Propagators and Violation Functions for Geometric and Workload Constraints Arising in Airspace Sectorisation *

Pierre Flener and Justin Pearson
Department of Information Technology
Uppsala University, Box 337, SE – 751 05 Uppsala, Sweden
Firstname.Surname@it.uu.se

Revision of 29 January 2014

Abstract

Airspace sectorisation provides a partition of a given airspace into sectors, subject to geometric constraints and workload constraints, so that some cost metric is minimised. We make a study of the constraints that arise in airspace sectorisation. For each constraint, we give an analysis of what algorithms and properties are required under systematic search and stochastic local search.

1 Introduction

We continue our work in [11, 12] on applying constraint programming (see Section 3) to airspace sectorisation (see Section 2). In that work, we used existing constraints or implemented new ones in an ad hoc fashion just to solve the problem at hand, without much theoretical analysis of the constraints and their underlying algorithms.

We give a complete theoretical analysis of constraints that arise in airspace sectorisation. For each constraint, we analyse what propagation is possible under systematic search. Under stochastic local search, we give efficient algorithms for maintaining constraint and variable violations, as well as efficient algorithms for probing the effect of local search moves; such algorithms are necessary if stochastic local search is to be done efficiently.

The remainder of this report is structured as follows. Towards making it self-contained, we first give in Section 2 a brief overview on airspace sectorisation, and in Section 3 a brief tutorial on constraint programming, especially about its core concept: a constraint is a reusable software component that can be used declaratively, when modelling a combinatorial problem, and procedurally, when solving the problem. Experts on airspace

*This work has been co-financed by the European Organisation for the Safety of Air Navigation (EUROCONTROL) under its Research Grants programme (contract 08-1214447-C). The content of the work does not necessarily reflect the official position of EUROCONTROL on the matter.

sectorisation or CP may safely skip these sections. Next, in Section 4, we lay the mathematical foundation for specifying, in Section 5, the constraints identified above to occur commonly in airspace sectorisation problems Finally, in Section 6, we summarise our contributions and identify avenues for future work.

2 Background: Airspace Sectorisation

Airspace *sectorisation* provides a partition of a given airspace into a given (or upper-bounded, or minimal) number of control sectors, subject to geometric constraints and workload constraints, so that some cost metric is minimised. The entire material of this section is a suitably condensed version of Section 2 of our report [6], where we have surveyed the algorithmic aspects of methods for automatic airspace sectorisation, for an intended readership of experts on air traffic management.

In this report, we focus on *region-based* models of airspace sectorisation, where the airspace is initially partitioned into some kind of regions that are smaller than the targeted sectors, so that the combinatorial problem of partitioning these regions in principle needs no geometric post-processing step. Sectorisation then starts from regions of (any combination of) the following granularities: a mesh of blocks of the same size and shape; ATC functional blocks (AFBs); elementary sectors, namely the ones of the existing sectorisation; control sectors; areas of specialisation (AOS); and air traffic control centres (ATCC).

We assume in this report that the airspace sectorisation is computed in *three* dimensions, hence the ideas scale down to two dimensions. For simplifying the discussion, we assume without loss of generality in this report that the number of sectors is a *given* constant, rather than upper-bounded or to be minimised.

Airspace sectorisation aims at satisfying some constraints. The following constraints have been found in the literature, so that a subset thereof is chosen for a given tool:

- *Balanced workload*: The workload of each sector must be within some given imbalance factor of the average across all sectors.
- *Bounded workload*: The workload of each sector must not exceed some upper bound.
- *Balanced size*: The size of each sector must be within some given imbalance factor of the average across all sectors.
- *Minimum dwell time*: Every flight entering a sector must stay within it for a given minimum amount of time (say two minutes), so that the coordination work pays off and that conflict management is possible.
- *Minimum distance*: 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.
- *Convexity* of the sectors. Convexity can be in the usual *geometric* sense, or *trajectory-based* (no flight enters the same sector more than once), or more complex.

- *Connectedness*: A sector must be a contiguous portion of airspace and can thus not be fragmented into a union of unconnected portions of airspace.

- *Compactness*: A sector must have a geometric shape that is easy to keep in mind.

- *Non-jagged boundaries*: A sector must have a boundary that is not too jagged.

For each sector, there are three kinds of workload: the *monitoring workload*, the *conflict workload*, and the *coordination workload*; the first two workloads occur inside the sector, and the third one between the sector and an adjacent sector. The quantitative definition of workload varies strongly between papers.

Airspace sectorisation often aims at minimising some cost. The following costs have been found in the literature, so that a subset thereof is combined into the cost function for a given tool, the subset being empty if sectorisation is not seen as an optimisation problem:

- *Coordination cost*: The cost of the total coordination workload between the sectors must be minimised.

- *Transition cost*: The cost of switching from the old sectorisation to the new one must be minimised.

- *Workload imbalance*: The imbalance between the workload of the sectors must be minimised.

- *Number of sectors*: The number of sectors must be minimised.

- *Entry points*: The total number of entry points into the sectors must be minimised.

- If any of the constraints above is soft, then there is the additional cost of minimising the number of violations of soft constraints.

Constraints are to be *satisfied*, hence the existence or not of a cost function whose value is to be *optimised* does not affect the design of a constraint, in the sense discussed in the following section. Hence we will no further discuss cost functions in this report.

3 Background: Constraint Programming

First, we show how to model combinatorial problems at a very high level of abstraction with the help of the declarative notion of *constraint* (Section 3.1). One distinguishes between *satisfaction problems*, where there is no objective function, and *optimisation problems*, where there is an objective function. Then, we describe two methods for solving combinatorial problems by a mixture of inference and search, upon reusing algorithms that implement these constraints (Section 3.2). The entire material of this section is a condensed version of our tutorial in Section 2 of [2].

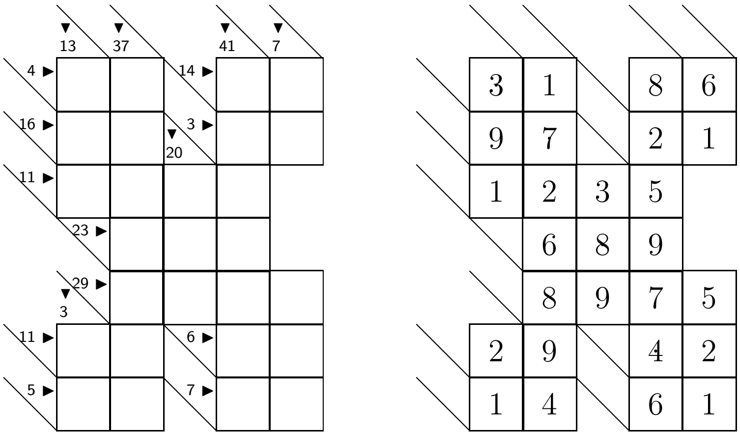


Figure 1: A Kakuro puzzle (on the left) and its solution (on the right), both taken from the crossword package for LaTeX by Gerd Neugebauer at `CTAN.org`.

3.1 Problem Modelling via Constraints

Constraint programming (CP) is a successful approach to the modelling and solving of combinatorial problems. Its core idea is to capture islands of structure, also called combinatorial sub-structures, that are commonly occurring within such problems and to encapsulate declaratively their procedural inference algorithms by designing specialised reusable software components, called *constraints*.

As a running example, let us consider the Kakuro puzzle, with the caveat that this is *not* meant to imply that the discussed techniques are limited to (small) puzzles. See the left side of Figure 1: Each word of a Kakuro puzzle is made of non-zero digits, under the two constraints that the letters of each word are pairwise distinct and add up to the number to the left (for horizontal words) or on top (for vertical words) of the word. Hence there is one decision variable for each letter of the puzzle, with the integer set $\{1, 2, \dots, 9\}$ as domain. This is a satisfaction problem, as there is no objective function. The solution to this puzzle is given on the right side of Figure 1; note that a Kakuro puzzle is always designed so as to have a unique solution.

Beside the usual binary comparison constraints ($<$, $\leq$, $=$, $\neq$, $\geq$, $>$), with arguments involving the usual binary arithmetic operators ($+$, $-$, $\cdot$, etc), a large number of *combinatorial constraints* (usually called *global constraints* in the literature), involving more complex structures and a non-fixed number of arguments, have been identified.

For example, the AllDifferent($\{x_1, \dots, x_n\}$) constraint requires its n decision variables x_i to take pairwise different values. This constraint is useful in many challenging real-life problems, as well as for modelling the first constraint of the Kakuro puzzle, as there is an AllDifferent constraint on each word. Each such constraint wraps a conjunction of $\frac{n\cdot(n-1)}{2}$ binary $x_i \neq x_j$ constraints into a single n-ary constraint; this enables a more global view of the structure of the problem, which is a prerequisite for solving it efficiently (as we shall see in Section 3.2).

As another example, the Linear($\{x_1, \dots, x_n\}$, *RelOp*, c) constraint requires the sum $x_1 + \cdots + x_n$ of the n decision variables x_i to be in relation *RelOp* with the constant c.

This constraint can be used for modelling the second constraint of the Kakuro puzzle, as there is a $\textsc{Linear}(\{x_1, \ldots, x_n\}, =, s)$ constraint for each word $[x_1, \ldots, x_n]$ with required sum s. The generalised constraint $\textsc{Linear}([a_1, \ldots, a_n], [x_1, \ldots, x_n], \mathit{RelOp}, c)$ requires the *weighted* sum $a_1 \cdot x_1 + \cdots + a_n \cdot x_n$ of the n decision variables x_i to be in relation *RelOp* with the constant c, where the weights a_i are constants.

Hundreds of useful constraints have been identified and are described in the *Global Constraint Catalogue* [4], the most famous ones being ELEMENT [20], CUMULATIVE [1], ALLDIFFERENT [17], and CARDINALITY [18].

Constraints can be combined to *model* a combinatorial problem in a declarative and very high-level fashion. For example, the Kakuro puzzle can essentially be modelled as follows, assuming that a hint for word w with required sum s is encoded as the pair $\langle w, s\rangle$:

$$\begin{array}{c} \textbf{for each } \mathit{hint}\ \langle [x_1, \ldots, x_n], s\rangle : \\ \textsc{AllDifferent}(\{x_1, \ldots, x_n\}) \wedge \textsc{Linear}(\{x_1, \ldots, x_n\}, =, s) \end{array} \tag{1}$$

For combinatorial problems, CP is *not* limited to decision variables whose domains are finite sets of integers. For instance, domains can be finite sets of *finite sets of integers*, and we then speak of *set decision variables* and *set constraints*. We call *universe* the union of the sets in the domain of a set decision variable; the set value eventually taken by a decision set variable is then a subset of the universe.

In the following sub-section, we show that constraints are not just a convenience for high-level modelling, but can also be exploited during the solution process.

3.2 Problem Solving by Inference and Search

By *intelligent search*, we mean search where at least some form of inference takes place at every step of the search in order to reduce the cost of brute-force search [9]:

$$\text{combinatorial problem solving} \;=\; \text{search} \;+\; \text{inference} \;+\; \cdots$$

In the following, we discuss two CP ways of solving problems that have been modelled using constraints. The classical approach is to perform systematic search (Section 3.2.1), and a more recent approach is to trade for time the guarantees of systematic search by performing stochastic local search (Section 3.2.2): the two approaches use completely different forms of inference, which is encapsulated in reusable fashion within the constraints (Section 3.2.3).

3.2.1 Systematic Search

Classically, CP solves combinatorial problems by systematic tree search, together with backtracking, and performs at every node of the search tree a particular kind of inference called *propagation*. For the purpose of this report, we only need to explain the propagation of a single constraint here: we refer to [2] for how to propagate multiple constraints and how systematic search is conducted.

For each individual constraint, a *propagation algorithm* (or *propagator*) prunes the domains of its decision variables by eliminating impossible values according to some desired level of *consistency*. For example, under *domain consistency* (DC) every domain *value* of every decision variable participates in some solution to the constraint that involves domain values of the other decision variables. Also, under *bound consistency* (BC) every domain *bound* of every decision variable participates in some solution to the constraint that involves domain values of the other decision variables.

For example, consider the constraint $\textsc{AllDifferent}(\{x, y, z\})$. Let x, y range over the domain $\{1, 3\}$ and z over $\{1, 2, 3, 4\}$: we write $\mathrm{dom}(x) = \mathrm{dom}(y) = \{1, 3\}$ and $\mathrm{dom}(z) = \{1, 2, 3, 4\}$ and denote this state by $\{x, y \mapsto \{1, 3\}, z \mapsto \{1, 2, 3, 4\}\}$. From this state, propagation to DC leads to the state $\{x, y \mapsto \{1, 3\}, z \mapsto \{2, 4\}\}$ since x and y must split the values 1 and 3 between themselves so that z cannot take any of these two values. From the same start state $\{x, y \mapsto \{1, 3\}, z \mapsto \{1, 2, 3, 4\}\}$, propagation to BC leads to the state $\{x, y \mapsto \{1, 3\}, z \mapsto \{2, 3, 4\}\}$ since there do exist solutions to the constraint where z takes its new lower bound value 2 or its old (and new) upper bound value 4, so that the unfeasibility of the intermediate value 3 is not even checked. Note that, in this case, the resulting DC state is strictly stronger than the resulting BC state: while the initial state encodes a set of $2 \cdot 2 \cdot 4 = 16$ candidate solutions, the BC state encodes a subset thereof with $2 \cdot 2 \cdot 3 = 12$ candidate solutions, and the DC state encodes a subset of both with only $2 \cdot 2 \cdot 2 = 8$ candidate solutions, including the 4 solutions. From a second start state $\{x, y, z \mapsto \{1, 2\}\}$, propagation to DC or BC leads to the propagator signalling *failure*, because it is impossible to assign two values to three variables so that the latter are pairwise distinct. From a third start state $\{x \mapsto \{1\}, y \mapsto \{3\}, z \mapsto \{2, 4\}\}$, propagation to DC or BC leads to no propagation, but the propagator can simultaneously detect that all $1 \cdot 1 \cdot 2 = 2$ candidate solutions actually are solutions, so that the propagator can signal *subsumption* (or *entailment*).

The propagation of a constraint amounts to *reasoning with possible domain values*, but there is *no* obligation to prune *all* the impossible domain values, as just witnessed when comparing DC and BC. If a constraint has multiple propagators achieving different strengths of consistency (under different time complexities), then there is a default propagator but the modeller may also choose one of them, possibly via experiments to find out which one leads to the best trade-off in search effort; this is the first non-declarative annotation that may be added to an otherwise declarative constraint model (and we will encounter a second one below). Also, the propagator of a constraint only reasons *locally*, namely about the decision variables of that constraint, rather than globally, about all the decision variables of the entire problem.

3.2.2 Stochastic Local Search

Systematic search (as just described) explores the *whole* search space, though *not* by explicitly trying all possible combinations of domain values for the decision variables, but *implicitly* thanks to the interleaving of search with inference. Suitable values are found *one-by-one* for the decision variables. Systematic search offers the guarantee of eventually finding a solution (or finding and proving an optimal solution, in the case of an optimisation problem), if one exists, and proving unsatisfiability otherwise. However, this may take too long and it may be more interesting in some situations to find quickly solutions that may violate some constraints (or may be sub-optimal). The idea of *stochastic local search* (SLS; see [10], for example) is to trade this guarantee for speed by not exploring the whole search space. Unsatisfiability of the constraints is *a priori* not detectable by SLS, and optimality of solutions is a priori not provable by SLS.

SLS starts from a possibly random assignment of domain values to *all* the decision variables, without concern for whether some constraints are violated. It then tries to find a better assignment (in the sense of violating fewer constraints, or violating some constraints less, or yielding a better value of the objective function) by changing the values of a few decision variables, upon probing the impacts of many such small changes, which are called *moves*, and then actually selecting and making one of these moves. The set of candidate moves is called the *neighbourhood*. This iterates, under suitable heuristics and meta-heuristics, until a sufficiently good assignment has been found, or until some allocated resource (such as running time or a number of iterations) has been exhausted.

SLS is an area of intensive research on its own, but the CP concept of constraint can be usefully imported into SLS, giving rise to what is known as *constraint-based local search* (CBLS; see [21] for example). In principle, the declarative part of a constraint model is thus the *same* as when solving the problem by classical CP (by systematic tree search interleaved with propagation). The inference counterpart of the propagator of a constraint are its violation functions and its differentiation functions, discussed next. For the purpose of this report, we only need to explain these functions for a single constraint here: we refer to [2] for how to evaluate them for multiple constraints and how stochastic local search is conducted.

For each individual constraint, the following functions are required in a CBLS system:

- The *constraint violation function* gives a measure of how much the constraint is violated under the current assignment. It must be zero if and only if the constraint is satisfied, and positive otherwise.

- The *variable violation function* gives a measure of how much a suitable change of a given decision variable may decrease the constraint violation.

- The *assignment delta function* gives the exact or approximated increase in constraint violation upon a probed $x := d$ assignment move for decision variable x and domain value d.

- The *swap delta function* gives the exact or approximated increase in constraint violation upon a probed $x :=: y$ swap move between two decision variables x and y.

A constraint or decision variable with higher violation is a stronger candidate for repair by a move. A negative delta reflects a decrease in constraint violation, hence smaller deltas identify better moves. Differentiation functions for other kinds of moves, such as multiple assignments, can be added. Ideally, violations are updated incrementally in constant time upon the actual making of a move, but this is not always possible. Similarly, deltas are ideally computed differentially in constant time rather than by subtracting the constraint violations after and before the probed move.

For example, consider the constraint $\textsc{AllDifferent}(\{v, w, x, y, z\})$. Under the assignment $\{v \mapsto 4, w \mapsto 4, x \mapsto 5, y \mapsto 5, z \mapsto 5\}$, the constraint violation could be 3, because three variables need to take a suitable new value in order to satisfy the constraint, and the variable violation of y could be 1, because the constraint violation would decrease by one if y were assigned a suitable new value, such as 6. Upon the assignment moves $y := 4$ and $y := 6$, the constraint violation increases by 0 and -1, respectively, so the latter probed move is better. Upon *any* swap move, the constraint violation increases by 0. When maintaining for every domain value the number of variables currently taking it, the violations can be updated in constant time upon an actual move, and the deltas can be computed in constant time for a probed move.

A neighbourhood can often be designed so that some constraints of the model remain satisfied if they are satisfied under the starting assignment. Such constraints are called *implicit* constraints, since they need not be given in the constraint model, whereas the constraints to be satisfied through search are called *explicit* constraints and must be given in the constraint model. Since the explicit constraints can be violated under the current assignment, they are often called *soft* constraints. Conversely, since the implicit constraints can never be violated, they are often called *hard* constraints.

For example, in a Sudoku puzzle, there is an AllDifferent constraint on each of the nine rows, columns, and 3×3 blocks: the row AllDifferent constraints can be made implicit upon using a neighbourhood with swap moves inside rows, since these constraints can be satisfied under the starting assignment (obtained by generating nine random permutations of the sequence $[1, 2, \ldots, 9]$) and remain satisfied upon swap moves.

3.2.3 Conclusion about Search and the Role of Constraints in Search

Both in constraint-based systematic search and in constraint-based local search, a problem solver software (or simply: *solver*) need only provide the master search algorithm, as well as implementations of the built-in (meta-)heuristics and constraints that are used in the problem model. The modeller is free to design custom (meta-)heuristics and constraints. A constraint fully declaratively encapsulates inference algorithms (propagators or violation functions and delta functions), which have been written once and for all and are invoked by the master search algorithm and the (meta-)heuristics in order to conduct the search for solutions.

The usage of constraints achieves code *reusability*. It also entails a clean separation between the declarative and non-declarative parts of the problem model (which together form the input to the solver), as well as a clean separation between search and inference within the solver itself. The slogan of constraint programming is:

$$\text{constraint program} \ = \ \text{model} \ + \ \text{search}$$

because we also have more code *modularity*.

4 Mathematical Formulation

We consider two models of airspace sectorisation: one model uses a decision variable per region, its domain being the set of sectors; the other model uses a set decision variable per sector, its universe being the set of regions. A third approach is to consider both models at the same time, and add channelling constraints between them that ensure the two models are consistent. All work we surveyed in [6] has been done on the first approach, as the set variable approach might be unrealistic because the sets could get too large for current CP solvers. We refer to the first model as the *graph colouring approach*, and to the second model as the *set covering approach*. In both models, there is the same background data:

1. A given set *Regions* of regions, where each region is uniquely identified.

2. For each region, there is a given workload value, or, if there is more than one type of workload, a tuple of workload values. The set of possible (tuples of) workload values is denoted by *Workloads*. The workload of each region is given by a function Workload: *Regions* $\rightarrow$ *Workloads*.

3. A function C that takes a set of regions and returns the combined workload for that set considered as a sector.

4. Each flight is given as the sequence of regions it visits, together with entry and exit time stamps.

5. The background geometry is given as a graph.

In most work, the background geometry was handled implicitly in the definitions of the constraints, while workload and flight were given as parameters to the model.

There are three different types of workload: monitoring workload, conflict workload, and coordination workload. If all three types of workload are considered, then the set *Workloads* is the Cartesian product $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$, without loss of generality. In our previous work [11, 12], we ignored coordination workload and combined monitoring and conflict workload into a single integer: the set *Workloads* was $\mathbb{N}$.

In general, the function C performs an addition or a weighted sum. It gets complicated when coordination workload is taken into account, because the coordination between two regions of the same sector need not be taken into account, so that coordination workload is not additive.

In what follows, we shall leave the handling of the aggregation of coordination workload as future work. We assume that the workload of a region is given as single integer, and that the workload combination function C sums up the workloads of the regions in a region set R constituting a sector, that is:

$$\mathrm{C}(R) = \sum_{r \in R} \mathrm{Workload}(r)$$

4.1 Background Geometry

We assume that the airspace is divided into a countable number of regions. These regions can be any shape with flat sides, that is polytopes. What is important is that we have a graph connecting adjacent regions. In the following, for generality of purpose and notation, we rename the set *Regions* into a vertex set V. Formally, the background geometry G induces an undirected graph

$$\langle V, E \subseteq V \times V \rangle$$

where E is a set of undirected edges. Given $v \in V$, we define $\mathrm{Adj}(v)$ to be the set of vertices that are adjacent to v:

$$\mathrm{Adj}(v) = \{w \in V \mid \langle v, w \rangle \in E\}$$

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 [14], 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 [14].

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 ∂ from vertices V to facets 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

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

Given a graph and a facet structure with facet function ∂, 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 $\mathrm{pred}(v)$ denote the predecessor of vertex v in V; if v is the leftmost vertex in V, then $\mathrm{pred}(v) = \perp$.
- Similarly, let $\mathrm{succ}(v)$ 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: $V \to \mathbb{N}$, such that $\mathrm{Volume}(r)$ returns the volume of region r.
- Area: $V \times F \to \mathbb{N}$, such that $\mathrm{Area}(r, f)$ 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.

4.2 Flight Data

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

$$p = [\langle v_1, t_1, t'_1 \rangle, \langle v_2, t_2, t'_2 \rangle, \ldots, \langle v_m, t_m, t'_m \rangle]$$

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 $\mathit{Plan}(f)$.

4.3 The Graph Colouring Approach

Consider a graph $G = \langle V, E \rangle$. Let the given number of sectors be n. For each vertex v in V, we create a decision variable $\mathit{Colour}(v)$ with domain $\{1, \ldots, n\}$. The goal is to assign values to each decision variable such that we have a valid sectorisation. A solution to such a constraint problem is a mapping $\mathit{Colour}\colon V \to \{1, \ldots, n\}$. We assume that solving starts with the special vertex $\perp$ being given a colour not made available to the vertices of G.

For stochastic local search, the current assignment α induces an undirected graph *ColourGraph* with V as vertex set: there is an edge between vertices v and w if and only if they are adjacent in G and have the same colour under α. Formally:

$$\langle v, w \rangle \in \mathit{ColourGraph} \Leftrightarrow \langle v, w \rangle \in E \wedge \alpha(\mathit{Colour}(v)) = \alpha(\mathit{Colour}(w)) \tag{2}$$

Note that *ColourGraph* dynamically changes during search as α changes.

For systematic search, the current state induces an undirected graph *ColourGraph* with V as vertex set: there is an edge between vertices v and w if and only if they are adjacent in G and currently have non-disjoint domains, that is they *may* eventually become part of the same connected component of *ColourGraph*. Formula (2) becomes:

$$\langle v, w \rangle \in \mathit{ColourGraph} \Leftrightarrow \langle v, w \rangle \in E \wedge \mathrm{dom}(\mathit{Colour}(v)) \cap \mathrm{dom}(\mathit{Colour}(w)) \neq \varnothing$$

In a sequence, we call *stretch* a maximal sub-sequence whose elements are all equal. For instance, the sequence $[1, 1, 4, 4, 4, 4, 4, 4, 1, 1, 1, 2]$ has four stretches, namely $[1, 1]$, $[4, 4, 4, 4, 4, 4]$, $[1, 1, 1]$, and $[2]$. A stretch in a sequence is the specialisation for a one-dimensional geometry of the notion of connected component in *ColourGraph*. In the formalisations of constraints in Section 5, we use the $\mathit{Stretch}(X, \ell, r)$ predicate, which holds if and only if the sequence X has a stretch from index ℓ to index r:

$$\mathit{Stretch}(X, \ell, r) \Leftrightarrow X[\mathrm{pred}(\ell)] \neq X[\ell] = X[r] \neq X[\mathrm{succ}(r)] \wedge \forall \ell \prec i \prec r : X[i] = X[\ell]$$

where X is assumed to start and end with a unique value, and ℓ and r are neither the first nor the last indices inside X.

4.4 The Set Covering Approach

Let the given number of sectors be n. We use n set decision variables $S_1, \ldots, S_n$, which are to be assigned subsets of the universe set V. In addition to the other constraints in the problem that restrict the assignments of $S_1, \ldots, S_n$ to be valid sectors, we require that all the sets be pairwise disjoint.

5 Geometric and Workload Constraints

We now consider the most important and most representative of the airspace sectorisation constraints listed in Section 2. We encapsulate each as a constraint in the constraint programming sense (as seen in Section 3.1), give its semantics, and design the encapsulated inference algorithms, namely propagators for systematic search (as seen in Section 3.2.1), as well as violation and differentiation functions for stochastic local search (as seen in Section 3.2.2). It is important to keep in mind that these algorithms and functions are designed in a problem-*independent* fashion, so as to make the constraints highly reusable.

We leave the set colouring approach as future work, and assume the graph colouring approach for all the constraints:

- We consider a graph G with vertex set V and edge set E.
- We look for a mapping $\mathit{Colour}\colon V \to \mathit{Colours}$, that is we create a colour decision variable $\mathit{Colour}(v)$ for every vertex $v \in V$. Let $\mathit{Colours} = \bigcup_{v \in V} \mathrm{dom}(\mathit{Colour}(v))$ be the set of available colours.
- Let *Value* be a sequence, indexed by V, of integer values.

The notation $[\phi]$ uses the Iverson bracket (typeset in red for the convenience of those viewing this document in colour) to represent the truth of formula ϕ, with truth represented by 1 and falsity by 0.

5.1 Connectedness

The $\textsc{Connected}(G, \mathit{Colour}, \mathit{RelOp}, N)$ constraint, with $\mathit{RelOp} \in \{\leq, <, =, \neq, >, \geq\}$, holds if and only if the number of colours used in the sequence *Colour* is in relation *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:

$$\begin{gathered} |\{\mathit{Colour}(v) \mid v \in V\}| \ \ \mathit{RelOp} \ N \ \land \\ \forall v, w \in V : \mathit{Colour}(v) = \mathit{Colour}(w) \Rightarrow \langle v, w \rangle \in E^*_{\mathit{Colour}(v)} \end{gathered} \tag{3}$$

where E^*_c denotes the transitive closure of E_c, which is the adjacency relation E projected onto adjacency for vertices of colour c:

$$E_c = \{\langle v, w \rangle \in E \mid \mathit{Colour}(v) = c = \mathit{Colour}(w)\}$$

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

Arbitrary Number of Dimensions. The CONNECTED(G, *Colour*, *RelOp*, N) constraint generalises the main aspect of the CONNECT_POINTS(w, h, d, *Colour*, N) constraint of [4], which considers *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. An initial idea for a propagator is outlined in [4]: we flesh it out below and also give violation and differentiation functions.

The constraint CONNECTED is a hard constraint in our prior work on airspace sectorisation using stochastic local search [12], hence no violation and differentiation functions are given there. A connectedness constraint was accidentally forgotten in our prior work on airspace sectorisation using systematic search [11], hence no propagator is given there.

One Dimension. The constraint 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 MULTI_GLOBAL_CONTIGUITY(*Colour*) constraint [4], which itself generalises the GLOBAL_CONTIGUITY(*Colour*) constraint of [13]: the latter constrains only one colour (value 1) but the former constrains several colours, and both lack the decision variable N and hence *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). A propagator for GLOBAL_CONTIGUITY is given in [13], and a decomposition based on the AUTOMATON constraint is given in [4]: we generalise these ideas below and also give violation and differentiation functions.

The trajectory-based convexity of an airspace sectorisation is achieved by posting for every flight a 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 $Colour(v)$ is $\{1, 2, \ldots, s\}$.

Such a trajectory-based convexity constraint is called the CONTIGUITY constraint in our prior work on airspace sectorisation under systematic search [11], but lacks the decision variable N and hence *RelOp*; further, the propagator outlined there prunes less strongly than the one describe below (as it lacks both steps 2).

Such a trajectory-based convexity constraint is a soft constraint in our prior work on airspace sectorisation under stochastic local search [12], but lacks the decision variable N and hence *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).

5.1.1 Violation and Differentiation Functions

The violation and differentiation 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.

Soft Constraint. If the 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 $\mathit{NCC}(c)$ 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*:

$$\mathit{NCC} = \sum_{c \in \mathit{Colours}} \mathit{NCC}(c) \tag{4}$$

We can now re-formalise the semantics (3): the CONNECTED$(G, \mathit{Colour}, \mathit{RelOp}, N)$ constraint is satisfied if and only if

$$\mathit{NCC}\ \mathit{RelOp}\ N \land \forall c \in \mathit{Colours} : \mathit{NCC}(c) \leq 1 \tag{5}$$

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

$$\text{violation}(\mathit{Colour}(v)) = \mathit{NCC}(\alpha(\mathit{Colour}(v))) - 1 \tag{6}$$

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:

$$\text{violation}(N) = 1 - [\mathit{NCC}\ \mathit{RelOp}\ \alpha(N)] \tag{7}$$

This variable violation is zero if $\mathit{NCC}\ \mathit{RelOp}\ \alpha(N)$ 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:

$$\text{violation} = \text{violation}(N) + \sum_{c \in \mathit{Colours}} \max(\mathit{NCC}(c) - 1,\ 0) \tag{8}$$

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

The impact on the constraint violation of a colour assignment move $\mathit{Colour}(v) := c$ is measured by the following *colour assignment delta* function:

$$\begin{aligned}
&\Delta(\mathit{Colour}(v) := c) = \\
&\textbf{let}\ p = [\forall w \in \mathrm{Adj}(v) : \alpha(\mathit{Colour}(w)) \neq c] \\
&\qquad m = [\forall w \in \mathrm{Adj}(v) : \alpha(\mathit{Colour}(w)) \neq \alpha(\mathit{Colour}(v))] \\
&\textbf{in}\ p - m + [\mathit{NCC}\ \mathit{RelOp}\ \alpha(N)] - [\mathit{NCC} + p - m\ \mathit{RelOp}\ \alpha(N)]
\end{aligned} \tag{9}$$

Indeed, only the violation of the unchanged N and the numbers of connected components of the old and new colour of vertex v need to be considered. On the one hand, the number of connected components of colour c increases by one if all vertices adjacent to

v do not currently have its new colour c, so that v forms a new connected component of colour c. On the other hand, the number of connected components of colour $\alpha(Colour(v))$ decreases by one if all vertices adjacent to v do not currently have that original colour of v, so that v was the last element of some connected component of colour $\alpha(Colour(v))$. To avoid increasing the number of connected components and increase the likelihood of decreasing their number, 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. An assignment move on vertex v can be differentially probed in time linear in the degree of v in G.

A colour swap move $Colour(v) :=: Colour(w)$, where vertices v and w exchange their colours, is the sequential composition of the two colour assignment moves $Colour(v) := \alpha(Colour(w))$ and $Colour(w) := \alpha(Colour(v))$. The *colour swap delta* is the sum of the deltas for these two moves (upon incrementally making the first move), and there is no asymptotically faster way to compute this delta, as the complexity of probing an assignment move does not depend on the number of vertices.

The additive impact on the constraint violation of a counter assignment move $N := n$ is measured by the following *counter assignment delta* function:

$$\Delta(N := n) = [NCC\ RelOp\ \alpha(N)] - [NCC\ RelOp\ n]$$

The violation increases by one if the number of connected components was but now is not in relation $RelOp$ with N. It decreases by one if the number of connected components now is but was not in relation $RelOp$ with N. An assignment move on N can be differentially probed in constant time.

To achieve *incrementality*, once a move has been picked and made, the internal data structures and the variable and constraint violations must be updated. From the colour assignment delta function (9), the following updating code for a colour assignment move $Colour(v) := c$ follows directly, where the new assignment α' is the old assignment α, except that $\alpha'(Colour(v)) = c$:

if $\forall w \in \text{Adj}(v) : \alpha(Colour(w)) \neq c$ **then** $\{v$ forms a new CC of colour $c\}$
    $NCC(c) := NCC(c) + 1$
    $NCC := NCC + 1$
    violation $:=$ violation $+ 1$
if $\forall w \in \text{Adj}(v) : \alpha(Colour(w)) \neq \alpha(Colour(v))$ **then** $\{v$ formed a CC of $\alpha(Colour(v))\}$
    $NCC(\alpha(Colour(v))) := NCC(\alpha(Colour(v))) - 1$
    $NCC := NCC - 1$
    violation $:=$ violation $- 1$
for all $v \in V$ **do**
    violation$(Colour(v)) := \alpha'(NCC(Colour(v))) - 1$
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 (4), and the violation functions, such as (6) to (8), are defined as *invariants* [21], so that the solver automatically updates these quantities incrementally, without the constraint designer having to write explicit code, such as lines 3, 4, and 7 to 11.

Hard Constraint. If the CONNECTED constraint is considered implicitly, as in our [12], 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 $N := n$.

5.1.2 Propagator

One Dimension. If G is induced by a one-dimensional space, then propagation goes as follows.

If a vertex v is given a colour c (by obtaining $\text{dom}(\mathit{Colour}(v)) = \{c\}$ in the current state through propagation of either another constraint or a search decision), then:

- Let ℓ be the rightmost vertex, if any, *to the left* of v where $c \notin \text{dom}(\mathit{Colour}(\ell))$.
- Let r be the leftmost vertex, if any, *to the right* of v where $c \notin \text{dom}(\mathit{Colour}(r))$.
- Let ℓ' be the rightmost vertex with $\ell' \preceq v$ where $\text{dom}(\mathit{Colour}(\ell')) = \{c\}$.
- Let r' be the leftmost vertex with $v \preceq r'$ where $\text{dom}(\mathit{Colour}(r')) = \{c\}$.

All occurrences of colour c must be strictly between vertices ℓ and r. All vertices from ℓ' to r' must get colour c. Hence we prune as follows:

- If ℓ exists, then:
  1. Prune value c from the domain of every vertex to the left of ℓ.
  2. If $\text{dom}(\mathit{Colour}(\ell)) = \{d\}$, then prune value d from the domain of every vertex to the right of v.
- If r exists, then:
  1. Prune value c from the domain of every vertex to the right of r.
  2. If $\text{dom}(\mathit{Colour}(r)) = \{d\}$, then prune value d from the domain of every vertex to the left of v.
- Set $\text{dom}(\mathit{Colour}(i)) = \{c\}$ for every vertex i with $\ell' \prec i \prec r'$.

To propagate on N, it is best for the propagator to have an internal data structure: see the multi-dimensional case for details.

This propagator is only worth invoking when the domain of one of its decision variables shrinks to a singleton. It achieves domain consistency. No propagation is possible when the domain of a decision variable loses a value without becoming a singleton.

Arbitrary Number of Dimensions. If G is induced by a space of more than one dimension, then the notions of 'left' and 'right' have to be generalised. We assume that the propagator initialises (at its first invocation) and updates (at subsequent invocations) the following internal data structures, which are strongly related to those we use in Section 5.1.1 for stochastic local search:[1]

- The induced graph *ColourGraph* (which we define but do not store for stochastic local search).
- Let $NCC(c)$ denote the number of connected components of *ColourGraph* whose vertices currently have colour c in their domains, with $c \in Colours$.
- Let NCC denote the current sum of all $NCC(c)$, as in formula (4).

The semantics (5) of the constraint gives us a constraint checker and a feasibility test.

Upon pruning of values (by propagation of either another constraint or a search decision) from the domain of a colour decision variable $Colour(v)$, we (possibly incrementally) update the internal data structures and use the graph invariants of [5] in order to obtain possible pruning on other colour decision variables if not the counter decision variable N. We conjecture that this achieves domain consistency, like in the one-dimensional case.

Upon pruning of values from the domain of the counter decision variable N, we presently do not know whether pruning is possible on the colour decision variables.

Discussion. For many choices of *RelOp* and N, the CONNECTED constraint is easy to satisfy, namely by colouring all vertices with the same colour, so that there is only one connected component. In other words, the density of solutions to the CONNECTED constraint may be very high within the Cartesian product of the domains of its decision variables. Such a situation is not very conducive to propagation, as discussed next.

Starting from full colour domains for all vertices, there potentially is only one connected component until the underlying space is cut into at least two sub-spaces by colouring an entire swathe of vertices in some colour that has already been eliminated for other vertices. Only a very specific search procedure would achieve this only situation that is conducive to domain pruning. While this situation often arises naturally in a one-dimensional space, it does not do so in a multi-dimensional space and systematic search with interleaved propagation essentially degenerates into generate-and-test search, because the propagator of CONNECTED can only be invoked near the leaves of the search tree.

In conclusion, we are pessimistic about the utility of a propagator for the CONNECTED constraint in a multi-dimensional space, at least when following the graph colouring approach. Unless a representation more conducive to propagation can be found, we advocate stochastic local search over systematic search in the presence of this constraint.

[1]The strong relationship between internal datastructures needed for the violation and differentiation functions of stochastic local search and internal data structures needed for propagators of systematic search was not obvious to the authors at the outset of this research: this issue is worth investigating further, as code generation might be possible.

5.2 Compactness

Consider a graph $G = \langle V, E \rangle$ induced by a space of at least two dimensions. The $\textsc{Compact}(G, \mathit{Colour}, t)$ constraint holds if and only if the sum of the sphericity discrepancies of the connected components of the graph *ColourGraph* 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 4.1, 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

  $$\mathit{Colour}(w) \neq \mathit{Colour}(v)$$

  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 $G_\perp$ for some given background geometry G. That is, there is a unique special vertex in V, called $\perp$, 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 *Colour* (such as a connected component of the induced graph *ColourGraph*), denoted by A_W, is the sum of the surface areas of the border facets of the vertices in W:

  $$A_W = \sum_{v \in W} \sum_{\substack{w \in \mathrm{Adj}(v) \\ f \in \partial(v) \cap \partial(w) \\ w \notin W}} [\mathit{Colour}(w) \neq \mathit{Colour}(v)] \cdot \mathrm{Area}(v, f) \tag{10}$$

  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 *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 [22] for the derivation of this formula):

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

  where

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

  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 *Colour*, denoted by Ψ_W, is the difference between the border surface area of W under *Colour* and the sphere surface area of W under *Colour*:

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

  We derive this concept from the *sphericity* Ψ of a shape p, defined in [22] 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 *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 our prior work on airspace sectorisation under stochastic local search [12], but lacks the threshold t there; we generalise the ideas of its violation and differentiation functions using the concept of sphericity discrepancy, and we describe them in much more detail.

There was no compactness constraint in our prior work on airspace sectorisation using systematic search [11], and we are not aware of any published propagator for any such constraint.

5.2.1 Violation and Differentiation Functions

Soft Constraint. 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 *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 *Border*(v) is defined as follows:

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

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

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

- Let *CCs* denote the current set of connected components of the induced graph *ColourGraph*, each encoded by a tuple $\langle \sigma, \nu \rangle$, meaning that it currently has total surface area σ and total volume ν.

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

$$\mathrm{violation}(\mathit{Colour}(v)) = f(\mathit{Border}(v))$$

where the weight function f can be the identity function, but can also suitably penalise larger border surface areas, provided $f(0) = 0$; in [12], 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:

$$\mathrm{violation} = \max\left(\sum_{\langle\sigma,\nu\rangle\in \mathit{CCs}} \left(\sigma - \pi^{1/3}\cdot(6\cdot\nu)^{2/3}\right) - t,\ 0\right) \tag{11}$$

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

To define the additive impact $\Delta(\mathit{Colour}(v) \coloneqq c)$ on the constraint violation of an assignment move $\mathit{Colour}(v) \coloneqq c$, we first define the impact on the violation of a variable $\mathit{Colour}(w)$ for a vertex w that shares a facet f with v:

$$\begin{aligned}&\delta(\mathit{Colour}(w), \mathit{Colour}(v) \coloneqq c)\\ &= \begin{cases} -\mathrm{Area}(v,f) & \text{if } \alpha(\mathit{Colour}(v)) \neq \alpha(\mathit{Colour}(w)) = c \wedge w \in \mathrm{Adj}(v) \wedge f \in \partial(v)\cap\partial(w)\\ +0 & \text{if } \alpha(\mathit{Colour}(v)) \neq \alpha(\mathit{Colour}(w)) \neq c\\ & \vee\ \alpha(\mathit{Colour}(v)) = \alpha(\mathit{Colour}(w)) = c\\ +\mathrm{Area}(v,f) & \text{if } \alpha(\mathit{Colour}(v)) = \alpha(\mathit{Colour}(w)) \neq c \wedge w \in \mathrm{Adj}(v) \wedge f \in \partial(v)\cap\partial(w)\end{cases}\end{aligned}$$

This differential probing can be done in constant time.

The impact on the violation of $\mathit{Colour}(v)$ itself is the sum of the impacts on the violations of the variables corresponding to the vertices adjacent to v:

$$\delta(\mathit{Colour}(v), \mathit{Colour}(v) \coloneqq c) = \sum_{w\in \mathrm{Adj}(v)} \delta(\mathit{Colour}(w), \mathit{Colour}(v) \coloneqq c) \tag{12}$$

This differential probing can be done in time linear in the degree of v in G.

The impact $\Delta(\mathit{Colour}(v) \coloneqq c)$ on the constraint violation of an assignment move $\mathit{Colour}(v) \coloneqq c$ can unfortunately not be computed cheaply. The connected component containing v with its *original* colour $\alpha(\mathit{Colour}(v))$ may be split into several components after the move. Further, the connected component containing v with its *new* colour c may be the merger of several components existing prior to the move. The same holds for the connected components containing the vertices adjacent to v: some of them may merge after the move. Hence the only way to measure the *exact* impact using all the connected components is first to compute the new connected components, in $\Theta(V+E)$ time, as well as their surface area and volume attributes, and then to subtract the old violation from the new violation computed using (11). One may of course choose not to probe at all the impact on the violation of the COMPACT constraint during search. One may also be content with a much more cheaply computed *approximate* impact, namely $\delta(\mathit{Colour}(v), \mathit{Colour}(v) \coloneqq c)$, which as we saw can be computed in time linear in the degree of v in G.

If probing *is* used during search, the described exact probing in time linear in the size of G is considered too expensive, and the described fast approximate probing is deemed too risky (as it can be an under-approximation, which is dangerous [8]), then we propose another measure of constraint violation, which does not need the data structure *CCs* but leads to a coarser approximation of sphericity. This requires first changing the semantics of the constraint: the $\textsc{Compact}(G, \mathit{Colour}, t)$ constraint holds if and only if the sum of the border surface areas of all vertices is at most the threshold t, whose value is given. We then redefine the *violation of the constraint* to be the excess, if any, of the current total weighted border surface area of all vertices:

$$\text{violation} = \max\left(\frac{1}{2} \cdot \sum_{i \in V} \text{violation}(\mathit{Colour}(i)) - t,\ 0\right) \tag{13}$$

The factor $\frac{1}{2}$ compensates for every shared facet being counted twice. The constraint violation is zero if the current total weighted border surface area does not exceed t.

The impact $\Delta(\mathit{Colour}(v) \coloneqq c)$ on the constraint violation of an assignment move $\mathit{Colour}(v) \coloneqq c$ can now be computed cheaply. Let s be the current value of the expression $\frac{1}{2} \cdot \sum_{i \in V} \text{violation}(\mathit{Colour}(i)) - t$ inside (13). Since the violations of the variables corresponding to v and its adjacent vertices are the only terms that potentially change in the evaluation of s, that expression changes due to (12) by $\delta(\mathit{Colour}(v), \mathit{Colour}(v) \coloneqq c)$ only. We get the following *assignment delta* function:

$$\Delta(\mathit{Colour}(v) \coloneqq c) = \max(s + \delta(\mathit{Colour}(v), \mathit{Colour}(v) \coloneqq c),\ 0) - \max(s,\ 0)$$

An assignment move on vertex v can be differentially probed in time linear in the degree of v in G.

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.

A swap move $\mathit{Colour}(v) \coloneqq\!: \mathit{Colour}(w)$, where vertices v and w exchange their colours, is the sequential composition of the two assignment moves $\mathit{Colour}(v) \coloneqq \alpha(\mathit{Colour}(w))$ and $\mathit{Colour}(w) \coloneqq \alpha(\mathit{Colour}(v))$. The *swap delta* is the sum of the assignment deltas for these two moves (upon incrementally making the first move), and there is no asymptotically faster way to compute this delta, as the complexity of probing an assignment move does not depend on the number of vertices.

We omit the rather clerical code for achieving incrementality.

Hard Constraint. It is very difficult to consider the $\textsc{Compact}$ constraint implicitly, as it is not obvious how to satisfy it cheaply in the start assignment and under what probed moves to maintain it satisfied.

5.2.2 Propagator

Even more so than a propagator for the $\textsc{Connected}$ constraint (see Section 5.1.2), a propagator for the $\textsc{Compact}$ constraint will not be able to perform much domain pruning until most of the decision variables have singleton domains. Indeed, there is no useful particular case of $\textsc{Compact}$ for one-dimensional spaces (unlike for $\textsc{Connected}$, where in a one-dimensional space connectedness reduces to stretch contiguity, for which significant pruning is possible, even to domain consistency), and for COMPACT there can be several connected components per colour (unlike for CONNECTED, where every colour has at most one connected component). In fact, if the threshold t is not exceeded by the sphericity discrepancy of the entire underlying space, then the COMPACT constraint is easy to satisfy (like CONNECTED), namely by colouring all vertices with the same colour, so that there is only one connected component. For the implications of this insight, see the discussion of propagation of the CONNECTED constraint in Section 5.1.2.

5.3 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 STRETCHSUM(G, *Colour*, *Value*, *RelOp*, t) constraint, with $\mathit{RelOp} \in \{\leq, <, =, \neq, >, \geq\}$, holds if and only if every stretch of the sequence *Colour* corresponds to a subsequence of *Value* whose sum is in relation *RelOp* with threshold t, whose value is given. Formally:

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

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 STRETCHSUM(G, *Colour*, *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:

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

The STRETCHSUM constraint is a soft constraint in our prior work on airspace sectorisation under stochastic local search [12], 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).

The STRETCHSUM constraint is called the SLIDINGSUM constraint in our prior work on airspace sectorisation under systematic search [11], but the propagator outlined there is very different from the one we describe below, as it is only worth invoking when the domain of one of its decision variables shrinks to a singleton.

5.3.1 Violation and Differentiation Functions

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

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

We say that a colour stretch with value sum σ is a *violating stretch* if σ is smaller than the threshold t:
$$\sigma \not\geq t$$

The *violation of a decision variable*, say $\mathit{Colour}(v)$ for vertex v with $\mathit{Stretch}(v) = \langle \ell, r, c, \sigma \rangle$, where $\alpha(\mathit{Colour}(v)) = c$, is defined as follows:
$$\text{violation}(\mathit{Colour}(v)) = \begin{cases} 0 & \text{if } \ v \notin \{\ell, r\} \\ 0 & \text{if } \ v \in \{\ell, r\} \wedge \sigma \geq t \wedge \sigma - \mathit{Value}(v) \not\geq t \\ \mathit{Value}(v) & \text{if } \ v \in \{\ell, r\} \wedge \sigma \geq t \wedge \sigma - \mathit{Value}(v) \geq t \\ 1 & \text{if } \ v \in \{\ell, r\} \wedge \sigma \not\geq t \end{cases}$$

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 $\mathit{Value}(v)$ 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:
$$\text{violation} = \sum_{\langle \_,\_,\_,\sigma \rangle \in \mathit{Stretch}} [\sigma \not\geq t]$$

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

Let us now measure the additive impact on the constraint violation of an assignment move $\mathit{Colour}(v) := d$, with $\mathit{Stretch}(v) = \langle \ell, r, c, \sigma \rangle$. For simplicity of notation, we assume $d \neq c$. Let the colour stretch to the left be $\langle \ell', r', c', \sigma' \rangle$, with $\text{pred}(\ell) = r'$ if it exists, and $\langle \bot, \bot, \gamma, +\infty \rangle$ otherwise, with $\gamma \notin \mathit{Colours}$. Similarly, let the colour stretch to the right be $\langle \ell'', r'', c'', \sigma'' \rangle$, with $\text{succ}(r) = \ell''$ if it exists, and $\langle \bot, \bot, \gamma, +\infty \rangle$ otherwise, with $\gamma \notin \mathit{Colours}$. The *assignment delta* function is defined as follows:

$$\Delta(\mathit{Colour}(v) := d)$$
$$= \begin{cases} \begin{array}{l} -\ [\sigma' \not\geq t \wedge [c' = d] \cdot \sigma' + \sigma + [c'' = d] \cdot \sigma'' \geq t] \\ -\ [\sigma \not\geq t] \\ -\ [\sigma'' \not\geq t \wedge [c' = d] \cdot \sigma' + \sigma + [c'' = d] \cdot \sigma'' \geq t] \end{array} & \text{if } \ \ell = v = r \\ [\sigma \geq t \wedge \sigma - \mathit{Value}(v) \not\geq t] - [\sigma' \not\geq t \wedge \sigma' + \mathit{Value}(v) \geq t] & \text{if } \ \ell = v \prec r \wedge d = c' \\ [\mathit{Value}(v) \not\geq t] + [\sigma \geq t \wedge \sigma - \mathit{Value}(v) \not\geq t] & \text{if } \ \ell = v \prec r \wedge d \neq c' \\ \text{(analogous to the two previous cases)} & \text{if } \ \ell \prec v = r \\ 2 & \text{if } \ \ell \prec v \prec r \wedge \sigma \not\geq t \\ \left[\sum_{\ell \preceq i \prec v} \mathit{Value}(i) \not\geq t \right] + [\mathit{Value}(v) \not\geq t] + \left[\sum_{v \prec i \preceq r} \mathit{Value}(i) \not\geq t \right] & \text{if } \ \ell \prec v \prec r \wedge \sigma \geq t \end{cases}$$

An assignment move on vertex v can be differentially probed in time at worst linear in the number $|V|$ of vertices. This is so only in the last case and when there currently is only one colour stretch. In all other cases, differential probing takes constant time.

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.

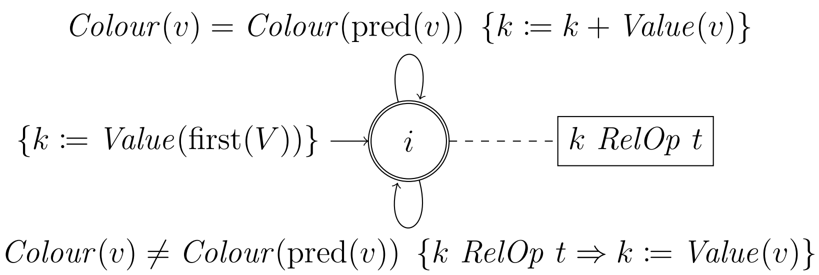
A swap move $\mathit{Colour}(v) :=: \mathit{Colour}(w)$, where vertices v and w exchange their colours, is the sequential composition of the two assignment moves $\mathit{Colour}(v) := \alpha(\mathit{Colour}(w))$ and $\mathit{Colour}(w) := \alpha(\mathit{Colour}(v))$. The *swap delta* is the sum of the assignment deltas for these two moves (upon incrementally making the first move), and there is no asymptotically faster way to compute this delta, as at most one of these two moves has the worst-case probing complexity that depends on the number of vertices.

We omit the rather clerical code for achieving incrementality.

Hard Constraint. If the StretchSum constraint is considered implicitly, then it can be satisfied cheaply in the start assignment, by greedily colouring the vertices from left to right so that every stretch has an adequate sum. However, it may be impossible to maintain this constraint as satisfied upon every move, even when only considering moves that re-colour one or more vertices at the border of a stretch to the colour of an adjacent stretch.

5.3.2 Propagator

A checker for the StretchSum$(G, \mathit{Colour}, \mathit{Value}, \mathit{RelOp}, t)$ constraint can be elegantly modelled using the following deterministic finite automaton (DFA) with a counter and predicates on the transitions:



There is a unique state, called i: it is the start state (as marked by a transition coming in from nowhere) and an accepting state (as marked by the double circle). There is a unique counter, called k: it is initialised on the start transition to the value of the leftmost vertex; without loss of generality, we assume $V \neq \varnothing$. The transitions are made on the truth values of predicates on a current vertex v and its predecessor. The counter evolves on each such transition, unconditionally in the top transition (because v is in the colour stretch of its predecessor) and conditionally in the bottom transition (because v starts a new colour stretch). If a conditional transition fails, then a transition is made to an implicit failure state, which has only self-looping transitions. Upon processing the rightmost vertex, the accepting state is actually accepting only if the attached *acceptance condition* (drawn in a box connected to it by a dashed line) is satisfied: it ensures that the last stretch has an adequate sum.

Such automata were introduced in [3]. Assume the following constraints, called *signature constraints*, for new Boolean decision variables $\mathit{Sig}(v)$, called *signature variables*, with initial domain $\{0, 1\}$:

$$\forall v \in V \setminus \{\text{first}(V)\} : \mathit{Colour}(v) = \mathit{Colour}(\text{pred}(v)) \Leftrightarrow \mathit{Sig}(v) = 1$$

We can feed these $|V \setminus \{\text{first}(V)\}|$ signature variables into a DFA with a counter, just like the one above, except that the top predicate is replaced by alphabet symbol 1, and the bottom one by 0. It is shown in [3] how to prototype rapidly a propagator from such a constraint checker using the AUTOMATON constraint. In this case, it probably does not achieve domain consistency, but one can try and boost its propagation using our ideas in [7].

5.4 Balanced Workload

The BALANCED(G, *Colour*, *Value*, μ, Δ) 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 μ as average and having discrepancies to μ that do not exceed the (possibly unknown) integer threshold Δ. If Δ is not given, then it may appear in the objective function, towards being minimised. Formally, the constraint can be decomposed into the following conjunction:

$$\forall i \in \mathit{Colours} : X[i] = \sum_{v \in V} ([\mathit{Colour}(v) = i] \cdot \mathit{Value}(v)) \wedge \Gamma(X, \mu, \Delta) \tag{14}$$

where constraint Γ is either SPREAD or DEVIATION, thereby giving a concrete definition to the used abstract concept of *discrepancy*:

- The SPREAD(X, Δ, μ) constraint [16] holds if and only if the n integer variables $X[i]$ have the (possibly unknown, and not necessarily integer) value μ 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 Δ.

- The DEVIATION(X, Δ, μ) constraint [19] holds if and only if the n integer variables $X[i]$ have the (possibly unknown, and not necessarily integer) value μ as average and the sum of the deviations $|n \cdot X[i] - n \cdot \mu|$ does not exceed the (possibly unknown) integer threshold Δ.

The multiplications by n in the definitions of discrepancy lift all reasoning to integer domains even when μ is not an integer, as $\sum_{i=1}^{n} X[i] = n \cdot \mu$ and the $X[i]$ are integer variables: the integer threshold Δ has to be calibrated accordingly. One could also use the RANGE(X, *RelOp*, Δ) constraint [4], which holds if and only if $\max(X) + 1 - \min(X)$ *RelOp* Δ, 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. 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 μ is known when the number n of sectors (and thus colours) is imposed:

$$\mu = \frac{\sum_{v \in V} \text{Workload}(v)}{n} \tag{15}$$

The workload balancing constraint is a soft constraint in our prior work on airspace sectorisation under stochastic local search [12], but its concept of discrepancy is defined in terms of a *ratio* rather than a difference with the average μ, 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.

A workload balancing constraint is also used in our prior work on airspace sectorisation under systematic search [11], but again under a different concept of discrepancy with the average μ, namely $1 - \Delta \leq X[i]/\mu \leq 1 + \Delta$ for each i (in the experiments, $\Delta = 0.25$ was used). The constraint is enforced not by a propagator, but by setting the domains of the $X[i]$ integer decision variables to $\{\lceil (1-\Delta) \cdot \mu \rceil, \dots, \lfloor (1+\Delta) \cdot \mu \rfloor\}$, which is possible when Δ is given as μ is a known constant by (15). This modelling trick cannot be applied with the definitions of discrepancy for Spread and Deviation.

5.4.1 Violation and Differentiation 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 $\mathit{Colours} = \{1, 2, \dots, n\}$.

Soft Constraint. If the Balanced$(G, \mathit{Colour}, \mathit{Value}, \mu, \Delta)$ constraint is considered explicitly, then we proceed as follows. For simplicity of notation, we assume Δ and μ are given, and that μ 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 Δ by n^2, giving the following simplified semantics of Balanced:

$$\sum_{i=1}^{n} X[i] = n \cdot \mu \tag{16}$$

and

$$\sum_{i=1}^{n} |X[i] - \mu| \leq \Delta \tag{17}$$

where

$$\forall i \in \{1, \dots, n\} : X[i] = \sum_{v \in V} ([\mathit{Colour}(v) = i] \cdot \mathit{Value}(v)) \tag{18}$$

Note that (18) implies

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

so that, using (16), we must have

$$\mu = \frac{\sum_{v \in V} \mathit{Value}(v)}{n} \tag{19}$$

similarly to (15). From now on, we assume the given *Value* and μ satisfy (19), so that we need not reason about (16), as it is then surely satisfied, because implied when (18) is satisfied. Formula (18) itself defines the auxiliary decision variables $X[i]$, so it suffices to set it up as a set of invariants [21]. In conclusion, we only need to deal with formula (17).

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 (18). This violation is the current deviation from μ 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 μ, and thus contributes nothing to the total deviation for all colours.

The *violation of the constraint* is the excess, if any, over Δ of the sum of the current deviations from μ 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 μ of the value sum for colour i. Hence the constraint violation is defined as follows:

$$\text{violation} = \max\left(\sum_{i=1}^{n} \text{violation}(X[i]) - \Delta,\ 0\right) \tag{20}$$

The constraint violation is zero if the total deviation for all colours currently does not exceed Δ.

The impact on the constraint violation of an assignment move $\mathit{Colour}(v) := c$ is measured as follows. We assume $\alpha(\mathit{Colour}(v)) = d$. Let s be the current value of the first argument of the maximisation expression in (20), and let s' be the same value except for colours c and d:

$$s' = \sum_{\substack{i=1 \\ i \notin \{c,d\}}}^{n} \text{violation}(X[i]) - \Delta = s - |\alpha(X[c]) - \mu| - |\alpha(X[d]) - \mu|$$

We get the following *assignment delta* function:

$$\begin{aligned}&\Delta(\mathit{Colour}(v) := c)\\ &= \max(s' + |\alpha(X[c]) + \mathit{Value}(v) - \mu| + |\alpha(X[d]) - \mathit{Value}(v) - \mu|,\ 0) - \max(s,\ 0)\end{aligned}$$

An assignment move can be differentially probed in constant time.

A swap move $\mathit{Colour}(v) :=: \mathit{Colour}(w)$, where vertices v and w exchange their colours, is the sequential composition of the two assignment moves $\mathit{Colour}(v) := \alpha(\mathit{Colour}(w))$ and $\mathit{Colour}(w) := \alpha(\mathit{Colour}(v))$. The *swap delta* is the sum of the assignment deltas for these two moves (upon incrementally making the first move), and there is no asymptotically faster way to compute this delta, as probing an assignment move takes constant time.

We omit the rather clerical code for achieving incrementality.

Hard Constraint. It is very difficult to consider the BALANCED constraint implicitly, as it is not obvious how to satisfy it cheaply in the start assignment and under what probed moves to maintain it satisfied.

5.4.2 Propagator

Due to the decomposition (14) of the BALANCED constraint using either the SPREAD or the DEVIATION constraint, we refer to [16, 19] for propagators for these constraints. Bounds consistency can be achieved in $\mathcal{O}(n \cdot \log n)$ time and $\mathcal{O}(n)$ time, respectively. The remaining part of (14), namely

$$\forall i \in \mathit{Colours} : X[i] = \sum_{v \in V} ([\mathit{Colour}(v) = i] \cdot \mathit{Value}(v))$$

can be further decomposed as follows, using reification and a 2D matrix of new Boolean decision variables $B[i, v]$:

$$\begin{aligned} &\forall i \in \mathit{Colours} : \forall v \in V : \mathit{Colour}(v) = i \Leftrightarrow B[i, v] = 1 \\ &\wedge \forall i \in \mathit{Colours} : \textsc{Linear}(\mathit{Value}, B[i, *], =, X[i]) \end{aligned}$$

The propagation of reified constraints and the LINEAR constraint are standard features of every constraint programming solver. We doubt the fixpoint of these propagators achieves domain consistency, but the development of a custom propagator for BALANCED along the lines of our [15] is future work.

5.5 Bounded Workload

The BOUNDED$(G, \mathit{Colour}, \mathit{Value}, \mathit{RelOp}, t)$ constraint, with $\mathit{RelOp} \in \{\leq, <, =, \neq, >, \geq\}$, holds if and only if every sum of the given integer values under *Value* of the vertices having the same colour under *Colour* is in relation *RelOp* with threshold t, whose value is given. Formally, the constraint can be decomposed into the following conjunction:

$$\forall i \in \mathit{Colours} : \sum_{v \in V} ([\mathit{Colour}(v) = i] \cdot \mathit{Value}(v)) \ \mathit{RelOp} \ t$$

This can be further decomposed as follows, using reification and a 2D matrix of new Boolean decision variables $B[i, v]$:

$$\begin{aligned} &\forall i \in \mathit{Colours} : \forall v \in V : \mathit{Colour}(v) = i \Leftrightarrow B[i, v] = 1 \\ &\wedge \forall i \in \mathit{Colours} : \textsc{Linear}(\mathit{Value}, B[i, *], \mathit{RelOp}, t) \end{aligned}$$

The handling of reified constraints and the LINEAR constraint are standard features of every constraint programming solver, so we need not develop any new propagators or violation and differentiation functions. We doubt the fixpoint of these propagators achieves domain consistency, but the development of a custom propagator for BOUNDED along the lines of our [15] is future work.

In airspace sectorisation, the workload of each sector must not exceed some upper bound. Hence one would take *Value* as the Workload function of Section 4, and *RelOp* as $\leq$. Recall that we only consider additive workloads, such as monitoring workload and conflict workload, but no non-additive workloads, such as coordination workload.

We did not consider a bounded-workload constraint in our prior work on airspace sectorisation under stochastic local search [12] and systematic search [11], but such a constraint has been considered by others, as discussed in our survey [6].

5.6 Balanced Size

In airspace sectorisation, the size of each sector must be within some given imbalance factor of the average across all sectors. Hence one can use the BALANCED constraint of Section 5.4, with *Value* as the Volume function of Section 4.

We did not consider a balanced-size constraint in our prior work on airspace sectorisation under stochastic local search [12] and systematic search [11], but such a constraint has been considered by others, as discussed in our survey [6].

5.7 Minimum Distance: No Border Vertices in Stretches

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

$$\begin{gathered}\forall \ell \preceq r \in P \setminus [\perp] : \mathit{Stretch}(\mathit{Colour}(P), \ell, r) \\ \Rightarrow \forall \ell \preceq v \preceq r \in P : \forall w \in \mathrm{Adj}(v) \setminus P : \mathit{Colour}(w) = \mathit{Colour}(v)\end{gathered}$$

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:

$$P = [\perp] \cup [v_i \mid \langle v_i, \_, \_ \rangle \in \mathit{Plan}(f)] \cup [\perp]$$

In our prior work on airspace sectorisation under systematic search [11], we did not need such a minimum-distance constraint, because any regions that cannot serve as border regions of sectors were pre-aggregated into AFBs. In our prior work on airspace sectorisation under stochastic local search [12], we initially used the same pre-aggregation into AFBs in order to avoid having such a minimum-distance constraint, but it then turned out that the COMPACT constraint (see Section 5.2) is very hard to satisfy when AFBs are used, so we had concluded that a minimum-distance constraint would be necessary after all, upon switching off the pre-aggregation into AFBs, but we did not design such a constraint.

5.7.1 Violation and Differentiation Functions

Soft Constraint. 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:

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

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.

The additive impact on the constraint violation of an assignment move $Colour(v) := c$ for vertex $v \in P$ is measured by the following *assignment delta* function:

$$\Delta(Colour(v) := c) = \sum_{w \in \text{Adj}(v) \setminus P} ([\alpha(Colour(w)) \neq c] - [\alpha(Colour(w)) \neq \alpha(Colour(v))])$$

An assignment move on vertex v can be differentially probed in time linear in the degree of v in G.

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.

A swap move $Colour(v) :=: Colour(w)$, where vertices v and w exchange their colours, is the sequential composition of the two assignment moves $Colour(v) := \alpha(Colour(w))$ and $Colour(w) := \alpha(Colour(v))$. The *swap delta* is the sum of the assignment deltas for these two moves (upon incrementally making the first move), and there is no asymptotically faster way to compute this delta, as the complexity of probing an assignment move does not depend on the number of vertices.

We omit the rather clerical code for achieving incrementality.

Hard Constraint. It is very difficult to consider the NonBorder constraint implicitly, as it is not obvious how to satisfy it cheaply in the start assignment and under what probed moves to maintain it satisfied.

5.7.2 Propagator

Designing a propagator for the NonBorder constraint is left as future work. Note that a propagator cannot even be prototyped using the Automaton constraint, as the constraint is not only on the $Colour(P)$ sequence of decision variables, but also on several $Colour(\text{Adj}(i))$ sequences of decision variables, the set of such vertices i not being known up front. A different kind of automaton (possibly with counters) would have to be imagined, and supported with a new constraint like Automaton, in order to get a propagator for NonBorder.

6 Conclusion

Airspace sectorisation provides a partition of a given airspace into sectors, subject to geometric constraints and workload constraints, so that some cost metric is minimised. We have studied the constraints that arise in airspace sectorisation, giving for each constraint an analysis of what algorithms and properties are required under systematic search and stochastic local search.

6.1 Discussion

The formal semantics or decompositions of several constraints exhibit a pattern that we had not encountered before: the decision variables of the problem functionally determine auxiliary decision variables, which are the arguments of well-known constraints. For example, the auxiliary variables $X[i]$ of the decomposition (14) of the BALANCED constraint in Section 5.4 are functionally determined by the $Colour(v)$ variables, but it is the $X[i]$ that are constrained by the well-known SPREAD or DEVIATION constraints. This level of indirection makes propagation difficult, but apparently often occurs in real-life problems: overcoming this is a major research challenge for all optimisation technologies, and thus beyond the scope of this report.

We were surprised how little propagation is possible for some airspace sectorisation constraints, especially the COMPACT and CONNECTED constraints: this explains why our prior work using systematic search [11] was not as successful as we had hoped. In retrospect, we advocate achieving sector compactness by using also a constraint on *maximum* dwell time, that is STRETCHSUM($Colour$, $Value$, $\leq, t$) for some suitable upper bound t on dwell time. We believe this would trigger more propagation before most of the decision variables have singleton domains (see Section 5.2.2). However, we do not know whether airspace sectorisation experts would be willing to quantify the upper bound t.

We have noticed that the violation and delta functions for stochastic local search are sometimes hard to design in a problem-independent fashion, towards making the constraints highly reusable. Indeed, one can often imagine several pairs of measures for variable and constraint violation (if not constraint semantics), each pair needing a rather different mathematical and algorithmic apparatus for incremental maintenance and differential probing, but the appropriate pair may depend on the actual problem or on the neighbourhood chosen for search. It thus seems that some constraints need to be offered in several incarnations, as witnessed with the COMPACT constraint and with other choices discussed in the literature we have pointed to: there is no reason to believe that one set of choices dominates all others. The same phenomenon occurs with propagators for systematic search, where different levels of consistency can be aimed at, and where different time and space complexity trade-offs exist even for propagators achieving the same level of consistency.

6.2 Related Work

For each constraint in Section 5, we have given a discussion of all similar constraints that we are aware of, including literature pointers.

6.3 Future Work

Coordination workload is not additive when aggregating the workload of a sector from the workloads of its constituent regions. We have only handled the monitoring and conflict workloads in this report, because they are additive and thus easier to reason with, and we have assumed they are combined into a single value. This work needs to be extended to coordination workloads, and possibly to a separate handling of all three kinds of workload.

For time reasons, we have not tackled all the constraints arising in airspace sectorisation (identified in Section 2), so the few remaining constraints need to be discussed in equal depth.

The constraints should also be tackled under the set covering approach, as we have assumed here the graph colouring approach for all the covered constraints. The representations of the two approaches are dual (in the sense that the set covering approach represents the inverse of the total function, of the graph colouring approach, from regions to sectors). It may turn out to be beneficial to switch the representation for *some* constraints, if not *always* to use both representations at the same time, and to channel between them.

The completion of our line of work, based on constraint programming as a combinatorial problem solving technology, will show that *all* the constraints can be used in the process of *computing* a sectorisation, rather than only using some and then evaluating the *results* of a sectorisation algorithm according to the other constraints.

References

[1] A. Aggoun and N. Beldiceanu. Extending CHIP in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[2] C. Allignol, N. Barnier, P. Flener, and J. Pearson. Constraint programming for air traffic management: A survey. *The Knowledge Engineering Review*, 27(3):361–392, September 2012.

[3] N. Beldiceanu, M. Carlsson, and T. Petit. Deriving filtering algorithms from constraint checkers. In M. Wallace, editor, *CP 2004, the 10th International Conference on Principles and Practice of Constraint Programming*, volume 3258 of *LNCS*, pages 107–122. Springer-Verlag, 2004.

[4] N. Beldiceanu, M. Carlsson, and J.-X. Rampon. Global constraint catalogue: Past, present, and future. *Constraints*, 12(1):21–62, March 2007. The current working version of the catalogue is at `http://www.emn.fr/z-info/sdemasse/aux/doc/catalog.pdf`.

[5] N. Beldiceanu, T. Petit, and G. Rochart. Bounds of graph parameters for global constraints. *RAIRO – Operations Research*, 40(4):327–353, October–December 2006.

[6] P. Flener and J. Pearson. Automatic airspace sectorisation: A survey. Technical Report 1311.0653, Computing Research Repository, November 2013. Available at `http://arxiv.org/abs/1311.0653`.

[7] M. A. Francisco Rodríguez, P. Flener, and J. Pearson. Generation of implied constraints for automaton-induced decompositions. In E. Grégoire and B. Mazure, editors, *ICTAI/CSP 2013, the special track on SAT and CSP technologies of the 25th IEEE International Conference on Tools with Artificial Intelligence*. IEEE Computer Society, 2013.

[8] J. He, P. Flener, and J. Pearson. Underestimating the cost of a soft constraint is dangerous: Revisiting the edit-distance based SoftRegular constraint. *Journal of Heuristics*, 19(5):729–756, October 2013.

[9] J. N. Hooker. *Integrated Methods for Optimization*. Springer-Verlag, second edition, 2011.

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

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

[12] P. Jägare, P. Flener, and J. Pearson. Airspace sectorisation using constraint-based local search. In S. Saunders-Hodge and C. Meckiff, editors, *ATM 2013, the 10th USA / Europe Seminar on Air Traffic Management R&D, Chicago, Illinois, USA*, 2013. Best paper of theme: Network and Strategic Traffic Flow Optimization.

[13] M. J. Maher. Analysis of a global contiguity constraint. In *Workshop on Rule-Based Constraint Reasoning and Programming, held at CP 2002, the 8th International Conference on Principles and Practice of Constraint Programming*, 2002.

[14] J. May. *A Concise Course in Algebraic Topology*. Chicago Lectures in Mathematics. University of Chicago Press, 1999.

[15] J.-N. Monette, N. Beldiceanu, P. Flener, and J. Pearson. A parametric propagator for discretely convex pairs of sum constraints. In C. Schulte, editor, *CP 2013, the 19th International Conference on Principles and Practice of Constraint Programming*, volume 8124 of *LNCS*, pages 529–544. Springer, 2013.

[16] G. Pesant and J.-C. Régin. SPREAD: A balancing constraint based on statistics. In P. van Beek, editor, *CP 2005, the 11th International Conference on Principles and Practice of Constraint Programming*, volume 3709 of *LNCS*, pages 460–474. Springer, 2005.

[17] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In B. Hayes-Roth and R. E. Korf, editors, *AAAI 1994, the 12th (US) National Conference on Artificial Intelligence*, pages 362–367. AAAI Press, 1994.

[18] J.-C. Régin. Generalized arc-consistency for global cardinality constraint. In D. Weld and B. Clancey, editors, *AAAI 1996, the 13th (US) National Conference on Artificial Intelligence*, pages 209–215. AAAI Press, 1996.

[19] P. Schaus, Y. Deville, and P. Dupont. Bound-consistent deviation constraint. In C. Bessière, editor, *CP 2007, the 13th International Conference on Principles and Practice of Constraint Programming*, volume 4741 of *LNCS*, pages 620–634. Springer, 2007.

[20] P. Van Hentenryck and J.-P. Carillon. Generality versus specificity: An experience with AI and OR techniques. In T. Mitchell and R. Smith, editors, *AAAI 1988, the 7th (US) National Conference on Artificial Intelligence*, pages 660–664. AAAI Press, 1988.

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

[22] H. Wadell. Volume, shape, and roundness of quartz particles. *The Journal of Geology*, 43(3):250–280, April/May 1935.