NMMS clearing algorithm

Documentation v1.0

(release 1.8.2)

optimeering

Document Information

Title	NMMS clearing algorithm - documentation
Version	1.0
Authors	Optimeering AS
Publication date	19.01.2024
Status	Confidential

Optimeering AS

Contact details	Universitetsgata 10 0164 Oslo, Norway
Web	www.optimeering.com

Disclaimer

Heading Level 1

Contents

1 Introduction

This document describes the NMMS bid selector and pricing algorithm developed by Optimeering, currently designed to clear the Nordic aFRR and mFRR capacity markets. As the code itself is well documented, this document focuses on the markets, with rules and design, how we have structured the problem, how the optimization problems are defined, input/output specifications as well as the overall structure of the python code and API.

2 Problem structure

The NMMS algorithm is currently designed to clear and calculate a price for the Nordic reserve markets aFRR and mFRR.

The algorithm consists of four main steps illustrated in the flow chart in Figure 1 below:

- Step 1: CZC allocation
- Step 2: Bid selection
- Step 3: Approximate Pricing
- Step 4: Clearing Prices

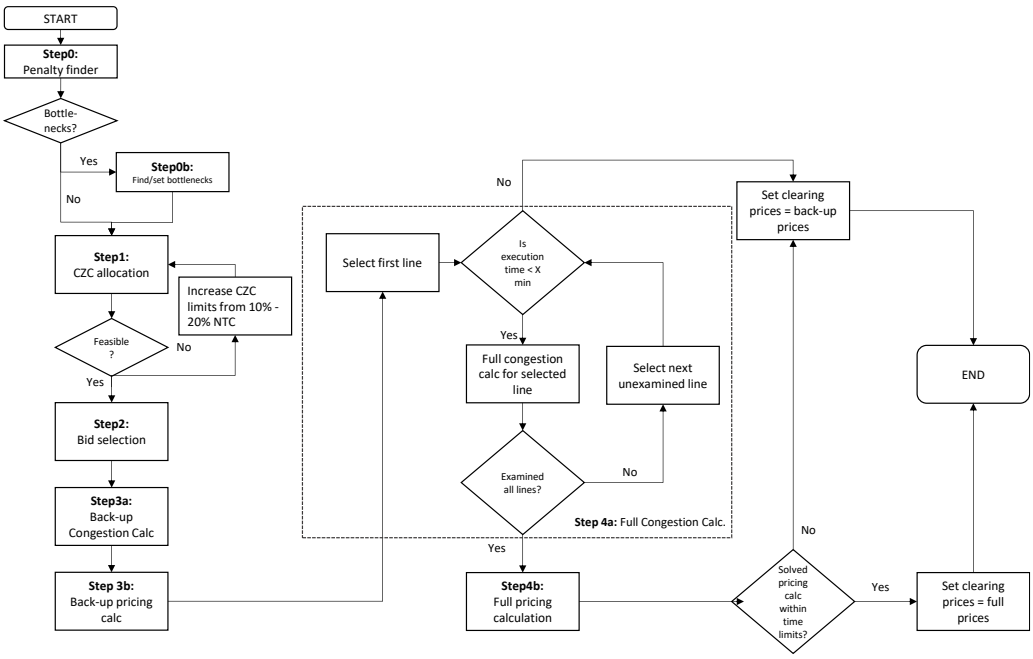


Figure 1 – Flow chart of the algorithm

As can be seen from the figure above, the algorithm also has a Step0 where we find the penalties needed to have a feasible solution as well as pre-defining bid values for bottleneck groups. These values will be fixed for steps 1 and 2.

In addition to this, the CZC allocation step (Step 1) consists of a fallback CZC allocation step where the algorithm automatically increases the CZC limits if the solution is not feasible. If the solution is feasible, the

algorithm will also run a minimize flow optimization if one or more lines have 0 CZC cost.

The pricing steps 3 and 4 can be further split in two, one congestion and one pricing calculation. The two pricing steps have the same pricing algorithm, but have different approaches to finding congestions.

In the following, we will go through the flow chart described above step-by-step and describe the goals, rules and how the various steps are linked together.

2.1 Step 0 – Penalty and bottleneck Finder

To avoid situations where the objective values in steps 1 and 2 are dominated by high penalty values, we have a Step0 where we find the values for the penalty variables that makes the problem feasible. In this step we also find the maximum accepted bid volumes for bids within bottleneck groups.

As described in 2.2, the algorithm has soft constraints with penalty variables to make sure results are provided to the user also in auctions with scarcity (i.e. too little reserve capacity and or CZC capacity). We have three of these types of constraints:

- Bidding zone demand
- Minimum reservation for bidding zones
- Minimum reservation for macro areas

This step will find the total for each hour and product direction (up/down) and fix these in steps 1 and 2. For a 24-hour auction we will find and store 24 (hours) x 2 (up/down) x 3 (types) values.

The optimization problem is similar to the problems in Step1 and Step2, but with CZC capacities set to the maximum level (typically 20% of NTC).

When these values have been found, they are fixed in the later steps and removed from the objective function. The reason why we want to take the penalty variables out of the objective function when we find CZC reservations and bid selection is that as the penalty values increase, the effort the algorithm uses to find the best allocation of CZC, and the selection of bids gets lower. In other words – high penalty values in the objective function will reduce the optimality gap very fast and in some cases below the optimality gap target set by the user even before the optimal selection of bids and allocation of CZC is found.

The other part of step0 is to find an upper limit of volume selected for bids within bottleneck groups. A bottleneck group consists of one or more bids where the sum of accepted volumes can be restricted to a maximum value. This constraint is only used in this Step0 and the volume for the bids in the following steps will be set according to the solution from this step.

For more information see the next sub chapters.

2.2 Step 1 & 2 – CZC allocation / Bid Selection

In this section we will describe Steps 1a, 1b and 2. These steps all use the same algorithm, but with minor changes in input and constraints.

Goal:

For each hour of the day, direction (up/down) and region (bidding zone), meet the demand in MW for reserve capacity at the lowest possible cost. This includes minimizing the sum of all bids *i* and cost of CZC allocation:

- $\boldsymbol{bid\_cost_i \cdot volume\_selected_i}$
- $\boldsymbol{czc\_cost_{from,to,hour} \cdot czc\_res_{from,to,hour}}$

For Step2, the CZC cost element above is not included in the objective function.

- Bid_cost (EUR/MW) is the cost associated with each bid given as input. Volume_selected (MW) is determined by the optimization algorithm for each bid and must be between the minimum and maximum volume for each bid defined in the input.
- For CZC (cross-zonal-capacities), the cost associated with each connection (from – to) and hour is given by input as *czc_cost*. The *czc_res* is a variable in MW determined by the algorithm and is the maximum of the UP and the opposite direction DOWN regulation reservation for CZC on the line. This is further described later in this Chapter.

Scope:

The market is cleared by the algorithm for a specific day and can handle any number of hours, i.e. days with daylight saving (23 or 25 hours) will be taken care of as long as the input from the user specifies it. Demand is specified per region (bidding zone), direction (up or down) and hour.

Bids can be linked in time and with other bids, as well as being part of exclusive bid groups. This is further described later in this Chapter.

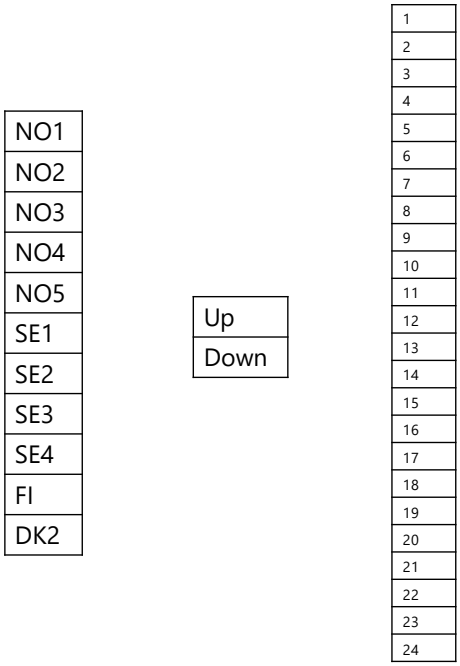


Figure 2 – The three dimensions of reserve requirements / demand

Rules:

1. All demand in each region (bidding zone), hour and direction must be met either by local or imported reserves. If not available, the problem is relaxed with a high penalty cost to the objective function (relaxation is not used in Step 1b and 2)
2. Each region can have restrictions on minimum and maximum locally reserved volume. If not possible to fulfill minimum constraint, the problem is relaxed with a high penalty cost to the objective function.
3. Each macro area can have restrictions on minimum and maximum locally reserved volume. If not possible to fulfill minimum constraint, the problem is relaxed with a high penalty cost to the objective function.
4. Bids can only be selected in MW quantity steps (this number can be changed by user input)
5. Between all regions there is a CZC which represents the maximum amount of volume to be transferred between the regions. If no CZC is specified, the algorithm will interpret it as 0

6. Each bid must be selected with a volume between its minimum and maximum volume constraint. If the minimum value is not defined, minimum volume will be equal to maximum volume.
7. Bids can be defined for one or consecutive (block bid) hours
8. Linked bids must either be selected (> minVolume) together or not at all
9. Bids can be defined together in exclusive groups. Only one (or two linked bids) bid in an exclusive bids group can be selected (> minVolume)
10. Fix the penalty values found in Step0
11. Bids can have restrictions on minimum resting time and maximum activation time
12. Total volume selected for bids within a Bottleneck Group can be restricted to a maximum value (hourly level)

Table 1 shows which rules/constraints are included in the different steps.

Table 1 – Constraints included in the various steps

Step	1	2	3	4	5	6	7	8	9	10	11	12
0	x	x	x	x	x	x	x	x	x		x	x
1	x	x	x	x	x	x	x	x	x	x	x	
2	x			x	x	x	x	x	x	x	x	

The only step that does not use constraints 1-9 is the bid selection step. In the bid selection step, we do not include rules on minimum and maximum reservation for bidding zones (2) and macro areas (3). From the table we can also see how the penalty values are fixed with constraint 10 after these values are found in Step0. In addition to this, rule number 5 is changed from CZC allocation less than or equal to limit to must be equal to the CZC allocation for the min reservation step. For the minimum reservation step, there is additional changes made to the rules for bids which are described in 2.2.1 below.

Regional specifications:

Bids are specified on an Elspot area level (NO1, NO2…, SE1, SE2), referred to as bidding zone. Macro areas are combinations of bidding zones. Each macro area can have one or more bidding zones, and a bidding zone can be part of one or several macro areas as shown below.

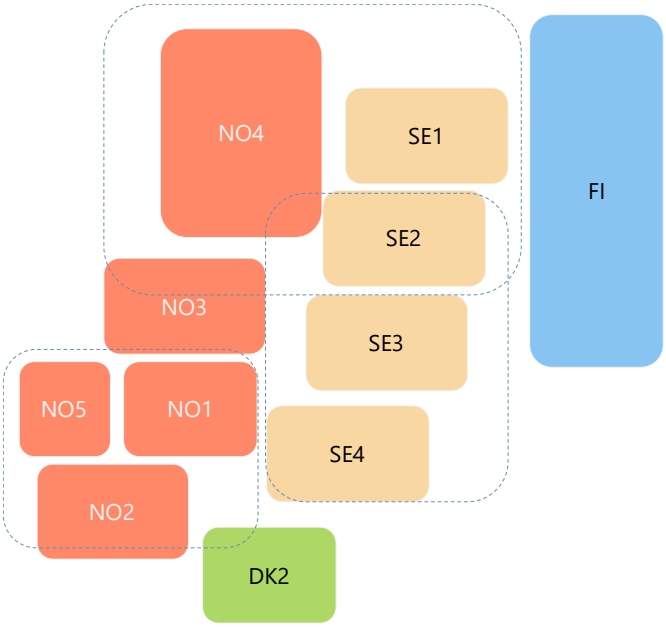


Figure 3 – Elspot and macro region definitions

As shown in the table below, demand, bids and cross-border-capacity (CZC) are specified on a bidding zone level, as well as minimum and maximum activated reserves constraints. Macro areas can have constraints for minimum and maximum activated reserves within the area.

Table 2 – Input types vs. region definitions

Input types	Bidding Zones	Macro Areas
Bids	X	
Demand	X	
CZC	X	
Minimum reservation	X	X
Maximum reservation	X	X

Bids:

Bids are specified with the following parameters in the bids-object in the JSON input file:

- Id (int)
- biddingZone (int)
- direction (int), 1 = up, 2 = down

- minVolume (int) (MW)
- block (true/false)
- duration (int)
- restingTime (int)
- price (float) (EUR/MW)
- points: list with hour and volume for 1 or more hours (int / int)

Each bid will have a unique *ID*. This ID is used as identification in the bid linking and exclusive bids groups (see below), as well as in the output.

Each bid is defined with a *biddingZone* and a *direction*. Further, each bid will have a *price*, as well as a *minVolume* that can be between 0 and the maximum *volume* in the points list. An indivisible bid will have *minVolume* = *volume*. If *minVolume* is not specified (None), the algorithm will set *minVolume* = *volume*.

A bid can as described above be selected between the *minVolume* and the *volume*, but only in volume steps determined by the *quantityFactor* specified in the input JSON config object. This is as default set to 5, which means that all bids must be selected in 5 MW steps.

A bid that is available for several hours can have restrictions on minimum *resting time* (consecutive hours not selected) after going from selected to not selected. It can also have restriction on maximum *activation time*, which is the maximum number of consecutive hours it can be selected for if selected in an hour. These types of bids are referred to as limited bids.

The points list includes one or more hour and volume pairs which tells the algorithm for which hour and at what volume it can be selected for. If the block value is TRUE, all or none hours must be selected.

CZC:

CZC's are defined in MW from and to biddingZones for each hour in the transferCapacitites JSON object. The value limits the amount of volume to be transferred from and to a region for each hour. If a connection in an hour is not defined it will be interpreted as 0 MW capacity. The CZC capacity is by default set to 10% of the Net Transfer Capacity (NTC), and must be greater or equal to the maximum of the up and the opposite down reservation as described by the following example for reserved CZC capacity (CZC_res) between SE1 and SE2 in hour 1.

- CZC_cap(SE1, SE2) = 100 MW
- CZC_res(SE1, SE2) = max{flow(SE1, SE2, UP), flow(SE2, SE1, DOWN)

- CZC_res(SE1, SE2) <= 100MW

Using the terminology from the above example, the cost of transferring (reserving) volume over CZC is determined by *CZC_res* variable multiplied with a cost per MW given by the input for each combination of fromRegion, toRegion and hour. In other words, the cost of reserving CZC capacity is a cost per MW multiplied with the maximum of the UP and the opposite DOWN reservation for a given line.

Linking (inclusive) and exclusive groups:

Two or more bid ID's can be linked together. By linking we mean that either both are selected or none. By selected we mean that the volume for each of the linked bids are between the *minVolume* and *volume (*if the *minVolume* is set to 0, a bid can be selected with volume 0). To link two or several bids together, we define a linkedBids list in the JSON input which for each of the n links, we have two bids defined by their Id. E.g.

[

[1, 241]

[2, 242]

[3, 243]

]

In this list, bids 1 and 241 are linked, as well as 2 and 242, and 3 and 243.

Further illustrated through the example below:

- E.g. bid_a and bid_b from table to the right
- Possible solutions:
- [a,b]
  - 0,0
  - 5,0
  - 5,10
  - 5,15
  - 10,0
  - 10,10
  - 10,15

	bid_a	bid_b
Price:	20	31
Direction:	Up	Down
MinVol:	5	0
MaxVol:	10	15

Figure 4 – Linked bids example

If two non-block bid ID's are linked together, the algorithm will link the individual hours of the two bids.

We also have the possibility to define a number of exclusive groups where only one of the bids included can be selected in the optimization (if two linked bids are included in the group, both can be selected). This is also a list with the *Id* of each bid representing the bids.

E.g. [1,5] are linked bids (1 and 5 are bidID's) and the exclusive bids list look like this:

- [1,5,7,9,10,11]

All of these are valid solutions (1 = selected, 0 = not selected):

- [1,1,0,0,0,0]
- [0,0,1,0,0,0]
- [0,0,0,1,0,0]
- [0,0,0,0,1,0]
- [0,0,0,0,0,1]
- [0,0,0,0,0,0]

If two non-block bid ID's are in an exclusive group, the algorithm will make several groups one for each hour of the non-block bid.

Technical linked groups:

Two or more limited bids with the same activation and resting time constraints can be linked together in 'technical linked groups'. When bids are linked this way, the max activation and minimum resting time constraints are moved from the individual bid level to a bid group level. If bid *i* and *j* are in a technical linked group and they have a resting time of 3 hours, both bids must be non-selected for at least three consecutive hours starting from the hour one or both bids go from selected to not selected. The same logic is applied on the maximum activation constraint.

In the figure below, we show how this works for five bids and two technical linked groups (A and B). The second table is an acceptable solution, the first table is the acceptable solution with technical linked groups, the second table is not.

					Hours								
bidID	TechLink	Price	Activation	RestingTime	1	2	3	4	5	6	7	8	9
a	A	1	3	2	10	10	15	15					
b	A	5	3	2				15	15	10	10		
c	A	10	3	2							10	10	10
d	B	2	5	3	5	5	5	5	5	5			
e	B	6	5	3					5	5	5	5	5
				Tot	5	15	20	35	5	0	20	10	15

					Hours								
bidID	TechLink	Price	Activation	RestingTime	1	2	3	4	5	6	7	8	9
a	A	1	3	2	10	10	15	15					
b	A	5	3	2				15	15	10	10		
c	A	10	3	2							10	10	10
d	B	2	5	3	5	5	5	5	5	5			
e	B	6	5	3					5	5	5	5	5
				Tot	15	15	20	5	25	15	25	15	15

Soft vs. hard constraints:

The market rules described above are implemented in the algorithm through a set of constraints. These can either be 'hard' (must be met) or 'soft' (should be met). If constraints are hard, they can cause the problem to be infeasible (not able to find a solution), and the algorithm will not return a solution. If the constraints are soft, we will be able to find a solution, but some of the rules above are 'relaxed' in order to find the solution.

As we can see from the flow chart in Figure 1, we will only move on from Step1a to Step2 when we have a feasible solution. The soft constraints described in this section will because of this be hard only for Step1a and soft for Step0, Step1b and Step2.

The algorithm is designed with the following main constraints:

Table 3 – Soft and hard constraints in the algorithm Step1a

Constraint	Description	Infeasible if	Hard/soft
Minimum Regulation	Must select at least X MW from bidding zone	It is not enough bids to cover the minimum regulation	Soft – penalty high
Maximum regulation	Must select at most X MW from bidding zone	It is not possible to satisfy the demand without violating the maximum production limits	Hard
Demand	Must select or import at least X	The demand in the area cannot be covered.	Soft – Penalty low

	MW from bidding zone		
Macro areas min	Must select at least X MW from macro area	It is not enough bids to cover the minimum regulation of a macro area	Soft – Penalty high
Macro areas max	Must select at most X MW from macro area	It is not possible to satisfy the demand without violating the maximum reserve limits	Hard
CZC max	Not reserve more than Y MW on line		Hard
Bid characteristics	Linked, Exclusive, 5MW		Hard

As seen from the table above, the algorithm is designed with three set of soft constraint types:

- Bidding zone demand
- Minimum reservation for bidding zones
- Minimum reservation for macro areas

The soft constraints cause the algorithm to still give feasible results even though the bid selection problem is infeasible. By infeasible we mean that the problem is not able to find a solution given the constraints and the data from the inputs. Examples can be that the demand in a region is greater than the available volumes both locally and from neighboring regions. With a soft constraint we allow the algorithm to set a value to the 'missing' volume at a very high cost, and still give a solution.

Hard constraint:

- *Sum(bids) + imports – exports >= demand*

Soft constraint:

- *Sum(bids) + imports – exports >= demand – penalty*

In order to tell the algorithm that using the penalty is the last option, we add a very high cost to the objective function if the penalty is used (only Step0). Since the bid selector is a minimization problem, it will avoid using the penalty if possible. The three constraints in the list above all have individual penalty cost parameters specified in the input JSON. As default, the bidding zone demand constraint have a lower penalty than the two minimum reservation constraints.

If a problem is infeasible, but can find a solution given the penalty variables, the problem will get a status OPT_RELAX and information on the 'missing' volume is written in lists in the output (see Chapter 5).

2.2.1 Minimize CZC reservation

If a daily auction has CZC between regions, but one or more of the costs of reserving is 0, the algorithm can return solutions with loop flow. The balancing capacity can in such cases be 'sent' in loop and/or reserve more CZC than needed because the amount of reserved volume on connections with 0 cost will not increase to the objective function that we are minimizing.

The avoid this problem the algorithm will first complete the normal CZC allocation (Step 1), then do a second optimization where we minimize total reserved CZC, with all bids and penalty variables fixed as well as CZC variables for connections with cost > 0 fixed to the solution from Step 1. In other words, we will reserve as little CZC as possible given the selected bids.

By doing this we avoid the problems described above. The algorithm will in other words always choose the solution that takes the shortest path and at the same time do not reserve more CZC capacity than is needed for the accepted bids in Step1a.

The objective for this post optimization is to minimize the total CZC reservation over the day.

2.3 Step 3 Approximate Congestion and Pricing

Step 3 consists of approximate congestion and pricing calculation. This sub chapter will only focus on the congestion calculation (Step3a). The pricing algorithm is described in 2.5). Since a full congestion calculation (Step4) may take long to finish, the algorithm calculates approximate congestions to be used as backup.

The backup congestion calculation goes through each connection and checks if it in Step1 has allocated volume ($f_{r\hat{r},UP,t}$) to the limit or not.

- For each$f_{r\hat{r},UP,t}$:
  - If $0 < f_{r\hat{r},UP,t} < CZC_{r\hat{r}t}$ then:

- $f_{r\hat{r},UP,t}$ is uncongested
- Else:
- $f_{r\hat{r},UP,t}$ is congested

- For each $f_{\hat{r},r,DOWN,t}$:
  - If $0 < f_{\hat{r},r,DOWN,t} < CZC_{r\hat{r}t}$ then:
    - $f_{\hat{r},r,DOWN,t}$ is uncongested
    - Else:
    - $f_{\hat{r},r,DOWN,t}$ is congested

The output used in the pricing algorithm will be a list of lines, potential export and import regions, direction (up/down) and hour like shown below.

Line	Export	Import	Direction	Hour	Cong.?
1	1	2	1	1	False
1	2	1	2	1	True
3	1	4	1	1	True
.	.	.	.	.	.
..	..	..	..	..	..

Each connection (line) between bidding zones will have 24x2 rows in the table, one for each hour and direction (up/down). As calculated output, we will include export and import bidding zones and if its congested or not.

2.4 Step4 – Full congestion calculation and Clearing Prices

This sub chapter will only focus on the congestion calculation (Step4a). The pricing algorithm is described in 2.5). For the full congestion

calculation, we will do *n* number of re-optimalizations of Step2 (bid selection) where *n* is the number of connections (from, to)

- For each n, we remove the CZC constraint and set the CZC cost on the line to minimum markup (0.1 EUR/MW) on the connection in question and,
- check if the CZC reservation values increase (for both up and down) and the total objective value is reduced

If the algorithm chooses to increase the CZC allocation for UP from bidding zone A to bidding zone B and the objective value is reduced compared to Step2 selection, the line is congested from A to B in UP. The same goes for DOWN regulation.

Output table from this step is a list of border directions for each up and down and hour.

Export	Import	Direction	Hour	Cong.?
1	2	1	1	False
1	2	2	1	True
2	1	1	1	True
.	.	.	.	.
..	..	..	..	..

If one of the *n* re-optimalizations reach the maximum solution time (set to 1 minute), Step3c will stop.

2.5 Pricing algorithm

From the flow chart in Figure 1 we see that the pricing algorithm is started either once or twice depending on the solution times in the full calculation algorithm. If the full congestion calculation is able to finish, the pricing algorithm is run twice (3b and 4b). However, the pricing algorithm is the same for 4b and 4b.

The pricing algorithm follows a two-step approach:

1. Find total payout
2. Determine prices

The pricing rules are the same for both steps and are as follows:

- Prices in all bidding zones in an uncongested area will get prices according to the following rules:
    - For uncongested line(a->b) and reserved_cap(a->b) == 0: P(a)>=P(b)
    - For uncongested line(a->b) and reserved_cap(a->b) >= 0: P(a)==P(b)
- Prices in bidding zones on each side of a congested line can be different according to the following rules:
    - For congested line (a->b) and reserved_cap(a->b) > 0: P(b)>=p(a)
- All selected bids must be in-the-money - If the price found in 2) does not cover the cost of accepted complex (block and linked) bids, the price is raised to a level such that none of these bids are out of the money. The "cost" of a block/linked bid may not be its bid price, but is assessed over the whole block/pair, given recovery in the other block hours/directions.
- We allow paradoxically rejected bids
- Prices in bidding zones that are not connected (capacity == 0) will not be affected by each other

The difference between the two steps is mainly the objective function. In the first step we minimize total payout, i.e., price in each bidding zone, hour and direction multiplied with the total procured capacity. In the second step we minimize the difference (absolute value) between price and bid costs for all selected bids. The reason why we do this second step is to avoid having very high prices in some hours and very low prices in others due to the block bid specific rule stated above. For the second step we add a condition that the total payout must be equal to the total payout found in the first step.

The following example illustrates the reason for the two-step approach.

BID_ID	PRICE	BLOCK	1	2	3
1	50	TRUE		10	10

If a block bid with price 50 is the only selected bid in hours 2 and 3 (see table above), the price can be set in multiple ways according to the rules above. A block bid does not have to be in-the-money for the individual hours, but it's sufficient that it is in-the-money over all hours in the block. Because of this we can have a price of 50 in both hours or we can have extreme cases like 0 in hour 2 and 100 in hour 3. For the price to send the correct signals to the market, we want the price to be spread as evenly as possible over the block. With the second step where we fix the total payout and minimize the difference between prices and bid costs, we achieve this. With a price of 50 in both hours, the absolute difference will be 0. With a price of 0 and 100, the difference will be 100.

3 Step1/2 Optimization problem

The optimization problem is a MILP with a minimize cost objective.

Below, we describe the most important variables, constraints, as well as the objective function. Steps 1a and 1b are equal, apart from the objective function which minimizes total CZC reserved and the fact that we fix all bid variables and penalty variables.

3.1 Mathematical formulation

Sets:

$\boldsymbol{BIDS}$	Set of all bids	$\boldsymbol{i}$
$\boldsymbol{BIDS^D}$	Set of all divisible bids $BIDS^D \subseteq BIDS$	i
$\boldsymbol{BIDS^I}$	Set of all indivisible bids, $BIDS^I \subseteq BIDS$	i
$\boldsymbol{T}$	Set of all hours	t
$\boldsymbol{D}$	Set of all auction types (up, down)	d
$\boldsymbol{R}$	Set of all regions	$r, \hat{r}$
$\boldsymbol{R^{IMP}}$	Set of all import regions $Rimp \subseteq R$	$r, \hat{r}$
$\boldsymbol{R^{EXP}}$	Set of all export regions $Rexp \subseteq R$	$r, \hat{r}$
$\boldsymbol{M}$	Set of all macro regions	m
$\boldsymbol{LINK}$	Set of all linked bids couples	L
$\boldsymbol{EXCL}$	Set of all exclusive groups	K

Variables:

$\boldsymbol{x_i}$	sales bid variable (integer), $i \in BIDS^D$
$\boldsymbol{x\_bin_i}$	binary bid variable (0 if not selected, 1 if selected), $i \in BIDS$

$f\_agg_{r\hat{r}t}$	flow variable (in MW) from region *r* to region $\hat{r}$ in hour *t*
$f_{r\hat{r}dt}$	flow variable (in MW) from region *r* to region $\hat{r}$ in hour *t*, in direction *d*
p_dem_{rdt}	penalty variable (in MW) for demand constraint, for region *r* t in hour *t*, in direction *d*
p_minreg_{rdt}	penalty variable (in MW) for minimum reservation constraint, for region *r,* in hour *t*, in direction *d*
p_minmac_{mdt}	penalty variable (in MW) for minimum reservation constraint, for macro region *m,* in hour *t*, in direction *d*

Parameters:

Dem_{rdt}	demand for reserves for region *r*, direction *d*, hour *t*
bid_cost_i	: cost of selecting 1MW of bid *i*
$minvolume_i$	minimum volume to be selected for each bid *i*
$maxvolume_i$	maximum volume to be selected for each bid *i*
$hourFrom_i$	the first hour bid *i* is valid from
$hourTo_i$	the first hour after hourFrom bid *i* is not valid (i.e. if a bid has hourFrom = 4 and hourTo = 6, the bid is valid in hour 4 and 5)
$flow\_cost_{r\hat{r}t}$	cost of reserving 1MW on the CZC from region *r* to region $\hat{r}$
$CZC_{r\hat{r}t}$	maximum reservation capacity on the CZC from region *r* to region $\hat{r}$
$minReg_{rdt}$	minimum volume selected for region *r*, direction *d*, hour *t*
$maxReg_{rdt}$	maximum volume selected for region *r*, direction *d*, hour *t*
$minMac_{mdt}$	minimum volume selected for macro region *m*, direction *d*, hour *t*

$\boldsymbol{maxMac_{mdt}}$	maximum volume selected for macro region *m*, direction *d*, hour *t*
$\boldsymbol{pencost\_dem_{rdt}}$	penalty cost per MW for demand for region *r*, direction *d*, hour *t*
$\boldsymbol{pencost\_minreg_{rdt}}$	penalty cost per MW for minimum regulation for region *r*, direction *d*, hour *t*
$\boldsymbol{pencost\_minmac_{mdt}}$	penalty cost per MW for minimum regulation for macro region *m*, direction *d*, hour *t*

Constants:

$\boldsymbol{iQuant}$	constant that defines the bid step size used. As default set to 5

Objective function:

The objective function minimizes total bid costs for all bids *i,* and all CZC reservation costs from region *r* to region $\hat{r}$ for hour *t*. For Step0, we also include all penalty costs for region *r,* direction *d* and hour *t*.

$$
\begin{aligned}
min \sum_{i \in BIDS^D} & x_i \cdot iQuant \cdot bid_{cost_i} \cdot (hourTo_i - hourFrom_i) \\
& + \sum_{i \in BIDS^I} x\_bin_i \cdot maxvolume_i \cdot bid_{cost_i} \\
& \cdot (hourTo_i - hourFrom_i) \\
& + \sum_r \sum_d \sum_t p_{dem_{rdt}} \cdot pencost_{dem_{rdt}} \\
& + \sum_r \sum_d \sum_t p_{minreg_{rdt}} \cdot pencost_{minreg_{rdt}} \\
& + \sum_m \sum_d \sum_t p_{minmac_{mdt}} \cdot pencost_{minmac_{mdt}} \\
& + \sum_r \sum_{\hat{r}} \sum_t f\_agg_{r\hat{r}t} \cdot flow\_cost_{r\hat{r}t}
\end{aligned}
$$

Constraints:

Demand constraint for region *r,* direction *d,* and hour *t*:

$$\sum_{\substack{i \in BIDS^D \\ if\ i_{region}=r \\ i_{hour}=t \\ i_{direction}=d}} x_i \cdot iQuant + \sum_{\substack{i \in BIDS^I \\ if\ i_{region}=r \\ i_{hour}=t \\ i_{direction}=d}} x\_bin_i \cdot maxvolume_i + \sum_{\hat{r}} f_{\hat{r}rdt} - \sum_{\hat{r}} f_{r\hat{r}dt} \geq Dem_{rdt} - p\_dem_{rdt}, \qquad \forall r, d, t$$

CZC constraint from region r to region $\hat{r}$ in hour *t*:

$$f\_agg_{r\hat{r}t} \leq CZC_{r\hat{r}t} \quad \forall\, r, \hat{r}, t$$

CZC constraints for aggregate flow variables from region *r* to region $\hat{r}$ in hour *t*:

$$f\_agg_{r\hat{r}t} \geq f_{r\hat{r},UP,t} \quad \forall\, r, \hat{r}, t$$

$$f\_agg_{r\hat{r}t} \geq f_{\hat{r},r,DOWN,t} \quad \forall\, r, \hat{r}, t$$

Min and max reservation constraints per region *r,* direction *d* and hour *t*:

$$\sum_{\substack{i \in BIDS^D \\ if\ i_{region}=r \\ i_{hour}=t \\ i_{direction}=d}} x_i \cdot iQuant + \sum_{\substack{i \in BIDS^I \\ if\ i_{region}=r \\ i_{hour}=t \\ i_{direction}=d}} x\_bin_i \cdot maxvolume_i \geq minReg_{rdt} - p\_minreg_{rdt}, \qquad \forall r, d, t$$

$$\sum_{\substack{i \in BIDS^D \\ if\ i_{region}=r \\ i_{hour}=t \\ i_{direction}=d}} x_i \cdot iQuant + \sum_{\substack{i \in BIDS^I \\ if\ i_{region}=r \\ i_{hour}=t \\ i_{direction}=d}} x\_bin_i \cdot maxvolume_i \leq maxReg_{rdt}, \qquad \forall r, d, t$$

Min and max reservation constraints per macro region *m,* direction *d* and hour *t*:

$$\sum_{\substack{i \in BIDS^D \\ if\ i_{macro}=m \\ i_{hour}=t \\ i_{direction}=d}} x_i \cdot iQuant + \sum_{\substack{i \in BIDS^I \\ if\ i_{macro}=m \\ i_{hour}=t \\ i_{direction}=d}} x\_bin_i \cdot maxvolume_i \geq minMac_{mdt} - p\_minmac_{mdt}, \qquad \forall m, d, t$$

$$\sum_{\substack{i \in BIDS^D \\ if\ i_{macro}=z \\ i_{hour}=t \\ i_{direction}=d}} x_i \cdot iQuant + \sum_{\substack{i \in BIDS^I \\ if\ i_{macro}=m \\ i_{hour}=t \\ i_{direction}=d}} x\_bin_i \cdot maxvolume_i \leq maxMac_{zdt}, \qquad \forall z, d, t$$

Minimum and maximum bid volume for each divisible bid *i*:

$$x_i \cdot iQuant \geq x_{bin_i} \cdot minvolume_i, \qquad \forall\, i \in BIDS^D$$

$$x_i \cdot iQuant \leq x_{bin_i} \cdot maxvolume_i, \qquad \forall\, i \in BIDS^D$$

Fixing penalty variables (Step1 and Step2):

$$\sum_r p_{dem_{rdt}} = STEP0PDEM_{dt} \qquad \forall\, t \in T, d \in D$$

$$\sum_r p_{minreg_{rdt}} = STEP0PMINREG_{dt} \qquad \forall\, t \in T, d \in D$$

$$\sum_r p_{minmac_{mdt}} = STEP0PMINREGMAC_{dt} \qquad \forall\, t \in T, d \in D$$

Bid linking constraints for all bid link couples *L* in linked set $LINK$:

$$x\_bin_{L1} = x\_bin_{L2}, \qquad \forall\, L \in BIDS^L$$

Exclusive group constraints for all *K* groups in exclusive group set $EXCL$

$$\sum_{\substack{i \in K \\ if\ i\ in\ K\ is\ in\ BIDS^L}} x\_bin_i \cdot 0.5 + \sum_{\substack{i \in K \\ if\ i\ in\ K\ is\ not\ in\ BIDS^L}} x\_bin_i \leq 1 \qquad \forall\, K \in EXCL$$

Demand constraint to add cuts to the problem for direction *d* and hour *t*:

$$\sum_{\substack{i \in BIDS^D \\ i_{hour}=t \\ i_{direction}=d}} x_i \cdot iQuant + \sum_{\substack{i \in BIDS^I \\ i_{hour}=t \\ i_{direction}=d}} x\_bin_i \cdot maxvolume_i \geq \sum_r Dem_{rdt} - p\_dem_{rdt}, \qquad \forall d, t$$

Maximum volume for bids within Bottleneck Groups (only Step0):

$$\sum_{\substack{i \in BOTTLENECKBIDS_b^D \\ i_{hour}=t}} x_i \cdot iQuant + \sum_{\substack{i \in BOTTLENECKBIDS_b^I \\ i_{hour}=t}} x\_bin_i \cdot maxvolume_i \leq maxVol_{b,t} \qquad \forall b \in BOTTLENECK, t$$

3.2 Comments to the optimization problem

- As bids can have minimum volume constraints, the bid selection problem is what is called a combinatorial optimization problem or an Integer Program. Bids may also be linked across time and with other bids, as well as be included in exclusive groups. The linking and non-divisibility of bids is a critical and fundamental characteristic of the bid selection problem that makes a traditional selection method of sorting the bids by price not a feasible approach to guarantee the most socioeconomic solution. The next section describes the how the algorithm is designed in order to be able to solve complex problems. Given the size of the problem, the algorithm will not always be able to find the optimal solution with the time limit set by the user. To be able to control the solver and get solutions that are close to optimal, the algorithm is designed with a set of control parameters (see chapter 7).
- In MILP problems, a set of inputs can have more than one optimal solution, i.e. two different bid selections and CZC reservations can give the exact same total cost. In these cases, Xpress solver by default will return the same solution if the user run the same problem several times. In order to control this behavior and to make sure that selecting among two equally good solutions is a random process, the algorithm is designed with a random seed value that controls this (see chapter 7).

3.3 Speeding up the algorithm

The clearing problem is a mixed integer linear problem (MILP). The difficulty of a Mixed-Linear Integer Programming (MILP) problem is

generally very hard to measure because it depends on too many factors. One way of approaching this is by a trial and error approach.

The reasons that we need to introduce integer variables are because of the following rules/constraints:

- Bid step sizes must be in integer steps
- Minimum volume constraints per bid
- Indivisibility for bids
- Block bids
- Linking
- Exclusive bids group

The number of these factors which are active in a clearing problem combined with the size of the problem (number of bids, regions, hours etc.) drives the complexity of the problem.

The algorithm is optimized with regards to solution quality and time. Three design features should be mentioned:

- Increasing the effort of finding subproblems
- Adding valid inequalities (cuts)
- Fixing variables

These changes came as a result of a mini-analysis on the various aFRR auction types. Test cases and results can be found in Appendix A.

Increasing the effort of finding subproblems

For the clearing problem, we can in some cases divide the auction into several subproblems. Because of the way CZC flow rules are designed, block bids, linking and exclusive groups, we are not able, to split the problem in 48 subproblems (24 hours, 2 directions), but in most cases we have seen, the aFRR auction could be split into more than one problem. In order to take advantage of this, we have increased the effort in which the solver detects subproblems in the presolving part of the optimization.

Adding cuts to the problem

In order to speed up the algorithm (find or prove optimal solution at low solution times) we define two types of valid inequalities (cuts). This is one of the most common techniques when models are struggling to close the optimality gap. An efficient cut does not remove any feasible solutions but makes the LP relaxation strictly more constraint.

Reserve requirements cut

Reserve requirements for aFRR is defined by bidding zone, hour and direction. In addition to this constraint, we add the following valid inequality:

$$\sum_{bids\ i\ for\ d,t} x_i \cdot quantity_{factor_i} \geq \sum_{r} demand \qquad \forall\ hours, directions$$

It tells the algorithm that the sum of all bids for the hour and direction must be greater than or equal to the total demand in the Nordics for the hour and direction (sum over all bidding zones *r* in the above equation). This cut works very efficiently when the total sum of demand is not in the bid quantity factor step size, e.g. if bid quantity step size is 5 MW and the total demand in an hour and a direction is 301 MW.

Cheapest bids first

In order to help the algorithm find the optimal solution faster and avoid illogical bid selection, we add cuts that says that if two bids are equal except the bid cost, the cheapest one should be selected before the more expensive one and that it should be selected 100% (if divisible). The following example shows its usefulness:

If you have defined a 0.0001 acceptable optimality gap, we say that a solution which is 0.01% form the best bound is an acceptable solution. In theory a problem with an objective value (total cost of an aFRR clearing) of 300 000 EUR, will stop if the solution is at most 30 EUR from the best theoretical solution. In theory you could end up choosing bids that are equal, but very close in price. E.g. if you have two bids of 5 MW and the first cost 1 EUR/MW and the other 2 EUR/MW, the difference between selecting these two will be 5 EUR which is well within the 30 EUR tolerance lever. So, to avoid these solutions, the cut defined above will help.

Fixing variables and reformulating

In order to speed up the solver, we suggest reformulating the problem by changing the treatment of indivisible bids. This helps fixing a lot of variables and reduce solution times for problems with many indivisible bids.

Originally, each bid has two variables – one integer variable that controls the amount of volume selected, and a binary variable that is 1 if the bid is selected and 0 if not. As indivisible bids can only be selected with one volume or none, we can remove all integer variables for these bids and

use the binary variable to both determine the amount of volume and whether the bid is selected or not.

4 Step3 Optimization problem

The optimization problem is an LP with a minimize cost objective.

4.1 Mathematical formulation

Sets:

$\boldsymbol{BIDS}$	Set of all bids	$\boldsymbol{i}$
$\boldsymbol{BIDS^L}$	Set of all linked bids $BIDS^L \subseteq BIDS$	i
$\boldsymbol{T}$	Set of all hours	t
$\boldsymbol{D}$	Set of all auction types (up, down)	d
$\boldsymbol{R}$	Set of all bidding zones	r
$\boldsymbol{PRICES}$	Set of all prices	p
$\boldsymbol{CONGESTIONLIST}$	Set of all rows in list from congestion calc	c
$\boldsymbol{LINK}$	Set of all linked bids couples	L

Variables:

$\boldsymbol{p_{r,d,t}}$	price variable (float) for bidding zone r, direction d and hour t

Parameters:

$\boldsymbol{bid_{volres_i}}$	selected volume for bid *i* from Step2
$\boldsymbol{bid_{cost_i}}$	cost of selecting 1MW of bid *i*
$\boldsymbol{hourFrom_i}$	the first hour bid *i* is valid from

$\boldsymbol{hourTo_i}$	the first hour after hourFrom bid *i* is not valid (i.e. if a bid has hourFrom = 4 and hourTo = 6, the bid is valid in hour 4 and 5)
$\boldsymbol{biddingzone_i}$	bidding zone for bid *i*
$\boldsymbol{direction_i}$	direction for bid *I* (1=up, 2=down)
$\boldsymbol{exp_c}$	exporting bidding zone for row c in congestion list
$\boldsymbol{imp_c}$	importing bidding zone for row c in congestion list
$\boldsymbol{direction_c}$	direction for row c in congestion list (1=up, 2=down)
$\boldsymbol{hour_c}$	hour for row c in congestion list
$\boldsymbol{congestion_c}$	true/false if line, direction and hour is congested or not

Objective function:

The objective function minimizes total payout over all bids *i*. As payout is volume selected multiplied with the price for each bid, we summarize over all selected bids and the prices associated with them.

$$\min c = \sum_{i,r,d,t} (price_{r,d,t} \cdot bid_{volres_i})$$

$$where\ biddingzone_i = r, direction_i = d, hourTo_i > t, hourFrom_i \leq t$$

Constraints:

All bids that are not in a linked pair should be profitable over all available hours

$$\sum_t (price_{r,d,t} \cdot bid_{volres_i}) \geq bid_{volres_i} \cdot bid_{cost_i} \cdot (toHour - fromHour) \quad \forall\ i\ not\ in\ BIDS^L$$

$$where\ biddingzone_i = r, direction_i = d, hourTo_i > t, hourFrom_i \leq t$$

Prices in unconstrained areas (neighboring) should be equal

$$price_{exp_c, direction_c, hour_c} = price_{imp_c, direction_c, hour_c} \qquad \forall\ c \in CONGESTIONLIST\ if\ congestion_c\ is\ false$$

Price in an import region must be larger or equal to the price in the exporting region if the connection between the import and export region is constrained

$$price_{exp_c,direction_c,hour_c} \leq price_{imp_c,direction_c,hour_c} \qquad \forall\, c \in CONGESTIONLIST\ if\ congestion_c\ is\ true$$

A linked bid pair [*i1, i2*] must be profitable over the sum of the two bids

$$price_{r1,d1,t1} \cdot bid_{volres_{i1}} + price_{r2,d2,t2} \cdot bid_{volres_{i2}} \geq bid_{volres_{i1}} \cdot bid_{cost_{i1}} + bid_{volres_{i2}} \cdot bid_{cost_{i2}} \qquad \forall\, i \in BIDS^L$$

5 Input requirements

Inputs are defined in a json-file used when running the algorithm. The structure and naming of objects cannot change without changes to the algorithm.

5.1 Control parameters

Each json-file have id and config objects. Within config we have parameters that controls the behavior of the algorithm and setup of dataobjects within the algorithm. Parameters contains control parameters for the Xpress solver (max time and allowed optimality gap). The control parameters are further described in chapter 7.


```
{
 "id": "65f95491-0e4d-4c51-b783-6f7e881b2363",
 "config": {
  "numberOfHours":       24,
  "numberOfBiddingZones": 3,
  "numberOfMacroAreas":   2,
  "maxProcessingTime":    600,
  "optimalityGap":        0.001,
  "gapIncrement":         0.00001,
  "gapInterval":          30,
  "randomSeed":           518412,
  "quantityFactor":       5,
  "penaltyRelaxDemandBiddingZone": 9999,
  "penaltyRelaxProdMinBiddingZone": 9999,
  "penaltyRelaxProdMinMacroArea": 9999
 },
```


5.2 Regional definition

numberOfBiddingZones and numberOfMacroAreas from the config object defines the size of the bidding zone and macro area sets.

In the biddingZonesMacroAreas, the link between bidding zones and macro areas is defined:

```
"biddingZonesMacroAreas": [
  {
   "biddingZone":      1,
   "macroArea":        1
  }, {
   "biddingZone":      1,
   "macroArea":        2
  }, {
   "biddingZone":      2,
   "macroArea":        2
  }, {
   "biddingZone":      3,
   "macroArea":        2
  }
 ],
```

5.3 Bids

For each bid there is a unique id and information on hours, bidding zone, direction, resting time, duration time, block bids, minimum volume, price, and a points list with hour and volume pairs.

Information on bids is found in the bids object:

```
{
   "id":              "KZ755A",
   "biddingZone":      1,
   "direction":        2,
   "minVolume":        0,
   "block":           false,
   "duration":         5,
   "restingTime":      3,
   "price":            7,
   "points": [
    {"hour": 7, "volume": 23},
    {"hour": 8, "volume": 23},
    {"hour": 9, "volume": 23}
   ]
```

5.4 Demand

Demand (in MW) is specified in the biddingZoneDemands object for each biddingZone, direction and hour:

```
"biddingZoneDemands": [
  {
   "hour":            1,
   "biddingZone":      1,
   "direction":       1,
   "demand":          80,
   "prodMin":         20,
   "prodMax":         120
  }, {
```

As well as demand, we can specify a minimum and maximum reservation volume for the bidding zone, direction and hour (prodMin and prodMax).

5.5 Macro Area constraints

As for bidding zones above, we can in macroAreaDemands specify a minimum and maximum reservation volume for the macro area, direction and hour (prodMin and prodMax).

```
"macroAreaDemands": [
  {
   "hour":            1,
   "macroArea":       1,
   "direction":       1,
   "prodMin":         20,
   "prodMax":         120
  }, {
```

5.6 CZC

Cross-border-capacities are defined in the transferCapacities-object between bidding zones:

```
"transferCapacities": [
  {
   "hour":            1,
   "biddingZoneIn":    1,
   "biddingZoneOut":   2,
   "capacity":        107,
   "cost":           4.35
  }, {
```

In this object we also find information on the cost (EUR/MW) for reserving CZC from bidding zone, to bidding zone for each hour.

5.7 Linked Bids

Defined in linkedBids:

```
"linkedBids": [
  [2, 3],
  [4, 5]
 ],
```

This is a list containing the ID of bids linked together.

5.8 Exclusive Bids

Defined in exclusiveBids:

```
"exclusiveBids": [
  [1, 2],
  [1, 5, 6]
 ]
```

This is a list containing the ID of bids that are in exclusive bids together.

5.9 Mapping and names

Names of bidding zones, macro areas and directions can be found in the three objects biddingZones, macroAreas and directions:


```
"biddingZones": [
  {
   "id":          1,
   "name":          "NO1"
  }, {
   "id":          2,
   "name":          "NO2"
  }, {
   "id":          3,
   "name":          "SE1"
  }
 ],

 "macroAreas": [
  {
   "id":          1,
   "name":          "South Norway"
  }, {
   "id":          2,
   "name":          "South of cut 2"
  }
 ],

 "directions": [
  {
   "id":          1,
   "name":          "Up"
  }, {
   "id":          2,
   "name":          "Down"
  }
 ],
```

6 Output specifications

6.1 Status and general information

{
"id": ‘asdf5481g-3h5hgb’,
"status": "OPT",
"log": ["RunId 1 has been cleared.", "Optimal solution has been found."],
"function": "clearmarket",
"objectiveFunctionValue": 35936.7,
"totalCost": 35936.7,
"optimalityGap": 0.0,
"optimizationDuration": 3892.3921,

- The id is the id given in the json input file.
- status can take the following values:

Return code	Textual description	Solution type	Time out?	Relaxing constraints?
OPT	Optimal solution	Optimal	No	No
OPT_RELAX	Optimal solution (after relaxing)	Optimal	No	Yes
SUB	Suboptimal solution (timeout)	Suboptimal	Yes	No
SUB_RELAX	Subotimal solution (timeout after relaxing)	Suboptimal	Yes	Yes
TIMEOUT	No solution (timeout)	No	Yes	-
FAIL	Failed	No	No	-

- function is the name of the python function running the bid selection problem
- objectiveFunctionValue is the value of the objective function from the problem
- totalCost is the sum of bid cost and CZC reservation costs as output from the problem. If the problem is not relaxed, this value equals the objectiveFunctionValue
- optimalityGap is the difference from the best bound found by the branch and bound algorithm

- optimizationDuration is the time in milliseconds that the branch and bound tree used to find the solution

6.2 Bid output

For each ID we report the acceptedVolume (in MW) as shown below:


```
"bids": [
  {
   "id": "IM61P4",
   "points": [
    {"hour": 1, "acceptedVolume": 20},
    {"hour": 2, "acceptedVolume": 25},
    {"hour": 7, "acceptedVolume": 23},
    {"hour": 8, "acceptedVolume": 20},
    {"hour": 9, "acceptedVolume": 15}
   ]
  }, {
   "id": "KZ755A",
   "points": [
    {"hour": 7, "acceptedVolume": 15},
    {"hour": 8, "acceptedVolume": 5},
    {"hour": 9, "acceptedVolume": 15}
   ]
  }
 ],
```


6.3 CZC output

For CZC output, we report how much MW is traded on all connections which have > 0 MW trade. The algorithm reports both for up and down regulation.

'transferCapacities': [{'biddingZoneIn': 2, 'biddingZoneOut': 1, 'direction': 1, 'hour': 1, 'reservedCapacity': 19.0},

6.4 Bidding zone and macro area results


```
"biddingZoneResults": [
 {
  "hour":            1,
  "biddingZone":      1,
  "direction":        1,
  "clearingPrice":    18.35,
  "approximatePrice":      18.29,
  "demandRelaxed":    80,
  "prodMinRelaxed":   20
 }
```


For each bidding zone (and hour and direction) we get the output as written above. Two prices, one clearingPrice which is the price payed to the BSO's, and one signal price which represents the 'correct' theoretical price to give signals to the market players.

The two soft constraints for bidding zones are captured in the same output, under demandRelaxed and prodMinRelaxed. For each relaxed bidding zone, direction and hour, we output the 'new' right hand side constraint value, i.e. the original requirement minus the relaxed volume for both the demand constraint (demand) and the minimum reservation constraint (prodMin). Both values are in MW.

For macro area minimum reservation constraint, we have similar output in MW in a separate list (macroAreaDemandsRelaxed):


```
"macroAreaResults": [
  {
   "hour":            1,
   "macroArea":       1,
   "direction":       1,
   "prodMinRelaxed":  20
  }
```

7 Python structure

7.1 Flask API

The Flask API is defined in the file /web.py.

The available endpoints are:

[POST] /solve

Used to start a new run, requires a request body (“application/json”) on the format described in the Input requirements chapter (0).

Waits until a solution is found before returning data:

```
{
        "id":           ...,
        "input":        ...,
        "result":       ...,
        "time":     ...,
}
```

[POST] /terminate

Used to terminate a given ongoing run, requires a request body (“application/json”) on the following format:

```
{
        "id":           <str>,
}
```

A call to the terminate endpoint will abort the run as fast as possible and return preliminary results if available.

[GET] /heartbeat

Check that everything is running.

[GET] /checklicense

Get information about the Xpress license.

[GET] /get_status

Returns the ids of the currently ongoing runs, along with information on the status of each particular run:

```
{
    "GAP":          <float>,
    "RUNTIME":          <float>,
    "UPPERBOUND":          < float>,
}
```

7.2 Xpress model

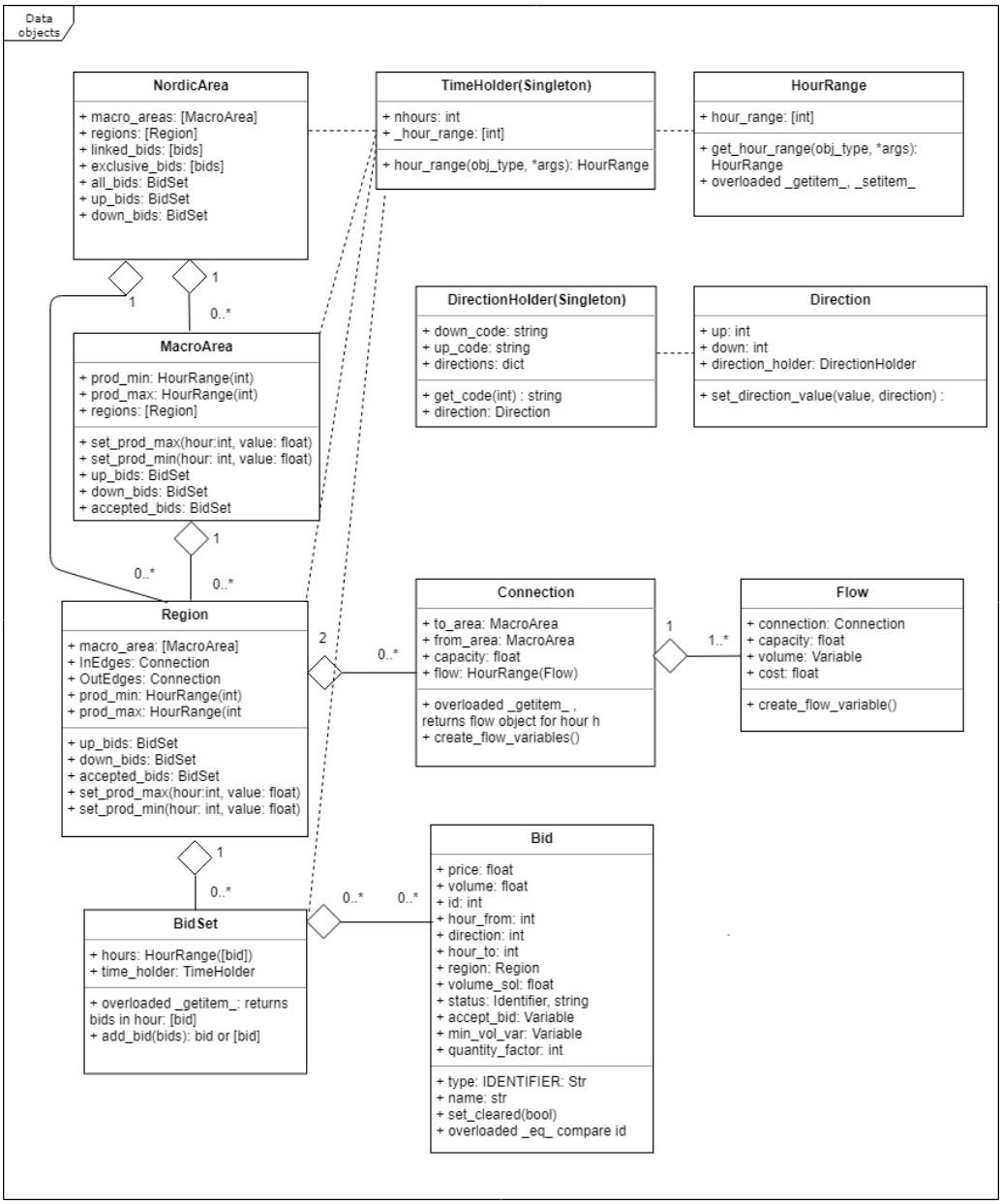
All functions and classes in the various modules have comments that describes the purpose of the functions and classes. Inputs and return values from functions are also described if defined. The table below gives an overview over each module and its content.

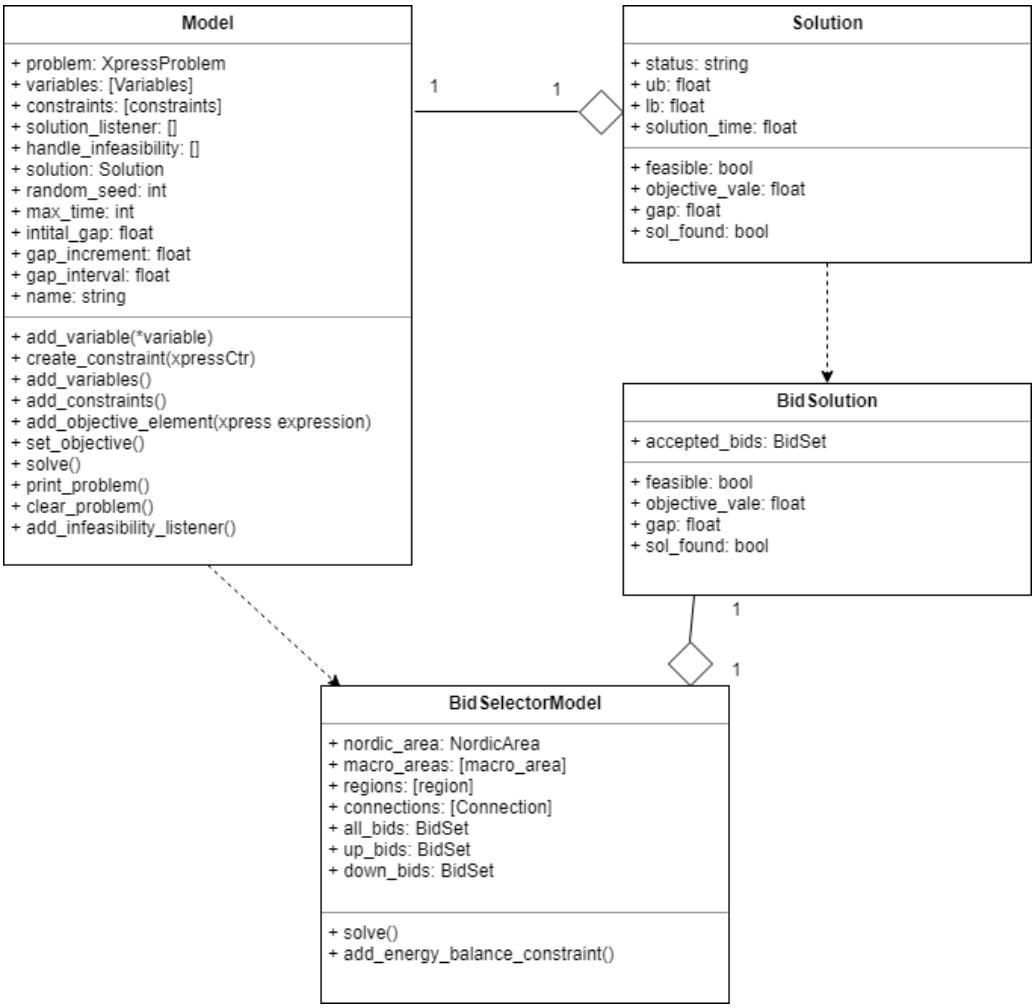
Modules

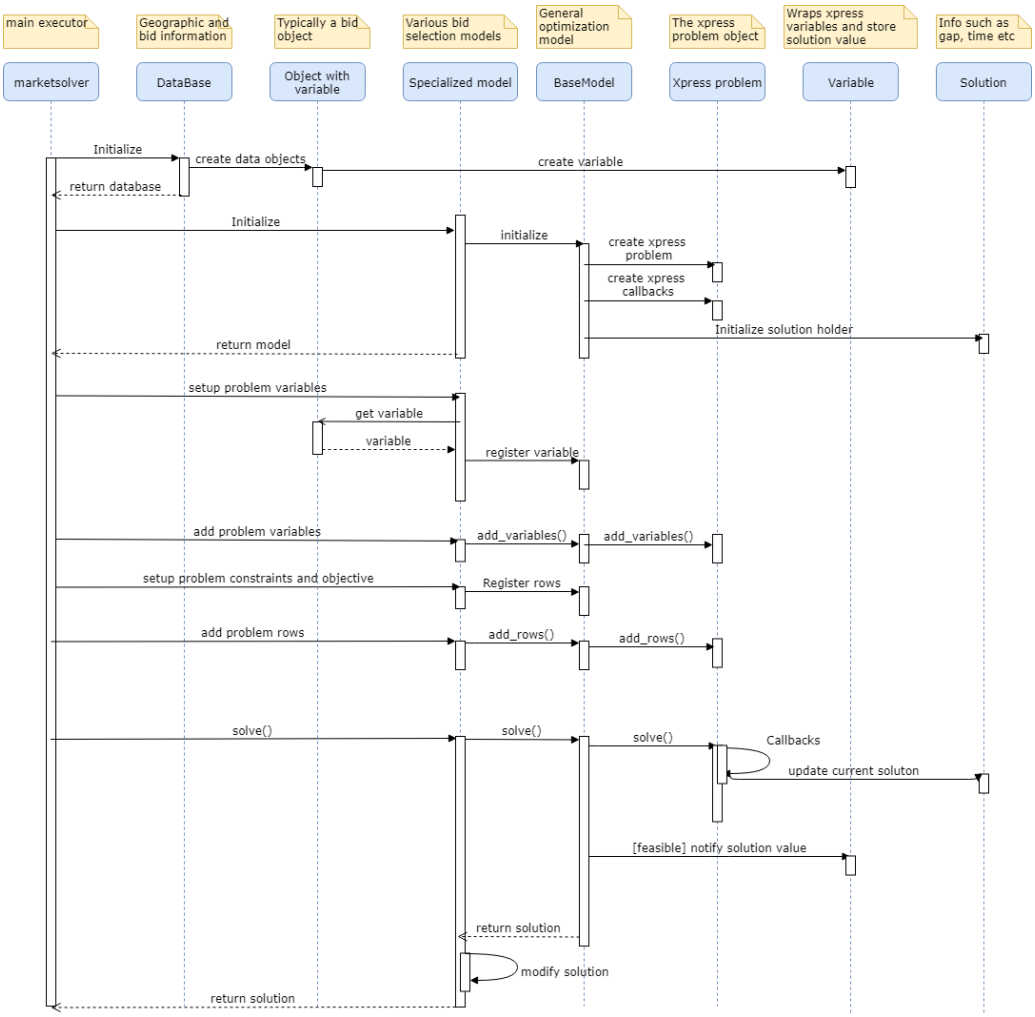
marketsolver.py	Includes the entrypoint to the clearmarket algorithm This function initializes json input data in data objects and runs the clearing algorithm and returns a json response
data_objects.py	Contains data objects used by the algorithm. This includes e.g. objects for regional

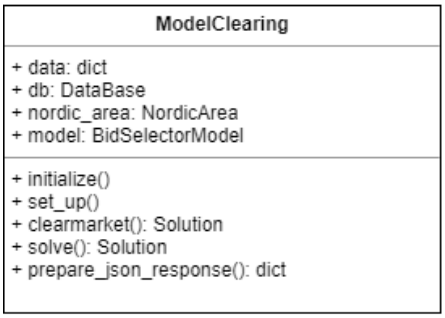
	information (TSO, macro area, region and interconnections (CZC's) and bids
database_initializer.py	Holds the class *database* which initializes data objects defined in *data_objects.py* from the json input file. A database instance holds references to the objects used by the algorithm
step_problems.py	Contains functions used to set up models and perform operations on data objects in the database. The class *SolutionHolder* is defined here. The class holds information on the solution from different problems. *Marketsolver.py* mainly interacts with functions in this module.
base_model.py	Contains all interactions with Xpress. The *Model* class is a general class that adds variables, rows and solves optimization problems The Model class is used as a subclass by all optimization models used by the program The module also defines the class *Variable* which is a wrapper for Xpress specific variables.
model_classes.py	Contains classes for the various optimization problems, which again are sub-classes of *Model*. The *BidSelectorModel* class is the class mostly used from this module. One instance of this class is defined every time a bid selector problem is solved. The class contains a number of methods to define and set up the optimization problem.
unittests.py	Contains unit tests for the program. We use the Pytest framework where we set up different classes to test the different parts of the program.
utilitites.py	Contains help classes used by the program

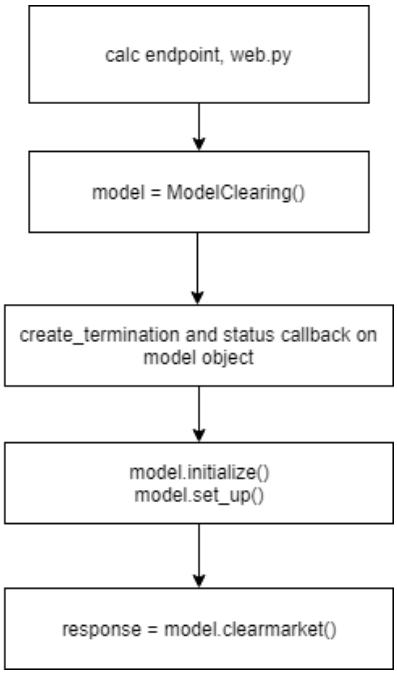
constants.py	Defines constants used, typically JSON-keys for input and response


Data objects
NordicArea
+ macro_areas: [MacroArea]
+ regions: [Region]
+ linked_bids: [bids]
+ exclusive_bids: [bids]
+ all_bids: BidSet
+ up_bids: BidSet
+ down_bids: BidSet
TimeHolder(Singleton)
+ nhours: int
+ _hour_range: [int]
+ hour_range(obj_type, *args): HourRange
HourRange
+ hour_range: [int]
+ get_hour_range(obj_type, *args): HourRange
+ overloaded _getitem_, _setitem_
DirectionHolder(Singleton)
+ down_code: string
+ up_code: string
+ directions: dict
+ get_code(int) : string
+ direction: Direction
Direction
+ up: int
+ down: int
+ direction_holder: DirectionHolder
+ set_direction_value(value, direction) :
MacroArea
+ prod_min: HourRange(int)
+ prod_max: HourRange(int)
+ regions: [Region]
+ set_prod_max(hour:int, value: float)
+ set_prod_min(hour: int, value: float)
+ up_bids: BidSet
+ down_bids: BidSet
+ accepted_bids: BidSet
Region
+ macro_area: [MacroArea]
+ InEdges: Connection
+ OutEdges: Connection
+ prod_min: HourRange(int)
+ prod_max: HourRange(int
+ up_bids: BidSet
+ down_bids: BidSet
+ accepted_bids: BidSet
+ set_prod_max(hour:int, value: float)
+ set_prod_min(hour: int, value: float)
Connection
+ to_area: MacroArea
+ from_area: MacroArea
+ capacity: float
+ flow: HourRange(Flow)
+ overloaded _getitem_ , returns flow object for hour h
+ create_flow_variables()
Flow
+ connection: Connection
+ capacity: float
+ volume: Variable
+ cost: float
+ create_flow_variable()
Bid
+ price: float
+ volume: float
+ id: int
+ hour_from: int
+ direction: int
+ hour_to: int
+ region: Region
+ volume_sol: float
+ status: Identifier, string
+ accept_bid: Variable
+ min_vol_var: Variable
+ quantity_factor: int
+ type: IDENTIFIER: Str
+ name: str
+ set_cleared(bool)
+ overloaded _eq_ compare id
BidSet
+ hours: HourRange([bid])
+ time_holder: TimeHolder
+ overloaded _getitem_: returns bids in hour: [bid]
+ add_bid(bids): bid or [bid]
1
0..*
2
1..*


Model
+ problem: XpressProblem
+ variables: [Variables]
+ constraints: [constraints]
+ solution_listener: []
+ handle_infeasibility: []
+ solution: Solution
+ random_seed: int
+ max_time: int
+ intital_gap: float
+ gap_increment: float
+ gap_interval: float
+ name: string
+ add_variable(*variable)
+ create_constraint(xpressCtr)
+ add_variables()
+ add_constraints()
+ add_objective_element(xpress expression)
+ set_objective()
+ solve()
+ print_problem()
+ clear_problem()
+ add_infeasibility_listener()
1
1
Solution
+ status: string
+ ub: float
+ lb: float
+ solution_time: float
+ feasible: bool
+ objective_vale: float
+ gap: float
+ sol_found: bool
Bid Solution
+ accepted_bids: BidSet
+ feasible: bool
+ objective_vale: float
+ gap: float
+ sol_found: bool
1
1
Bid SelectorModel
+ nordic_area: NordicArea
+ macro_areas: [macro_area]
+ regions: [region]
+ connections: [Connection]
+ all_bids: BidSet
+ up_bids: BidSet
+ down_bids: BidSet
+ solve()
+ add_energy_balance_constraint()


main executor
Geographic and bid information
Typically a bid object
Various bid selection models
General optimization model
The xpress problem object
Wraps xpress variables and store solution value
Info such as gap, time etc
marketsolver
DataBase
Object with variable
Specialized model
BaseModel
Xpress problem
Variable
Solution
Initialize
create data objects
create variable
return database
Initialize
initialize
create xpress problem
create xpress callbacks
Initialize solution holder
return model
setup problem variables
get variable
variable
register variable
add problem variables
add_variables()
add_variables()
setup problem constraints and objective
Register rows
add problem rows
add_rows()
add_rows()
solve()
solve()
solve()
Callbacks
update current soluton
[feasible] notify solution value
return solution
modify solution
return solution


ModelClearing
+ data: dict
+ db: DataBase
+ nordic_area: NordicArea
+ model: BidSelectorModel
+ initialize()
+ set_up()
+ clearmarket(): Solution
+ solve(): Solution
+ prepare_json_response(): dict


calc endpoint, web.py
model = ModelClearing()
create_termination and status callback on model object
model.initialize()
model.set_up()
response = model.clearmarket()

8 Execution Controls

The execution controls are specified in the JSON input file and handled automatically by the algorithm. The controls available to the operator are the following:

1. maxProcessingTime. The maximum time taken for Xpress execution in the algorithm (in seconds).
2. optimalityGap. The allowed gap between best-known solution and a bound. This value is the gap when you start the model.
   An optimality gap is the difference between a best-known solution and a value that bounds the best possible solution. For instance, a solution with optimality gap 10% has an objective value no worse than 10% than the objective value of the optimal solution (float between 0 and 1)
3. gapIncrement. The increase in allowed gap for each time interval (see number four below) (float between 0 and 1)
4. gapInterval. The time (in seconds) between each increase in allowed gap controlled by gapIncrement
5. randomseed. A 32-bit integer that controls how the Xpress solver organizes the optimization problem. If the same randomseed is used for two runs with the exact same input, the user will get the exact same solution in return. If the randomseed number is changed, the user can get a different solution if two or more equally good solutions exist. To make sure bids are not favorized because of the order in the bid list, we recommend the user to use different randomseeds between each auction.

In the following two sections, we further explain how the optimality gap and the time controls should be interpreted.

8.1 Optimality gap

The term optimality gap is the relative difference in objective value between the best solution found and the best bound value. Mixed integer linear problems (MILP) as the bid selector problem is, will use a

branch and bound technique to search through the solution space for the optimal solution and in this process, the algorithm will always keep track of two values:

- Objective value of best solution found (the total cost in the market of the best feasible solution found so far)
- Best bound value (objective value of the MIP relaxation, i.e. the best theoretical solution of the remaining nodes, but not necessarily a feasible solution)

The difference between best found solution and best bound can best be illustrated through a simple example. Say we have four indivisible bids of 15 MW with a cost of 5 EUR/MW each and a reserve requirement of 40 MW. The initial best bound value is found by relaxing the indivisibility requirements and will be 40 * 5 = 200 EUR. However, this is not a feasible solution and the optimal solution will be 45 * 5 = 225 EUR. This is not a difficult problem and the branch and bound process will be able to find and prove the optimal solution ('close the gap') of 225 EUR straight away.

When the solver is going through the nodes of the branch and bound tree, these two values move closer together until optimal solution is found and or proven.

The optimality gap is defined as:

$$gap = \frac{Obj.function\_value - bestbound\_value}{Obj.function\_value}$$

In any run of the solver, the user will define an acceptable gap. Whenever the value from the equation above is equal to or lower than the user defined optimality gap, the solver will stop and return the solution as an optimal solution (Status: OPT).

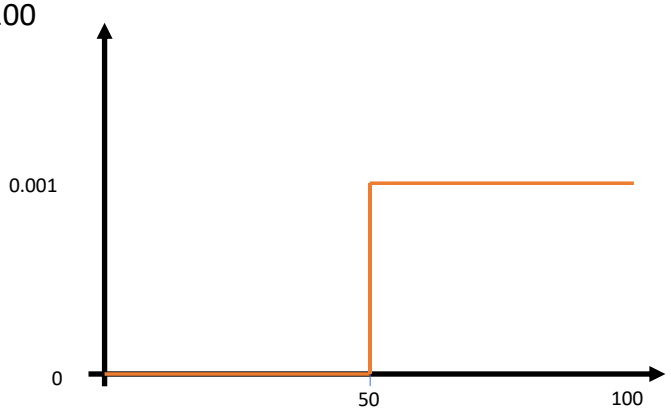
This means that an optimality gap of 0.0001 can guarantee that the found solution is not worse than 0.01% of the best theoretical bound.

As an example: 300 MW demand for up and down regulation for 24 hours and an average weighted cost per bid of 15 EUR/MW would give objective function values of around 200 000 EUR/MW. With an optimality gap of 0.0001, the best found solution would be no worse than about 20 EUR per day.

8.2 Time control and optimality gap

To explain how the optimiality gap and the time intervals are used, see the figure below.

- maxProcessingTime: 100
- optimalityGap: 0
- gapIncrement: 0.001
- gapInterval: 50



The maximum time we want the algorithm to run is 100 seconds. After this, the algorithm returns the best solution so far. In the example we start with optimality gap of 0. This means that we want the solution that is proven to be the best. However, we say that after 50 seconds (gapInterval), we want to increase the allowed gap to 0.001 (i.e. the solution is at least 99.9% of the optimal solution). If the user wants to keep the optimality gap constant throughout, the user can for example say that maxProcessingTime = gapInterval.

9 Error handling

9.1 General

There could be a variety of reasons why the algorithm fails. For that reason, several error types are defined to tell the user what went wrong. In the main json-output, the status will be 'FAILED' and in the log it will reference the error type, a short error message and a path that points in the direction of where in the code the algorithm failed. In addition to this, a separate json file is written with a more detailed description of the error.

The program separates between input error and implementation error. Input error indicates that the clearing algorithm is unable to run for the current input. Implementation error indicates errors in the algorithm implementation. The list below describes the defined error classes. They are defined in the module afrr_errors.

9.2 ErrorTypes

- InputError: Raised if the objects employed in the model cannot be initialized or there is inconsistent logic in the input data.
- ImplementationError: Raised if the algorithm fails. Concerns anticipated errors which indicates an error in the implementation. An implementation error requires changes in the algorithm.
- SystemError: Raised for certain XpressErrors
- InitializationError: Raised if object initialization fails. Subclasses input error
- XpressError: Raised if calls to the xpress API result in an exception. For instance, if there are multiple variables or constraints with equivalent names. An xpressError is furter raised as an ImplementationError or SystemError

Any raised error is handled by exception handlers in web.py. Any unforeseen exceptions are handled by a general exception handler. Some errors are handled within the application and logged as warnings.

In Appendix B we list functions/class methods which raise errors. The tables list the raised error type, the cause of the error, the error message, function/method in which the error originates and any unit tests. The unit tests listed are tests which explicitly triggers an error in the function/method. Tests for correct functionality are covered in other tests.

10 Appendix A – mini study

Test cases and instances

For the analysis of 1 MW reserve requirements and bid quantity size, we have used the 2018 simulation input data form 1st January to 4th February (35 test instances). On average the data each day consist of over 4000 bids. In the test data, all bids are indivisible and about 50% are block bids

Further, we want to test two changes to the input data assumptions:

- Going from 5 MW to 1 MW reserve requirements
- Going from 5 MW to 1 MW quantity factor for bid volumes

In order to test the algorithm, we define a set of dimensions in which we will test:

A. Demand step per bidding zone, hour and direction
B. Total demand per hour and direction
C. Amount of indivisible bids
D. Bid volume step sizes (1) and quantity factor (2) (1 / 2)
E. CZC reservation step sizes

Further, we define 6 test cases (1-6) that combines the dimensions listed above.

Test case	A	B	C	D	E
1	5 MW	330 MW	100%	5 / 5	1
2	1 MW	330 MW	100%	5 / 5	1
3	1 MW	331 MW	100%	5 / 5	1
4	1 MW	330 MW	100%	5 / 5	5
5	1 MW	70 MW	100%	1 / 1	1
6	1 MW	330 MW	90%	5 / 1	1

The first test (1) is the base case with 5 MW of demand steps and 5 MW of bid step sizes. In the second test (2) we change the demand step size from 5 MW to 1 MW (e.g. using 3, 9, 11, instead of 5, 10, 15), but keeping the total demand at a 5 MW step size, i.e. 330 MW. The third test (3) investigates the new demand cut with increasing the demand in one of the regions by one, and therefore the total sum each hour and direction to 331 MW. Test case 4 uses the test case 2 assumptions, but a 5 MW instead of 1 MW step size for CZC reservation. In test 5, we use a lower total demand, and reduce the bid sizes to 1 MW (essentially dividing all bid volumes by 5). For test 6, we use the assumptions from test 2, but reduce the number of indivisible bids used and change the quantity factor to 1 MW, more specifically reducing the minimum volume by 5 MW for every 10th bid. For this test we also change the quantity bid factor to 1.

Test results

The tables below show algorithm performance for the test cases described above. First, we present results before implementing the changes proposed in this document, secondly we show the same table, but with the new changes. For each test case we show the median duration to reach 0% optimality gap, the share of instances with optimality gap greater than 0 after 10 minutes, and the average gap for these problems.

Before changes to algorithm as described in this document

	Share of problems with optgap > 0 after 10 min	Average duration for problems with optgap = 0 before 10 min (seconds)	Average gap for problems with optgap > 0
1	0/35	2	-
2	32/35	81	0.071%
3	32/35	81	0.071%
4	0/35	11	-
5	0/35	1	-

6	13/35	120	0.0223%

The base case with 5 MW demand sizes and bid quantity factor solves to optimality (0% gap) for all problems, and the average duration is 2 seconds. If we change the individual demands for each bidding zone to 1 MW steps but keep the total per hour and direction unchanged from the base case, we see that almost all problems (91%) does not reach optimality within 10 minutes. The gap is low (average 0.071%) and is found within 1 minute for all problems, but the algorithm is not able to close the gap to 0 within the available 10 minutes. Test case 3 gives the same results as the effect of the demand cut is not active.

Changing the step size for CZC reservation (test case 4) brings results back to the base case, mainly because it reduces the number of feasible solutions. Test case 5 with 1 MW reserve requirements and with 1 MW bid sizes shows the same overall results as the base case with all instances solving to optimality.

The last test case (6) build on test case 2, but with 90% share of indivisible bids and a quantity factor of 1. We can see a clear effect of indivisibility - when increasing the number of bids that have a minimum volume requirement different to the maximum volume requirement. As we can see, making some of the bids divisible and reducing the quantity factor to 1, will also reduce solution times, because we introduce bids that can take values in 1 MW steps (e.g. if a bid has minVolume 5 and maxVolume 10, it can be selected with 5, 6, 7, 8, 9 or 10).

After changes to algorithm as described in this document

	Share of problems with optgap > 0 after 10 min	Average duration for problems with optgap = 0 before 10 min (seconds)	Average gap for problems with optgap > 0
1	0/35	2	-
2	17/35	100	0.071%
3	0/35	2	-
4	0/35	2	-
5	0/35	1	-
6	1/35	4	0.0223%

After implementing the four changes proposed in this document, we see that all but one test case give/prove optimal solution in all instances. The test case 2 is still the most difficult to solve, but only one instance did not go to 0% optimality gap. The average duration for this setup is around 100 seconds.

As an additional test, we ran one instance of test case 2 with no block bids, i.e. all hours independent of each other. This makes the problem much less complex as the solver can split the original problem into 24 subproblems with the new feature implemented. For the instance we tested, it did not solve to optimality within the 10-minute time limit with the original setup but found optimal solution after 36 seconds with no block bids.

Conclusions

For combinatorial problems as the aFRR problem, it is not easy to understand what drives solution times, but this mini-study has given us an indication of specific situations that are specifically challenging.

For all the tests we have done in this mini-study, the main issue for the solver comes when there is a combination of the following characteristics in the selection problem:

a) Close to 100% of all bids are indivisible
b) 1 MW step size for reserve requirements
c) 1 MW step size for CZC reservation
d) 5 MW step size in bid volumes
e) Bid quantity factor of 5
f) Total reserve requirements for each hour and direction is in 5 MW steps

As part of this analysis, we have done four changes to the algorithm which reduced the number of instances where optimality was not found/proven drastically:

1) Introduced a Nordic demand cut
2) Added inequalities for similar bids based on bid prices/costs
3) Increased the effort of finding subproblems
4) Changed the formulation of indivisible bids

11 Appendix B – Error types

ErrorType	Cause	Message	Method	Info	Unittest
InputError	Catch all	JSON initialization failed + error informasjon	Constructor of DataBase		
InputError	KeyError in dict	KeyError in json request	Constructor of Database		Test_general_input_error
InputError	Input data is not of type dict	Data is not of type dict	Database.initialize		Test_general_input_error_3
InputError	InputData is None	Data is None	ModelClearing.initialize		Test_general_input_error2
InputError	NumberOfHours in config is not integer.	n_hours is not integer	Init_time_holder	Initialization of shared timeholder object	TestTimeHolder->test_init_error
InputError	Wrong direction code	Direction code is not UP or DOWN	Init_direction_holder	Expects UP and DOWN as codes. Direction object is shared	Test_error_direction_holder

InputError	penaltyRelaxProdMinBidding-Zone in config is not integer or float	Penalty is not integer or float	Area.prod_min_penalty	Relaxed coefficient	TestMacroAreaObject->test_init_error
InputError	BiddingZone for demand is not region. I.e. not in interval [1, numberOfBiddingZones]	Region under demand is not of type Region	Database.init_demand		test_error_demand
InitializationError	Demand penalty is not integer or float	Demand penalty in bidding zone is not integer or float	Region.demand_penalty	Relaxed coefficient	TestRegionAreaObject->test_init_error
InitializationError	Hour for demand is not integer or hour is outside time domain	Hour for demand + reg.id + is outside timerange of problem	Region.set_demand	Called in init_demand in Database	TestRegionAreaObject->test_set_demand_error_1
InitializationError	Demand value is not integer or float	Demand is not integer or float	Region.set_demand		TestRegionAreaObject->test_set_demand_error_2
InitializationError	Direction is not part of directions in direction object	Direction is not part of directions in direction object	Direction.set_direction_value	Called for all values with a direction	TestDirectonHolder-

					>test_error_set_direction_value
Warning	Prodmin in biddingZone or macro area is not integer or float or hour is outside time domain	'Cannot set production limits for area: '+area.id + value is not integer or hour is outside time range'	Area.set_prod_min	Defaults to 0 for remaining areas	
Warning	Prodmax in biddingZone or macro area is not integer or float or hour is outside time domain	Cannot set production limits for area: ' + area.id +value is not integer or hour is outside time range'	Area.set_prod_max	Defaults to None for remaining areas , i.e. no prodmax constraints	
InitializationError	BiddingZone under macroArea is not region. I.e. not in interval [1, numberOfBiddingZones]	Region needs to be of class Region	MacroArea.regions	Cannot initialize macroarea for regions which does not exist	TestMacroAreaObject->test_region_error
InputError	MacroArea is not macroarea. I.e. not in	Macroarea does not exist. Not in interval [1,	Database.init_macro_areas		Test_init_macro_error

	interval [1, numberOfMacroAreas]	numberofMacroAreas]			
InputError	BiddingZoneOut or BiddingZoneIn under transferCapacitites is not of type Region. I.e. region not in interval [1, numberOfBiddingZones]	Connection from or to area is not region	Database.init_connections		Test_init_connection_error
InitializationError	Hour for transfer cost is outside time domain of probem	Hour is outside time range of problem'	Connection.set_cost	Called from Dtabase.init_connections	TestConnectionObject->test_error_hour
InitializationError	Transfer cost is not float or integer	Transfer cost is not int or float	Flow.cost	Called from Connection.set_cost	TestConnectionObject->test_error_cost
InitializationError	Transfer capacity is not float or integer	Transfer capacity is not int or float	Flow.capacity		TestConnectionObject->test_error_cap

InitializationError	Bid direction is not direction under direction object	Bid direction is not a direction in direction object	Bid.direction	Called from Database.init_bids	TestBidObject->Test_error_dir
InitializationError	Minimum bid volume is not integer or float	Bid min volume is not integer or float	Bid.min_volume	If None, min volume defaults to bid volume	TestBidObject->Test_error_min_vol
InitializationError	Bid region is not region from [1 to nBiddingZones]	Bid region is not of class Region in	Bid.Region		TestBidObject->test_error_region
InitilizationError	Bid volume is not integer or float	Volume is not integer or float for bid + bid.id	Bid.volume		TestBidObject-> test_error_volume
InitializationError	Bid price is not integer or float	Price is not integer or float for bid + bid.id	Bid.price		TestBidObject-> test_error_price
InitializationError	Bid hour from is not integer or not in time domain of probelm	Hour from is not integer or hour is not in time	Bid.hour_from		TestBidObject->test_hour_outside_time_range

		range of problem for bid' + bid.id			
InitializationError	Bid hour to is not integer or not in time domain of probelm	Hour to is not integer or hour is not in time range of problem for bid' + bid.id	Bid.hour_to	Bid hour to is < hour, i.e. hour_to == highest_hour+1 is accepted	
InitializationError	Quantity factor for bid is not integer for float	Quantity factor is not integer or float	Bid.quantity_factor		
InputError	Bid ids are not unique	Bid ids are not unique, cannot initialize optimization model	Database.control_bid_ids	Bid ids are used as variables names. Cannot have duplicate variable names	Test_control_bids
InputError	Bid id in linked bid group is not bid id	Link bid id is not bid id	Database.link_bids		Test_link_bids_error
InputError	Bid ids are not consecutive	Bid ids are not consecutive	Database.link_bids	Bid ids are used as array position to link bids. If bid ids	Test_non_consecutive_bids

				are not consecutive this logic does not hold	
InputError	Bid id in exclusive bid group is not bid id	Exclusive bid id is not bid id	Database.exclusive_bids		Test_exclusive_bid_error
InputError	If random seed is not int32	Optimization parameter is not integer. Integer values is not int32		The value is first checked for integer type, then int32. Thus, only one error message is applied to the InputError	Test_error_random_seed
SystemError	Error code from xpress is one of the following 706, # No memory to add sets 251, # Out of memory 245, # Not enough memory to presolve matrix 120, # Problem has to many rows, typically license problem 114, # Fatal error 50, # Inconsistent basis 20, # Insufficient memory	Xpress system error + xpress error information	ModelClearing.solve ModelClearing.set_up		

	for array 52, # Too many non-zero elements, typically license problem 167, # Failed to allocate memory of size <bytes> 394, # Network error 419, # Not enough space to store cuts 459,				
ImplemenatinError	XpressError which imply an implementationError	Algoritm implementation error. Xpress error + xpress error infomation	ModelClearing.solve ModelClearing.set_up	If error code is not in the code list under system errors, an implementation error is raised	TestXpressModelLogic->test_xpress_error
ImplementatonError	If default reverse connection is not defined	Default reverse connection should be defined	BidSelectorModel. add_flow_link	To create the down flow constraint we need a reverse connection. A default connection is initialized if not provided in the transfer capacity list	

ImplementationError	Solution object for model is not of type solution	Solution object provided is not of type solution	Model constructor
ImplementationError	Variable object is not of type variable	Variable added is not of type Variable	Model.add_variable
ImplementationError	Bid status is not a defined status	Attempted to set illegal bid status	Bid.status