

ASPaeroFlow: Decomposition Heuristics for Joint Air Traffic Flow & Capacity Management

Alexander Beiser ✉ 

TU Wien, Vienna, Austria

Markus Hecher ✉ 

University of Potsdam, Potsdam, Germany

CNRS, Artois University (CRIL), Artois, France

Nysret Musliu ✉ 

TU Wien, Vienna, Austria

Georg Trausmuth ✉

Frequentis AG, Vienna, Austria

Stefan Woltran ✉ 

TU Wien, Vienna, Austria

Abstract

While mathematical models act as vital decision support systems for operational Air Traffic Flow and Capacity Management (ATFCM), existing approaches isolate Air Traffic Flow Management (ATFM) from Dynamic Airspace Configuration (DAC). This separation introduces an unresolved circular dependency between fixed-demand and fixed-capacity assumptions. Although joint optimization resolves this gap, the enlarged search space renders exact models computationally intractable for medium- to large-scale instances. To bridge this gap, we propose *ASPaeroFlow*: a heuristic for the joint ATFCM; it combines instance-space decomposition heuristics with a local exact approach using Answer Set Programming. We benchmark ASPaeroFlow from small to industry-sized instances and compare it with exact and alternative approaches. The results indicate that (1) the heuristic provides a computational middle ground between exact methods and operational baselines; (2) simultaneous optimization can outperform sequential optimization on joint ATFCM; and (3) an ablation study indicates that DAC has a larger impact on solution quality than flow measures.

2012 ACM Subject Classification Computing methodologies → Artificial intelligence

Keywords and phrases Air Traffic Flow and Capacity Management, ATFCM, Answer Set Programming, Heuristics, Open Data

Digital Object Identifier 10.4230/OASICS.ATMOS.2026.10

Related Version *Extended Version*: <https://arxiv.org/abs/2608.09315>

Supplementary Material *Dataset*: <https://doi.org/10.5281/zenodo.21869481>

Funding This research was supported by Frequentis and the Austrian Science Fund (FWF), grant 10.55776/COE12.

Markus Hecher: Hecher has been supported by the French National Research Agency (ANR), grant ANR-25-CE23-7647-01, and the Austrian Science Fund (FWF), grant 10.55776/J4656.

Acknowledgements We thank Dr. Alba Agustín for sharing the literature benchmarks.

1 Introduction

Operational *Air Traffic Flow and Capacity Management* (ATFCM) aims to balance demand and capacity to ensure the safe and efficient flow of air traffic. Demand is the number of flights intending to traverse through a *sector*, while capacity is the maximum number of



© Alexander Beiser, Markus Hecher, Nysret Musliu, Georg Trausmuth, and Stefan Woltran; licensed under Creative Commons License CC-BY 4.0

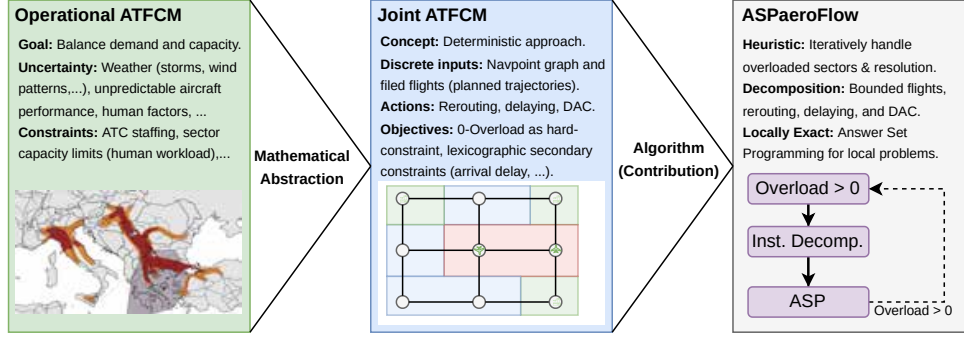
26th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2026).

Editors: Valentina Cacchiani and Stefan Funke; Article No. 10; pp. 10:1–10:20

OpenAccess Series in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Illustrative schematics of the problem setting including the main contribution.

flights a sector can safely handle. Sector capacity is limited by human factors, as *Air Traffic Controllers* (ATCs) are responsible for ensuring separation between aircraft. *Overloading* a sector means that more flights are in the sector than can be handled by ATCs. To ensure safety, demand must remain below capacity at all times [13].

Automated decision support systems help operational ATFCM by optimizing on mathematical models, which suggest aircraft reroutings, delayings, or sector restructurings to optimize airspace usage. However, the operationally used algorithm *Computer-Assisted Slot Allocation* (CASA), which works according to a “First Planned - First Served” scheme, has been shown to yield unsatisfactory results compared to more sophisticated optimization models [19, 10]. To circumvent this limit, state-of-the-art (SOTA) work proposes mathematical models for optimizing ATFCM. Still, SOTA approaches can be largely grouped into one of two classes: work focusing on the aircraft-level (so-called *Air Traffic Flow Management* (ATFM)) via delays and reroutes [8, 2, 21] and work focusing on the sector-level (so-called *Dynamic Airspace Configuration* (DAC)) [16, 33, 27, 11]. Treating ATFM and DAC separately introduces unresolved circular dependencies and makes it hard to compare the actual benefits of ATFM/DAC. In contrast, *Joint ATFCM* methods [5] would (1) resolve the circular dependency between fixed-capacity and fixed-demand assumptions, (2) render algorithms for isolated subproblems comparable against a joint benchmark, and (3) enable simultaneous optimization paradigms, which exploit the enlarged search space. However, while joint ATFCM models exist, their enlarged search space renders them computationally intractable for medium- and large-scale instances when solved exactly. Therefore, the primary research gap is the lack of *scalable* joint ATFCM methods for industry-sized instances.

Contributions. This paper partially addresses this gap by proposing *ASPaeroFlow*: a heuristic for *joint ATFCM* which optimizes ATFM and DAC simultaneously. See Figure 1 for the conceptual steps. In detail, our contributions are as follows:

- **ASPaeroFlow.** We propose heuristics that derive 0-overload solutions from initial flight schedules and sector configurations. The approach employs an iterative instance-space decomposition to partition global airspace overloads into localized, timestep-specific subproblems. These tractable instances are solved exactly via the Logic Programming technique Answer Set Programming (ASP) [17], leveraging its expressive rule-based modeling and native integration of soft constraints for domain-specific optimization.
- **Data Generator.** We generate realistic industry-sized (large) instances on real-world navpoint graphs for scaling analysis.

- **Benchmarks & Ablation.** We evaluate ASPaeroFlow in small [5] to large instances, and other instances from the literature [2]. ASPaeroFlow bridges the scalability challenge of exact models while significantly outperforming operational baselines. Further, the experiments suggest that simultaneous optimization outperforms sequential approaches when DAC alone does not suffice, and an ablation study highlights DAC as the primary driver for overload reduction.

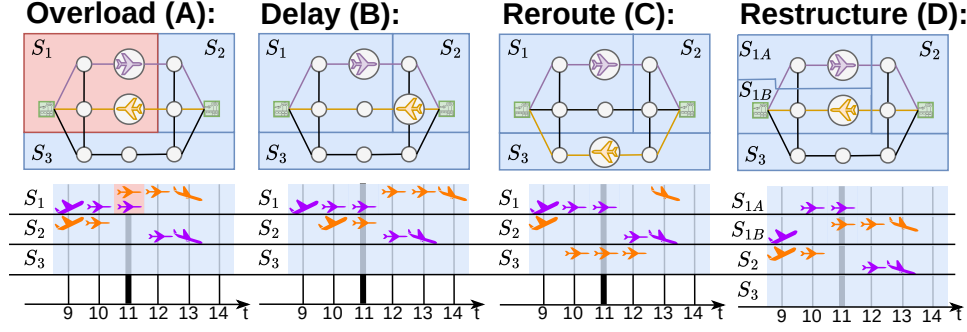
Related Work. **Dynamic Airspace Configuration (DAC)** constructs airspace from atomic blocks, distinguishing it from free-partitioning DAS [26]. Automated sector design utilizes genetic algorithms, Voronoi diagrams [30, 33, 32, 11], graph methods [20], machine learning [31], and MIP [27]. Heuristics for sector design exist: sectors should be connected; flights should stay in a sector for a minimum amount of time and not re-enter [30]. Joint ATFCM constructs composite sectors (containing multiple atomic sectors) from atomic sectors (which represent exactly one navpoint vertex). ASPaeroFlow ensures connected sectors. **Flow Optimization (ATFM)** aims at routing network flow (aircraft) optimally, while adhering to side constraints. The operationally used *Computer-Assisted Slot Allocation* (CASA) heuristics work according to the “First Planned - First Served” principle and delay aircraft [19, 10]. Optimization models addressing delay and rerouting explore fairness [6], lexicographic objectives [15], and computational aspects [7, 9, 14, 3]. It was shown that the restriction to delaying is already *NP-hard* [8]. Crucially, most existing methods integrate DAC solely as a fixed input to ATFM [2, 21]; the recently proposed joint ATFCM model [5] circumvents this, however, *joint sequential or simultaneous optimization for industry-sized instances remains an open challenge*, as exact methods struggle already on small instances. **Answer Set Programming (ASP)** is a symbolic-AI paradigm rooted in logic programming offering exact optimization with high interpretability, deployed for example in industrial scheduling [4, 12, 1]. Regarding aviation, proposals for the usage of ASP for efficient drone scheduling [28] or optimization for mission schedules on vertiports [25] exist. We use ASP for several reasons, including its “natural” problem modeling, rapid prototyping, and future Explainable AI (XAI) integration potential. To circumvent ASP’s primary scalability limitation – the *grounding bottleneck* [29] – we embed it within a problem-decomposition heuristic [18].

Next, we present the necessary preliminaries (Section 2), followed by the introduction of the ATFCM model in Section 3 and ASPaeroFlow (Section 4). We present the experiments and their results in Section 5 and then conclude (Section 6).

2 Preliminaries

Graph Theory. We consider weighted undirected graphs $G = (V, E)$ using standard Euclidean or geodesic distance metrics. A trajectory tr is a simple path augmented with strictly increasing timestamps, yielding a sequence of (v_i, t_i) pairs.

Answer Set Programming (ASP). ASP [24] is a declarative, logic-based problem-solving paradigm. We omit formal semantics (e.g., stable models) for brevity, referring to standard literature [17], and focus on the syntax necessary for our encodings. An ASP program consists of rules $H \leftarrow B$, where the head H is derived if the body B is satisfied. Variables are capitalized. Generating an answer set (solution) to an ASP program is usually done by first *grounding* (instantiating the variables) and then solving. We can use systems such as Clingo [22] to generate an answer set. We utilize standard extensions [23] such as constraints



■ **Figure 2** Schematics of the ATFCM problem with available actions. At time 11 an overload in sector S_1 occurs (A). The model allows for three ways to resolve this issue: Apply a delay to a flight (B), reroute a flight (C), or restructure the airspace (D).

(empty head, which invalidate a solution if the body is true), choice rules (to define search spaces), aggregates, and soft constraints (for lexicographic optimization). The following snippet illustrates these constructs.¹

```

1 % Facts (no variables and no body) describing the input data:
2 config(0,10). config(1,5). fpl(0,5,1). fplt(0,7,2).
3 % Rule expressing flights are rerouted unless they choose traj. 0.
4 reroute(ID) ← flightID(ID), not chtrj(ID,0).
5 % Choice rule: Choose exactly one trajectory P for each flight ID.
6 1{chtrj(ID,P):trj(ID,P)}1 ← flightID(ID).
7 % Rule with Aggregate: Count demand and flag overloads.
8 overload(SEC,T,LOAD-CAP) ← sec_cap(SEC,CAP,T),#count{ID:sec_f(ID,SEC,T)}=LOAD,LOAD>CAP.
9 % Constraint: Reject solutions where a flight does not occur.
10 ← flightID(ID), not flight_occurs(ID).
11 % Soft Constraint: Minimize overload at priority level 10.
12 ← overload(SEC,T,OVER). [OVER@10,SEC,T]

```

3 Joint ATFCM Model

We present the Joint ATFCM model [5], where Figure 2 shows our running example.

► **Definition 1** (Navpoint Graph). *The navpoint graph $G = (V, E)$ is an undirected graph, where each vertex is labeled with spatial geodesic coordinates. We distinguish between airports and en-route vertices, where with $\text{airport}(G)$ and $\text{en-route}(G)$ we denote the respective sets. Let $d : V^2 \rightarrow \mathbb{R}$ be the distance function.*

► **Example 2.** Figure 2 shows a navpoint graph G (gray). Let $G = (V, E)$ with $V = \{a_0, a_1, v_0, \dots, v_8\}$, where $\{a_0, a_1\} \subseteq \text{airport}(G)$ and $\{v_0, \dots, v_8\} \subseteq \text{en-route}(G)$. Further, $E = \{(a_0, v_0), (a_0, v_3), (a_0, v_6)\} \cup \{(v_0, v_1), (v_1, v_2), (v_3, v_4), (v_4, v_5), (v_6, v_7), (v_7, v_8)\} \cup \{(v_2, a_1), (v_5, a_1), (v_8, a_1)\}$. We assume $\forall v_i, v_j \in V : d(v_i, v_j) = 1$

► **Definition 3** (Time model). *T_{gran} discretizes time into T_{gran} -timesteps per hour. We assume one-day instances; then the set of timesteps T is $T := \{t \mid t \in \mathbb{N}, 0 \leq t \leq T_{\text{gran}} \cdot 24\}$.*

► **Example 4.** For $T_{\text{gran}}=4$ we make 15-minute timesteps, i.e., $4 \cdot 24 + 1 = 97 = |T|$ timesteps.

¹ Note that the predicates used in this snippet (e.g., `reroute/1`, `overload/3`) are illustrative; the complete ASP encoding for our model is defined later in Section 4.4. and the appendix.

► **Definition 5** (Sector Configuration). A sector i is a set of vertices $\text{sec}(i, t) \subseteq V$, identified by a representative sector vertex $i \in V$, where $i \in \text{sec}(i, t)$ ensures the sector index directly corresponds to one of its contained vertices. ATC-teams have a fixed capacity and have a fixed location, implying atomic capacities of vertices c_i for $i \in V$ ($t = 0$): $\text{cap}(i, 0) = c_i$. A composite sector has capacity $\text{cap}(i, t) = \max_{j \in \text{sec}(i, t)} \text{cap}(j, 0)$, simulating that the best ATC-team of a region takes command. To get the sector i for a vertex $v \in V$ for $t \in T$, we define $i = \text{sec}^{-1}(v, t)$.

► **Example 6.** Ctd. example. Static model: $\forall t \in T$. $\text{sec}(a_0, t) = \{a_0\}$, $\text{sec}(a_1, t) = \{a_1\}$, $\text{sec}(v_0, t) = \{v_0, v_1, v_3, v_4\}$ (S_1), $\text{sec}(v_2, t) = \{v_2, v_5\}$ (S_2), and $\text{sec}(v_6, t) = \{v_6, v_7, v_8\}$ (S_3). Dynamic model, for: $9 \leq t \leq 13$: $\text{sec}(v_0, t) = \{v_0, v_1\}$ (S_{1A}) and $\text{sec}(v_3, t) = \{v_3, v_4\}$ (S_{1B}). Atomic capacity: $\forall v \in V : \text{cap}(v, 0) = 1$. Static model: $\forall t > 0$: $\text{cap}(a_0, t) = \text{cap}(a_1, t) = \text{cap}(v_0, t) = \text{cap}(v_2, t) = \text{cap}(v_6, t) = 1$, with a combined capacity of 5. Dynamic model (**SEC**): for $9 \leq t \leq 13$ the combined capacity is 6 (increase of capacity through splitting).

► **Definition 7** (Flight Model). Flights $f \in F$ are modeled as trajectories on the navpoint graph. Let $f \in F$ be a flight and f_{id} be its flight id: $f = (id_f, ((v_0, t_0), (v_1, t_1), \dots, (v_n, t_n)))$.

► **Definition 8** (Aircraft Model). We map flights to aircraft (as a single physical aircraft can have multiple flights per day, meaning a delay in an early flight can propagate downstream): an aircraft $a \in A$ is a three tuple (id_a, v_a, F_a) , where id_a is the unique id, v_a is the velocity, and F_a is its set of flights.

► **Example 9.** Ctd. example. Let $A = \{p_0, p_1\}$ with $p_0 = (id_{p_0}, v_{p_0}, \{f_0\})$ and $p_1 = (id_{p_1}, v_{p_1}, \{f_1\})$. Let $f_0, f_1 \in F$ with $f_0 = (0, ((a_0, 9), (v_0, 10), (v_1, 11), (v_2, 12), (a_1, 13)))$ and $f_1 = (1, ((a_1, 9), (v_5, 10), (v_4, 11), (v_3, 12), (a_0, 13)))$.

► **Definition 10** (ATFCM Instance). An ATFCM instance is $I = (G, T_{gran}, T, \text{sec}, \text{cap}, A)$, where G is a navpoint graph, T_{gran} defines the time granularity, T the timesteps, sec the initial sector configuration, cap the atomic capacities, and A the aircraft. We denote with F the set of filed flights. All ids are unique.

► **Definition 11** (ATFCM Solution). The ATFCM solution is given by: $\mathbf{S} = (\mathbf{T}, \mathbf{SEC}, \mathbf{A})$, where $\mathbf{T}$ are the solution timesteps, $\mathbf{SEC}$ is the solution DAC, and $\mathbf{A}$ are the solution aircraft. $\mathbf{T}$ might be expanded beyond the initial T (an instance is always solvable). The flown flights $f \in \mathbf{F} = \bigcup_{(id_a, v_a, F_a) \in \mathbf{A}} F_a$ may be delayed. If a flight is delayed and/or rerouted, this is indicated by $r_f = 1$. Sectors in $\mathbf{SEC}$ are allowed to be changed. We require that airport sectors remain atomic and en-route sectors remain connected to partially accommodate operational requirements. **Demand.** Let $f \in \mathbf{F}$ be a flight reaching navpoint $v_a \in V$ at time t_a and then flying to navpoint $v_b \in V$, which it reaches at time t_b . Assuming that $\Delta t = t_b - t_a$, then f is in the timespan $t \in [t_a, t_a + \frac{\Delta t}{2}]$ in sector $s_i = \text{sec}^{-1}(v_a, t)$ ($\text{over}(f, s_i, t) = 1$), and for $t' \in [t_a + \frac{\Delta t}{2} + 1, t_b]$ in sector $s_{i+1} = \text{sec}^{-1}(v_b, t')$ ($\text{over}(f, s_{i+1}, t') = 1$). For a time $t \in T$, and a sector $i \in V$, we measure demand $q(i, t)$, as the number of aircraft currently in the sector, i.e., $q(i, t) = \sum_{f \in \mathbf{F}} \text{over}(f, i, t)$. For each timestep $t \in \mathbf{T}$, we define with $\mathbf{SEC}$ the DAC. We require for a solution $\mathbf{S}$ to be accepted that for all $t \in \mathbf{T}$, $i \in V$, demand $q(i, t) \leq \text{cap}(i, t)$.

► **Example 12.** Overload (A): Let $\mathbf{S} = (T, \text{sec}, A)$, then $q(v_0, 11) = 2 > \text{cap}(v_0, 11) = 1$. Delay (B): Let $\mathbf{S} = (T, \text{sec}, \mathbf{A})$ with $\mathbf{A}$, where $\mathbf{F} = \{f_0, f'_1\}$ s.t. $f'_1 = (1, ((a_1, 10), (v_5, 11), (v_4, 12), (v_3, 13), (a_0, 14)))$. Reroute (C): Let $\mathbf{S} = (T, \text{sec}, \mathbf{A})$ with $\mathbf{A}$, where $\mathbf{F} = \{f_0, f'_1\}$ s.t. $f'_1 = (1, ((a_1, 9), (v_8, 10), (v_7, 11), (v_6, 12), (a_0, 13)))$. Reconfig (D): Let $\mathbf{S} = (T, \mathbf{SEC}, A)$ with $\mathbf{SEC}$ (Ex. 6).

■ **Table 1** Representative parameter bounds used in $ASPaeroFlow_{r,d}$'s local ASP subproblems. Exact values are variant dependent (cf. Section 5).

Chosen bounds in instance-space decomposition		
Parameter	Operational rationale	$ASPaeroFlow_{r,d}$
$f_{\max}$	Limit the local subproblem to flights that actually contribute to the current overload; keeps grounding and solving stable while capturing the dominant contributors.	2 flights
$d_{\max}$	Strategic regulations usually consider bounded departure shifts; a short delay window provides a realistic control measure without exploding combinations.	5 steps
$r_{\max}$	A small set of diverse candidate routes (e.g., k -shortest with similarity filtering) reflects that only few reroute options are operationally acceptable and keeps the action set manageable.	3 routes
$p_{\max}$	Use a small set of sector split options for the overloaded sector; captures the main capacity measure (partitioning) while avoiding enumeration of arbitrary partitions.	2 splits

► **Definition 13** (Optimization Problem). *A solution $\mathbf{S}$ is optimized according to efficiency (Arrival Delay, Active en-route sectors, and Navpoint-sector changes) and stability/fairness (Reroutes/delays and Reconfigurations) soft constraints. An accepted solution $\mathbf{S}$ is strictly preferred to $\mathbf{S}'$ ($\mathbf{S} \prec \mathbf{S}'$) if its objective tuple is lexicographically smaller: $(K_d, K_{\#s}, K_{ds}, K_r, K_c)\mathbf{S} \prec_{lex} (\dots)\mathbf{S}'$. Let t_f^a and t_f^e be the actual and expected arrival times. The individual criteria to be minimized are defined as follows, utilizing indicator function $\mathbb{I}(\cdot)$:*

$$K_d = \sum_{f \in \mathbf{F}} d_f = \sum_{f \in \mathbf{F}} t_f^a - t_f^e \quad (\text{Arrival delay})$$

$$K_{\#s} = \sum_{t \in \mathbf{T}_{>0}} \sum_{i \in V} \mathbb{I}(|\mathbf{SEC}(i, t)| > 0) \quad (\text{Active Sectors})$$

$$K_{ds} = \sum_{t \in \mathbf{T}_{>1}} \sum_{v \in V} \mathbb{I}(\mathbf{SEC}^{-1}(v, t-1) \neq \mathbf{SEC}^{-1}(v, t)) \quad (\text{Navpoint-sector changes})$$

$$K_r = \sum_{f \in \mathbf{F}} r_f \quad (\text{Reroutes/delays})$$

$$K_c = \sum_{t \in \mathbf{T}_{\geq 0}} \sum_{v \in V} \mathbb{I}(\mathbf{sec}^{-1}(v, t) \neq \mathbf{SEC}^{-1}(v, t)) \quad (\text{Reconfigurations})$$

where $\mathbf{SEC}^{-1}(v, t)$ returns the representative sector index i containing vertex v at time t .

4 New Decomposition Heuristics

Although some large-scale ATFM models can be solved exactly by exploiting inherent structures [3], we expect that integrating DAC introduces dense coupling constraints. These constraints likely create structural difficulties that prevent scaling exact solution approaches to medium- and large-scale instances. Moreover, the required sequential bounding for the five-level lexicographic objective severely degrades solver performance. Consequently, exact methods suffer from combinatorial explosion, rendering multi-objective exact solving computationally intractable even for relatively small instances [5]. To ensure operational feasibility, we propose a heuristic approach: $ASPaeroFlow^2$. $ASPaeroFlow$ decomposes the

² GitHub Link: <https://github.com/alex14123/ASPaeroFlow-Optimizer>

■ **Algorithm 4.1** ASPaeroFlow.

Data: Instance $\mathcal{I}$, Encoding E , Parameters $\mathcal{P}$

```

1  $\mathbf{M} \leftarrow \text{convertInstance}(\mathcal{I})$  ;       $\text{overloads} \leftarrow \text{computeOverloads}(\mathbf{M})$  ;
2  $\mathcal{P}' \leftarrow \text{copy}(\mathcal{P})$  ;
3 while  $\text{overloads} > 0$  do
4    $I \leftarrow \text{decomposeInstance}(\mathbf{M}, \mathcal{P}')$  ;
5    $\mathbf{M}' \leftarrow \text{LocalOptimization}(I, E)$  ;       $\text{overloads}' \leftarrow \text{computeOverloads}(\mathbf{M}')$  ;
6   if  $\text{overloads}' < \text{overloads}$  then
7      $\mathbf{M} \leftarrow \mathbf{M}'$  ;       $\text{overloads} \leftarrow \text{overloads}'$  ;
8      $\mathcal{P}' \leftarrow \text{copy}(\mathcal{P})$  ;
9   else if  $\text{exhausted}(\mathcal{P}')$  then
10    return  $(\mathbf{M}, \text{False})$  ;      // Termination with residual overload
11  else
12     $\mathcal{P}' \leftarrow \text{adjustParameters}(\mathcal{P}')$  ;
13 return  $(\mathbf{M}, \text{True})$  ;      // Successful termination

```

problem around the next overloaded sector, building a bounded local subproblem containing only the most relevant flights and the overloaded sector. Specifically, we cap (i) the number of considered flights ($f_{\max}$), (ii) the delay window per flight ($d_{\max}$), (iii) the number of alternative routes per flight ($r_{\max}$), and (iv) the number of partition options for the overloaded sector ($p_{\max}$); see also Table 1. An exact solver then selects the best local solution under our lexicographic objective, and the procedure iterates until all overloads are resolved. The chosen local bounds balance search-space exploration with solver runtimes. Empirically, increasing the flight limit ($f_{\max} > 2$) or the partitioning limit ($p_{\max} > 2$) drastically increases solving times. Conversely, the routing ($r_{\max}$) and delay ($d_{\max}$) bounds are computationally more forgiving; the chosen parameters were determined via empirical parameter tuning to ensure rapid local convergence while maintaining high solution quality.

4.1 Optimization Loop

Algorithm 4.1 shows the overall algorithm. In Lines (1)–(2) we do pre-processing steps, by converting the instance to the internal matrix representation $\mathbf{M}$ and computing the overloads. We consider as overloads the sum of exceedances: $K_o = \sum_{t \in T, i \in V} \max\{0, q(i, t) - \text{cap}(i, t)\}$. The algorithm iteratively solves overloads until $K_o = 0$ (Line (3)). The decomposed instance I is computed by instance space decomposition (Line (4) – Algorithm 4.2), which is then given with the encoding E to the local optimization procedure (Line (5)). The local optimization procedure computes a new candidate solution $\mathbf{M}'$. A *correct local optimization encoding* selects among its available choices the solution that reduces the magnitude of overload the most, followed by adhering to the criteria in Definition 13. The candidate solution $\mathbf{M}'$ is accepted if it has fewer overloads (Lines (6)–(8)). Otherwise, the parameters of the decomposition are adapted to allow for other solutions to be explored to ensure termination (Line (12)).

4.2 Instance Space Decomposition

Algorithm 4.2 shows our instance decomposition approach. We reduce the search space, by limiting the number of investigated flights $f_{\max}$, the delay $d_{\max}$, the alternative routes $r_{\max}$, and the possible DAC partitions $p_{\max}$ (Line (1)). Line (2) obtains the time t and sector s of the first overloaded sector (lexicographically time and sector-wise). In Line (3) we limit the search space to $f_{\max}$ flights. The next part in Lines (4)–(12) is to generate the instance for

■ **Algorithm 4.2** Instance Space Decomposition.

Data: Converted Instance $\mathbf{M}$, Parameters $\mathcal{P}$

```

1  $f_{\max}, d_{\max}, r_{\max}, p_{\max} \leftarrow \mathcal{P}$  ;
2  $s, t \leftarrow \text{computeFirstOverload}(\mathbf{M})$  ;
3  $F \leftarrow \text{flightsInOverload}(\mathbf{M}, s, t)$        $F \leftarrow \text{sort}(F)$        $F \leftarrow F[: f_{\max}]$  ;
4  $I_F \leftarrow \emptyset$  ;
5 for  $f$  in  $F$  do
6    $\text{config} \leftarrow 0$  ;
7    $R \leftarrow \text{bestRoutes}(f, G, r_{\max})$  ;
8   for  $d$  in  $\text{range}(d_{\max} + 1)$  do
9     for  $r$  in  $R$  do
10       $f' \leftarrow \text{computeNavpointFlightPlan}(f, d, r)$  ;
11       $I_F \leftarrow I_F \cup \{f'\}$  ;
12       $\text{config} \leftarrow \text{config} + 1$  ;
13  $I_S \leftarrow \emptyset$  ;
14 for  $p$  in  $\text{range}(1, p_{\max} + 1)$  do
15    $\text{partition}_p \leftarrow \text{computePartition}(s, p, G)$  ;
16    $I_S \leftarrow I_S \cup \{\text{partition}_p\}$  ;
17  $I \leftarrow I_F \cup I_S$  ;
18 return  $I$  ;

```

each of the $f_{\max}$ flights, by computing the alternatives $r_{\max}$ and considering the maximum delay $d_{\max}$. This is followed by the generation of the instance for DAC, where we consider $p_{\max}$ partitions (Lines (13)–(16)), ranging from the current configuration $p = 1$ to a partition of the sector s into $p_{\max}$ parts. Finally, we combine the instance in Line (17) and return it.

Reroute Generation. We compute the $r_{\max}$ routes (Algorithm 4.2 in Line 7), using an iterative algorithm that computes the shortest routes on G between departure and destination airports. On each iteration the current shortest path p is penalized and a diverse set of routes is ensured by only accepting a path p if its Jaccard similarity to any other already considered path p' is small enough: $\frac{p' \cap p}{p' \cup p} < j_m$, where $j_m = 0.6$. Let $w(p)$ be the path weight, then each edge e of $e \in p$ is penalized by $w_{t+1}(e) \leftarrow w_t(e) + 0.1 \frac{w(p)}{|p|}$, where $w_0(e) = d(e)$.

Alternative Airspace Configurations. The $p_{\max}$ graph partitions are computed in Algorithm 4.2 in Line (15) by an iterative breadth-first search algorithm. We partition a sector s into p sectors ($\{s_1, \dots, s_p\} = S$) of approximate size $\frac{|sec(s, t)|}{p}$ iteratively. Assuming we generated $|S| = i - 1$ sectors so far, we add a new sector s_i via two steps: first, we select a $seed \in sec(s, t) \setminus (\cup_{s_j \in S} sec(s_j, t))$ vertex and add it to the new sector $s_i = \{seed\}$. Second, we add vertices $v \in sec(s, t) \setminus (\cup_{s_j \in S} sec(s_j, t))$ to s_i in a breadth-first manner (ensuring connection of s_i) until $|sec(s_i, t)| > \frac{|sec(s, t)|}{p}$; then we add s_i to S .

Adjust Parameters. If search stalls, `adjustParameters` expands the temporal horizon using a rolling delay (+5 timesteps). After 10 non-improvement steps, the flight limit reduces to $f_{\max} = 1$. Assuming any flight can be routed in isolation, the algorithm *terminates* and returns $\mathbf{M}$ (Line 10) if $f_{\max} = 1$, the subsequent step yields no improvement, and the single flight f remains unroutable due to absolute capacity limits (e.g., zero-capacity sectors).

4.3 Correctness

We provide a brief argument for correctness of the algorithm. Let $\mathcal{I}$ be an instance of the optimization problem and let $\mathcal{S}$ be the set of solutions. For the following proofs, we take the operational assumption that all filed flights can be flown in principle; meaning that

$\forall v \in V : \text{cap}(v, 0) \geq 1$. We refer to this as *operational feasibility*. Operational feasibility guarantees that a single flight from an overloaded sector can always be delayed to reduce the magnitude of overload – consequently, if delaying the single flight does not lead to a reduction in overload magnitude, the algorithm can safely terminate.

► **Theorem 14.** *Let $\mathcal{I}$ be an instance of the optimization problem and let the operational feasibility assumption hold. Then, for any correct encoding E , if ASPaeroFlow (Algorithm 4.1) produces an output, its output is a valid solution.*

Proof (Sketch). The Algorithm 4.1 returns the solution $\mathbf{M}$ in Line 13, or in Line 10. We prove that with the operational feasibility assumption, if Algorithm 4.1 terminates, it terminates by returning the solution $\mathbf{M}$ in Line 13: Towards a contradiction assume Algorithm 4.1 terminates prematurely on exhaustion (Line 10). For this, it must be the case that for at least 10 subsequent steps the algorithm did not experience a reduction of overload, although it increased the time window start to t_s and reduced the number of considered aircraft to $f_{\max} = 1$. Let the considered flight for optimization be $f = (id, tr)$ for sector $i \in V$ and time $t' \in \mathbf{T}$. Further, let f' be the flight in the current solution $\mathbf{M}$ with the maximum actual arrival delay $t_{f'}^a$. Then, to exit prematurely on exhaustion in Line 10, flight f cannot be scheduled after flight f' , so it must hold that whenever $t_s > t_{f'}^a$, no improvement is found. However, such an improvement is found by local optimization due to the operational assumption: adjust tr to tr_s , by keeping the path, but starting at t_s , the overload of i decreases and no increase in overload of any other sector is experienced, as by the operational feasibility assumption $\forall (v, t) \in tr_s : \text{cap}(v, t) \geq q(v, t) = 1$. This is a contradiction to the assumption.

It remains to argue that the returned $\mathbf{M}$ from Line 13 is indeed a valid solution (Def. 11). First, there are no overloads remaining in $\mathbf{M}$ due to Line 3. Further, **A** and **SEC** are valid. **A** is valid, as (a) the instance space decomposition generates valid trajectories and ensures that for each considered flight its subsequent flights are also part of the instance passed to local optimization and thereby cannot overlap (Lines (5)–(12) of Algorithm 4.2) and (b) local optimization ensures that exactly one trajectory is selected for each flight. Further, **SEC** is valid as bounded connected partitions introduced by Lines (13)–(16) of Algorithm 4.2, combined with local optimization ensure valid partitions. ◀

► **Observation 15.** *Let $\mathcal{I}$ be an instance of the optimization problem. For any correct encoding E without assuming operational feasibility, Algorithm 4.1 may return an incorrect solution.*

Proof (Sketch). When the operational feasibility assumption is violated, it is not guaranteed that a flight f can actually occur; however, it can be the case that ASPaeroFlow terminates prematurely on exhaustion (Line 10) in Algorithm 4.1, thereby producing an invalid solution $\mathbf{S}$, although in principle a solution $\mathbf{S}'$ exists by Def. 11: this can happen by (1) ASPaeroFlow not considering routes, which are outside of the bounded rerouting $r_{\max}$ options, or (2) ASPaeroFlow not considering sector configurations, which are not evaluated due to bounded sectorization $p_{\max}$. ◀

► **Theorem 16.** *Let $\mathcal{I}$ be an instance of the optimization problem. Then, for any correct encoding E , ASPaeroFlow (Algorithm 4.1) terminates.*

Proof (Sketch). In each iteration of Line 3 of Algorithm 4.1 either the overload strictly decreases (a new solution is accepted as the current one and the parameter bounds are reset to their initial values), or the parameter bounds are expanded via *adjustParameters*. If the total

overload reaches 0, the algorithm terminates successfully (Line 13). Otherwise, the algorithm ensures $f_{\max} = 1$ by setting it after 10 non-improvement steps via *adjustParameters*. The *exhausted* condition then becomes true if the earliest starting time t_f of the single ($f_{\max} = 1$) considered flight f is strictly later than the arrival time t_f^a , of any other flight f' : $t_f > t_f^a$, (and no improvement occurs). When exhausted, the algorithm terminates with a residual overload (Line 10). ◀

4.4 Local Optimization via ASP

We proceed to describe an encoding in ASP. Conceptually, ASP is similar to Boolean satisfiability, but allows for more natural problem modeling, rapid integration of changed problem definitions, and directly provides minimization statements.

For brevity, the detailed generation of facts, trajectory guesses, and structural mappings are deferred to the appendix. Here, we focus on the optimization objectives. We treat the minimization of the magnitude of overload as the primary objective – as the ASP model operates locally and cannot solve global overloads. Besides that, we minimize lexicographically according to the ATFCM^{OPT} problem. Lines (1) and (2) describe minimization of overloads, while Line (3) describes minimization of arrival delay. By construction, Line (4) minimizes both the number of sectors and navpoint-sector changes. Finally, Line (5) minimizes reroutes/delays and Line (6) minimizes divergences from *sec*.

```

1  $\leftarrow$  overload(X,T,OVER). [OVER@10,X,T]
2  $\leftarrow$  chc(CONFIG), config(CONFIG,TOTAL_OVER). [TOTAL_OVER@10,CONFIG]
3  $\leftarrow$  arrival_delay(ID,DIFF). [DIFF@9,ID]
4  $\leftarrow$  chc(CONFIG), config_number_sectors(CONFIG,NUMBER). [NUMBER@8,CONFIG]
5  $\leftarrow$  reroute(ID). [1@7,ID]
6  $\leftarrow$  chc(CONFIG). [CONFIG@6,CONFIG]
```

► **Lemma 17.** *The ASP encoding is a correct local optimization encoding.*

Proof (Sketch). We argue for two properties: (a) the found solution $\mathbf{M}'$ locally adheres to the specifications of an ATFCM solution (Def. 11) except the overload; and (b) the found solution $\mathbf{M}'$ is locally optimal s.t. it reduces overloads while adhering to side constraints (Def. 13). (a): To prevent invalid double assignments or double sector splits, exactly one path ($\text{chtrj}(\text{ID},\text{P})$) and one configuration ($\text{chc}(\text{CONFIG})$) is selected. (b): the primary optimization criterion is to reduce K_o , as finding a globally $K_o = 0$ solution by local optimization cannot be guaranteed; the objective combines local effects computed in ASP ($\text{overload}(\text{X},\text{T},\text{OVER})$) with effects computed by Algorithm 4.2 ($\text{config}(\text{CONFIG},\text{TOTAL_OVER})$). The other optimization criteria adhere to the specification of Def. 13: for $[\text{NUMBER@8},\text{CONFIG}]$ we use the fact that for any two configurations produced by instance space decomposition, if the number of resulting splits is lower ($K_{\#s}$), the number of sector changes is as well (K_{ds}). The number of reroutes K_r directly translates from Def. 13; the number of reconfigurations is minimized by the fact that $\text{chc}(\text{CONFIG})$ for $\text{chc}(0)$ has 0 introduced config changes. ◀

5 Benchmarks and Experiments

To demonstrate the operational feasibility of our approach, we design our computational experiments to evaluate three primary dimensions: (i) the solution quality of our decomposition heuristic relative to exact models, (ii) the scalability of the approach compared to related heuristic approaches across progressively larger, real-world-sized scenarios, and (iii) the effects of integrating DAC into ATFM via an ablation study. To quantify these dimensions, we extract the following evaluation metrics: computational performance (execution time and peak RAM usage) and model metrics (unresolved overloads, K_d , $K_{\#s}$, K_{ds} , K_r , K_c).

■ **Table 2** Compared variants and their action bounds. Global approach search space $\approx d_{\max}^{|F|} \cdot r_{\max}^{|F|} \cdot p_{\max}^{|V|}$, local search space (per iteration, assuming decomposition $f_{\max} = 2, |V| = 1$) $\approx d_{\max}^2 \cdot r_{\max}^2 \cdot p_{\max}$.

Variant	$p_{\max}$	$r_{\max}$	$d_{\max}$	Notes
Basic Baseline				
Initial	0	0	0	Result if no action is taken (naive baseline).
Global (Exact) Approaches [5]				
ASP-P	4	4	expanded	Global decisions; partial DAC; partial Rerouting, partial delaying; delays expanded up to feasibility; initially $d_{\max} \approx T_{gran} \cdot 24$
MIP	1	2	expanded	Global decisions; partial rerouting; small delay bound iteratively expanded up to feasibility; initially $d_{\max} \approx T_{gran} \cdot 24$
Local Heuristic Approaches				
ASPaeroFlow _{r,d}	2	3	5	Default configuration.
ASPaeroFlow _{$\neg r, d_p$}	2	1	1	Only sectorization; if stuck repeatedly delaying possible.
ASPaeroFlow _{$\neg r, d$}	2	1	5	Sectorization and delay (no rerouting).
ASPaeroFlow _{r, d_p}	2	3	1	Sectorization and rerouting; if stuck repeatedly delaying possible.
ASPaeroFlow _{seq}	2	3	5	Sequential execution of DAC and (full) ATFM.
ATFM _{r, d}	1	3	5	Flow-only ATFM (no DAC).
ATFM _{r, d_p}	1	3	1	Rerouting-only baseline (no delay); if stuck repeatedly delaying possible.
ATFM _{$\neg r, d$}	1	1	5	Delay-only ATFM (no rerouting).
CASA	1	1	5	Single flight delaying (decomposition $f_{\max} = 1$).

5.1 Benchmarked Variants

We benchmarked 12 variants. One (*Initial*) is our baseline, if no action is performed. The *exact* solvers comprise *ASP-P*, which is *ASP _{r_p, d_p, s_p}* – a bounded action exact model which balances time to first and time to best solution – and *MIP*, a mapping of state-of-the-art ATFM MIP formulations to the joint ATFCM model (meaning it selects optimal bounded reroutings and delays, but has no sectorization actions) [5]. *ASPaeroFlow* comprises the remaining nine variants under various parameter configurations. Variants executing joint optimization of ATFM and DAC are denoted as *ASPaeroFlow*, while restricted variants optimizing only ATFM are denoted as *ATFM*. Subscripts $r \in \{r, \neg r\}$ and $d \in \{d, d_p\}$ indicate whether rerouting or delaying is permitted. To simulate operational baselines, we include *CASA*, a greedy “First Planned - First Served” simulation restricting optimization to single-flight delays ($f_{\max} = 1$). Furthermore, the *ASPaeroFlow_{seq}* *sequential* variant models standard collaborative decision-making by separating the pipeline: optimizing sectorization first, followed by a *ATFM _{r, d}* flow optimization.

5.2 Experiment Scenarios and Setup

We conducted benchmarks on three sets of instances: *Small* [5], *SOTA* [2], and *Large*. Table 3 shows an overview of the considered instances.

The Small scenario. Regarding traffic density, we vary the number of flights $|F| \in \{10, 20, \dots, 100\}$. For each $|F|$, we evaluate three distinct random seeds. En-route vertices possess a fixed capacity of 1, while airport capacities are assumed to be unconstrained. Time is partitioned into 1-hour timesteps ($T_{gran} = 1$).

SOTA Instances. The SOTA instances are standard benchmarks in the literature [7, 2]. Instances *c01*, *c02*, and *c03* are randomized scenarios designed to test comprehensive ATFCM features. BLO instances are similarly randomized, where mBLO represents specific

10:12 Decomposition Heuristics for Joint Air Traffic Flow & Capacity Management

■ **Table 3** Instance Description: $|V|$ is the sum of navpoint and airport vertices, $|S|$ is the maximum sector size, C the maximum capacity (or range of maximum capacity), TG the time granularity, $|F|$ the number (range) of flights, and $|A|$ the number (range) of aircraft.

Small Instances [5]								
Instance	Navpoints	Airports	$ V $	$ S $	C	T_{gran}	$ F $	$ A $
EA-3x3	9	7	16	3	1	1	10 to 100	6 to 41
CE-5x5	25	7	32	5	1	1	10 to 100	6 to 48
IND-4x10	40	8	48	6	1	1	10 to 100	6 to 43
USA-7x7	49	8	57	7	1	1	10 to 100	5 to 50
EUR-10x10	100	17	117	10	1	1	10 to 100	7 to 53
SOTA Instances [2]								
BLO1	107	40	147	8	34	4	1054	568
BLO2	1172	20	1192	59	88	4	3196	3196
BLO3	2707	30	2737	135	130	4	6475	6475
IC	5932	47	5979	118	25	4	3157	3157
c01	107	40	147	8	34	4	1054	568
c02	180	40	220	7	30	4	1330	671
c03	147	45	192	8	43	4	1545	788
mBLO1	100	4	104	9	67	4	812	812
mBLO3	2707	30	2737	135	130	4	6475	6475
Large Instances								
$P \cdot 1.0 \rightarrow P \cdot 0.1$								
CE-G	49	7	56	11	3 to 1075	4	1000 to 31622	202 to 5978
USA-G	200	8	208	20	5 to 1615	4	1000 to 31622	272 to 8647
EUR-G	800	17	817	85	5 to 1719	4	1000 to 31622	279 to 7506
EA-G	1600	7	1607	34	4 to 2015	4	1000 to 31622	322 to 14007
CE	627	10	637	35	11 to 3716	4	1000 to 31622	594 to 19069
DACH	1283	324	1607	50	3 to 1065	4	1000 to 31622	587 to 18699
EUR	8479	1588	10067	50	2 to 719	4	1000 to 31622	886 to 28020
USA	18101	1509	19610	50	2 to 1042	4	1000 to 31622	976 to 30586

modifications (we do not possess mBLO2). The *IC* instance reflects an industry case, capturing a typical day of flight schedules and paths for a segment of the European ATM network. Following related literature, time is partitioned into 15-minute intervals ($T_{gran} = 4$).

Large instances. The large scenario considers larger graphs with more flights per day. The number of flights is $|F| \in \{1000, 1778, 3162, 5623, 10000, 17782, 31622\}$ flights, where for each $|F|$ we consider three seeds (11904657, 150699, 42), and capacity is relative to nominal capacity (optimal weather/staffing). We scale atomic sector capacities uniformly from 100% down to 10% in 10% decrements ($P \cdot 1.0 \rightarrow P \cdot 0.1$). We set $T_{gran} = 4$ (as for SOTA).

Technical Setup. We implemented the heuristics in Python (3.13.7), and used Clingo (5.8.0) for ASP and Gurobi (13.0.0 build v13.0.0rc1) for MIP computation. All experiments were conducted on the Copperhead CPU Cluster at TU Wien, with 2x Intel Xeon Silver 4314 CPUs, 512GB RAM, and Ubuntu 22.04 (Kernel 5.15.0-131-generic). We considered a TIMEOUT of 1800s and a MEMOUT of 35GB.

5.3 Results and Discussion

Heuristic vs. Exact Approaches. We show the number of solved instances ($\#S$) and tournament wins ($\#W$) in Table 4. While for the small experiments, the ASP-P variant achieves strong results ($\#W : 83$), the ASPaeroFlow $_{r,d_p}$ variant achieves competitive results ($\#W : 51$), while CASA wins only $\#W : 4$. However, this is mostly due to the fact that even for small instances, the ASP-P approach runs into computational solving bottlenecks – even on the small instances, ASP-P is not able to solve all instances in the time limit. Comparing these methods on instances, which ASP-P solves to 0 overload, ASPaeroFlow $_{r,d_p}$ achieves a lower arrival delay (an average value of 25.62 ± 2.25) compared to CASA (an average value of 64.83 ± 10.19). MIP achieves worse results compared to ASP for solved small instances

■ **Table 4** Tournament wins $\#W$ (lexicographic: $K_o \succ K_d \succ K_{\#s} \succ K_{ds} \succ K_r \succ K_c$) and number of solved $\#S$ instances ($K_o = 0$) by scenario.

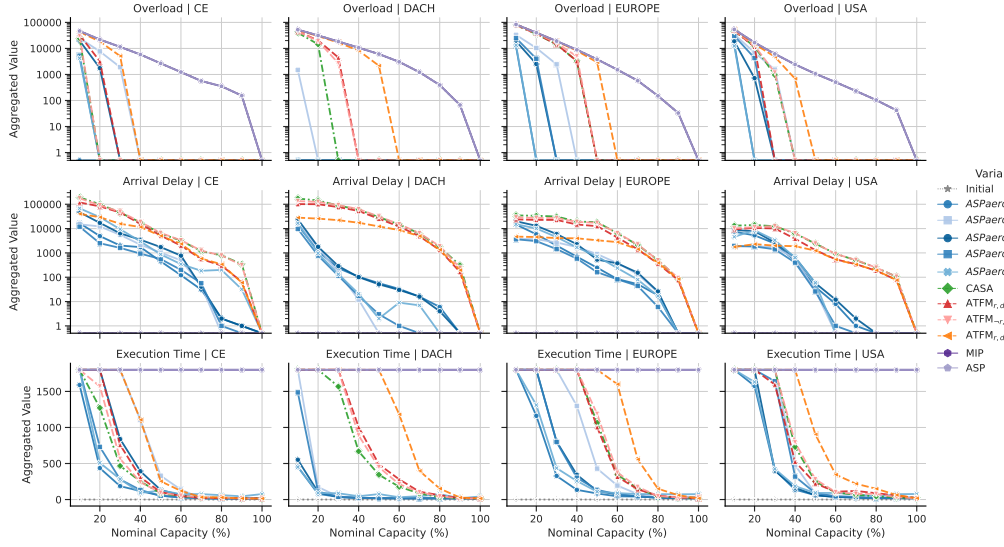
Scenario	#I	ASPaeroFlow _{r,d} #W	ASPaeroFlow _{r,d} #S	ASPaeroFlow _{seq} #W	ASPaeroFlow _{seq} #S	ASPaeroFlow _{r,d_p} #W	ASPaeroFlow _{r,d_p} #S	ASPaeroFlow _{~r,d} #W	ASPaeroFlow _{~r,d} #S	Draws #D
Large Instances (PCAP > 0.5)	840	46	834	343	840	127	831	1	826	250
Large Instances (PCAP ≤ 0.5)	840	211	674	213	685	290	605	6	631	82
Small Instances	200	15	200	13	200	51	200	0	200	8
SOTA Instances	9	0	9	1	9	1	7	0	9	2
Overall	1889	272	1717	570	1734	469	1643	7	1666	342
Scenario	#I	ASPaeroFlow _{~r,d_p} #W	ASPaeroFlow _{~r,d_p} #S	ATFM _{r,d} #W	ATFM _{r,d} #S	ATFM _{r,d_p} #W	ATFM _{r,d_p} #S	ATFM _{~r,d} #W	ATFM _{~r,d} #S	Draws #D
Large Instances (PCAP > 0.5)	840	52	815	2	824	4	754	0	820	250
Large Instances (PCAP ≤ 0.5)	840	37	525	0	407	0	205	0	411	82
Small Instances	200	6	200	0	200	1	200	0	200	8
SOTA Instances	9	0	9	3	9	2	7	0	8	2
Overall	1889	95	1547	5	1440	7	1166	0	1439	342
Scenario	#I	CASA #W	CASA #S	Initial #W	Initial #S	MIP #W	MIP #S	ASP-P #W	ASP-P #S	Draws #D
Large Instances (PCAP > 0.5)	840	0	824	0	168	15	180	0	112	250
Large Instances (PCAP ≤ 0.5)	840	0	448	0	0	1	1	0	3	82
Small Instances	200	4	200	0	1	19	82	83	92	8
SOTA Instances	9	0	9	0	0	0	0	0	0	2
Overall	1889	4	1481	0	169	35	263	83	207	342

■ **Table 5** Ablation study aggregated over large scenarios, showing absolute mean values and SEMs. We study the effects of enabling/disabling rerouting (r), delaying (d), and restructuring (s).

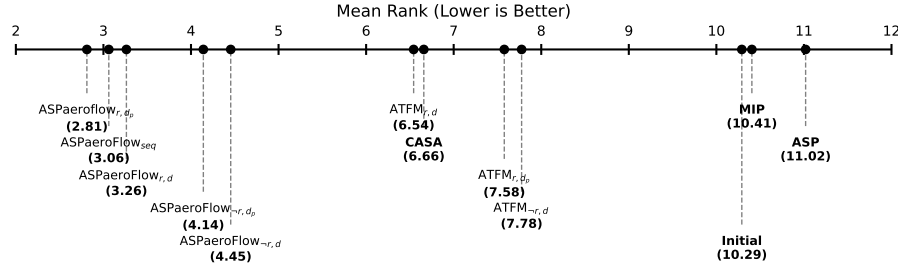
Option	Overload [#]	Arrival Delay [h]	Sector-Number [#]	Sector-Diff [#]	Reroute [#]	Reconfig [#]
r	9696.23 ± 424.57	20410.69 ± 662.55	90925.69 ± 1729.46	159.90 ± 6.00	3250.78 ± 66.75	16016.11 ± 810.76
$\neg r$	10526.43 ± 432.70	18834.70 ± 741.47	84098.67 ± 1629.73	144.91 ± 5.36	2929.04 ± 66.10	15651.06 ± 760.66
d	8995.97 ± 416.58	32960.03 ± 940.81	94563.00 ± 1806.79	162.97 ± 6.27	3465.57 ± 67.44	17611.50 ± 896.61
d_p	11226.69 ± 440.04	6285.36 ± 225.54	80461.35 ± 1539.89	141.84 ± 5.04	2714.25 ± 65.13	14055.68 ± 656.57
s	5269.25 ± 306.63	5192.79 ± 267.19	85727.03 ± 1647.63	304.81 ± 7.61	5035.24 ± 83.87	31667.18 ± 1077.64
$\neg s$	14953.42 ± 516.28	34052.61 ± 924.97	89297.32 ± 1713.15	0.00 ± 0.00	1144.58 ± 25.92	0.00 ± 0.00

on arrival delay (an average value of 5.28 ± 0.52), but is overall able to solve slightly more instances ($\#S : 263$ vs. $\#S : 207$). We emphasize that this should not be interpreted as ASP being in general a better solving technology than MIP; rather, the two approaches operate in fundamentally different feasible decision spaces. Specifically, we attribute the difference in results to ASP-P’s ability to utilize DAC actions, where ASP-P uses on average $K_{ds} = 188.00 \pm 8.60$, compared to 0 for MIP. A similar behavior can also be seen when comparing the results of MIP and ASP-P for EA-3x3 and EUR-10x10. EA-3x3 is a tiny graph with effectively no possibility for DAC ($|V| = 16$ with $\max |S| = 3$), whereas for EUR-10x10 the possibilities for DAC are plenty ($|V| = 117$ with $\max |S| = 10$). As expected, MIP performs better on EA-3x3, whereas ASP-P performs better on EUR-10x10. In the Appendix Table 6 we show the detailed results for the small and SOTA scenarios, for 5 selected variants: Baseline Initial, the best-performing heuristic variant for the small experiments *ASPaeroFlow_{r,d_p}*, CASA, and the exact methods MIP and ASP-P. More detailed results are shown in the supplementary material.

Scaling and Heuristic Approaches. In Figure 3 we show the scaling results comparing nominal capacity. The full scaling results comparing graph and flights size are available in the supplementary material. For the navpoint and flights scaling results, we compare 4 heuristic approaches on 10% nominal capacity (more in the supplementary material), where *ASPaeroFlow_{r,d}* and *ASPaeroFlow_{seq}* are able to solve more instances with lower arrival delay than CASA/ATFM_{r,d}. Interestingly, just comparing these results we can see that simultaneous optimization of *ASPaeroFlow_{r,d}* achieves competitive results compared to *ASPaeroFlow_{seq}*, as it is able to solve more instances on sizes $|V| \in \{CE - G, USA - G, CE\}$, whereas *ASPaeroFlow_{seq}* achieves better results on $|V| \in \{EUR - G, DACH\}$. This behavior can also be seen in Figure 3, where nominal capacity alterations are shown for large graphs



■ **Figure 3** Nominal capacity scaling results on the four real-world topography large instances, comparing heuristic and exact methods. Exact methods run into time-/memouts in this setting.



■ **Figure 4** Significance graph showing the results of the post-hoc pair-wise Wilcoxon tests, where ranks indicate a hierarchy. All differences are statistically significant ($p < 0.05$).

($|V| \in \{CE, DACH, EUR, USA\}$): the ASPaeroFlow variants use approximately an equal execution time, whereas CASA/ATFM-variants are slower. Furthermore, we can observe that the results are instance-dependent and while ASPaeroFlow-variants are able to solve instances with nominal capacity $\geq 20\%$, the methods relying on ATFM-measures are only able to solve instances $\geq 40\%$. On the SOTA scenarios, we show the strongest results in Table 6. ASPaeroFlow_{r,d,p}, ASPaeroFlow_{seq}, ATFM_{r,d,p}, and ATFM_{r,d} achieve the strongest results. The strong results for the ATFM only methods can be explained by studying the differences between the instances. While the generated instances BLOX, c0X, and mBLOX have little to no possibility for DAC, the industrial instance IC allows for plenty of reconfiguration. Therefore, the results of BLOX, c0X, and mBLOX, are similar, where the ATFM methods have a slight edge as they are (implicitly) more likely to consider reroutes – which also explains results of negative delay, which happen as for the SOTA instances the filed path is not necessarily the optimal one. Comparisons to SOTA work besides the paper introducing ASP-P and MIP [5] however, are hardly useful, due to differences in the model stemming from dynamic computation, (slightly) different definitions of demand/capacity measurement, objective functions, and time.

Statistical Significance. To evaluate the statistical significance of the algorithmic differences, we first performed a non-parametric Friedman test. The test indicated a statistically significant difference in the lexicographic mean ranks across the 12 variants ($\chi_F^2(11, N = 1889) = 15459.90, p < 0.001$). Consequently, we conducted post-hoc pairwise Wilcoxon signed-rank tests, applying the Holm-Bonferroni correction to adjust for multiple comparisons. The results demonstrate that all pairwise differences are statistically significant ($p < 0.001$), establishing a strict performance hierarchy of $ASPaeroFlow_{r,d_p}$, $ASPaeroFlow_{seq}$, $ASPaeroFlow_{r,d}$, followed by the others. The final ordering of the variants, based on their mean ranks, is depicted in Figure 4.

Sequential vs. Simultaneous Optimization. We observe that simultaneous $ASPaeroFlow$ is highly competitive to $ASPaeroFlow_{seq}$. However, relying solely on aggregate statistical tests or scaling results across the entire benchmark yields inconclusive results: while all simultaneous variants together solve and win more than $ASPaeroFlow_{seq}$, no single simultaneous variant beats $ASPaeroFlow_{seq}$ (Table 4). This occurs because sequential optimization possesses two inherent advantages: first, if an instance can be solved solely via DAC, sequential optimization achieves superior results since $K_d = 0$; second, the smaller search space per step permits more search steps compared to simultaneous optimization. Conversely, a major deficit exists: if pure DAC fails to eliminate overload, sequential optimization defaults to $ATFM_{r,d}$, which performs worse than simultaneous optimization. We compare in more detail the variants $ASPaeroFlow_{r,d}$ and $ASPaeroFlow_{seq}$: evaluating all instances ($n = 1889$) on the two methods, $ASPaeroFlow_{r,d}$ achieves $\#W : 864$ and $\#S : 1717$, while $ASPaeroFlow_{seq}$ achieves similar results ($\#W : 843, \#S : 1734$); with 182 draws. However, isolating the subset of instances where $ASPaeroFlow_{seq}$ fails to solve via DAC alone ($n = 1318$), $ASPaeroFlow_{r,d}$ demonstrates a clear advantage with $\#W : 864$ ($\#S : 1154$) compared to $\#W : 441$ ($\#S : 1171$) for $ASPaeroFlow_{seq}$; with 13 draws. This divergence is reflected in the statistical analysis. Comparing only these two variants across the full dataset ($n = 1889$), the Wilcoxon signed-rank test yields mean ranks of 1.49 for $ASPaeroFlow_{r,d}$ and 1.51 for $ASPaeroFlow_{seq}$ ($p = 0.61$, not significant). Note that the loss of significance compared to Figure 4 stems exclusively from reducing the test space from $k = 12$ to $k = 2$; without the other variants skewing the global rank distribution, the high number of ties between these two highly competitive variants becomes statistically apparent. Interestingly, when applying the test to the dataset where $ASPaeroFlow_{seq}$ fails to solve via DAC alone ($n = 1318$), the mean ranks shift to 1.34 for $ASPaeroFlow_{r,d}$ and 1.66 for $ASPaeroFlow_{seq}$, yielding a statistically significant difference ($p < 0.001$).

Ablation Study. Table 5 evaluates the impact of enabling rerouting (r), delaying (d), and restructuring (s). Each feature reduces overloads, however for $r/\neg r$ the results are strictly speaking inconclusive due to overlapping SEMs. Restructuring yields the largest reduction in both overload and arrival delay. While the heuristic optimizes locally, a sector split at t_1 persists, providing global temporal benefits by resolving future overloads at $t_2 > t_1$. Conversely, rerouting and delay effects remain temporally localized. For multi-leg flights, local delays may fail to resolve subsequent conflicts automatically and risk propagating downstream, inducing secondary overloads.

6 Conclusion

$ASPaeroFlow$ demonstrates that bounded instance-space decomposition provides a computationally viable heuristic for joint ATFCM, scaling exact ASP to industry-sized instances. Our ablation study reveals that DAC is the primary driver for overload reduction because

sector partitioning provides global temporal benefits, whereas flow measures risk downstream propagation of overloads. Furthermore, simultaneous optimization outperforms sequential approaches when DAC alone fails. Future work will address the open challenges of stochastic disruptions and the integration of *Explainable AI (XAI)* to extract rule-based justifications for automated scheduling decisions.

References

- 1 Dirk Abels, Julian Jordi, Max Ostrowski, Torsten Schaub, Ambra Toletti, and Philipp Wanko. Train Scheduling with Hybrid Answer Set Programming. *TPLP*, 21(3):317–347, 2021. doi:10.1017/S1471068420000046.
- 2 Alba Agustín, Antonio Alonso-Ayuso, Laureano F. Escudero, and Celeste Pizarro. On air traffic flow management with rerouting. Part I: Deterministic case. *Eur. J. Oper. Res.*, 219(1):156–166, 2012. doi:10.1016/J.EJOR.2011.12.021.
- 3 Hamsa Balakrishnan and Bala G. Chandran. Optimal large-scale air traffic flow management. Technical report, Massachusetts Institute of Technology, 2014.
- 4 Lucia Balázová, Richard Comploi-Taupe, Susana Hahn, Nicolas Rühling, and Gottfried Schenner. Smart expansion techniques for ASP-based interactive configuration. *TPLP*, 25(4):473–488, 2025. doi:10.1017/S147106842510029X.
- 5 Alexander Beiser, Markus Hecher, Nysret Musliu, and Stefan Woltran. Joint air traffic flow and capacity management via answer set programming. In *18th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR26)*, 2026.
- 6 Dimitris Bertsimas and Shubham Gupta. Fairness and collaboration in network air traffic flow management: An optimization approach. *Transp. Sci.*, 50(1):57–76, 2016. doi:10.1287/TRSC.2014.0567.
- 7 Dimitris Bertsimas, Guglielmo Lulli, and Amedeo Odoni. An integer optimization approach to large-scale air traffic flow management. *Operations Research*, 59:211–227, 2011. doi:10.1287/OPRE.1100.0899.
- 8 Dimitris Bertsimas and Sarah Patterson. The Air Traffic Flow Management Problem with Enroute Capacities. *Oper. Res.*, 46(3):406–422, 1998. doi:10.1287/OPRE.46.3.406.
- 9 Tatjana Bolić, Lorenzo Castelli, Luca Corolli, and Desirée Rigonat. Reducing ATFM delays through strategic flight planning. *Transp. Res. Part E: Logist. Transp. Rev.*, 98:42–59, 2017.
- 10 Razvan Bucuroiu and Stephanie Vincent. European network operations plan 2025/2026 - 2029. EUROCONTROL, 2025.
- 11 Aitichya Chandra, Sayan Hazra, and Ashish Verma. The integration of En route flow optimization, complex network clustering, and rule-based approach to airspace sub-sectorization for enhanced air traffic monitoring. *J. Air Transp. Res. Soc.*, 3:100036, 2024.
- 12 Richard Comploi-Taupe, Andreas Falkner, Susana Hahn, Torsten Schaub, and Gottfried Schenner. Interactive Configuration with ASP Multi-Shot Solving. In *ConfWS-23*, 2023.
- 13 Andrew Cook, editor. *European Air Traffic Management: Principles, Practice and Research*. Ashgate, 2007.
- 14 Veronica Dal Sasso, Franklin Djeumou Fomeni, Guglielmo Lulli, and Konstantinos G. Zografos. Planning efficient 4d trajectories in air traffic flow management. *EJOR*, 276(2):676–687, 2019. doi:10.1016/J.EJOR.2019.01.039.
- 15 Ramon Dalmau, Gilles Gawinowski, and Jacek Kopec. Multi-objective air traffic flow management through lexicographic optimisation. In *SID24*, 2024.
- 16 D. Delahaye, J.-M. Alliot, M. Schoenauer, and J.-L. Farges. Genetic algorithms for partitioning air space. In *CAIA-94*, pages 291–297, 1994.
- 17 Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web*, volume 5689, pages 40–110, 2009. doi:10.1007/978-3-642-03754-2_2.

- 18 Mohammed M. S. El-Kholany, Martin Gebser, and Konstantin Schekotihin. Problem Decomposition and Multi-shot ASP Solving for Job-shop Scheduling. *TPLP*, 22(4):623–639, 2022. doi:10.1017/S1471068422000217.
- 19 EUROCONTROL Experimental Centre. Innovative Slot Allocation: an Overview. EUROCONTROL, 2001.
- 20 Ke Feng, Dahai Liu, Yongxin Liu, Hong Liu, and Houbing Song. Graphdac: A graph-analytic approach to dynamic airspace configuration. In *IRI23*, pages 235–241, 2023. doi:10.1109/IRI58017.2023.00048.
- 21 David García-Heredia, Antonio Alonso-Ayuso, and Elisenda Molina. A Combinatorial model to optimize air traffic flow management problems. *Comput. Oper. Res.*, 112:104768, 2019. doi:10.1016/J.COR.2019.104768.
- 22 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, Philipp Wanko, and Max Ostrowski. Theory solving made easy with clingo 5 (extended version), 2016.
- 23 Martin Gebser, Philipp Wanko, Roland Kaminski, Benjamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Torsten Schaub, and Sven Thiele. *Clingo Guide*. Univ. of Potsdam, 2019.
- 24 Michael Gelfond and Nicola Leone. Logic programming and knowledge representation—The A-Prolog perspective. *Artif. Intell.*, 138(1-2):3–38, 2002. doi:10.1016/S0004-3702(02)00207-2.
- 25 Jeongseok Kim and Kangjin Kim. Enhancing Urban Air Mobility Scheduling Through Declarative Reasoning and Stakeholder Modeling. *Aerospace*, 12(7):605, 2025.
- 26 Go Nam Lui, Guglielmo Lulli, Luigi De Giovanni, Martina Galeazzo, Iciar Garcia-Ovies Carro, and Rebeca Llorente Martinez. A robust optimization approach for dynamic airspace configuration. In *ATRS25*, 2025.
- 27 Go Nam Lui, Guglielmo Lulli, M Florencia Lema-Esposto, and Rebeca Llorente Martinez. Airspace sector design: An optimization approach. In *SID24*, 2024.
- 28 Anh-Duong Nguyen, Loc Pham, Nathan Lindsay, Liang Sun, and Son C. Tran. Optimized eVTOL Aircraft Scheduling - An Answer Set Programming Based Approach. In *AIAA AVIATION/ASCEND 2024*, 2024.
- 29 Veronika Semmelrock and Gerhard Friedrich. Investigating the grounding bottleneck for a large-scale configuration problem: Existing tools and constraint-aware guessing. In Martin Gebser, Daniela Incezan, Francesco Ricca, Manuel Carro, and Mirosław Trzuszczynski, editors, *Proceedings 41st International Conference on Logic Programming*, Rende, Italy, 12–19th September 2025, volume 439 of *Electronic Proceedings in Theoretical Computer Science*, pages 482–495. Open Publishing Association, 2026. doi:10.4204/EPTCS.439.33.
- 30 Marina Sergeeva, Daniel Delahaye, and Catherine Mancel. 3d airspace sector design by genetic algorithm. In *MT-ITS15*, pages 499–506, 2015. doi:10.1109/MTITS.2015.7223300.
- 31 Qihang Xu, Yutian Pang, and Yongming Liu. Dynamic airspace sectorization with machine learning enhanced workload prediction and clustering. *J. Air Transp. Manag.*, 121:102683, 2024.
- 32 Min Xue. Airspace sector redesign based on Voronoi diagrams. *J. Aerosp. Inf. Syst.*, 6(12):624–634, 2009. doi:10.2514/1.41159.
- 33 Cheryl Wong Sze Yin, T.K. Venugopalan, and S. Suresh. A multi-objective approach for 3D airspace sectorization: A study on Singapore regional airspace. In *IEEE SSCI 2016*, pages 1–8, 2016.

A Appendix

A.1 AI Usage Statement

To improve spelling and wording we used ChatGPT (various ChatGPT-5 versions) and Gemini 3.1 Pro. Further, we used the same AI tools as coding assistants, for example to query software libraries, during debugging, and for documentation, however, substantial

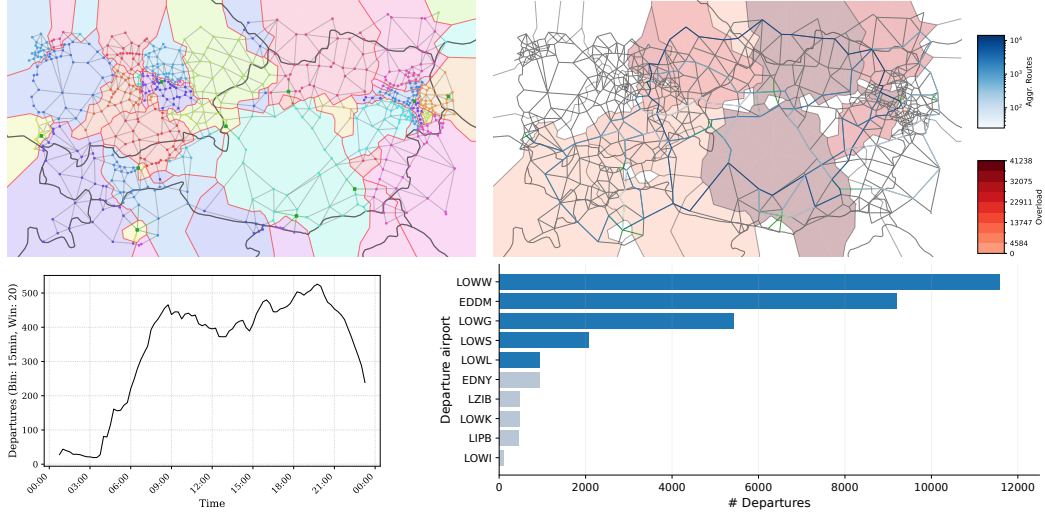


Figure 5 Figure displaying an instance of CE (Central Europe), from the data generator. Left top: Initial navpoint-sector figure. Right top: Initial total overload exceedances (K_o) and aggregated trajectory-counts figure. Left bottom: Filed flights departure times. Right bottom: Histogram of departure airports. Instance: 31622 flights, CE.

parts of the prototype were manually constructed (e.g., the entire ASP encoding). Lastly, we used AI tools to help in the data analysis part to generate Python scripts which are able to generate $\text{\LaTeX}$ tables and matplotlib figures.

A.2 Limitations

(1) *Heuristic nature.* ASPaeroFlow is a heuristic without global optimality guarantees; solution quality can depend on decomposition bounds and the overload resolution order. (2) *Operational fidelity.* Reroutes and sector partitions are generated on a navpoint graph and may violate unmodeled constraints; we also omit tactical interventions and weather uncertainty. Time is discretized for discrete optimization, which may induce errors. (3) *Capacity and occupancy abstraction.* Capacity assumes partitioning enables parallel staffing. (4) *Network scope and objectives.* We partially model airport/terminal constraints, so mitigating en-route overload may shift congestion elsewhere; efficiency is proxied mainly by arrival delay and stability terms, without explicit fuel/emissions, heterogeneous airline costs, or stakeholder fairness constraints.

A.3 Additional Experimental Results

In Table 6 we show the detailed results for the small and SOTA scenarios. Further, in Figure 5 we show an example of a generated instance produced by the data generator and Figure 6 shows the detailed significance results.

■ **Table 6** Small and SOTA scenarios (selected) results. We show overload (O) and arrival delay (D) of the considered variants.

Instance	EA-3x3 (Seed 150699)										CE-5x5 (Seed 150699)									
	Initial		ASPaeroFlow _{r,dp}		CASA		MIP		ASP-P		Initial		ASPaeroFlow _{r,dp}		CASA		MIP		ASP-P	
	O	D	O	D	O	D	O	D	O	D	O	D	O	D	O	D	O	D	O	D
10	10	0	0		56	0	20	0	0	1	9	1	0	0	0	0	2	0	2	0
20	22	0	0		311	0	156	0	15	7	33	5	0	0	30	0	11	0	5	0
30	53	0	0		687	0	919	0	65	24	54	13	0	0	102	0	79	0	11	0
40	80	0	0		1370	0	1247	0	230	54	102	26	0	0	213	0	231	0	90	0
50	117	0	0		2115	0	2740	3	422	88	96	37	0	0	342	0	506	0	187	0
60	133	0	0		3285	0	3351	12	482	95	127	38	0	0	401	0	562	0	193	0
70	178	0	0		4504	0	5366	35	382	134	160	59	0	0	748	0	903	0	305	0
80	204	0	0		6106	0	6302	59	250	157	178	76	0	0	1005	0	1288	18	271	3
90	241	0	0		7651	0	9517	88	186	192	230	94	0	0	1356	0	2013	38	208	9
100	287	0	0		9520	0	11034	120	111	241	199	111	0	0	1578	0	2365	56	170	17

Instance	IND-4x10 (Seed 150699)										USA-7x7 (Seed 150699)									
	Initial		ASPaeroFlow _{r,dp}		CASA		MIP		ASP-P		Initial		ASPaeroFlow _{r,dp}		CASA		MIP		ASP-P	
	O	D	O	D	O	D	O	D	O	D	O	D	O	D	O	D	O	D	O	D
10	15	0	0		69	0	36	0	7	0	3	1	0	0	10	0	1	0	1	0
20	21	0	0		192	0	152	0	34	0	5	12	0	0	66	0	27	0	10	0
30	45	0	0		577	0	503	0	166	0	28	26	0	0	312	0	251	0	63	0
40	70	0	0		904	0	1030	9	169	2	59	49	0	0	689	0	468	9	117	4
50	92	0	0		1395	0	1689	23	255	10	97	61	0	0	1031	0	968	15	146	5
60	136	0	0		2420	0	3115	77	133	38	125	80	0	0	1368	0	1513	26	129	10
70	135	0	0		3190	0	3241	72	177	35	147	89	0	0	1721	0	1791	30	168	9
80	186	0	0		4418	0	5481	131	85	54	177	117	0	0	2472	0	2871	47	161	14
90	193	0	0		5145	0	6016	138	111	77	186	140	0	0	2988	0	3168	65	175	18
100	233	0	0		6453	0	9523	181	87	100	197	180	0	0	4083	0	5584	94	186	38

Instance	EUR-10x10 (Seed 150699)										SOTA									
	Initial		ASPaeroFlow _{r,dp}		CASA		MIP		ASP-P		Initial		ASPaeroFlow _{r,d}		ASPaeroFlow _{seq}		ATFM _{r,d}		ATFM _{r,dp}	
	O	D	O	D	O	D	O	D	O	D	O	D	O	D	O	D	O	D	O	D
10	8	0	0		2	0	15	0	4	0	0	0	BLO1	466	0	0	423	0	431	0
20	14	0	0		87	0	69	0	11	0	3	0	BLO2	294	0	0	1897	0	1843	0
30	25	0	0		93	0	128	0	33	0	1	0	BLO3	1884	0	0	27074	0	27074	0
40	49	0	0		199	0	503	0	158	0	5	0	IC	3368	0	0	85	0	222	0
50	62	0	0		305	0	1045	7	172	0	13	0	c01	466	0	0	423	0	431	0
60	88	0	0		408	0	1621	23	208	0	41	0	c02	91	0	0	56	0	89	0
70	110	0	0		621	0	2370	42	194	0	68	0	c03	171	0	0	83	0	39	0
80	148	0	0		943	0	3637	62	210	0	117	0	mBLO1	14	0	0	-4	0	0	0
90	156	0	0		884	0	4009	69	194	0	99	0	mBLO3	1884	0	0	27074	0	27074	0
100	172	0	0		1181	0	5115	105	148	5	140						27074	0	27074	84

A.4 ASP Encoding Details

Generated Facts: Flight Trajectories

$\text{trj}(f, p)$	Defines the available trajectories (route-delay options) p for flight f .
$\text{fpl}(f, n, t)$	Specifies the presence for a filed flight f at navpoint n at time t .
$\text{actd}(f, t, p)$	Indicates the actual departure time t for flight f using path p .
$\text{plana}(f, t)$	Defines the originally planned arrival time t for flight f .
$\text{acta}(f, t, p)$	Denotes the actual arrival time t for flight f given the chosen trj . p .
$\text{npx}(\dots)$	Encodes sequential trajectory segments: $(f, p, n_{i-1}, t_{i-1}, n_i, t_i)$.
$\text{chp}(\dots)$	Propagates trajectory selection: $(f_{\text{aff}}, p) \leftarrow (f, p)$.

Generated Facts: Sector Configurations

$\text{config}(c, k_{\text{conf}})$	Sector configuration c associated with k_{conf} underlying conflicts.
$\text{sec_cap_conf}(\dots)$	Sets the capacity k_{cap} of sector s at time t under configuration c – assuming all flights fixed except to-be-optimized flights.
$\text{nav_assign_conf}(\dots)$	Maps navpoint n to sector s at time t within configuration c .

The following part of the encoding shows how guesses are performed. Line (1) shows how guesses for trajectories, and Line (2) shows guesses for sector configurations.

```

1 if{chtrj(ID,P):trj(ID,P)}1 ← flightID(ID).
2 if{chc(CONFIG):config(CONFIG,_)}1.

```

Next, we map the chosen trajectory to the navpoint trajectory that is actually flown, which is needed to compute the updated converted instance $\mathbf{M}'$ in Line 5 in Algorithm 4.1. Normal flights are shown in Lines (1) and (2), which map for each flown edge (v_0, v_1) the navpoints to the trajectory. Line (3) is a special case for handling atomic flights.

```

1 nvpt_f(ID,NAV0,T) ← actd(ID,TSTART,P),chtrj(ID,P),npx(ID,P,NAV0,T0,_,_),T=T0+TSTART.
2 nvpt_f(ID,NAV1,T) ← actd(ID,TSTART,P),chtrj(ID,P),npx(ID,P,_,_,NAV1,T1),T=T1+TSTART.
3 nvpt_f(ID,NAV0,T) ← actd(ID,TSTART,P),chtrj(ID,P),single_pos(ID,P,NAV0,T0),T=T0+TSTART.

```

Line (1) maps a chosen sector configuration to the effects it has on the navpoint sector

10:20 Decomposition Heuristics for Joint Air Traffic Flow & Capacity Management

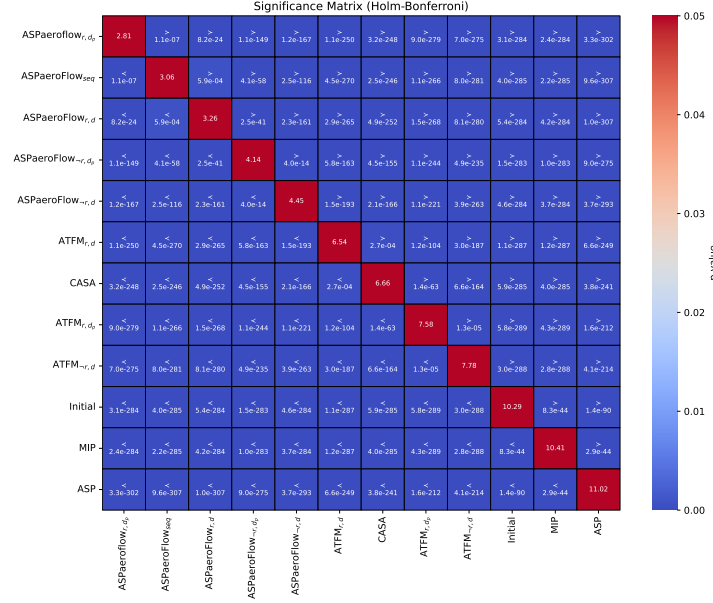


Figure 6 Matrix showing pairwise statistical significance results.

assignments and Line (2) maps the capacities of the chosen sector configuration to the *sec_cap* predicate.

```
1 nav_ass(NAV, SEC, TIME) ← chc(CONFIG), nav_assign_conf(NAV, SEC, TIME, CONFIG).
2 sec_cap(SEC, CAPACITY, TIME) ← chc(CONFIG), sec_cap_conf(SEC, CAPACITY, TIME, CONFIG).
```

Combining the chosen trajectory and the chosen config, we are left to map the flown navpoint trajectory to the actual effects on the sectors. Lines (1) and (2) handle the standard behavior for a trajectory; they encode the standard behavior for a flight $f \in \mathbf{F}$ reaching navpoint v_a at t_a , followed by arriving at v_b at t_b . Then f is in sector $sec^{-1}(v_a, t)$ in $t \in [t_a, t_a + \lfloor \frac{\Delta t}{2} \rfloor]$ and in sector $sec^{-1}(v_b, t')$ for $t' \in [t_a + \lfloor \frac{\Delta t}{2} \rfloor + 1, t_b]$. Additionally, we consider two special cases: first atomic flights (Line (3)) and then flights staying at a navpoint (Line (4)).

```
1 sec_f(ID, SECO, T) ← actd(ID, TSTART, P), chtrj(ID, P), time(T), nxp(ID, P, NAV0, T0, NAV1, T1), NAV0
  ≠ NAV1, T >= T0 + TSTART, T <= T1 + TSTART, DT = (T1 - T0) / 2, T <= DT + T0 + TSTART, nav_ass(NAV0, SECO, T).
2 sec_f(ID, SEC1, T) ← actd(ID, TSTART, P), chtrj(ID, P), time(T), nxp(ID, P, NAV0, T0, NAV1, T1), NAV0
  ≠ NAV1, T >= T0 + TSTART, T <= T1 + TSTART, DT = (T1 - T0) / 2, T > DT + T0 + TSTART, nav_ass(NAV1, SEC1, T).
3 sec_f(ID, SECO, T) ← actd(ID, TSTART, P), chtrj(ID, P), single_pos(ID, P, NAV0, T0), nav_ass(NAV0,
  SECO, T), T = T0 + TSTART.
4 sec_f(ID, SECO, T) ← actd(ID, TSTART, P), chtrj(ID, P), time(T), nxp(ID, P, NAV0, T0, NAV0, T1),
  T >= T0 + TSTART, T <= T1 + TSTART, nav_ass(NAV0, SECO, T).
```

If an overload occurs, we store this in the predicate *overload* (Line (1)), an arrival delay is stored in *arrival_delay* (Line (2)), and if a flight is rerouted, it is stored in *reroute* (Line (3)).

```
1 overload(SEC, T, LOAD - CAPACITY) ← sec_cap(SEC, CAPACITY, T), #count{ID: sec_f(ID, SEC, T)} =
  LOAD, LOAD > CAPACITY.
2 arrival_delay(ID, Y - T) ← chtrj(ID, P), plana(ID, T), acta(ID, Y, P).
3 reroute(ID) ← flightID(ID), not chtrj(ID, 0).
```