Optimal Air Traffic Flow Management Regulations Scheme with Adaptive Large Neighbourhood Search

Ramon Dalmau, Gilles Gawinowski & Camille Anoraud
EUROCONTROL Innovation Hub (EIH)
Brétigny-Sur-Orge, France

***Abstract*—In the European air transportation network, overloads (i.e., critical imbalances between traffic demand and capacity) are generally resolved by activating air traffic flow management regulations, which delay flights on ground using a - more or less - *first-planned, first-served* principle. According to a recent research study, some of the regulations that are requested by the flow managers across Europe may not be strictly necessary due to the complex interactions between them. Results using an adaptive tabu search algorithm revealed that some regulations could be safely removed, thereby reducing the delay without causing problems elsewhere in the network. Such an algorithm, however, was designed to suggest which regulations among those requested by the flow managers could be cancelled, not to change the parameters of existing regulations or to propose new ones. This paper addresses this limitation by proposing a simple yet effective optimisation algorithm based on adaptive large neighbourhood search, which is able to determine the *best* set of regulations minimising air traffic flow management delay from scratch. The potential of the proposed method is assessed for various challenging scenarios using historical traffic data.**

***Keywords*—Air traffic flow management; demand and capacity balance; adaptive large neighbourhood search**

NOMENCLATURE

β	Destruction factor
δ	Success of the heuristic
ϵ	Tolerance for initial temperature
Ω	Set of operators
ϕ	Temperature decay factor
ρ	Weight decay factor
$\boldsymbol{p}$	Probability vector
$\boldsymbol{w}$	Weights vector
$\boldsymbol{x}$	Solution vector
c	Objective function (cost)
T	Simulated annealing temperature

I. INTRODUCTION

Adapting capacity to demand is the first step in aligning demand and capacity. This can be accomplished by modifying the airspace configuration, for example. If demand still cannot be met, air traffic flow management (ATFM) measures are implemented to match demand with available capacity. At present, the most common measure used by flow managers in coordination with the Network Manager consists of limiting the rate at which aircraft enter the congested traffic volumes[1] during a given period of time, i.e., to activate regulations.

[1]A traffic volume is related to a single geographical *entity* (either an aerodrome, a set of aerodromes, an airspace sector or a navigation point), and may consider all traffic passing through that entity or only specific flows.

The flights subject to one or several ATFM regulations are issued with a ground delay (i.e., the ATFM delay) on a first-planned, first-served basis by the Computer Assisted Slot Allocation (CASA) system [1] so that the maximum entry rates in the corresponding traffic volumes are not exceeded during the regulated periods. The mechanisms that are used to assign and manage ground delays using ATFM regulations are described in previous publications. The reader is refereed to Refs. [2], [3] for further details about this exciting subject.

During 2020, there were 2.6M minutes of ATFM delay in the European network, a drop of 89% when compared to 2019 [4]. The en-route ATFM delay was 0.33 min per flight (79% decrease with respect to 2019), and airport ATFM delay was 0.20 min per flight (67% decrease with respect to 2019). En-route air traffic control (ATC) disruptions and airport weather were the main reasons for ATFM delay in the network. This exceptional reduction of ATFM delay, however, was also consequence of the extremely low traffic levels caused by the COVID-19 outbreak (55% less traffic than in 2019). Because of the low levels of traffic, the current network infrastructure had plenty of capacity to meet the demand and less ATFM regulations were required. Despite the traffic demand forecasts are more uncertain than ever due to the unclear impact of the pandemic on the travel habits [5], domain experts forecast that the upward trend is going to resume once travel restrictions lighten [6]. With air traffic likely to increase, regulations (and delays) will inevitably come back to ensure safe operations.

Countless scientists have been interested for many years in optimising ATFM measures in order to resolve demand and capacity imbalances in the *best* way possible, albeit the concept of best is obscure because different actors with often conflicting interests are part of the optimisation problem.

For instance, Ref. [7] proposed a mixed-integer optimisation algorithm to assign ATFM delays in a way that minimises delay propagation to subsequent flights, simultaneously increasing flight adherence to departure slots at coordinated airports. Similarly, Ref. [8] formulated an integer programming model for strategic redistribution of flights so as to respect nominal sector capacities, in short computation times for large-scale instances. A variant that considers aircraft rotations through the turn-around time constraints was also proposed by [9], and recently Ref. [10] suggested a convenient way of visualising the results of that model. In consonance with the previous authors, Ref. [11] proposed a collaborative ATFM framework aiming to improve the cost-efficiency for airspace users when

facing ATFM measures. Arguably, one of the main difficulties of the aforementioned works was to translate the ATFM delay into a monetary cost. Reference [12] addressed this issue by defining quantitative and qualitative indicators to assess the expected impact of the costs that regulations could cause.

The paradigm of ATFM used in the current operations, however, is way far from a massive cherry-picking that optimally assigns specific delays to individual flights. Even though previous research demonstrated through realistic use cases that optimisation models could significantly reduce total ATFM delay, the current delay assignment by CASA is regarded as *equitable* by airspace users, and is a legacy and trusted system that has proven to ensure safe flight operations for decades.

In this context, Ref. [13] presented a new ATFM technique that, exploiting some of the rules and mechanisms existing in CASA, could potentially allow airspace capacity to be used more effectively, thus reducing ATFM delays and their associated costs for the airspace users, with relatively *small* changes in the operational system that is currently in operations.

Getting closer to current operations, a recent work [3] was founded on the hypothesis that, especially during busy days, only some of the requested regulations are actually indispensable, i.e., a set of the regulations requested by flow managers could still solve all overloads with less delay. Instead of creating new regulations, however, the proposed algorithm takes the regulations requested by the flow managers as starting point, assuming that otherwise all of them will be active, and tries to cancel those regulations that are ineffective. Accordingly, determining the optimal set of regulations becomes a binary optimisation problem, where each regulation can be either active or cancelled.

Such an algorithm, however, was designed to suggest which regulations among those requested by the flow managers could be cancelled, not to change the parameters of existing regulations or to propose new ones. This paper addresses this limitation by proposing an *hyper-heuristic* algorithm (i.e., an heuristic to select heuristics) based on adaptive large neighbourhood search (ALNS), which is able to determine the *best* set of regulations minimising air traffic flow management delay from scratch [14]. The performance of the proposed algorithm is demonstrated for several busy days during 2019.

II. Adaptive large neighbourhood search

Recently, heuristics based on large neighbourhood search (LNS) [15] have shown outstanding performance in solving a wide variety of transportation problems [16]. LNS gradually improves the initial solution ($\boldsymbol{x}_0$) by alternately *destroying* and *repairing* it. The steps of LNS are listed in Algorithm 1.

The basic idea behind LNS is to alternate between feasible and infeasible regions, similar to how strategic oscillations work [17]. At each iteration of the algorithm, the destroy heuristic creates an infeasible solution, which is brought back into the feasible region by the repair heuristic. It is worth noting that the destroy and repair heuristics can be thought of as fix-and-optimise operations, respectively. In particular, the fix operation (equivalent to the destroy heuristic) freezes some variables of the solution to their current values, whereas the optimise operation (equivalent to the repair heuristic) seeks to improve the solution by modifying the variables that are free. Furthermore, the LNS also has many similarities to the iterated greedy [18] and the ruin and recreate [19] algorithms.

In the problem addressed in this paper, the destroy heuristic is in charge of removing some of the regulations in the current solution. After destruction, the solution often presents imbalances between demand and capacity, i.e., it is infeasible. Starting from an infeasible solution, the repair heuristic takes action to resolve overloads by creating new regulations. A very simple destroy heuristic could consists of cancelling those regulations that generate most ATFM delay, while a straightforward repair heuristic could consists of activating regulations in the traffic volumes where major overloads exist.

Algorithm 1 Large neighbourhood search (LNS)

Require: $\boldsymbol{x}_0$
$\boldsymbol{x} \leftarrow \boldsymbol{x}^* \leftarrow \boldsymbol{x}_0$
repeat
  $\boldsymbol{x}' \leftarrow \textsc{Repair}\,(\textsc{Destroy}\,(\boldsymbol{x}))$
  if $\textsc{Accept}\,(\boldsymbol{x}, \boldsymbol{x}')$ **then**
    $\boldsymbol{x} \leftarrow \boldsymbol{x}'$
    if $c\,(\boldsymbol{x}) < c\,(\boldsymbol{x}^*)$ **then**
      $\boldsymbol{x}^* \leftarrow \boldsymbol{x}$
    end if
  end if
until Termination
return $\boldsymbol{x}^*$

According to Algorithm 1, at each iteration of the LNS algorithm a new candidate solution ($\boldsymbol{x}'$) is generated as a result of sequentially applying the destroy and repair heuristics to the current solution ($\boldsymbol{x}$). The candidate solution replaces the current solution if it satisfies the acceptance criteria, otherwise is discarded. The acceptance criteria can be implemented in different ways. Despite the simplest choice is to only accept better solutions (the so-called high-climbing acceptance), other methods such as simulated annealing, threshold acceptance or old bachelor acceptance are often implemented in order to reduce the chances of getting stuck in local minima [20].

If the objective function of the candidate solution is better than the best found so far, it becomes the new best solution ($\boldsymbol{x}^*$). Finally, the algorithm terminates when a certain condition is triggered, e.g., after a number of iterations, when the maximum execution time is exceeded, when the best objective has not improved during a given number of iterations, etc.

Note that the destroy heuristic must be designed in such a way that the entire search space can be reached, i.e., it cannot constantly focus on destroying the same part of the solution. For this reason, some degree of randomness is commonly introduced in the destroy heuristic. Conversely, in most implementations the repair heuristic makes use of variants of the greedy paradigm, e.g. performing the locally best (or least bad) action from the currently infeasible solution.

Using this kind of design, the destroy heuristic guarantees *diversification*, while the repair heuristic fosters *intensification*.

Note that LNS uses exactly one destroy and one repair heuristic. Adaptive large neighbourhood search (ALNS) is an extension of LNS originally introduced by [21], which allows multiple destroy and repair heuristics to be used during the optimisation. This algorithm has been used to solve difficult problems like the multi-period vehicle routing [22].

ALNS was designed on the premise that different problem instances, and even different solutions to the same problem, are managed with varying success by distinct destroy and repair heuristics. It may be difficult, however, to guess which heuristics will be the most successful. ALNS uses a rather simple yet effective mechanism to *learn*, on the fly, the likelihood that an heuristic will generate a good candidate [14].

Let Ω^- and Ω^+ be the catalogue of possible destroy and repair heuristics, respectively (note that both Ω^- and Ω^+ must be defined by the user beforehand). Each heuristic has an associated weight, which determines the likelihood of selecting the heuristic at each iteration. The vector of weights of the destroy and repair heuristics are denoted by $\boldsymbol{w}^-$ and $\boldsymbol{w}^+$, respectively. The probability to select the i^{th} destroy and the j^{th} repair heuristics is given, respectively, by:

$$p_i^- = \frac{w_i^-}{\sum_{k=1}^{|\Omega^-|} w_k^-}; \quad p_j^+ = \frac{w_j^+}{\sum_{k=1}^{|\Omega^+|} w_k^+}. \tag{1}$$

That is, the higher the weight of an heuristic, the more likely it will be selected to destroy/repair the current solution.

The weight of the different heuristics must be updated in order to increase the probability of selecting successful heuristics, and to decrease that of less effective ones. In order to accomplish this goal, the quality (δ) of the candidate solution generated by the current destroy and repair heuristics is determined at each iteration of the algorithm. The quality, which has a purpose similar to that of the *reward* in reinforcement learning, depends on its success. In particular:

$$\delta = \begin{cases} \delta_* & \text{if } \boldsymbol{x}' \text{ is the better than } \boldsymbol{x}^* \\ \delta_b & \text{else if } \boldsymbol{x}' \text{ is better than } \boldsymbol{x} \\ \delta_a & \text{else if } \boldsymbol{x}' \text{ is not better than } \boldsymbol{x} \text{ but is accepted} \\ \delta_r & \text{Otherwise } (\boldsymbol{x}' \text{ is rejected}), \end{cases} \tag{2}$$

where $\delta_{(\cdot)}$ are fixed parameters of the algorithm.

After determining the quality of the candidate solution with Eq. (2), the weights of the destroy and repair heuristics that generated the candidate solution are updated according to:

$$w_i^- = \rho\delta + (1-\rho)w_i^-; \quad w_j^+ = \rho\delta + (1-\rho)w_j^+, \tag{3}$$

where the decay factor (ρ) controls the influence of the recent success of the applied heuristic on its weight.

The steps of the ALNS are detailed in Algorithm 2. Note that a local search step could be also applied when finding a new best solution ($\boldsymbol{x}^*$), e.g., with tabu search. The objective of the (optional) local search step is to thoroughly explore the close neighbourhood of the current best solution and attempt to reach a local minima before to continue exploring a large (i.e., far) neighbourhood. In this paper, however, the local search has not been implemented for the sake of simplicity and clarity.

Algorithm 2 Adaptive large neighbourhood search (ALNS)

Require: $\boldsymbol{x}_0$, $\boldsymbol{w}_0^-$, $\boldsymbol{w}_0^+$
$\boldsymbol{w}^- \leftarrow \boldsymbol{w}_0^-$
$\boldsymbol{w}^+ \leftarrow \boldsymbol{w}_0^+$
Initialise $\boldsymbol{p}^-$ and $\boldsymbol{p}^+$ according to Eq. (1)
$\boldsymbol{x} \leftarrow \boldsymbol{x}^* \leftarrow \boldsymbol{x}_0$
repeat
  DESTROY $\leftarrow$ Sample from Ω^- according to $\boldsymbol{p}^-$
  REPAIR $\leftarrow$ Sample from Ω^+ according to $\boldsymbol{p}^+$
  $\boldsymbol{x}' \leftarrow$ REPAIR (DESTROY ($\boldsymbol{x}$))
  if ACCEPT $(\boldsymbol{x}, \boldsymbol{x}')$ **then**
    if $c(\boldsymbol{x}') < c(\boldsymbol{x}^*)$ **then**
      $\delta \leftarrow \delta_*$
      $\boldsymbol{x}^* \leftarrow \boldsymbol{x}'$
    else if $c(\boldsymbol{x}') < c(\boldsymbol{x})$ **then**
      $\delta \leftarrow \delta_b$
    else
      $\delta \leftarrow \delta_a$
    end if
    $\boldsymbol{x} \leftarrow \boldsymbol{x}'$
  else
    $\delta \leftarrow \delta_r$
  end if
  Update $\boldsymbol{w}^-$ and $\boldsymbol{w}^+$ according to Eq. (3)
  Update $\boldsymbol{p}^-$ and $\boldsymbol{p}^+$ according to Eq. (1)
until TERMINATION
return $\boldsymbol{x}^*$

III. Optimisation Algorithm

Section II described the working principle of the generic ALNS algorithm. The designer must specify the following functions in order to customise the ALNS algorithm to a specific problem: the initial solution generator ($\boldsymbol{x}_0$); the catalogue of DESTROY and REPAIR heuristics (Ω^- and Ω^+, respectively); the ACCEPT criteria; and the TERMINATION criteria. Next sections describe the functions that have been selected for tailoring the general-purpose ALNS algorithm to the problem of optimising the ATFM regulations scheme.

The following paragraphs explain the various data structures used by the optimisation algorithm. In the problem tackled in this paper, a solution $\boldsymbol{x}$ consists of a set of ATFM regulations, where each ATFM regulation is described by:

- the traffic volume identifier, and
- the list of applicability periods, where each applicability period is composed of:
  - the start time,
  - the end time and
  - the maximum entry rate (in entries per hour).

The cost of a feasible solution $c(\boldsymbol{x})$ is the total ATFM delay generated by the corresponding set of regulations applied to a certain traffic demand, where each flight of the traffic demand is a data structure composed of:

- the flight identifier,
- whether the flight is exempt from ATFM measures and
- the traffic volume profile, where each element of the traffic volume profile is a data structure composed of:
  - the traffic volume identifier and
  - the estimated time over (ETO) the traffic volume.

Last but not least, the declared set of capacities is also required in order to detect overloads as well as to determine the rate of the regulations created during the optimisation process. Each element of this data structure is composed of:

- the traffic volume identifier,
- the capacity periods, where each capacity period is composed of:
  - the start time,
  - the end time and
  - the declared capacity (in entries per hour).

A. Initial solution

The initial solution is a key component of presumably any optimisation algorithm, because the closer the initial solution to the (unknown) best solution, the less time it will take to converge. Generating high-quality initial solutions, however, may be difficult or time consuming. In most of the cases, there exists a trade-off between the quality of the initial solution and the time it takes to generate it. In this paper, a simple yet effective greedy optimisation algorithm has been implemented to create relatively good initial solutions in a short amount of time. The steps of this algorithm are listed in Algorithm 3.

Algorithm 3 Greedy regulations

Require: Traffic demand & capacity
1: Identify overloaded time windows taking into account `overload_20` and `overload_60` tolerances
2: **repeat**
3: Activate a new regulation on every overloaded time window, using the corresponding capacity as regulation rate
4: Group consecutive regulations with same rate within the same traffic volume (or which gap is lower than a `max_gap` tolerance)
5: Remove regulations which duration is less than `min_duration`
6: Run the CASA algorithm
7: Shift the traffic demand according to the delays assigned by CASA
8: Identify overloaded time windows taking into account `overload_20` and `overload_60` tolerances
9: **until** All overloads are resolved
10: **return** Set or regulations that resolve overloads in a greedy way

Algorithm 3 starts by identifying the set of time windows that are overloaded given the traffic demand and the declared capacity. In this process, both windows of 20 min duration slicing every 20 min as well as windows of 60 min duration slicing every 20 min are monitored. For the sake of clarity, Fig. 1 shows the time windows monitored for an hypothetical traffic volume, considering different combinations of duration (Λ) and slice (λ) . It is worth noting that certain tolerances may be taken into account when deciding whether or not an entry counts value that exceeds a time window's capacity is considered to be an overload. For instance, an overload could exist if and only if the entry counts value on a 20 min (resp. 60 min) window is 15% (resp. 5%) higher than the capacity. These tolerances are denoted as `overload_20` and `overload_60`, respectively, and the default value is 0%.

At each iteration, regulations are activated on the overloaded time windows, using the corresponding capacity as regulation rate. Then, consecutive regulations with same rate within the same traffic volume (or which gap is lower than a `max_gap` tolerance) are grouped. Following this step, regulations which duration is less than another parameter, called `min_duration`, are removed. Finally, the CASA algorithm is executed with the current set of regulations, the traffic demand is shifted according to the delays assigned by CASA, and overloads are identified again in the new situation. Algorithm 3 stops when all overloads are resolved by the current set of regulations.

The current implementation of the algorithm uses the high-fidelity replica of CASA included in the R-NEST (Research Network Strategic Monitoring Tool) tool of EUROCONTROL.

B. DESTROY *heuristics*

In this paper, three DESTROY heuristics have been defined, which select a subset of regulations to be removed from the current solution $\boldsymbol{x}$. As discussed in Section II, any destroy heuristic should include some randomness in order to foster exploration. For this reason, a β percentage of the regulations are randomly removed from the current solution as a result of the destroy phase. The probability of each regulation being removed is determined by the specific destroy heuristic.

1) High delay: The objective of this heuristic is to explore solutions with a high potential of decreasing the objective function. As a result, the likelihood of each regulation being removed from $\boldsymbol{x}$ is proportional to the ATFM delay that it generates: the higher the delay, the greater the probability.

2) Low efficiency: The efficiency of a regulation is defined as the ratio between the number of flights that have the regulation as most penalising (MPR), and the total number of flights affected by the regulation. Accordingly, a regulation concerning a lot of flights which delay is *forced* by other regulations in the network is considered inefficient. When this heuristic is selected, the likelihood of each regulation being removed from $\boldsymbol{x}$ is inversely proportional to its efficiency: the lower the efficiency, the greater the probability.

3) Random: The random heuristic, as the name implies, assigns the same probability to all regulations in $\boldsymbol{x}$.

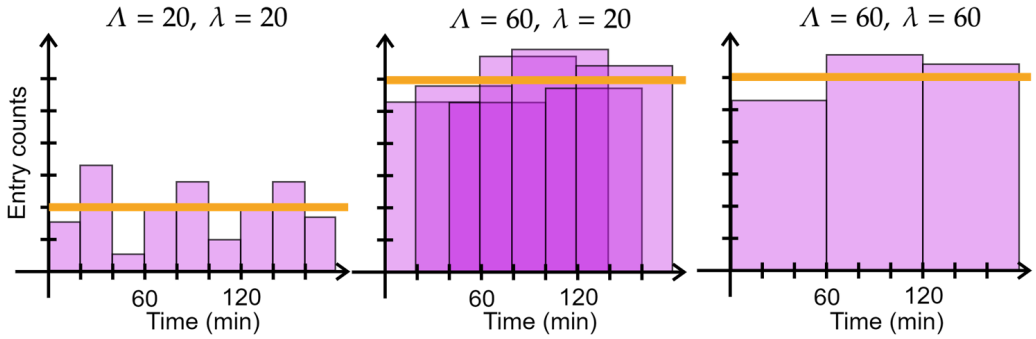


Fig. 1: Time windows monitored for an hypothetical traffic volume, considering different combinations of duration (Λ) and slice (λ). Each magenta bar represents a time window. The width denotes the duration (min), while the height represents the number of flights which ETO falls within the time window. The capacity is indicated by the horizontal orange line.

C. REPAIR *heuristic*

As discussed in Section II, any REPAIR heuristic should optimise the variables that are free after the destroy phase, while keeping those fixed untouched. After the destroy phase, some overloads will most likely be formed as a result of cancelling regulations; alternatively, the repair phase could be effectively skipped and more regulations destroyed.

In this paper, a single repair operator has been defined, which acts greedily by regulating overloaded time windows with Algorithm 3. Note that, in contrast to the generation of the initial solution, where Algorithm 3 starts from scratch (i.e., from a clean solution without regulations), the repair heuristic starts with the destroyed solution, which (likely) has some regulations in place. This implies that the traffic demand has the effect of these regulations. Algorithm 3 can only add more regulations, but cannot remove existing ones (which comprise the fixed variables in the destroy-repair argon).

D. ACCEPT *criteria*

In this paper, simulated annealing has been adopted for the ACCEPT criteria of the ALNS algorithm. Using this method, each candidate solution $\boldsymbol{x}'$ is accepted with probability $\exp \frac{c(\boldsymbol{x})-c(\boldsymbol{x}')}{T}$, where T is the so-called temperature.

Following the approach suggested by [23], The initial temperature T_0 is chosen so that a hypothetical candidate solution that is ϵ percent worse than the initial solution $\boldsymbol{x}_0$ has a 50% percent chance of being accepted. Accordingly:

$$T_0 = \epsilon \frac{c(\boldsymbol{x}_0)}{\log 2}. \tag{4}$$

The temperature parameter is decreased by a factor ϕ in each iteration. As a result, as the search progresses, the algorithm tolerates fewer and fewer deteriorating solutions. Both ϵ and ϕ are fixed parameters to be defined by the user.

E. TERMINATION *criteria*

In the current implementation, the ALNS algorithm stops after a given number of destroy-repair `iterations`.

IV. DESCRIPTION OF THE EXPERIMENT

In this paper, the performance of the algorithm presented in Section III to optimise the scheme of ATFM regulations was assessed for various realistic scenarios. Section IV-A particularises the parameters of the algorithm used during the experiment, and Section IV-B describes the different scenarios. Finally, Section IV-C describes our implementation of the ANLS algorithm for the problem at hand.

A. *Parameters of the algorithm*

The algorithm described in Section III may react quite differently depending on the settings. The values of the various parameters used in this experiment are shown in Table I.

TABLE I: Parameters of the optimisation algorithm

Algorithm	Parameter	Value
OVERLOADEDWINDOWS	`overload_20`	0.
	`overload_60`	0.
GREEDYREGULATIONS	`max_gap`	60 min
	`min_duration`	0
TERMINATION	`iterations`	200
ALNS	$(\delta_*, \delta_b, \delta_a, \delta_r)$	(3, 2, 1, 0.5)
	β	0.2
	ρ	0.8
ACCEPT	ϵ	0.05
	ϕ	0.995

According to Table I, a feasible solution could not present any overload in 20 or 60 min windows; overloaded windows separated by less than 60 min within the same traffic volume were grouped together under the same regulation; any overloaded window (regardless of the duration) was regulated; the ALNS algorithm attempted to optimise regulations by destroying and repairing the solution 200 times; in the first iteration, any candidate solution 5% worse than $\boldsymbol{x}_0$ was accepted with 50% probability; the temperature parameter at a given iteration was 99.5% the temperature of the previous iteration; 20% of the active regulations were removed as a result of the destroy phase; and the decay factor for the weights in Eq. (3) was 0.8.

B. Scenarios

In this experiment, the performance of the algorithm was assessed for 8 different scenarios, corresponding to the morning (AM) and afternoon (PM) periods of 4 very busy days during summer 2019: 26th and 27th of July, and 9th and 27th of August. It should be noted that only a sub-region of the network was considered, which includes the Area Control Centres (AACs) involved in the ISOBAR[2] project: LFMM-CTA (Marseille), LFEECTA (Reims), LECMCTA (Madrid), LECBCTA (Barcelona) and LECPCTA (Palma). The demand and capacity data for each scenario were obtained from the EUROCONTROL's Demand Data Repository (DDR).

For each scenario, three different *pictures* of the traffic demand can be obtained from DDR: the initial (or last-filled flight plan), the regulated, and the actual (i.e., what was actually flown). Because the regulated demand already has the effect of the ATFM regulations that were activated during the day, and the actual demand has further tactical modifications, the initial demand was used to populate the traffic data.

For determining the capacity, one could use the actual opening scheme of each ACC (i.e., which sector configurations were used along the day) as reported in the DDR. Comparing the initial demand with the actual opening schemes, however, may result in a large number of exaggerated overloads that are difficult (or even impossible) to solve. For this reason, the capacity values were obtained from opening schemes optimised to minimise the overloads subject to manpower constraints. In particular, the ICO (Improved Configuration Optimizer) tool [24] was executed by considering the initial demand in order to obtain, for each ACC, an opening scheme more in line with the demand. The capacity values according to these optimal opening schemes were used to run the algorithm.

It should be noted that any research can fully replicate the scenarios by using the network strategic tool (NEST) v1.8 tool of EUROCONTROL: NEST can load historical data from DDR and can execute the original ICO algorithm on them.

C. Implementation details

The ALNS algorithm was implemented in Python v3.8. The typical execution time per algorithm using the Red Hat Enterprise Linux release 8.5 with 12x Intel(R) Xeon(R) W-2235 CPU @ 3.80 GHz processors and 256 GB of RAM is:

- Greedy regulations (Algorithm 3): Each run of the CASA algorithm of R-NEST takes around 0.5 s. The greedy regulations algorithm converges, most of the time, in 5 iterations. Therefore, the execution time is around 3 s.
- ALNS (Algorithm 2): in the current implementation, the ALNS consists of executing the greedy regulations Algorithm 200 times. Accordingly, the execution time is around 10 min. The authors admit that 200 iterations is insufficient to ensure a thorough exploration of the state space. The algorithm, however, was used in real-time validation exercises where human operators expected a relatively "good" solution in a short amount of time. In a point of fact, at least 1000 destroy-repair iterations may be required to reach a near-optimal solution.
- For comparison purposes, solving the exact slot allocation problem for the same scenarios using mixed-integer linear programming (MILP) [25] takes at most 1 min by using the CBC (COIN-OR Branch and Cut) solver (release 2.10.5). It should be noted, however, that the slot allocation problem does not consider the first-planned first-served rule that governs the CASA algorithm.

[2]https://www.sesarju.eu/projects/isobar

V. Results

This section presents the main results of the experiment. In particular, Section V-A thoroughly describes one of the scenarios for illustrative purposes. The aggregated performance metrics are presented and discussed in Section V-B.

A. Illustrative example

The scenario corresponding to the 9th of August 2019 during the PM period has been selected as illustrative example. Figure 2 shows the maximum overload (i.e., absolute entry counts above the capacity) per airspace sector in the region of interest, considering monitoring windows of 20 min. Note that Fig. 2 has been generated by considering the initial demand and the capacity from the optimal (ICO's) opening schemes.

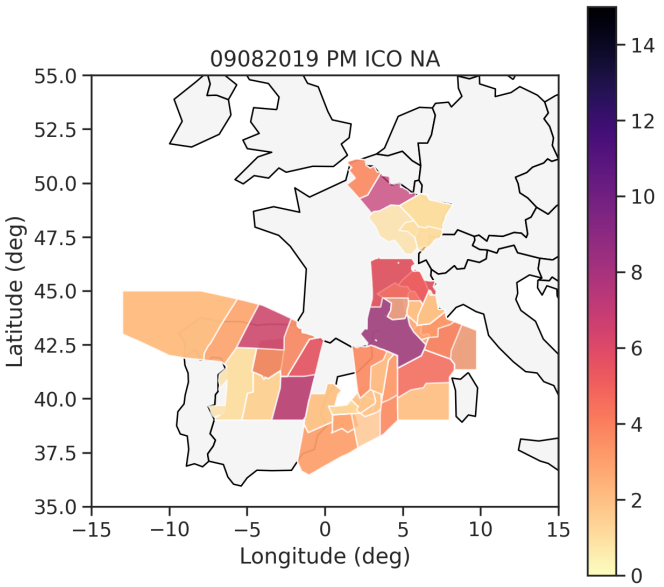
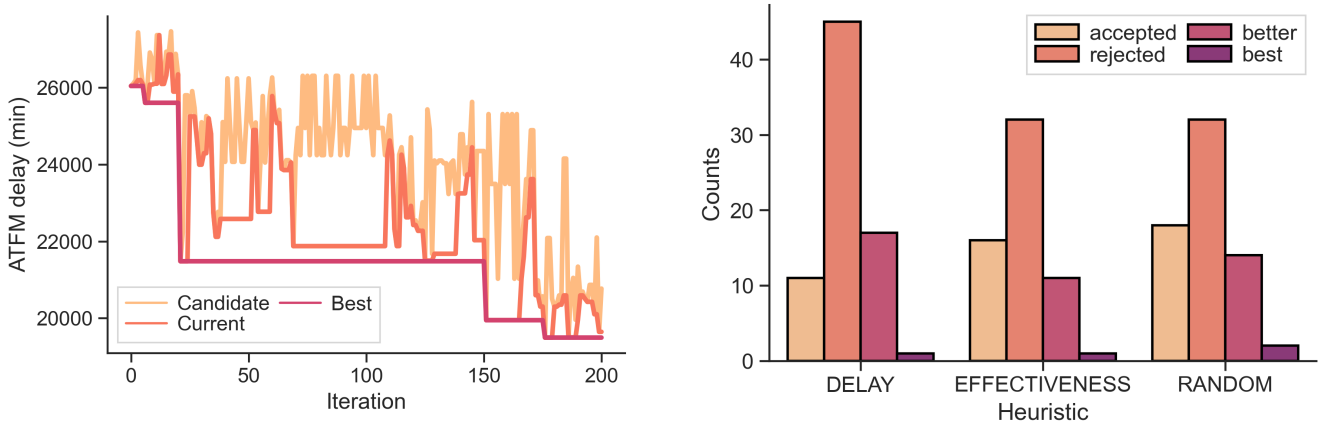


Fig. 2: Overloads (counts above capacity) for 09082019 PM

According to Fig. 2, all the ACCs involved in the project presented severe demand-capacity imbalances in this scenario.

The ALNS algorithm was executed to resolve these imbalances, repeatedly destroying and repairing the initial (greedy) solution - with a cost of 26043 min - during 200 iterations. Figure 3 shows some metrics of the optimisation process. In particular, Fig. 3a shows the cost c of the current ($\boldsymbol{x}$), candidate ($\boldsymbol{x}'$) and best ($\boldsymbol{x}^*$) solutions in each iteration; and Fig. 3b shows the histogram of success for each destroy heuristics.



(a) Convergence of the algorithm

(b) Success of the heuristics

Fig. 3: Optimisation metrics for 09082019 PM

According to Fig. 3a, and as expected, the more the number of iterations, the better the best solution found so far. Furthermore, the convergence of the best solution follows an exponential curve, with a significant improvement in the early iterations. For instance, the best solution improved from around 26K minutes of delay to approximately 21.5K after 25 iterations. In this scenario, the cost of the best solution was 19154 min, 6889 min (26%) better than the initial, greedy, solution. In Fig. 3a, one can also observe several cases in which the candidate solution was not better than the current one, but was accepted due to the acceptance strategy based on simulated annealing, which objective is to escape from local minima (for instance iteration 100, when the candidate solution is accepted even if being worse than the current one).

Figure 3b shows that, for this particular scenario, the most successful destroy heuristic consisted of removing regulations randomly, with 2 best instances and 30 rejections. Destroying regulations with low effectiveness presents a similar histogram, albeit with less better and accepted cases. The least successful destroy heuristic, on the other hand, was to remove regulations with high delay, with just 1 best instance and 46 rejections.

Figure 4 shows the total ATFM delay per regulated airspace sector in the region of interest for the greedy (4a) solution and for the best solution (4b), as a result of the optimisation.

Both ATFM delay maps shown in Fig. 4 are consistent with the absolute overloads map shown in Fig. 2, i.e., airspace sectors with high overloads typically require more ATFM delay to resolve the corresponding demand-capacity imbalance. ATFM delays shown in Fig. 4b, however, are generally lower than those shown Fig. 4a for the same airspace sectors, thereby illustrating the benefits of the suggested optimisation method.

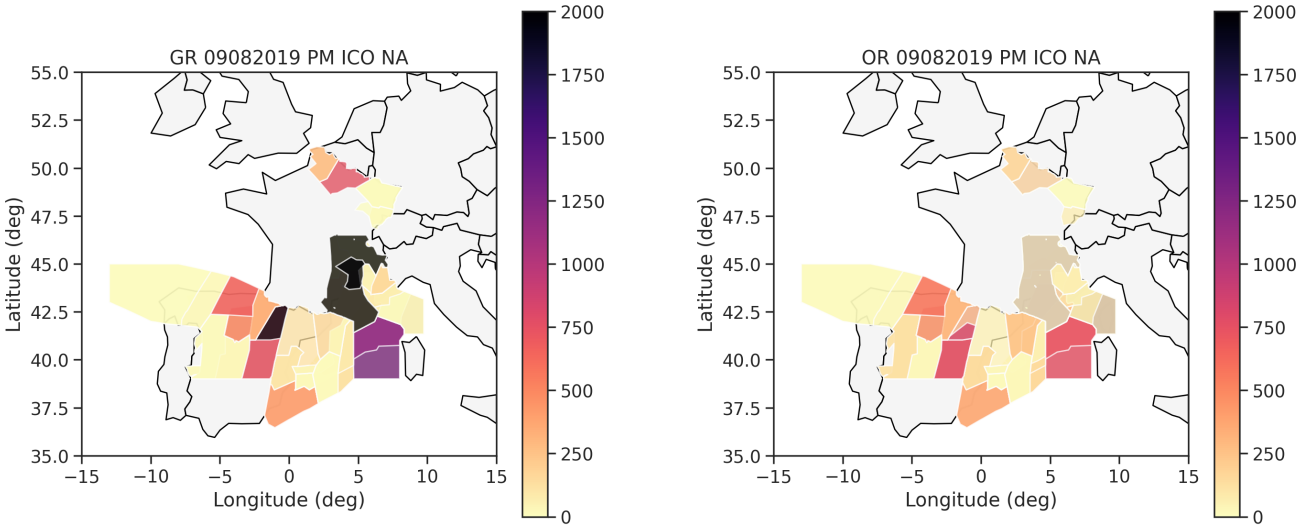
B. Aggregated performance metrics

Table II shows the aggregated performance metrics for the 8 scenarios, including the number of regulated flights, the number of regulated flights with a delay higher than 30 min, the maximum, average, median, and maximum ATFM delay as well as its standard deviation. In Table II, GR and OR stand for greedy regulations (i.e., the initial solution, or $\boldsymbol{x}_0$) and best regulations (i.e., the best solution, or $\boldsymbol{x}^*$), respectively.

TABLE II: Performance metrics

Scenario Date	Period	Solver	Flights (count) Delayed	>30 min	ATFM Delay (min) Max.	Mean	Median	Std.	Sum
26072019	AM	GR	1973	492	84	22	14	19	**44071**
		OR	1950	423	69	19	13	15	**37692**
	PM	GR	1604	167	83	16	11	15	**26428**
		OR	1534	166	83	16	11	16	**25080**
27072019	AM	GR	2031	373	80	20	14	16	**41153**
		OR	1916	378	80	20	13	17	**39238**
	PM	GR	1737	83	52	14	13	8	**25983**
		OR	1627	52	45	13	12	7	**22374**
09082019	AM	GR	2108	552	65	21	15	14	**44403**
		OR	2017	559	64	21	16	14	**43323**
	PM	GR	1716	208	42	15	12	9	**26043**
		OR	1561	9	33	12	11	6	**19154**
27082019	AM	GR	1743	221	102	18	13	17	**31707**
		OR	1665	196	102	17	12	17	**28899**
	PM	GR	1500	194	58	15	12	12	**23302**
		OR	1416	145	55	14	11	10	**20205**

According to Table II, the ALNS algorithm proposed herein was able to improve the initial (greedy) solution in all scenarios. The relative improvement, however, varies greatly among scenarios. For instance, in the scenario of the illustrative example (09082019 PM) the cost of the initial solution was reduced by around 26%, while in the AM period of the same day the reduction was just 2.5%. The precise difference between the cost of the initial and the best solutions depends on (1) the quality of the initial solution and (2) the success of the heuristics during the 200 destroy-repair iterations. Overall, meta-heuristics cannot mathematically guarantee the global optimality of the best solution, thus one must be enthusiastic with finding a sufficiently decent local optimum in a reasonable amount of time for such a difficult combinatorial problem.



(a) Greedy (initial) solution

(b) Best solution

Fig. 4: ATFM delay (min) for 09082019 PM

Table II also suggests that the total number of delayed flights is significantly reduced when optimising the greedy ATFM regulations scheme with ALNS. The number of flights that are delayed more than 30 min in the OR schemes is also lower when compared to the corresponding GR schemes, except for the 09082019 AM scenario - aside from the fact that the difference in this example is minor. The maximum and average delay per regulated flight are also improved when optimising the greedy ATFM regulations scheme with ALNS. In the best case (26072019AM), the maximum and average delay decrease from 84 and 22 min to 69 and 19 min, respectively. Following the same trend, the median delay is smaller for the OR schemes, except for the 09082019 AM scenario.

VI. Conclusions

This paper proposed an hyper-heuristic algorithm based on adaptive large neighbourhood search (ALNS) to construct from scratch the best plan of air traffic flow management (ATFM). The proposed algorithm could be used during the pre-tactical phase (D-1) as well as during the day of operations, provided that the Network Manager (NM) has access to accurate traffic demand and capacity information.

Results show that by iteratively removing regulations and regulating the resulting overloads again, the algorithm is able to determine where and when to regulate. This paper set the basic principles, but the authors encourage the research community to explore other configurations of the algorithm, e.g., a different pool of destroy and repair heuristics and/or acceptance criteria, as well as to perform a comprehensive comparison of the different settings. It would also be interesting to determine the value (if any) of conducting a local search whenever a best solution is discovered. Furthermore, this problem may be also solvable with machine learning techniques. Particularly attractive is the use of graph neural networks combined with state-of-the-art multi-agent reinforcement learning algorithms. Nodes could represent traffic volumes during specific time windows, and the actions would be binary: to regulate or not. The authors encourage researchers to explore this challenging yet scientifically interesting approach.

Currently, flow managers define the regulations in their area of responsibility. Furthermore, they are encouraged to provide as little ATFM delay as possible. Future work must also consider the practical implications of having a centralised authority manage the process of several flow managers. Another interesting study would be to examine the sensitivity of the solution, in terms of delay as well as overloads, to traffic fluctuations (specifically, the entry time of the flights at the various sectors). Actually, removing ATFM regulations may leave sectors vulnerable (i.e., unprotected) to traffic bunching if the actual trajectories differs from the planned ones. This potential issue must be considered in future work.

Acknowledgements

ISOBAR project has received funding from the SESAR Joint Undertaking (JU) under grant agreement No 891965. The JU receives support from the European Union's Horizon 2020 research and innovation programme and the SESAR JU members other than the Union.

References

[1] *ATFCM Users Manual*, EUROCONTROL, November 2018, rev. 22.1.
[2] R. Dalmau, B. Genestier, C. Anoraud, P. Choroba, and D. Smith, "A Machine Learning Approach to Predict the Evolution of Air Traffic Flow Management Delay," in *14th USA/Europe Air Traffic Management Research and Development Seminar*, 2021.
[3] R. Dalmau, Z. Zerrouki, C. Anoraud, D. Smith, and B. Cramet, "Are All the Requested Air Traffic Flow Management Regulations Actually Indispensable?" in *11th SESAR Innovation Days (SID)*, 2021.
[4] Network Manager, "Network Operations Report 2020," EUROCONTROL, Brussels, Belgium, Tech. Rep., April 2021.

[5] F. Serrano and A. Kazda, "The future of airports post covid-19," *Journal of Air Transport Management*, vol. 89, p. 101900, 2020.

[6] Network Manager, "Forecast Update 2021-2024. European Flight Movements and Service Units. Three Scenarios for Recovery from COVID-19," EUROCONTROL, Tech. Rep., May 2021.

[7] N. Ivanov, F. Netjasov, R. Jovanović, S. Starita, and A. Strauss, "Air traffic flow management slot allocation to minimize propagated delay and improve airport slot adherence," *Transportation Research Part A: Policy and Practice*, vol. 95, pp. 183–197, 2017.

[8] T. Bolić, L. Castelli, L. Corolli, and D. Rigonat, "Reducing atfm delays through strategic flight planning," *Transportation Research Part E: Logistics and Transportation Review*, vol. 98, pp. 42–59, 2017.

[9] P. Pellegrini, T. Bolić, L. Castelli, and R. Pesenti, "Sosta: An effective model for the simultaneous optimisation of airport slot allocation," *Transportation Research Part E: Logistics and Transportation Review*, vol. 99, pp. 34–53, 2017.

[10] T. Bolić, L. Castelli, G. Scaini, G. Frau, and S. Guidi, "Flight flexibility in strategic traffic planning: visualisation and mitigation use case," *CEAS Aeronautical Journal*, vol. 12, no. 4, pp. 847–862, 2021.

[11] Y. Xu, R. Dalmau, M. Melgosa, A. Montlaur, and X. Prats, "A framework for collaborative air traffic flow management minimizing costs for airspace users: Enabling trajectory options and flexible pre-tactical delay management," *Transportation Research Part B: Methodological*, vol. 134, pp. 229–255, 2020.

[12] L. Delgado, G. Gurtner, T. Bolić, and L. Castelli, "Estimating economic severity of air traffic flow management regulations," *Transportation Research Part C: Emerging Technologies*, vol. 125, p. 103054, 2021.

[13] S. Ruiz, H. Kadour, and P. Choroba, "An innovative safety-neutral slot overloading technique to improve airspace capacity utilisation," in *SESAR Innovation Days (SID) 2019*, Athens, Greece, December 2019.

[14] D. Pisinger and S. Ropke, *Large Neighborhood Search*. Boston, MA: Springer US, 2010, pp. 399–419.

[15] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *Principles and Practice of Constraint Programming*, M. Maher and J.-F. Puget, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 417–431.

[16] D. Wolfinger, "A large neighborhood search for the pickup and delivery problem with time windows, split loads and transshipments," *Computers & Operations Research*, vol. 126, p. 105110, 2021.

[17] F. Glover and J.-K. Hao, "The case for strategic oscillation," *Annals OR*, vol. 183, pp. 163–173, 03 2011.

[18] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2033–2049, 2007.

[19] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck, "Record breaking optimization results using the ruin and recreate principle," *Journal of Computational Physics*, vol. 159, no. 2, pp. 139–171, 2000.

[20] A. Santini, S. Røpke, and L. M. Hvattum, "A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic," *Journal of Heuristics*, vol. 24, no. 5, pp. 783–815, 2018.

[21] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation Science*, vol. 40, pp. 455–472, 11 2006.

[22] I. Dayarian, T. G. Crainic, M. Gendreau, and W. Rei, "An adaptive large-neighborhood search heuristic for a multi-period vehicle routing problem," *Transportation Research Part E: Logistics and Transportation Review*, vol. 95, pp. 95–123, 2016.

[23] R. Lutz, "Adaptive large neighborhood search," Master's thesis, Ulm University, 2014.

[24] C. Verlhac, A. Schweitzer, E. Dumont, and S. Manchon, "Improved configuration optimizer," EUROCONTROL Experimental Centre, Tech. Rep. 2005-016EEC Note 2005/11, 2005.

[25] P. B. M. Vranas, "Optimal slot allocation for european air traffic flow management," *Air Traffic Control Quarterly*, vol. 4, no. 4, pp. 249–280, 1996.