**RAIRO Operations Research**
RAIRO Oper. Res. **39** (2005) 105–122
DOI: 10.1051/ro:2005005

# AIRSPACE SECTORIZATION WITH CONSTRAINTS

Huy Trandac[1], Philippe Baptiste[2] and Vu Duong[3]

**Abstract.** We consider the Airspace Sectorization Problem (ASP) in which airspace has to be partitioned into a given number of sectors, each of which being assigned to a team of air traffic controllers. The objective is to minimize the coordination workload between adjacent sectors while balancing the total workload of controllers. Many specific constraints, including both geometrical and aircraft related constraints are taken into account. The problem is solved in a constraint programming framework. Experimental results show that our approach can be used on real life problems.

**Keywords.** Airspace sectorization, constraint programming.

**Mathematics Subject Classification.**

## 1. Introduction

Sectorization is a fundamental architectural feature of Air Traffic Management (ATM). Airspace is divided into a number of sectors, each one is assigned to a team of controllers. Controllers of a given sector have (1) to monitor flights; (2) to avoid conflicts between aircrafts and (3) to exchange information with adjacent sectors to take or to hand-on the control responsibility of flights. These tasks induce a workload which is often decomposed into three corresponding parts [8–10]:

- *The monitoring workload* (MW) comes from the repeated checking of aircraft trajectories.

Received September 6, 2003. Accepted March 24, 2005.

[1] College of Information Technology, Can Tho University, Viet Nam.
[2] CNRS LIX, École Polytechnique, 91128 Palaiseau, France;
`Philippe.Baptiste@polytechnique.fr`
[3] Eurocontrol Experimental Centre, Centre de Bois des Bordes, BP15, 91222 Bretigny sur Orge Cedex, France.

- *The conflict workload* (CW) results from resolution and avoidance of conflicts between aircrafts.
- *The coordination workload* (OW) is basically related to communications that have to be performed with controllers of adjacent sectors and with pilots.

Air traffic volume changes over the day, and often leads to an unbalanced workload between sectors. Furthermore, it is better to have more sectors in dense traffic periods of the day than in the low loaded periods. Hence a tool to *dynamically* re-sectorize the airspace (more precisely a part of airspace – *e.g.* the sectors of an En-Route Air Traffic Control Center) is required to cope with the evolution of the traffic.

We rely on the following **model**: airspace is made of routes that cross each other (routes are provided as a list of successive 3D points). In the following, $G = (V, E)$ denotes the graph representing the airspace. $V$ is the set of crossing points $v_i$, and $(v_i, v_j) \in E$ if and only if there is a direct route from $v_i$ to $v_j$.

The graph $G$ is labelled both on its vertices and edges. The valuations are used to model the three parts of workloads mentioned above. We use a static estimation of each of the workloads (achieving a more precise estimation is a very complex task and it is beyond the scope of this paper):

- The coordination workload of an edge is proportional to the number of aircrafts passing this edge. The cost is only taken into account when the edge crosses two sectors.
- The monitoring workload is proportional to the total time that aircrafts fly along the edge. So when an edge crosses two sectors, one should compute the length of the edge in each of the sectors. To simplify the presentation, we split the monitoring cost of an edge in two equal values, that we allocate to each of the two vertices of the edge.
- The conflict workload is associated to the vertices and is proportional to the total number of conflicts that occur at the corresponding crossing point.

So we have two valuations on $G$: $\omega_i$, the weight of $v_i \in V$ corresponding to the aggregation of the monitoring workload and the conflict workload, and $\omega_{ij}$, the weight of $(v_i, v_j) \in E$ corresponding to the coordination workload. The similar workload decomposition schemes have been used (see for instance [10]) and have been validated by air-traffic controllers.

We follow the idea of [10] that instead of defining sectors through a *geometric description*, we can define a sector as a convex *set of vertices.* The main advantage of this approach is to resume to a purely discrete problem. And when a solution of the discrete problem is found, we will compute the boundaries for the sectors.

We now look for a partition of the set $V$ into $k$ subsets $V_1, V_2, ..., V_k$; $\forall i \neq j$ : $V_i \cap V_j = \phi$ which minimizes the weighted sum of cut routes (called *cut-size* in the following)

$$\min \sum_{v_i \in V_p; v_j \in V_q; p \neq q} \omega_{ij}$$

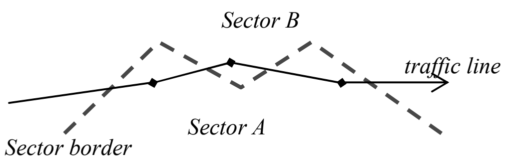


Figure 1. Route-based convexity constraint.

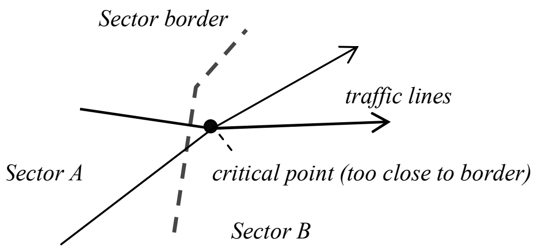


Figure 2. Minimum distance constraint.

such that the weights of the subsets are balanced within a tolerance ratio of $0 \leq \delta < 1$:

$$\forall p \in \{1, ..., k\}, \left| \sum_{v_i \in V_p} \omega_i - \frac{1}{k} \sum_{i=1}^{n} \omega_i \right| \leq \delta \frac{1}{k} \sum_{i=1}^{n} \omega_i .$$

On top of this balance constraint, several specific ATC (Air Traffic Control) constraints have also to be taken into account (a formal definition will be provided in the Sect. 3):

- *Route-based convexity constraint.* The same aircraft can not enter the same sector twice. It is not sensitive, but it happened as a constraint in the past, *e.g.* national boundaries in European airspace. For instance, the following case in Figure 1 is not admissible.
- *Minimum distance constraint.* The distance between a sector border and a vertex must be not less than a given distance (see Fig. 2). This constraint ensures that the controller has enough time to solve conflicts at this vertex.
- *Minimum sector crossing time constraint.* The aircraft must stay in each crossed sector at least a given amount of time $T_{min}$ (see Fig. 3). This constraint ensures that the controller has enough time to control the aircraft.
- *Connectivity constraint.* The sector can not be fragmented. For example, the solution in Figure 4 is not feasible.

A genetic algorithm [15] to solve the ASP has been proposed in [8]. Chromosomes are defined as sets of sector center points; the sector is then defined as the Voronoi diagram [24] associated to the set of center points (*i.e.*, a sector is a set of points that are closer to its center point than to any others). Voronoi-like sectors are geometrically convex but this property does not ensure that sectors are route-based convex (aircraft enter twice the same sector). In Figure 5, by joining $v_1$

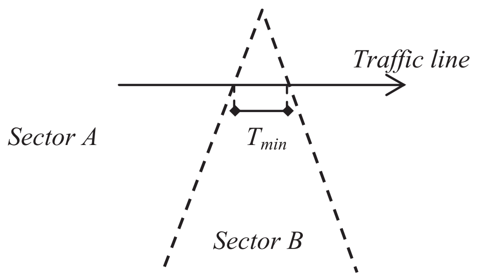


Figure 3. Minimum sector crossing time constraint.

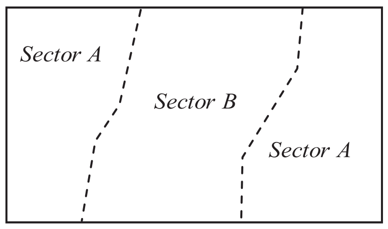


Figure 4. Connectivity constraint.

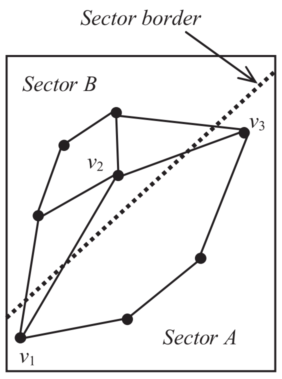


Figure 5. Sector is not convex by joining $v_1$ and $v_3$ *via* $v_2$.

and $v_3$ *via* $v_2$, the route-base convexity constraint is violated although the sector is geometrically convex.

Delahaye *et al.* [10] have tried to improve this approach. A sector is defined by a set of connected vertices of the network and the chromosome contains all information needed to define the sectors. But "connected" sectors do not ensure the convexity constraint. Again as showed in the example of Figure 5, all vertices of sectors are connected, but sectors are not route-based convex by joining $v_1$ and $v_3$ *via* $v_2$.

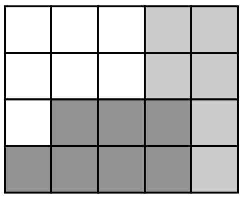


FIGURE 6. Sectors defined by joining elementary units.

A more recent approach suggests that airspace be divided into small volume units and a sector is obtained by joining some of these elementary units [23] (see Fig. 6). Unfortunately, the most specific ATC constraints can not be taken into account and for instance, the sectors can be fragmented in the suggested solution.

In this paper, we rely on the mathematical formulation of the ASP mentioned above. This problem is closely related to Graph Partitioning, a widely studied combinatorial problem. Although it is NP-complete [13], very efficient heuristic procedures have been introduced in the last 30 years and very large problems can be solved efficiently in a reasonable amount of CPU time [2,11,12,14,16–20].

We firstly show in Section 2 how to compute sectors boundaries when we have a solution to the discrete partitioning problem. In the Section 3, we introduce a highly flexible constraint programming (CP) formulation which can help us to solve small instances of the ASP. To solve larger instances, we use a two-phase approach: firstly, we apply a recursive bisection schema and a restricted Kernighan/Lin (RKL) heuristic to find a *good* solution; and in the second phase, we enter a re-optimization loop, always relying on the CP model, that improves this solution (Sect. 4). Experimental results are reported in Section 5.

## 2. From sets to geometrical sectors

Assume that we have an airspace which has to be sectorized into a number of sectors, each represented by a set of vertices, we must then compute non-overlapping sector boundaries such that each sector boundary contents all its vertices and, as mentioned above, can not be fragmented.

For example in the 2D case, we must determine for each sector a simple polygon which contents all its vertices. We propose a way to compute such sector boundaries for the 2D airspace case. Note that it can be easily extended to the 3D case.

**Definition 2.1.** A *Point-based Polygonal Tessellation* of a plane with a set of $n$ points $V$, denoted by $PT(V)$, is a partitioning of the plane into $n$ non-overlapping polygons, called tiles, such that each polygon $P(v)$ contains exactly one point $v \in V$.

**Definition 2.2.** The *Neighboring Relative Graph* of a point-based polygonal tessellation $PT(V)$, denoted by $NRG(PT(V))$, is the graph constituted by $(V, E)$,

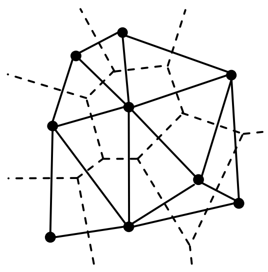


FIGURE 7. Delauney triangulation (dashed line) and Voronoi diagram.

where edge $(u, v)$ belongs to $E$ if and only if $P(u)$ and $P(v)$ have at least one common side.

The Figure 7 illustrates us an example of the case in which the Delauney Triangulation is the NRG of Voronoi Diagram [24]. The Voronoi diagram (also known as the Direchlet tessellation or Theissen tessellation) is a subdivision of a plane into a number of tiles; each tile has one sample point in its interior called a generating point. All other points inside the polygonal tile are closer to the generating point than to any other. Its dual, the Delauney triangulation [24] (the term *triangulation* is defined in the next), is created by connecting all generating points which share a common tile edge.

**Proposition 2.3.** *Let $PT(V)$ be a point-based polygonal tessellation of a set of points $V$ in a plane and the corresponding $NRG(PT(V))$. For all subsets $V_i \subset V$, if a subgraph of $NRG(PT(V))$ corresponding to $V_i$ is connected, then there exists a simple polygon which contains all points of $V_i$ and no point in $V \backslash V_i$.*

The proof of this proposition is given by construction of such a polygon: we group the tiles corresponding to all points in $V_i$ and remove all shared sides. Since each tile contains only one point of $V$ by the definition of $PT(V)$, this polygon contains only the points of $V_i$, but no point in $V \backslash V_i$.

Hence, the boundary of a sector can be obtained by grouping the corresponding tiles of its vertices and a sector $V_i$ is un-fragmented if $NRG(PT(V_i))$ is connected. Our problem is now how to obtain a point-based polygonal tessellation of a set of points in a plane and the corresponding NRG.

**Definition 2.4** [5]**.** A *Triangulation* of a set of $n$ points $V$ in the plane, denoted by $T(V)$, is joining the points of $V$ by non-intersecting straight line segments such that all regions are triangles.

**Observation 2.5.** For every triangulation $T(V)$, we can always determine a point-based polygonal tessellation $PT(V)$ such that $T(V) = NRG(PT(V))$.

For instance, in Figure 8, for each triangle, we choose a point inside the triangle and the tiles are defined by joining these points and the center points of edges.

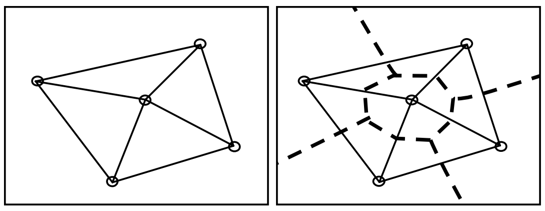


FIGURE 8. Triangulation of 5 points and a Tessellation (dotted lines).

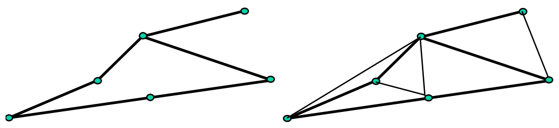


FIGURE 9. A constrained triangulation (right) of a given graph (left).

But in our problem, two vertices of an edge in the graph representing airspace must be considered as neighbors each other. It means that this edge belongs to the set of edges of NRG. What we need is then a *constrained triangulation.*

**Definition 2.6** [5]**.** Given a planar graph $G = (V, E)$, a *constrained triangulation*, denoted by $CT(G)$, with respect to $G$ is a triangulation $T(V)$ such that all edges of $E$ are edges of $T(V)$.

Figure 9 gives an example of a constrained triangulation.

A constrained triangulation can be obtained by adding edges that do not intersect any of existing edges, till no more new edges can be added. This technique has a poor complexity of $O(n^4)$. In [5], an $O(n \log n)$ constrained triangulation algorithm is described.

Now let us come back to our problem: given an airspace represented by $G = (V, E)$, we construct a constrained triangulation $CT(G)$ with respect to $G$. The airspace then will be sectorized into a number of subsets $V_i$ such that, for all $V_i$, the subgraph of $CT(G)$ corresponding to $V_i$ is connected. From this constrained triangulation, we determine the corresponding point-based polygonal tessellation, and the boundary for each sector is constructed by grouping the tiles of its vertices.

Note that for a given graph, we can obtain several constrained triangulations. In fact, in order to respect the connectivity constraint, a vertex $v_i$ can be in the subset $V_j$ if and only if it is connected (*via* the NRG, *i.e.* the constrained triangulation) with other vertices in $V_j$. An arbitrary choice of constrained triangulation may reduce the search space, so the quality of "best" solution. But it is difficult and outside the scope of this paper to determine which one is the "best".

## 3. A Constraint Programming Formulation for the ASP

We firstly give a brief overview of the principles of Constraint Programming (CP). For more details on CP and its application, we refer to [1, 3]. Constraint Programming is a paradigm aimed at solving Constraint Satisfaction Problems (CSP). An instance of the CSP is described by:

- a set of variables $X = x_1, ..., x_n$;
- for each variable $x_i$, a set $D_i$ of possible values (domain of variable);
- a set of constraints between the variables.

A *solution* of the CSP is an instantiation (assignment of values for all variables), such that all constraints are satisfied. Note that CSPs are decision problems; when one wants to optimize some objective function, a common technique to look for an optimal solution is to solve successive decision variants of the CSP.

CSPs can be solved as follows: a search tree is created and the variables are instantiated sequentially. Each node of the search tree represents a partial solution (partial assignment) and the algorithm attempts to extend it to a full solution by assigning a value to an uninstantiated variable. Whenever a partial solution violates a constraint, backtracking is performed.

The searching process may need a *search heuristic* that describes which decision is taken at a point of search. For instance, we can define a heuristic determining which variable to be instantiated with which value.

A key idea of CP is *Constraint Propagation*: when the domain of a variable is changed, the constraints are not only used to check the validity of solution, but they are also used to deduce new constraints, to remove inconsistent values of uninstantiated variables, so to reduce the search space.

In practice, CP tools such as ILOG Solver[25], PROLOG III, IV [6,7], ECLISPE [27], CHOCO [21] allow the users to create CSPs by defining variables and constraints. Constraints may be stated as one of pre-defined constraints (arithmetic constraints on integers, constraints on sets, ...) with corresponding propagation algorithms. But new constraints with particular constraint propagation algorithms may also be defined by the user. Furthermore, these tools also allow the users to specify their own specific search heuristics.

### 3.1. Variables and constraints

We partition the airspace $G = (V, E)$, $|V| = n$, $|E| = m$, as mentioned in Section 1, with respect to the specific constraints and to the balance constraint, into $k$ subsets (sectors) $V_1, ..., V_k$, while minimizing the cut-size. To model ASP, we introduce $n$ variables $x_i$, which can take the value in $\{1, ..., k\}$ ($x_i = j$ means that the vertex $v_i$ is in the subset $V_j$). To simplify the definition of the objective function and the balance constraints, we introduce the following redundant variables:

| | | |
|---|---|---|
| $m$ variables $c_{ij} \in \{0, 1\}$ | where | $c_{ij} = 0 \Leftrightarrow x_i = x_j$ |
| $nk$ variables $y_{ij} \in \{0, 1\}$ | where | $y_{ij} = 1 \Leftrightarrow x_i = j$ and |
| | | $\forall i \in \{1, ..., n\}, \sum_{j=1}^{k} y_{ij} = 1.$ |

The *Objective Function* is now defined as follows:

$$\min \sum c_{ij}\omega_{ij}.$$

The *balance constraint* is given by

$$\forall j \in \{1, ..., k\}, (1-\delta)\frac{1}{k}\sum_{i=1}^{n}\omega_i \leq \sum_{i=1}^{n} y_{ij}.\omega_i \leq (1+\delta)\frac{1}{k}\sum_{i=1}^{n}\omega_i$$

and it is propagated thanks to well-known arc-B-consistency algorithms. To state the specific constraints, we assume that for each flight $i$, the *flight plan* is given as an ordered list of vertices $(v_1^i, v_2^i, ..., v_p^i)$.

The *Minimum Sector Crossing Time Constraint* states that aircraft must stay in each crossed sector at least a given amount of time. We can then deduce the minimum distance $l_{1-\min}$ that the aircraft has to perform in each crossed sector.

Given a fight plan $F^i = (v_1^i, ..., v_l^i)$, a value $j \in D(x_c^i)$, $1 \leq c \leq l$ is consistent for the Minimum Sector Crossing Time Constraint along $F^i$ if and only if there are $p$, $q$ such as:

$$\begin{array}{l} 1 \leq p \leq c \leq q \leq l \\ \forall a \in [p,q]: \quad j \in D(x_a^i) \;\; et \;\; \displaystyle\sum_{a=p}^{q} l(v_a^i) \geq l_{1-\min} \end{array}$$

where $l(v_p^i) = \frac{1}{2}(l(v_{p-1}^i, v_p^i) + l(v_p^i, v_{p+1}^i))$ is the pseudo-length of the vertex $v_p^i$.

Now for each flight plan, we define a global constraint who relates all its variables. And an propagation algorithm will be called whenever a domain of a variable related to the global constraint is modified, to eliminate any value that does not satisfy (*inconsistant* with) this constraint.

The *Minimum Distance Constraint* means that the distance between a sector border and a vertex must be at least a given distance $l_{2-\min}$. Thus, if we have two vertices such that the corresponding edge has a length less than $2.l_{2-\min}$, they must be in the same sector. Let $l(v_a^i, v_b^i)$ be the length of the edge $(v_a^i, v_b^i)$ belonging to the flight plan $i$, this constraint can be stated as:

$$\forall i, \forall v_a^i, v_b^i : l(v_a^i, v_b^i) \leq 2.l_{2-\min} \Rightarrow x_a^i = x_b^i.$$

The *Route-based Convexity Constraint* states that aircraft can not enter the same sector twice. Given a flight plan $i$ and a triplet $x_a^i$, $x_b^i$, $x_c^i$ , where $a < b < c$, if $x_a^i$ and $x_c^i$ are already in the same sector (have the same value), we can then deduce that $x_b^i$ is also in this sector:

$$\forall i, \forall a < b < c : \quad x_a^i = x_c^i \;\Rightarrow\; x_a^i = x_b^i.$$

But if we state this constraint for all flight plans and for all triplets $a < b < c$, the number of constraints can be large. More practically, for each flight plan $i$, we can define a global constraint $C_i$ and the following propagation algorithm based on the semantic of this constraint is called whenever a variable $v_a$ related to $C_i$ is instantiated:

- If there is a variable "before" $v_a$ in the constraint $C_i$ that is instantiated to a value $p$ which differs from the value of $v_a$, it means that the aircraft has entered another sector, so the aircraft can not return to the previous sector and hence, we can remove $p$ from the domain of all variables after $v_a$.
- If a variable $v_b$ $(b \neq a)$ is instantiated and has the same value as $v_a$, then all variables between them are instantiated to this value.

Finally, the *Connectivity Constraint* ensures that the sectors are not fragmented. As mentioned above we have an airspace $G = (V, E)$ and a constrained triangulation $CT(G)$ (see Sect. 2). The airspace then will be sectorized into $k$ subsets $V_j$ and the connectivity holds if all subgraphs $V_j$ of $CT(G)$ are connected. We define a global constraint to ensure the connectivity. Our propagation algorithm is applied whenever a vertex is assigned to a sector $p$:

- We first compute the set of variables $V_p = \{x_i | p \in D_i\}$ which can take the value $p$ and determine the connected components of the subgraph of $CT(G)$ corresponding to $V_p$:

$$V_{p1} \cup V_{p2} \cup ... \cup V_{pk} = V_p.$$

- No more than one of these subsets belongs to sector $p$ hence, if a variable $x_i$ in a subset $V_{pl}$ takes already the value $p$, then $p$ can be removed from the domain of the variables of all other subsets.

The first step can be performed in $O(n)$; the second one is in $O(m)$ by the Tarjan algorithm [26] and the last one in $O(n)$. We check the connectivity for all partitions, the algorithm is then in $O(k.m)$

## 3.2. Heuristic for variable and value selecting

Inspired from the notion of gain of the Kernighan/Lin algorithm for Graph Partitioning, we propose a heuristic for variable and value ordering, which can reduce significantly the complexity of backtrack search.

Let $X : V \to \{1, ..., k\}$ be the partitioning vector of the graph $G = (V, E)$ ($X(u) = i$ means that the vertex $u$ is in the partition $V_i$) and $\omega_{uv}$ be the weight of edge $(u, v)$. The internal cost and the external cost of the vertex $u$ are defined as follows (see example in Fig. 10):

$$\mathrm{int}(u) = \sum_{(u,v) \in E, X(u)=X(v)} \omega_{uv}; \quad \mathrm{ext}(u) = \sum_{(u,v) \in E, X(u) \neq X(v)} \omega_{uv}.$$

Then, the gain of moving a vertex $u$ from its partition to the other is given by:

$$g(u) = \mathrm{ext}(u) - \mathrm{int}(u).$$

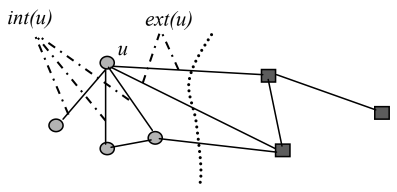


FIGURE 10. Internal and external cost.

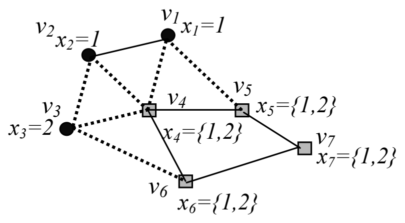


FIGURE 11. Temporary cut-edges.

Now, based on this notion, we introduce the *estimated gain* for each uninstantiated variable and for each value in its domain. Given a partial instantiation, the set $V$ of the graph's vertices is partitioned into two subsets: $V_{\text{known}}=\{v_i|D_i \textit{ is singleton}\}$ and $V_{\text{unknown}} = V \setminus V_{\text{known}}$. The *temporary cut-edge set* $C^t$ is defined as follow:

$$\begin{array}{rcl} C^t = \{(v_i,v_j) & | & (v_i,v_j) \in E, \\ & & (D_i \neq D_j) \quad \vee \quad (v_i \in V_{\text{known}}, v_j \notin V_{\text{known}}) \\ & & \qquad\qquad\quad \vee \quad (v_j \in V_{\text{known}}, v_i \notin V_{\text{known}})\}. \end{array}$$

Informally speaking, for every edge $(v_i, v_j)$: if the domains of corresponding variables $x_i$ and $x_j$ are different, we are sure that the edge belongs to the cut-edge set; if one of $x_i$ and $x_j$ is instantiated, we consider the edge to be in the cut-edge set; in other cases, we ignore it. In Figure 11, $C^t = \{(v_2, v_3), (v_1, v_4), (v_1, v_5), (v_2, v_4), (v_3, v_4), (v_3, v_6)\}$.

We define an estimated internal cost and an estimated external cost for each vertex $v_i$ such that $x_i$ is not yet instantiated, and for each value *val* in its domain $D_i$, as follows:

$$\begin{array}{rcl} int^*(v_i, val) & = & \{(v_i, v_j)|(v_i, v_j) \in E, x_j = val\} \\ ext^*(v_i, val) & = & \{(v_i, v_j)|(v_i, v_j) \in E, v_j \notin V_{\text{known}}\}. \end{array}$$

The $int^*(v_i, val)$ is the subset of $C^t$ which become *internal* edges (and can be removed from $C^t$) if vertex $v_i$ is put in partition $V_{val}$; while the $ext^*(v_i, val)$ is the set of new edges which will belong to $C^t$.

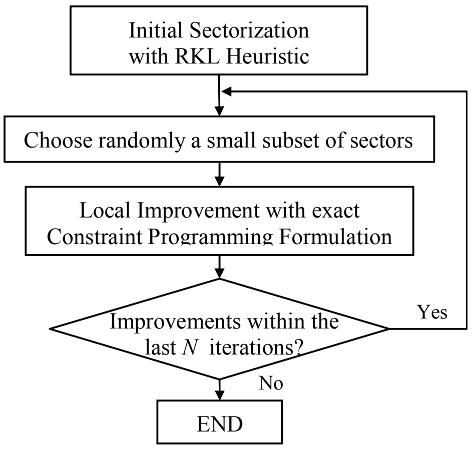

Initial Sectorization with RKL Heuristic
Choose randomly a small subset of sectors
Local Improvement with exact Constraint Programming Formulation
Improvements within the last N iterations?
Yes
No
END

FIGURE 12. A two-phase approach for airspace sectorization.

So, the estimated gain, if the variable $x_i$ is instantiated to $val$ (vertex $v_i$ is in partition $V_{val}$) is:

$$g^*(x_i, val) = \sum_{(v_i,v_j)\in int*(v_i,val)} \omega_{ij} - \sum_{(v_i,v_j)\in ext*(v_i,val)} \omega_{ij}.$$

For instance, in the previous example, $g^*(x_4, 1) = (\omega_{14} + \omega_{24}) - (\omega_{45} + \omega_{46})$; $g^*(x_4, 2) = \omega_{34} - (\omega_{45} + \omega_{46})$.

We use the following heuristic at each node of the search tree: the variable with maximal estimated gain is chosen. It is instantiated to the value leading to the largest estimated gain.

## 4. A two-phase approach

Although the above CP formulation gives good results as reported in the next section, it can not find an optimal solution for large size instances.

Therefore we propose a two-phase approach: firstly, we try to find a good solution, and then, we re-optimize it locally with our efficient CP algorithm. The behavior of our approach is illustrated in Figure 12.

### 4.1. Finding an initial solution

Given a huge airspace, which has to be partitioned into $k$ sectors, we can not find directly an optimal solution. We must firstly find a good initial solution. This phase is performed by using a recursive bisection scheme. To handle the case in which the number of sectors $k$ is not a power of 2, we can use the *unbalanced* recursive bisection: the graph representing the airspace is at first divided into two unbalanced subsets, the first one has a desired weight of a sector (can be computed from a given $k$) and the second one is the remaining. We fix the first subset and continue to bisect the second one, and so on.

At each step of the bisection, an initial solution can be found with the constraint programming formulation (however, it is far from optimal yet) and then improved by using the idea of the Kernighan/Lin (KL) heuristic. The KL heuristic is used to improve locally a solution of graph bisection. The algorithm takes as input an initial solution $V = V_1 \cup V_2$, and tries to find a sequence of vertices pair exchanges that leads to a better solution.

Let $g(u)$ and $g(v)$ be the gains of vertices $u \in V_1$ and $v \in V_2$ if we move them from their partition to the other, and $\omega_{uv}$ be the weight of edge $(u, v)$. The gain of exchanging these vertices is:

$$g(u, v) = g(u) + g(v) - 2\omega_{uv}.$$

An iteration of the KL algorithm (called a pass) is as follows [20]:

```
procedure **KLPass**($V1, V2$) : boolean {
    Unmark and compute gain for all vertices;
    $i := 1$;
    Repeat {
      Select $u^i \in V_1$, $v^i \in V_2$ such that $g(u^i, v^i)$ is max;
      Mark $u^i$ and $v^i$;
      Update gain for all unmarked vertices such
      as $u^i$, $v^i$ have been exchanged;
      $i = i + 1$;
    }
    Until all vertices of $V_1$ or $V_2$ are marked;
    Choose $j$ such that $G = \sum_{i=1}^{j} g(u^i, v^i)$ is max;
    If $(G > 0)$ Then {
      Move $u^1..u^j$ to $V_2$, $v^1..v^j$ to $V_1$;
      Return true; }
    Else   Return false;
}
```

In a KL pass, firstly we unmark and compute the gains for all vertices. In each step $i$, we find an unmarked pair $u^i \in V_1$, $v^i \in V_2$ such that the gain $g(u^i, v^i)$ of exchanging $u^i$ and $v^i$ is maximum (it may be negative). We mark $u^i$ and $v^i$ and update the gain values of all remaining unmarked vertices such as $u^i$ and $v^i$ have been exchanged. Repeat this procedure of pair selecting until all vertices in one of $V_1$, $V_2$ are marked. Now we have an ordered list of pairs $(u^i, v^i)$, and we find the index $j$ such that $\sum_{i=1}^{j} g(u^i, v^i)$ is maximal. If this sum is positive, we perform

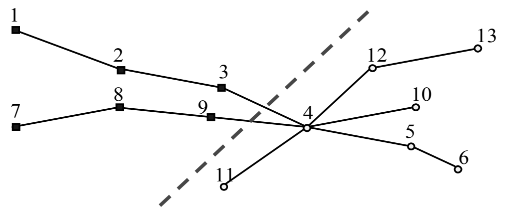


FIGURE 13. Example for RKL Heuristic.

the exchanges of $j$ vertices pairs and the result of this pass can be taken as input of another pass. Otherwise, we end the algorithm.

```
procedure KLBisection(Graph $G = (V, E)$){
    Find an initial bisection $V = V_1 \cup V_2$;
    Repeat {changed := KLPass($V_1, V_2$)}
    Until not(change)
}
```

In the KL algorithm, the search of a sequence of vertices pairs to be exchanged is performed for all vertices of the graph. But the exchange of an arbitrary vertices pair could violate our ATC constraints. We propose so a Restricted Kernighan/Lin (RKL) heuristic for airspace bisection: at each step, we find a set of vertices such that their moves do not violate the ATC constraints; furthermore, the validity of each exchange is verified before it is performed.

Consider the graph in Figure 13 for example, we have three flight plans: (1, 2, 3, 4, 5, 6), (7, 8, 9, 4, 10) and (11, 4, 12, 13). We can distinct two cases as follows:

- If all the vertices of a flight plan are in the same sector, as (11, 4, 12, 13), it is easy to see that only its two extremities, 11 and 13 in this case, can be moved to other sector without violation of the convexity constraint and without violation of connectivity constraint if they are connected to the other sector by the edges of constrained triangulation. We put them in a set of *potential vertices*. The others must be put in a set *unchangeable vertices*.
- If the flight plan crosses the border of sectors, as (1, 2, 3, 4, 5, 6) and (7, 8, 9, 4, 10), only two extremities of the cut route can be exchanged and then can be put in the potential vertices set (but note that these two vertices can not be exchanged directly). We have three vertices 3, 4 and 9 for the case.

So, we find a pair to be exchanged only among the potential vertices, but which are not unchangeable.

Because of minimum sector crossing time and minimum distance constraints, some vertices must be in the same sector. To meet these constraints, we introduce the notion of *cluster of exchange*: if $v_1, v_2 ... v_q$ must be in the same sector, we call them a cluster. Naturally, vertices of a cluster must be moved all at once and, if one of them is unchangeable, so is the cluster.

And at last, the CP formulation is used to validate the new solution after all exchanges.

### 4.2. Random local re-optimization

To improve the solution obtained in the previous step, we propose a random local re-optimization scheme as follows. We repeat to choose randomly a group of adjacent sectors and use the constraint programming formulation to find its optimal solution. The procedure ends if after predefined $N$ consecutive iterations, the solution is not longer improved.

## 5. Experimental results

The constraint based model has been implemented with the constraint programming library CHOCO[22] in language CLAIRE [4]. All programs have been run on a PC Athlon 2000+, 1GB RAM, under Windows XP. For our experimental study, we have generated several classes of graphs; each class containing 50 problem instances with the same number of vertices. At first, some vertices representing airports are generated, the coordinates are uniformly distributed. We have an edge between two vertices if there is at least a flight between these two airports. The probability of existence of an edge between two vertices is uniform and number of flights passing this edge is also from uniform distribution $n \in [1, 200]$. We then compute all crossing points and treat them as vertices of the graph $G$. If the number of vertices does not correspond to the desired number, the graph is "rejected" and the random graph generation procedure is run again. The values of the parameters relates to the specific constraints have been chosen to be close to real life situations.

Table 1 and Figure 14 report the results obtained when we try to find the optimal solution of the bisection problem ($k = 2$). In these results, it is clear that, as far as bisection is concerned, the model is applicable for up to 100 vertices instances, with a reasonable execution time.

Table 2 reports the execution time to find first solution of the partitioning 100 vertices into 8 sectors, 200 vertices into 16 sectors, 500 vertices into 40 sectors and 1000 vertices into 80 sectors. It shows also the performance of the re-optimization phase with $N = 5, 10, 15, 20$ and at each iteration, we re-optimize two neighbor sectors with the optimal bisection. Avg CPU time is the average execution time to find a first solution. Avg 1st cutsize and Avg re-optimized cut-size are respectively the average cutsize of the first solution and the average cutsize of the solution re-optimized. In this result, we can see that, the bigger $N$ is, the more the cutsize is improved, but in most cases, $N = 20$ is sufficient so that the cutsize approaches the minimum.

And at last, Figure 15 give us a view of a 500-vertices graph which is sectorized into 40.

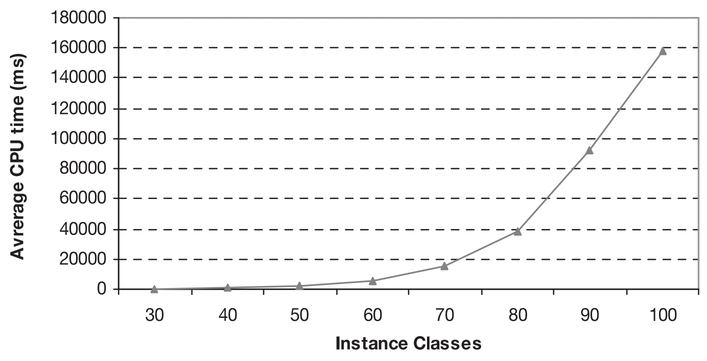


FIGURE 14. Average CPU times for finding the optimal bisections.

TABLE 1. Average backtracks and Average CPU time for an Optimal Bisection.

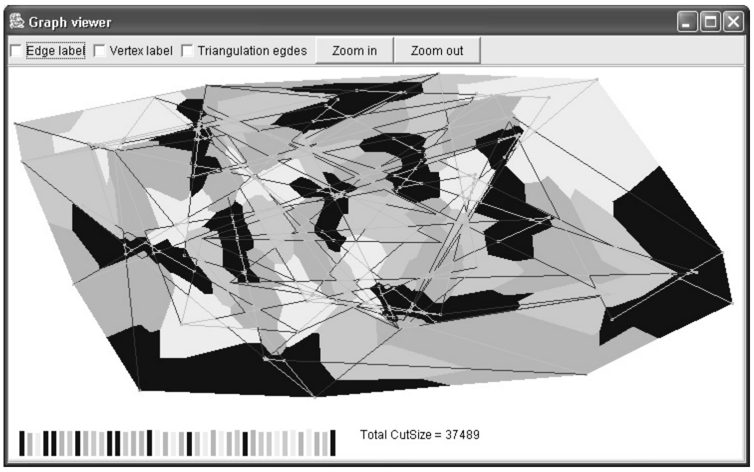
| Instance classes | Average cutsize | Average Backtracks | Average CPU time (ms) |
|---|---|---|---|
| 30 | 659 | 352 | 167 |
| 40 | 739 | 934 | 604 |
| 50 | 823 | 2599 | 2173 |
| 60 | 829 | 4375 | 5370 |
| 70 | 921 | 9870 | 15 191 |
| 80 | 886 | 18 228 | 38 231 |
| 90 | 987 | 39 295 | 92 356 |
| 100 | 1019 | 56 144 | 158 015 |

TABLE 2. Experimental results of finding first solutions and re-optimizations.

| Class/k | Avg 1st | Avg CPU | Avg reoptimized cutsize CPU (ms) | | | | % reduction |
|---|---|---|---|---|---|---|---|
| | cutsize | time (ms) | $N = 5$ | $N = 10$ | $N = 15$ | $N = 20$ | $(N = 20)$ |
| 100/8 | 6909 | 524 | 5432 | 4689 | 4617 | 4615 | 33 |
| 200/16 | 15 349 | 6283 | 13 240 | 11 584 | 10 976 | 10 638 | 31 |
| 500/40 | 39 114 | 95 254 | 37 046 | 33 359 | 31 296 | 30 041 | 23 |
| 1000/80 | 40 632 | 44 1860 | 39 905 | 37 497 | 35 185 | 35 092 | 14 |

## 6. Conclusion

In this paper, we have proposed a constraint programming formulation to optimize the airspace sectorization that satisfies all the specific ATC constraints. Based on the notion of gain of the Kernighan/Lin heuristic for Graph Partitioning, we have defined a heuristic for variables and values ordering while searching solutions. A restricted Kernighan/Lin heuristic is also proposed to improve the initial solution of a bisection. This formulation can find optimal solution for small



Total CutSize = 37489

FIGURE 15. A 500-vertices graph sectorized into 40.

size instances of the problem. For larger instances, we use a two-step approach: find an initial solution and then re-optimize it locally. This model is implemented and has been tested on several sets of data. The initial results are promising since a 1000-vertices graph can be sectorized into 40 sectors within 8 minutes. However, to improve and extend the model to even more realistic situations, many of prospects should be taken into account, such as parallelization of the search for solution, optimization of the geometrical form of the sectors, taking into account the third dimension of airspace, the military zones...

## REFERENCES

[1] S. Barbara, A tutorial on constraint programming. Technical Report 95.14, School of Computer Studies, University of Leeds (1995).

[2] S.T. Barnard and H.D. Simon, A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems, in *Proc. of The sixth SIAM conference on Parallel Processing for Scientific Computing* (1993) 711–718.

[3] R. Bartak, Online guide to constraint programming, `http://ktilinux.ms.mff.cuni.cz/bartak/constraints/` (1998).

[4] Y. Caseau, F.X. Josset and F. Laburthe, CLAIRE: Combining sets, search and rules to better express algorithms, in *Proc. of the 16th International Conference on Logic Programming*, Las Cruces, New Mexico – USA, Nov.–Dec. (1999).

[5] J. Chen, *Computational geometry: Methods and applications.* Department Computer Science, Texas A&M University (February 1996).

[6] A. Colmerauer, An introduction to PROLOG-III. *Commun. ACM* **33** (1990) 69–90.

[7] A. Colmerauer, *Les bases de Prolog IV.* Technical report, Laboratoire d'Informatique de Marseille (1996).

[8] D. Delahaye, *Optimisation de la sectorisation de l'espace aérien par algorithmes génétiques.* Ph.D. Thesis, École Nationale Supérieure de l'Aéronautique et de l'Espace (1995).

[9] D. Delahaye, J.M. Alliot, M. Schoenauer and J.L. Farges, Genetic algorithms for automatic regrouping of air traffic control sectors, in *Proc. of the Fourth International Conference on Evolutionary Programming*, MIT Press (March 1995) 657–672.

[10] D. Delahaye, M. Schoenauer and J.M. Alliot, Airspace sectoring by evolutionary computation, in *IEEE International Congress on Evolutionary Computation* (1998).

[11] C.M. Fiduccia and R.M. Mattheyses, A linear time heuristic for improving network partitions, in *Proc. of 19th ACM/IEEE Design Automation Conference* (1982) 175–181.

[12] P.O. Fjallstrom, Algorithms for graph partitioning: A survey. *Linkoping Electronic Articles in Computer and Information Science* **3** (1998).

[13] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman & Co. (1979).

[14] J.R. Gilbert, G.L. Miller and S.H. Teng, Geometric mesh partitioning: Implementation and experiments. *SIAM J. Sci. Comput.* **19** (1998) 2091–2110.

[15] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley (1989).

[16] B. Hendrickson and R. Leland, An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM J. Sci. Comput.* **16** (1995) 452–469.

[17] B. Hendrickson and R. Leland, A multilevel algorithm for partitioning graphs, in *Proc. of the 1995 ACM/IEEE Conference on Supercomputing*, San Diego, California, USA, ACM Press (1995).

[18] G. Karypis and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* **20** (1998) 359–392.

[19] G. Karypis and V. Kumar, Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing* **48** (1998) 96–129.

[20] B.W. Kernighan and S. Lin, An efficient heuristic procedure for partitionning graphs. *BELL System Technical Journal* (February 1970) 291–307.

[21] F. Laburthe, *CHOCO – A Constraint Programming kernel for solving combinatorial optimization problems* (2002).

[22] F. Laburthe and the OCRE group, *CHOCO: Implementing a CP kernel*, in *CP00 Post Conference Workshop on Techniques for Implementing Constraint programming Systems (TRICS)*, Singapore (2000).

[23] S. Manuel, M.L. José, M.B. Victor and M.R. José, Genes: a genetic algorithms and fast time simulation, in *3rd ATM R&D Symposium*, Spain (2002).

[24] A. Okabe *et al.*, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* New York, Wiley (1992).

[25] J.-F. Puget, A C++ implementation of CLP, in *Proc. of the Second Singapore International Conference on Intelligent Systems*, Singapore (1994).

[26] R.E. Tarjan, Depth-first search and linear graph algorithms. *SIAM J. Comput.* **1** (1972) 146–160.

[27] M. Wallace, S. Novello and J. Schimpf, *Eclipse: A platform for constraint logic programming.* Technical report, IC-Parc, Imperial College, London (1997).