Tenth USA/Europe Air Traffic Management Research and Development Seminar (ATM2013)

Airspace Sectorisation using Constraint-Based Local Search

Peter Jägare, Pierre Flener, and Justin Pearson
Department of Information Technology
Uppsala University
751 05 Uppsala, Sweden
PJaegare@gmail.com, Pierre.Flener@it.uu.se (contact), Justin.Pearson@it.uu.se

***Abstract*—Airspace sectorisation provides a partition of a given airspace into sectors, subject to geometric constraints and workload constraints, so that some cost is minimised. Using the constraint programming paradigm, we define plug-and-play airspace sectorisation constraints and use them to model declaratively a problem of free-form static airspace sectorisation starting from a regular mesh of cells. We design a local search meta-heuristic that operates on the model and we compare it to the existing NEVAC Sector Builder Algorithm.**

***Index Terms*—airspace sectorisation; configuration; sector; constraint programming; constraint-based local search; tabu search**

I. Introduction

Airspace *sectorisation* provides a partition of a given airspace into a given (or upper-bounded, or minimal) number of sectors, subject to geometric constraints and workload constraints, so that some cost is minimised.

In this introduction, we first motivate the particular approach we have taken to airspace sectorisation, namely free-form static sectorisation starting from a regular mesh of cells (Section I-A). We also motivate the algorithm design technology we have chosen for implementing our problem model, namely the local-search approach to constraint programming (Section I-B). Finally, we highlight the contributions and major lessons of this work, before outlining the structure of the remainder of the paper (Section I-C).

A. Chosen Approach to Airspace Sectorisation

We distinguish between two approaches to airspace sectorisation [1]. In a *graph-based* approach, a graph is constructed whose vertices represent the intersections of the existing trajectories, and whose edges thus represent segments of the existing trajectories. The core problem of sectorisation is then essentially the NP-complete combinatorial problem of graph partitioning [2]. A graph partition does not define the sector boundaries, so actual sectors have to be constructed from the resulting vertex sets in a geometric post-processing step. In a *cell-based* approach, the airspace is initially partitioned into some kind of cells that are smaller than the targeted sectors, so that the combinatorial problem of partitioning these cells needs no geometric post-processing step.

In this paper, we take a cell-based approach, because the EUROCONTROL Experimental Centre (which commissioned this study) has a tool called ASTAAC, which partitions the airspace into a mesh of cells with a hexagonal or square base (of arbitrary side length and height). However, nothing in our model is specific to the shape of the input cells, so that our tool can start from an airspace partition of arbitrary granularity, including an existing sectorisation into elementary sectors, and produce a sectorisation of arbitrary granularity. For instance, rather than lumping hexagonal cells into elementary sectors (as shown in Section IV), we could lump elementary sectors into functional airspace cells (FAB), or ATC centres (ACC or ATCC), or areas of specialisation (AOS).

Airspace sectorisation can be invoked with different frequencies: *static sectorisation* is strategic or pre-tactical, while *dynamic sectorisation* is tactical but occurs at pre-determined times (and is thus different from *configuration*, which provides a schedule for the grouping and splitting of elementary sectors into control sectors that are suitable for a given number of available controllers and the expected traffic structure).

In this paper, we aim at a tool that is fast enough to be used either statically or dynamically, except that in the dynamic case we currently exclude *base-line* sectorisation [3], where the output sectorisation should be reasonably close to the input one (because the new sectorisation bears a transition cost from the previous one), and we focus instead on *free-form* sectorisation, where the output sectorisation can be arbitrarily different from the input sectorisation.

B. Chosen Optimisation Technology

Rather than *designing* an ad hoc algorithm, we advocate relying on off-the-shelf tools from mature optimisation technologies. With such a tool, one can on the one hand explicitly describe the constraints and optional cost function of a

This work has been financed by the European Organisation for the Safety or Air Navigation (EUROCONTROL) under its Care INO III programme (grant 08-121447-C). The content of the paper does not necessarily reflect the official position of EUROCONTROL on the matter.

problem in a declarative *modelling* language, without concern for the procedural aspects of how to solve the problem, that is satisfying the constraints while optionally minimising the value of a cost function. On the other hand, such tools offer generic *solving* algorithms that perform some kind of search (either systematic or stochastic) and are either push-button or user-tuneable by the provision of (meta-)heuristics. The first advantage of this clean separation of concerns between the declarative aspect of modelling and the procedural aspect of solving is that one reuses very advanced solving algorithms that are the results of decades of research. Another advantage is that constraints, cost functions, and (meta-)heuristics can be added or revised in plug-and-play fashion, which allows rapid prototyping and easy exploration of the algorithm design space. This is crucial in an area such as airspace sectorisation, where suitable combinations of constraints and cost functions still have to be identified.

In this paper, we rely on *constraint programming* (CP, see [4] for instance). A CP tool offers a very-high-level modelling language: every combinatorial substructure of a problem can be modelled naturally, because it can be captured by a modelling construct called a *constraint*, which is expressed on *decision variables* (or *unknowns*), each having a set of possible values, called its *domain*. For example, the ALLDIFFERENT$(x_1, \ldots, x_n)$ constraint requires the decision variables $x_1, \ldots, x_n$ to take pairwise different values in a solution; if $x_1 = 3$ is tried while solving the problem, then either the value 3 can be removed from the domains of $x_2, \ldots, x_n$ by an *inference* process called *propagation*, or the further occurrence of, say, $x_5 = 3$ can be sanctioned with a numeric penalty by an inference process called *penalty maintenance*. For airspace sectorisation now, assuming we start from n cells and aim at k sectors, let decision variable s_j of domain $\{1, \ldots, k\}$ represent the sector to which cell $j \in \{1, \ldots, n\}$ is assigned: another example constraint is CONNECTED$(s_1, \ldots, s_n)$, which requires any two cells i and j with $s_i = s_j$ to be reachable from each other in steps that only cross to a neighbour cell assigned to the same sector. While every CP solver comes with inference algorithms for the frequently used ALLDIFFERENT constraint, the CONNECTED constraint is more unusual and a user-designed inference algorithm may have to be provided for it.

Classical CP solvers work by *systematic search*, performing an intelligent exploration of the entire search space (the Cartesian product of the domains of the decision variables), interleaved with propagation at every search step and guided by a default or user-designed heuristic, until a solution is found (that is, until every decision variable is given a value from its domain so that all constraints are satisfied and, optionally, the value of the cost function is minimised) or a proof of the non-existence of solutions is established. Since some problems are very hard or typically have very large instances, modern CP tools interface their modelling language also with stochastic *local search* (LS, see [5] for instance): the CP solver then performs what is called *constraint-based local search* (CBLS, see [6]), interleaved with penalty maintenance at every search step and guided by a default or user-designed (meta-)heuristic.

In this paper, we opt for CBLS, because our chosen airspace sectorisation constraints (see Section II) are hard to satisfy (see Section IV) and because our ultimate objective is to be able to sectorise the entire European airspace in one go, starting from a possibly very fine-grained mesh of cells. Note that we do *not* claim that CP in general, or CBLS in particular, is best for airspace sectorisation: we use CP simply because it is our area of deepest optimisation expertise and because we have argued that some optimisation technology should be used rather than none. In [1], we provide a tutorial on CP and survey its use in air traffic management.

In summary, the slogan of CP is that a *constraint program* is made from two orthogonal components, namely a declarative *model* (with the definitions of the decision variables, their domains, the constraints, and optionally the cost function of the problem), plus a procedural *search* component, consisting of a (meta-) heuristic that guides a search algorithm. Reusable components called *constraints* not only facilitate the expression of the model, but also accelerate the search by providing powerful inference algorithms that are specific to the combinatorial substructure captured by the constraints.

C. Contributions, Lessons, and Structure of this Paper

The *contributions* and major *lessons* of this paper are as follows:

- We define plug-and-play airspace sectorisation constraints and design inference algorithms for them that are suitable for constraint-based local search (CBLS). Some of these constraints are geometric; since current CBLS solvers do not feature built-in geometric constraints, this is also a contribution to CBLS itself, and we conjecture that these constraints will be useful in other application areas.
- We declaratively model an airspace sectorisation problem with the help of these constraints, and design a local search (meta-)heuristic that operates on the model and we compare it to the NEVAC Sector Builder Algorithm, which comes with ASTAAC.
- We demonstrate that constraints can be used actively in the *processes* of modelling and computing a sectorisation, rather than only passively in evaluating the *results* of a sectorisation algorithm.

The *structure* of the remainder of this paper is as follows. In Section II we describe our declarative constraint model, while in Section III we describe our local search algorithm that operates on our constraint model. Next, in Section IV, we discuss the computational performance of our approach and compare it to NEVAC. Finally, in Section V, we summarise this work, compare it to related work, and outline promising directions for future work.

II. MODEL

We now discuss the declarative aspects of the airspace sectorisation problem we tackle, namely a formal model of the problem, that is its parameters (Section II-A), its decision variables (Section II-B), its constraints (Section II-C), and its cost function (Section II-D).

A. Parameters and Preprocessing

The parameters of a sectorisation can be divided into three categories. First, the mesh is specified by the horizontal, lower, and upper bounds of the airspace to sectorise, the horizontal shape (square or hexagonal) of the cells, as well as the horizontal and vertical sizes of the cells. Second, the date and time period specify which historical traffic data is to be used for generating trajectories through the specified mesh of cells. Finally, the desired number k of sectors is to be used during the sectorisation.

Preprocessing (by ASTAAC) computes the actual mesh and trajectories from the first two categories of parameters.

Each cell is endowed with longitude, latitude, and altitude (which are all used to create an initial sectorisation), the list of its neighbouring cells, and a workload value. For each portion of airspace in general, and for each cell in particular, there are typically three kinds of workload: the *monitoring workload*, the *conflict workload*, and the *coordination workload*; the first two workloads occur inside the cell, and the third one between the cell and adjacent ones. Since coordination workload is not additive when amalgamating cells into sectors, it is not used here. The other two workloads are computed for each cell, based on the number of conflicts and the flight time inside the cell. The *workload of a cell* is then the sum of its monitoring and conflict workloads.

Each trajectory runs through a sequence of cells, with time stamps describing entry and exit of each cell. These values are used for defining the convexity and minimum-dwell-time constraints described below.

B. Decision Variables and their Domains

Assuming the computed mesh has n cells and considering that we aim at k sectors, let decision variable s_i of domain $\{1, \ldots, k\}$ represent the sector to which cell $i \in \{1, \ldots, n\}$ is assigned.

C. Constraints and their Penalties

We now discuss the constraints of our model. A *hard* constraint must be satisfied, whereas a *soft* constraint can be violated, although its violation earns a numeric penalty used in the cost function (explained in the following sub-section).

a) Balanced Workload: For each sector, the workload must be within some given balance factor β (we arbitrarily chose $\beta = 1.05$ for our experiments) of the average across all sectors. The *workload of a sector* is here defined as the sum of the workloads of its constituent cells.

This constraint is soft. Workload imbalance is here measured by comparing the workload of each sector to the average sector workload a, which is the known total workload of all cells divided by the given number k of sectors. The *workload penalty of a sector* is zero if its workload w is such that $w \leq \beta \cdot a$, and $\beta \cdot a - w$ otherwise. Only sectors with too large workloads are directly penalised, on the reasoning that an underworked air traffic controller is not a problem in the same way that an overworked one is. The *penalty of the balanced-workload constraint*, that is the workload penalty of a sectorisation, is the sum of the workload penalties of all the sectors.

b) Minimum Dwell Time: Every flight entering a sector must stay within it for a given minimum amount δ of seconds (we use $\delta = 60$ seconds in our experiments), so that the coordination work pays off and that conflict management is possible.

This constraint is soft. The *dwell-time penalty of a flight* is zero if it dwells at least δ seconds in each sector it passes, and otherwise the minimum number of cells that would need to be reassigned to eliminate the constraint violation. The *penalty of the minimum-dwell-time constraint*, that is the dwell-time penalty of a sectorisation, is the sum of the dwell-time penalties of all the flights.

c) Convexity: We do not consider convexity of the sectors in the usual geometric sense, but a *trajectory-based* convexity, meaning that no trajectory enters the same sector more than once.

This constraint is soft. Convexity is here measured by dividing each trajectory into sequences of contiguous cells that belong to the same sector: each such sequence represents an uninterrupted pass through a sector. The *convexity penalty of a trajectory* is zero if all its sequences pass through distinct sectors, and otherwise measured in a manner illustrated by the following example. Consider the sequence $[1, 1, 2, 1, 1, 1, 4, 4, 1, 1, 4, 4]$. Each number represents a cell the trajectory passes through, the value being the sector that the cell is part of. This trajectory revisits sector 1 twice, and sector 4 once. Additionally, between visits to sector 1, the trajectory passes through three cells from sectors 2 and 4, and between visits to sector 4 it passes through two cells from sector 1 (those being the cells that would have to be reassigned in order to fix the violation). When there are a total of $r > 0$ revisits and a total of b cells between the different visits, we define the convexity penalty of a trajectory to be $\gamma \cdot r + b$ (we experimentally determined $\gamma = 3$ for our experiments), and 0 if $r = 0$. In the given example, we have $r = 2 + 1 = 3$ revisits and $b = 3 + 2 = 5$ cells, so the penalty is $3 \cdot 3 + 5 = 14$. The *penalty of the convexity constraint*, that is the convexity penalty of a sectorisation, is the sum of the convexity penalties of all the trajectories.

d) Compactness: A sector must have a geometric shape that is easy for the controllers to keep in mind.

This constraint is soft. Compactness (or rather the lack thereof) is here measured by the surface area of the border between sectors, calculated in a simplified manner that normalises the area of each face (vertical or horizontal) of a cell to 1 unit. Due to the nature of the constraint (there is no compact versus non-compact distinction, only more or less compact), the *compactness penalty of a sector* is zero only if the sector covers the entire airspace to be sectorised (thus having no neighbours), which is possible only when $k = 1$. If the only part of the sector that touches another sector is a flat 5 by 10 wall of cells, the penalty is 50, and if the border area is larger and uneven, then the penalty is larger. In different words, the penalty is the number of distinct pairs (a, b) such that a is a cell in the sector and b is a neighbour of a that is assigned to a different sector. Squaring this penalty value seems to give slightly better results. The *penalty of the compactness constraint*, that is the compactness penalty of a

sectorisation, is the sum of the compactness penalties of all the sectors.

e) Connectedness: A sector must be a contiguous portion of airspace and can thus not be fragmented into a union of unconnected portions of airspace. This constraint is hard, and is actually automatically enforced by the search algorithm (see Section III-B) rather than being an explicit constraint of the model.

f) Minimum Distance: Each existing trajectory must be inside each sector by a minimum distance (say 10 nautical miles), so that each conflict management is entirely local to one sector. This constraint is currently not part of our model, for reasons explained in Section IV.

D. Cost Function and Weights

The only hard constraint is connectedness, automatically enforced by the search algorithm. The cost function is here the weighted sum of the penalties of the soft constraints. The *weights* were not chosen for any particular theoretical reason but rather experimentally determined, based on what seems to give the best results. The experimental results (in Section IV) include results from different sets of weights, but we chose as default weights 1 for balanced workload and compactness, and 6 for minimum dwell time and convexity. Typically, the compactness term of the cost ends up being larger than the other costs combined. Increasing the other weights to the point where this is not the case has a negative effect on the resulting shapes, outweighing the improvement on other constraints.

We do not explicitly minimise the number of entry points into the resulting sectors, on the premise that this value is strongly tied to both convexity and compactness.

III. Search

A local search algorithm starts from an initial candidate solution and iteratively tries to improve it by making at each iteration a *move*, which is a reassignment of some of the decision variables to other values in their domains, until a sufficiently good solution is found or an allocated amount of resources (such as an amount σ of runtime seconds or an amount of moves) is exhausted.

After explaining the *constraint-based* local search (CBLS) philosophy on local search (Section III-A), we outline our CBLS algorithm, showing how it fulfils the required conditions of preserving the hard constraints (Section III-B), generally improving the current candidate solution (Section III-C), and escaping local minima of the cost function (Section III-D).

A. Constraint-Based Local Search (CBLS)

For each constraint C, there are two CBLS inference algorithms (as announced in Section I-B). The first algorithm maintains the penalty of C, denoted by $\text{penalty}(C)$, after each actually made move (here a reassignment of one cell from a sector to another sector). The penalty of a constraint must be zero if and only if the constraint is satisfied, and otherwise some positive number (that is suitable for guiding the search). The penalty is best computed *incrementally*, that is by only considering the decision variables actually modified by the move, rather than all the decision variables. The second algorithm evaluates the increase, denoted by $\Delta(C, m)$, of the penalty of C upon a probed move m. The penalty increase is *not* computed by first storing the current $\text{penalty}(C)$ as pre, then tentatively making the move m, which triggers the (incremental) update of $\text{penalty}(C)$, so that $\Delta(C, m)$ can be set to $\text{penalty}(C) - pre$, and finally undoing the move m; instead, it is best computed *differentially*. Depending on the constraint, these algorithms may require auxiliary data structures in order to have the lowest possible time complexity (ideally constant time), as these algorithms are at the core of the search and are typically invoked millions of times. The encapsulation of these inference algorithms in a constraint makes them declaratively available when modelling a problem as well as reusable for other problems: this is what distinguishes CBLS from classical local search.

Giving these inference algorithms for the constraints of our model and their penalties (as given in Section II-C) is beyond the scope of this paper (and would take too much space). Note however that the given penalties of our constraints satisfy the stated condition of being zero if and only if the constraint is satisfied, and positive otherwise.

For a model with constraint set S, a CBLS solver has a master algorithm that does two things. First, after each actually made move, it incrementally maintains the sum $\sum_{C \in S} \text{penalty}(C)$ of the penalties of all the constraints C of the model. If this quantity is positive, then there necessarily is some violated constraint. Second, for each probed move m, it differentially evaluates the sum $\sum_{C \in S} \Delta(C, m)$ of the penalty increases of all the constraints C of the model: an *improving move* has a negative penalty increase, a *worsening move* has a positive penalty increase, and a *best move* has the minimal penalty increase. These two sums can be queried by the user-provided heuristic and meta-heuristic towards driving the search in a good direction.

For our airspace sectorisation model with constraint set $S = \{a, b, c, d\}$ (with a being the balanced workload constraint, b the minimum dwell time constraint, c the convexity constraint, and d the compactness constraint), our CBLS heuristic and meta-heuristic are outlined in Algorithm 1 and are described next. Note that, due to the CBLS framework, the heuristic need not contain any code for the operations by the master algorithm, and hence is highly readable.

B. Heuristic: Initial Candidate Solution and Probed Moves

The initial candidate solution (that is, the initial sectorisation here) is computed (line 1) by assigning each decision variable s_j (representing the sector to which cell $j \in \{1, \ldots, n\}$ is assigned) to a sector identifier of the domain $\{1, \ldots, k\}$ in the following fashion, aiming at a low penalty of the compactness constraint. First, give each sector a single randomly picked cell. Then, give the sector with the currently smallest workload a still unassigned cell that neighbours one of its cells. When picking this cell, candidates are evaluated and scored according to three criteria: the candidate scores one point if it shares both longitude and latitude with a cell already in the sector, one

Algorithm 1 Our CBLS heuristic and meta-heuristic for airspace sectorisation

```
1: s ← initial sectorisation {see Section III-B for details}
2: s* ← s {s* is the current best sectorisation}
3: i ← 0 {i is the iteration counter}
4: while there is a violated constraint and no time-out do
5:   if i mod 600 > 500 then
6:     pick a random legal non-tabu move m minimising
       Δ(d, m) {improve compactness}
7:   else if random chance of 50/(500 + i) then
8:     pick a random legal move m {diversify}
9:   else
10:    pick a random legal non-tabu best move m
11:  end if
12:  if s is better than s* and m is worsening then
13:    s* ← s
14:  end if
15:  make the move m, destructively updating s
16:  while there is an improving move m' similar to m do
       {see Section III-C for details}
17:    make the move m', destructively updating s
18:  end while
19:  i ← i + 1
20: end while
21: if s is better than s* then return s else return s* endif
```

point if it shares both longitude and altitude with a cell already in the sector, and one point if it shares both latitude and altitude with a cell already in the sector. By always picking a highest-scoring cell, the sectors are kept more or less box-shaped, so that the initial sectorisation indeed has a low penalty of the compactness constraint. Repeat until there are no unassigned cells left. If this process ends with one sector having less than half the average sector workload, then start over. In our experiments (see Section IV), it was very rare that two or more such restarts were needed, but even then this polynomial-time initialisation takes but a tiny fraction of the search time. Actually, five initial sectorisations are generated, and the most compact one is picked to use.

Note that the initial candidate solution necessarily satisfies the connectedness constraint. If we restrict the probed moves to cell reassignments that take us to another connected sectorisation, then (as announced in Section II-C) the connectedness constraint is indeed hard (in the sense that it is guaranteed to be satisfied no matter how many or how few moves are made) and we need not design CBLS inference algorithms for it; in fact, the constraint need not even be made explicit in the model.

In order for a probed move, that is the reassignment of a cell to another sector, to be considered *legal*, each of its neighbours of the same sector must be connected to every other cell in the sector by a path that remains within the sector but does not pass the cell to be reassigned. However, since determining whether a probed reassignment would lead to disconnectedness is computationally expensive, a slightly different constraint is instead enforced, namely *local connectedness*: paths connecting cells are only searched for in a small area, consisting of those cells that are either neighbours or neighbours of neighbours to the cell being considered. A connected sectorisation computed by the initialisation described above is necessarily locally connected.

Only enforcing local connectedness has the unfortunate side effect that some sectorisations have undesirable shapes but a compactness penalty that cannot be improved. Consider two strands of cells reaching out from one sector into another, and then meeting, forming an arc like the handle of a mug. (We have actually observed this phenomenon in our experiments.) Since disconnecting such an arc violates local connectedness, this shape once formed cannot be undone. To avoid this, moves whose inverse would be illegal are also illegal.

Our heuristic destructively updates the current sectorisation s, initialised to the initial one (lines 1, 15, and 17). It also maintains the current best sectorisation s^*, initialised to the initial one (lines 2, 12 to 14, and 21), and a move iteration counter i, initialised to zero (lines 3 and 19). As long as there are violated constraints and the allocated σ runtime seconds have not elapsed (line 4), our heuristic iterates as follows: it first probes moves modulo the meta-heuristic of Section III-D (lines 5 to 14), then makes a move (line 15), and finally decides whether a more complex move (see Section III-C) would actually be suitable (lines 16 to 17). Upon termination, it returns the best solution (line 21).

C. Heuristic: Optimisation

We consider all moves that reassign a cell that has a neighbour in a different sector to a neighbouring sector, and probe the legal moves in order to pick a random best move (line 10). Note that a best move need not improve the current sectorisation, but may worsen it. If the current sectorisation s is better than the current best sectorisation s^* but the picked move will worsen it, then the current best sectorisation is updated (lines 12 to 14) before making the move (line 15).

As described in Section III-D below, a move may not be probed even if it is legal.

Since the number of cells can be very large (in the tens or hundreds of thousands), in order to speed up the search, after each move we also consider the cells in a small area around the reassigned cell, as well as cells that share a trajectory with that cell (lines 16 to 18). Only the sector the cell was switched to is considered for these moves, and only improving moves are performed. At the start of the search, the considered small area consists of the cells within three steps of the reassigned cell. To achieve a more fine-grained search, this distance is decreased to two steps after $1/3 \cdot \sigma$ seconds, and to one step after $2/3 \cdot \sigma$ seconds.

D. Meta-Heuristic: Escaping Local Optima

There are many meta-heuristics to avoid being stuck in a local optimum of the cost function (with no hope of reaching its global optimum), and we use several.

Our main method is a widely used meta-heuristic called tabu search [7]: every time a cell is reassigned, it is marked as being *tabu* for some number of moves. This prevents the search, once it has exhausted all immediate improvements, from switching

back and forth the sector membership of some particularly inconsequential cell. Generally, we mark cells as being tabu for $\tau = 200$ iterations, though for $n < 8000$ cells we choose instead $\tau = n/40$ iterations.

In the case of reassigning multiple additional cells in a small area around a reassigned cell (as described in Section III-C), the tabu tenure τ is ignored when considering the legality of a move, and the cells that are reassigned are considered tabu for only $\tau/3$ moves.

In addition to tabu search, we occasionally make a random legal but possibly tabu move instead of a best one in order to diversify the search (line 8). The chance of random moves starts at $50/500$ (or 10%), but decreases by adding one to the denominator at each iteration (lines 7 and 19).

Finally, for the last 100 out of every 600 iterations (line 5), we consider only the penalty increase of the compactness constraint when probing non-tabu moves (line 6), instead of the penalty increase of all the constraints.

IV. Experimental Results

We implemented our model using COMET [8], a state-of-the-art CP tool, which features both systematic search and constraint-based local search (CBLS) solvers. We implemented our search (meta-)heuristic and the inference algorithms of our new constraints using the CBLS solver [6] of COMET.

The preprocessing and part of the result evaluation are done by the research tool ASTAAC (*Arithmetic Simulation Tool for ATFCM and Advanced Concepts*) developed by EUROCONTROL. Before the sectorisation, it computes from the problem parameters the partition of the airspace into cells and extrapolates air traffic data from historical data. After the sectorisation, it computes the actual sector workloads (whose balance is targeted by the workload constraint, with balance parameter $\beta = 1.05$), the actual number of sector entry points (which we deemed not needing explicit handling by the model), the actual number of re-entering flights (whose number is kept low by the convexity constraint, with weight parameter $\gamma = 3$), and the actual number of short dwell times (whose number is kept low by the minimum dwell time constraint, with dwell parameter $\delta = 60$ seconds), as well as providing a 3D visualisation of the resulting sectors. The version of ASTAAC we used is "2.5.0.2 SECTO".

Along with ASTAAC comes a sectorisation algorithm called *NEVAC Sector Builder Algorithm*, with *NEVAC* standing for *Network Estimation Visualisation ACC Capacity*. For brevity, we refer to that algorithm as NEVAC. We have used it for comparison with our own algorithm. A representative sectorisation computed by NEVAC is given in Figure 1.

Our experiments with both algorithms were run on the same computer, which has an Intel Core i7 processor, with 2.2 GHz and 8 GB of RAM. For our algorithm, the COMET portion was allowed to run for five minutes ($\sigma = 300$ seconds), while NEVAC ran until it deemed itself finished (in the case with about 35000 cells, this took approximately two minutes).

Our experiments were run on two airspaces (Munich North ACC, Germany, and Madrid South ACC, Spain), with two hexagonal cell sizes each (embedded in squares of side sizes

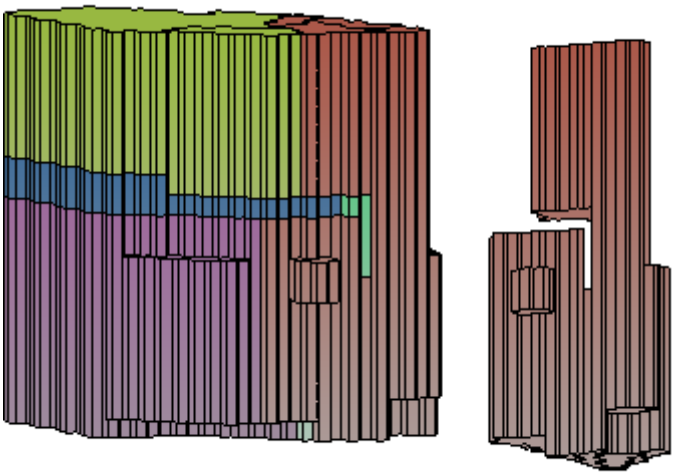

Figure 1. A representative sectorisation by NEVAC, for Munich North ACC, with 4,879 cells of size 10 NM × 10 NM × 2000 feet. Left: all five sectors. Right: an awkward sector.

5 NM and 10 NM, respectively 10 NM and 20 NM). For all experiments, the lower limit of the airspace was set to 100 flight levels, and the upper limit to 500 flight levels. We chose to target the NEVAC default value of $k = 5$ sectors. All experiments were run using trajectory data from 2008-07-12, between 10:00 and 12:00.

For our algorithm, we ran two variations of the weights: one with the defaults mentioned in Section II-D, and one where the default weights of the minimum-dwell-time penalty and the convexity penalty were multiplied by five, in order to examine how varying weights affect the balance between compactness and the other constraints.

The results of our algorithm are averages over 5 runs, rounded to the nearest integer. NEVAC is deterministic and thus the result of a single run is sufficient.

Visually, some results by our algorithm are given in Figures 2 to 4. The shapes of the resulting sectors were judged by informal visual inspection, and sectorisations were rated by us as "poor" (if at least one sector is very awkward), "okay" (if some sectors are probably short of acceptable, but not outright terrible), or "good" (if all sectors could plausibly be viewed as having acceptable shapes).

Numerically, all results are compiled into Tables I to IV. Our algorithm is called ASTRA there (that being a temporary name, because simply the name of our research group). We can conclude that we have similar results in terms of entry points, despite not directly optimising for them, and clearly superior results in terms of avoiding short dwell times. For reentering flights the results are more mixed, being sometimes worse, sometimes better. In general, the results of our algorithm have significant variance except for workload, which is consistently reasonably balanced. The five-fold increase of the weights of the minimum-dwell-time and convexity constraints almost always pays off on the numeric evaluation criteria, but some-

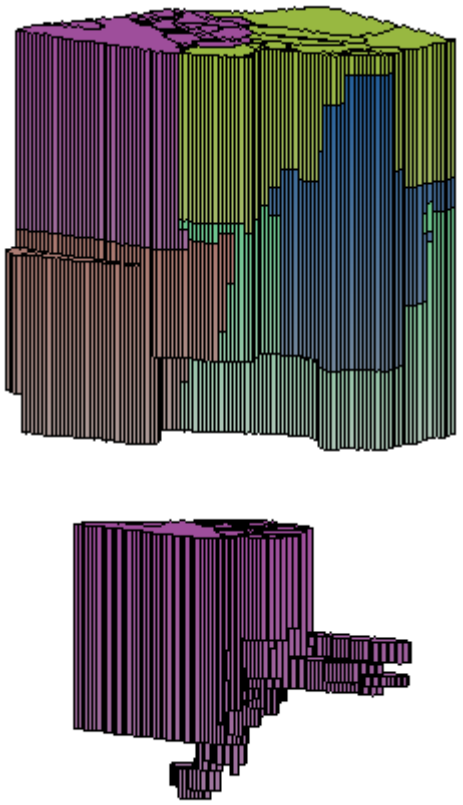

Figure 2. A fairly representative sectorisation by our algorithm, for Munich North ACC, with 35,306 cells of size 5 NM × 5 NM × 1000 feet. Top: all five sectors. Bottom: an awkward sector.

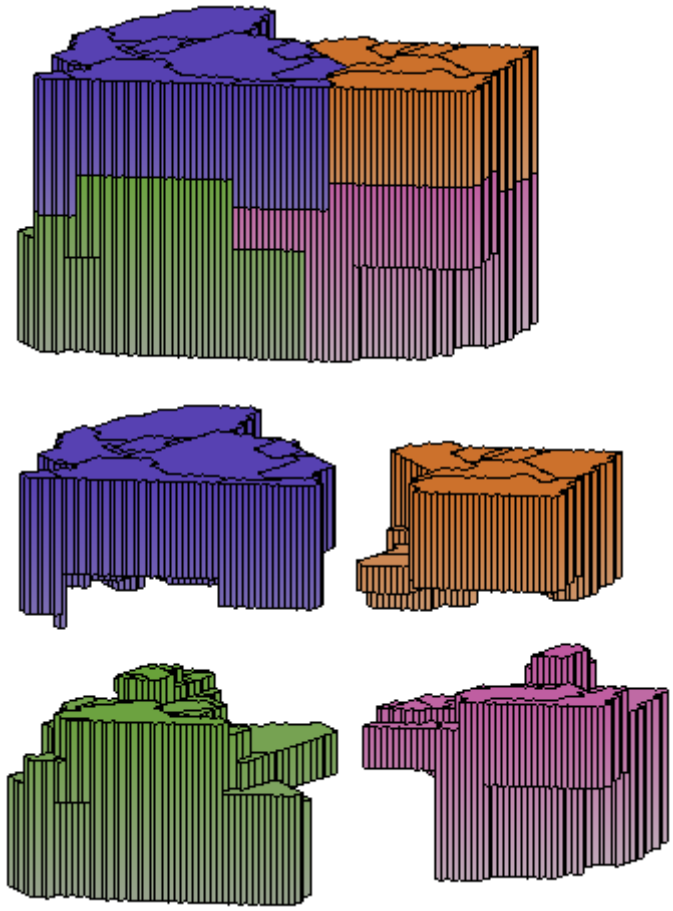

Figure 3. A good sectorisation by our algorithm, for Madrid South ACC, with 14,088 cells of size 10 NM × 10 NM × 2000 feet. Top: all five sectors. Bottom: four (of five) sectors separated out for a closer look.

times backfires on the visual criterion (the compactness of the resulting shapes).

ASTAAC features an option of lumping some of the hexagonal cells into ATC functional blocks (AFBs) prior to the actual sectorisation, (by NEVAC or our algorithm), with a guaranteed enforcement of the minimum-distance constraint. However, when starting from a mixed partition of the airspace into hexagonal cells and AFBs, (at least) our algorithm has difficulties in achieving a low penalty of the compactness constraint, because the AFBs tend to have non-compact shapes. Hence we did not use this feature in the experiments mentioned above, even though we were planning on using it (by not having a minimum-distance constraint). This means that the search space for a given problem instance is much larger (because we start only from the hexagonal cells) and that the problem itself is harder (because we also have to enforce the minimum-distance constraint, which is now left as future work).

V. Conclusion

We now summarise the contributions of this paper (in Section V-A), compare them with the state of the art (in Section V-B), and outline promising directions for future work (in Section V-C).

A. Summary

We have defined reusable plug-and-play airspace sectorisation constraints and we have designed penalty-maintaining inference algorithms for them that are suitable for constraint-based local search (CBLS). Our convexity and compactness constraints are geometric; since current CBLS solvers in general, and the used COMET solver in particular, do not feature built-in geometric constraints, this is also a contribution to CBLS itself, and we conjecture that these constraints will be useful in other application areas.

We have declaratively modelled, with the help of these constraints, a problem of free-form static airspace sectorisation starting from a regular mesh of cells. We have designed a constraint-based local search (meta-)heuristic, based on tabu search, that operates on the model, and we have compared it to the NEVAC algorithm.

We have demonstrated that constraints can be used actively in the *processes* of modelling and computing a sectorisation, rather than only passively in evaluating the *results* of a sectorisation algorithm.

Table I
EXPERIMENTAL RESULTS, MUNICH NORTH ACC, CELL SIZE 5 NM × 5 NM × 1000 FEET, $n = 35306$ CELLS, $k = 5$ SECTORS

Algorithm	Compactness	Workload range	Entry points	Reentries	Short dwell times
NEVAC	okay	32–38	210	20	86
ASTRA	okay	31–38	216	35	28
ASTRA 5-fold weights	okay	31–38	211	22	16

Table II
EXPERIMENTAL RESULTS, MUNICH NORTH ACC, CELL SIZE 10 NM × 10 NM × 2000 FEET, $n = 4879$ CELLS, $k = 5$ SECTORS

Algorithm	Compactness	Workload range	Entry points	Reentries	Short dwell times
NEVAC	okay	44–49	166	61	80
ASTRA	poor to okay	42–49	178	61	36
ASTRA 5-fold weights	poor to okay	40–49	169	41	26

Table III
EXPERIMENTAL RESULTS, MADRID SOUTH ACC, CELL SIZE 10 NM × 10 NM × 2000 FEET, $n = 14088$ CELLS, $k = 5$ SECTORS

Algorithm	Compactness	Workload range	Entry points	Reentries	Short dwell times
NEVAC	okay	49–54	229	44	83
ASTRA	okay to good	45–54	221	53	32
ASTRA 5-fold weights	poor to okay	47–54	217	39	29

Table IV
EXPERIMENTAL RESULTS, MADRID SOUTH ACC, CELL SIZE 20 NM × 20 NM × 4000 FEET, $n = 1959$ CELLS, $k = 5$ SECTORS

Algorithm	Compactness	Workload range	Entry points	Reentries	Short dwell times
NEVAC	okay	52–61	164	30	36
ASTRA	okay to good	52–59	153	16	12
ASTRA 5-fold weights	okay to good	53–59	156	11	3

Figure 4. A poor sectorisation by our algorithm, for Munich North ACC, with 4,879 cells of size 10 NM × 10 NM × 2000 feet. Top: all five sectors. Bottom: a very awkward sector.

B. Related Work

Related work is compared in Table V. We focus on *cell-based* approaches: the inputs are either cells with a hexagonal or square base (whose typical side length in nautical miles is sometimes indicated) or existing elementary sectors. The outputs are either elementary sectors, or control sectors, or an entire ATCC. Dimensionality nD means that the sectorisation is computed in n dimensions, while 2.5D means that the sectorisation is only computed in 2 dimensions, because airspace layers are considered to be independent. Constraints are only marked if they are explicitly mentioned in the corresponding paper; some unmarked constraints may have been taken for granted by the authors of some papers or may turn out to be logical consequences of the post-condition of their sectorisation algorithm. The given cost functions are meant to be minimised. The constraints and cost function of a model can be implemented using any combination of algorithm design methodologies or optimisation technologies, such as stochastic local search (LS), constraint programming (CP), mixed integer programming (MIP), evolutionary algorithms (EA), computational geometry, or ad hoc algorithm design. An airspace sectorisation tool can be tested at continental scale, such as the European Civil Aviation Conference (ECAC) area or its core 11 countries along the London-Frankfurt-Rome axis, or over a given number of ATCCs. An airspace sectorisation tool can be tested on historical data, or data extrapolated from historical data according to some forecast of future flight patterns and volumes, or data resulting from fast-time simulations of historical or extrapolated flight schedules.

The full details behind Table V are given in our sectorisation

Table V
RELATED WORK

Criterion \ Paper	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	this
Input granularity	hex(24)	elem	sq	sq(1)	elem	hex	elem	elem	hex(24)	hex	hex / sq
Output granularity	elem	ATCC	elem	elem	control	elem	control	control	elem	elem	elem
Dimensionality	2.5D	3D	2D	2D	3D	2.5D	(n/a)	2.5D	3D	3D	3D
Constraints:									(none)		
Balanced size		hard									
Balanced workload	hard	hard	hard				hard			hard	soft
Bounded workload				hard	hard						
Minimum distance								hard		hard	
Minimum dwell time	hard									hard	soft
Convexity	soft							hard		hard	soft
Compactness	soft							hard			soft
Connectedness	hard				hard	hard	hard				hard
Non-jagged boundaries				hard							
Cost function:			(none)						(complex)		$\sum$ penalty
Workload cost		✓				✓	✓	✓			
Workload imbalance	✓										
Number of sectors				✓	✓	✓	✓				
Number of entry points										✓	
Technology	MIP	LS	adhoc+LS	geo+LS	MIP	MIP	B&B	geo+MIP	geo+EA	CP	CBLS
Test scale	USA	ECAC(11)	USA(W)	USA	ATCC(1)	ATCC(1)	ATCC(5)	ATCC	ATCC	ATCC	ATCC
Test data	extra	hist	extra	(?)	hist	hist	hist	sim	sim	extra	extra

algorithm survey [19], including a comparison with graph-based approaches (such as [20], [21], [22]), approaches based on both graphs and cells (such as [23], [24]), and approaches aiming at a base-line dynamic sectorisation (such as [25], [26]). A complementary survey [3] compares the sectorisation methods of [12], [16], [22], [24], [25], [26] on actual inputs.

A detailed look at Table V reveals that our method (in the right-most column) is more ambitious than the others, either in dimensionality, or in the number and complexity of enforced constraints, or in the size of the explored search space (as we are not starting from elementary sectors).

C. Future Work

More work is needed on our sectorisation method. In particular, as noticed in the experiments, a better grip on the compactness and convexity constraints is needed. Also, since we eventually had to switch off the prior lumping by ASTAAC of cells into AFBs in order to have better control on sector compactness, we have lost the explicit handling of the minimum-distance constraint and now need to add it to our model.

A more recent version of ASTAAC has another sectorisation tool than NEVAC, so our algorithm must now be compared to that one as well.

Further, as mentioned in the introduction, our method is not yet suitable for a dynamic base-line sectorisation, where additional constraints or cost function terms are needed to model proximity to the input sectorisation and the transition cost between consecutive sectorisations.

We argue that the plug-and-play nature of our declarative model and local search algorithm allow us to experiment in a nimble fashion with any additions or alternatives that come up.

ACKNOWLEDGEMENTS

We thank Nicolas Boulin, Marc Dalichampt, and Leïla Zerrouki at the EUROCONTROL Experimental Centre (Brétigny, France) for their help with ASTAAC and NEVAC as well as for their feedback on our progress. Many thanks also to the anonymous referees, for their useful suggestions, and to Pascal Van Hentenryck, for useful conversations.

REFERENCES

[1] C. Allignol, N. Barnier, P. Flener, and J. Pearson, "Constraint programming for air traffic management: A survey," *The Knowledge Engineering Review*, vol. 27, no. 3, pp. 361–392, September 2012.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability*. W.H. Freeman and Company, 1979.

[3] S. Zelinski and C. F. Lai, "Comparing methods for dynamic airspace configuration," in *Proceedings of the 30th Digital Avionics Systems Conference (DASC)*. IEEE, October 2011.

[4] F. Rossi, P. van Beek, and T. Walsh, Eds., *Handbook of Constraint Programming*. Elsevier, 2006.

[5] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations & Applications*. Elsevier / Morgan Kaufmann, 2004.

[6] P. Van Hentenryck and L. Michel, *Constraint-Based Local Search*. The MIT Press, 2005.

[7] F. Glover and M. Laguna, "Tabu search," in *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, 1993, pp. 70–150.

[8] Dynamic Decision Technologies, Inc, "Comet tutorial (version 2.1.1)," 2010, available at http://dynadec.com/.

[9] A. Yousefi and G. L. Donohue, "Temporal and spatial distribution of airspace complexity for air traffic controller workload-based sectorization," in *Proceedings of the 4th AIAA Aviation Technology, Integration and Operations (ATIO) Forum*. American Institute of Aeronautics and Astronautics, 2004, pp. 822–834.

[10] C.-E. Bichot and N. Durand, "A tool to design functional airspace blocks," in *Proceedings of ATM 2007, the 7th USA/Europe R&D Seminar on Air Traffic Management*, S. Saunders-Hodge and V. Duong, Eds., 2007, pp. 169–177, available at http://www.atmseminar.org/seminarContent/seminar7/papers/p_169_DAM.pdf.

[11] R. S. Conker, D. A. Moch-Mooney, W. P. Niedringhaus, and B. T. Simmons, "New process for "clean sheet" airspace design and evaluation," in *Proceedings of ATM 2007, the 7th USA/Europe R&D Seminar on Air Traffic Management*, C. Pusch and S. Saunders-Hodge, Eds., 2007, pp. 91–100, available at http://www.atmseminar.org/seminarContent/seminar7/papers/p_091_DAM.pdf.

[12] C. R. Brinton, K. Leiden, and J. Hinkey, "Airspace sectorization by dynamic density," in *Proceedings of the 9th AIAA Aviation Technology, Integration and Operations (ATIO) Forum*. American Institute of Aeronautics and Astronautics, 2009.

[13] M. C. Drew, "A method of optimally combining sectors," in *Proceedings of the 9th AIAA Aviation Technology, Integration and Operations (ATIO) Forum*. American Institute of Aeronautics and Astronautics, 2009.

[14] S.-L. Tien and R. Hoffman, "Optimizing airspace sectors for varying demand patterns using multi-controller staffing," in *Proceedings of ATM 2009, the 8th USA/Europe R&D Seminar on Air Traffic Management*, S. Saunders-Hodge and V. Duong, Eds., 2009, pp. 128–137, available at http://www.atmseminar.org/seminarContent/seminar8/papers/p_128_DACM.pdf.

[15] D. Gianazza, "Forecasting workload and airspace configuration with neural networks and tree search methods," *Artificial Intelligence*, vol. 174, no. 7-8, pp. 530–549, 2010.

[16] M. Xue, "Three dimensional sector design with optimal number of sectors," in *Proceedings of the AIAA Conference on Guidance, Navigation, and Control (GNC) and Modeling and Simulation Technologies (MST)*. American Institute of Aeronautics and Astronautics, 2010.

[17] S. Kulkarni, R. Ganesan, and L. Sherry, "Static sectorization approach to dynamic airspace configuration using approximate dynamic programming," in *Proceedings of the 11th Integrated Communications, Navigation and Surveillance Conference (ICNS)*. IEEE Computer Society, 2011, pp. J2.1–J2.9.

[18] P. Jägare, "Airspace sectorisation using constraint programming," Master's thesis, Uppsala University, Sweden, Report IT 11 021, Faculty of Science and Technology, 2011, available at http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-155783.

[19] P. Flener and J. Pearson, "Automatic airspace sectorisation: A survey," November 2012, deliverable D2Y4 of a project with EUROCONTROL; available at http://www.it.uu.se/research/group/astra/publications/D2Y4.pdf.

[20] D. Delahaye, M. Schoenauer, and J.-M. Alliot, "Airspace sectoring by evolutionary computation," in *Proceedings of the IEEE International Congress on Evolutionary Computation*. IEEE Computer Society, 1998, pp. 218–223.

[21] H. Tran Dac, P. Baptiste, and V. Duong, "Airspace sectorization with constraints," *RAIRO Operations Research*, vol. 39, no. 2, pp. 105–122, April 2005.

[22] J. Li, T. Wang, M. Savai, and I. Hwang, "A spectral clustering based algorithm for dynamic airspace configuration," in *Proceedings of the 9th AIAA Aviation Technology, Integration and Operations (ATIO) Forum*. American Institute of Aeronautics and Astronautics, 2009.

[23] S. A. Martinez, G. B. Chatterji, D. Sun, and A. M. Bayen, "A weighted-graph approach for dynamic airspace configuration," in *Proceedings of the AIAA Conference on Guidance, Navigation, and Control (GNC)*. American Institute of Aeronautics and Astronautics, 2007.

[24] G. R. Sabhnani, A. Yousefi, and J. S. B. Mitchell, "Flow conforming operational airspace sector design," in *Proceedings of the 10th AIAA Aviation Technology, Integration and Operations (ATIO) Forum*. American Institute of Aeronautics and Astronautics, 2010.

[25] K. Leiden, S. Peters, and S. Quesada, "Flight level-based dynamic airspace configuration," in *Proceedings of the 9th AIAA Aviation Technology, Integration and Operations (ATIO) Forum*. American Institute of Aeronautics and Astronautics, 2009.

[26] M. Bloem and P. Gupta, "Configuring airspace sectors with approximate dynamic programming," in *Proceedings of the 27th International Congress of the Aeronautical Sciences (ICAS)*, 2010.

AUTHOR BIOGRAPHIES

Peter Jägare received his MSc in Computing Science in 2011 from Uppsala University, Sweden. He is a member of the ASTRA research group on constraint programming at Uppsala University.

Pierre Flener is a Professor of Computing Science at the Department of Information Technology of Uppsala University, Sweden. He received his PhD in Computer Science in 1993 from the Université Catholique de Louvain, Belgium. He is the founder and leader of the ASTRA research group on constraint programming at Uppsala University. He is a co-founder of SweConsNet, the Network for Sweden-based researchers and practitioners of Constraint programming.

Justin Pearson is an Associate Professor at the Department of Information Technology at Uppsala University, Sweden. He received his PhD in Electronic Engineering in 1996 from the University of Kent at Canterbury, UK. He is a member of the ASTRA research group on constraint programming at Uppsala University. He is a co-founder of SweConsNet, the Network for Sweden-based researchers and practitioners of Constraint programming.