https://doi.org/10.37420/j.eeer.2024.003

Application of Compact Representation of Ordered Set to Air Traffic Conflict Resolution

Ruixin Wang[1*] ,Nicolas Barnier[2]

[1]Laboratory of Complex System Safety and Intelligent Decisions, CAUC-ENAC Joint Research Center of Applied Mathematics for ATM, Civil Aviation University of China, Tianjin, China,wang@recherche.enac.fr

[2]ENAC Lab, Université de Toulouse, France,nicolas.barnier@recherche.enac.fr

[*]Corresponding author,E-mail:ruixin.wang@recherche.enac.fr

Abstract

The motivation of implementing Adaptive Compact Tree is to improve the performance of Constraint Programming(CP) solvers operations, which is used for en-route conflict resolution in. AC-Trees could not only help CP solver to solve the conflict problem much faster, but also outperform classic BSTs on standard set operations. In addition, we will also present optimizations that improve the performance of the conflict detection for the conflict problem previously mentioned. With these optimizations, the conflict detection phase can be completed in a few seconds. Together with our contribution of the ACTree data structure used in CP solver which accelerates conflict resolution, the total execution time has been reduced significantly, which opens the way to a real-time implementation in an operational context.

Keywords: adaptive data structure, constraint programming, conflict resolution

I INTRODUCTION

Currently, many CP application have been proposed to solve ATM problems. Allignol et al. (2012) Have classified these categories in three parts: strategic, tactical and operational management. In addition,(Allignol et al., 2013) implements a CP algorithm to solve an en-route conflict problem, which uses range sequence as domain representation.

Presented a new framework for the generation and resolution of air traffic conflict problem, which can detect and solve conflicts for almost all instances inside the benchmark indicated in (Allignol et al., 2013) (i.e. from 5 to 20 aircraft with 151 maneuvers).

The production of instances is highly configurable: the density of the conflicts, the number of authorized maneuvers and the level of uncertainty are taken into account. The output is a conflict matrix containing all the undesirable trajectories for all pairs of aircraft with all pairs of maneuvers. (Allignol et al., 2013) proposes two methods for the resolution of the conflicts: Evolutionary Algorithm and Constraint Programming. Most of these cases were solved in less than 10 seconds, however, for the harder ones, it will take a few minutes to solve. With the CP algorithm, the optimality proof has been obtained in most cases and cases without solution have been proven so in less than a second.

As generation of instances is separated from conflict resolution, different algorithms can be tested to solve this problem by taking the conflict matrix as inputs, which is obtained during the conflict detection process. But, in a real-time operational context, the total execution time, including conflict detection and resolution, should be short enough to provide the controllers with optimized maneuvers for all aircraft that solve all the conflicts. We present some optimizations in the conflict detection process in section 2.1 to minimize the detection time. In addition, just as previously mentioned, gap intervals trees, the complement of range sequences, could be implemented in a CP solver to reduce resolution time. Nevertheless, gap intervals trees are unbalanced, so their operations are efficient only if elements are inserted in random order which is generally not the case in en-route conflict problems. Hence, we implement a new data structure called AC-Tree, similar to the self-balanced binary search tree, replacing range sequences (i.e.the domain representation used in FaCiLe CP solver) to get better performances for the solver described (Allignol et al., 2013) .

Obviously, an effective conflict solver relies on a realistic trajectory prediction. In the benchmark framework proposed in (Allignol et al., 2013) , finding future positions of aircraft can be done using any simulator taking into account different uncertainties such as heading change, wind, beginning and ending maneuver positions, etc. Once the future positions are found, an algorithm proposed in (Allignol et al., 2013) can detect conflicts for each pair of trajectories and store this information in a conflict matrix C which specifies the problem entirely and is given as input to the solver. In section 2.1, we will present trajectory model, instances generation and conflict detection where i have made many contributions and compare the resolution time using range sequence or AC-Tree data structure in section 2.2.

2 AIR CONFLICT RESOLUTION

Give realistic model to build the trajectories at each time step according to the chosen maneuver options and the uncertainties taken into account. To be compatible with current ATC practice and FMS capabilities, a limited number of maneuvers is defined for each aircraft involved in a conflict, then each pair of maneuvers for two aircraft are tested to verify if there are conflicts between them or not. However, the conflict detection is not fast enough to realize a real-time system. So, we will firstly introduce the model of trajectory prediction in section 2.1 and present three optimizations to accelerate conflict detection: using Trees to describe aircraft trajectories, applying Bounding Box to approximate the convex hulls used to represent the possible positions of aircraft and parallelizing the conflict detection operations in a more effective way section 2.1.

Firstly, we give a clear definition of 'En-route conflict':

Figure 1 represents the volume of the inherent protection in the standard separation for en-route phase of flight.

Definition 3 (En-route conflict between two aircraft) The en-route conflict between two aircraft (i, j) occurs, if the aircraft j is inside the volume of aircraft i.

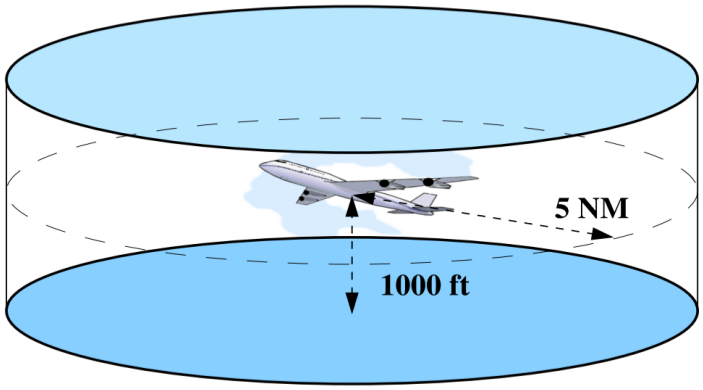

Figure 1 Standard separation for en-route phase. No other aircraft could be found in this volume.

2.1 Trajectory Prediction

We detail here the uncertainty model, the maneuver options and how the convex hulls of trajectories are built for every time step.

Maneuvers

The trajectory prediction model will use a discretization of time into steps of duration τ (10 s in our experiments) to describe maneuvers, which is chosen small enough to catch every possible conflict.[1]

Trajectories are described in the horizontal plane. Initial routes are defined by a list of points, the first point O is the origin and the last point D is the destination (i.e. a segment of trajectory between two waypoints). Thanks to the Flight Management System of aircraft, several errors can be corrected and do not increase with time, however, various other sources of uncertainties cannot be reduced and must be taken into account to obtain a realistic model.

In this trajectory model, maneuvers (e.g. heading changes) are engaged on a point of the initial trajectory referenced by decision variable d0 which represents the distance from the origin O. A distance error ε0 is added around this point which means that aircraft may change its heading ε0 nautical miles before or after d0 as a result of uncertainties on the exact location of the turn. An uncertainty εα is also associated to the heading change angle α at the turning point corresponding to d0. Then the maneuver ends at a curvilinear distance d1 from d0 (i.e. at d0 + d1 from the origin O) with an associated error ε1, when the aircraft returns towards its destination point D.

This simple maneuver, depicted in figure 2, is representative of current Air Traffic Control practice and can be easily implemented by pilots and current FMS technologies.

1 Two facing aircraft flying at 450kt get only 1.25NM closer every 10s, so no conflict will be missed with such a small τ value.

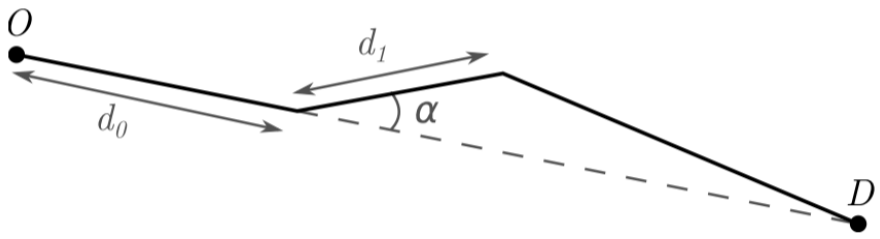

Figure 2 Maneuver Model.

If taking n0 = 5 values for d0, n1 = 5 values for d1 and nα = 7 possible angles i.e. 0, 10, 20 or 30 degrees to the left or the right of the current heading (there is no use to combine a null heading change α = 0 with various d0 and d1 values, so that only one maneuver is added when the aircraft is not deviated), the number of maneuvers for each aircraft is:

$$n_{man} = n_0 \times n_1 \times (n_\alpha - 1) + 1$$

So for the values chosen above, nman = 5 × 5 × 6 + 1 = 151. For an instance with n aircraft, the search space is then of size nmann, i.e. ≈ 6.1021 for a 10-aircraft instance.

Decision Variables

To simplify the access to the conflict matrix C and reduce the number of combinations to the useful ones (e.g. only one possible maneuver for α = 0), the three decision variables d0, α and d1 associated with aircraft i are aggregated into a single decision variable mi by a bijection from the allowed triples to interval [1, nman]. We call M the set of decision variables of the problem:

$$M = \{m_i, i \in [1, n]\} \quad (1)$$

Handing Uncertainties

In [1], the trajectories envelop are built in order to be able to detect conflicts between two maneuvers for two different aircraft, while taking various uncertainties into account. The maneuvers descriptions are stored in a table that defines for each aircraft and each maneuver the possible future positions of the aircraft by their convex hull, which is computed with Graham's algorithm.

Each aircraft position is described at multiples of the time step τ (i.e. 0, τ, 2τ, 3τ . . .) by three convex hulls corresponding to the three possible states of the aircraft:

- S_0 if it has not been maneuvered yet;
- S_1 if it is currently maneuvered;
- S_2 if it is heading towards its destination after a maneuver.

Once the three convex hulls are defined for every time step, they are merged in a single one whenever several envelops coexist for the same time step (e.g. around turning points of the trajectory).

Actually, different aircraft could have different uncertainties and different maneuver options according to their ability to follow a route. We only need a convex hull of the possible future positions of an aircraft at every time step of the trajectory [1].

2.2 Conflict Detection

As presented in [1], the trajectory prediction calculated in the previous section will be used to produce the data for the benchmark described in this section, generated by detecting conflicts between trajectories. Then three optimizations that improve the efficiency of the conflict detection are presented.

Trajectory Tree

To optimize conflict detection, the trajectories of aircraft can be represented by a prefix tree, which is an ordered tree data structure, often used to represent a set of strings in a compact way and be able to efficiently retrieve words with a common prefix. In a prefix tree, all the descendants of a node have a common prefix obtained with the path from the root to the node, the root itself being associated with the empty string. As trajectories of an aircraft can be seen as strings of convex hulls of the same length, they can be efficiently stored and processed with such a data structure.

Once the trajectory prediction described in section 2.1 has been finished, we first start with one node R as the root of the tree storing the convex hull of the aircraft at t = 0 which is shared by all its trajectories. The children of a node associated with time step t are recursively created with all the convex hulls at t + τ (i.e. τ is the duration between two successive time steps, see section 2.1) and each child corresponds to a different maneuver option. Once we reach the last time step, the nodes are leaves of the tree.

Figure 3 gives an example of trajectory tree with 5 possible trajectories (nman = 2 × 1 × 2 + 1 = 5):

- for each time step, there exists several different nodes that contain different possible convex hulls (e.g. at t = 5τ, 3 possible nodes represent the convex hulls of turning left, turning right and going straight);
- each node describes the same convex hull for several trajectories: e.g. the green node in figure 3.5 contains the convex hull for all the trajectories at t = 4τ and the yellow node contains the convex hull for trajectories 2, 3 and 4 at t = 5τ.

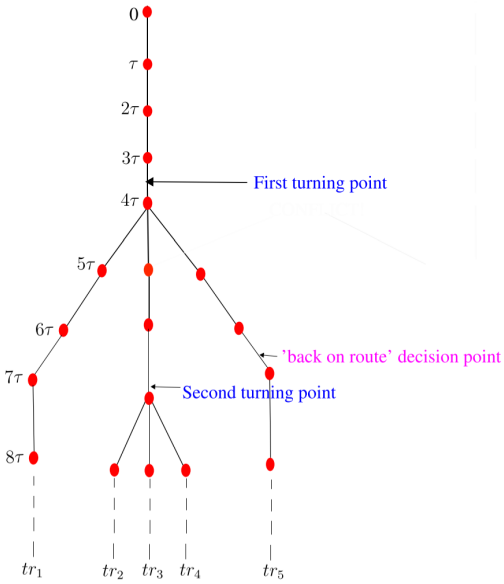

Figure 3 A tree trajectory with nman = 5

Trajectory trees can then be used to improve the efficiency of conflict detection in the following way:

lCompare the root of Trajectory Tree of one aircraft to the other one;

lIf the envelops stored in these two nodes which represent the possible positions are not correctly separated, all the trajectories associated with the corresponding subtrees will be in conflict, and then cartesian product is added to the result of the conflict detection;

lOtherwise, it has to check all the children of the first trajectory tree against all the children of the second.

This algorithm helps us to minimize the number of trajectories to compare: once a node is already in conflict with another one, the other trajectories that share this node don't have to be tested in the future. Figure 4 shows the comparison of execution time with different trajectory representations to detect conflicts. In green, there are nman independent trajectories corresponding to all the possible maneuver options for each aircraft. In orange, all the trajectories of one aircraft are described by only one trajectory tree containing

all maneuver options. With Trajectory Tree, it's able to detect all the conflicts almost 20 percent faster than with the independent trajectory representation.

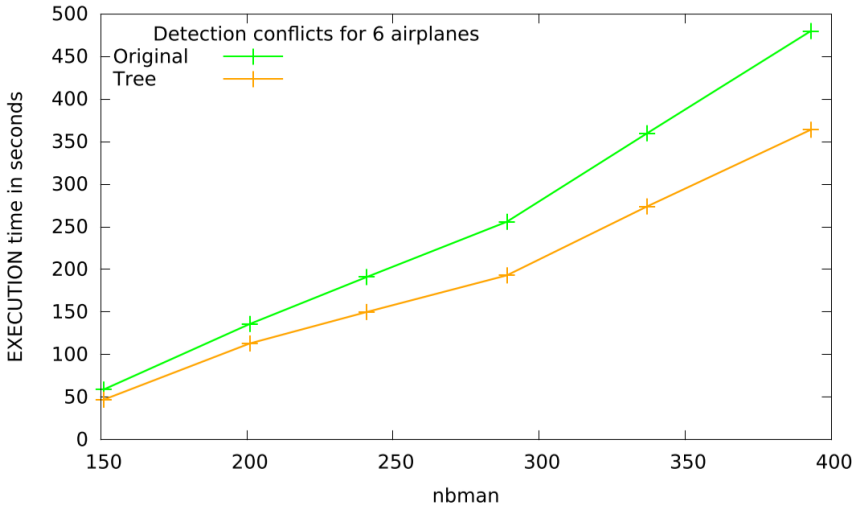

Figure 4 Comparison between Trajectory Tree and independent representation.

Bounding Box

During the conflict detection process, the distance between the two two convex hulls stored in the two nodes representing the possible positions of two aircraft is computed and compared to the standard separation norm (5 NM).

In most cases, convex hulls are irregular polygons with a lot of vertices and edges, so to test whether two convex hulls are separated correctly is quite time-consuming. In order to accelerate computation, we present Bounding Box to describe the envelop of the aircraft more concisely. Once all the the convex hulls are calculated, we use a regular rectangle to bound the convex hull. As a rectangle is a regular polygon with only four vertices and edges with all the edges parallel with x or y axis, we just need to compare the lower bound of a node to the upper bound of the other one along the x and y axes. If one of these two bounds are separated correctly, these two nodes aren't in conflict.

Algorithm 1 shows in pseudocode how the conflicts are detected between two trajectory trees of two aircraft using tree representation and bounding box. This detect_conflict recursive function first starts from two trajectory tree roots, if they are in conflict, all the trajectories associated with the corresponding subtrees will be in conflict and returns all the couples of conflicting trajectories; if not, it will check all the children of the first trajectory tree against all the children of the second. At last, it returns the set of all the couples of conflicting trajectories for these two aircraft.

- bbox(n) returns the bounding box stored in node n;
- in_conflict(bb1,bb2) returns a boolean indicating if the bounding box bb1 is in conflict with the bounding box bb2;
- trajs(n) returns all the indices of trajectories that share node n;
- cartprod(trajs1,trajs2) returns the cartesian product of indices sets of trajs1 and trajs2;
- children(n) returns all the children nodes of node n at the next time step.
-

```
set detect_conflict(node n1, node n2):
  if in_conflict(bbox(n1),bbox(n2)) then
      return cartprod(trajs(n1),trajs(n2))
  else
      nogoods =: empty
      for every c1' in children(n1) do
        for every c2' in children(n2) do
          nogoods =: union(detect_conflict(c1',c2'),nogoods)
        end
      end
  return nogoods
```

Algorithm 1 Conflict detection between two trajectory trees

Figure 5 shows the comparison of execution times using Convex Hulls and Bounding Box to detect conflicts. This graph confirms that using Bounding Box can save almost 85% of the computation time.

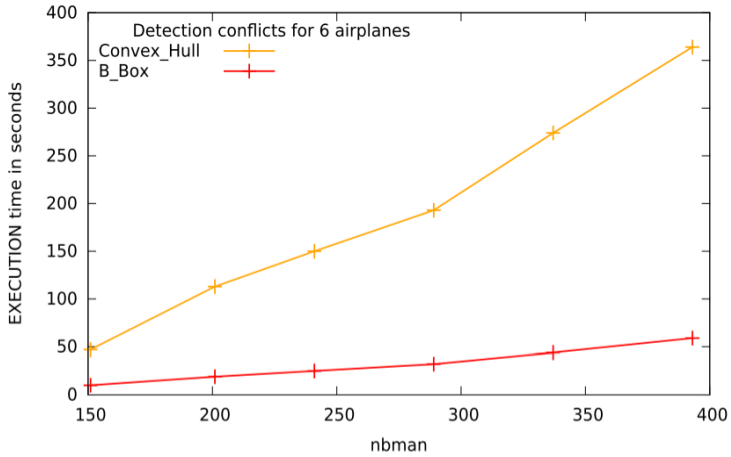

Figure 5 Computation time comparison between Vertex Hull and Bounding Box.

Parallelization

To speed up computation times, the conflict detection can be parallelized very easily: each pair of the aircraft can be tested on a separate processor. All the trajectories are computed by a "master" program on a single processor and then sent to a number of "slave" programs executed on multiple processors. Then the master program dispatches the pairs of aircraft to check to the slave program.

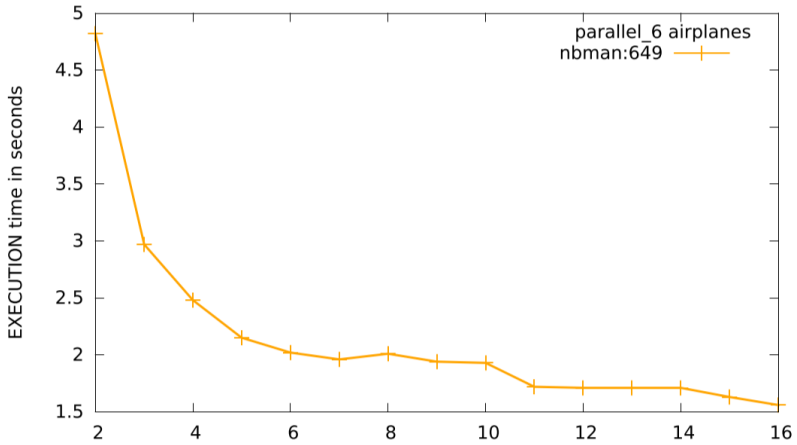

Figure 6 Computation time using different number of processors.

Actually, for an en-route conflict problem with n aircraft, there are n(n 1)/2 trees to be tested. Figure 6 shows the execution time of conflict detection for 6 aircraft (i.e. 15 pairs of trajectory trees) with different number of processors (i.e. from 2 with only one "slave" programs). Generally, the execution time reduces with the augmentation of number of processors and reaches its minimum of less than 1.6s when using 16 processors at the same time. However, as can be observed in figure 6, execution time obviously does not drop linearly with the number of processors. In fact, even if we consider that conflict detection takes exactly the same time tnorm for every pair of trajectory trees, using 12 or 13 "slave" programs will both cost 2 tnorm for a total of 15 pairs of trajectory trees (i.e. $1 < 15/12 < 2$ and $1 < 15/13 < 2$). There must be at least 15 slaves for the execution time to decrease to tnorm, as it is bounded by [nbpairs/nbproc]*tnorm. Moreover, the computation times for various pairs of trajectory trees differ and cannot be predicted, so relationship between the execution time and the number of processors is combinatorial.

2.3 Conflict Resolution with Constraint Programming

We have used AC-Trees to replace the range sequences that represent finite domains in the FaCiLe CP solver [4] and solved the conflicts generated in section 2.2.

CSP Model

The set M of decision variables of the CSP is the one defined by equation (1) in section 2.1, where each variable mi is the index of the maneuver for aircraft i and thus takes a value in [1, nman].

The constraints are expressed as binary constraints, i.e. constraints involving exactly two variables. For a given couple of aircraft i and j ($i < j$), the constraint cij between variables mi and mj is defined as the set.

Solution Search

The exploration of the search space is based on an enhanced version of a systematic tree-search algorithm called backtracking, where an inference phase prunes the unfeasible values of each variable at every node of the tree by propagating local consistency properties in the constraint network. In our algorithm, the search tree is explored by following the weighted degree adaptive heuristic which learns from the failures during the search, so that the variables involved in the constraints that have been violated the most so far are instantiated first. This heuristic proved to be particularly efficient on this problem, as it dynamically focuses on the hardest parts of the CSP first.

The optimization criterion c simply is the sum of the costs of each single maneuver. The optimization algorithm used to solve the CSP is an adaptation of the backtracking algorithm called branch and bound: each time a solution with cost cs is found, the constraint $c < cs$ is dynamically added to the CP model, and the search is resumed to look for a better solution. Eventually, the search for a better solution will fail, proving that the best solution so far was optimal (or, if no solution has been previously found, that there is no solution satisfying all the constraints). In order to quickly obtain solutions of good quality, which is mandatory in an operational real-time context, our search strategy first focus on maneuvers that least increase the cost.

Figure 7 shows the comparison of execution times with range sequences (in solid line) or AC-Trees (in dashed line) to solve the problem. With AC-Trees, FaCiLe is up to 1.5 times faster than with range sequence.

https://doi.org/10.37420/j.eeer.2024.001

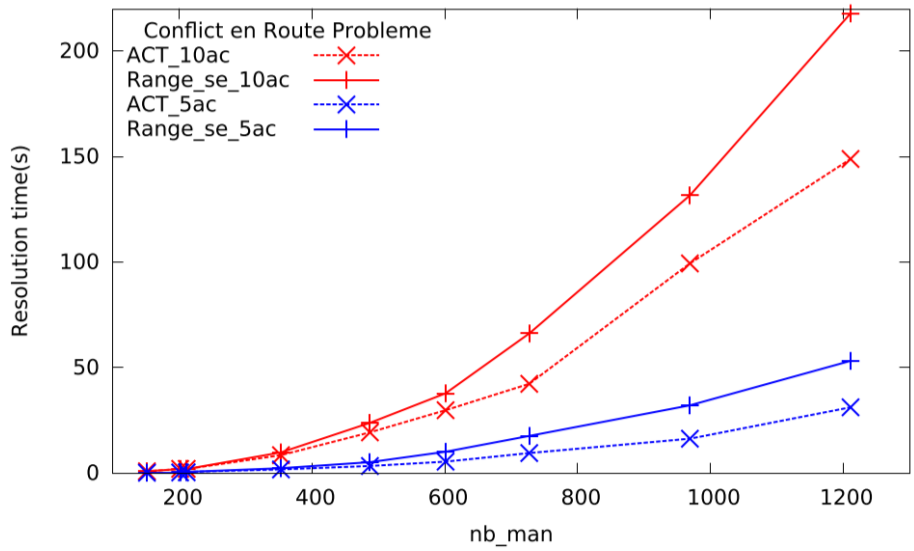

Figure 7 Resolution time comparison between Range Sequence and AC-Tree.

3 Conclusion and Further Works

- we present three optimizations to the framework proposed in Allignol et al. (2013) for solving en-route conflicts:
- using trees to describe aircraft trajectories;
- applying Bounding Boxes to approximate the convex hulls used to represent the possible positions of aircraft;
- parallelizing the conflict detection operations in a more effective way.

With these three optimizations, the conflict detection phase can be completed in a few seconds (e.g. 1.6s for an en-route conflict problem including 6 aircraft with 649 maneuvers), which could be compatible with a real-time system. Moreover, we have applied AC-Trees to represent finite domains in the FaCiLe CP solver and solved the conflict problem much faster than with range sequences. Finally, the total execution time including conflict detection and resolution has been reduced significantly by our optimizations which opens the way to a real-time implementation in an operational context.

However, to further optimize the conflict detection phase, a sweep and prune algorithm Baraff (1992) could also be used to compute the conflicting bounding boxes very efficiently.

ACKNOWLEDGMENTS

[1]This article is supported by diversified investment fund of Tianjin under Grant 21JCQNJC00790.

REFERENCES

Allignol, C., Barnier, N., Durand, N., & Alliot, J.-M. (2013). A new framework for solving en-routes conflicts. In ATM 2013, 10th USA/Europe Air Traffic Management Research and Development Seminar.

Allignol, C., Barnier, N., Flener, P., & Pearson, J. (2012). Constraint programming for air traffic management: A survey. The Knowledge Engineering Review, 27(03), 361–392.

Baraff, D. (1992). Dynamic simulation of non-penetrating rigid bodies. Technical Report. Cornell University.

Barnier, N., & Brisset, P. (2001). FaCiLe: A Functional Constraint Library. In CICLOPS – Colloquium on Implementation of Constraint and Logic Programming Systems, CP'01 Workshop, Paphos, Cyprus, December 2001.