Optimal combinations of Air Traffic Control sectors using classical and stochastic methods

David Gianazza Jean-Marc Alliot Géraud Granger
LOG (Laboratoire d'Optimisation Globale) CENA/ENAC
7, avenue Edouard Belin 31055 Toulouse Cedex, FRANCE

Abstract: *This paper introduces several algorithms which build optimal configurations of Air Traffic Control sectors, taking into account traffic prediction, sector capacities and the number of available control positions.*

Keywords: Genetic Algorithm, Branch&Bound, A^*, Air Traffic Flow Management, Sector Configuration

1 Vocabulary and acronyms

Sector **:** the airspace under the responsibility of each Air Traffic Control Center is divided into smaller volumes called *sectors*. Sectors may be combined with other sectors and controlled by a team of two air traffic controllers.

Control position **:** a piece of furniture with communication and surveillance equipment (radar screens, radio, telephone,...), manned by a team of two controllers who ensure sufficient separation between the aircraft under their responsibility.

Sectors configuration **:** the mapping of n sectors onto k control positions.

Capacity **:** a threshold value of the traffic flow through a sector or group of sectors, above which it is considered overloaded. If too many aircraft fly through the same sector at the same time, the controllers in charge of that sector cannot safely handle the traffic.

CFMU **:** the European Central Flow Management Unit ensures that the traffic flows do not exceed the capacities, generally by delaying the departure of some aircraft when necessary.

ACC or ATCC **:** Air (Traffic) Control Center.

Flow management position (FMP) **:** a piece of furniture with telephones and computers, manned by a team of FMP operators, which are the correspondents of the CFMU in each ACC.

ACC schedule **:** an estimation of the sector configurations that will be used the next day (or the day after), taking account of the predicted traffic and the available resources of the ACC (controllers and control positions).

2 Introduction

The first step of the Air Traffic Flow Management (ATFM) process is to define the predicted schedule for each Air Traffic Control Center one or two days ahead. This is done by forecasting the traffic and by comparing this estimation to the available resources, in order to determine what sector configuration would be the most adapted to the predicted traffic and if some overloads can be foreseen.

In Europe, the ACC schedule is elaborated by the Flow Management Position (FMP) of each ATC Center. Some automated tools are available to help the operators to perform this task. For example the French FMP operator can choose a sector configuration among a set of pre-defined configurations, and match it with the traffic demand so that traffic overloads appear immediately. But apart from his own experience, he has no way of knowing if the chosen configuration is the most adequate, or if another one could better balance the traffic between the manned control positions. An additional drawback of the current method is that it is limited to a set of statically defined configurations, which is a fairly small subset of all the possible ways to combine sectors, as we will see later.

The present paper describes two classical tree search algorithms and a genetic algorithm that take as input the traffic flows and build optimal sector configurations considering the airspace capacity constraints and also the maximum number of con-

trol positions that can be manned at each time of the day. This optimization is made in a realistic context, using the airspace description data, the traffic data, the number of available control positions, and the sector capacities of the French ATC centers.

3 Related work

The problem of optimizing sector configurations has already been addressed in [1], although with a less realistic model, and also in [6]. In [1], genetic algorithms were used in a toy ATC network to balance workloads by combining sectors, with a chosen number of control positions. Only convex sectors were considered.

In [6], integer programming techniques were used in a realistic context to minimize the sum of traffic overloads (called deficits of capacity) by selecting patterns among a reduced set of statically defined configurations. A pattern is a configuration associated to a time period. The number of control positions is an input parameter, as in [1]. An attempt is made to consider the traffic throughput as a variable, by allowing some macroscopic shifting of traffic loads along the time axis or from a sector to another. The convergence of the iterative algorithm which is used in that case seems not sure. However, the results show significant improvements when compared to the current manual methods.

The optimization problem addressed in the present paper is different : the optimum we are trying to reach is the sector configuration for which the traffic load is as close as possible to the capacity of each sector or group of sectors of the configuration, so traffic overloads as well as traffic under-loads are considered. The search of an optimal configuration is not restricted to a subset of manually entered configurations, but explores the whole set of possible configurations which can be obtained by combining operational ATC sectors. The number of control positions is a variable of the cost function we are trying to minimize, constrained by an upper limit as there may not always be enough ATC controllers to man all the positions that would be needed.

4 Model

A "good" configuration is a configuration for which there are no traffic overloads (or the smallest possible ones), and for which the traffic load is balanced as well as possible between the manned control positions, while arming the minimum number of positions. Let us formulate this as a minimization problem.

Let us define the function Δ by

$$\Delta(x,t) = workload(x,t) - capacity(x,t)$$

where x is a sector or group of sectors, t is the time, $workload$ is the traffic load (for example N aircraft entering sector x between t and $t+w$, where w is a chosen time window), and $capacity$ is the threshold value for the workload. The capacity of each sector or group of sectors may also depend on specific criterions (like military activity...).

The operator may allow some tolerances around the value of the capacity. These lower and upper tolerances l and u are taken into account in the evaluation of a configuration. For this, we will need to define the overloads and underloads as follows :

$$\Delta_{++}(x,t) = \left| \begin{array}{ll} \Delta(x,t) & \text{if } \Delta(x,t) > u \\ 0 & \text{otherwise} \end{array} \right.$$

$$\Delta_{+}(x,t) = \left| \begin{array}{ll} \Delta(x,t) & \text{if } 0 \leq \Delta(x,t) \leq u \\ 0 & \text{otherwise} \end{array} \right.$$

$$\Delta_{-}(x,t) = \left| \begin{array}{ll} |\Delta(x,t)| & \text{if } l \leq \Delta(x,t) \leq 0 \\ 0 & \text{otherwise} \end{array} \right.$$

$$\Delta_{--}(x,t) = \left| \begin{array}{ll} |\Delta(x,t)| & \text{if } \Delta(x,t) < l \\ 0 & \text{otherwise} \end{array} \right.$$

Let us then define the following functions :

$N_{pos}(t)$ the number of control positions in the configuration

$$C_{++}(t) = \sum_{x \in config} (\Delta_{++}(x,t))^2$$

$$C_{+}(t) = \sum_{x \in config} \Delta_{+}(x,t)$$

$$C_-(t) = \sum_{x \in config} \Delta_-(x,t)$$

$$C_{--}(t) = \sum_{x \in config} (\Delta_{--}(x,t))^2$$

The problem will then consist in minimizing the following cost function :

$$cost_{config} = a.C_{++} + b.N_{pos} + c.C_{--} + d.(C_+ + C_-) \quad (1)$$

while respecting the constraint : $N_{pos}(t) \leq M_{pos}(t)$ where a, b, c, and d chosen factors of decreasing value and M_{pos} is the maximum number of control positions available at each time of the day.

Instead of minimizing $cost_{config}$, and in order to take better account of the relative weights of the different costs, we will in fact maximize $eval_{config}$, such that the k_1 most significant digits of $eval_{config}(x,t)$ refer to the cost C_{++}, the next k_2 digits refer to N_{pos}, and so on:

$$\underbrace{xxxxx}_{k_1}\underbrace{xx}_{k_2}\underbrace{xxxxx}_{k_3}\underbrace{xxx}_{k_4}$$

$$\begin{aligned} eval_{config} = \quad & 10^{k_2+k_3+k_4} \times N(k_1, C_{++}) \\ & + 10^{k_3+k_4} \times N(k_2, N_{pos}) \\ & + 10^{k_4} \times N(k_3, C_{--}) \\ & + N(k_4, C_+ + C_-) \end{aligned} \quad (2)$$

where N is a function such that

$$N(k_i, C) = \lfloor max(0, (10^{k_i} - 1) - C) \rfloor$$

and k_1, k_2, k_3, and k_4 are chosen such that $(10^{k_i} - 1)$ is an upper bound of the corresponding cost, if possible.

5 Problem complexity

The difficulty of the problem is mostly due to the high number of possible configurations which can be built from a set of sectors. A configuration is the mapping of n sectors onto k control positions.

Let us first try to find how many ways there are to part a set of n elements into k subsets. If $P(n,k)$ is this number, it verifies the following equations :

$\forall n \geq 1$
$P(n,1) = 1$ (one group of n elements)
$P(n,n) = 1$ (one partition of n groups of one element)
$P(n,k) = 0$ if $k > n$ (we cannot make more than n groups)
$P(n,k) = k * P(n-1,k) + P(n-1,k-1)$ if $1 < k < n$

The number of all possible partitions will then be $P(n) = \sum_{k=1}^{n} P(n,k)$. For 17 sectors (Brest ATCC), this would give around 83 billion possibilities. However, this value is not realistic : many of the partitions could not be used in an operational context. For example, a partition containing a group in which one sector is a neighbor of no other sector in the group is not a valid configuration. The set of all possible partitions (that we have counted above) could theoretically be obtained by exploring the tree of all possibilities , although this would become quickly impractical when the number of sector increases.

In order to estimate the difficulty of the real problem, let us consider only the operational sectors and groups of sectors defined in each ACC database. We will obtain all operational configurations by exploring a tree which nodes are lists of couples $(g, \mathcal{G})$ (cf figure 1 illustrating the Branch&Bound), where g is a group under construction and $\mathcal{G}$ is the set of valid groups compatible with g in the context of the configuration under construction. An element h of $\mathcal{G}$ is "compatible" with g if it contains all the sectors of g, but no sector of the other groups in the configuration (the other "g"'s of the node). If one of the $\mathcal{G}$ sets is empty, then there is no need to continue the search from the considered node : it will lead to no valid configuration.

	Number of sectors		Number of	Number of
	elem.	*groups*	partitions	configs
Aix	24	42	4.4610^{17}	123,965
Bordeaux	22	65	4.4510^{15}	551,032
Brest	17	52	8.2810^{10}	14,832
Paris (West)	11	17	678,570	192
Paris (East)	12	22	4,213,597	399
Reims	12	17	4,213,597	249

Table 1: Number of possible sector configurations for the French ATCCs (in 1999)

The methods presented above allow us to count the number of configurations and estimate the difficulty of the problem. The results for the French ATC centers are shown in table 1, which highlights the combinatorial relation between the number of partitions and the number of sectors.

The number of operational configurations which we can build with the sectors or groups of sectors described in the ATCCs airspace data goes from a few hundreds to a half million. So, we can expect that the use of exhaustive tree search techniques

will lead to optimal configurations within a reasonable computation time, at least for relatively small ATC centers and with sector combinations restricted to the groups described in the ATCCs airspace data. However, as the sector configuration optimization problem may be extended to larger ATCCs, and as we may wish to use a wider range of sector combinations, genetic algorithms were also experimented to solve the problem.

6 Classical algorithms

A basic algorithm: In the previous section, we have mentioned an algorithm which builds all the possible operational configurations. A basic optimization method consists in evaluating the cost (cf definition 1) of each configuration in order to find the best one. The basic algorithm explores all the configurations. It is memory and time consuming, but complete : we can be sure that it returns the optimum of the evaluation function. So it will be used as a reference for the other algorithms presented hereafter.

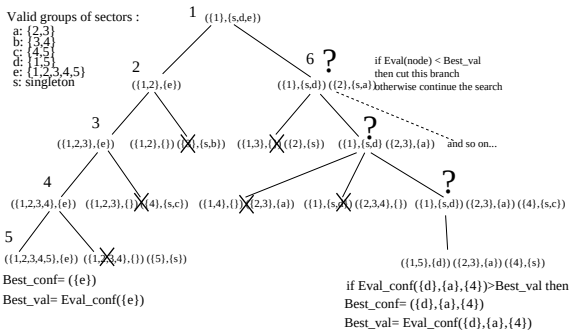

Figure 1: *Branch & bound* search for an optimal sector configuration

A *Branch & bound* **algorithm:** The idea of the *branch & bound* algorithm illustrated in figure 1 is to avoid exploring every branch of the tree. To do so, it needs to evaluate the nodes in order to decide, by comparison with the best configuration found so far, whether to continue or not the search along a given branch. The cost (resp. evaluation) of a node must be a lower bound (resp. an upper bound) of all the costs (resp. evaluations) of configurations which can be reached from that node. A node of our tree search algorithm is a configuration under construction (see figure 1), i.e. a list of couples $(g, \mathcal{G})$, where g is a group under construction and $\mathcal{G}$ is the set of valid groups compatible with g in the context of the configuration under construction.

For a better understanding of this notion of compatibility, let us consider node 6 of the example shown in figure 1. This node is represented by : $(\{1\}, \{s, d\})$ $(\{2\}, \{s, a\})$. The valid groups compatibles with the group under construction $\{1\}$ are the singleton s, which is $\{1\}$, and the group $d = \{1, 5\}$. These groups are the only ones which contain $\{1\}$ but not $\{2\}$.

The cost function for a node is similar to the cost of a configuration. Let us define a function $best$ such that $best(\mathcal{G})$ returns the element h (a sector or group of sectors) of $\mathcal{G}$, for which the difference $workload(h,t) - capacity(h,t)$ is the smallest possible underload, or the smallest possible overload if all groups of $\mathcal{G}$ are overloaded. We will then define the cost (resp. evaluation) of a node as in definition 1 (resp. 2), except that N_{pos} will be the number of couples $(g, \mathcal{G})$ in the node, and that only Δ_{++} and Δ_{+} will be considered, taking $best(\mathcal{G})$ as input instead of an operational sector x of a configuration.

An algorithm inspired from A^*: Like the *Branch & bound*, the A^* is a tree search algorithm. However, instead of simply storing the best leaf found so far, all the explored nodes are stored and sorted by valuation.

In order to do so, the A^* as described in [5] needs a cost function defining the cost of each state transition, and a function (called heuristic) underestimating the cost of the remaining transitions between the current node and the end of the search.

For our sector configuration problem, we will try to minimize the $cost_{config}$ function described in definition 1, by searching a path in the tree illustrated in figure 1. But instead of comparing the node evaluation with the evaluation of the best configuration found so far like in figure 1, the nodes already explored are stored into a priority queue sorted according to the node eval-

uation. The A^* will then iteratively consider the node with lowest cost, evaluate its children nodes and insert them in the priority queue, until a leaf of the tree is reached. This leaf is then the optimal sector configuration. The evaluation of a configuration and a node is the same as in 1 and the *Branch & bound* algorithm respectively.

7 Genetic algorithm

A genetic algorithm has also been tested to solve the sector configuration problem. The genetic algorithm considers a population of chromosomes, which evolves by crossover, mutation, and selection of the fittest individuals, as described in [3] and [4].

A *chromosome* will be a sector configuration. Each chromosome is composed of several genes. A gene is either an elementary sector or a group of sectors. A fitness value is assigned to each chromosome. The raw fitness f of a configuration will be given by $eval_{config}$ (see definition 2).

A *clusterized sharing* operator is applied to the raw fitnesses. The aim of sharing is to avoid that a chromosome with a good fitness reproduces itself to the detriment of the other chromosomes, thus narrowing the search to a single optimal or suboptimal mode. The chromosomes clustering of this sharing operator is based on distance criteria. For our problem, the difficulty to implement sharing lies in the definition of a "distance" between sector configurations. Like the distance of Hamming which uses the number of genes that differ between two chromosomes, the pseudo-distance that was chosen for our problem is based on a gene comparison, except that the two chromosomes may not have the same number of genes. Let n_i (respectively n_j) be the number of genes of the chromosome i (respectively j). The chosen pseudo-distance is defined by $d(i,j) = min(n_i, n_j) - n$ where n is the number of identical genes between the two chromosomes. In addition to the sharing operator, a scaling operator (*sigma truncation*) is applied, in order to smooth the differences between fitnesses of good chromosomes and bad ones. This way, the selection will let a better chance to bad chromosomes to reproduce themselves, and the domain will be more widely explored.

The crossover operator splits the two parent configurations and completes each half configuration with the other parent's genes. The resulting incomplete chromosome is then completed with valid groups or sectors. A local fitness is associated to each gene in order to select the parents genes in decreasing order of local fitness. It was shown in [2] that an adapted crossover operator using local fitnesses increases the convergence rate for the optimization of partially separable functions. In our problem, the local fitness is similar to the raw fitness of the chromosome, although it takes account of the overloads and underloads of a single gene.

The mutation operator first randomly chooses one gene of the chromosome. Another gene is randomly chosen among a list comprising the first chosen gene and its neighbors (sectors or groups of sectors of the configuration which have a common border with the first chosen sector or group of sector). The sectors of the chosen genes are then recombined into one or several new genes (up to three). The splitting of a single group would allow only mutations towards less loaded configurations with an increasing number of control positions, whereas the recombination of two neighbors may lead to configurations with less items, which may be interesting when the traffic is low.

8 Results

A graphical interface has been developed to display the results of the optimizations. The languages Ocaml and OcamlTk were used to code respectively the algorithms and the interface. The program runs on a PC Pentium IV (1.8 GHz), with Linux as operating system.

The input parameters are the date, the selected ATCC, the type of traffic (raw demand, final demand...), the capacity tolerances, the maximum number of available control positions, the chosen time step and time horizon (for the computation of traffic flows). The program displays an optimized sector configuration for each time subdivision of the day. The color code is : Green (grey) when the traffic load is under the capacity minus the lower tolerance for the capacity ; Yellow (white) when the traffic load is between the margins of tolerance ; Red (dark grey) when the traffic load is over the capacity plus the upper tolerance.

The upper part of figure 2 shows the result of

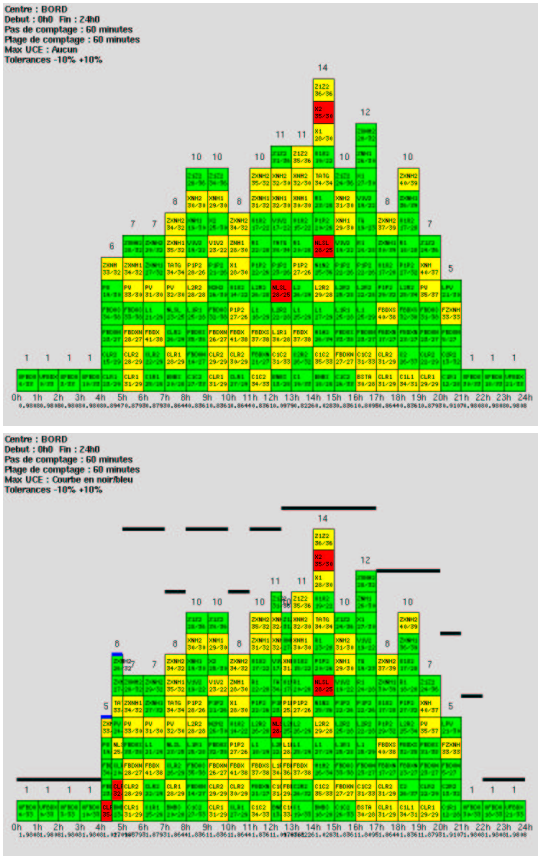

Figure 2: Optimal sector configurations with (up) or without (down) constraints on the number of available positions (example of Bordeaux ACC)

the optimization for Bordeaux ATCC, with the raw traffic demand and with no constraint on the available number of control positions. In this case, the remaining overloaded sectors are necessarily elementary sectors, or operational groups which cannot be split (some elementary sectors are defined and used only to improve flexibility by choosing different ways to combine two of them into an operational group).

The lower part of figure 2 shows the same optimization but with constraints on the available number of positions. These constraints are taken into account in the evaluation functions by dividing by two the evaluation when the number of control positions in the configuration is greater than the available maximum, thus penalizing these configurations. The constraints used in the right part of figure 2 are directly issued from the ACC schedule that was sent to the CFMU by Bordeaux FMP for that day. These constraints induce additional overloads (like between 4 and 5 o'clock in the example) on combined sectors which the algorithms are unable to split because there are not enough available control positions.

By comparing the two schedules we see that Bordeaux ATCC was under-capacitive in the early morning, and over-capacitive at other times of the day.

The tree search algorithms provide optimal sector configurations for each ATCCs, verified with the basic algorithm when possible (for Brest, Paris and Reims ATCCs). The table 2 shows the computation times for these algorithms, with the raw traffic demand and no constraint on the number of maximum positions. The fastest is the *branch & bound*. The A^* is slower because the chosen heuristic is not a very good estimate of what it really costs to reach the best configuration achievable from a given node, thus inducing high backtracking.

	Aix	Bord.	Brest	Paris(E)	Paris(W)	Reims
Basique	n.t.	n.t.	14.44	0.06	0.03	0.02
B&B	34.06	6.06	1.45	0.08	0.07	0.04
A^*	134.88	10.12	10.05	0.08	0.04	0.02

Table 2: Computation times (in seconds) for the deterministic algorithms

Although there are more possible configurations for Bordeaux ATCC than for Aix ATCC (see table 1 about the problem complexity), the computation time for Aix ATCC is higher than for Bordeaux as there are more sectors in Aix than in Bordeaux. The tree of all possible configurations is searched "depth first", and the depth depends on the number of sectors.

For the genetic algorithm, 10 different values of the random seed were tested for Bordeaux ATCC, with a crossover probability of 0.6 and a mutation probability of 0.2. Table 3 shows for each time step of the day the number of configurations which where different from the optimum. The lines labelled $(+1)$ show the number of occurrences when there was exactly one more control position in the configuration found by the GA than in the optimal configuration. The lines labelled $(> +1)$ show the number of occurrences when the difference was greater than one control position.

When they were not optimal, the solutions found were qualitatively close to the optimum.

The genetic algorithm is slower than the tree search methods, but provides several optimal or near-optimal solutions. It is not very significant

	Step	0	1	2	3	4	5	6	7	8	9	10	11
220 gen.	fails	0	0	0	0	0	0	0	0	2	0	1	5
120 elem.	+1	0	0	0	0	0	0	0	0	0	0	0	2
	$> +1$	0	0	0	0	0	0	0	0	0	0	0	0
220 gen.	fails	0	0	0	0	0	0	0	0	3	0	0	3
130 elem.	+1	0	0	0	0	0	0	0	0	0	0	0	1
	$> +1$	0	0	0	0	0	0	0	0	0	0	0	1
300 gen.	fails	0	0	0	0	0	0	0	0	0	0	0	0
220 elem.	+1	0	0	0	0	0	0	0	0	0	0	0	0
	$> +1$	0	0	0	0	0	0	0	0	0	0	0	0

	Step	12	13	14	15	16	17	18	19	20	21	22	23
220 gen.	fails	2	0	1	3	0	1	0	0	1	0	0	0
120 elem.	+1	0	0	1	0	0	1	0	0	1	0	0	0
	$> +1$	0	0	0	0	0	0	0	0	0	0	0	0
220 gen.	fails	3	0	2	2	0	0	0	0	1	0	0	0
130 elem.	+1	0	0	2	0	0	0	0	0	1	0	0	0
	$> +1$	0	0	0	0	0	0	0	0	0	0	0	0
300 gen.	fails	0	0	1	0	0	0	0	0	0	0	0	0
220 elem.	+1	0	0	0	0	0	0	0	0	0	0	0	0
	$> +1$	0	0	0	0	0	0	0	0	0	0	0	0

Table 3: GA results for Bordeaux ATCC

to compare the overall computation times of the genetic algorithm and the tree search algorithm when the minimization problem is easily solved. In such cases (at night for example), the genetic algorithm will run the same number of generations whatever the difficulty of the problem, whereas the B&B will find an optimal solution among the first branches it explores. Table 4 compares for Bordeaux ATCC the computation times of each time step between 5 a.m and 7 p.m., for 220 generations and a population size of 120 configurations:

Step	5	6	7	8	9	10	11
B&B	0.23	0.05	0.17	0.55	0.95	0.21	0.38
GA	15.69	14.88	17.19	16.08	15.37	15.80	15.75
Step	12	13	14	15	16	17	18
B&B	0.54	0.21	0.66	0.59	0.58	0.35	0.56
GA	16.42	15.62	17.41	16.85	15.65	14.98	13.70

Table 4: GA and B&B detailed computation times for Bordeaux ATCC

9 Conclusion

The tree search algorithms as well as the genetic algorithm provide optimal sector configurations, using the same parameters and constraints than in the current FMP/CFMU process. The results are computed in a time short enough for an operational use. The tree search algorithms are faster than the genetic algorithm when applied to the French ATC centers and while restricting the ways to group sectors to a set of operational groups (issued from each ATCC airspace description database).

However, the combinatorial relation that we have shown between the number of sectors and the number of partitions and configurations is such that the tree search algorithms may prove impractical in the context of larger ATC Centers or if a wider set of operational groups is used. In such a context, the genetic algorithm is a good alternative. It provides several optimal or near-optimal configurations in a chosen computation time.

So far, we have considered the sector configuration optimization only in the context of the pre-tactical ACC schedule estimation, based on the traffic flows and the capacity constraints. The proposed algorithms could as well be envisioned, with adapted workload and constraints definitions, to tactically propose sector configurations to control room managers.

References

[1] Daniel Delahaye, Jean-Marc Alliot, Marc Schoenauer, and Jean-Loup Farges. Genetic algorithms for automatic regroupement of air traffic control sectors. In *Proceedings of the Conference on Evolutionary Programming*, 1995.

[2] N. Durand and J. M. Alliot. Genetic crossover operator for partially separable functions. In *Proceedings of the third annual Genetic Programming Conference*, 1998.

[3] David Goldberg. *Genetic Algorithms*. Addison Wesley, 1989. ISBN: 0-201-15767-5.

[4] Zbigniew Michalewicz. *Genetic algorithms+data structures=evolution programs*. Springer-Verlag, 1992. ISBN: 0-387-55387-.

[5] Judea Pearl. *Heuristics*. Addison-Wesley, 1984. ISBN: 0-201-05594-5.

[6] C. Verlhac and S. Manchon. Optimization of opening schemes. In *Proceedings of the fourth USA/Europe Air Traffic Management R&D Seminar*, 2001.