Airspace Sectorisation Using Constraint-Based Local Search

Patrik Ehrencrona Kjellin
Abstract

Airspace Sectorisation Using Constraint-Based Local Search

Patrik Ehrencrona Kjellin

Airspace sectorisation is the optimisation problem of dividing a given airspace into sectors in a way that promotes efficient air traffic management by some quantifiable measure. In this thesis, we apply constraint-based local search on real world airspace data collected from airports in Europe, given a selection of relevant constraints.
Acknowledgements

I would like to express my gratitude and appreciation to my supervisor Pierre Flener, for providing me with such an interesting problem, and for his continued encouragement and support. Thank you Jean-Noël Monette, for offering invaluable insight and advice on the programming aspect of this project. Thanks Peter Jägare, for taking the time to meet with me and helping me a great deal with ASTAAC and discussing the problem in general.
## Contents

1 Introduction .................................................. 1
   1.1 Airspace Sectorisation .................................... 1
   1.2 Combinatorial Optimisation and Objective ............. 1
   1.3 Objective ................................................ 1
   1.4 Choices ................................................. 1

2 Constraint Programming ..................................... 2
   2.1 Constraint Problems ..................................... 2
   2.2 Local Search ............................................. 3
   2.3 Metaheuristics .......................................... 4
      2.3.1 Tabu Search ....................................... 5
      2.3.2 Simulated Annealing ............................... 5
      2.3.3 Other Metaheuristics ............................. 6

3 Tools ......................................................... 6
   3.1 OscaR.cbls ............................................. 6
   3.2 ASTAAC ................................................ 6

4 Model ......................................................... 6
   4.1 Data and Pre-Processing ................................. 6
   4.2 Sectorisation Constraints ............................... 7
      4.2.1 Background Geometry ............................... 9
      4.2.2 Connectedness .................................... 10
      4.2.3 Balanced Workload ................................. 13
      4.2.4 Minimum Dwell Time: Minimum Stretch Sum ...... 15
      4.2.5 Minimum Distance: No Border Vertices in Stretches 16
      4.2.6 Compactness ...................................... 17
   4.3 Constraint Implementations in OscaR.cbls ............ 19

5 Search ......................................................... 24
   5.1 Algorithm .............................................. 24
   5.2 Parameters ............................................. 26

6 Post-Processing .............................................. 27

7 Experiments .................................................. 27
   7.1 Results ................................................. 27
   7.2 Improving Sector Shapes ............................... 30

8 Discussion .................................................. 36
   8.1 Conclusion .............................................. 36
   8.2 Future Work ............................................ 37
# 1 Introduction

## 1.1 Airspace Sectorisation

An airspace sectorisation is the partitioning of airspace into a given number of control sectors subject to a variety of geometric and workload constraints [4, 3]. In air traffic management (ATM), pairs of controllers are assigned the task of handling all traffic in such a sector. Historically and presently, airspace has been divided primarily according to national borders with little concern for efficient ATM operation. In the ongoing work of creating better sectorisations more suited for efficient operation, a number of different criteria have been proposed, many of which are listed in Section 4.2.

## 1.2 Combinatorial Optimisation and Objective

Optimisation problems in general are ubiquitous and have been subject to much research. In many cases for hard combinatorial problems, finding optimal solutions can be so costly that it is better to use approximation algorithms for finding near optimal solutions. Stochastic local search (SLS, see [5] for instance) is one example of such an algorithm design technique which has been successfully applied on a wide range of problems.

## 1.3 Objective

In this thesis, we implement a constraint-based local search (CBLS, see [16] and [5]) approach to the problem of airspace sectorisation. The differences with the prior thesis [7], which has almost the same objective, are that here we are going to use local search, whereas in [7] systematic search was used, and that [7] is a master thesis whereas this is a bachelor thesis. The differences with the paper [8], which has the same technical objective, are that we benefit from the new insights of [3] and that our non-technical objective is to demonstrate how far a novice in ATM and CBLS can go in only 10 weeks, when using the right tools.

## 1.4 Choices

All the constraint descriptions, as well as the background geometry detailed in Section 4.2 are taken from [3] with full permission granted from the authors, the first of which being the supervisor of this thesis. These descriptions are the basis for all constraints implemented for the purpose of this thesis.

Rather than implementing all search and modelling components from scratch, we use the pre-existing solver OscaR.cbls with a dedicated framework for the implementation of constraints and invariants.

Code for the pre-processing of the data to be used for experiments is taken directly from [8], with permission granted from its authors.

In order to make this thesis self-contained, the initial Section 2 contains a brief overview of constraint programming (CP) in general and CBLS in particular. In Section 3, we present the tools utilised for the making and evaluation of our sectorisation algorithm. We then move on to describing the relevant data and the structure of our model in Section 4, and present the search algorithm and post-processing in Sections 5 and 6. In Section 7,
we present the results of our work. In Section 8 we give our conclusions and elaborate on
the tasks, beyond the scope of this thesis, that lie ahead in airspace sectorisation.

2 Constraint Programming

CP is a programming paradigm wherein a problem is defined as a set of decision variables
with relations imposed on them by declarative constraints. [1, 3] In essence, constraints
define the properties of the sought solution and guide the search toward an optimised
state. There are clear benefits to using declarative constraints kept separate from the
search component of the solver. Namely, constraints are handled in a plug-and-play
manner, which gives rise to high code reusability and modularity.

2.1 Constraint Problems

A constraint satisfaction problem [1] (CSP) in its most basic form consists of:

- A set of decision variables \{x_1, \ldots, x_n\}.
- For each decision variable \(x_i\), a finite domain \(\text{dom}(x_i)\) of its possible value assign-
ments. The domain of a variable need not necessarily be comprised of integers.
- A set of constraints \{c_1, \ldots, c_m\} imposed on decision variables.

A solution is an assignment of the decision variables to values in their domains such that
all constraints are satisfied in systematic search, or violated to a sufficiently low degree
in local search. [3, 5] In CBLS, the sum of the violations of all constraints constitute
the objective function, which is to be minimised. Within the context of local search, a
candidate solution is any assignment of decision variables to values in their domains.

Besides the basic binary constraints (=, <, >, ≤, ≥, ≠), many advanced global
constraints have been invented for the modelling of all kinds of complex problems. For
example, the \text{Linear}(\{x_1, x_2, \ldots, x_3\}, RelOp, c) constraint ensures that the sum of \{x_1,
x_2, \ldots, x_3\} is in relation \(\text{RelOp} \in \{=, <, >, \leq, \geq, \neq\}\) with \(c\). Another example of a global
constraint is \text{AllDifferent}(\{x_1, x_2, \ldots, x_n\}) which penalises non-distinct values in
the \(n\) decision variables. The violations of \text{AllDifferent} as well as \text{Linear} can, for
example, be measured by the number of moves required to fully satisfy them. Many other
global constraints can be found in [2].

Constraint programming consists of two components, modelling and search. [3, 1]
During the modelling stage, choices have to be made on how the problem should be
represented in terms of decision variables, variable domains, and constraints. Creating a
suitable model is imperative for the performance of CP and can often be quite tricky. To
illustrate the importance of good modelling it may be helpful to start off by presenting
a naive CP approach to solving the NQueens problem, which will be used as a running
example throughout this section.
The NQueens problem consists of placing $N$ chess queens on an $N \times N$ chess board in such a way that none of them threaten each other. That is, none of the queens may share the same row, column, or diagonal on the board.

**Example 2.1** Let the board be represented as an $N \times N$ Boolean matrix $B[1..N][1..N]$ of $N^2$ decision variables, where for each position on the board $B[i][j] = 1$ means the square $B_{i,j}$ is occupied by a queen, whereas $B[i][j] = 0$ represents an empty square.

Now it is time to apply some constraints on the decision variables. The Linear constraint, for example, can be applied on each row, column, and diagonal with parameters $RelOp = \leq$ and $c = 1$.

**Example 2.2** Another way to model the same problem is to represent the row position of each queen as an array $Q[1..N]$ of only $N$ decision variables, each with domains $\{1, ..., N\}$. Here, the index represents the column position of the queen, and so the queens are kept separate in the columns by the formulation of the problem. Additionally, the required ALLDIFFERENT constraint imposed on the columns can be handled implicitly (i.e., they are satisfied in the initial assignment and may never be broken), by only applying moves where the values of pairs of queens are swapped. Not only does this result in a significantly smaller amount of decision variables, the problem also requires fewer constraints. Two ALLDIFFERENT constraints applied on the diagonals are now sufficient for solving the problem.

### 2.2 Local Search

The standard approach to searching in constraint programming, systematic search [3, 1], involves pruning the domains of decision variables until either a solution is found, or is decidedly confirmed not to exist. Under systematic search, the entirety of the search space (that is, the set of all variable assignments) is explored. Local search [3, 5], on the other hand, trades the completeness property of systematic search in favour of speed and efficacy. In local search, an initial location of the search space is selected with possibly random assignments to each decision variable. The search then proceeds by iteratively moving to neighbouring solutions that differ from the current solution only in one or a few of its decision variables.

Navigating through the search space by some sort of inference process can be done in several manners. A simplistic search procedure might select the most improving move in each iteration. More often than not, in real applications, other techniques must be applied in order for the search to be able to escape local optima in the underlying objective function, or in order to narrow down the search space to a manageable size. [5]

**Example 2.3**

Continuing with the NQueens model of Example 2.2, one might carry out a search by, in each iteration, identifying the two most violating queens and row-wise swapping them. For a problem of size $N = 4$, and an initial assignment $\{x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4\}$, each queen contributes to the violation of a diagonal ALLDIFFERENT constraint. Recall
that the violation of this constraint is equal to the number of moves necessary to resolve
the conflict, and so the constraint violation becomes 3. The sum of the violations of both
constraints in the system constitutes the objective function, which in this case would
obtain a value of 3.

Considering any pair of queens, after having exchanged values, would cause a decrease in
the objective function, any pair is eligible for a swap move. In the case when several moves
are as good, one is typically randomly selected. If we proceed with a $x_1 := x_2$ move, the
two constraints obtain a violation of 1, causing the objective function to decrease by 1.
Similarly, $x_2 := x_4$ and $x_3 := x_4$ are subsequently selected before the problem is solved.
The figure above, from left to right, shows the state of the board after each consecutive
move, with the leftmost board being the initial assignment.

When comparing a set of possible moves, rather than actually performing each move
and subsequently reversing it in order to measure the change in the objective function, it
is often preferable to use a delta function that efficiently measures the impact of a given
move. OscaR.cbls does not support delta functions, hence we do not discuss them in this
thesis, but we refer to [16] for details.

2.3 Metaheuristics

A fundamental problem associated with local search is its tendency to get stuck within
local minima of the underlying objective function. Local search in its simplest form,
called iterative improvement, starts from some possibly randomly selected point in the
search space and merely moves to a neighbouring candidate solution that improves the
objective function. In this type of primitive search, moving away from local optima is
not possible as any such move will be promptly undone in subsequent iterations. In order
to address this problem, a variety of sophisticated mechanisms exist for escaping local
minima. In many cases, using a hybrid of several such heuristics can additionally improve
the performance of the search. [5]
2.3.1 Tabu Search

Tabu search (TS, see [5]) is a common technique that has been successfully applied on numerous optimisation problems. It extends the iterative improvement strategy by introducing a list of forbidden moves derived from recent search history.

A generic approach to building the tabu list is to mark any decision variable involved in a move as prohibited for the next \( n \) iterations, where the value of \( n \) has to be tuned experimentally. In order to illustrate the benefits of prohibiting certain moves, consider a search space with 5 different states, as depicted to the left. Let the objective function (which is to be minimised) for each state 1..5 evaluate according to: \( \text{Objective}(1) := 3 \), \( \text{Objective}(2) := 1 \), \( \text{Objective}(3) := 4 \), \( \text{Objective}(4) := 0 \), and \( \text{Objective}(5) := 2 \). If the search is initialised at state 1 in the search space, an iterative improvement strategy (selecting either the most improving or least worsening move) will move to state 2 as it is the most improving among the immediate neighbours. In the next step, however, the search will be inclined to move back to state 1, as that is the least worsening move. It is clear that this approach is not going to succeed in finding an optimal solution.

Now let us imagine that upon moving to state 2 in the first iteration, we forbid the search from moving to state 1 in the subsequent \( n = 1 \) iterations. The search is forced to move to the worsening state 3, upon which the globally optimal solution in state 4 is within reach.

The value of \( n \), or the length of the tabu list has a large impact on the performance of TS, as the goal is to escape local minima without moving to a more worsening candidate solution than absolutely necessary. In some instances, it is beneficial to dynamically adjust the tabu length during the search in such a way that it increases as the search approaches a local minimum, and decreases whenever the search enters an escape phase.

TS can also be enriched with various aspiration criteria, which allow certain solutions of some desired quality (or alternatively, for diversification purposes) to bypass the restriction.

2.3.2 Simulated Annealing

The idea behind simulated annealing (SA, see [5]) is to occasionally accept worsening moves, while gradually decreasing their frequency as the search progresses. The acceptance function for SA determines the probability for accepting a worsening move:

\[
f(\Delta) = \frac{1}{1 + e^{\frac{\Delta}{k}}}
\]

where \( \Delta \) is the change in the objective function upon the move. Because \( \Delta \) and \( T \) are positive (since the move is worsening, which is to say the objective function is increasing), the probability for accepting the worsening move is somewhere between 0 and \( \frac{1}{2} \). The temperature variable \( T \) is systematically decreased in each iteration according to some cooling schedule and the annealing parameter \( k \). For example, a schedule resulting in a
fairly rapid decrease in $T$ is $T_n = \frac{T_{n-1}}{k}$ with the initial temperature $T_0$ being set to some appropriate constant [5, 6].

### 2.3.3 Other Metaheuristics

Detailed descriptions of a wide range of sophisticated methods for escaping local minima in local search can be found in [5].

### 3 Tools

#### 3.1 OscaR.cbls

OscaR.cbls [12] (Scala in OR (Operations Research)), formerly Asteroid, is an open-source solver built on top of Scala. It allows us to create complex models and provides a large set of tools for the programming of efficient search techniques. Apart from a fairly sizeable collection of standard invariants and constraints, OscaR.cbls offers a framework for declaring custom, incrementally maintained invariants and constraints. For an in-depth view of the internals of OscaR.cbls, we refer to the OscaR.cbls reference document [9].

#### 3.2 ASTAAC

ASTAAC (Arithmetic Simulation Tool for ATFCM (Air Traffic Flow and Capacity Management) and Advanced Concepts) is a software provided by EUROCONTROL that contains real flight data collected from all over Europe and is able to calculate the cell and trajectory values for a given sectorisation scenario. For our own experiments, ASTAAC allowed us to easily generate an airspace divided into a mesh of cells coupled with real flight data. Furthermore, ASTAAC enabled us to analyse and visualise the results of our sectorisation. In order to interface the sectorisation algorithm with ASTAAC, as well as pre-process data, this work relies completely on the prior work by Peter Jägare in [8].

ASTAAC also ships with a built-in sectorisation algorithm, which uses greedy programming and shall be used as a benchmark for comparisons with our sectorisations in terms of solution quality, performance, etc.

### 4 Model

#### 4.1 Data and Pre-Processing

Airspace sectorisation can be approached either as a graph colouring or as a set covering problem [3]; the constraints described in Section 4.2 were intended for the implementation of the former.

In the graph colouring approach, the background geometry is represented as an undirected graph $G = (V, E \subseteq V \times V)$. For each of the $n$ vertices $v \in V$ representing a cell (or region, here used interchangeably) of the geometry, a decision variable $\text{Colour}(v)$ keeps track of its sector $i \in \{1, 2, \ldots, k\}$, where $k$ is the desired number of sectors. The
edges $E$ connect the adjacent cells of the geometry. The cells can be in the shape of any type of polyhedron with flat sides.

ASTAAC conveniently outputs an air control centre (ACC) airspace in the form of a mesh of cells, each with an associated workload (see below). In the pre-processing step, cells are then greedily divided into an initial solution of $k$ connected components, while attempting to maintain reasonably balanced and somewhat compact sectors. ASTAAC and the pre-processing algorithm of [8] also provide the option of clustering multiple cells into a larger air functional block (AFB), which is guaranteed not to violate the minimum distance constraint.

The trajectories of aircraft are represented as sequences of cells along with timestamps at each cell entry and exit, and will be used in order to define the minimum dwell time, trajectory-based convexity, and minimum distance constraints.

Because there is usually a lot of overlap in the trajectories, the data is somewhat simplified by ASTAAC in the form of combining several trajectories into flows. We did however refrain from using the flows, as doing so would make it difficult to maintain accurate violations in certain constraints on the trajectories.

The workload of a given sector (see [3, 4]) is a measure of the amount of work required from its controllers and can be further categorised as:

1. Monitoring workload, which is the work associated with keeping track of aircraft trajectories within a sector.
2. Conflict workload from managing conflict avoidance and resolution between flights.
3. Coordination workload, which occurs whenever aircraft traverse between adjacent sectors.

The monitoring and conflict workloads are additive in nature and for that reason only they are included in the workload values constructed during pre-processing. We use a function $C$ that takes a set of cells and returns the combined workload for that set considered as a sector. $C$ sums up the workloads of the vertices in a region set $R$ constituting a sector, that is:

$$C(R) = \sum_{v \in R} \text{Workload}(v)$$

In ASTAAC, some parameters are available for defining the characteristics of the data. For the purpose of testing our sectorisation, we used the default settings unless otherwise specified, which divides the airspace into hexagonal cells of dimension 5 NM $\times$ 5 NM (nautical miles) and a height of 1000 feet.

During search, a large portion of the cells can be mostly ignored as a result of having an associated workload of zero (we will refer to these as empty cells). These cells need not be considered for moves as they do not affect the workload or trajectory constraints. They cannot however be completely excluded, as in order to enforce connectivity in the sectors, it is necessary to look at cells that are not part of any trajectory as well.

### 4.2 Sectorisation Constraints

Several constraints have been proposed in previous work, and details for the implementations of constraints were provided for the writing of this thesis in [3]. For simplicity,
we will categorise constraints as either hard or soft. Hard constraints must be satisfied whereas soft constraints may be violated to some degree in a solution. Some constraints may be handled implicitly. Such constraints are satisfied in the initial solution and remain so throughout the search by only allowing non-violating moves (i.e., the way our different row constraints were handled in the NQueens problem). Here are the constraints in brief:

<table>
<thead>
<tr>
<th>Objective</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c_0) Balanced workload</td>
<td>Build sectors in such a way that the workload is evenly distributed among the air traffic controllers. We impose a soft, explicit constraint BALANCED on all the decision variables. Using the workload function (C(R)), we keep track of the total workload of each sector of the current solution. We associate a violation with any cell belonging to a sector that deviates from the average workload across all sectors to a certain extent. In order to search for a solution, we only need to consider those cells (v) where (Workload(v) &gt; 0).</td>
</tr>
<tr>
<td>(c_1) Connectedness</td>
<td>Make sure that each sector consists of a single connected component. This constraint is hard, and must be satisfied in a solution. We opt to handle it implicitly, by only reassigning decision variables to sectors such that all sectors retain the connectedness property.</td>
</tr>
<tr>
<td>(c_2) Trajectory-Based Convexity</td>
<td>Minimise the occurrence of flights entering a sector multiple times. We impose a soft, explicit constraint CONNECTED on each of the flight trajectories, represented as arrays of decision variables. The CONNECTED constraint associates a degree of violation with cells belonging to sector (i) iff sector (i) appears as (n &gt; 1) connected components in the trajectory. For this constraint, the neighbourhood is suitably defined as all cells that lie on the border of a stretch within the trajectories.</td>
</tr>
<tr>
<td>(c_3) Minimum Dwell Time</td>
<td>Minimise the occurrence of flights entering sectors for a very short period of time. We impose a soft, explicit constraint STRETCH-SUM on each of the flight trajectories, represented as arrays of decision variables. The STRETCH-SUM constraint associates a degree of violation with any cell that is part of a stretch that lingers for less than (t) seconds in a sector.</td>
</tr>
<tr>
<td>(c_4) Minimum Distance</td>
<td>Attempt to keep flight trajectories a certain distance from the borders of a sector, in order to avoid controllers of multiple sectors having to keep track of the same flights. We impose a soft, explicit constraint NONBORDER on each of the flight trajectories, represented as arrays of decision variables. A violation is imposed on a cell belonging to sector (i) iff an adjacent cell that is not part of the trajectory is assigned to another sector. This works under the assumption that the cells are large enough that a distance of one cell apart from the border is sufficient.</td>
</tr>
<tr>
<td>(c_5) Compactness</td>
<td>Aim to obtain reasonably compact sectors, that are not overly complicated for air traffic controllers to keep in mind. We impose a soft, explicit constraint COMPACT on all the decision variables. The COMPACT constraint associates a penalty on any cell belonging to a sector that has an above average surface area.</td>
</tr>
</tbody>
</table>
Recall that the following subsections, describing the constraints, as well as the geometric representation used, are taken with permission from [3], with a correction to the CONNECTED constraint given in Section 4.3.

Below, the current assignment of some decision variable $x$ is denoted by $\alpha(x)$.

### 4.2.1 Background Geometry

In order to describe the constraints below, a brief overview of the underlying geometry and notation used is required.

When considering constraints on the shape of sectors, we need access to the shape of its constituent regions. In particular, if we are considering regions that are polytopes, then each region has a number of lower-dimensional facets. In three dimensions, the facets are two-dimensional surfaces, and in two dimensions, the facets are one-dimensional lines. In algebraic topology [11], there is a very general theory of simplexes, which are representations of geometric objects together with facets, and facets of facets, all the way down until zero-dimensional points are reached. Although we do not need the full machinery of algebraic topology, the formalisation given here is inspired by its standard treatment [11].

Given a graph $G = \langle V, E \subseteq V \times V \rangle$, a facet structure for $G$ is a set of facets $F$ and a function $\partial$ from vertices $V$ to sets of facets from $F$. The facet function needs to satisfy the condition that given any two adjacent vertices $v$ and $w$ (that is $\langle v, w \rangle \in E$), there is exactly one facet $f \in F$ such that $f \in \partial(v)$ and $f \in \partial(w)$.

A facet $f$ is a border facet if there is exactly one vertex $v$ such that $f \in \partial(v)$. A border vertex is a vertex that has a border facet.

Given a graph $G = \langle V_G, E_G \subseteq V_G \times V_G \rangle$, we denote the set of border vertices of $G$ by $B_G$. The enveloped graph of $G$, denoted by $G_\perp$, is the graph

$$\langle V_G \cup \{\perp\}, E_G \cup (\{\perp\} \times B_G) \cup (B_G \times \{\perp\}) \rangle$$

That is, there is a unique new vertex $\perp$ that is connected to all border vertices. The extended facet function $\partial_\perp$ is defined as

$$\partial_\perp(v) = \begin{cases} \partial(v) & \text{if } v \neq \perp \\ \{\perp\} & \text{otherwise} \end{cases}$$

Given a graph and a facet structure with facet function $\partial$, the extended facet function satisfies the property that for any $v \neq \perp$ and $w \neq \perp$ there is exactly one facet $f \in F$ such that $f \in \partial_\perp(v)$ and $f \in \partial_\perp(w)$.

Sometimes, we need to see a vertex set $V$ as an ordered set:

- Let $v \prec w$ denote that vertex $v$ is to the left of vertex $w$ in the vertex set $V$.
- Let $v \preceq w$ denote that vertex $v$ is to the left of vertex $w$ in $V$, with possibly $v = w$.
- Let $\text{pred}(v)$ denote the predecessor of vertex $v$ in $V$; if $v$ is the leftmost vertex in $V$, then $\text{pred}(v) = \perp$.
- Similarly, let $\text{succ}(v)$ denote the successor of vertex $v$ in $V$, if any, else $\perp$.  

The vertex set $V$ is seen as a sequence rather than as a set whenever we use the $\prec$ or $\preceq$ relation to specify the semantics of a constraint.

It will sometimes be useful to have information about the volume of a region and the surface area of a facet of a region. We assume that we are given two functions:

- **Volume**: $V \rightarrow \mathbb{N}$, such that $\text{Volume}(r)$ returns the volume of region $r$.
- **Area**: $V \times F \rightarrow \mathbb{N}$, such that $\text{Area}(r,f)$ returns the area of facet $f$ or region $r$, with $f \in \partial(r)$. Technically, the region $r$ is a redundant argument, but we keep it for clarity.

We use the terminology of a 3D space. In a 2D space, the facet area would be a side length, and the region volume would be a surface area.

For the flights, we assume without loss of generality that time stamps are given as integers. A **flight plan** is a sequence of regions, each with an entry time stamp and an exit time stamp, such that the times are increasing between entry and exit time stamps. Formally a flight plan $p$ is a member of $(V \times \mathbb{N} \times \mathbb{N})^*$ such that if $p = [(v_1, t_1, t'_1), (v_2, t_2, t'_2), \ldots, (v_m, t_m, t'_m)]$ then for all $1 \leq i \leq m$ we have $t_i < t'_i$, and for all $1 \leq i < m$ we have $t'_i = t_{i+1}$. Note that we require a strict inequality between entry and exit timestamps since aircraft have a finite velocity. Let the flight plan of flight $f$ be denoted by $\text{Plan}(f)$.

### 4.2.2 Connectedness

The CONNECTED$(G, \text{Colour}, \text{RelOp}, N)$ constraint, with $\text{RelOp} \in \{\leq, <, =, \neq, >, \geq\}$, holds if and only if the number of colours used in the sequence Colour is in relation $\text{RelOp}$ with $N$, and there is a path in the graph $G = \langle V, E \rangle$ between any two vertices of the same colour that only visits vertices of that colour. Formally:

\[
|\{\text{Colour}(v) \mid v \in V\}| \text{RelOp} N \land \\
\forall v, w \in V : \text{Colour}(v) = \text{Colour}(w) \Rightarrow \langle v, w \rangle \in E_{\text{Colour}(v)}^*
\]

(1)

where $E_{\text{Colour}(v)}^*$ denotes the transitive closure of $E_c$, which is the adjacency relation $E$ projected onto adjacency for vertices of colour $c$:

\[
E_c = \{(v, w) \in E \mid \text{Colour}(v) = c = \text{Colour}(w)\}
\]

Hence the total number of connected components in the induced graph ColourGraph must be in relation $\text{RelOp}$ with $N$, and the number of connected components per colour must be at most 1.

**Arbitrary Number of Dimensions.** The CONNECTED$(G, \text{Colour}, \text{RelOp}, N)$ constraint generalises the main aspect of the CONNECT_POINTS$(w, h, d, \text{Colour}, N)$ constraint of [2], which considers $\text{RelOp}$ is “=” and considers $G$ to be induced by a $w \times h \times d$ cuboid divided into same-sized regions; further, there is a special colour (value 0) for which there is no restriction on the number of connected components.

The constraint CONNECTED is a hard constraint in prior work on airspace sectorisation using stochastic local search [8], hence no violation and differentiation functions are given there.
One Dimension. The constraint \texttt{CONNECTED}(G, Colour, RelOp, N) for a graph G induced by a one-dimensional geometry (in which connected components are called stretches) generalises the main aspect of the \texttt{MULTIGLOBAL\_CONTIGUITY}(Colour) constraint \cite{2}, which itself generalises the \texttt{GLOBAL\_CONTIGUITY}(Colour) constraint of \cite{10}: the latter constrains only one colour (value 1) but the former constrains several colours, and both lack the decision variable \texttt{N} and hence \texttt{RelOp}; further, both have a special colour (value 0) for which there is no restriction on the number of stretches (there are at most two such stretches when there is only one constrained colour).

The trajectory-based convexity of an airspace sectorisation is achieved by posting for every flight a \texttt{CONNECTED}(G, Colour, \leq, s) constraint on the sequence Colour of decision variables denoting the sequence of colours of its one-dimensional visited region sequence \( V \), where \( s \) is the imposed or maximum number of sectors. The initial domain of each colour decision variable \texttt{Colour}(v) is \{1, 2, \ldots, s\}.

Such a trajectory-based convexity constraint is a soft constraint in prior work on airspace sectorisation under stochastic local search \cite{8}, but lacks the decision variable \texttt{N} and hence \texttt{RelOp}; further, the constraint violation is defined differently there (in a manner that requires an asymptotically higher runtime to compute than the one we give below), and the variable violation and differentiation functions are not given there (though they are in the unpublished code underlying the experiments).

Violation Functions. The violation functions described below have no asymptotically better specialisation to the case of \( G \) being induced by a one-dimensional space. Hence they apply to both connectedness in a space of an arbitrary number of dimensions and to contiguity in a one-dimensional space.

If the \texttt{CONNECTED} constraint is considered explicitly, then we proceed as follows. For representing the induced graph ColourGraph, we show that it suffices to initialise and maintain the following two data structures, which are internal to the constraint:

- Let \( NCC(c) \) denote the number of connected components (CCs) of ColourGraph whose vertices currently have colour \( c \).
- Let \( NCC \) denote the current number of connected components of ColourGraph:

\[
NCC = \sum_{c \in \text{Colours}} NCC(c)
\]  

We can now re-formalise the semantics (1): the \texttt{CONNECTED}(G, Colour, RelOp, N) constraint is satisfied if and only if

\[
NCC \text{ RelOp } N \land \forall c \in \text{Colours} : NCC(c) \leq 1
\]

The violation of a colour decision variable, say Colour\( (v) \) for vertex \( v \), is the current excess number, if any, of connected components of ColourGraph for the colour of \( v \):

\[
\text{violation}(\text{Colour}(v)) = NCC(\alpha(\text{Colour}(v))) - 1
\]  

This variable violation is zero if \( v \) currently has a colour for which there is exactly one connected component in ColourGraph, and positive otherwise.
The violation of the counter decision variable $N$ is 0 or 1 depending on whether $NCC$ is in relation $RelOp$ with the current value of $N$:

$$\text{violation}(N) = 1 - [NCC ~ RelOp ~ \alpha(N)]$$

This variable violation is zero if $NCC ~ RelOp ~ \alpha(N)$ currently holds, and one otherwise.

The violation of the constraint is the sum of the variable violation of $N$ and the current excess number, if any, of connected components for all colours:

$$\text{violation} = \text{violation}(N) + \sum_{c \in \text{Colours}} \max(NCC(c) - 1, 0) \quad (4)$$

The constraint violation is zero if $NCC ~ RelOp ~ \alpha(N)$ holds and there currently is at most one connected component in $ColourGraph$ for each colour.

To achieve incrementality, once a move has been picked and made, the internal data structures and the variable and constraint violations must be updated. The following updating code (which contains an error that we will fix in in Section 4.3) for a colour assignment move $Colour(v) := c$ applies, where the new assignment $\alpha'$ is the old assignment $\alpha$, except that $\alpha'(Colour(v)) = c$:

1: if $\forall w \in \text{Adj}(v) : \alpha(\text{Colour}(w)) \neq c$ then \{ $v$ forms a new CC of colour $c$ \}
2: \quad $\text{NCC}(c) := \text{NCC}(c) + 1$
3: \quad $\text{NCC} := \text{NCC} + 1$
4: \quad $\text{violation} := \text{violation} + 1$
5: \textbf{end if}
6: if $\forall w \in \text{Adj}(v) : \alpha(\text{Colour}(w)) \neq \alpha(\text{Colour}(v))$ then \{ $v$ formed a CC of $\alpha(\text{Colour}(v))$ \}
7: \quad $\text{NCC}(\alpha(\text{Colour}(v))) := \text{NCC}(\alpha(\text{Colour}(v))) - 1$
8: \quad $\text{NCC} := \text{NCC} - 1$
9: \quad $\text{violation} := \text{violation} - 1$
10: \textbf{end if}
11: \textbf{for all } v \in V \textbf{ do}
12: \quad $\text{violation}(\text{Colour}(v)) := \alpha'(\text{NCC}(\text{Colour}(v))) - 1$
13: \textbf{end for}
14: $\text{violation}(N) := 1 - [NCC ~ RelOp ~ \alpha(N)]$

Incremental updating for a colour assignment move takes time linear in the degree of vertex $v$ in $G$ and linear in the number $|V|$ of vertices (and colour decision variables). Code follows similarly for the colour swap and counter assignment moves.

For the remaining constraints, we assume that some relationships among internal data structures, such as (2), and the violation functions, such as (3) to (4), are defined as invariants [16], so that the solver automatically updates these quantities incrementally, without the constraint designer having to write explicit code, such as lines 3, 4, 8, 9 and 11 to 14.

**Hard Constraint.** If the CONNECTED constraint is considered implicitly, as in [8], then it can be satisfied cheaply in the start assignment, by partitioning $G$ into connected components and setting $N$ according to $RelOp$, and maintained as satisfied upon every move, by only considering moves that re-colour a vertex at the border of a connected
component to the colour of an adjacent connected component. For instance, if RelOp is equality, then one can partition \( G \) into \( n = \max(\text{dom}(N)) \) connected components and set \( N := n \).

### 4.2.3 Balanced Workload

The Balanced \((G, Colour, Value, \mu, \Delta)\) constraint holds if and only if the sums of the given integer values under Value of the vertices having the same colour under Colour are balanced, in the sense of having the (possibly unknown, and not necessarily integer) value \( \mu \) as average and having discrepancies to \( \mu \) that do not exceed the (possibly unknown) integer threshold \( \Delta \). If \( \Delta \) is not given, then it may appear in the objective function, towards being minimised. Formally, the constraint can be decomposed into the following conjunction:

\[
\forall i \in Colours : X[i] = \sum_{v \in V} ([Colour(v) = i] \cdot Value(v)) \land \Gamma(X, \mu, \Delta)
\]

where constraint \( \Gamma \) is either Spread or Deviation, thereby giving a concrete definition to the used abstract concept of discrepancy:

- The Spread\((X, \Delta, \mu)\) constraint [13] holds if and only if the \( n \) integer variables \( X[i] \) have the (possibly unknown, and not necessarily integer) value \( \mu \) as average and the sum of the squared differences \((n \cdot X[i] - n \cdot \mu)^2\) does not exceed the (possibly unknown) integer threshold \( \Delta \).

- The Deviation\((X, \Delta, \mu)\) constraint [14] holds if and only if the \( n \) integer variables \( X[i] \) have the (possibly unknown, and not necessarily integer) value \( \mu \) as average and the sum of the deviations \(|n \cdot X[i] - n \cdot \mu|\) does not exceed the (possibly unknown) integer threshold \( \Delta \).

The multiplications by \( n \) in the definitions of discrepancy lift all reasoning to integer domains even when \( \mu \) is not an integer, as \( \sum_{i=1}^{n} X[i] = n \cdot \mu \) and the \( X[i] \) are integer variables: the integer threshold \( \Delta \) has to be calibrated accordingly. One could also use the Range\((X, RelOp, \Delta)\) constraint [2], which holds if and only if \( \max(X) + 1 - \min(X) \ RelOp \Delta \), but we do not pursue this option further.

In airspace sectorisation, the workload of each sector must be within some given imbalance factor of the average across all sectors. Hence one would take Value as the Workload function of Section 4.1. Recall that we only consider additive workloads, such as monitoring workload and conflict workload, but no non-additive workloads, such as coordination workload. Note that \( \mu \) is known when the number \( n \) of sectors (and thus colours) is imposed:

\[
\mu = \frac{\sum_{v \in V} \text{Workload}(v)}{n} \tag{5}
\]

The workload balancing constraint is a soft constraint in prior work on airspace sectorisation under stochastic local search [8], but its concept of discrepancy is defined in terms of a ratio rather than a difference with the average \( \mu \), namely \( X[i]/\mu \leq 1 + \Delta \) for each \( i \) (in the experiments, \( \Delta = 0.05 \) was used). This concept of discrepancy is less related to standard concepts in statistics.
Violation Functions. We here handle the BALANCED constraint for the case \( \Gamma = \text{DEVIATION} \). Handling the case \( \Gamma = \text{SPREAD} \) can be done using the same ideas. We assume \( \mathit{Colours} = \{1, 2, \ldots, n\} \).

If the \( \text{BALANCED}(G, \mathit{Colour}, \mathit{Value}, \mu, \Delta) \) constraint is considered explicitly, then we proceed as follows. For simplicity of notation, we assume \( \Delta \) and \( \mu \) are given, and that \( \mu \) is an integer. Relaxing these assumptions can be done using the same ideas as below. The multiplications by the number \( n \) of colours in the definition of the underlying DEVIATION constraint can then be eliminated, upon dividing \( \Delta \) by \( n^2 \), giving the following simplified semantics of \( \text{BALANCED} \):

\[
\sum_{i=1}^{n} X[i] = n \cdot \mu \quad (6)
\]

and

\[
\sum_{i=1}^{n} |X[i] - \mu| \leq \Delta \quad (7)
\]

where

\[
\forall i \in \{1, \ldots, n\} : X[i] = \sum_{v \in V} ([\mathit{Colour}(v) = i] \cdot \mathit{Value}(v)) \quad (8)
\]

Note that (8) implies

\[
\sum_{i=1}^{n} X[i] = \sum_{v \in V} \mathit{Value}(v)
\]

so that, using (6), we must have

\[
\mu = \frac{\sum_{v \in V} \mathit{Value}(v)}{n} \quad (9)
\]

similarly to (5). From now on, we assume the given \( \mathit{Value} \) and \( \mu \) satisfy (9), so that we need not reason about (6), as it is then surely satisfied, because implied when (8) is satisfied. Formula (8) itself defines the auxiliary decision variables \( X[i] \), so it suffices to set it up as a set of invariants [16]. In conclusion, we only need to deal with formula (7).

The violation of a decision variable, say \( \mathit{Colour}(v) \) for vertex \( v \), is the same as the violation of the auxiliary decision variable \( X[\alpha(\mathit{Colour}(v))] \), which is functionally dependent on \( \mathit{Colour}(v) \) under (8). This violation is the current deviation from \( \mu \) of the value sum for the current colour of \( v \):

\[
\text{violation}(\mathit{Colour}(v)) = \text{violation}(X[\alpha(\mathit{Colour}(v))]) = |\alpha(X[\alpha(\mathit{Colour}(v))] - \mu|
\]

The variable violation is zero if \( v \) currently has a colour \( i \) whose value sum \( X[i] \) is equal to \( \mu \), and thus contributes nothing to the total deviation for all colours.

The violation of the constraint is the excess, if any, over \( \Delta \) of the sum of the current deviations from \( \mu \) of the value sums for all colours. We need not initialise and maintain any internal data structure for this purpose, as each auxiliary decision variable \( X[i] \) contains the current value sum for colour \( i \), and its variable violation is the current deviation from \( \mu \) of the value sum for colour \( i \). Hence the constraint violation is defined as follows:

\[
\text{violation} = \max \left( \sum_{i=1}^{n} \text{violation}(X[i]) - \Delta, 0 \right)
\]
The constraint violation is zero if the total deviation for all colours currently does not exceed $\Delta$.

We omit the rather clerical code for achieving incrementality.

4.2.4 Minimum Dwell Time: Minimum Stretch Sum

Consider a graph $G = \langle V, E \rangle$ induced by a one-dimensional space, so that the two sequences Colour and Value are indexed by a vertex sequence $V$ rather than vertex set. The $\text{STRETCHSUM}(G, \text{Colour}, \text{Value}, \text{RelOp}, t)$ constraint, with $\text{RelOp} \in \{\leq, <, =, \neq, >, \geq\}$, holds if and only if every stretch of the sequence Colour corresponds to a subsequence of Value whose sum is in relation $\text{RelOp}$ with threshold $t$, whose value is given. Formally:

$$\forall \ell \preceq r \in V : \text{Stretch}(\text{Colour}, \ell, r) \Rightarrow \left( \sum_{\ell \preceq v \preceq r} \text{Value}(v) \right) \text{RelOp} t$$

Note that there is no limit on the number of stretches per colour.

In airspace sectorisation, every flight entering a sector must stay within it for a given minimum amount of time (say $t = 120$ seconds), so that the coordination work pays off and that conflict management is possible. This minimum dwell-time constraint is achieved by posting for every flight $f$ a $\text{STRETCHSUM}(G, \text{Colour}, \text{Value}, \geq, 120)$ constraint on the sequence Colour of decision variables denoting the sequence of colours of its visited region sequence $V$, with Value storing the durations of the flight $f$ in each region:

$$\text{Value} = [t'_i - t_i \mid \langle \ell_i, t_i, t'_i \rangle \in \text{Plan}(f)]$$

The $\text{STRETCHSUM}$ constraint is a soft constraint in the prior work on airspace sectorisation under stochastic local search of [8], but the constraint violation is defined differently there (in a manner that requires an asymptotically higher runtime to compute than the one we give below), and the variable violation and differentiation functions are not given there (though they are in the unpublished code underlying the experiments).

Violation Functions. If the $\text{STRETCHSUM}$ constraint is considered explicitly, then we proceed as follows. For simplicity of notation, we assume that $\text{RelOp}$ is $\geq$. The other values of $\text{RelOp}$ are handled analogously. We initialise and incrementally maintain the following data structure, which is internal to the constraint:

- Let $\text{Stretch}(v)$ denote the tuple $\langle \ell, r, c, \sigma \rangle$, meaning that vertex $v \in V$ is currently in a colour stretch, from vertex $\ell$ to vertex $r$, whose colour is $c$ and value sum is $\sigma$:

$$\sigma = \sum_{\ell \preceq i \preceq r} \text{Value}(i)$$

We say that a colour stretch with value sum $\sigma$ is a violating stretch if $\sigma$ is smaller than the threshold $t$:

$$\sigma \not\geq t$$
The violation of a decision variable, say \( \text{Colour}(v) \) for vertex \( v \) with \( \text{Stretch}(v) = (\ell, r, c, \sigma) \), where \( \alpha(\text{Colour}(v)) = c \), is defined as follows:

\[
\text{violation(\text{Colour}(v))} = \begin{cases} 
0 & \text{if } v \notin \{\ell, r\} \\
0 & \text{if } v \in \{\ell, r\} \land \sigma \geq t \land \sigma - \text{Value}(v) \not\geq t \\
\text{Value}(v) & \text{if } v \in \{\ell, r\} \land \sigma \geq t \land \sigma - \text{Value}(v) \geq t \\
1 & \text{if } v \in \{\ell, r\} \land \sigma \not\geq t
\end{cases}
\]

The variable violation is zero if \( v \) is currently either not at the border (leftmost or rightmost element) of its colour stretch (so that flipping its colour would break its current stretch into three stretches) or at the border of a non-violating colour stretch that would become violating upon losing \( v \). The variable violation is positive if \( v \) is currently at the border of a colour stretch that is either non-violating and would remain so upon losing \( v \) (so that it can contribute \( \text{Value}(v) \) to the value sum of the adjacent colour stretch, if any) or violating (so that there is an incentive to drop \( v \) and eventually eliminate this stretch).

The violation of the constraint is the current number of violating colour stretches:

\[
\text{violation} = \sum_{(\_, \_, \_, \sigma) \in \text{Stretch}} [\sigma \not\geq t]
\]

The constraint violation is zero if there currently is no violating colour stretch.

It is advisable to use a neighbourhood where vertices at the border of a stretch are re-coloured using a currently unused colour or the colour of an adjacent connected component.

We omit the rather clerical code for achieving incrementality.

4.2.5 Minimum Distance: No Border Vertices in Stretches

Let \( P \) be the sequence of vertices of a simple path in graph \( G = (V, E) \), plus the special vertex \( \perp \) at the beginning and at the end. The NONBORDER\((G, \text{Colour}, P)\) constraint holds if and only if all vertices of all stretches of the projection, denoted by \( \text{Colour}(P) \), of \( \text{Colour} \) onto the vertices of \( P \) (and in the vertex ordering of \( P \)) only have adjacent vertices outside \( P \) of the same colour:

\[
\forall \ell \leq r \in P \setminus [\perp] : \text{Stretch(\text{Colour}(P), \ell, r)} \Rightarrow \forall \ell \leq v \leq r \in P : \forall w \in \text{Adj}(v) \setminus P : \text{Colour}(w) = \text{Colour}(v)
\]

This constraint is trivially satisfied when the graph \( G \) is induced by a one-dimensional geometry, as every vertex in \( P \) then has no adjacent vertices outside \( P \). Note that there is no limit on the number of stretches per colour.

In airspace sectorisation, each existing trajectory must be inside each sector by a minimum distance (say ten nautical miles), so that conflict management is entirely local to sectors. If the airspace is originally divided into same-sized regions whose diameter is (at least) that minimum distance, then the border regions of each sector can only serve as sector entry and exit regions for all flights. Hence one could use a NONBORDER constraint for each flight \( f \), by setting \( P \) to its sequence of visited regions:

\[
P = [\perp] \cup [v_i | (v_i, \_, \_) \in \text{Plan}(f)] \cup [\perp]
\]
**Violation Functions.** If the NONBORDER constraint is considered explicitly, then we proceed as follows, without needing any internal datastructures.

The *violation of a decision variable*, say $Colour(v)$ for vertex $v$, is defined as follows:

$$
\text{violation}(Colour(v)) = \begin{cases} 
0 & \text{if } v \notin P \\
\sum_{w \in \text{Adj}(v) \setminus P} [\alpha(\text{Colour}(w)) \neq \alpha(\text{Colour}(v))] & \text{if } v \in P
\end{cases}
$$

The variable violation is zero if $v$ is not in $P$ or has no adjacent vertices outside $P$ that currently have a different colour.

The *violation of the constraint* is the current sum of the violations of its variables:

$$
\text{violation} = \sum_{v \in P} \text{violation}(Colour(v))
$$

The constraint violation is zero if the constraint is satisfied.

It is advisable to use a neighbourhood where vertices at the border of a stretch of $Colour(P)$ are re-coloured using a currently unused colour or the colour of an adjacent vertex outside $P$.

We omit the rather clerical code for achieving incrementality.

### 4.2.6 Compactness

Consider a graph $G = \langle V, E \rangle$ induced by a space of at least two dimensions. The $\text{COMPACT}(G, Colour, t)$ constraint holds if and only if the sum of the sphericity discrepancies of the connected components of the graph $Colour\text{Graph}$ induced by $G$ and the sequence $Colour$ is at most the threshold $t$, whose value is given.

The sphericity discrepancy of a connected component is defined as follows. Recall that in $G$ each facet is endowed with a surface area, and each vertex is endowed with a volume. We define the following concepts:

- In Section 4.2.1, we formalised the notion of border vertices, that is vertices at the edge of the geometry. Here we are dealing with colouring, so we need to formalise the border of coloured regions. A facet $f$ of a vertex $v$ is a *border facet under a sequence Colour* if $v$ has no adjacent vertex for $f$ or $v$ has a different colour than the adjacent vertex that shares $f$. Formally, a facet $f \in \partial(v)$ of a vertex $v$ is a border facet if and only if the following statement
  $$
  Colour(w) \neq Colour(v)
  $$
  holds for the vertex $w$ that shares $f$ with $v$.

To simplify the formalisation, we assume that we are working with background geometry of the form $G_\bot$ for some given background geometry $G$. That is, there is a unique special vertex in $V$, called $\bot$, that shares a facet with every vertex where there otherwise is no adjacent vertex for that facet. Now a vertex has exactly one adjacent vertex for each of its facets.
• The border surface area of a set $W$ of vertices that have the same colour under a sequence $\text{Colour}$ (such as a connected component of the induced graph $\text{ColourGraph}$), denoted by $A_W$, is the sum of the surface areas of the border facets of the vertices in $W$:

$$A_W = \sum_{v \in W} \sum_{w \in \text{Adj}(v)} \sum_{\substack{f \in \partial(v) \cap \partial(w) \atop w \notin W}} \left[ \text{Colour}(w) \neq \text{Colour}(v) \right] \cdot \text{Area}(v, f)$$

(10)

Recall that $|\partial(v) \cap \partial(w)| = 1$ when vertices $v$ and $w$ are adjacent.

• The sphere surface area of a set $W$ of vertices that have the same colour under a sequence $\text{Colour}$, denoted by $S_W$, is the surface area of a sphere that has as volume the total volume $V_W$ of the vertices in $W$ (see [17] for the derivation of this formula):

$$S_W = \pi^{1/3} \cdot (6 \cdot V_W)^{2/3}$$

where

$$V_W = \sum_{w \in W} \text{Volume}(w)$$

In case $G$ is induced by a 3D cuboid divided into same-sized regions, we can rather define the sphere surface area as the surface area of the smallest collection of regions that contains the sphere that has as volume the total volume of the vertices in $W$. We omit the mathematical details, as they are specific to the shape of the regions: a space can be tiled by any kind of polyhedra, such as cubes or beehive cells.

• The sphericity discrepancy of a set $W$ of vertices that have the same colour under a sequence $\text{Colour}$, denoted by $\Psi_W$, is the difference between the border surface area of $W$ under $\text{Colour}$ and the sphere surface area of $W$ under $\text{Colour}$:

$$\delta \Psi_W = A_W - S_W$$

This concept was derived from the sphericity $\Psi$ of a shape $p$, defined in [17] to be the ratio between the surface area $S_p$ of a sphere that has the same volume as $p$ and the surface area $A_p$ of $p$; note that $\Psi = 1$ if $p$ is a sphere, and $0 < \Psi < 1$ otherwise, assuming $p$ is not empty. Our concept would be defined as the subtraction $A_p - S_p$ rather than as the ratio $S_p/A_p$, as we need (for stochastic local search) a non-negative metric that is 0 in the good case, namely when $p$ is (the smallest over-approximation of) a sphere.

Note that the COMPACT constraint imposes no limit on the number of connected components of $\text{ColourGraph}$ per colour: if there are several connected components for a colour, then the total sphericity discrepancy may be unnecessarily large.

Such a compactness constraint is a soft constraint in prior work on airspace sectorisation under stochastic local search [8], but lacks the threshold $t$ there; we generalise the ideas of its violation functions using the concept of sphericity discrepancy, and we describe them in much more detail.
Violation Functions. If the COMPACT constraint is considered explicitly, then we proceed as follows. We initialise and incrementally maintain the following data structures, which are internal to the constraint:

- Let \( \text{Border}(v) \) denote the border surface area of the vertex set \( \{v\} \). If every vertex only has facets of unit surface area (for instance, when \( G \) is induced by a space divided into same-sized cubes or squares), then \( \text{Border}(v) \) is defined as follows:

\[
\text{Border}(v) = \sum_{w \in \text{Adj}(v)} [\alpha(\text{Colour}(w)) \neq \alpha(\text{Colour}(v))]
\]

Otherwise, the formula needs to be generalised as follows, using (10):

\[
\text{Border}(v) = \sum_{w \in \text{Adj}(v)} \sum_{f \in \partial(v) \cap \partial(w)} [\alpha(\text{Colour}(w)) \neq \alpha(\text{Colour}(v))] \cdot \text{Area}(v, f)
\]

- Let \( \text{CCs} \) denote the current set of connected components of the induced graph \( \text{ColourGraph} \), each encoded by a tuple \( \langle \sigma, \nu \rangle \), meaning that it currently has total surface area \( \sigma \) and total volume \( \nu \).

The violation of a decision variable, say \( \text{Colour}(v) \) for vertex \( v \), is its current weighted border surface area:

\[
\text{violation}(\text{Colour}(v)) = f(\text{Border}(v))
\]

where the weight function \( f \) can be the identity function, but can also suitably penalise larger border surface areas, provided \( f(0) = 0 \); in [8], we found that using \( f \) as \( \lambda x : x^2 \) works well enough. The variable violation is zero if \( v \) is not a border facet.

The violation of the constraint is the current excess, if any, of the sum of the sphericity discrepancies of the connected components:

\[
\text{violation} = \max\left( \sum_{\langle \sigma, \nu \rangle \in \text{CCs}} (\sigma - \pi^{1/3} \cdot (6 \cdot \nu)^{2/3}) - t, 0 \right)
\]

The constraint violation is zero if the total sphericity discrepancy does not exceed \( t \).

It is advisable to use a neighbourhood where vertices at the border of a connected component are re-coloured using a currently unused colour or the colour of an adjacent connected component.

We omit the rather clerical code for achieving incrementality.

4.3 Constraint Implementations in OscaR.cbls

All constraints described in Section 4.2 were written in a problem-independent fashion, and were the basis for all the constraints in our model. Below we elaborate on how the constraints were implemented, and point out any modifications or mere observations that were made along the way.
$c_0$: Balanced Workload. For the BALANCED constraint, the clustering invariant part of the OscaR.cbls library was used in order to maintain $k$ sets of Colour decision variables according to sector $i \in \{1, \ldots, k\}$. Another invariant was applied on the Colour sets of each sector as well as the workload values, in order to incrementally obtain the sum of the workloads $X[i]$ for each sector $i$. The violation of the constraint is then the sum of the violations of $k$ distinct LE ($\leq$) constraints posted on the $X[i]$ variables:

$$\text{violation}(X[i]) = \text{LE}(X[i], \mu \cdot (1 + \Delta)).\text{violation}$$

$$\text{violation} = \sum_{i=1}^{k} \text{violation}(X[i])$$

where the violation of a LE($a, b$) constraint is $\max(0, a - b)$ and $\gamma$.violation denotes the violation of constraint $\gamma$. Note that this version of the BALANCED constraint differs somewhat from the constraint described in Section 4.2.3. My constraint does in fact only impose an upper bound on the workload, whereas the BALANCED constraint in Section 4.2.3 imposes an upper as well as a lower bound. This was due to a mistake of mine that was unfortunately discovered toward the very end of this project and would have been unduly expensive to fix for all experiments in Section 7. While our implementation was successful in balancing the sectors in all our experiments, the lack of a lower bound could potentially result in poor workload balance. For instance, in a scenario where $k - 1$ sectors are near the upper bound, the remaining sector could acquire a very low workload value without giving cause to any violation in the constraint.

When this mistake was discovered, after the experiments in Section 7.1 were conducted, but before the experiments in Section 7.2, a GE ($\geq$) constraint was added on each of the sectors, in order to enforce a lower bound on the workloads as well. The constraint violation of the BALANCED constraint was then given by:

$$\text{violation}(X[i]) = \text{LE}(X[i], \mu \cdot (1 + \Delta)).\text{violation} + \text{GE}(X[i], \mu \cdot (1 - \Delta)).\text{violation}$$

$$\text{violation} = \sum_{i=1}^{k} \text{violation}(X[i])$$

where the violation of a GE($a, b$) constraint is $\max(0, b - a)$.

$c_1$: Connectedness. For the 3-dimensional connected constraint applied on $G$, we have opted for implicitly handling the constraint by exclusively considering non-violating moves during the search. Naturally, this approach requires that the connected constraint is fully satisfied in the initial solution after pre-processing. A move $\text{Colour}(v) := c$ is allowed if the following condition holds:

*For all vertices $w \in \{\text{Adj}(v) \mid \text{Colour}(w) = \alpha(\text{Colour}(v))\}$ there exists a set of edges $\{(w_1, v_1), (v_1, v_2), \ldots, (v_n, w_j)\} \subseteq E$ such that for all vertices $v_i \in \{v_1, \ldots, v_n\}$ we have $\text{Colour}(v_i) = \alpha(\text{Colour}(v))$. That is, there must exist a path between all cells of the same sector that does not visit a cell belonging to another sector.*
Ensuring that such is the case for every probed move would become unduly expensive. We define the property of local connectedness. For a sector $i$ to remain locally connected upon a move $\text{Colour}(v) := c$, where $\alpha(\text{Colour}(v)) := i$, we only concern ourselves with the subgraph that results from vertices that either belong to $\text{Adj}(v)$, or have two neighbours in $\text{Adj}(v)$. If a path exists between every such vertex $w$ where $\alpha(\text{Colour}(w)) := i$, we have local connectedness.

During the search, checking whether the above condition is met upon a move $\text{Colour}(v) := c$ is done by the means of performing a breadth-first search (BFS, see [15]) on the neighbours and secondary neighbours (that is, any vertices neighbouring at least two vertices adjacent to $v$). We disallow the search from moving to vertices of a different colour than $\alpha(\text{Colour}(v))$. If all the neighbours of $v$ belonging to sector $\alpha(\text{Colour}(v))$ are visited during the search, the graph is locally connected and we can safely proceed with the move.

The time complexity of a BFS on some graph $G = \langle V, E \rangle$ can be expressed as $\Theta(|V| + |E|)$, although in this case we limit ourself to searching locally. [15] For confirming the local connectedness of the neighbours of some vertex $v$, let $V'$ be composed of every vertex in $\text{Adj}(v)$, as well as any vertex that has at least two neighbours in $\text{Adj}(v)$. Let $E'$ be the subset of edges in $E$ that connect two vertices in $V'$. The time complexity is asymptotically bounded by:

$$\Theta(|V'| + |E'|)$$

$c_2$: Trajectory-Based Convexity. The soft, one-dimensional constraint described in Section 4.2.2 (and taken from [3]) is used in order to enforce trajectory-based convexity on each flight path. Because OscaR.cbls does not use differentiation functions, only the updating code was of interest for this thesis. There was however a flaw in the updating code that needed to be corrected. Namely, the NCC variables were incorrectly updated in the case when a move split one connected component into three, or vice versa. The following updating code correctly maintains the NCC variables in the one-dimensional case. Note that in this code, the NCC variable and the violations are assumed to be maintained as invariants based on $\text{NCC}(c)$.

```
1: if $\forall w \in \text{Adj}(v) : \alpha(\text{Colour}(w)) \neq c$ then \{v forms a new CC of colour c\}
2: \text{NCC}(c) := \text{NCC}(c) + 1
3: if $|\text{Adj}(v)| = 2$ and $\forall w \in \text{Adj}(v) : \alpha(\text{Colour}(w)) = \alpha(\text{Colour}(v))$ then \{Old CC is split into three\}
4: \text{NCC}(\alpha(\text{Colour}(v))) := \text{NCC}(\alpha(\text{Colour}(v))) + 1
5: end if
6: end if
7: if $\forall w \in \text{Adj}(v) : \alpha(\text{Colour}(w)) \neq \alpha(\text{Colour}(v))$ then \{v formed a CC of $\alpha(\text{Colour}(v))$\}

8: \text{NCC}(\alpha(\text{Colour}(v))) := \text{NCC}(\alpha(\text{Colour}(v))) - 1
9: if $|\text{Adj}(v)| = 2$ and $\forall w \in \text{Adj}(v) : \text{Colour}(w) = c$ then \{three CCs are reduced to one\}
10: \text{NCC}(c) := \text{NCC}(c) - 1
11: end if
12: end if
```
Apart from the additions at lines 3, 4, 9, and 10, our constraint is the same as that of Section 4.2.2. The NCC variable is maintained as an invariant \( \sum_{i=1}^{k} NCC(i) \).

**Border Cell Invariant.** For the convexity constraints to be effective it is necessary to only consider cells on the border of at least one CC for potential moves during the search. Additionally, cells that are part of at least one trajectory and lie on the boundary between two sectors make a suitable neighbourhood for the problem as a whole, considering these cells have an associated non-zero workload and moving them often will not break the (implicit) connectedness constraint.

In order to incrementally maintain a set of boundary cells, we used a custom invariant \( \text{BorderCells}(G,P) \) where \( P \) is the set of all trajectories. For each cell \( v \in \bigcup_{t \in P} t \) that is part of at least one trajectory \( t \), the set \( \text{Adj}_v \) is initialised containing cells that precede or succeed \( v \) in the trajectories.

Then, on the update of some cell \( v \), \( v \) and all members of \( \text{Adj}_v \) can be inserted or removed from the set of boundary cells by checking whether they have any adjacent cells of a different colour. The time complexity of the updating operation is asymptotically bounded by:

\[
\Theta(|\text{Adj}_v| + \sum_{w \in \text{Adj}_v} |\text{Adj}_w|)
\]

In the worst case this can be quite expensive, and there are likely more efficient ways of updating the invariant.

**c₃: Minimum Dwell Time.** The \( \text{Stretch}(v) \) data structure is initialised as an array of length \( |\text{Colour}| \), where \( \text{Colour} \) is a sequence of cells representing some trajectory. The \( \text{Stretch} \) 4-tuple \( \langle \ell, r, c, \sigma \rangle \) to which each vertex belongs is stored under the index corresponding to the vertex position in the trajectory \( \text{Colour} \). While updates, beyond access, may require deletion and insertion of stretches (in the case when stretches cease to exist, or when new stretches are formed), this approach allows constant-time updates in the normal case.

The one exception to this is when a stretch is split into three parts following an update \( \text{Colour}(v) \bowtie c \). Consider a cell \( v \), with \( \text{Stretch}(v) = \langle \ell, r, \alpha(\text{Colour}(v)), \sigma \rangle \) where \( v \neq \ell \) and \( v \neq r \). When a move \( \text{Colour}(v) \bowtie c \) occurs, \( v \) becomes a new, singleton stretch \( \langle v, v, c, \text{Value}(v) \rangle \). The set of vertices to the left of \( v \):

\[
\{ w \mid \ell \preceq w \prec v \}
\]

retain the old stretch, with values \( r \) set to whichever vertex precedes \( v \), and with \( \sigma \) modified accordingly. The set of vertices in the old stretch to the right of \( v \):

\[
\{ w \mid v \prec w \preceq r \}
\]

must however have their \( \text{Stretch}(w) \) values updated with an entirely new stretch, prompting an update which is asymptotically upper-bounded by \( O(|\text{Colour}|) \) in the worst case, when \( \text{Colour} \) consists of a single stretch. This will however not occur whenever a neighbourhood consisting of cells at the border of the stretches is used.
In experiments, we found that there were cases where the data would contain sequences that traverse beyond the border of the area control centre (ACC). For the STRETCH-SUM and CONNECTED constraints alike, it is not clear how to handle trajectories that exit and re-enter the ACC. In the ASTAAC analysis, experiments where out of bounds cells in the sequences were simply discarded resulted in overall better values than keeping them under a unique sector assignment, so this approach was kept from here on.

$c_4$: Minimum Distance. The minimum distance constraint, on its own, appears to have little effect on the search. This is likely because the constraint does not impose any violations on the neighbours of the cells in the trajectories. In order for the NON-BORDER constraint to have any effect, it is probably necessary to include some sort of compactness constraint which penalises empty cells (that is, cells that have an associated workload of zero). As it is, with only constraints $\{c_0, \ldots, c_4\}$ present in the model, the minimum distance constraint will have little impact other than penalising a majority of the moves that improve on the CONNECTED and STRETCH-SUM constraints.

$c_5$: Compactness. A COMPACT constraint was included towards the end of this project, and is described in Section 7.2.
5 Search

In Section 5.1 we present the search algorithm with its heuristics and meta-heuristics. In Section 5.2 we define and determine the values of the various parameters involved in the search.

5.1 Algorithm

Algorithm 1 Search

1: \( s := \) initial solution from pre-processing
2: \( s' := s \)
3: \( it := 0 \)
4: while violation > 0 and \( \text{maxit} > it \) do
5: \( \text{if random float in } [0, 100] > \frac{100}{100 + it} \) then
6: \( m := \) best non tabu move
7: else
8: \( m := \) random move
9: end if
10: if \( m \) does not break connectedness then
11: \( \text{if } m \text{ is worsening and } s \text{ is better than } s' \text{ then} \)
12: \( s' := s \)
13: end if
14: \( s := s \text{ updated with move } m \)
15: mark \( m \) as tabu for \( t \) iterations
16: end if
17: \( it := it + 1 \)
18: end while
19: if \( s' \) is better than \( s \) then
20: \( s := s' \)
21: end if
22: return \( s \)

The search, of which a general outline is provided in Algorithm 1, is carried out in the form of a TS as described in Section 2.3.1. Additionally, the search incorporates an element of randomness by sometimes allowing randomly selected, possibly tabu moves at a decreasing frequency as can be seen on line 5. This is similar to the SA meta-heuristic described in Section 2.3.2. Rather than having an acceptance function determine whether to accept worsening moves, we simply allow a random move to be selected without concern for the minimisation of the objective function or whether the move is marked as tabu. The result is a slightly more diversified search.

We begin the search by assigning \( s \) and \( s' \) to the initial solution as computed in the pre-processing step (lines 1 and 2). In the normal iterations (line 6), we probe a neighbourhood consisting of the intersection between the BORDERCELLS invariant and an
additional invariant keeping track of non-tabu cells. In the randomised iterations (line 8), the BORDERCELLS invariant is used as the neighbourhood. We also experimented with several other types of neighbourhoods. Notably, an invariant maintaining the set of most violating cells proved to be very effective for the purpose of narrowing down the neighbourhood, resulting in the selections being significantly less costly to make. The reason we proceeded with the BORDERCELLS invariant as the neighbourhood was that for the trajectory-based convexity, experiments indicated the variable violations to be less effective than the constraint violations for the purpose of guiding the search toward minimising the amount of re-entries.

In order to select a move (here defined as the re-assignment of some Colour decision variable), OscaR.cbls offers several different selectors that either minimise some function while requiring a conditional statement to evaluate to truth, or select a move at random. In the normal iterations (line 6), we used a SelectMin(neighbourhood, 1..k) selector, which identifies whichever move causes the most improving or least worsening change in the objective function. Here 1..k denotes the range of sectors \{1, 2, \ldots, k\}, and neighbourhood is the set of all decision variables eligible for a move (as described above). In the case when several equally good moves are found, one is selected at random. For the randomised iterations (line 8), we used the SelectFrom(neighbourhood, 1..k) selector, which randomly picks a move without concern for minimisation. In both cases, we also required that, in order to probe a move \text{Colour}(v) := c, a Boolean function must evaluate to truth ensuring that \text{Colour}(v) does not currently have value c, and that v has at least one adjacent cell in sector c.

Because OscaR.cbls does not inherently support the usage of delta functions, each move \text{Colour}(i) := j is instead probed by sequentially performing the move and reversing it, while measuring the change in the objective function (see Algorithm 2).

**Algorithm 2 OscaR.cbls probing**

1: OldObjective := Objective
2: OldVal := Colour(i)
3: Colour(i) := j
4: Δ := Objective − OldObjective
5: Colour(i) := OldVal

In order to make this procedure reasonably efficient (while not as efficient as dedicated delta functions), OscaR.cbls implements a mechanism for registering certain variables for partial propagation. Any such variable, when updated, only triggers changes in invariants that have registered dependencies on the variable, while postponing updates for anything that does not immediately rely on the variable [9].

Whenever a cell is assigned to a new sector, it is subsequently entered into the tabu list and prohibited for the next t iterations. Similarly, any cell that, when moved, would break the implicit CONNECTED constraint is marked as tabu. If a move \text{Colour}(v) := c has been selected, but turns out to break the implicit connectedness constraint, an attempt is made at resolving this by reassigning any empty adjacent cells to the old sector \(α(\text{Colour}(v))\). The move may only be performed if this succeeds (lines 10 – 11). In our experiments, the reassignment of empty, adjacent cells described above sometimes cause a loop of cells to exit and re-enter the sector. The problem with this is that such cells cannot
be reassigned in subsequent iterations, as doing so would break the local connectedness of its neighbours (recall that we check this by finding a path between neighbours and secondary neighbours (see Section 4.3)). We resolve this issue by only reassigning cells when the reversal of the assignment does not break local connectedness. In the results presented in Section 7.1, this error was yet to be discovered. Before conducting the final experiments presented in Section 7.2, the problem was resolved.

During the search, we want to make sure that the currently best solution encountered is never discarded in favour of a worsening solution. Whenever a move is made, we therefore check whether it is worsening, and whether the current solution is the best one encountered so far (lines 12 – 13). If that is the case, the solution is saved in $s'$ prior to the move being executed. At the end of the search, solution $s'$ is restored unless the current solution $s$ turns out to be superior (lines 20 – 23).

5.2 Parameters

The parameters for the search (see Algorithm 1) are listed below, along with their default values:

- The tabu length $t$ determines the number of iteration for which cells become prohibited during the search. For our experiments, this value was set to $\frac{n}{70}$ where $n$ is the number of cells.

- The $w_{\text{Connected}}$ and $w_{\text{StretchSum}}$ parameters control the added weight (multiplier) to the violations of the $\text{CONNECTED}$ and $\text{STRETCHSUM}$ constraints. By default, these were set to $\frac{\mu}{1000}$ and $\frac{\mu}{2000}$ respectively, where $\mu$ is the average workload of the sectors. This was so that the $\text{BALANCED}$ constraint would not completely dominate the search when the data encompassed long time-spans resulting in large workload values.

- The balance factor $\Delta$ sets the allowed discrepancy from the average workload in the $\text{BALANCED}$ constraint. This was set to 0.05.

- $\text{maxit}$ is the maximum amount of iterations. In the experiments, the search was allowed to run for 30000 iterations if the number of cells $n$ was below 20000, and otherwise 20000 iterations. The search was not allowed to run for more than 20 minutes.

For the constraint weights $w_{\text{Connected}}$ and $w_{\text{StretchSum}}$, as well as for the tabu length $t$, our default values were not picked for any particular reason other than experimentation yielding a vague idea of suitable values. A tabu length of about $t = \frac{35000}{70} = 500$ for a sectorisation involving $n = 35000$ cells, or $t = \frac{14000}{70} = 200$ for $n = 14000$ seemed to be sufficient for forcing the search out of locally optimal parts of the search space. That is not to say that this particular tabu length works well for every air space, as the trajectory patterns may differ and the ratio between empty and non-empty cells affects the significance of $n$ for the purpose of setting the tabu length.

The weight for the $\text{BALANCED}$ constraint was set to 1 (i.e., no added weight). The weight for the $\text{CONNECTED}$ constraint was twice that of the $\text{STRETCHSUM}$ constraint, since the $\text{STRETCHSUM}$ appeared to be more easily satisfied.
6 Post-Processing

A very straightforward post-processing algorithm is applied on the empty cells. First, the algorithm visits each empty cell and counts the occurrences of each sector in its adjacency list. Whichever sector is found to be most represented among the neighbours becomes the new sector of the cell, unless the move breaks connectedness. We (arbitrarily) chose to repeat this step 5 times. Upon visual inspection, this appeared to somewhat improve the shapes of the sectors.

In a second step, empty cells are once again visited in order to check whether they have non-empty neighbours belonging to a different sector. If they do, and reassigning them will not break connectedness, they are modified to share the colour of their neighbouring non-empty cells. Assuming the size of cells is above the minimum distance as described in Section 4.2.5, this step, while not entirely satisfying it, may improve the minimum distance quality of the solution.

7 Experiments

For the CBLS sectorisations, being non-deterministic, the values presented below are averages over 5 different runs. For the ASTAAC sectorisations, a single computation is sufficient given that the algorithm is fully deterministic.

All the statistics and airspace visualisations shown in this section were computed by the ASTAAC tool. The data on which the sectorisation algorithm was performed was also produced by the ASTAAC tool.

Due to the randomness of CBLS, the visualisations of our results often had to be rotated to some degree in order to give a meaningful representation of the sectorisation. This was unfortunate, since it makes it difficult to compare different sectorisations of the same airspace.

In Section 7.1, we present the results of our sectorisation algorithm. In Section 7.2, we present the results of some further experiments, after the addition of a Compact constraint.

7.1 Results

In an intermediate set of experiments, only constraints \{c_0, c_1, c_2, c_3\} were included in the model, with no concern for compactness or minimum dwell time. The searches were allowed to run for 30000 iterations at most, which normally amounted to somewhere between 10 and 20 minutes runtime on an Intel Pentium III Xeon, 2200 MHz system with 4 GB of RAM. The ASTAAC algorithm, ran on a system with an Intel Core i7, 2200 MHz processor and 8 GB of RAM, only requires about two minutes of runtime given a sectorisation involving 35000 cells [8].

In the experiments, we made an additional sectorisation (marked CBLS* in the tables below) wherein the threshold for building AFBs was increased from the default value of 3 flights to 5, yielding slightly different results for the CBLS sectorisations. This configuration causes ASTAAC to lump fewer cells into AFBs, and the result is a slightly more fine-grained mesh of cells. The same adjustment only appeared to have negative consequences for the ASTAAC algorithm, so that result is omitted below.
The parameters of the search were set to the default values described in Section 5, except in the first experiment on Madrid South ACC, where the workload balance factor $\Delta$ was decreased to 0.005 from the default value of 0.05 due to a mistake.

Along with each sectorisation scenario, we present the number of flows present in the experiments. This value is not equal to the number of flights, as a single flow may represent several flights. In the experiments, we used the actual flights rather than the flows, and here the number of flows merely serves as a lower bound on the number of flights for reference.

A short dwell time in these experiments occurs whenever a flight lingers within a sector for less than 60 seconds, hence the threshold $t$ for the STRETCHSUM constraint was always set to 60 seconds.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Workload Range</th>
<th>Entry Points</th>
<th>Re-entries</th>
<th>Short Dwell Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASTAAC</td>
<td>49 – 54</td>
<td>229</td>
<td>44</td>
<td>83</td>
</tr>
<tr>
<td>CBLS</td>
<td>51 – 52</td>
<td>232</td>
<td>21</td>
<td>9</td>
</tr>
<tr>
<td>CBLS*</td>
<td>51 – 52</td>
<td>225</td>
<td>11</td>
<td>8</td>
</tr>
</tbody>
</table>

Experiment 1: Madrid South ACC 2008-07-12, 10:00 – 12:00.
Cell dimensions: 10 NM × 10 NM × 2000 ft.
14088 cells, $k = 5$ sectors, and 110 flows. $\Delta = 0.005$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Workload Range</th>
<th>Entry Points</th>
<th>Re-entries</th>
<th>Short Dwell Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASTAAC</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>CBLS</td>
<td>56 – 66</td>
<td>256</td>
<td>19</td>
<td>9</td>
</tr>
</tbody>
</table>

Experiment 2: Munich North ACC 2008-07-12, 10:00 – 12:00.
Cell dimensions: 5 NM × 5 NM × 1000 ft.
35306 cells, $k = 5$ sectors, and 72 flows. $\Delta = 0.05$.

ASTAAC did not succeed in finding a sectorisation for this ACC, so here we did not proceed with additional CBLS* experiments.
While the results of our sectorisations varied somewhat between runs, the values were consistently superior to those of the ASTAAC sectorisation tool. The number of entry points, which is something we did not attempt to minimise in any explicit way, appears to be similar in the two algorithms. In [8], the authors point out that the number of entry points is strongly tied to the convexity as well as the compactness of the sectorisation, and
can thus be enforced by the use of such constraints rather than any explicit consideration in the form of a custom constraint.

The implicit CONNECTED constraint ($c_1$) was satisfied in all the solutions. The shapes resulting from these sectorisations (see Figures 1, 2, and 3), on the other hand, were typically very poor (as one would expect, considering compactness was not enforced). The sectors had very jagged edges and took shapes that are likely far from optimal for the purpose of ATC. The ASTAAC sectorisation tool was much more successful in this regard, as is shown in Figure 4.

![Figure 4: Madrid ACC South, ASTAAC sectorisation for reference. Full ACC to the left, individual sectors to the right.](image)

In conclusion, the constraints of [3] appear to do a good job at minimising their respective values, but expectedly cause the sectors to obtain some peculiar shapes. The implicit CONNECTED constraint provides connected sectors, but also requires that empty cells be somewhat included in the search. An alternative approach would be to only consider the trajectories during search, and somehow try to construct connected, compact components in the post-processing.

### 7.2 Improving Sector Shapes

In an attempt at improving compactness, beyond constraints \{c_0, c_1, c_2, c_3\}, we include a simplified version of the COMPACT constraint described in Section 4.2.6. The area variables $\sigma$ for each sector is initialised, and kept updated on an incremental basis just as in Section 4.2.6. However, given that an implicit CONNECTED constraint is present in our model, we do not have to deal with the concern of a move merging two components, or splitting them apart. This allows us to perform an update $Colour(v) := c$ in time linear in the degree of $v$, rather than in time $\Theta(|V| + |E|)$ (it is only a matter of summing the new borders between $v$ and every $w \in \text{Adj}(v)$ while incrementing or decrementing the borders of the cells in $\text{Adj}(v)$).

The constraint violation subject to minimisation of this simplified constraint is:

$$\sum_{(\sigma, v) \in \text{CCs}} \sigma$$
i.e., the sum of the border surfaces of all sectors (since, again, we only permit one CC per sector). Violations for the Colour decision variables are set to their total amount of surface area toward other sectors.

The search is largely performed as in Section 5, with the exception that out of every 100 iterations, the last 70 are spent improving the shape of the sectors. The distribution between normal and shape iterations was determined experimentally by visual inspection of the resulting sectors. In the iterations improving the shapes, a set consisting of the most violating cells with respect to the COMPACT constraint intersected with the currently non-tabu cells is considered for moves during probing. The most violating cells are maintained as an invariant, which incrementally computes the output variables at updates. When several cells share the highest current violation, they are all considered. In the shape iterations, we forbid the reassignment of any cells containing non-zero workload. We only allow improving moves when considering compactness, given that the constraint violation cannot actually be minimised to zero, and it is not possible to determine whether some solution is optimal with regard to compactness. Because the shape iterations were not very costly, we also increased the maximum amount of iterations to twice those in Section 5.2 (although we kept the 20 minute cap). For the tabu length \( t \) to have the same effect as before, we set it to twice the previous value of \( \frac{n}{70} \). We did not perform the post-processing step in these experiments, as it did not appear to have any significantly positive effect on the results when the sectors were not as jagged as in previous experiments.

Another mechanism was also implemented as part of the search; whenever a cell is moved in a normal iteration (i.e., not a shape iteration), we attempt to move adjacent cells to the new sector. Furthermore, for each adjacent cell successfully moved, we also attempt to reassign its adjacent cells. We do however refrain from entering such additional moves into the tabu list.

Here we discovered a problem with the BALANCED constraint, which was resolved prior to running the experiments below (see Section 4.3). As in Section 7.1, CBLS* signifies a lowered AFB threshold.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Workload Range</th>
<th>Entry Points</th>
<th>Re-entries</th>
<th>Short Dwell Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASTAAC</td>
<td>49 – 54</td>
<td>229</td>
<td>44</td>
<td>83</td>
</tr>
<tr>
<td>CBLS w/ compactness</td>
<td>49 – 54</td>
<td>222</td>
<td>29</td>
<td>5</td>
</tr>
<tr>
<td>CBLS w/o compactness</td>
<td>51 – 52</td>
<td>232</td>
<td>21</td>
<td>9</td>
</tr>
<tr>
<td>CBLS* w/ compactness</td>
<td>49 – 54</td>
<td>223</td>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>CBLS* w/o compactness</td>
<td>51 – 52</td>
<td>225</td>
<td>11</td>
<td>8</td>
</tr>
</tbody>
</table>

Experiment 4: Madrid South ACC 2008-07-12, 10:00 – 12:00.
Cell dimensions: 10 NM \( \times \) 10 NM \( \times \) 2000 ft.
14088 cells, \( k = 5 \) sectors, and 110 flows. \( \Delta = 0.05 \) in the experiments with compactness, and \( \Delta = 0.005 \) in those without.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Workload Range</th>
<th>Entry Points</th>
<th>Re-entries</th>
<th>Short Dwell Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASTAAC</td>
<td>32 – 38</td>
<td>210</td>
<td>20</td>
<td>86</td>
</tr>
<tr>
<td>CBLS w/ compactness</td>
<td>34 – 38</td>
<td>232</td>
<td>39</td>
<td>7</td>
</tr>
<tr>
<td>CBLS w/o compactness</td>
<td>32 – 38</td>
<td>205</td>
<td>21</td>
<td>13</td>
</tr>
<tr>
<td>CBLS* w/ compactness</td>
<td>34 – 38</td>
<td>229</td>
<td>27</td>
<td>6</td>
</tr>
<tr>
<td>CBLS* w/o compactness</td>
<td>33 – 37</td>
<td>219</td>
<td>24</td>
<td>5</td>
</tr>
</tbody>
</table>

Experiment 5: Munich North ACC 2008-07-12, 10:00 – 12:00.  
Cell dimensions: 5 NM × 5 NM × 1000 ft.  
35306 cells, k = 5 sectors, and 72 flows. Δ = 0.05.

Figure 5: CBLS w/ compactness sectorisation of Madrid South ACC: A fairly typical result of our algorithm.
Figure 6: CBLS* w/ compactness sectorisation of Madrid South ACC: A fairly typical result of our algorithm.

This version of the COMPACT constraint does have some positive effects on the shapes of the sectors (see Figures 5, 6, and 9), but often left a lot to be desired in terms of overall shape. An obvious improvement relative to previous sectorisations is that the sectors were not nearly as jagged (see Figure 7). The sectors also suffered to a lesser extent from the protruding sequences of cells that would sometimes occur in previous experiments (although they still occur; see Figures 8, 10, and 11 for instance).

Numerically, the results were typically similar to those in previous experiments, but there were some differences. Notably, in experiment 5, the number of re-entries was nearly twice that of the same scenario prior to compactness being enforced. One plausible explanation for this is that sectors were not allowed to grow as single strands of cells along the trajectories like they were before, as such formations severely violate the COMPACT constraint. This would likely somewhat restrict the effect of the CONNECTED constraint. The CBLS* sectorisations were more successful in this regard, likely as a result of a finer granularity in the mesh of cells. Moving large AFBs is often costly with regard to compactness, which would restrict the ability of the search to minimise the violation of the CONNECTED constraint.

The sectorisations of Madrid South ACC, with its 14088 cells, generally turned out much better than the sectorisations of Munich North ACC, with 35306 cells, both in terms of shapes and numerically. We are not sure of the exact cause of this, but the larger data set of Munich would likely need a longer search in order for the sectorisations to reach similar results to those in the Madrid case. It may also be that the values of the constraint weights for CONNECTED and STRETCHSUM, as well as the tabu length \( t \), were more suitably tuned for the Madrid case. These values are dependent on the number of cells as well as the amount of traffic in the sectorisation scenarios.
Unsurprisingly, the compactness results appear to be somewhat dependent on the quality of the initial sectorisations as constructed in the pre-processing step. For some ACCs of peculiar shapes, coupled with the presence of AFBs, compactness becomes very hard to satisfy.

Figure 7: Left: Two sectors from Madrid South ACC after compactness was enforced. Right: Two significantly more jagged sectors from the same ACC before compactness was enforced.

The COMPACT constraint of Section 4.2.6 likely provides a much better measure of compactness, but due to time constraints we were not able to attempt such a constraint, as it would require a fairly comprehensive revision of the pre-processing algorithm (given that such a constraint would need access to the volume of AFBs among other things).

Figure 8: CBLS w/ compactness sectorisation of Madrid South ACC. Some sectors turned out better than others. Bottom: a very awkward sector, despite the compactness constraint, partly as a result of the shape of the ACC, and the presence of AFBs in the experiment.
Figure 9: Left: CBLS w/ compactness sectorisation of Munich North ACC. Right: 4 individual sectors.

Figure 10: Left: Another CBLS w/ compactness sectorisation of Munich North ACC. Right: 5 individual sectors.
Figure 11: Left: CBLS* w/ compactness sectorisation of Munich North ACC. Right: 5 individual sectors, some of which were very poor.

8 Discussion

8.1 Conclusion

We have implemented a CBLS approach to airspace sectorisation using the CBLS engine of the OscaR solver, based on the constraints detailed in [3]. The resulting algorithm has been applied on real air space data and compared against the ASTAAC sectorisation tool with regard to several relevant constraints previously outlined in [3, 4]. Finally, an attempt was made at improving the shapes resulting from our sectorisation algorithm by the use of a COMPACT constraint. This yielded certain positive results, but appeared to take a toll on the effect of the other constraints in the model.

As was touched upon earlier, systematic search, while lacking the scalability of local search, offers completeness and optimality. For a problem of this kind, however, sub-optimal solutions can arguably be of sufficient quality for the purpose of ATC, making the trade-off in completeness and optimality worthwhile. We also want our sectorisation algorithm to be able to handle possibly huge problem instances such as entire continents, which is another argument in favour of local search. There are however many other techniques for the solving of difficult optimisation problems, and whether local search is the best alternative among them is another issue entirely.

In OscaR.cbls, the lack of delta functions was problematic due to the fact that probing the moves became very expensive as the number of cells grew large, unless measures were taken to significantly narrow down the neighbourhood by various heuristics. In some cases, a delta function providing a cheap over-approximation of the impact of a given move is highly preferable to computing the exact delta. The compactness constraint of [3], for instance, relies on this type of estimation in order to avoid the costly $\Theta(|E| + |V|)$ time (for a graph $G = \langle V, E \rangle$ representing an airspace) operations required for measuring the exact impact. Due to the lack of delta functions, we are not able to make such cheap approximations, granted that the constraint itself must provide a more exact measure of the dissatisfaction of some condition.
Implementing constraints in a problem-independent fashion is something that has proven somewhat difficult during this work. It often seems to be the case that certain modifications have to be made depending on the characteristics of the problem and the structure of the data.

8.2 Future Work

As was noted in Section 7.2, despite minimisation of the surface area between the sectors, our sectorisations still left a lot to be desired in terms of sector shapes. The COMPACT constraint of Section 4.2.6 remains to be implemented. Such a constraint would likely provide a more reliable measure of compactness, and provides a penalty that can be minimised near to zero, which was not the case for the simplistic COMPACT constraint in this thesis.

We speculate that in order to further speed up the search algorithm, it may not be necessary to compute the most improving move in each iteration. Rather, a search procedure might compute the $j$ most improving moves and perform them one by one, before computing the next set of $j$ moves, where the value of $j$ would have to be tuned experimentally.

CBLS in general appears to be a well-suited alternative for airspace sectorisation. Due to the flexible manner in which constraints are handled, our algorithm can easily be configured to satisfy other constraints as they become of interest.
References


