Airspace Sectorisation using Constraint-Based Local Search

Justin Pearson
justin.pearson@it.uu.se

Uppsala University

June 2013
We implemented a prototype airspace sectorisation system using Constraint-Based Local Search (CBLS).

CBLS is a declarative framework for combinatorial optimisation that separates model and search.
Sectorisation

- Given a set of flights over some time horizon divide the airspace in sectors such that their workload is balanced.
- Many definitions of workload exist, but our model is workload definition agnostic.
Two major approaches:

- **Graph based**: Make a graph based on flight routes. Nodes represent route crossings. Sectorisation is then a partition of nodes.

- **Cell Based**: Divide the airspace into cells. Then do some geometric reasoning to get the sectorisation.

We used a cell based approach.
Sectorisation is an NP-complete problem.

This means that there is no known algorithm that always efficiently solves the problem. If there was, then many unsolved problems in computer science would be answered.

Most approaches try to use heuristics that work well in practice.

By using a high-level declarative framework, we avoid having to write ad hoc algorithms.
Constraint Programming

Constraint Programming (CP) offers methods and tools for:

- Effectively **modelling** constraint problems.
- Efficiently **solving** constraint problems, by **global search** (in Sudoku fashion) or by **local search**.

Slogan of CP:

\[
\text{Constraint Program} = \text{Model} + \text{Search}
\]
Local Search

- Values are given to all the variables at the same time.
- Search proceeds by moves, which make small updates to complete assignments, upon probing the impacts of many candidate moves, called the neighbourhood.
- Stop when a good enough assignment has been found or when an allocated resource (running time, or a number of iterations) has been exhausted.
A problem is modelled as a conjunction of constraints, which declaratively encapsulate inference algorithms specific to common combinatorial substructures and are thus reusable.

Each constraint has an associated penalty that measures how far the current solution is from satisfying the constraint.

A master search algorithm operates on the model, guided by user-indicated/designed (meta-)heuristics. The main idea is to find moves that reduce the penalty.
Example (\texttt{AllDifferent}\{(x_1, x_2, x_3, x_4)\})

- When $x_1 = 5$, $x_2 = 5$, $x_3 = 5$, and $x_4 = 6$:
  - The constraint violation is 2, since at least two variables must be changed to reach a satisfying assignment.
  - The variable violations of $x_1$, $x_2$, and $x_3$ are 1.
  - The variable violation of $x_4$ is 0.

This would guide the CBLS system to consider changing $x_1$, $x_2$ or $x_3$. 
Constraint Programming = Model + Search

Model:

- Variables: One variable per cell as output from ASTAAC.
- Constraints
  - Balanced workload (soft) (Combination of conflict and monitoring workload)
  - Minimum dwell time (soft)
  - Convexity (soft) Trajectory based
  - Connectedness (hard)
  - Compactness (soft)

- The cost is the sum of the penalties of the constraints above and is minimised.
Search:

- Use standard local search techniques: tabu lists, random restarts . . . .
- Number of entry points is not minimised, because we conjecture that it is minimised as an effect of minimising the cost.
- Connectedness is our only hard constraint. Our initial solution is constructed to be connected. We then pick our neighbourhoods so that all moves preserve connectedness. This is a common technique in local search.
Define convexity in terms of flight trajectories.
Consider the sequence

\[1, 1, 2, 1, 1, 1, 4, 4, 1, 1, 4, 4\]

Each number is the currently assigned cell.

We penalise the number of reentries. Sector one is revisited twice and hence given a convexity penalty of two.

For compactness we minimise the surface area of each sector.
Results

Good

Bad

Sectorisation using CBLS
Conclusions and Future work

- Using a declarative approach (CBLS) allowed easy prototyping of various alternatives.
- Future work includes looking more closely at compactness and develop more reusable constraints.
- Incorporating more realistic operational constraints.
- Dynamic baseline sectorisation.