clearing prices and a set of orders satisfying them, which clear fully against each other (i.e. there are no unmatched tokens), while guaranteeing maximal volume execution, and further best execution constraints. In particular, it does this in the presence of conditional orders.

1.1 Challenge

Clearing a single book is conceptually straightforward - regular exchanges do it all the time. The difference in the Nomisma set-up is the presence of Conditional Orders - orders on more than one token, which execute conditional on the net price of the tokens, and conditional on the presence of other orders to clear against. Suddenly, the clearing of one book depends on another — a conditional order may provide a competitively-priced fill, but only if its other leg is also filled in another book, etc.

Given prices, it is fairly straightforward to determine a set of clearing orders that maximises volume. Conversely, given a set of clearing orders, it is straightforward to find a price that satisfies them. What is difficult is to do both at the same time.

To overcome this difficulty, Mixed Integer (Linear) Programming is used.

1.2 Mixed Integer Programming

MIP is an optimisation technique, solving Linear Programming problems, but with the additional stipulation that some variables must be integers, or even just 0 or 1. Such binary variables can be used to encode logical decisions, such as "if price is greater than 100, this order cannot fill". This document: http://web.mit.edu/15.053/www/AMP-Chapter-09.pdf is a nice introduction to some of the possibilities this opens.

2 Formulating the clearing problem as MIP

At the most general way, the clearing problem can be specified as follows:

$$\text{Maximise} \sum_j x_j \tag{1}$$

$$\text{subject to} \sum_j w_j x_j = 0 \tag{2}$$

$$i\text{th order consistent with prices } p : w_j \cdot p \leq l_j \forall j \tag{3}$$

$$0 \leq x_j \leq X_j \tag{4}$$

where

1. $x_j \in \mathbb{R}, x \geq 0$ is the amount filled in the ith order
2. $w_j \in \mathbb{R}^n$ is a vector describing which of n possible tokens, and in what sign and proportion, are acquired, when a unit amount of the order is executed. For example, if $w_j = [1, 0.5, 0, -1]$, executing a unit amount of the order will give its owner 1 lot of token 1, 0.5 lot of token 2 and -1 lot of token 4, with nothing in token 3.
3. X_j is the overall size of the ith order, and the maximal executable amount.

Without condition (3), this would be a linear program, however we must also search over prices, and in turn eligibility of the orders for clearing.

Assuming for a moment we can solve this problem via some kind of iterative approach, we will find that for any price vector, there are various orders satisfying (3). An iterative solver would try to maximise the x_j filled amounts, subject to the clearing condition. But then it would also try to adjust the prices p, to come up with a better subset of orders. This sounds like computationally infeasible.

Fortunately, this kind of condition is readily expressible in the MIP framework. Consider the following inequality:

$$x - y - Mi \leq 0 \tag{5}$$

$$x, y \in \mathbb{R}, i \in \{0, 1\} \tag{6}$$

where M is a large parameter. If $i = 0$, the inequality reduces to $x \leq y$, but if $i = 1$ and M is sufficiently large, the inequality is automatically satisfied, and the condition specifies no relationship between x and y.

The problem above can thus be augmented as follows:

$$\text{Maximise} \sum_j x_j \tag{7}$$

$$\text{subject to} \sum_j w_j x_j = 0 \tag{8}$$

$$w_j \cdot p - l_j + A_j i_j \geq 0 \ \forall j \tag{9}$$

$$w_j \cdot p - l_j + B_j (1 - i_j) \leq 0 \ \forall j \tag{10}$$

$$i_j \in 0, 1 \ \forall j \tag{11}$$

$$0 \leq x_j \leq X_j \tag{12}$$

$$x_j \leq i_j X_j \tag{13}$$

Note that all inequalities are linear, so we satisfy the requirements of linearity. Further, conditions (9) and (10) imply that $w_j \cdot p \leq l_j \iff i_j = 1$. Indeed, for carefully chosen values oif A and B constrants, if $w_j \cdot p \leq l_j$, then (9) implies $i_j = 1$ (otherwise the inequality cannot be satisfied). Conversely, if $i_j = 1$, inequality (10) reduces to $w_j \cdot p - l_j \leq 0$. Finally, inequality (13) implies that if $i_j = 0$, the associated traded quantity x_j must also be 0.

Consequently, the following are true:

1. The optimisation program is a linear, mixed-integer program, and readily solvable with off-the-shelf solvers

2. The optimisation problem also encodes (part of) the Nomisma rules for clearing books: choose prices and orders satisfying them, such that the orders clear, and such that the overall crossed volume is maximised

3. Thanks to the formulation, the order selection and price determination happen automatically. The search over prices and order sets occurs inside the MIP solver, in a computationally-efficient way.

4. A regular linear program would not be sufficient here; if the variables i_j are allowed to take fractional values between 0 and 1, the logic encoding no longer works.

The problem is phrased slightly differently in the prototype, to enable other features as well, but the general idea is the same, and this is a good starting point for understanding the approach.

3 Full problem specification

In this section, I describe in full how the problem is specified in the solver. It is conceptually the same as the example above, but has some further features.

The clearing process consists of solving a sequence of closely related optimisation processes. This is because the clearing process in fact maximises a number of objectives in sequence: first maximise volume, then minimise surplus

subject to maximising volume, then maximise price surplus subject to keeping the other quantities constant. Finally, select a single price satisfying a final disambiguation criterion. The first three problems are MIP problems, the last one is a Quadratic Program (QP), with possibly simpler formulations being possible.

Overall, the problem is split logically as follows:

1. Order data is pre-processed
2. MIP problem is solved to maximise volume
3. MIP problem is solved to minimise surplus, subject to keeping volume constant
4. MIP problem is solved to maximise price surplus, subject to keeping volume and surplus constant
5. QP is solved to choose final price

3.1 Data pre-processing

The following steps are taken:

1. Orders are grouped by side, tokens, and relative weights of the tokens. The following are examples of orders that would get grouped:

   - All buy orders for a single token X
   - All conditional orders to buy 1 X, sell 1 Y
   - Conditional order to sell 1 X, buy 1 Y, and conditional order to sell 10 X, buy 10 Y

   If orders are on opposite sides, or acquire different proportions of tokens, they are not grouped. Each group has *weights* w associated with it, describing what tokens are being traded in the group, normalised so that $\sum_i |w_i| = 1$. For the above examples, the weights will be (1), $(0.5, -0.5)$, $(-0.5, 0.5)$.

   I refer to these associations as *groups*.

2. Within a group, I order and sub-group orders by price, from most aggressive to least aggressive. A sub-group of orders in the same group, and with the same price, is referred to as an *interval*. Each interval has a quantity q associated with it, which the maximal amount of tokens that can be filled at this price. The price condition is expressed as

   $$w \cdot p \leq l$$

   . Examples:

   - For a buy order for at most 100: $(1) \cdot (p) \leq 100$.
   - For a sell order for at least 150: $(-1) \cdot (p) \leq -150$.

- For a conditional order to buy 0.5 X, sell 0.5 Y, receive 10 premium: $(0.5, -0.5) \cdot (p_X, p_Y) \leq -10$

3. Quantities A and B are also calculated for each interval, to be defined later.

3.2 MIP problems

The three MIP problems share a common structure, and differ by objective function, and minor internal details.

3.2.1 Common setup

The common part of the setup goes as follows:

1. For kth token, introduce a price variable $p_k \in \mathbb{R}$ with an upper and lower bound, L_k and U_k

2. For ith group, generate a *quantity variable* $x_i \geq 0$. This denotes the overall number of real tokens filled in this group.

3. Introduce the clearing constraint. All tokens must clear fully. In other words
$$\sum_i x_i w_i = 0$$
where w_i are the *weights* of the ith group. This is a vector constraint.

4. For jth interval of the ith group, introduce a breakpoint variable $b_{i,j} \in \{0, 1\}$. This variable is associated with the orders belonging to the interval. If $b = 0$, no orders from this interval are executed.

   This gives rise to a number of consistency conditions:

   (a) Breakpoint variables must satisfy the inequality:
   $$b_{i,1} \geq b_{i,2} \geq b_{i,3} \geq \dots$$
   If for some j $b_{i,j} = 0, b_{i,j+1} = 1$, that implies orders in group j are all unfilled, but some orders in group $j + 1$ may be filled, which contradicts the price priority conditions.

   (b) If $b_{i,j} = 1$, the price must be consistent with the interval's price inequality. This is expressed with the following inequality:
   $$w_{i,j} \cdot p - B_{i,j}(1 - b_{i,j}) \leq l_{i,j}$$
   This way, if $b = 1$, this inequality is equivalent to $w_{i,j} \cdot p \leq l_{i,j}$, as defined for the interval. If $b = 0$, $B_{i,j}$ (defined below) is large enough that the inequality $w_{i,j} \cdot p - B_{i,j} \leq l_{i,j}$ is large enough that for any value of p within its bounds, the inequality is satisfied — i.e. the constraint becomes trivial.

(c) Similarly, if $b_{i,j} = 0$, we stipulate that the inequality must *not* be satisfied. This way we achieve that $b = 0 \iff$ interval inequality holds. This is done via:

$$w_{i,j} \cdot p + A_{i,j} b_{i,j} \geq 0$$

When $b = 0$, we get the opposite condition to the interval's price inequality. Meanwhile, if $b = 1$, the extra A term makes the inequality trivially satisfied.

These inequalities are at the core of the MIP matching engine logic. They create a structure of binary variables associated with intervals. I use them to constrain the filled quantities, calculate surplus and maximise price premium subsequently.

This is done as follows:

(a) Quantity constraining:

$$x_i \leq \sum_j b_{i,j} q_{i,j}$$

with $q_{i,j}$ being the interval quantities.

(b) Surplus computation: quantity available to trade is expressed as

$$\sum_{i,j} b_{i,j} q_{i,j}$$

. The difference between this and volume is the surplus.[1]

(c) Price premium is covered in later parts of this document.

5.

The above does not specify the objective functions; indeed, these vary depending on the type of problem being solved. When maximising volume, the above problem is taken, with an objective of maximising volume $= \sum_i x_i$. When surplus is being minimised, the surplus variable is being minimised, with the constraint that the previous value of volume must hold, etc. The common structure of the problem guarantees that any solution produced by the MIP solver satisfies the basic rules of clearing.

It may help to consider some stylised scenarios around point 4 of the above structure.

Maximising volume When maximising volume, the solver wishes to make the x_i variables large, subject to the clearing constraint. Since x_i is bound by $\sum_j q_{i,j} b_{i,j}$, increasing values of x_i force more and more $b_{i,j}$ values to set to 1. This in turn forces the movement of the price variable p via the set of inequalities in point 4 above.

[1] In fact, when surplus is being minimised, volume is being held constant, so it suffices to minimise the quantity of orders available to trade.

Minimising surplus . When volume is fixed, minimising surplus is equivalent to minimising $\sum_{i,j} q_{i,j} b_{i,j}$, i.e. setting as many $b_{i,j}$ to 0 as possible. This in turn adds increasingly strict inequalities to p, confining it to a smaller region of the solution.

3.2.2 Price premium

The final objective is maximising price premium. For an order with price defined by the inequality $w \cdot p \leq l, \sum_k |w_k| = 1$, *price premium* of an order is defined simply as

$$l - w \cdot p$$

It is clearly non-negative for fillable orders, and it is used for ordering fills by overall priority; orders with higher price premium have more aggressive prices and have priority when receiving fills, the same way as a more aggressively priced order has priority in a standard, single-instrument auction.

Priority optimisation schemes First, suppose each interval has priority $P_{i,j}$; we will use price premium to define it. But how is it even included in the optimisation in the first place?

Suppose you know the maximal volume and minimum surplus. Further, let $y_{i,j}$ be the amount filled specific to interval i, j. We then have

$$x_i = \sum_j y_{i,j}$$

Further, we can maximise order priority by maximising:

$$\sum_{i,j} P_{i,j} y_{i,j}$$

subject to maintaining the same volume and surplus. If two intervals, i, j and k, m are competing for the same fill, the one with the higher priority will get it.

Complications There are two complications: first, we have not defined the quantities $y_{i,j}$ in the problem, and second, the priorities P in our case depend on the prices, which seemingly invalidates the linear nature of the problem. An expression like $P \cdot y$ expands to $(l - w \cdot p) \cdot y$, leading to a second-order expression.

Vanishing price coefficients Fortunately, the price coefficients cancel out. We have from the clearing condition that

$$\sum_i w_i x_i = \sum_{i,j} w_i y_{i,j} = 0$$

therefore also

$$\begin{aligned}\sum_{i,j}(l_{i,j} - w_i \cdot p) \cdot y_{i,j} &= \sum_{i,j} l_{i,j} y_{i,j} - p \cdot \sum_{i,j} w_i y_{i,j} \\ &= \sum_{i,j} l_{i,j} y_{i,j} - p \cdot 0 \\ &= \sum_{i,j} l_{i,j} y_{i,j}\end{aligned}$$

which is again nicely linear. This means that in fact, price premium can be maximised without knowing the actual prices, it suffices to maximise this expression within the MIP problem framework.

One concern could be whether the final selection of orders is consistent with the prices — could it be that by ignoring the clearing prices in the optimisation, the final selection of orders violates the price conditions. However, by the structure imposed on the problem above, we know that this cannot happen; if an interval does not satisfy the prices, its indicator variable is set to 0 and the quantity for that interval is forced to be 0. Conversely, if a fill for an interval is positive, the indicator variable must be positive too, forcing the price to be consistent with it.

Interval fills The problem specification, for simplicity, did not include the $y_{i,j}$ variables, however they are easy to add. We add one for each interval and stipulate that

$$\sum_j y_{i,j} = x_i$$

Further, we need to ensure that fills are accrued in order; an interval can only receive fills if all of its predecessor intervals are fully filled.

This can be established by the following inequalities:

$$\begin{aligned}y_{i,j} &\leq q_{i,j} b_{i,j} \\ y_{i,j} &\geq q_{i,j} b_{i,j+1}\end{aligned}$$

This way, if an interval's b variable is 0, its associated fill is also 0. If the next interval's b variable is 1, then the current interval must be fully filled. The variable is only free if $b_{i,j} = 1$ and $b_{i,j+1} = 0$.

3.2.3 Optimisation objectives

The previous section outlines the structure of the problem, it also needs some optimisation objectives.

Three problems are solved in sequence:

Maximise volume Optimisation objective is $\max \sum_i x_i$

Minimise surplus Optimisation objective is $\min \sum_{i,j} b_{i,j} q_{i,j}$ while keeping volume constant.

Maximise price premium Optimisation objective is $\max \sum_{i,j} y_{i,j} l_{i,j}$

3.3 Final price determination

Solving the three rounds of MIP problems yields final fills, and various constraints on prices. It does not necessarily narrow down on a unique price; the simplest example would be a market with two orders:

- Buy X for 150
- Sell X for 100

In single-instrument auctions, typically the price is chosen to minimise distance from a previous price.[2] Here, this is more complicated, since there are potentially conditional orders setting multivariate bounds on order prices.

The analogy therefore is to minimise distance between new and previous price over all instruments simultaneously, subject to constraints from the last MIP stage.

3.3.1 Constraints

At this stage, we know which intervals have fills, and which ones do not. We know the orders with fills must satisfy the final price, and the orders without fills should not satifsy the price, to minimise price surplus.

More specifically, if for some i, j $b_{i,j} = 1$ then $w_i \cdot p \leq l_{i,j}$ and if $b_{i,j} = 0$ then in fact $w_i \cdot p > l_{i,j}$. The b variables are now fixed, so there is no need to solve this with a MIP solver.

3.3.2 Objective

The natural distance function would be the ℓ_1 metric:

$$\min \sum_k |p_k^{\text{prev}} - p_k|$$

However, it may not be sufficient to disambiguate prices in this context. It is likely that conditional orders will introduce conditions like $p_x - p_Y \leq C$ for some tokens X, Y and some constant C; then there may be different valid prices p with the same objective value; this is demonstrated on figure 1 on page 10.

For this reason, I instead propose to use the ℓ_2 metric:

$$\min \sum_k (p_k^{\text{prev}} - p_k)^2$$

While admittedly the ℓ_2 metric does not have an intuitive meaning in this context, its properties should make the final price much less ambiguous.

[2] Plus some fairly arbitrary condition if there is no last price, for example choosing minimal price, or mid-price of possible values

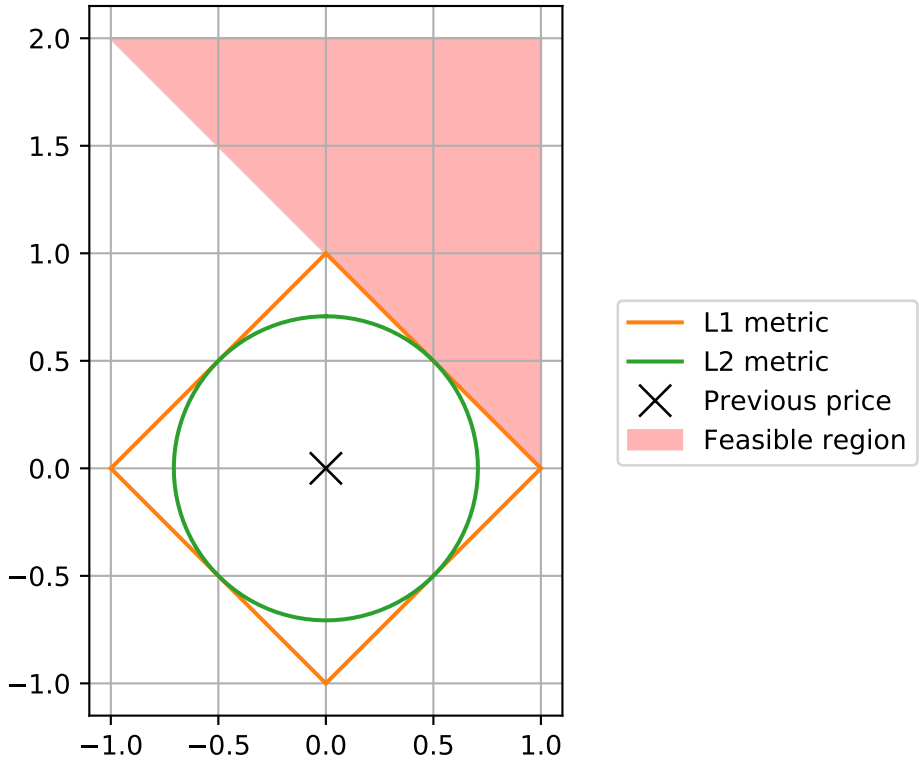

Figure 1: L1 and L2 metrics comparison. The square and circle represents points equidistant from previous price according to L1 and L2 metrics. The L1 metric yields many equidistant points in the feasible region, whereas the L2 metric only returns 1.

3.4 Quadratic solver

This problem now needs to be solved using a quadratic solver; one good option is OSQP. For numerical stability, it is good to express p as

$$p = p^{\text{prev}} + \delta p$$

then convert the inequalities on p into ones on δp and minimise $\ell_2(\delta p)$.

4 Replication

A Computing constants A, B

Still to come

B Volume counting

Still to come