Simulated-Annealing Hyper-Heuristic for Demand-Capacity Balancing in Air Traffic Flow Management

Ahmed Khassiba, Daniel Delahaye

Simulated-Annealing Hyper-Heuristic for Demand-Capacity Balancing in Air Traffic Flow Management

Ahmed Khassiba
Capgemini Engineering, TEC
4 avenue Didier Daurat
31700 Blagnac, France
E-mail: ahmed.khassiba@capgemini.com

Daniel Delahaye
ENAC
7 avenue Edouard Belin
31400 Toulouse, France
E-mail: delahaye@recherche.enac.fr

***Abstract*—Very critical convective weather leads to sharp drop in air-traffic en-route sector capacity keen on creating severe demand-capacity imbalances, called also hotspots. Following the knock-on effect, these imbalances spread across the network, causing the so-called netspots. The problem of demand-capacity hotspot/netspot mitigation with demand-side measures aims at solving these imbalances using ground delays and reroutings, while minimizing the total delay at arrival. A simulated-annealing hyper-heuristic relying on efficient neighborhood operators is proposed. A study on the combination of different delay-based and rerouting-based neighborhood operators is conducted. The best combination is used to solve a large-scale and challenging instance, in a relatively short computation time.**

***Keywords*—demand-capacity balancing, hyper-heuristics, simulated annealing**

I. Introduction

Up to 2019 just before Covid-19 crisis, "ATFM en-route" has been the second most important cause of delay (after airlines operations) in air transport in Europe [1], with the highest delays being recorded in summer 2018. The upturn in air traffic, more recently, paves the way unfortunately for delay figures to soar again. Delays due to "ATFM en-route" are typically those impacting flights take-off time, when en-route sectors cannot accommodate the high air traffic demand. A sector capacity can fall during some time intervals due to local critical convective weather conditions (*e.g.,* storms), which consequently, creates an imbalance between demand and capacity. An imbalanced sector during a given period of time, *e.g.,* a *sector-hour*, is called a *demand-capacity hotspot*. In the current operational process, on the day of operations, Flow Manager Positions (FMPs) use Computer-Assisted Slot Allocation (CASA) system to mitigate such hotspots, by defining *regulations* for the imbalanced sectors. CASA issues new (delayed) take-off times for flights that are impacted by regulations. Following the *knock-on* effect, *regulated* flights propagate delay through the network, by creating new demand-capacity hotspots, hundreds of nautical miles far from the convective weather zone, as observed in summer 2018. The aggregation of demand-capacity hotspots spread across the network, *e.g.*, the European airspace, created by the knock-on effect, while their origin is a local convective weather, is called a *netspot*. The use of conventional CASA regulations reveals inefficient to solve netspots as excessively large delays are generated, especially for flights that are captured by several regulations. Several improvements of the current operating concepts in air traffic flow management have been identified to "hedge" against the impact of very-critical weather conditions on the air traffic situation at the network level. In this context, SESAR-H2020-funded project ISOBAR (for Artificial Intelligence Solutions to Meteo-Based DCB Imbalances for Network Operations Planning) seeks to leverage Artificial-Intelligence techniques to better predict high-risk convective weather and sector capacity drop, to identify and to efficiently mitigate hotspots at the network level, *i.e.*, netspots, both at pre-tactical (one day before operations) and tactical (the day of operations) stages.

This paper focuses on hotspot mitigation at the pre-tactical stage, defined as the problem of accommodating air traffic through the airspace, by delaying take-off times and changing planned routes, in order to meet reduced sectors capacity, while minimizing deviations with respect to the original flight plans. A hyper-heuristic algorithm using an efficient combination of neighborhood operators is proposed as a solution method. In the following Subsection I-A, the general principles of hyper-heuristics are presented. The scope of the paper is presented in Subsection I-B.

A. Hyper-heuristics

Search algorithms called *metaheuristics* are approximate solution methods that iteratively search for a good-quality solution to a given optimization problem, while trying to escape local optima. Metaheuristics operate directly on the solution space, and keep track of either: a single solution (*e.g.*, simulated annealing, tabu search, etc), or a population of solutions (*e.g.*, genetic algorithm, etc). On the other hand, *hyper-heuristics* are approximate optimization algorithms with a higher level of automation than metaheuristics. They operate on a space of low-level heuristics (each of which operates directly on the solution space), and use a strategy for either

selecting or *generating* low-level heuristics, throughout the iterations. For an extensive literature review on hyper-heuristics, the reader is redirected to [2], [3]. A *selection-based* hyper-heuristic relies on: (1) a pool of low-level heuristics, (2) a strategy to select the low-level heuristic for the next iteration, and (3) a strategy to update low-level heuristics scores. Low-level heuristics (LLH), called also *neighborhood operators* in the metaheurisitcs vocabulary, are procedures that operate on the solution space in order to generate a new solution from a current one. Low-level heuristics are problem-specific, and can be classified either as diversification or as intensification operators. Very often, an initial score is given to each low-level heuristic, then is updated throughout the iterations based on each LLH performance. There are several strategies to select the LLH to apply for the next iteration, among them: the max strategy, and the roulette-wheel strategy. The max strategy selects always the LLH with the best score, while the roulette-wheel randomly selects LLHs with probabilities that are proportional to their scores. Hyper-heuristics have been applied successfully to many optimization problems; to name a few recent applications in air traffic management: [4], [5].

B. Scope of the paper

In this paper, we consider an optimization problem similar to the one presented in [6], where demand-capacity hotpsots are to be mitigated using two types of demand-side measures: ground delay, and alternative trajectories selection. Alternative trajectories are assumed to be agreed on between airspace users and the Network Manager at a strategic level. We focus on situations of very-critical convective weather, leading to sharp en-route sector capacity drop with respect to nominal values. We opt for entry count as the capacity metric, since finer-grained metrics, such as occupancy count [7] or geometrical complexity [8], cannot be used with a lookahead time of 24 hours before operations, due to high uncertainty. We propose a simulated-annealing-based hyper-heuristic using efficient low-level heuristics relying on the concept of *flight overload factor*. Recently, a similar concept has been applied successfully to airspace congestion mitigation using departure slots assignment [9]. We conduct a numerical study to identify the best combination of neighborhood operators, the best strategy to select them, and the best strategy to update their score, at every iteration.

The remainder of this paper is organized as follows. Section II presents more formally the demand-capacity-balancing problem, with demand-side mitigation measures. Mathematical notations are summarized in Table I. The solution method based on a simulated-annealing hyper-heuristic is described in Section III. The numerical study is reported in Section IV. Section V concludes this paper.

TABLE I. NOTATIONS SUMMARY

Notation	Definition
$\mathcal{F}$	Set of flight indices
$\bar{\mathcal{F}} \subset \mathcal{F}$	Set of indices of flights eligible for hotspot mitigation measures
$\bar{\mathcal{F}}^{\max}$	Set of indices of highly-overload flights
$\mathcal{P}$	Set of counting periods
$\mathcal{S}$	Set of sectors of interest
$\mathcal{S}_f \subset \mathcal{S}$	Set of sectors crossed by flight $f \in \mathcal{F}$
$\mathcal{T}_f$	Set of alternative trajectories for flight $f \in \bar{\mathcal{F}}$
P	Number of counting periods in $\mathcal{P}$
$\underline{t}_p$	Start time of counting period p
$\bar{t}_p$	End time of counting period p
m	Time shift between two consecutive time periods
l	Length of one time period
C_s^p	Capacity of sector s during period p
D_s^p	Demand of sector s during period p
O_s^p	Overload of sector s during period p
$T_f \in \mathcal{T}_f$	Selected trajectory decision variable for flight $f \in \bar{\mathcal{F}}$
T_f^{org}	Original trajectory for flight $f \in \mathcal{F}$
$t_{f,0}$	Take-off time decision variable for flight $f \in \bar{\mathcal{F}}$
$t_{f,0}^{org}$	Original take-off time for flight $f \in \mathcal{F}$
$t_{f,s}$	Entry time of flight $f \in \mathcal{F}$ to sector $s \in \mathcal{S}$
$t_{f,*}$	Arrival time for flight $f \in \bar{\mathcal{F}}$
$t_{f,*}^{org}$	Original arrival take-off time for flight $f \in \mathcal{F}$
$d_f^{\max}$	Maximum possible take-off delay for flight $f \in \bar{\mathcal{F}}$
o_f	Flight overload factor for flight $f \in \mathcal{F}$
$o_{\max}$	Maximum flight overload factor over all flights in $\mathcal{F}$
v	Overload tolerance threshold
$T = (T_f)_{f \in \bar{\mathcal{F}}}$	Vector of the selected trajectories decision variables,
$t_0 = (t_{f,0})_{f \in \bar{\mathcal{F}}}$	Vector of the take-off times decision variables,
w_1	Weight for the total arrival delay term
w_2	Weight for the total unsustainable overload term
$w \in (0, 1]$	User-defined coefficient in roulette-wheel selection
$\alpha \in (0, 1)$	Scale factor of the cooling schedule
β	Minimum acceptance ratio threshold during cooling

II. PROBLEM STATEMENT

Consider a time horizon decomposed into P equal-length, possibly overlapping, time periods. Each time period $p \in \mathcal{P} = \{1, 2, \ldots, P\}$ has a start time $\underline{t}_p$ and an end time $\bar{t}_p > \underline{t}_p$, such that: $\bar{t}_p - \underline{t}_p = l > 0$, where l is the time-period duration or *length*; the same for all time periods, *e.g.*, $l = 60$ minutes. We have also: $\underline{t}_{p+1} - \underline{t}_p = m > 0$ and $m \leq l$, where m is the time shift, or *move*, between two consecutive time periods such that $p \in \{1, 2, \ldots, P-1\}$. Typically, $m = 20$ minutes.

Consider a set of S sectors indexed by $s \in \mathcal{S}$. Let $C_s^p \geq 0$ denote the capacity of sector s during time period $p \in \mathcal{P}$, expressed as the maximum number of flight entries to sector s during time interval $[\underline{t}_p, \bar{t}_p)$, that can be handled by air traffic controllers (ATCOs). During a given time period $p \in \mathcal{P}$, a sector $s \in \mathcal{S}$ can be either *open* or *closed*. An *open* sector is controlled by a couple of ATCOs, hence it has a positive capacity $C_s^p > 0$. When sector s is grouped with other sectors into a larger *collapsed* sector, it is no more monitored on its own, while the larger sector is. In this case, sector s is said to be *closed* during time period p, and its capacity is set to $C_s^p = \infty$.

Consider a set of flights indexed by $f \in \mathcal{F}$. Each flight f has an expected take-off time (commonly abbreviated ETOT) $t_{f,0}^{org}$, and an original trajectory T_f^{org}. A trajectory is represented as a sequence of entry times to a subset of sectors. Assume a flight f crossing a subset of sectors $\mathcal{S}_f \subset \mathcal{S}$ following a trajectory T_f, then:

$$T_f = \{(t_{f,s}, s), s \in \mathcal{S}_f\} \tag{1}$$

where $t_{f,s}$ is the entry time of flight f to sector s. Let $\mathcal{F}_s^p$ denote the set of flight indices that enter sector s during period p:

$$\mathcal{F}_s^p = \{f \in \mathcal{F} \mid \quad s \in \mathcal{S}_f \quad \text{and} \quad t_{f,s} \in [\underline{t}_p, \bar{t}_p)\} \tag{2}$$

Let D_s^p denote the demand of sector s during time period $p \in \mathcal{P}$, expressed as the number of flight entries to sector s during the time interval $[\underline{t}_p, \bar{t}_p)$. We establish that:

$$D_s^p = \begin{cases} |\mathcal{F}_s^p| & \text{if sector } s \text{ is open during period } p \\ 0 & \text{if sector } s \text{ is closed during period } p \end{cases} \tag{3}$$

When demand exceeds capacity in a given sector s during a time period p, *i.e.*, $D_s^p > C_s^p$, we say that sector s is *overloaded* during period p. The overload of open sector s during period p is expressed as:

$$O_s^p = \begin{cases} \dfrac{D_s^p - C_s^p}{C_s^p} & \text{if sector } s \text{ is open during period } p \\ 0 & \text{if sector } s \text{ is closed during period } p \end{cases} \tag{4}$$

In practice, small overloads can still be managed by ATCOs. Hence, we define $v \in [0, 1)$ an overload-tolerance threshold, set by the user. A typical value for overload tolerance is $v = 10\%$. A demand-capacity *hotspot* occurs in a sector s during period p when: $D_s^p > C_s^p \times (1 + v)$, *i.e.*, $O_s^p > v$. In order to *mitigate* hotspots, that is to ensure that demand and capacity are balanced, flight departure times can be delayed and original trajectories can be changed.

We consider $\bar{\mathcal{F}} \subset \mathcal{F}$ a subset of flights that can be modified, *i.e.*, delayed or rerouted, while flights in $\mathcal{F} \backslash \bar{\mathcal{F}}$ are considered as fixed. For example, in Europe, flights eligible for modification are those departing from European airports, *i.e.*, typically their airport code starts with L, E, GC, or GM. However, flights planned to depart from America or Asia and to use the European airspace cannot be modified by the Network Manager Operating Center. We assume that every flight in $\bar{\mathcal{F}}$ has a set of alternative trajectories $\mathcal{T}_f \ni T_f^{org}$.

The Demand-Capacity Balancing (DCB) problem in ATFCM, defined for a set of counting periods $\mathcal{P}$ and set of sector indices $\mathcal{S}$, consists in finding for every flight $f \in \bar{\mathcal{F}}$, an adjusted take-off time $t_{f,0} \geq t_{f,0}^{org}$, and a trajectory $T_f \in \mathcal{T}_f$, such that all demand-capacity hotspots are mitigated, *i.e.* $O_s^p \leq v$ for all $(s, p) \in \mathcal{S} \times \mathcal{P}$, while minimizing the total delay at *arrival*, incurred by all flights. Moreover, the take-off delay per flight may be bounded by $d_f^{\max}$ for every flight $f \in \bar{\mathcal{F}}$. The complete model of the demand-Capacity Balancing (DCB) problem in ATFCM reads:

$$\min_{t_0, T} \quad \sum_{f \in \bar{\mathcal{F}}} (t_{f,*} - t_{f,*}^{org}) \tag{5}$$

$$\text{s.t.} \quad O_s^p \leq v \qquad (s, p) \in \mathcal{S} \times \mathcal{P} \tag{6}$$

$$t_{f,0} - t_{f,0}^{org} \in \left[0\,,\, d_f^{\max}\right] \qquad f \in \bar{\mathcal{F}} \tag{7}$$

$$T_f \in \mathcal{T}_f \qquad f \in \bar{\mathcal{F}} \tag{8}$$

where $t_{f,*}^{org}$, and $t_{f,*}$ denote respectively the original, and the adjusted, arrival time of flight f. These arrival times can be computed directly from take-off times, $t_{f,0}$ and $t_{f,0}^{org}$, using the flight duration of the corresponding trajectories.

Relaxed DCB problem

Finding a feasible solution to the DCB problem is not straightforward, especially when the initial overload situation is too critical. For this reason, we consider the relaxed problem where capacity constraints (6) are integrated into the objective function, as follows:

$$\min_{t_0, T} \; w_1 \sum_{f \in \bar{\mathcal{F}}} (t_{f,*} - t_{f,*}^{org}) + w_2 \sum_{(s,p) \in \mathcal{S} \times \mathcal{P}} \max{}^{+} (O_s^p - v)$$

s.t. Constraints (7) and (8)

(Relaxed_DCB)

where w_1 and w_2 are user-defined weights, and $\max^{+}(\cdot) = \max(0, \cdot)$. Note that for the relaxed problem, the solution with no delay at departure and no change in trajectories, is a feasible solution. We call the term $\sum_{(s,p) \in \mathcal{S} \times \mathcal{P}} \max{}^{+} (O_s^p - v)$ the *total unsustainable overload*. A feasible solution of the relaxed problem (Relaxed_DCB) that has a null total unsustainable overload is a feasible solution of the original problem, given by equations (5) – (8). We define the *best* solution as the one that has, most importantly, the lowest total unsustainable overload (ideally no unsustainable overload). Then, for two solutions with equal total unsustainable overload, the one with the smallest total arrival delay is the better.

III. Simulated-annealing hyper-heuristic

Simulated annealing (SA) is a well-known metaheuristic that mimics the annealing process from metallurgy [10]. While classical SA relies on a single neighborhood operator, we propose a simulated-annealing-based hyper-heuristic to solve the relaxed version of the DCB problem (Relaxed_DCB), using a pool of neighborhood operators, designed using the idea of *flight overload factor*. Each neighborhood operator is given a performance score, that is updated along the iterations according to predefined rules. At every iteration, a neighborhood operator is selected according to specific strategies. In our study, we evaluate different rules to update the performance scores of neighborhood operator, and different strategies to select the neighborhood operator at every iteration. In Subsection III-A, we briefly review SA applied to our problem. The hyper-heuristic procedure is presented in Subsection III-B. The pool of neighborhood operators is presented in Subsection III-C. The studied rules to update the operators' score and the strategies to select them are presented in Subsection III-D.

A. Simulated annealing

SA starts with an initial solution, represented by a vector of decisions. In our problem, each decision specifies, for each flight, the on-ground delay and the trajectory, *i.e.* $\left(t_{f,0} - t_{f,0}^{\text{org}}\,,\, T_f\right)$. Using a predefined neighborhood operator, the current solution is evolved into a new one, and evaluated in terms of objective function. The acceptance of the new

solution is driven by the Metropolis acceptance criterion [11], that stipulates that an improving solution is always accepted, while a non-improving solution is accepted with a probability controlled by an internal parameter, called the *temperature*. Assuming obj^{new} and obj^{curr} are the objective function values of the new and the current solutions respectively, and Temp the temperature parameter, the acceptance probability, AP, according to the Metropolis criterion, in the minimization case, reads:

$$AP = \begin{cases} 1 & \text{if} \quad \text{obj}^{\text{new}} < \text{obj}^{\text{curr}}, \\ e^{-(\text{obj}^{\text{new}} - \text{obj}^{\text{curr}})/\text{Temp}} & \text{otherwise.} \end{cases}$$

In SA, the temperature parameter is initialized, either statically or dynamically through a *heat-up procedure*, then it decreases following a *cooling schedule*. The Metropolis criterion ensures that as the temperature decreases, the probability of acceptance of non-improving solutions converges to zero, which allows for exploration at the early search stages, and exploitation at the late ones. Note that a too fast cooling schedule could lead to pre-mature convergence of the search to a local optimum, while a too slow cooling may extend the computing time. Re-heating or temperature reset are also common strategies to help the search escape from local optima. Several strategies to set the initial temperature, as well as to conduct cooling, and reheating or temperature reset, are proposed in [12].

B. Hyper-heuristic procedure

Algorithm 1 Hyper-heuristic procedure

```
1:  Initialize neighborhood operators' score
2:  Initialize the temperature
3:  Initialize the current solution
4:  while stopping criterion is not met do
5:      for a number of iterations do
6:          Select a neighborhood operator
7:          Generate a new neighboring solution
8:          if Metropolis acceptance criterion is met then
9:              Update the current solution
10:         end if
11:         Update the selected neighborhood operator's score
12:     end for
13:     Temperature cooling or reheating
14: end while
```

We propose a hyper-heuristic algorithm based on simulated annealing, as sketched in Algorithm 1. In the initialization phase (lines 1–3), we assign and initialize a performance score to each neighborhood operator, we compute the initial temperature using a heat-up procedure (see Subsection III-B1), and we set the current solution to the "no delay, no rerouting" solution. At the beginning of each round of iterations, a neighborhood operator is selected according to a predefined strategy (line 6), then used to generate a new solution (line 7). The Metropolis criterion is applied to decide whether to accept the new solution or not (lines 8–10). Subsequently, the performance score of the selected neighborhood operator is updated according to predefined rules (line 11). After every round of iterations, the temperature parameter is revised, either cooled or re-heated (line 13), as described in Subsection III-B2. The algorithm stops either when the temperature is too low, or when a predefined time limit is exceeded (line 4).

1) Heat-up procedure: We set the temperature to a first value, for example, computed as the maximum difference between two neighbor solutions [10]. We generate a random solution. Then, it is evolved during nb_iter iterations using neighborhood operators, where nb_iter is a user-defined parameter. The heat-up procedure is stopped if the acceptance ratio over the last nb_iter iterations is greater than or equal to a predefined target value (typically 99%). Otherwise, the temperature is increased by a heat-up factor (*e.g.,* by a factor of 2), and a new round of neighboring solution generation is restarted for another nb_iter iterations.

2) Cooling and re-heating schedule: We opt for a geometric progression with a scale factor $\alpha \in (0, 1)$ for temperature cooling; $\text{Temp}^{i+1} = \text{Temp}^{i} \times \alpha$, where Temp^{i} and Temp^{i+1} denote the temperature values for two successive iterations. We define β the minimum acceptance ratio threshold over nb_iter iterations, typically $\beta = 90\%$. During the cooling process, if the acceptance ratio over the last nb_iter iterations is below β, then we suspend the cooling schedule. The temperature is kept at the same value for additional nb_iter_add iterations. The cooling schedule is resumed when either the acceptance ratio is greater than or equal to β, or a predefined maximum number of iterations at the same temperature, denoted max_nb_iter, is reached. This procedure has the merit to avoid too early convergence without causing divergence, unlike re-heating schemes where the temperature is simply increased.

C. Neighborhood operators based on flight overload factor

In order to design an efficient hyper-heuristic, one needs to design well-performing neighborhood operators tailored to the problem at hand. For the DCB problem with demand-side decisions, one key idea is to modify flights that are very much involved in hotspots. To measure the degree of involvement of individual flights in the overall demand-capacity overload situation across the region of interest, we define the *flight overload factor* $o_f \geq 0$, for a given flight $f \in \mathcal{F}$, as the sum of all fractions of unsustainable overload of the sector-periods crossed by f. It reads:

$$o_f = \sum_{s \in \mathcal{S}_f} \sum_{p \in \mathcal{P} \ | t_{f,s} \in [\underline{t}_p, \bar{t}_p)} \max{}^{+} (O_s^p - v) \tag{9}$$

The definition of flight overload factor not only takes into account the number of unsustainably overloaded sector-periods crossed by a given flight but also the amplitude by which the overload tolerance is exceeded in each sector-period. Note that a flight f with a null overload factor, *i.e.,* $o_f = 0$, is a one that does not cross any hotspot. We shall call a flight with a positive overload factor, *i.e.*, $o_f > 0$, an *overloaded* flight. The neighborhood operators should focus on flights with high overload factor, by either delaying them, to overstep the overloaded time periods, or by rerouting them, to hopefully deviate from some overloaded sectors. To allow a flexible

selection among highly overloaded flights, we rely on roulette-wheel selection based on flights overload factor, as follows. First, the largest flight overload factor, $o_{\max}$, is found. Then, all flights with an overload factor within $[w \times o_{\max}\ ,\ o_{\max}]$ are shortlisted, where $w \in (0, 1]$ is a user-defined coefficient. Finally, a roulette-wheel selection is applied on the shortlisted flights where each flight's selection probability is proportional to its own overload factor, as described in Algorithm 2.

Algorithm 2 Roulette-wheel selection based on flight overload factor

Require: $w \in (0, 1]$ a user-defined coefficient
1: Find the largest flight overload factor, $o_{\max}$
2: Select all flights with an overload factor within $[w \times o_{\max}\ ,\ o_{\max}]$, and store them in $\bar{\mathcal{F}}^{\max}$.
3: Let: $\text{prob}_f = \frac{o_f}{\sum_{k \in \bar{\mathcal{F}}^{\max}} o_k}$, for every $f \in \bar{\mathcal{F}}^{\max}$
4: Apply a roulette-wheel selection among the selected flights where prob_f is the selection probability of flight $f \in \bar{\mathcal{F}}^{\max}$.

The proposed neighborhood operators are summarized in Table II. The operators from DO-0 through DO-2 apply delay, while operators from RO-1 and RO-2 apply rerouting. The neighborhood operator LS is a local search procedure, that can be applied directly after any DO-x operator, so that the delay assigned to the flight selected in DO-x can be locally optimized. When applied in a given iteration, DO-x operators assign a new delay to a selected flight, regardless its last delay value, from the previous iterations. Hence, DO-x can be considered as diversification operators, while LS can be seen as an intensification operator. DO-2 and RO-2 are further described in Algorithms 3 and 4 respectively.

TABLE II. NEIGHBORHOOD OPERATORS

Identifier	Neighborhood operator description
DO-0	Delay a random flight
DO-1	Delay the most overloaded flight
DO-2	Delay a highly-overloaded flight
LS	Search for the best delay of a given flight, around a given delay value
RO-1	Reroute the most overloaded flight
RO-2	Reroute a highly-overloaded flight

Algorithm 3 DO-2 : Delay a highly-overloaded flight

Require: $d^{\min}$ and $d^{\max}$
1: Select a flight using the roulette-wheel procedure defined in Algorithm 2
2: Apply a random delay between $d^{\min}$ and $d^{\max}$ to the selected flight.

Algorithm 4 RO-2 : Reroute a highly-overloaded flight

1: Select a flight using the roulette-wheel procedure defined in Algorithm 2
2: Randomly select an alternative trajectory for the selected flight.

D. Strategies for neighborhood operators management

We study two strategies to update neighborhood operators scores:

- **Additive:** if the new solution improves the objective function's value, the score is incremented by 1. Otherwise, the score is decremented by 1, while ensuring that the minimum score is 1.
- **Customized:** if the new solution improves the objective function's value, the score is incremented by 1. Otherwise, the score is square-rooted.

We study two strategies to select the next neighborhood operator:

- **Max strategy:** the neighborhood operator with the highest score is systematically selected.
- **Roulette-wheel strategy:** neighborhood operators are selected with probabilities proportional to their scores.

IV. NUMERICAL STUDY

We conduct a numerical study to identify the best combination of neighborhood operators, the best rule to update neighborhood operators' score, and the best strategy to select the next operator. First, we test the three delay-based neighborhood operators, with and without the local search procedure, introduced in Subsection III-C. The best delay-based operator is the one that solves (ideally) all unsustainably-overloaded sector-periods, while generating the smallest total delay. After selecting the best delay-based neighborhood operator, combinations with rerouting operators are tested, assuming that at every iteration of the hyper-heuristic, the two neighborhood operators are selected with a 50% probability each. The best combination between delay-based and rerouting-based operators is selected to test the rules to update neighborhood operators' score, and the best strategy to select the next operators, among those introduced in Subsection III-D.

Details about the studied instance and the parameter values is given Subsection IV-A, and Tables III and IV therein. Results of delay-based neighborhood operators are presented in Subsection IV-B. Results of the combination of the best delay-based neighborhood operator and rerouting operators are given in Subsection IV-C. Rules to update neighborhood operators' score and strategies to select neighborhood operators are studied in Subsection IV-D. Finally, more insights are given in Subsection IV-E about the best solution obtained in terms of the overload situation, the issued delay and rerouting. The proposed hyper-heursitic was developed in JAVA programming language, and all results are obtained on a Windows 11 platform, with an Intel(R) Core(TM) i5 CPU @ 1.60GHz and 12 GB RAM. The maximum time limit is set to 5 minutes.

A. Instances and parameter values

We focus on July 27th 2019, where very high delay was recorded due to very convective weather in Europe. All elementary and collapsed sectors in ACCs: Karlruhe (EDUUUAC), Zagreb (LDZOACC), Barcelona (LECBACC), Marseille (LFMMACC), and Vienna (LOVVACC) are considered. Eurocontrol DDR2 traffic files, so6 and t5, are used. Entry times to sectors are retrieved from t5 files, while meta data on flights (callsigns, aircraft operators, etc) are taken from so6 files. Predicted capacity data was generated by machine

learning, and provided as an input from ISOBAR consortium partners. Details on the instance size and the initial overload situation are provided in Table III. The hyper-heuristic parameter values are given in Table IV.

Alternative trajectories: To retrieve a set of alternative trajectories for each flight, we analyse the so6 and t5 files for the day of interest, and we group all trajectories by four criteria: origin (O), destination (D), aircraft type (AC) and airspace user (AU). Then, for every set of trajectories sharing the same values for the four criteria, we eliminate *equivalent* trajectories. Two trajectories are assumed to be equivalent if they cross the same sectors in the same order. The final sets of non-equivalent trajectories grouped by (O, D, AC, AU) are used to fill in the set of alternative trajectories for every flight. It is noteworthy that the merit of the retrieved alternative trajectories is twofold: they are realistic, and they integrate the airspace user's dimension.

TABLE III. INSTANCE DETAILS

Day of traffic:	
Nb counting periods ($l = 60$ min, $m = 20$ min)	72
Nb sectors of interest	175
Nb flights crossing airspace of interest	12 224
Nb flights eligible for hotspot mitigation measures (departing from airports: L* E* GC* GM*)	10 924
Predicted overload situation:	
Maximum overload per sector-period	56%
Total unsustainable overload	48.84
Nb overloaded sector-periods	522
Nb unsustainably-overloaded sector-periods	330

TABLE IV. PARAMETER VALUES

Parameter	Value	Parameter	Value
α	0.99	w_1	1.0
β	0.90	w_2	2000.0
w	0.2	v	10%
nb_iter	200	$d_f^{\max}$	180 min
nb_iter_add	50		
max_nb_iter	400		

B. Results of delay-based neighborhood operators

Results of the three delay-based neighborhood operators, with and without the local search procedure, are given in Table V and Figure 1. Figure 1 gives the evolution of the best solution's total unsustainable overload (upper figure) and total delay (lower figure) over iterations, for the 6 delay-based neighborhood operators. Note that total unsustainable overload of the best solution decreases over the iterations, for all neighborhood operators. DO-2 and DO-2+LS exhibit the fastest decrease, while DO-1 and DO-1+LS are stuck, very early, at a large value of the total unsustainable overload. The best solution with random delay operators, DO-0 and DO-0+LS, has a slow decrease in total unsustainable overload, but still ends at a low value. In terms of total delay, the best solutions using DO-1 and DO-1+LS present the lowest delay. DO-0 and DO-0+LS have a bell-shaped total delay evolution. For DO-0, after a dramatic increase, the total delay decreases significantly to reach competitive values with DO-2 and DO-2+LS. However, DO-0+LS reveals to be much slower (due to the additional local search at every iteration) than DO-0, hence, it has not enough time to decrease its total delay figure. Delaying systematically the most overloaded flight (DO-1) at every iteration reveals to be inefficient, since the total unsustainable overload is only slightly decreased. Clearly, highly overloaded flights are the best candidates to mitigate hotspots. The first expected effect of the additional local search procedure is to extend the time to accomplish one iteration, and hence to reduce the number of iterations made within the time limit of 5 minutes. Nevertheless, a better performance of DO-1+LS and DO-2+LS was achieved compared to their simple counterparts, in less iterations. All in all, the best delay-based neighborhood operator is DO-2+LS, since it mitigates all unsustainable overloads, while generating the smallest total delay. In the next subsection, we present the results of the combination of DO-2+LS with the two proposed rerouting operators RO-1 and RO-2.

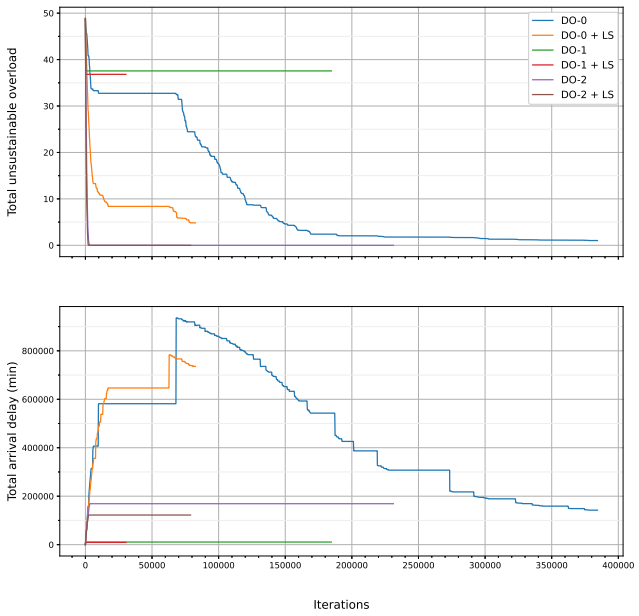


Figure 1. Evolution of the best solution's total unsustainable overload (upper figure) and total delay (lower figure) over iterations for the 6 single neighborhood operators based on delay. The same time limit of 5 minutes was set.

C. Results of the combination of delay and rerouting operators

Table VI shows the total residual unsustainable overload, as well as information on the total delay, the number of delayed flights, and the number of rerouted flights (column "# rerouted flights"), for the two combinations of DO-2+LS with RO-1 and RO-2. Since both tests reached the time limit of 5 minutes, the "CPU" column was omitted. We notice that using rerouting operators decreases significantly the total arrival delay, as

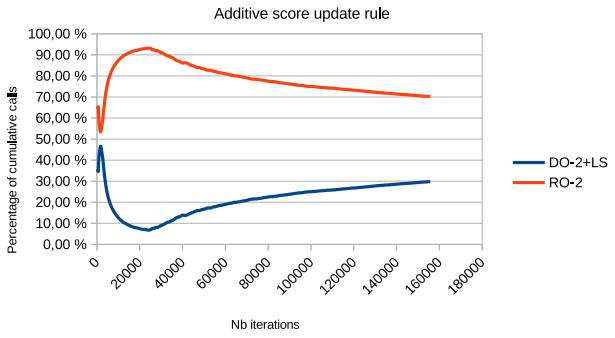
TABLE V. RESULTS FOR THE DELAY-BASED NEIGHBORHOOD OPERATORS. TILIM STANDS FOR TIME LIMIT = 5 MINUTES.

Neigh. Op.	CPU (sec)	Res. unsus. overload	Total delay (min)	# delayed flights	Avg. delay (min)	Max delay (min)
DO-0	215.7	1.03	142 346	9980	14.3	178
DO-1	TiLim	37.56	10 775	104	103.6	177
DO-2	TiLim	0	168 980	1850	91.3	180
DO-0+LS	TiLim	4.83	736 058	9633	76.4	180
DO-1+LS	TiLim	36.82	9 172	104	88.2	178
DO-2+LS	TiLim	0	122 205	1403	87.1	180

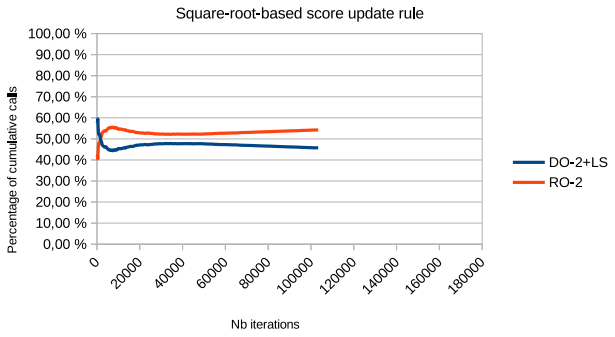
expected. Moreover, using a roulette wheel over flights with a high overload factor performs better than automatically selecting the flight with the highest factor at every iteration. The best combination with rerouting operators is DO-2+LS+RO-2, where delay and rerouting operators are selected with a 50% fixed probability each, along the iterations. In the following subsection, we study the effect of the two proposed score update rules (additive, customised) and the two neighborhood selection strategies (max, roulette wheel).

D. Results for strategies to select neighborhood operators and to update their score

Table VII shows the results for the retained neighborhood operators DO-2+LS+RO-2 with the four possible combinations of score update rules and neighborhood operator selection strategies, in addition to the baseline case (denoted "No update, Roulette-Wheel") retained from Subsection IV-C. All tests reached the time limit of 5 minutes. Notice that the max rule to select neighborhood strategies lead to not calling the rerouting operator. On the other hand, updating the neighborhood scores, either with the additive or the square-root rule, improves on the case with no update. This reveals that at different search stages, delay and rerouting operators do not have the same performance, and the hyper-heuristic calls them with different probabilities. The percentage of cumulative calls to the two operators is plotted in Figures (2a) and (2b), for the additive and the custom score update rules respectively. The best combination as reported in Table VII is to update scores using the additive rule and to use the roulette-wheel for neighborhood operator selection. In the next subsection, we show more details on the solution returned by this best combination.



(a) Additive rule.



(b) Square-root-based rule.

Figure 2. Percentage of cumulative number of calls to delay and rerouting operators, with a roulette-wheel selection strategy, and different score-update rule.

E. Best solution details

The best combination of neighborhood operators (DO2-LS-RO2) with an additive score-update scheme, and a roulette-wheel selection over operators was applied again on the studied instance with a time limit of 1 minute. The same solution was returned, showing that the best solution is found in the early stages of the search and is not updated later. Also, this highlights the performance of the designed hyper-heuristic to find very-good quality solutions in a relatively short computation time. Some details of the best solution are given in Figures 3, 4a, and 4b. The distribution of overloads over sector-periods, before and after optimization, is given in Figure 3. All unsustainably overloaded sector-periods are mitigated, at the price of an increase in the overload ranges of $-10-0\%$ and $0-10\%$. The distribution of arrival delay, and of extra mileage are given in Figures 4a and 4b respectively. Flights with a negative arrival delay, correspond to rerouted flights, with alternative trajectories that are shorter than their planned trajectories. Although, such reroutings may be interesting to solve the DCB problem, in real life, airlines may not be keen on accepting such reroutings. Indeed, there are situations where the shortest trajectory yields a high operating cost for the airline, due to high overflight fees of some countries. Moreover, 24.73% of the rerouted flights do not cross the sectors considered for the DCB problem anymore. In this sense, they do not contribute to the load of the studied sectors, however, their new trajectories must increase the load of sectors beyond the region of interest. This suggests that neighbouring sectors should also be monitored for demand-

TABLE VI. RESULTS FOR THE COMBINATION OF THE TWO REROUTING NEIGHBORHOOD OPERATORS WITH THE RETAINED DELAY-BASED OPERATOR.

Neigh. Op.	Res. unsus. overload	Total Arrival delay (min)	# delayed flights	Avg. delay (min)	# rerouted flights
DO-2+LS+RO-1	0	102 950	1185	85.4	75
DO-2+LS+RO-2	0	88 761	1177	75.4	181

TABLE VII. RESULTS FOR THE STRATEGIES TO SELECT NEIGHBORHOOD OPERATORS AND TO UPDATE THEIR SCORES.

Score-update rule	Neigh. Op. selection	Res. unsus. overload	Total Arrival delay (min)	# delayed flights	Avg. delay (min)	# rerouted flights
No update	Roulette-Wheel	0	88 761	1116	75.4	181
Additive	Roulette-Wheel	0	73 893	985	68.5	279
	Max	0	115 543	1329	86.9	0
Custom	Roulette-Wheel	0	85 096	1086	73.2	244
	Max	0	115 543	1329	86.9	0

capacity balancing, and new DCB problems, with a larger geographical scope, may need to be solved.

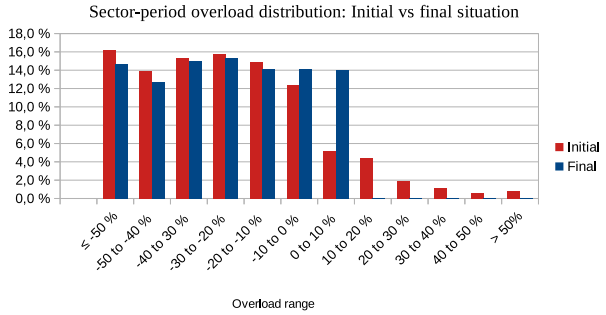
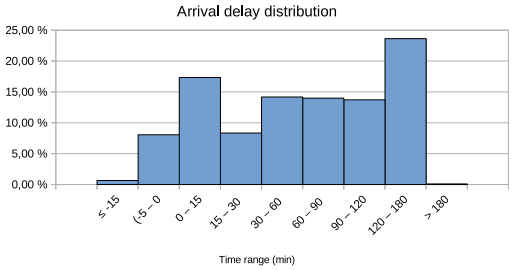


Figure 3. Sector-period overload distribution: comparison between the initial and the final situations.

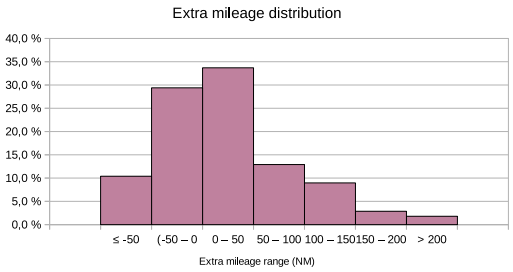
V. CONCLUSION

The problem of demand-capacity hotspot mitigation using ground delay and rerouting was considered in the case of very critical convective weather, leading to a sharp decrease in en-route sector capacity. A simulated-annealing-based hyper-heuristic using efficient low-level heuristics relying on the concept of flight overload factor was proposed. A numerical study was conducted to identify the best combination of low-level heuristics. Applying a roulette-wheel on a subset of flights with a high overload factor reveals to be the best flight selection rule along the algorithm iterations, either to apply delay or rerouting. Moreover, following the delay heuristic by a local search procedure decreased significantly the total arrival delay. Future work will focus on integrating uncertainty on capacity data, as predictions 24 hours before operations cannot have high accuracy. Also, a two-stage procedure can be put in place in order to update the mitigation plan, once a more accurate capacity drop prediction is provided on the day of operations.

Although a DCB solver based on the optimization of individual flights, called *cherry-picking* measures, can reduce



(a) Arrival delay distribution.



(b) Extra-mileage distribution among rerouted flights.

Figure 4. Best solution characteristics.

the total delay compared to CASA, its applicability is still very questionable. On the one hand, FMPs think usually in terms of regulations applied to sector-periods or traffic volumes. Also, airspace users may not be keen on delaying their flights, just for the sake of "global optimality", while they do not cross hotspots. Consequently, such a paradigm change may need time, and hybrid DCB mitigation technologies as well as, more collaborative operations, may be required.

ACKNOWLEDGMENT

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] C. Walker, "All-causes delay and cancellations to air transport in Europe - annual report for 2020," Eurocontrol, Tech. Rep., 03 2021. [Online]. Available: www.eurocontrol.int/publication/all-causes-delay-and-cancellations-air-transport-europe-2020

[2] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches: revisited," in *Handbook of metaheuristics*. Springer, 2019, pp. 453–477.

[3] E. K. Burke, M. Gendreau, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[4] P. Juntama, D. Delahaye, S. Chaimatanan, and S. Alam, "Hyperheuristic approach based on reinforcement learning for air traffic complexity mitigation," *Journal of Aerospace Information Systems*, vol. 19, no. 9, pp. 633–648, 2022.

[5] R. Dalmau, G. Gawinowski, and C. Anoraud, "Comparison of various temporal air traffic flow management models in critical scenarios," *Journal of Air Transport Management*, vol. 105, p. 102284, 2022.

[6] 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.

[7] M. Dalichampt and C. Plusquellec, "Hourly entry counts versus occupancy count– relationship, definitions and indicators," 2007.

[8] D. Delahaye, S. Puechmorel, J. Hansman, and J. Histon, "Air traffic complexity map based on non linear dynamical systems," *Air traffic control quarterly*, vol. 12, no. 4, pp. 367–388, 2004.

[9] J. Lavandier, A. Islami, D. Delahaye, S. Chaimatanan, and A. Abecassis, "Selective simulated annealing for large scale airspace congestion mitigation," *Aerospace*, vol. 8, no. 10, 2021.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[11] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[12] A. Franzin and T. Stützle, "Revisiting simulated annealing: A component-based analysis," *Computers & Operations Research*, vol. 104, pp. 191–206, 2019.