IT 11 021
Examensarbete 30 hp
Maj 2011

# Airspace Sectorisation Using Constraint Programming

Peter Jägare

Institutionen för informationsteknologi
*Department of Information Technology*

UPPSALA
UNIVERSITET

**Teknisk- naturvetenskaplig fakultet**
**UTH-enheten**

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

# Abstract

## Airspace Sectorisation Using Constraint Programming

*Peter Jägare*

Given a set of cells and a set of flight routes passing through these cells, we need to cluster cells into a given number of sectors, ensuring an even workload over all sectors, and fulfilling several other constraints on the wellformedness of sectors. The sectorisation is done by using constraint programming. Several propagators are designed to ensure the correctness of the sectorisation.

Handledare: Pierre Flener
Ämnesgranskare: Justin Pearson
Examinator: Anders Jansson
IT 11 021
Sponsor: European Organisation for the Safety of Air Navigation (EUROCONTROL)

Tryckt av: Reprocentralen ITC

Table of Contents


| | |
|---|---|
| Abstract | 1 |
| Table of Contents | 2 |
| Acknowledgements | 3 |
| Background | 4 |
| Air Traffic Management and Airspace Sectorisation | 4 |
| Constraint Programming | 4 |
| Methodology | 6 |
| Data | 6 |
| Problem description | 6 |
| Preprocessing | 7 |
| Model | 7 |
| The contiguity propagator | 10 |
| The sliding sum propagator | 11 |
| The number of subsequences propagator | 12 |
| Search | 13 |
| Symmetry breaking | 13 |
| Postprocessing | 13 |
| Initial results | 14 |
| Toward Sector Compactness | 17 |
| Conclusions and further work | 21 |
| Literature review | 22 |
| References | 26 |

## Acknowledgements

Leïla Zerrouki, Nicolas Boulin and the people at EUROCONTROL in general, for answering questions, fixing bugs, and providing an interesting problem to work with[†].

Pierre Flener and Justin Pearson for guidance, ideas, and patience (probably too much of the last one).

[†] This work has been co-financed by the European Organisation for the Safety of Air Navigation (EUROCONTROL) under its innovative research grant scheme (grant 08-121447-C). The content of the work does not necessarily reflect the official position of EUROCONTROL on the matter.

# Background

## Air Traffic Management and Airspace Sectorisation

For the purposes of air traffic management (ATM), the airspace is divided into a number of disjoint sectors. A pair of controllers is assigned to each sector, and they are in communication with every plane within their sector, for the purpose of safely and efficiently directing traffic.

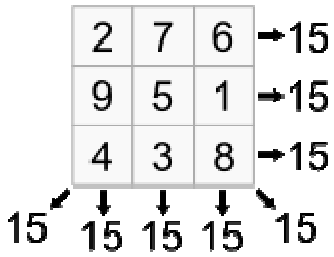
The aspect of ATM that we are interested in here is the creation of these sectors. With increasing requirements on the sectors come increasing requirements on automated methods for sectorisation. These methods must be able to evaluate and create good sectorisations. For this purpose, well defined criteria for what constitutes a good sectorisation have been provided by EUROCONTROL Experimental Centre (EEC); these constraints are described in the problem description.

## Constraint Programming

This section is a brief explanation of constraint programming. For a more in-depth understanding, a good resource is Handbook of Constraint Programming (F. Rossi, P. van Beek, and T. Walsh, editors; Elsevier, 2006).

A **decision variable** is a data structure whose state is a finite set of possible values (this set is known as the variable's **domain**). A decision variable with only one possible value is considered to be **assigned**. Constraint programming is a technique for finding correct or optimal assignments of values to a set of decision variables subject to some set of problem specific constraints. It consists of two parts: **propagators**, which are algorithms that enforce a constraint on the decision variables by removing incompatible values from their domains, and **search**, where we speculatively add simple constraints (such as variable A is equal to 1, or the opposite), in order to continue forwards when the propagators have exhausted their culling.

In order to understand the concepts better, let us look at a simple example: the magic square problem. A magic square of size n is an n by n square of $n^2$ different numbers, 1 to $n^2$. They are arranged in such a way as to make the sum of each row, column and major diagonal the same. Here is an example of a size 3 magic square:




Courtesy of Wikimedia Commons.

One way to model this problem is to represent each position in the square as a decision variable (let's call them $s_{11}$, ..., $s_{33}$) with a domain of {1, ..., 9}, and impose appropriate constraints:

First, each of the nine decision variables must have a different value (this is a very common constraint, and is often called the *alldifferent* constraint). Secondly, there are also eight sum constraints, each stating that the sum of three decision variables (that share a row, column, or major diagonal) must be 15. This value can be calculated from the fact that the magic value of a magic square of a given size n is the sum of all the possible values divided by the number of rows ((1 + 2 + ... + 9)/3 = 15), or $n(n^2+1)/2$ for a reduced but more obfuscated formula.

When finding a solution, the first thing that happens is that the propagator belonging to each constraint is run once, to see if any values can be removed right from the start (in this particular case, no values would be removed).

When the propagators have done what they can, it is time for search, which is done by adding some simple constraint. For example, in this case, let's add the constraint that $s_{11}$ (the decision variable representing the upper left corner) is equal to 1. Since the domain of a decision variable has changed, the propagators interested in this decision variable are run again. In this case, these are the propagators belonging to the alldifferent constraint and the three sum constraints involving $s_{11}$. This leads to 1 being removed from all the other variables (due to the alldifferent constraint), and 1, 2, 3 and 4 from the decision variables that share sum constraints with $s_{11}$ (since these values would force the value of the third decision variable to be greater than 9). This in turn activates the propagators interested in those decision variables, and so on. After the propagation is done, the decision variables may be in one of three different states:

- They have all been assigned, that is, the size of the domain of every decision variable is one. This means we have found a solution.
- The search has failed: the size of at least one decision variable domain is zero. This means we must backtrack to the most recent search node, resetting the domains of the decision variables to whatever they were before we added the last constraint, and add the opposite constraint (in this case, $s_{11} \neq 1$), since we have proven that the previous constraint cannot lead to a solution. If there is no prior search node, the entire search fails as no solution that fulfils all constraints exists.
- Otherwise, we search on by the same procedure.

Constraint programming can also be used for optimisation problems. This works by specifying some expression over the decision variables that is to be minimised or maximised. Whenever a solution is found, the search is restarted with a new constraint on this expression, stating that the new solution must be better than the previous solution.

There are a number of languages, libraries and other systems for constraint programming. For this thesis we used Comet, a C style language for solving combinatorial problems, which is a product of Dynadec (http://dynadec.com). Comet also supports other methods for solving combinatorial problems, such as linear programming and constraint based local search. Another option is Gecode, a C++ library for constraint programming (http://www.gecode.org).

# Methodology

## Data

The data for the sectorisation is provided via ASTAAC (Arithmetic Simulation Tool for ATFCM (Air Traffic Flow & Capacity Management) and Advanced Concept), the simulation tool provided by EEC. The tool calculates the cell and flight route data for a given sectorisation scenario by the parameters given to it. Our sectorisation method was mainly tested using the default parameters, which divides the airspace into hexagonal cells with a diameter of five nautical miles and a height of 1000 feet. Under these settings, a sectorisation scenario typically involves a few thousand to a few hundred thousand cells, to be clustered into five sectors.

Each cell is endowed with two key pieces of data that give us the workload of the cell: the number of conflicts occurring in that cell, and the total amount of flight time spent by airplanes in that cell. The software also comes with a formula to calculate the workload of a group of cells:

workload = (number of conflicts)*(workload per conflict) + (flight time in seconds)*(workload per minute of flight time)/60

where the workload per conflict and the workload per minute of flight time are cell-independent constants, while number of conflicts and flight time in seconds are values associated with each cell.

A key feature of this formula that must be noted is the division by 60. When calculating the workload of an entire sector, this is not a problem, as the flight time of all the cells is summed up first, but when calculating the workload of a single cell, this can lead to severe rounding errors. In order to avoid this, we instead multiply the workload by 60:

workload = 60*(number of conflicts)*(workload per conflict) + (flight time in seconds)*(workload per minute of flight time)

In addition to the cells, there is also in the data a set of flight routes. Each route is a sequence of cells, along with time stamps signifying when the route enters and exits each cell. The time stamps are relative to the route (the time of entry for the first cell is 0); absolute time stamps for individual planes are not used.

Finally, there is also a set of Abstract Functional Blocks (AFBs), which are sets of cells that ASTAAC has concluded must all be part of the same sector.

## Problem description

A sectorisation means that each of the cells must be assigned to precisely one sector. In order to be an acceptable sectorisation, certain constraints have to be satisfied.

- $C_1$: The sector workloads must be balanced.
- $C_2$: The number of entry points should be minimised.
- $C_3$: Avoid re-entering flights.
- $C_4$: Avoid short crossing flights.
- $C_5$: Ensure a sufficient distance between potential conflicts and the sector boundary.
- $C_6$: Ensure a sufficient distance between trajectories and sector boundaries.

$C_5$ and $C_6$ are covered within ASTAAC by the generation of AFBs. These are therefore enforced as a single constraint:

- $C_{AFB}$: Each cell within an AFB must be assigned to the same cell as all the other cells within that AFB.

For $C_1$, the workload of a sector is defined as the sum of the workloads of the cells belonging to that sector.

In the case of $C_2$, we define the number of entry points per flight route as the number of sectors that route passes through, and the total number of entry points as the sum of the per route values. It turns out that ASTAAC defines the number of entry points differently, in that it rather counts the number of cells that are entry points. That is, if two flight routes have an entry point in the same cell, then this would only count as one entry point, while our definition counts this as two. However, due to the constraint $C_5$, this should only occur when two or more flight routes start from the same cell, and consequently the difference between the two measurements should be constant, and so minimising the one will minimise the other. It does not seem motivated to use the more complicated formula required to represent the correct number of entry points.

A re-entering flight ($C_3$) is a flight that at some point enters a sector, then leaves it, and finally enters it again.

A short crossing flight ($C_4$) is a flight that remains within a sector a period of time shorter than 60 seconds.

Preprocessing

Looking at the data, there are several simplifications that can be made. The first is that the constraints on the problem only touch workloads and flight routes. If a cell has a workload of zero and is not a member of any flight route (and, in fact, these two conditions imply each other), then it can be disregarded. Typically over 90% of the cells are irrelevant in this manner.

Furthermore, the no short crossing flights constraint ($C_4$) implies that the first 60 seconds of each flight route must belong to the same sector, and symmetrically, so must the last 60 seconds. This creates further sets of cells that can be added to and merged with the AFBs.

The data on the cells, workloads, flight routes, and AFBs is then packaged and sent on to the constraint solver.

Model

The obvious way to model the problem is to create one integer decision variable for each cell, whose value signifies which sector the cell is assigned to. This in turn leads to representing flight routes as a sequence of these decision variables. An alternate representation would be to make each sector a set decision variable, signifying the set of cells that make up that sector, but it would be harder to express the flight routes in terms of this model.

In order to define the model, we first name some constants that are part of the data sent to the constraint solver, or which was calculated from that data.

- `nbSectors` - the number of sectors the sectorisation should create.
- `cellWorkload` - an array storing the workload of each cell.
- `totalWorkload` - the workload of the entire area being sectorised. Calculated as the sum of all entries in `cellWorkload`.
- `averageWorkload` - the average workload per sector. Calculated as `totalWorkload/nbSectors`.
- `workloadMin` - the minimum workload of a sector. Calculated as `averageWorkload - averageWorkload/4`. This is for enforcing $C_1$, see below.
- `workloadMax` - the maximum workload of a sector. Calculated as `averageWorkload + averageWorkload/4`. This is for enforcing $C_1$, see below.
- `afbs` - an array of integer arrays. Each inner array corresponds to an AFB, each integer in the inner array is the identifier of one cell in the AFB.
- `nbRoutes` - the number of flight routes.
- `sectorRange` - the range `0..(nbSectors-1)`. It is the range of sector identifiers.
- `cellRange` - the range of cell identifiers. It ranges from 0 to one less than the number of cells.
- `routeRange` - the range `0..(nbRoutes-1)`. It is the range of route identifiers.

The most central part of the model is

- `sector` - an array of integer decision variables relating the identifier of a cell to the sector that cell is assigned to. The domain of each decision variable is initialised to `0..(nbSectors-1)`.

Redundant decision variables (decision variables whose value is tied to the value of `sector` by some formula) are

- `sectorWorkload` - an array of integer decision variables relating the identifier of a sector to the workload of that sector. Each domain is `workloadMin..workloadMax`.
- `nbEntryPointsPerRoute` - an array of integer decision variables containing the number of entry points for each flight route. The domain of these decision variables is `1..nbSectors`. Their values are enforced by a custom propagator explained below.
- `nbEntryPoints` - an integer decision variable containing the total number of entry points of the entire sectorisation. Its domain is `nbRoutes..(nbRoutes*nbSectors)`.

Some channelling constraints tie these decision variables together. The workload of a sector is the sum of the workloads of its constituent cells:

```
forall (i in sectorRange)
    cp.post(sectorWorkload[i] == sum(j in cellRange) (sector[j] == i)* cellWorkload[j]);
```

The total number of entry points is the sum of the entry points per flight route:

```
cp.post(nbEntryPoints == sum(i in routeRange)nbEntryPointsPerRoute[i]);
```

The problem constraints also need to be enforced. $C_1$ is enforced by the domain of `sectorWorkload`. The true possible range of `sectorWorkload` would be `0..totalWorkload`; by restricting it to being within a quarter of the average workload, we require the workload to be reasonably balanced. Currently the model is hardcoded to allow a 25% variation, but this could easily be changed into a parameter.

In the early form of our model we tried minimising both the number of entry points and the workload imbalance, using their appropriately weighted sum as the cost function. This turned out to lead to very poor propagation, which is why we chose to make the workload imbalance a hard constraint.

$C_2$ is ensured by making `nbEntryPoints` the cost function that is to be minimised by the solver. Furthermore, that `nbEntryPointsPerRoute` corresponds to the number of entry points per route is ensured by a propagator described below as the *number of subsequences* propagator.

$C_3$ and $C_4$ are, like the entry points, enforced by a custom propagator described below as the *contiguity* and *sliding sum* propagators, respectively.

Finally, $C_{AFB}$ is enforced by requiring the appropriate entries in `sector` to be equal:

```
forall (i in afbs.getRange())
   for (int j = afbs[i].getLow(); j < afbs[i].getUp(); j++)
      cp.post(sector[afbs[i][j]] == sector[afbs[i][j+1]]);
```

Each of the three custom propagators share a similarity in that they operate on a sequence of integer decision variables (taken from `sector`) that correspond to the cells a given flight route passes through.

All of these are fairly problem specific constraints, and we did not find any already invented propagators that would help in enforcing them. Of them, the entry points, and perhaps the no re-entering flights constraint, could be modelled with the introduction of a large number of Boolean decision variables, signifying where a flight route passes into a new sector (and into what sector), but writing custom propagators seemed like a better idea.

All of the propagators enforce value consistency only, reacting only to decision variables being assigned values.

The contiguity propagator

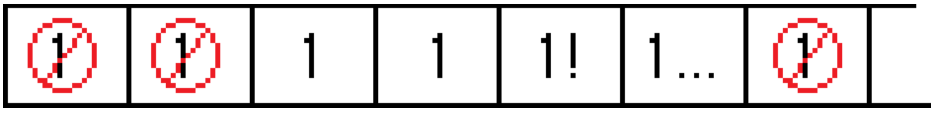
The propagator ensuring no re-entering flights functions by identifying which parts of the sequence *must* contain a given value, and which parts of the sequence *may* contain the value. Consider a sequence that looks, in part, like this:


The cells labelled only "1" have been assigned the value 1. The cells labelled "1 ..." are decision variables whose domains contain the value 1, but also other values. The cells with a crossed out 1 are decision variables whose domains do not contain 1. The cell with the exclamation mark is the decision variable that was assigned, and as a consequence activated the propagator.

The propagator starts by finding the nearest decision variables (to the assigned one), to the right and left, that do not contain the assigned value (the crossed out ones in the example). These form the bounds of the area of the sequence that may contain the assigned value (if no such decision variable is found to the left or right, the area is bounded by the beginning or end of the sequence), and any decision variable outside of them must have the assigned value removed from its domain.

Next, within this identified subsequence, the propagator finds the outermost decision variables that have been assigned. These form the bounds of the area of the sequence that must have the assigned value, and any decision variables between them must be assigned the given value.

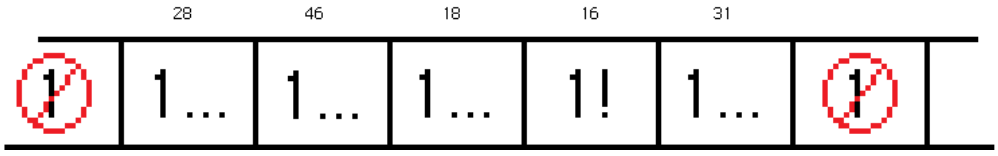
After these changes, the sequence looks like this (note the changes in the first and fourth cells):



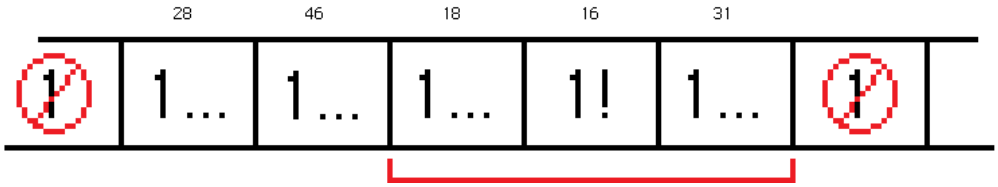
The sliding sum propagator

This propagator, which eliminates short crossing flights, like the contiguity propagator, starts by finding the maximal possible bounds of the subsequence it is working with.

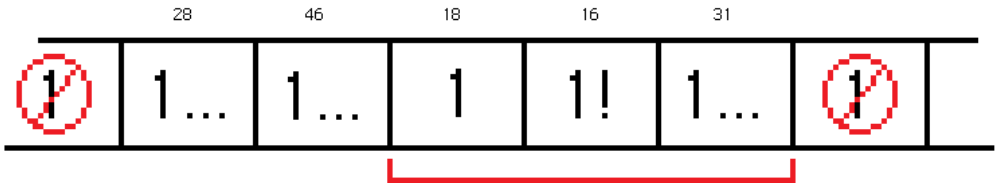
(In the following images the numbers along the top signify how long the flight remained in the cell.)




The propagator subsequently counts out a sufficiently long (measured by the time stamps) sequence from each end. In this example, the last two cells last 16 and 31 seconds, for a total of 47 seconds. This being less than the requisite 60 seconds means that the next cell to the left (duration 18 seconds) must be added.




Any decision variables within this area further from the relevant edge than a bound decision variable (the 18 second cell in this example) must be bound to the given value, as otherwise the stay in the sector would be too brief.

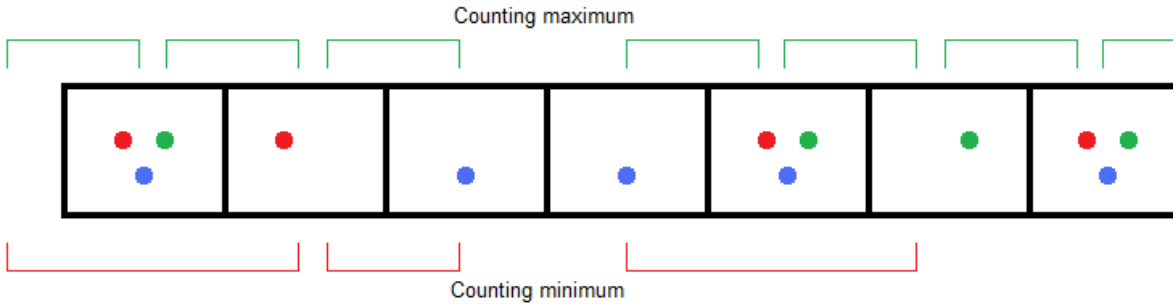



Since the binding of decision variables may have affected the possible length of subsequences to either side, the propagator then runs the same algorithm on any subsequences next to the current one, until it can find no more changes.

The number of subsequences propagator

The propagator for number of entry points, similarly to the contiguity propagator, is concerned with subsequences of flight routes. Consider the sequence of cells ($c_1$, ..., $c_n$) that makes up some given flight route. Any subsequence ($c_i$, ..., $c_j$) such that $c_k = c_{k+1}$ for k in [i,j-1], with $c_{i-1} \neq c_i$ or i=1, and $c_j \neq c_{j+1}$ or j=n (that is, a maximal same-valued subsequence) corresponds to a part of the flight route that is within the same sector. The number of such subsequences then corresponds to the number of sectors the flight route passes through, and consequently also to the number of entry points for that route.

Whenever a cell in the flight route is bound to a sector, the propagator paces through the decision variables and counts the maximum and minimum number of subsequences. The maximum number is defined as the number of unbound decision variables plus the number of decision variables bound to a value other than the previous decision variable. The minimum number is defined as the number of decision variables bound to another value than the nearest previous bound decision variable. These values are subsequently used to update the bounds of the number of entry points.




If, on the other hand, the decision variable representing the number of entry points should become bound, and the value it is bound to is also the minimal number of subsequences, then only the already attested to subsequences can exist; consequently any sequence of unbound decision variables must have all values except those of the bound decision variables at the endpoints removed from their domain.

Note that this is not the strongest possible propagator. Consider for example the case of a decision variable bound to 0, followed by an unbound decision variable whose domain does not contain 0, followed by another decision variable bound to 0. According to the propagator these three decision variables could be part of the same subsequence, even though this is impossible. This cannot be a problem for the specific problem due to the presence of the no re-entering flights constraint (which this sequence of decision variables would violate), but even in some other context, the propagator would accurately enforce the constraint, if perhaps not with optimal efficiency.

The presence of the no re-entering flights constraint also means that this constraint could be enforced using the nValue (also known as nbrDistinct) constraint, which enforces that a given decision variable is equal to the number of distinct values held by some set of other decision variables. Since all instances of a given value (or sector) must come in an unbroken sequence, the number of values is the number of subsequences. This might give better propagation. However, there are no fast domain-consistent propagators for this constraint, so it is unknown if this would gain us anything.

Search

The search is simple: at any choice point, an unassigned cell is picked, and speculatively assigned to the sectors in order based on the current minimum value for the workload of each sector. So if there are currently two sectors with one assigned cell each (with workloads of 72 and 13), and one sector with no cells assigned, the search will first try to assign the new cell to the sector with no cells (since it has a minimum workload of zero), next the sector with a minimum workload of 13, and last to the sector with workload at least 72.

Symmetry breaking

The numbers representing the sectors are only identifiers, and there are no constraints that in any way distinguish "sector 1" from "sector 4". Thus the decision variables representing which sector each cell is assigned to have full value symmetry. This means that symmetrical solutions can easily be discarded by dynamic symmetry breaking: when assigning a cell to some sector, we only consider at most one empty sector. So if in the above example sectors one and two had cells assigned, while sectors three, four, and five were empty, the search would still only try to assign the new cell to the first three sectors, since the last two would currently be identical to the third, all being empty.

Postprocessing

The cells that were ignored as being irrelevant to the constraints during the sectorisation must be assigned to sectors after the main event. Currently this is done by a very simple algorithm. Consider each unassigned cell. If all of the neighbouring cells are likewise unassigned, then ignore it for now. If at least one of the neighbouring cells has been assigned to a sector, then count how many times each sector is represented among the neighbouring cells. Assign the cell to the most widely represented sector. Repeat until all cells have been assigned.

## Initial results

The model as described in the previous sections will typically result in sectorisations that are in numerical terms quite good, when compared with the results of the NEVAC Sector Builder algorithm that came with ASTAAC. The results of three sectorisations are given below. In each case the sectorisation was based on flight data between 10 to 12 am on the same day.

All sectorisations were run on a single core of an AMD Athlon 64 X2 QL-65 (2100 MHz) under Windows Vista. The constraint programming model was implemented and run through the constraint solver Comet (by Dynadec (http://dynadec.com)), using its constraint programming backend. Comet was instructed to let the sectorisation run for four minutes. The NEVAC algorithm ran until it deemed itself finished, generally one or two minutes.

A note on cell sizes: the results below are from using cells with sizes between 5 to 7 nautical miles (nm). Larger sizes can be used up to a point, but at 20 nm and beyond the constraint solver typically does not find any solutions that satisfy all constraints.

Munich North ACC

| Constraint programming solver | | | | | NEVAC Sector Builder algorithm | | | |
|---|---|---|---|---|---|---|---|---|
| workload | entry points | re-entering flights | short crossing | sector | workload | entry points | re-entering flights | short crossing |
| 33 | 35 | 0 | 0 | 1 | 37 | 34 | 2 | 8 |
| 35 | 40 | 0 | 0 | 2 | 37 | 46 | 2 | 0 |
| 35 | 49 | 0 | 0 | 3 | 38 | 31 | 3 | 3 |
| 35 | 48 | 0 | 0 | 4 | 35 | 45 | 8 | 2 |
| 41 | 71 | 0 | 0 | 5 | 32 | 54 | 5 | 18 |
| totals: | 243 | 0 | 0 | | totals: | 210 | 20 | 31 |

London ACC Central

| Constraint programming solver | | | | | NEVAC Sector Builder algorithm |
|---|---|---|---|---|---|
| workload | entry points | re-entering flights | short crossing | sector | |
| 65 | 54 | 0 | 0 | 1 | no solution found |
| 60 | 49 | 0 | 0 | 2 | |
| 53 | 72 | 0 | 0 | 3 | |
| totals: | 175 | 0 | 0 | | |

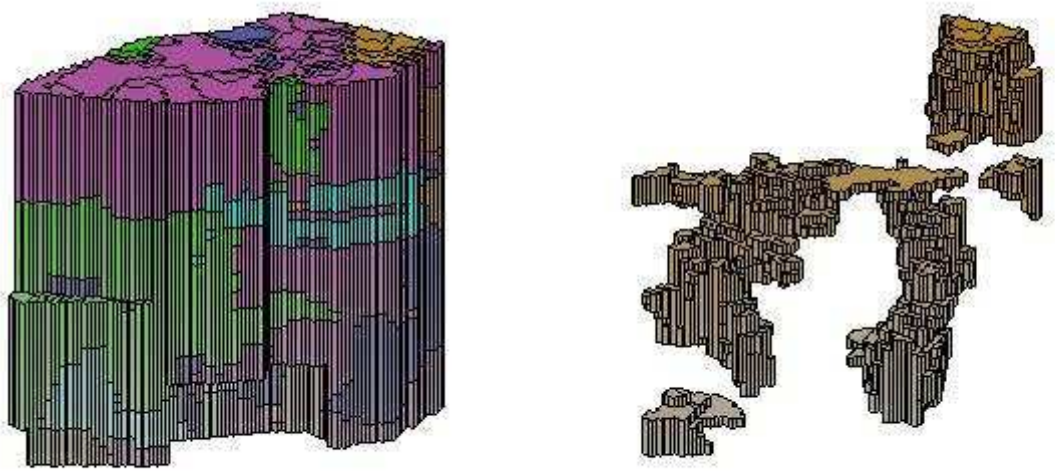
Paris East

| Constraint programming solver | | | | | NEVAC Sector Builder algorithm | | | |
|---|---|---|---|---|---|---|---|---|
| workload | entry points | re-entering flights | short crossing | sector | workload | entry points | re-entering flights | short crossing |
| 64 | 48 | 0 | 0 | 1 | 60 | 61 | 39 | 19 |
| 63 | 43 | 0 | 0 | 2 | 69 | 47 | 18 | 6 |
| 64 | 42 | 0 | 0 | 3 | 64 | 60 | 2 | 13 |
| 64 | 57 | 0 | 0 | 4 | 61 | 76 | 10 | 24 |
| 59 | 69 | 0 | 0 | 5 | 61 | 57 | 18 | 3 |
| totals: | 259 | 0 | 0 | | totals: | 301 | 87 | 65 |

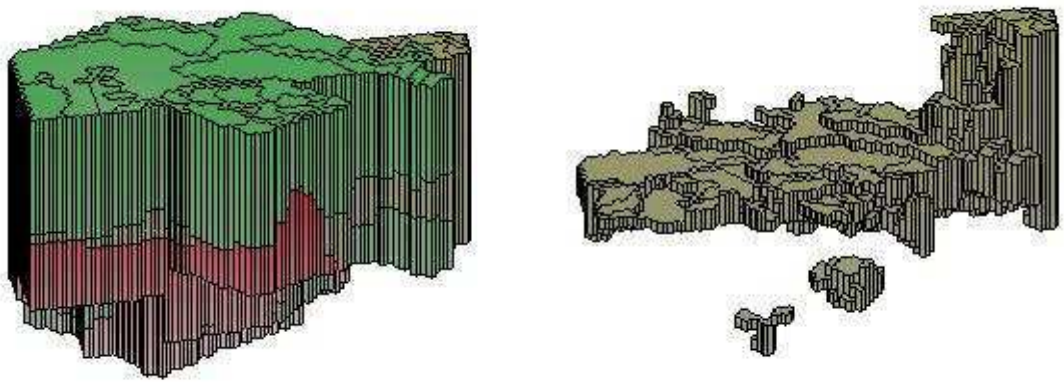
Of immediate note is the total elimination of re-entering flights and short crossing flights. The number of entry points is also comparable between the two algorithms; sometimes the constraint programming solution has a bit more, sometimes fewer (such as in the sectorisation of Paris East above), causing it to have clearly better values in each category.

Also of note is that NEVAC will sometimes fail to find a solution in cases where one is known to exist, despite lacking some of the hard constraints that our model has.

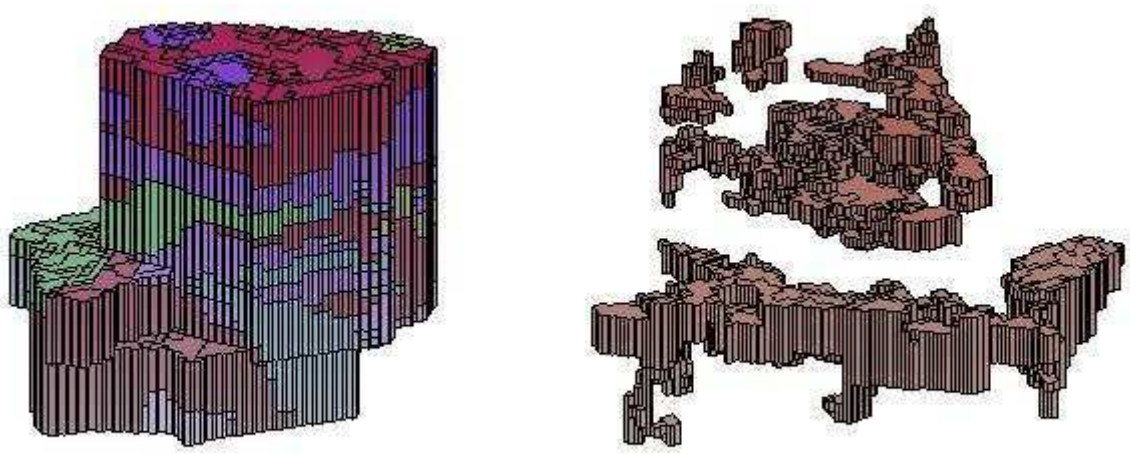
However, all is not perfect with our solution. The resulting sectors display a high degree of irregularity to their shape, typically including unconnected parts within the same sector. A graphical view of the same sectorisations as above:



A sectorisation of Munich North ACC. To the left, all sectors. To the right, one sector.



A sectorisation of London ACC Central. To the left, all sectors. To the right, one sector.



A sectorisation of Paris East. To the left, all sectors. To the right, one sector.

These sectorisations were performed using quite small cells (5 nautical miles in diameter in the first two cases, 7 in the third). Some experiments with larger cells were also done. The results were not greatly different in most cases, in terms of number of entry points or the resulting shape of the sectors. But in a few cases the constraint solver conclusively proved that there existed no solutions that satisfied all constraints, so a high resolution of the airspace can be necessary.

## Toward Sector Compactness

We speculate that the results above are sectorisations that human air traffic controllers would find difficult to wrap their heads around. It is likely that some sort of constraint on the shape of the sectors (beyond those created by the already defined constraints) is required. Consequently, we tested several methods for improving the compactness of the sectors.

In the tables below, numbers such as "29-37, 123" indicate that the sectorisation was successful, the minimum sector workload was 29 (minutes of workload), the maximum 37, and the number of entry points was 123. A mark of "--" indicates that the solver ran for the allotted time of ten minutes but did not find a solution. A mark of ":(" indicates that the solver proved no solution existed and returned before the time was up. Finally, a mark of "XX" indicates that the solver crashed due to too many constraints.

Sectorisations were run on six different data sets, which were identified within ASTAAC with the following names and codes:

| | |
|---|---|
| LFMNFCTA | NICE SECTORS |
| EDMMNCTA | MUENCHEN NORTH ACC |
| EGTTCCTA | LONDON ACC CENTRAL |
| LFFFECTA | PARIS EAST |
| LECMNCTA | MADRID ACC NORTH |
| ENOSCTA | OSLO ATCC |

To create a baseline, some sectorisations with different cell sizes but without any compactness constraints were made. The sizes tried were 5 nautical miles (nm) by 5 nm by 1000 feet (ft), 10 nm by 10 nm by 2000 ft and 15 nm by 15 nm by 3000 ft.

| | LFMNFCTA | EDMMNCTA | EGTTCCTA | LFFFECTA | LECMNCTA | ENOSCTA |
|---|---|---|---|---|---|---|
| 5nm×5nm×1000ft | 9-11, 36 | 32-41, 248 | :( | 48-75, 291 | 78-78, 545 | 14-22, 139 |
| 10nm×10nm×2000ft | 9-12, 31 | :( | 37-38, 198 | 63-68, 254 | 66-85, 382 | 14-22, 150 |
| 15nm×15nm×3000ft | :( | :( | :( | :( | 77-96, 297 | 17-21, 124 |

Smaller sizes were not viable due to the resulting large number of cells, and since four out of six data sets were probably impossible for the highest size, that size and larger ones were not tried in further tests.

The compactness of these sectorisations (as judged by visual inspection) was consistently very poor, with only marginal improvements for larger cell sizes.

Subsequently, we evaluated three different models for enforcing compactness, with varying parameters.

Compactness via the contiguity constraint

The contiguity constraint is used to enforce no re-entering flights, but with a different perspective it can also be used to improve compactness. The constraint functions on a sequence of cells; in the

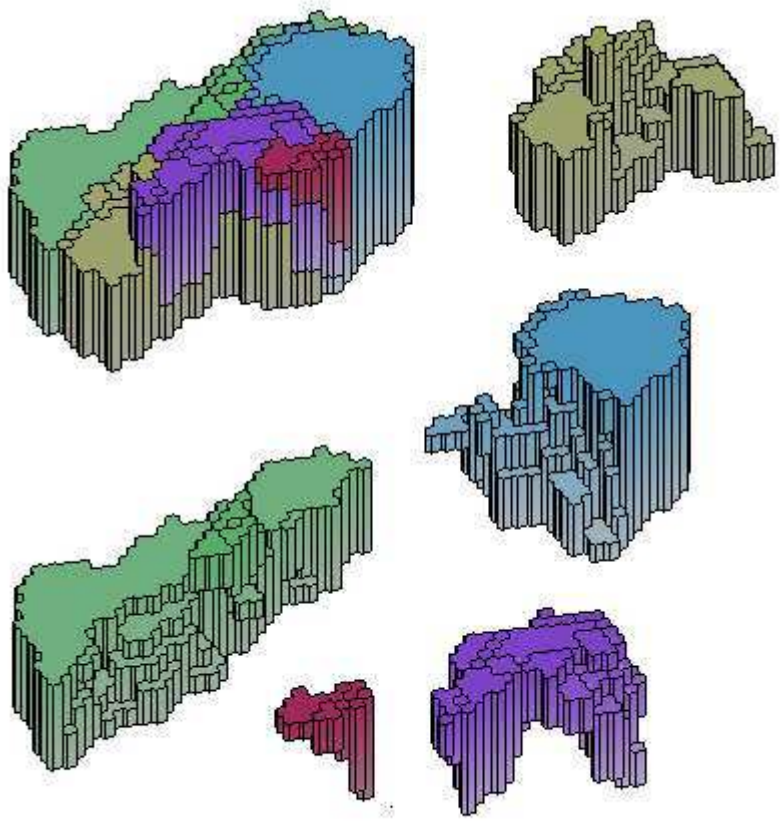
case of re-entering flights the cells that make up a flight route. In this case the sequence instead consists of all the cells that share latitude and longitude, but have differing altitudes (ordered by altitude), or similarly with latitude or longitude as the differing element.

An analogy would be if we were sectorising a chess board: to ensure compactness of the sectors, we could place contiguity constraints on each row and column. This means that all squares in a given column (or row) that are assigned to a given sector, must lie in an unbroken sequence, instead of possibly being separated from each other by squares belonging to other sectors. In the airspace sectorisation case, the problem has three dimensions, and so we can place such constraints on every row, column, and "pillar" of cells.

To begin with, when such constraints are placed on all three axes we get the following results:

| | LFMNFCTA | EDMMNCTA | EGTTCCTA | LFFFECTA | LECMNCTA | ENOSCTA |
|---|---|---|---|---|---|---|
| 5nm×5nm×1000ft | 9-11, 37* | -- | :( | -- | -- | -- |
| 10nm×10nm×2000ft | 9-12, 32 | :( | :( | 54-69, 302 | -- | -- |

This was clearly not entirely successful. What is notable here is that the sectorisation marked with an asterisk (*) is considerably more compact than the average sectorisation, and indeed the most compact result of any sectorisation we have run. In it, each sector consists of one connected piece, which is mostly convex. To be specific, it looks like this:



The upper left image is the five sectors together, the rest are the shapes of the individual sectors. Worth noting is that this data set covers a much smaller area than any of the other tested data sets. This and most other sectorisations on the LFMNFCTA data set finished before the allotted time was up, meaning that the results are provably optimal in terms of the optimisation criterion (that is, number of entry points). It is also worth noting that the addition of these contiguity constraints increased the minimal number of entry points only by one.

Since for most of the data sets no solution was found in the allotted time using these constraints, some tests using a weakened version of the constraint were also made. In this case, the contiguity constraints were placed only along the latitudes and longitudes, requiring compactness within each altitude layer, but allowing fragmentation between them.

| | LFMNFCTA | EDMMNCTA | EGTTCCTA | LFFFECTA | LECMNCTA | ENOSCTA |
|---|---|---|---|---|---|---|
| 5nm×5nm×1000ft | 8-11, 37 | 36-36, 297 | :( | 62-64,375 | -- | -- |
| 10nm×10nm×2000ft | 9-12, 32 | :( | -- | -- | -- | 18-19, 225 |

Those solutions that were found were disappointing: they had only slight increases in compactness. The very even workloads and high numbers of entry points (in comparison to the baseline tests) suggest that the solver did not have time to optimise well. It is possible that these constraints can enforce or be part of enforcing good solutions, but they do not seem to aid propagation enough to make for efficient searching.

Distance based compactness

A further attempt at enforcing compactness was made based on requiring cells distant from each other to be part of different sectors. Distance was measured as the Manhattan distance for simplicity, and based on the longitude, latitude and altitude values for each cell. A maximum distance was calculated based on the maximal and minimal values among all the cells, and cells that were further apart than some percentage of this maximal distance were required to be part of different sectors.

With this percentage being 80%, the results were:

| | LFMNFCTA | EDMMNCTA | EGTTCCTA | LFFFECTA | LECMNCTA | ENOSCTA |
|---|---|---|---|---|---|---|
| 5nm×5nm×1000ft | 9-11, 36 | 32-41, 247 | :( | 48-75, 291 | XX | 17-19, 157 |
| 10nm×10nm×2000ft | 9-12, 31 | :( | 37-39, 198 | 63-68, 254 | 65-84, 378 | 14-22, 173 |

A maximal distance of 60% gave this:

| | LFMNFCTA | EDMMNCTA | EGTTCCTA | LFFFECTA | LECMNCTA | ENOSCTA |
|---|---|---|---|---|---|---|
| 5nm×5nm×1000ft | 7-12, 37 | -- | :( | -- | XX | 18-18, 159 |
| 10nm×10nm×2000ft | 9-13, 31 | :( | 37-38, 229 | 49-70, 315 | -- | 18-19, 182 |

Setting the requirement to 40%, we got:

| | LFMNFCTA | EDMMNCTA | EGTTCCTA | LFFFECTA | LECMNCTA | ENOSCTA |
|---|---|---|---|---|---|---|
| 5nm×5nm×1000ft | 7-12, 39 | :( | :( | XX | XX | -- |
| 10nm×10nm×2000ft | :( | :( | :( | :( | -- | -- |

These results were also disappointing. The compactness did increase somewhat as the constraint tightened, but did little to prevent fragmentation. Once again, the largest effect was for LFMNFCTA.

Compactness by connectedness

A third attempt was with enforcing connectedness directly. This was done in two variants: in one case enforcing connectedness on the entire sector, in the other case enforcing connectedness within each altitude level. Initial results were unpromising, so for these sectorisations the time limit was doubled to 20 minutes, to see if this would help get any useful results.

Global connectedness:

| | LFMNFCTA | EDMMNCTA | EGTTCCTA | LFFFECTA | LECMNCTA | ENOSCTA |
|---|---|---|---|---|---|---|
| 5nm×5nm×1000ft | -- | -- | :( | -- | -- | -- |
| 10nm×10nm×2000ft | 8-12, 41 | :( | -- | -- | -- | -- |

Slicewise connectedness:

| | LFMNFCTA | EDMMNCTA | EGTTCCTA | LFFFECTA | LECMNCTA | ENOSCTA |
|---|---|---|---|---|---|---|
| 5nm×5nm×1000ft | -- | :( | :( | :( | -- | -- |
| 10nm×10nm×2000ft | 8-13, 40 | :( | :( | -- | -- | -- |

The results for LFMNFCTA were a significant improvement over the baseline, but not to the level of the contiguity case shown in a previous section (entitled Compactness via the contiguity constraint, page 17). Moreover, that case actually had global connectedness. The fact that in this case, no solution was found within twice the time illustrates that the connectedness propagator is very inefficient.

Another relevant thing to note regarding connectedness is that, unlike the other compactness constraints, it is affected by the preprocessing: those empty cells that were removed can serve to connect other cells together. More accuracy could be achieved by excluding the preprocessing, but one could not expect to find solutions by increasing the number of decision variables tenfold, most of which would only be touched by a single very weakly propagating constraint.

## Conclusions and further work

Overall, the problem is that the search space of possible solutions is very large, and the propagators too weak. This means that there is no possibility to explore more than a very small fraction of the possible solutions. At the start of the solving, none of the propagators can do anything, which basically means that the solver has to arbitrarily assign some cells to sectors, and let the propagation proceed from there. Since the search space is so large, the search never has time to backtrack to the root of the search tree, so those initial choices do not ever get re-evaluated, leading to strange sectorisations. This could possibly be avoided, by use of some clever heuristic for search: when choosing a cell and a sector to assign it to, choose a combination that is likely to lead to a good solution.

The heuristic in use (based on sector workloads) only helps with satisfying the workload balance constraint. Other attempts were based on proximity to sectors: either measured by calculating a midpoint for each sector (based on the positions of cells already assigned to that sector) and assigning cells based on what midpoint they are closest to, or assigning cells based on the sector of the nearest assigned cells (that is, based on distance to the nearest sector border rather than midpoint), but these led to no solutions. Adding workload concerns to these heuristics (such as considering sectors with low workloads to be closer) also did not lead to solutions.

How do we solve this then? Since the problems are related to the inability of searching through the entire solution space, perhaps a method not based on attempting to do so would be more appropriate. Constraint based local search (CBLS) is a method that shares several similarities to constraint programming, particularly in the modularity of modelling problems by tying variables together by constraints. See Constraint-Based Local Search (Pascal Van Hentenryck and Laurent Michel; The MIT Press, 2005) for an in-depth resource. Unlike constraint programming, CBLS starts with all variables assigned a value, and evaluates how badly this assignment violates the constraints. Then it calculates which (single) variable should be switched to which value in order to maximize the improvement, repeating this step (using various techniques to avoid getting stuck in local minima) until the solution will not improve further. CBLS is better than constraint programming at multicriterion optimisation, and would be able to search more effectively for solutions to for example connectedness. It could also search for solutions using other constraints that would be difficult for a constraint programming solver to propagate, such as minimising the size of the sector boundary (that is, the number of cells that have a neighbouring cell in a different sector), which is possibly even closer to the desired property than connectedness is.

Comet (the system used for constraint programming in this thesis) also supports and is a good choice for constraint based local search.

## Literature review

Optimized Sectorization of Airspace with Constraints [1]

The paper by Trandac, Baptiste, and Duong stands out as being the most directly relevant to our topic. The problem to be solved is essentially the same: divide some piece of airspace into some number of sectors. This sectorisation further has to satisfy most of the same constraints: no re-entering flights, no short crossing flights, balanced workload, and so forth.

The problem is modelled as a graph where the vertices correspond to beacons and crossing points, and an edge between two vertices exists if and only if there is a direct route between the two points. This is a somewhat more abstracted view of the problem than that presented by dividing all of the airspace into cells; essentially it only considers the most "interesting" cells.

The final sectorisation consists of associating with each vertex a polygon and letting the polygons associated with the vertices that make up a sector define the borders of that sector.

The solution is optimised by minimising the coordination workload, a value associated with each edge. The workload of an edge is only included in the total if that edge connects nodes in different sectors. Minimising this workload seems fairly directly equivalent to minimising the number of entry points as in our version of the problem. Arguably it is a more fine-grained measurement as it does not count only the number of entry points, but more precisely how much work each one entails.

The solution is obtained by means of a constraint solver. For small random instances the solver manages to prove optimality; for larger instances further optimisation is obtained by randomly picking a small number of adjacent sectors from the results, and resectorising them by the same method, repeatedly until no further improvements show up.

The model is capable of efficiently finding solutions to problem sizes of up to 80 vertices. One detail to note is that the experiments were run on randomly generated 2D-instances of the problem, rather than real-world flight data.

Temporal and Spatial Distribution of Airspace Complexity for Air Traffic Controller Workload-Based Sectorization [2]

Yousefi and Donohue note that methods to measure the workload of air traffic control are not widely agreed upon. Additionally, many of the metrics used require that the boundaries of sectors already be defined, making them not useful for sectorisation. Like in our work they base their sectorisation on dividing the airspace into hexagonal cells. However, due to the method they chose to calculate workload, they arrive at a minimum cell size of a side length of 24 nautical miles, dividing the entire US airspace into 2566 hexagonal cells (times three for three altitude layers, each of which is sectorised separately). They calculate hourly workload values for each cell by means of a large scale simulation of air traffic, in the end keeping the peak values occurring at 20:00 to 21:00 UTC.

The actual sectorisation is performed by defining a large number of equally spaced potential centres for sectors. The problem is modelled as a facility location problem, where each potential centre is a facility, and each cell is a customer; each cell must be assigned to a centre, but not every centre needs to have cells assigned to it.

The clustering process is required to provide an even workload as well as to enforce some constraints: contiguity, avoid highly concave sectors, avoid extremely large sectors, limit the number of sectors, etc. The sectorisation is performed by a linear integer minimisation program.

The authors note that the problem is too large to be solved for the entire US airspace at once, and consider dividing the problem into smaller pieces, such as a Pacific-Atlantic split, or by splitting along the time zones. They further consider getting a higher resolution by calculating intermediate workload values through interpolation between hexagons.

A Tool to Design Functional Airspace Blocks [3]

Bichot and Durand seek to evaluate and develop methods for functional airspace blocks (FABs), which are sets of sectors created to enforce regional cooperation and integrate management of the airspace. FABs should not be confused with the abstract functional blocks (AFBs) that are relevant to sectorisation.

The problem is analogous to that of sectorisation, in that it involves clustering smaller building blocks into some number of larger entities. Some differences lie in the building blocks in this case being relatively large sectors rather than the much smaller cells used to build those sectors, and in the constraints on an FAB being much different from those on a sector.

The problem is modelled as a graph with vertices corresponding to sectors and edges corresponding to flows of aircraft between sectors. Both the vertices and edges are weighted according to the number of aircraft passing through them.

Having modelled the problem as a graph partitioning problem, the paper compares a number of graph partitioning libraries as well as their own method called Fusion Fission, which is based on the fusion and fission of physics. It functions by ejecting vertices that are loosely bound to their vertex cluster, and merging them with neighbouring clusters, triggering fission in them. They find that fusion fission gives better results for the specific FAB problem, though at a cost of considerably longer running time.

"Airspace Playbook": Dynamic Airspace Reallocation Coordinated with the National Severe Weather Playbook [4]

Klein, Kopardekar, Rodgers, and Kaing consider the effects of the application of the National Severe Weather Playbook (NSWP). The NSWP contains a set of predefined air traffic reroutings in response to specific weather conditions. Applying these reroutings leads to air traffic taking routes through other sectors than planned, leading to workload imbalance.

The paper proposes an Airspace Playbook to go with the NSWP, containing re-sectorisations to go with reroutings of the various severe weather scenarios. For the purposes of the study the NAS (National Airspace System) Airspace Dynamics Analysis and Partitioning Tool (NAS-ADAPT) was used for re-sectorisation. NAS-ADAPT grows sectors by starting from seed locations.

A re-sectorisation being a sectorisation, it is easy to see how our research is applicable to the problem. One difference to note is that a resectorisation may have slightly different constraints (such as valuing a similarity to the standard sectorisation).

Introducing Structural Considerations into Complexity Metrics [5]

Histon, Hansman, Aigoin, Delahaye, and Puechmorel examine the factors that link structural complexity (complexity in the composition of sectors and other demands on ATC) and cognitive complexity (the actual degree of difficulty experienced by air traffic controllers in maintaining situational awareness and managing safe operations).

Based on interviews and observations the authors identify a number of factors relevant for sectorisation: sector shape, size, and area of regard (border parts of neighbouring sectors the air traffic controller also needs to pay attention to), the number of points of ingress/egress, as well as the orientations of and interactions between flows.

The constraints imposed on our sectorisation are basically all responses to these factors. One notable exception is that there were no explicit constraints on the shape of the sectors, and this is very noticeable in the results of the sectorisation.

New Process for "Clean Sheet" Airspace Design and Evaluation [6]

Conker, Moch-Mooney, Niedringhaus, and Simmons describe a semi-automated method for airspace sectorisation. Some of their main goals are to avoid parochialism and create a process that is objective, transparent, and repeatable. Therefore they discard the traditional idea of getting subject matter experts to create the initial design and then using quantitative analyses to evaluate the design, and instead start with the quantitative analyses to create an objective design and subsequently test the design for feasibility and refine it.

The airspace is divided into square cells, and these are used as the basic building blocks of the sectorisation. These cells are sectorised into dense clusters, and the workloads of the clusters are equalised by moving cells from low workload clusters to high workload clusters. After this the method leaves the cell-based representation in order to remove the serrated edge effect that a cell clustering method of sectorisation creates.

Since the initial sectorisation is done without considering anything but a per-cell based complexity measure, the sector boundaries must then be evaluated for problems with traffic flow interactions. The suggestions from this evaluation can then be used to adjust the sectors to not violate the operational requirements. The evaluation-adjustment loop is repeated until the result is deemed satisfactory.

This method differs from our work mainly in the requirement for human input as part of the sectorisation, and by the division of workload balancing and constraint satisfaction into separate parts of the process.

Analysis of Aircraft Clusters to Measure Sector-Independent Airspace Congestion [7]

Bilimoria and Lee seek to find a better way of measuring congestion than looking only at the airplanes within a given sector. The idea is to identify clusters of aircraft independently of their location relative to the sector borders. By using projected future aircraft positions, clusters signifying congested air traffic situations can be detected before they occur.

The cluster identification is performed by means of defining a threshold distance, and assigning to the same cluster any two planes within the threshold distance of one another; the clusters are defined by the transitive closure of this proximity relation.

All clusters created by this process are not equal; a clearly defined cluster has a considerable larger distance to its nearest neighbours outside the cluster than the average within-cluster distance. The authors conclude that their method yields results consistent with human intuition, and that identified "high quality" clusters correspond to real world problem situations.

The paper is not clearly related to our work in a direct way. Indeed, the first thing the authors do is discard the notion of sectorisation as obscuring actual incidences of congestion. Furthermore the work is about studying the actual and projected locations of individual airplanes once they are actually in the air, while our work in sectorisation is based on flight routes where individual flights are only represented by contributing to aggregate values.

Improved Configuration Optimiser ICO Methodology to Use a Decision Support Tool [8]

This paper was not studied in detail as we judged it to be non-relevant.

References


[1]

| | |
|---|---|
| Title | Optimized Sectorization of Airspace with Constraints |
| Authors | Huy Trandac, Philippe Baptiste, Vu Duong |
| In | Proceedings of the 5th FAA and EUROCONTROL ATM Conference (2003) |
| URL | http://atm2003.eurocontrol.fr/past-seminars/5th-seminar-budapest-hungary-june-2003/papers/paper_071/view |

[2]

| | |
|---|---|
| Title | Temporal and Spatial Distribution of Airspace Complexity for Air Traffic Controller Workload-Based Sectorization |
| Authors | Arash Yousefi, George L. Donohue |
| In | Proc of AIAA Aviation Technology Integration and Operations Forum (2004) |
| URL | http://pdf.aiaa.org/preview/CDReadyMATIO2004_1008/PV2004_6455.pdf |

[3]

| | |
|---|---|
| Title | A Tool to Design Functional Airspace Blocks |
| Authors | Charles-Edmond Bichot, Nicolas Durand |
| In | Proceedings of the 7th Air Traffic Management seminar (ATM) (2007) |
| URL | http://liris.cnrs.fr/publis?id=4114 |

[4]

| | |
|---|---|
| Title | “Airspace Playbook”: Dynamic Airspace Reallocation Coordinated with the National Severe Weather Playbook |
| Authors | Alexander Klein, Parimal Kopardekar, Mark D. Rodgers, Hong Kaing |
| In | Proceedings of the 7th AIAA Aviation Technology, Integration and Operations Conference (2007) |
| URL | http://www.aviationsystemsdivision.arc.nasa.gov/publications/strategic/Klein_AIAA_ATIO_2007_1.pdf |

[5]

| | |
|---|---|
| Title | Introducing Structural Considerations into Complexity Metrics |
| Authors | Jonathan M. Histon, R. John Hansman, Guillaume Aigoin, Daniel Delahaye, Stephane Puechmorel |
| In | Air Traffic Control Quarterly, Vol. 10, No. 2 (2002) pages 115-130 |
| URL | http://dspace.mit.edu/handle/1721.1/37320 |

[6]

| | |
|---|---|
| Title | New Process for "Clean Sheet" Airspace Design and Evaluation |
| Authors | Robert S. Conker, Debra A. Moch-Mooney, William P. Niedringhaus, Brian T. Simmons |

| | |
|---|---|
| In | 7th US/Europe ATM Seminar (2007) |
| URL | http://www.atmseminar.org/seminarContent/seminar7/papers/p_091_DAM.pdf |

[7]

| | |
|---|---|
| Title | Analysis of Aircraft Clusters to Measure Sector-Independent Airspace Congestion |
| Authors | Karl D. Bilimoria, Hilda Q. Lee |
| In | AIAA 5th Aviation Technology, Integration, and Operations Conference (2005) |
| URL | http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20050182926_2005183224.pdf (extended abstract only) |

[8]

| | |
|---|---|
| Title | Improved Configuration Optimiser ICO Methodology to Use a Decision Support Tool |
| Authors | C. Verlhac, S. Manchon |
| In | EEC Technical/Scientific Report 2005 |
| URL | http://www.eurocontrol.int/eec/public/standard_page/DOC_Report_2005_015.html |