OPTIMIZED SECTORIZATION OF AIRSPACE WITH CONSTRAINTS

Huy Trandac, Philippe Baptiste - Heudiasyc Laboratory, UMR CNRS 6599, University of Technology of Compiègne, Centre de Recherches de Royallieu, BP 20529, F-60205 Compiègne cedex, France.

Vu Duong - Eurocontrol Experimental Centre, Centre de Bois des Bordes, BP15, F-91222 Bretigny sur Orge cedex, France.

Abstract*—In this paper we consider the Optimized Airspace Sectorization Problem (ASP) with constraints in which a given airspace is to be partitioned into a number of sectors. The objective of ASP is to minimize the coordination workload between adjacent sectors. We proposed a constraint programming approach to optimize the sectorization that shall satisfy all specific constraints e.g. the controllers' workload is balanced among the sectors, the sectors are not fragmented, aircraft can not enter twice the same sector; aircraft cannot stay less than a given amount of time in each sector crossed, sectors cannot be fragmented etc.*

Introduction

Sectorization is a fundamental architectural feature of the Air Traffic Control (ATC) system. The airspace is divided into a number of sectors, each of them is assigned to a team of controllers (Control Positions). Controllers of a given sector have (1) to monitor the flights, (2) to avoid conflicts between aircraft and (3) to exchange information with adjacent sectors where aircraft have planned to go. These tasks induce a workload which is often decomposed into three corresponding parts [1, 2, 3]:

- The **monitoring workload (MW)** comes from the cyclic checking of aircraft trajectories.
- The **conflict workload (CW)** results from resolution and avoidance of conflicts between aircraft.
- The **coordination workload (OW)** is basically related to the exchanges that have to be performed between controllers of adjacent sectors and pilots of aircraft that are crossing through.

But the air traffic changes over the day. This often leads to workload imbalance between the sectors. Furthermore, it is desirable that there are more the sectors (then more control positions) in the dense traffic periods of the day than the weak periods. Hence a tool to "dynamically" re-sectorize the airspace (more precisely a part of airspace – e.g. the sectors of a Air Traffic Control Center) is required to cope with the evolution of the traffic.

When the sectors are designed, not only the balance constraint must be hold (in term of workload), but also that several following specific constraints have to be taken into account:

- *Convexity constraint.* The same aircraft can not enter twice the same sector. It is not sensible, but it happened in the past, *e.g.* national boundaries in European airspace. For instance, the following case in the Figure 1 is not admissible:

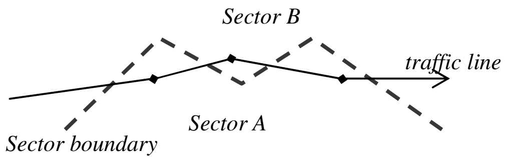


Figure 1: Convexity Constraint

- *Minimum distance constraint*. The distance between a sector border and a network node must be not less than a given distance (see Figure 2). This constraint ensures that the controller has enough time to solve conflicts which may occur at this node.

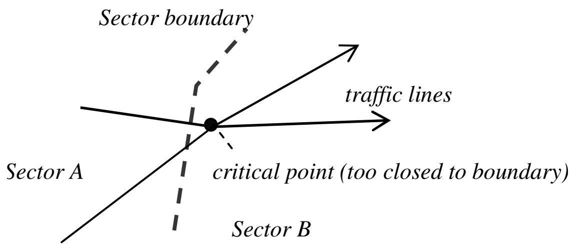


Figure 2: Minimum Distance Constraint

- *Minimum sector crossing time constraint.* The aircraft must stay in each crossed sector at least a given amount of time T_{min} (see

Figure 3). This constraint ensures the controller has enough time to control the aircraft.

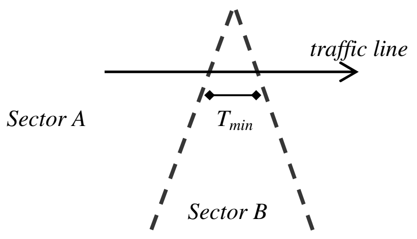


Figure 3: Minimum Sector Crossing Time Constraint

- *Connectivity constraint.* The sector can not be fragmented. For example, the solution in Figure 4 is not feasible.

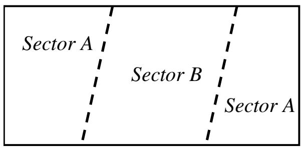


Figure 4: Connectivity Constraint

It is easy to see that, when the airspace is sectorized, more the routes are cut, more the coordination workload is induced. Hence, the objective of the optimization of Airspace Sectorization Problem (ASP) is to minimize the sum of cut routes.

A genetic algorithm [4] to solve ASP has been proposed in [1]. Chromosomes are defined as sets of sectors' center points; the sector is then defined as the Voronoï diagram [5] associated to the set of center points (*i.e.*, a sector is a set of points that are closer to its central point than to any other center points). Voronoï-like sectors are geometrically convex but in practice, sectors convex in the sense of routes (the same route does not cross the same sector twice). Hence a Voronoï-like sectorization might be sub-optimal. Furthermore, sectors built by the Voronoï diagram may lead to unfair load distribution.

Delahaye et al. [3] 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 again, sectors built by synthesis of connected vertices do not ensure the convexity constraint.

More recently, airspace has been divided in small volume units and a sector is obtained by joining some of these elementary units [6]. Unfortunately, the most specific constraints can not be taken into account and for instance, the sectors can be fragmented in the solution.

Our initial investigation on this problem has been published in [7], but without the connectivity constraint. In this paper, we introduce a constraint-programming formulation to solve ASP. Our goal is to take into account all geometrical constraints, as defined in the next section, when building sectors. Our approach includes a heuristic for variables and values ordering based on the notion of gain of Kernighan/Lin heuristic [8] for Graph Partitioning Problem. With this model, we can compute optimal solutions for small size instances of ASP. For the large size instances, we use a two-phase approach: firstly, we apply a restricted Kernighan/Lin (RKL) heuristic to find a "good" solution; and in the second phase, we enter a re-optimization loop, relying on the constraint programming model, that improves this solution.

Modeling

In this section, we firstly propose a discrete model for ASP. The airspace is modeled by a valuated graph and a sector in a solution of ASP is defined as a set of vertices, without geometrical boundaries. And secondly, we propose a way to compute the sectors' boundaries from a solution of ASP.

Airspace Sectorization and Graph Partitioning

We rely on the following model: Airspace is made of routes that cross each other. In the following, $G = (V, E)$ denotes the graph representing the airspace, where:

- V (the set of vertices) is the set of beacons and crossing points u,
- and E (the set of edges) is such that (u, v) belongs to E if and only if there is a direct route from u to v.

The graph G is valuated both on its vertices and edges as follows (see Figure 5):

- c_v : conflict workload, induced by the conflicts that occur at v, is assigned to v
- m_e : monitoring workload belongs to an edge $e=(u,v)$. It is divided in two equal parts $m_u=m_v=m_e/2$ that are assigned to u and v
- o_e : coordination workload assigned to the edge e. This workload is set to 0 if the vertices of the edge are in the same sector.

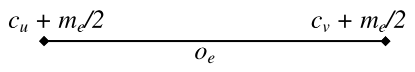


Figure 5: Graph Valuation

For practical reasons, it is much more convenient to have a static estimation of each of the workloads. For instance, we can consider that coordination workload of an edge is proportional to the number of aircraft passing this edge and monitoring workload is proportional to total time that aircraft fly along the edge, *etc*... Achieving more precise estimation is a very complex task and it is beyond the scope of this paper.

We follow the idea of [3] 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 that we now work on a purely discrete problem which is very similar to pure weighted Graph Partitioning problem, where the set V of valuated vertices will be partitioned into k subsets such that (1) the weights of subsets are bounded by given minimum and maximum weights (2) the sum of cut-edges (cut-size) is minimized.

The Graph Partitioning problem has been widely studied. It should be observed that this problem is NP-complete [9]. Very efficient heuristic procedures have been developed in the last 30 years and very large problems can be solved efficiently in a reasonable amount of CPU time. For more details on Graph Partitioning, we refer [8, 10, 11, 12, 13, 14, 15, 16, 17, 18]

Because our problem is slightly different from a pure graph partitioning problem, we rely on a constraint programming formulation which can help us to find a solution that satisfies the specific constraints. This formulation will be presented in the next section, but we consider firstly the sectors' boundaries computation and the sector connectivity problem.

Sectors Boundaries Computation and Sector Connectivity

Suppose that we have an airspace sectorized into a number of sectors, each made of a set of vertices, we must then compute non-overlapped sectors' boundaries such that each sector boundary contents all its vertices and, as mentioned above, a sector boundary can not be fragmented. For example in the 2D case, we must determine for each sector one simple polygon which contents all constituent vertices. We propose then a way to compute such sectors' boundaries for the case of 2D airspace. Note that it can be easily extended for 3D case.

Definition: A *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-overlapped polygons, called *tiles*, such that each polygon $P(v)$ contains exactly one point $v \in V$.

Definition: The *Neighbouring Relative Graph* of a $PT(V)$, denoted by $NRG(PT(V))$, is the graph constituted by $\{V,E\}$, where edge (u,v) belongs to E if and only if $P(u)$ and $P(v)$ have at least one common side.

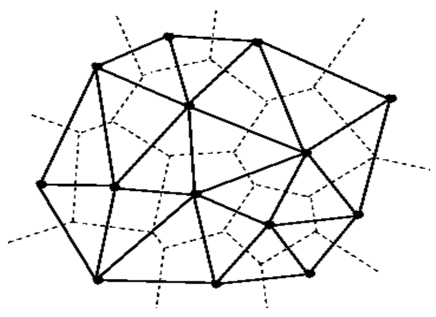


Figure 6: Delauney Triangulation (dashed line) and Voronoi Diagram

Example: 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 [5] (the term *triangulation* is defined in the next), is created by connecting all generating points which share a common tile edge. Thus formed, the triangle edges are perpendicular bisectors of the tile edges. Hence, the Delauney triangulation is the NRG of Voronoi Diagram (see Figure 6)

Proposition: given a polygonal tessellation $PT(V)$ of a set of points V in a plane and the corresponding $NRG(PT(V))$. For all subset $V_i \subset V$, if a sub-graph of $NRG(PT(V))$ corresponding to V_i is connected, then there exit a simple polygon which contains all points of V_i, but not any point in $V \setminus V_i$.

The proof of this proposition is given by construction of such polygon: we group the tiles corresponding of all points in V_i and remove all shared sides. Since each tile contains only one point of V by the definition of $PT(V)$, so this polygon contains only the points of V_i, but not any point in $V \setminus V_i$

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 polygonal tessellation of a set of points in a plane.

Definition: 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 every regions are triangles.

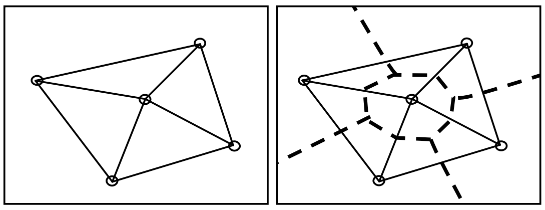


Figure 7: Triangulation of 5 points and A Tessellation (dotted lines)

Observation: for every triangulation $T(V)$, we can always determine a polygonal $PT(V)$ such that $T(V)=NRG(PT(V))$. For instance, in the Figure 7, for each triangle, we choose a point inside the triangle and tiles are defined by these points and the centers points of edges.

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: given a planar graph $G=(V,E)$, a *constrained triangulation,* denoted by $CT(V)$, with respect to G is a triangulation $T(V)$ such that all edges of E are edges of $T(V)$. Figure 8 gives an example of a constrained triangulation.

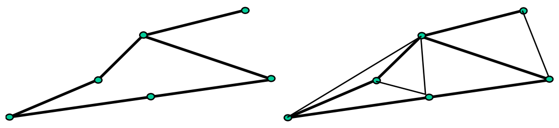


Figure 8: A constrained triangulation (right) of a given graph (left)

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 [19], we can find an algorithm for constrained triangulation in O($n$log$n$).

Now let back to our problem: given an airspace represented by $G=(V,E)$, we construct a constrained triangulation $CT(V)$ with respect to G. The airspace then will be sectorized into a number of subsets V_i such that, for all V_i, the sub graph of $CT(V)$ corresponding to V_i is connected.

Note that for a given graph, we can obtain different constrained triangulations. It is difficult to known which one is the best. We must be content to use just one of them to ensure the connectivity constraint while finding a solution for ASP.

A Constraint Programming Formulation for ASP

Constraint Programming

We first give a brief overview of the principles of Constraint Programming (CP). For more details on CP and its application, we refer [20, 21, 22, 23, 24, 25, 26, 27, 28].

Constraint Programming is a paradigm aimed at solving Constraint Satisfaction Problems (CSP). An instance of the CSP is described by:

- a set of variables $X=\{x_1, x_2, .., x_n\}$,
- 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 tree search is created and the variables are instantiated sequentially. Each node of the tree search 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 any of constraints, 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 next variable and which its value is chosen to be instantiated.

A key idea of CP is *Constraint Propagation:* when a variable is instantiated, 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, to solve a CSP with CP tools such as ILOG Solver [29], PROLOG III, IV [30, 31], ECLISPE [32], CHOCO [33], users create the CSP by defining its variables and constraints among them. Constraints may be stated as one of pre-defined constraints (arithmetic constraints on integers, constraints on sets …) with corresponding propagation algorithms. But new constraints may be also defined by the user with particular constraint propagation algorithms. Furthermore, these tools also allow the users to specify their own specific search heuristics.

Constraint Programming Model for ASP

Let us get back now to our problem. Given our graph $G=(V,E)$, $|V|=n, |E|=m$, let ω_i be the weight of the vertex v_i and ω_{ij} be the weight of the edge (v_i,v_j). We partition V, with respect to the specific constraints, into k subsets (sectors) $V_1, V_2, .., V_k$ such that the balance constraint is met (the sum of the vertices in each subset is bounded by W_{min} and W_{max}) and the cut-size is minimized.

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 $c_{ij}=0 \Leftrightarrow x_i= x_j$

$n.k$ variables $y_{ij}\in\{0,1\}$ where $y_{ij}=1 \Leftrightarrow x_i=j$

$$\text{and } \forall i=1..n, \quad \sum_{j=1}^{k} y_{ij} = 1$$

Objective Function

The objective function is now defined as follows: $min \sum C_{ij}.\omega_{ij}$

Balance Constraint

The balance constraint is given by:

$$\forall j=1..k, \qquad W_{\min} \le \sum_{i=1}^{n} y_{ij}.\omega_i \le W_{\max}$$

To state the specific constraints, we assume that for each flight i, the flight plan is represented by a ordered list of vertices $(v_1^i, v_2^i, ..., v_{pi}^i)$.

Minimum Sector Crossing Time Constraint

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 $l_{1\text{-}min}$ that the aircraft has to perform in each crossed sector.

Now, for all triplet $a<b<c$, and for all the flight plans i, if the distance $l(v^i_a, v^i_c)$ of the edge (v^i_a,v^i_c) is less than a given $l_{1\text{-}min}$, the three vertices v^i_a, v^i_b, v^i_c can not be in three different sectors:

$$\forall i, \forall a<b<c: l(v^i_a, v^i_c) < l_{1\text{-min}} \Rightarrow (x^i_a= x^i_b)\vee(x^i_b= x^i_c)$$

Minimum Distance Constraint

This constraint means that the distance between a sector border and a network node must be at least a given distance $l_{2\text{-}min}$. Thus, if we have two nodes such that the corresponding edge has a length less than $2.l_{2\text{-}min}$, they must be in the same sector. Let $l(v^i_a, v^i_b)$ be the length of the edge (v^i_a, v^i_b) belonging to the flight plan i, this constraint can be stated as:

$$\forall i, \forall v^i_a, v^i_b : l(v^i_a, v^i_b) \le 2.l_{2\text{-min}} \Rightarrow (x^i_a= x^i_b)$$

Convexity Constraint

The convexity constraint states that, during the flight, aircraft can not enter twice the same sector. It is naturally to see that, for all flight plan i, for all triplet x^i_a, x^i_b, x^i_c , where $a<b<c$, if we know that x^i_a and x^i_c are already in the same sector (have the same value), we can then deduce that x^i_b is also in this sector:

$$\forall i, \forall a<b<c : \qquad x^i_a= x^i_c \Rightarrow x^i_a= x^i_b$$

But if we state this constraint for all flight plan and for all triplet $a<b<c$, number of constraints can be important. More practically, we can define, based on the semantic of the constraint and for each flight plan i, a global constraint concerning all its corresponding variables $(x_1^i, x_2^i, ..., x_{pi}^i)$. A constraint propagation algorithm for the convexity constraint is proposed as follows:

```
procedure ConvexConstraintPropag(C,idx){
  For all variable i before idx in C
      If val(i)<>val(idx) then
      {Remove val(i) from domains of all
         variables after idx in C}
      Else {Instantiate to val(idx) all
            variables between i and idx}
  For all variable j after idx in C
      If val(j)<>val(idx) then
       {Remove val(idx) from domains of
          all variables after j in C}
      Else {Instantiate to val(idx) all
           variables between idx and j}
}
```

Where *idx* is the index in the constraint *C* of the variable, which is just instantiated to the value *val(idx)*; *val(i)*is the value of variable *i*.

The procedure is called whenever one of variables concerning constraint C is instantiated. If there are variables before idx in the constraint C that have different values of $val(idx)$, it means that the aircraft has entered another sector, so the aircraft can not return to the traversed sectors. This leads us to remove these values from domains of all variables after idx. If there is any variable i before idx that has the same value of idx, then all variable between them must take also this value. We apply the same principle of deduction for all variables after idx.

Connectivity Constraint

This constraint ensures that the sectors are not fragmented. As mentioned above, given an airspace represented by $G=(V,E)$ and a constrained triangulation $CT(V)$ with respect to G. The airspace then will be sectorized into a number of subsets V_i and the connectivity is hold if all sub graphs V_i of $CT(V)$ are connected.

We will define a global constraint concerning all variable and propose an algorithm for connectivity constraint propagation when ever a variable is instantiated, for each partition p, as follow:

- Determine a set of variables which can take the value p:
  $V_p = \{x_i \mid p \in \text{Domain}(x_i)\}$
- Determine the connected components of the sub-graph of $CT(V)$ corresponding to V_p :
  $$V_{p1} \cup V_{p2} \cup \ldots \cup V_{pk} = V_p$$
- We can deduce that, only one of these subsets can take the value p (be in the partition P_p). So, if a variable x_i in a subset V_{pl} takes already the value p, then all variables in other subsets can not take this value. We can remove p from $Domaine(x_j)$, $\forall x_j \in V_{pm}$, $m \neq l$.

Heuristic for Variable and Value Selecting

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

Kernighan and Lin introduced one of the earliest graph partitioning algorithms in 1970 [8]. This algorithm is based on the notion of *gain* of a vertex by moving it from its partition to the other. Let $P: V \rightarrow \{1,..,k\}$ be the partitioning vector of the graph $G=(V,E)$ ($P(u)=i$ means that the vertex u is in the partition V_i) and ω_{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 Figure 9):

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

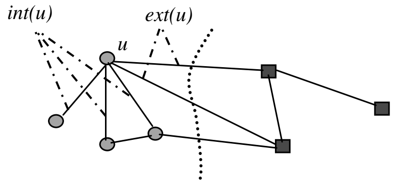


Figure 9: Internal and External Cost

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

$$g(u) = ext(u) - int(u)$$

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 graph's vertices is partitioned into two subsets: V_{known} is set of vertices v_i that the value of corresponding variable x_i is already known (instantiated) and $V_{unknown} = V \setminus V_{known}$. Let D_i be the domain of variable x_i, the *temporary cut-edge set* C^t is defined as follow:

$$C^t = \{(v_i,v_j) \mid (v_i,v_j) \in E, (D_i \neq D_j) \vee (v_i \in V_{known}, v_j \notin V_{known}) \vee (v_j \in V_{known}, v_i \notin V_{known})\}$$

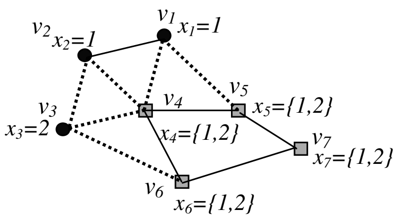


Figure 10: Temporary Cut-Edges

Informally speaking, for all 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 10, $C^t = \{(v_2,v_3), (v_1,v_4), (v_1,v_5), (v_2,v_4), (v_3,v_4), (v_3,v_6)\}$

We define *estimated internal cost* and *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 follow:

$$int^*(v_i,val) = \{(v_i,v_j) \mid (v_i,v_j) \in E, x_j = val\} \text{ and}$$

$$ext^*(v_i,val) = \{(v_i,v_j) \mid (v_i,v_j) \in E, v_j \notin V_{known}\}$$

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.

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.

A Two-Phase Approach

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

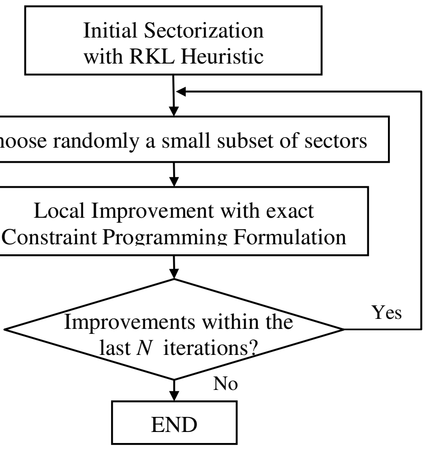

Initial Sectorization with RKL Heuristic
Choose randomly a small subset of sectors
Local Improvement with exact Constraint Programming Formulation
Improvements within the last N iterations?
Yes
No
END

Figure 11: A Two-phase Approach for Airspace Sectorization

Therefore we propose in the following 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 11.

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 recursive bisection schema.

At each step of 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 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 node pair exchanges that leads to a better solution.

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

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

An iteration of the KL algorithm (called a pass) is as follows.

```
procedure KLPass($V_1$,$V_2$) : boolean {
  Unmark and compute gain for all nodes;
    $i:=1$;
    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 nodes such
      as $u^i$, $v^i$ have been exchanged;
     $i$++;
    } Until all nodes 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 nodes. We find an unmarked pair $u\in V_1$, $v\in V_2$ such that the gain $g(u, v)$ of exchanging *u* and *v* is maximum (it may be negative). We mark *u* and *v* and update the gain values of all remaining unmarked nodes such as *u* and *v* have been exchanged. Repeat this procedure of pair selecting until all nodes 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 maximum. If this sum is positive, we perform the exchanges of *j* node 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 sequence of node pairs to be exchanged is performed for all nodes of graph. But the exchange of an arbitrary node 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 nodes such that their moves do not violate the ATC constraints; furthermore, the validity of each exchange is verified before it is performed.

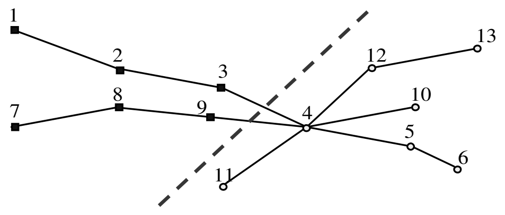


Figure 12: Example for RKL Heuristic

Consider the graph in Figure 12 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 nodes 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 nodes". The others must be put in a set "unchangeable nodes".

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 edge can be exchanged and then can be put in the potential nodes set (but note that these two nodes can not be exchanged directly). We have three nodes 3, 4 and 9 for the case.

So, we find a pair to be exchanged only among the potential nodes, 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 $v_1, v_2 \ldots v_q$ 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.

At last, the balance constraint must be verified before each exchange.

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 is stopped if after predefined *N* consecutive iterations, the solution is not longer improved.

Experimental Results

The constraint-based model has been implemented with the constraint programming library CHOCO [33] on top of CLAIRE [34]. All programs have been run on a PC Athlon 2000+, 512MB 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. An edge between two vertices means that there is at least a flight joining these airports. The probability of existence of edge between two vertices is uniform and number of flights passing this edge is also from uniform distribution $n \in [1,200]$. And then we calculate all crossing points of graph and consider them as vertices. The graph is taken if it has a desired number of vertices.

Table 1 and Figure 13 report the results obtained when we try to find the optimal solution of the bisection problem ($k=2$). In this table, Avg Bk and Avg CPU are respectively the average number of backtracks performed and required CPU time to solve an instance. %<4min is percentage of instances which can be solved within 4 minutes. In these results, it is clear that, as far as bisection is concerned, the model is applicable for up to 80 vertices instances, with a reasonable execution time.

Table 1: Experimental Results of Finding Optimal Solutions

Instance size	Avg Bk	Avg CPU(ms)	%< 4min
30	480	268	100
40	1278	1009	100
50	3472	3586	100
60	7173	10667	100
70	17232	30925	100
80	28986	71342	78

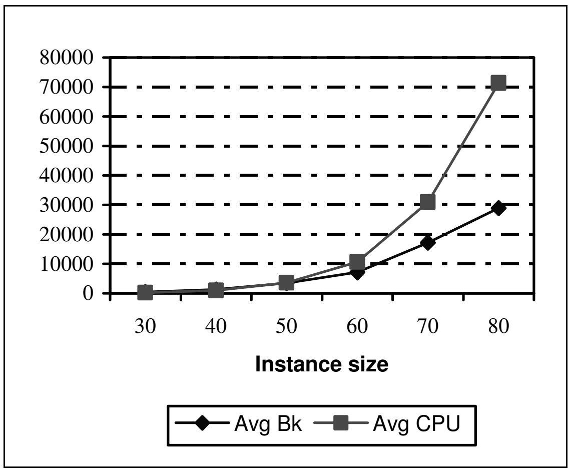
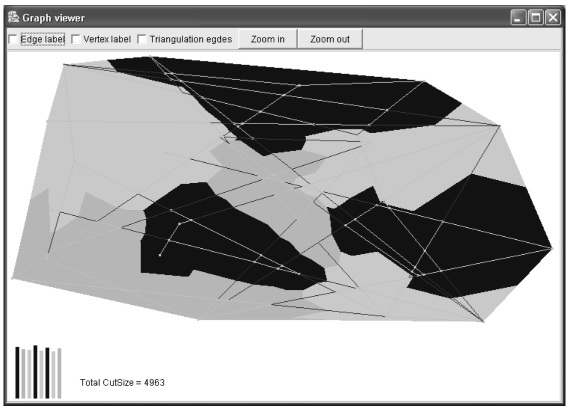


Figure 13: Average Bk and CPU Time for an Optimal Bisection

Table 2: Experimental Results of Finding First Solution and Re-Optimization

Class	Avg first CutSize	Avg re-optimized Cutsize	Avg total CPU (ms)	Avg % reduction
60/4	3203	2124	4107	34%
70/5	4157	2847	4818	32%
80/6	5161	3469	6192	33%
90/7	6043	4321	5503	28%
100/8	6884	4957	6363	28%
200/16	15349	11728	19196	24%
500/40	41217	34028	203912	17%



Total CutSize = 4963

Figure 14: A 100-Vertices Graph Sectorized Into 8

Table 2 reports the performance of the re-optimization phase and the execution time to partition 60 vertices into 4 sectors, 70 vertices into 5 sectors, …, and 500 vertices into 40 sectors. Avg first CutSize and Avg re-optimized CutSize are respectively the average cutsize of the first solution and the average cutsize of the solution re-optimized. Avg total CPU is the average total execution time (to find a first solution and to re-optimize it)

Conclusion

In this paper, we have proposed a constraint programming formulation to optimize the sectorization that satisfies all the specific constraints. Based on the notion of gain of 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 bisection. This formulation can find optimal solution for small 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 experimented with some randomized data. The initial results are promising, a 500-vertices graph can be sectorized into 40 sectors within 4 minutes. The study is pursued to improve and extend the model. One of next extensions is to take into account the flight levels to separate a "heavy" network node to several sub-nodes.

***Keywords**: Airspace Sectorization, Constraint Programming, Graph Partitioning*

References

[1] Delahaye D., 1995. *Optimisation de la sectorisation de l'espace aérien par algorithmes génétiques.* Doctorat Informatique ENSAE.

[2] Delahaye D., J.M. Alliot, M. Schoenauer and J. L. Farges, 1995. *Genetic Algorithms for automatic regroupement of Air Traffic Control sectors.* Proc. of the 4th Int. Conference on Evolutionary Programming.

[3] Delahaye D., M. Schoenauer and J. M. Alliot, 1998. *Airspace Sectoring by Evolutionary Computation.* IEEE International Congress on Evolutionary Computation.

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

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

[6] Manuel S., M. L. José, M. B. Victor, M.R. José, 2002. *GENES : a Genetic Algorithms and Fast Time Simulation*. 3nd ATM R&D Symposium, Spain.

[7] TranDac H., Baptiste P., Duong V., 2002. *A Constraint Programming Formulation for Dynamic AirSpace Sectorization.* Proc. of 21st Digital Avionics Systems Conference.

[8] Kernighan B.W. and S. Lin, 1970. *An efficient heuristic procedure for partitioning graphs*. The Bell System Technical Journal, 49:291-307.

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

[10] Barnard S.T. and H. D. Simon,1993. *A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*. In Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing, pages 711-718.

[11] Fiduccia C. and R. Mattheyses, 1982. *A linear time heuristic for improving network partitions*. In 19th IEEE Design Automation Conference, pages 175-181.

[12] Gilbert J.R., G. L. Miller and S. H. Teng, 1995. *Geometric mesh partitioning: Implementations and experiments*. In Proc. International Parallel Processing Symposium, pages 418--427.

[13] Leland R. and B. Hendrickson, 1994. *An empirical study of static load balancing algorithms*. In Proc. Scalable High Performance Comput. Conf., pages 682-685.

[14] Hendrickson B. and R. Leland, 1995. *An improved spectral graph partitioning algorithm for mapping parallel computations.* SIAM J. Sci. Comput., 16(2):452--469.

[15] Hendrickson B. and R. Leland, 1995. *A multilevel algorithm for partitioning graphs*. In Proc. Supercomputing'95.

[16] Karypis G. and V. Kumar, 1995. *A fast and high quality multi-level scheme for partitioning irregular graphs.* Technical Report 95-035, University of Minnesota, Department of Computer Science.

[17] Karypis G. and V. Kumar, 1995. *Multilevel k-way partitioning scheme for irregular graphs.* Technical Report 95-064, University of Minnesota, Department of Computer Science.

[18] Fjällström P-O. 1998. *Algorithms for Graph Partitioning: A Survey.* Linköping University Electronic Press.

[19] Chen J., 1996. *Computational Geometry: Methods and Applications.* Computer Science Department, Texas A&M University

[20] Baptiste Ph., C. Le Pape and W. Nuijten. *Constraint Based Scheduling*. Kluwer Academic Publishers

[21] Bartak R.,1998. *Online Guide to Constraint Programming.* http://kti.mff.cuni.cz/~bartak

[22] Berlandier B., 1995. *Improving Domain Filtering using Restricted Path Consistency.* Proceedings of the IEEE CAIA-95

[23] Barbara M. S., 1995. *A tutorial on Constraint Programming*

[24] Han C., C. Lee, 1988. *Comments on Morh and Henderson's path consistency algorithm.* Artificial Intelligence, 36:125-130.

[25] Kumar V., 1992. *Algorithms for Constraint Satisfaction Problems: A Survey*. AI Magazine 13(1):32-44.

[26] Mackworth A., 1977. *Consistency in Networks of Relations.* Artificial Intelligence 8:99-118.

[27] Montanari U., 1974. *Networks of Constraints: Fundamental Properties and Applications to Picture Processing.* Information Science,7:95-132.

[28] Mohr R. and T. Henderson, 1986. *Arc and path consistency revisited.* Artificial Intelligence, 28:225-233.

[29] Puget, J-F, 1994. *A C++ Implementation of CLP*. Proceedings of SPICIS 94, Singapore.

[30] Colmerauer A., 1990. *An introduction to PROLOG-III.* Communications of the ACM, 33(7):69—90.

[31] Colmerauer A., 1996. *Les bases de Prolog IV.* Publication interne du LIM. http://www.lim.univ-mrs.fr/~colmer/

[32] Wallace M., S. Novello, J. Schimpf, 1997. *ECLiPSe : A Platform for Constraint Logic Programming.* Technical report, IC-Parc, Imperial College, London. http://www.icparc.ic.ac.uk/eclipse

[33] Laburthe F. *CHOCO – a Constraint Programming kernel for solving combinatorial optimization problems.*

[34] Caseau Y. and F. Laburthe. *Introduction to the CLAIRE Programming Language.*

Biographies of the authors

Huy TranDac is a PhD student of the University of Technology of Compiègne. He graduated in Computer Science in 1994 from the Polytechnic University of Ho-Chi-Minh City. From 1994 to 1997, he was assistant professor at the Faculty of Information Technology, CanTho University. He received two MSc degrees in Computer science: from "L'Institut Francophone d'Informatique" de Hanoi (1999) and from the Institut National Polytechnique de Toulouse (2000). He is currently preparing his PhD thesis at EUROCONTROL on the problem of dynamic sectorization. His research interests include operations research and graph theory.

Dr. Philippe Baptiste is a researcher at the French National Research Foundation (CNRS, Heudiasyc) *and* associate professor at Ecole Polytechnique. His main research interests are combinatorial optimization and constraint programming. He completed a PhD from the University of Technology of Compiègne in 1998. He has received both the ``Robert Faure" award from the French Operations Research Society and the ``Cor Baayen" award from the European Research Consortium for Informatics and Mathematics. Philippe Baptiste has published more that 10 papers in international journals and has contributed to 30 international conferences. He is an associate editor of "Journal of Scheduling". Philippe is conducting a research contract with EUROCONTROL on dynamic sectorization of airspace.

Dr. Vu Duong is currently Business Area Manager for Innovative Research at EUROCONTROL Experimental Center, and EUROCONTROL CARE Innovative Research Action Manager. Prior to this position, he had been at the origin of FREER initiative as its Project Manager, then Program Manager Simulator Development Program. Vu holds a Master Degree in Engineering and a PhD in Artificial Intelligence, both from Ecole Nationale des Ponts et Chaussees, Paris.