Allocating 3D-trajectories to air traffic flows using A* and genetic algorithms

David Gianazza, Nicolas Durand, Nicolas Archambault

Allocating 3D-trajectories to air traffic flows, using A^* and genetic algorithms

D. Gianazza and N. Durand and N. Archambault
LOG[1] */ CENA*[2]
7, avenue Edouard Belin 31055 Toulouse Cedex
{gianazza,durand,archamba}@recherche.enac.fr

Abstract

This paper introduces two methods, tested on a toy problem, which allocate optimal separated 3D-trajectories to air traffic flows. The first approach is a 1 *vs.* n *strategy which applies an* A^* *algorithm iteratively to each flow. The second is a global approach using a genetic algorithm, applied to a population of trajectory sets.*

1 Introduction

The critical problem of airspace congestion over Europe is currently being handled by the Air Traffic Flow Management (ATFM) by allocating departure time slots to aircraft, thus generating costly ground delays. Besides, airlines operators are free to choose the flight path through the routes network, and the requested flight level for each of their flights[3]. Very few constraints are put on the airlines operators concerning these choices. A consequence is that many flights request the same flight levels, thus increasing the airspace congestion. The continuous traffic increase is now pushing the current system to its limits, and the need for alternatives of higher capacity arises.

In this paper, we propose to reduce congestion by allocating full 3D-trajectories to the main traffic flows, in order to obtain a set of optimal trajectories which do not interfere one with each other. Section 2 brings a short overview of previous works on ATM[4] problems. Section 3 describes the fairly simplified problem we have chosen to address first, with a discussion on the complexity and the solving strategies. Section 4 deals with the application of the A^* algorithm to our problem, with a 1 *vs.* n strategy. Section 5 describes a *global strategy* using a genetic algorithm, and its results. In the conclusion, the results are summarized, and the further developments and applications to real traffic are discussed.

2 Context and related work

The Air Traffic Control (ATC) system is currently organized in predefined routes, crossing airspace sectors. Each sector is handled by a team of two controllers (a *planning* controller

[1]LOG : *Global Optimization Laboratory*

[2]CENA : the *Centre d'Etudes de la Navigation Aérienne* is the French Civil Aviation institution in charge of developing new concepts and applications for the future air traffic control automated systems.

[3]although their choices may be limited to a subset of routes when a traffic orientation scheme is defined.

[4]ATM: Air Traffic Management.

and a *radar* controller), whose job is to monitor aircraft trajectories within their sector, detect potential conflicts, and issue lateral or vertical manoeuvres to conflicting aircraft when necessary. We may distinguish the *en-route* control with aircraft following predefined routes, and the *approach* control in Terminal Areas (TMAs) around the main airports, with specific arrival and departure procedures. When the traffic within a TMA or an airspace sector becomes too high, the controllers workload may reach a limit beyond which safety is not guaranteed. To avoid this situation, the Air Traffic Flow Management (ATFM) services may delay the take-off of departing aircraft. This system is now reaching a limit : a small amount of additionnal traffic generates a great increase in the cumulated delays (c.f. Eurocontrol report [1]).

Numerous papers related to Air Traffic Management (ATM) appear in the litterature, many of them dealing with capacity problems ([2], [3], [4]) or with dynamic flight planning through a congested airspace ([5], [6]), using a variety of deterministic or stochastic methods. The solving of these problems is not in the scope of this paper, which deals with static 3D-trajectory design. The routes network design is adressed in [7], using Voronoï diagrams and clustering methods which iteratively move and merge the crossing points of the network. Although quite interesting, this approach is mainly bi-dimensionnal and does not take account of the vertical evolutions of aircraft. In [8] and [9], Graph Colouring techniques are used to assign cruising flight levels to aircraft flying on direct routes, in order to ensure vertical separation during the cruise. The climbing or descending trajectory segments are not considered. In [10], an interesting concept of TMA-to-TMA handover is assessed through statistical studies and traffic simulations. The idea is to remove a percentage of the traffic from the current ATC system (and from the slot allocation process) by defining conflict-free routes between the main terminal areas. Aircraft flying on these routes would have priority over the rest of the traffic and would be handled by specific departure and arrival management tools. Removing even a small percentage of traffic may drastically reduce delays, as long as the impact on the overall system capacity is limited. However, only horizontal separation between routes is discussed: crossing routes are either forbidden or allowed in a very limited way. So only a few traffic flows could be considered, without significant profit in terms of conflict reduction. In addition, it is not specified how aircraft would be sequenced on each route and how separation from other traffic would be achieved.

In what follows, we propose to build static conflict-free 3D-trajectories between the main Terminal Areas using two alternative methods. The first is an iterative 1 *vs.* n strategy where flows are considered in decreasing order of size, and the second is a stochastic approach with a global strategy. The chosen model, the complexity of the problems addressed by each strategy and the choice of the algorithms is discussed in the next section.

3 The toy problem.

3.1 Trajectory model

Let us first consider a fairly simplified model for our trajectory design problem. The set of flows shall be arbitrarily chosen (origins and destinations on a grid, or on a circle, for example). A

flow is defined as a set of flights between a departure airport and an arrival airport. The following simplifications are made : the airspace is considered as an Euclidean space, where all airports are at altitude 0. Latitudes and longitudes on the ellipsoïd earth surface are converted into (x, y) coordinates by a stereographic projection, and the altitude in feet shall be our z coordinate [5] . All aircraft fly with identical performances and follow linear slopes of climb and descent, and all aircraft belonging to a given flow i request a same cruise flight level RFL_i[6], and follow by default a direct route between departure and arrival.

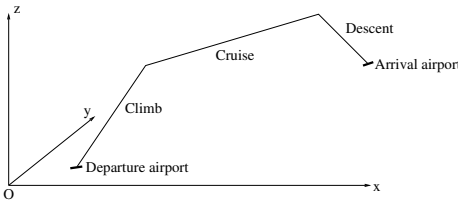

Figure 1: *Basic model of a default trajectory*

These simplifications allow us to allocate only one trajectory per flow. A trajectory shall be a sequence of line segments in our Euclidean space. Figure 1 shows a default trajectory between a departure airport and an arrival airport, with a climb towards a requested flight level RFL_i, and a descent downto the destination airport.

In order to avoid conflicts between trajectories, we shall introduce some lateral or vertical deviations. Horizontally, we shall allow only three possibilities, as shown on the left part of figure 2 : the direct route, or left or right parallel routes. The radii around the departure and arrival airports, and the value of the offset, are input parameters.

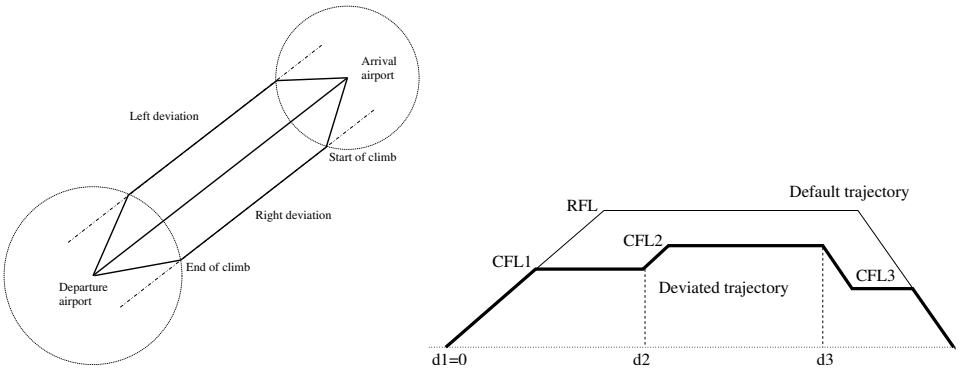

Figure 2: *Horizontal (left) and vertical (right) deviations*

Vertically, we shall allow a succession of different flight levels (CFL stands for Cleared Flight

[5]This approximation is possible only as long as we stay in an area around the projection center which is not too large. It also introduces some errors in the computation of distances between trajectories : aircraft usually follow orthodromic routes over the earth surface, which will not be projected as straight lines on our stereographic plane. These errors can be balanced in our problem by increasing the separation minima between trajectories.

[6]Requested Flight Level

Level), as illustrated on the right part of figure 2. Each of the flight levels shall be between a minimum flight level FL_{min} and the requested flight level RFL. The vertical deviations are then characterized by a sequence of pairs (d_j, CFL_j), where d_j is the distance (along the route) at which the vertical evolution towards CFL_j begins. Each trajectory i is then completely represented by the following variables : the choice of a route, represented by a variable r (equal to 0 for the direct route, 1 when the deviation is to the right, and -1 when it is to the left), and the sequence of pairs (d_j, CFL_j).

We are now able to define a cost related to the lateral and vertical deviations. The cost of a vertical deviation depends on the surface between the effective vertical profile and the cruise level : $l_i \times RFL_i - surface(profile_i)$ where l_i is the length of the chosen route. This cost should not depend on the distance between origin and destination (otherwise small deviations on long flights may cost as much as big deviations on short flights), so we shall divide this expression by the route length l_i. The cost of a lateral deviation depends on the route elongation $\frac{(l_i - lref_i)}{lref_i}$, where $lref_i$ is the length of the direct route. Finally, the total cost of a trajectory is a combination of the two, with K a chosen factor : :

$$cost(i) = RFL_i - \frac{surface(profile_i)}{l_i} + K \times \frac{(l_i - lref_i)}{lref_i}$$

3.2 Detecting interfering trajectories

Let us now define the notion of *interference* between trajectories. We will deliberately avoid to use the term *conflict*, which in the aviation community refers to the fact that two aircraft are (or will be) closer than the allowed separation standard (usually 5 nautical miles horizontally and 1000 feet vertically). The *conflict* definition refers to the horizontal and vertical distances between *points* in space, not 3D line segments. Let us call N_h and N_v the standard horizontal and vertical separations. A first definition of the *interference* between trajectories is the following : two trajectories $\mathcal{T}_1$ and $\mathcal{T}_2$ are interfering when there exists a pair of points (p_1, p_2), with $p_1 \in \mathcal{T}_1, p_2 \in \mathcal{T}_2$, such that $d_h(p1, p2) < N_h$ and $d_v(p1, p2) < N_v$. In this definition, d_h is the horizontal distance, and d_v is the vertical distance.

With this definition, detecting interferences means to find the parts of trajectories which simultaneously violate the horizontal and vertical separation constraints, and this is not an easy exercise of three-dimensional geometry. This is why we shall slightly modify the definition of the *interference*, by introducing a new distance. Let us first define the following scalar product :

$$\begin{pmatrix} x1 \\ y1 \\ z1 \end{pmatrix} . \begin{pmatrix} x2 \\ y2 \\ z2 \end{pmatrix} = \frac{x_1.x_2 + y_1.y_2}{N_h^2} + \frac{z_1.z_2}{N_v^2}$$

Let us call d the distance defined by this scalar product. The distance $d(a, b)$ between two points a and b is equal to the squared root of the scalar product of the vector ab by itself ($d(a, b) = \sqrt{ab.ab}$). It can easily be shown that for each pair of trajectories $(\mathcal{T}_1, \mathcal{T}_2)$, if p_1 and p_2 are the closest point with this new distance d, then :

$$d(p_1, p_2) > \sqrt{2} \Rightarrow \left(\begin{array}{c} \forall (q_1, q_2) \in \mathcal{T}_1 \times \mathcal{T}_2 \\ d_h(q_1, q_2) > N_h \\ d_v(q_1, q_2) > N_v \end{array} \right) \quad (1)$$

In the rest of this paper, we shall consider that two trajectory segments $s1 \subset \mathcal{T}_1$ and $s2 \subset \mathcal{T}_2$ are interfering when the closest points (using the distance d) are at a distance less than $\sqrt{2}$, when at least one segment is climbing or descending. When both segments are at a constant altitude, they will interfere when the altitude difference is less than N_v and when their horizontal separation is closer than N_h. The use of distance d brings an additional margin in the separation of trajectories when compared to our initial definition, but the implementation of the detection of interferences is much easier this way.

3.3 Problem description and discussion

Let us consider a set of N traffic flows. For each flow there is a default trajectory between departure and arrival. These trajectories may interfere one with each other, in the sense of the above definition. Our goal is to find a set of N new trajectories, as close as possible to the default trajectories, but separated in space according to the distance criterion defined in the previous subsection. This is a minimization problem, for which we may choose among two strategies. A first approach, that we will call 1 *vs.* n, is to consider each flow in turn and try to find a new trajectory of minimal cost, separated from the previous n trajectories ($n < N$). With this strategy, the solution found (when there is one) depends on the order in which the flows are considered : different sequences will lead to different solutions. The second strategy, that we may call *global strategy*, consists in minimizing a cost associated to the set of trajectories, without any constraint of order on the flows. The global cost should cumulate the costs of each individual trajectories. So there are in fact two different problems, depending on the chosen solving strategy : *global* or 1 *vs.* n.

Two difficulties may arise with the 1 *vs.* n problem. The first one is directly related to the number of constraints put on the trajectory we are trying to built, i.e. by the amount of airspace occupied by the already computed trajectories. If there are too much constraints, there may be no solution to our problem. The second one is the number of combinations of possible trajectory deviations. It may take too much time to find an optimal solution while exploring a great number of possible trajectories. However, these well-known difficulties may not prevent us to try a classical tree-search method on this problem. As we are trying to find the shortest path through the airspace while avoiding obstacles (the other trajectories), the A^* algorithm seems adequate.

The global problem is similar, in terms of complexity, to the conflict resolution problem discussed in [11] (section 2.3). Let us consider two flows of aircraft following interfering default trajectories T_1 and $\mathcal{T}_2$. There are two possibilities when trying to build two separated trajectories $\mathcal{T}_1'$ and $\mathcal{T}_2'$ for these flows : either the tube of section $\sqrt{2}$ defined around $\mathcal{T}_1'$ (with the distance d) crosses the surface delimited by the trajectory and the straight line between departure and arrival of $\mathcal{T}_2'$, as shown on figure 3, or this tube is completely outside this surface.

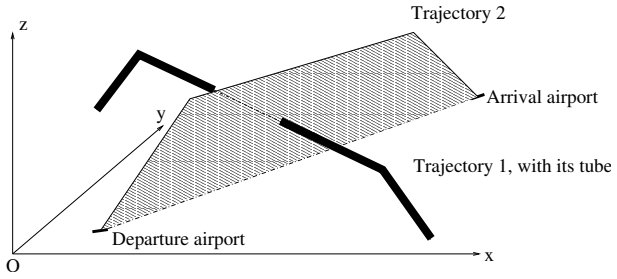

Figure 3: *Example of solution with trajectory 1 inside the surface of trajectory 2*

As aircraft cannot fly under the earth surface, there is no way to pass from one type of solution (crossing the surface) to the other (outside the surface) without discontinuity. So the set of solutions for this two-flows problem is split in two different connected components.

For a given number n of interfering trajectories, we must consider the $\frac{n(n-1)}{2}$ pairs of trajectories, and the number of connected components is now $2^{\frac{n(n-1)}{2}}$. This huge number of connected components (more than thirty thousand billions for 10 trajectories) and the fact that we don't know *a priori* in which component(s) is (or are) the optimal solution(s), make the global problem highly combinational and prevent to use local methods (gradient, BFGS, and so on...). Furthermore, when the number of trajectories increases, we have to face big size problems which are generally difficult to handle with deterministic optimization (as A^* or Branch & bound for example), so we will use a genetic algorithm for the global problem.

4 Solving the *1 vs. n* problem, using an A^* algorithm

4.1 Tree-search description

The A^* algorithm is applied iteratively to each flow. Its aim is to find the shortest trajectory from departure to arrival, while avoiding the already computed trajectories.

The idea of the A^* algorithm (cf. [12]) is to search the best path through a tree of possibilities, restarting at each step from the best possible node encountered so far during the search. To do this, we need to define a cost function for the transitions between states (tree nodes), and a heuristic function which shall estimate the cumulated cost of the transitions remaining between the current state and a possible solution. In our problem, the states shall represent choices in the possible deviations (horizontal or vertical), made at each step of the trajectory. The cost and heuristic functions shall depend on the extent of the trajectory deviations.

As discussed before, a trajectory is completely defined by the choice of a route r, and a list of pairs (d_j, CFL_j). Such a pair is a constraint on the vertical profile, which meaning is : at distance d_j, start a vertical evolution towards flight level CFL_j. The choice of a distance d_j is not free : it can only be a multiple of one tenth of the route length. There is also an additional implicit constraint : the vertical profile must end at the arrival airport, following the final descent

slope. A trajectory state is simply represented by $(r, \{(d_j, CFL_j) \ / \ j \in [0, 9]\}$. As an example, the default trajectory will be $(0, [(0, RFL)])$. The trajectory is built step by step : climb (or descend) towards CFL_j then stay levelled until you climb or descend towards CFL_{j+1}. So each constraint (d_j, CFL_j) will induce an evolutive flight segment followed by a levelled segment.

So a node is completely defined by the index j and the current vertical evolution $evol$, which may be either *Climb* or *Descent* (during the evolution towards the current CFL_j), or *Levelled*.

The tree root is a specific initial state for which no choice of route or level is made yet. This initial state has three sons (at most), corresponding to the three possible parallel routes. For each route, we compute the highest level CFL_0 (within the bounds $[FL_{min}, RFL]$) that can be reached without interfering with other trajectories during the initial climb. Such a son is then described by $(r, [(0, CFL_0)], Climb)$. If there is no such initial climb, the son is not valid and is rejected.

Every other node usually has two sons corresponding either to the best possible next step without interference with other trajectories, or to an alternative step with either an evolution towards a lower flight level if the current node represents a vertical evolution, or a shorter levelled segment if the current evolution is $Levelled$. When the current node is a $Levelled$ step which ends at the top of descent, an additional son is computed representing the final descent towards the arrival airport.

This tree-search mechanism allows to explore all possible states, if necessary, without generating too many sons at each processing step. The tree is explored by considering at each step the best node computed so far. To do this, the A^* handles a *priority queue* which stores the already explored nodes. The search ends when the arrival airport is reached. The trajectory built by the A^* is then the one closest to the default trajectory that does not interfere with the other trajectories.

4.2 Transition cost and heuristic

At each processing step, we need to compute the costs of the transitions between the father node, extracted from the *priority queue*, and its sons. For our problem, the cost of a transition between a state s_1 and a state s_2 is :

- $K \times \frac{l - lref}{lref}$, the cost of the route elongation, when s_1 is the initial state,
- the surface of the area delimited by the requested flight level RFL and the trajectory part corresponding to constraint j (between the distances d_j and d_{j+1}) divided by the route length l, if the transition between s_1 is the levelled segment of constraint j and s_2 is the evolutive segment of constraint $(j + 1)$,
- O in any other case.

Note that the cumulated cost of all transitions from departure to arrival is equal to the cost of the overall trajectory defined in section 3.

The heuristic must absolutely be a lower bound of the remaining transitions costs, otherwise the estimated cost of a node leading to the optimal solution could be over-estimated, preventing to explore the corresponding sub-tree (the one with the optimum) as other nodes may be extracted from the *priority queue* first. So we will compute a vertical profile starting at the current point and joining the default profile (cruise level at RFL) corresponding to the chosen route r. The heuristic will be the surface delimited by this rejoining profile and the RFL, divided by the route length.

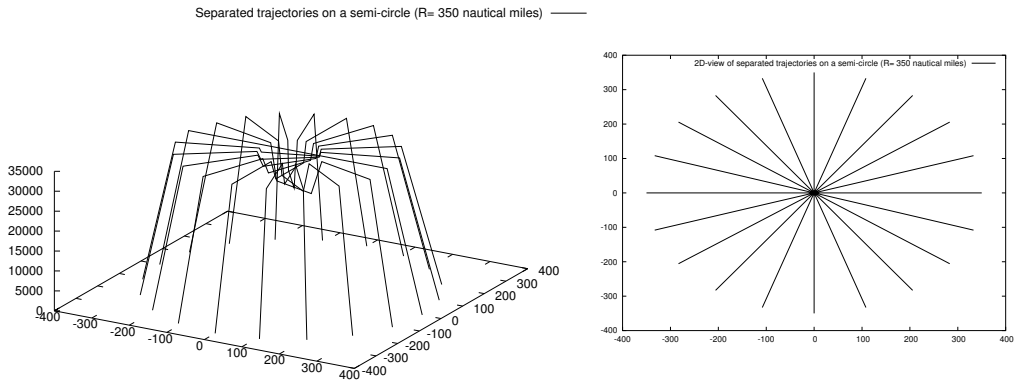

Figure 4: *A^* results for 10 flows on a semi-circle (SC)*

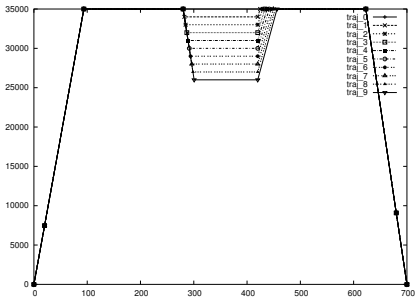

Figure 5: *Side view of A^* results for 10 flows on a semi-circle (SC)*

4.3 A^* results

The algorithm has been tested with different configurations of flows. Let us focus on the semi-circle configuration (SC), with flows converging towards a same point, and requesting each a same flight level (35000 feet, also noted FL350). For this configuration, the A^* algorithm computes non-interfering trajectories as shown on figure 4. Figure 5 shows the side views of each trajectory. The cumulated cost of all deviations is 52.068, and the computation time on a Xeon CPU 2.8 GHz is 5.76 seconds.

5 Solving the global problem using a genetic algorithm

5.1 Description

Genetic algorithms are based on the paradigm of natural evolution. Optima are reached through a process of crossing, mutation and selection of the fittest individuals. This process is applied to a population of chromosomes. The reader may refer to [13] and [14] for an overview of the latest algorithms based on the evolutionnary paradigm, or to [15] and [16, 17] for more details on genetic algorithms. A good state of the art of optimization using genetic algorithm may also be found in [18] and [11], with a practical application to the Air Traffic Control domain (specifically to conflict solving) in the latter. In our problem, a chromosome will be a set of n trajectories. Each chromosome of the initial population is generated by randomly choosing its n trajectories, within the bounds described in subsection 3.1.

The *fitness* criterion allowing to select the best trajectory sets at each step is directly related to the cumulated cost of trajectory deviations. However, the fitness function also takes account of the interferences between trajectories : chromosomes with interfering trajectories shall be penalized. Let us first define a triangular matrix C, which diagonal values indexed (i, i) are the deviation costs, defined in subsection 3.1, of each trajectory. The other values, indexed (i, j) with $i \neq j$ store the number of interfering segments between trajectories $\mathcal{T}_i$ and $\mathcal{T}_j$.

$$\begin{array}{rll} & C_{ii} = & cost(i) \\ \forall i \neq j & C_{ij} = & \sum_{(s_p,s_q)\in\mathcal{T}_i\times\mathcal{T}_j} \delta(s_p, s_q) \end{array}$$

where

$$\begin{array}{rl} \delta(s_p, s_q) = & 1 \text{ if } d(s_p, s_q) \leq \sqrt{2} \\ & 0 \text{ otherwise.} \end{array}$$

Let us note $f(i)$ the sum $\sum_{j\in[1,n],j\leq i} C_{ij}$. This sum is equal to zero when trajectory $\mathcal{T}_i$ is not interfering with any other trajectory. For a chromosome with completely separated trajectories, we will have $\sum_{i\in[1,n]} f(i) = 0$.

The fitness criterion $\mathcal{F}$ of a given chromosome (a set of n trajectories) is defined by :

$$\mathcal{F} = \left| \begin{array}{ll} 1 + \frac{n}{1+\sum_i C_{ii}} & \text{if } \sum_i f(i) = 0 \\ \frac{1}{\sum_i f(i)} & \text{if } \sum_i f(i) > 0 \end{array} \right.$$

Fitness values shall then be less than 1 for chromosomes with interfering trajectories, and above 1 for chromosomes with separated trajectories. In this last case, the smaller the trajectory deviations will be, the greater the fitness will be.

The crossover operator takes advantage of the partial separability or our problem to recognize and favor good genes combinations in the chromosomes (the reader may refer to [19] for more details on this subject). To do so, we will need to compare trajectories of different chromosomes, using a *local fitness* computed for each trajectory in each chromosome. This local fitness is

simply the sum $f(i) = \sum_{j\in[1,n],j\leq i} C_{ij}$ that we have seen above, which represents the number of segments of trajectory $\mathcal{T}_i$ interfering with other trajectories.

Let us now consider two chromosomes a and b. The crossover operator chooses with different probabilities among two following strategies : copy in a all trajectories of b whose local fitness is better (strictly, or with an additional margin) than in a, and do the same for b, or generate two new chromosomes by a barycentric crossover. The first crossover strategy is more elitist than the second, as we deliberately try to produce better fit chromosomes. The crossover is applied to the population with a given probability which is a parameter of the program.

The probability of mutation is also a parameter of the program. For each chromosome chosen for mutation, the following actions are performed : a trajectory is randomly[7] chosen among the n ones of the chosen chromosome, and replaced by a new trajectory computed using the A^* algorithm described in the previous section. If no such trajectory is found, random modifications are introduced in the initial trajectory.

5.2 GA results

Run	1	2	3	4	5	6	7	8	9	10
$\mathcal{F}_{best}$	1.1897	1.1897	1.1901	1.1897	1.1885	1.1885	1.1897	1.1893	1.1901	1.1885
Cost	51.735	51.735	51.620	51.735	52.068	52.068	51.735	51.842	51.620	52.068
Gen. nb	71	125	116	89	75	78	109	98	84	78
Time (s)	591.03	1307.13	1033.93	848.83	742.89	894.82	1124.39	1107.01	714.64	728.49

Table 1: GA results for flows on a semi-circle (SC)

On the semi-circle problem, the genetic algorithm was run ten times with different seeds for the random generator, with a population of 250 elements evolving over 150 generations. The probability of crossing is 0.6, and the probability of mutation is 0.05. Table 1 shows for each run the fitness of the best element, the corresponding cumulated cost, and the generation number at which the best element was found. We see that all solutions found by the genetic algorithm are better than, or equivalent to, the solution found by the A^* (cost 52.068 with a corresponding fitness of 1.1885). Figures 6 and 7 show the best solution found by the genetic algorithm. We see that the lateral deviations of trajectories 8 and 9 allow slighly different vertical deviations for these trajectories than in the A^* solution, thus allowing to separate the trajectories using 8 flight levels instead of 10.

6 Conclusion and further research

Both algorithms exhibit good results on our simplified problem. The *global strategy* usually finds better results with the genetic algorithm than the 1 *vs.* n strategy with the A^*, which was to be expected as the latter does not aim at solving the global problem.

[7] A more elitist strategy may be used, by choosing the trajectory with the worse local fitness

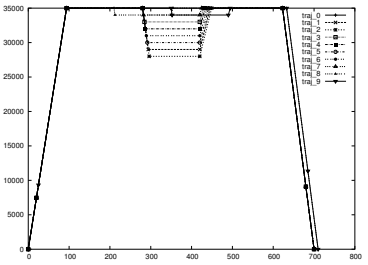

Figure 6: *Side view of GA results for 10 flows on a semi-circle (SC)*

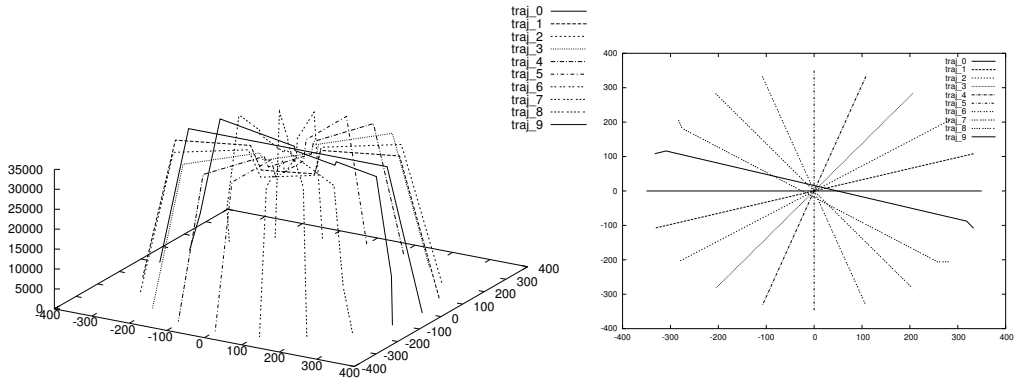

Figure 7: *GA results for 10 flows on a semi-circle (SC)*

The algorithms have already been successfully applied to real traffic over France and Europe, involving several modifications in the model. For example, the case of traffic flows which do not start and/or end within the considered geographic areas must be handled : entry and exit flight levels must be added to the trajectory description. These results will be described in a future paper.

Further research may deal with the flow definition, and particularly in the time dimension. It would not make sense to allocate a full 3D trajectory during the whole day to a flow with a few flights early in the morning and another few in the evening, for example. The fact that all aircraft do not have similar flight performances should also be taken into account. These real-life constraints may slightly change our model of traffic flows. The great number of origin/destination pairs in real traffic (310 flows of more than 10 aircraft per day on the 21st may 1999, in France) also adds some difficulty, although we expect to avoid it by splitting flows along the time axis. However, this does not change the nature of the results presented in this paper.

References

[1] Performance Review Commision. Performance review report, an assessment of air traffic management in europe during the calendar year 2000. Technical report, Eurocontrol, 2001.

[2] D. Bertsimas and S. Stock Patterson. The traffic flow management rerouting problem in air traffic control: A dynamic network flow approach. *Transporation Science*, 34(3):239–255, August 2000.

[3] D. Delahaye and A. Odoni. Airspace congestion smoothing by stochastic optimization. In *Evolutionary Programming VI*, 1997.

[4] N. Barnier, P. Brisset, and T. Rivière. Slot allocation with constraint programming: Models and results. In *Proceedings of the fourth USA/Europe Air Traffic Management R&D Seminar*, 2001.

[5] M.R. Jardin. Real-time conflict-free trajectory optimization. In *Proceedings of the fifth USA/Europe Air Traffic Management R&D Seminar*, June 2003.

[6] C.H.M. van Kemenade, C.F.W. Hendriks, H.H. Hesselink, and J.N. Kok. Evolutionnary computation in air traffic control planning. In *Proceedings of the Sixth International Conference on Genetic Algorithm*. ICGA, 1995.

[7] Karim Mehadhebi. A methodology for the design of a route network. In *Proceedings of the Third Air Traffic Management R & D Seminar ATM-2000*, Napoli, Italy, June 2000. Eurocontrol & FAA.

[8] Vincent Letrouit. *Optimisation du réseau des routes aériennes en Europe*. PhD thesis, Institut National Polytechnique de Grenoble, 1998.

[9] Nicolas Barnier and Pascal Brisset. Graph coloring for air traffic flow management. In *CPAIOR'02: Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, pages 133–147, Le Croisic, France, March 2002.

[10] L. Maugis, J.-B. Gotteland, R. Zanni, and P. Kerlirzin. TOSCA-II - WP3: Assessment of the TMA to TMA hand-over concept. Technical Report TOSCA/SOF/WPR/3/03, SOFREAVIA, 1998.

[11] Nicolas Durand. *Optimisation de trajectoires pour la résolution de conflits aériens en route*. PhD thesis, Institut National Polytechnique de Toulouse, 1996.

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

[13] T. Baeck, D.B. Fogel, and Z. Michalewicz. *Evolutionary Computation 1 : Basic Algorithms and Operators.* Institute of Physics Publishing, 2000.

[14] T. Baeck, D.B. Fogel, and Z. Michalewicz. *Evolutionary Computation 2 : Advanced Algorithms and Operators.* Institute of Physics Publishing, 2000.

[15] J.H Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan press, 1975.

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

[17] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.

[18] Yann Le Fablec. Optimisation par algorithmes génétiques parallèles et multi-objectifs. Master's thesis, Ecole Nationale de l'Aviation Civile (ENAC), 1992.

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