Concurrent constraint programming-based path planning for uninhabited air vehicles

Stefano Gualandi*, Bruno Tranchero**
*Dept. of Computing Science and Engineering, Université Catholic de Louvain,
B-1458 Louvain-la-neuve, Belgium, stegua@info.ucl.ac.be
** Alenia Aeronautica - Advanced Information Technology Laboratory,
10146 Torino, Italy, btranchero@aeronautica.alenia.it

ABSTRACT

This paper describes a study performed with the objective of investigating Concurrent Constraint Programming (CCP) as the main tool for the design and the implementation of a software path planner. The main features of the path planning are introduced along with some modeling and implementation issues. The CCP approach is motivated by the facility of translating complex models with domain specific features into efficient implementations. In order to use CCP, the path planning is formalized as a constraint satisfaction problem by defining variables, domains, and constraints. The proposed solution is as general as possible, and it is widely applicable in several contexts. A demo application has been implemented, and it is described in this paper. The application comes with a graphical interface that allow the user to define the mission constraints. The output of the application is a path plan, i.e. a list of waypoints that can also be displayed on a geographical map. The estimated path length, fuel consumption, and path risk are given as well.

1. INTRODUCTION

Path planning is a well-known problem in many domains, ranging from robotic to transportation in general. In air transportation, it becomes relevant when defining an air vehicle mission. Due to all the domain constraints to be considered, it is a time consuming task, and usually it requires experienced people. Thus, the possibility of having an automatic path planner is very interesting. During the last twenty years, flight path planning has been tackled many times using different approaches and technologies. The goal was to support, during the flight mission, both the human planner on the ground and the pilot on the aircraft. Today there exist decision support systems for mission planning, but most of them are located at the ground-based station.

The problem of supporting path planning during the flight is still open. Nevertheless, this aspect is becoming increasingly important when considering Uninhabited Air Vehicles (UAV) with a high degree of autonomy. Nowadays, in the commercial UAV systems there is no pilot, but human operators control them remotely ([21], [19]). Anyway, there is a strong industrial interest in highly autonomous systems with onboard re-planning capability. In navigation, the main interest concerns the possibility of having an automatic path planner, capable of tracing a flight route within a bounded amount of time.

1.1. Path Planning

The principal task of navigation consists of finding a route on a map from a departure position to a target, while minimizing a cost function. When planning for air vehicles, flight configuration constraints (i.e., speed, threat exposure, fuel consumption, maximal turn angle) must be taken into account. Weather and atmospheric conditions have to be considered as well.

Several classes of algorithms have been tested to solve path planning. The classical approach consists of algorithms taking as input a graph. Each node in the graph is a possible navigation point (or waypoint). The edges of the graph are possible legs of the flight path. The path planning is therefore a search process on the graph performed with uninformed or heuristic search algorithms. The most popular algorithms for path planning are based on Dijkstra and A* ([15], [8]). Nevertheless, these approaches need a big development effort to be adapted to different vehicles and/or mission requirements. In addition, it is difficult to define a heuristic function that correctly evaluates all the domain constraints that must be satisfied.

There are path solvers for mission planning that are located at the ground-based stations, but human pilots still perform path re-planning by using their own experience. A highly autonomous platform, such as a UAV, will need a fully functional path planner in order to increase its autonomy level. During mission execution, both can happen weather

conditions could change or failures in the aircraft systems could occur. In these cases, the amount of time required to repair or re-plan a new route is crucial. An UAV path solver should be able to react to such critical situation in a very few seconds. It should also interact with the Flight Management System to gather data about the current status of on-board systems, the available amount of fuel, the possible threats, and other flight related information.

1.2. Related work

Mitchell [10] suggests three general types of the shortest path problem:

1. *Obstacle avoidance in the plane*. Maps are binary representation of the space, where in some positions it is impossible to fly over. The length of the path is calculated in two dimensions (i.e., it is a planar length).
2. *Shortest path on a surface*. Maps give information on terrain elevation, and the path is define on a surface. Therefore, the length of the path is calculated in three dimensions.
3. *Weighted regions*. Maps describe regions with different cost (weight) for unit of distance. The length of the path is measured as a weighted length, that is the length time's weight.

The model described in this paper was initially designed to solve the obstacle avoidance in the plane. Then, after having refined the model and added new constraints, the shortest path on a surface was addressed. Finally, the weighted regions problem has been tackled.
A common issue when modeling a navigation planner is the representation of the search space, that is the environment where the vehicle has to navigate. There are mainly two ways of representing the search space: by regular cells (grid) and by polygons. The first method requires a large amount of memory in order to store big geographical regions, and it is not really convenient when small objects have to be represented [10]. Nevertheless, it is easy to manage. The polygons-based method does not require much memory while allowing an accurate description of the objects. However, the pre-processing computation has a considerable complexity. Development of planning algorithms is strictly correlated to the search space representation.
A robust algorithm for real-time aircraft route planning is described in Szczerba [18]. This algorithm solves the weighted regions problems using a regular grid representation of the search space. The Digital Terrain Elevation Data (DTED) is used as input to the route planner. First, a pre-processing computation constructs a Sparse A* Search Tree (SAST) considering some aircraft configuration constraints. After that, the tree is visited by a modified A* algorithm, called Sparse A* Search.
Mata and Mitchell [9] have defined an algorithm for computing shortest path in weighted planar regions. Their work is relevant because it uses a polygonal space representation. However, constraint satisfaction is not considered. The problem of path planning is translated into the problem of describing the search space in an optimized form. The authors have implemented an algorithm that starts from a planar polygonal subdivision and constructs a relative sparse graph, called pathnet. Once the graph is defined, the Dijkstra's shortest path algorithm is applied.
A first attempt to use constraint programming to approach path planning is described by Allo [1]. They have implemented a planner that takes as input a graph where the nodes are the possible navigation point. Then, the path planning is modeled as a constraint satisfaction problem on the graph, and it is resolved by using constraint programming. They domain of the variables range over the possible navigation points, and the constraint considered are the same that we have taken into account in our planner (see next sections).
Finally, Quesada [14] solves the path planning by using concurrent constraint programming, and his approach is similar to the previous one. The path planning is modeled as a constrained path problem. Quesada proposes efficient implementations of the constraints for improving the performance of the path planner. He also shows that the problem we aim at solving is NP complete, that is a combinatorial problem of exponential complexity.

1.3. Overview of CCP

Concurrent Constraint Programming is a software technology that allows solving combinatorial problem efficiently ([16], [3], and [17]). The problem with combinatorial problems is that the number of combinations to be considered may grow exponentially. Therefore, it is not possible to generate and test all the combinations. Generate and test quickly leads to a combinatorial explosion even for small problems. CCP helps to avoid the combinatorial explosion. The idea is to perform simple inferences first in order to avoid superfluous case distinctions. This method is called '*propagate and distribute*' where *propagation* performs simple inferences and *distribution* case distinctions.

Constraint Store and Propagators. Most combinatorial problems can be modeled naturally by using some kind of logic formulas. We call logic formulas constraints. Two forms of constraints can be distinguished: basic constraints are so simple that they can be solved deterministically, whereas complex constraints cannot. Typical examples for basic constraints are symbolic equations as used in first-order unification. Good examples for complex constraints are arithmetic equations. For what concerns CCP, basic constraints are accumulated incrementally in a constraint store whereas complex constraints are turned into propagators. Propagators are agents that observe the constraint store. A propagator infers consequences of the constraints in the store and the formula by which it is defined. Such consequences are new constraints, which can be either basic or complex again.

An example. To see the application of the *propagate* and *distribute* consider the problem specified by the following constraints [4]:

$X \neq 7$ $\quad Z \neq 2$ $\quad X - Z = 3 \cdot Y$

$X \in [1\#8]$ $\quad Y \in [1\#10]$ $\quad Z \in [1\#10]$

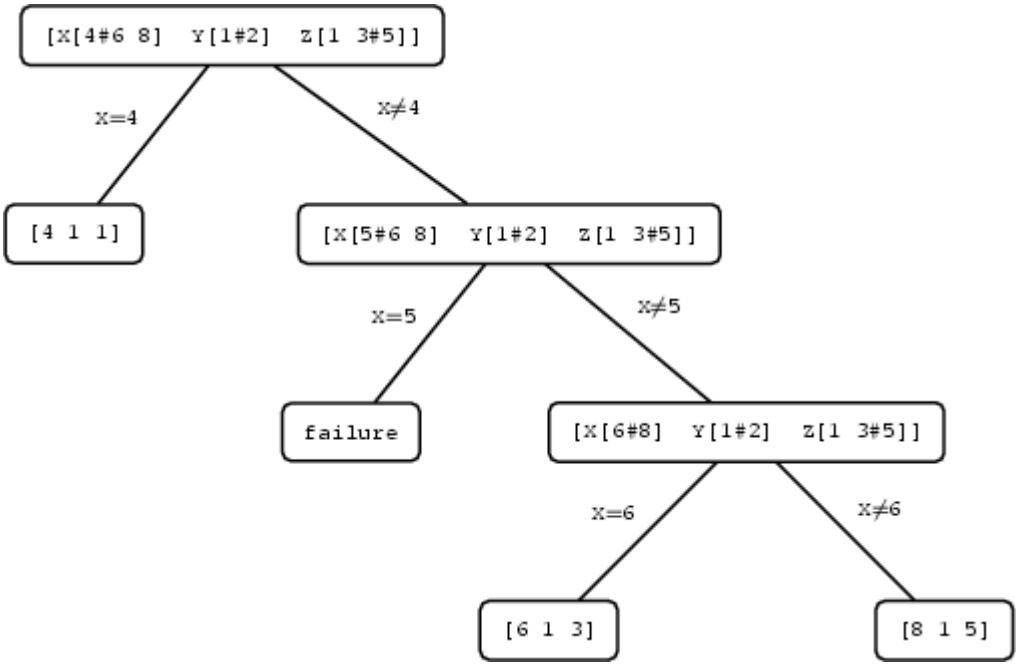

Figure 1 - A Search tree containing 3 choice nodes, 1 failure node, and 3 solution nodes.

To solve the problem, we start with a space whose store constrains the variables *X*, *Y*, and *Z* to the given domains. We also create three propagators imposing the constraints $X \neq 7$, $Z \neq 2$, and $X - Z = 3 \cdot Y$. We assume that the propagator for $X - Z = 3 \cdot Y$ realizes interval propagation, and that the propagators for the inequalities $X \neq 7$ and $Z \neq 2$ realize domain propagation.

The propagators for the inequalities immediately write all their information into the store and disappear. The store then knows the domains

$X \in [1\#6\ 8]$ $\quad Y \in [1\#10]$ $\quad Z \in [1\ 3\#10]$

where $[1\ 3\#10]$ denotes the finite domain $\{1\} \cup \{3,\ldots,10\}$. The interval propagator for $X - Z = 3 \cdot Y$ can now further narrow the domains to

$X \in [4\#6\ 8]$ $\quad Y \in [1\#2]$ $\quad Z \in [1\ 3\#5]$

Now the space is stable but neither failed nor solved. Thus, we continue with a first distribution step. We choose to distribute with the constraint. Figure 1 shows the resulting search tree.

The space obtained by adding a propagator for $X = 4$ can be solved by propagation and yields the solution

$X = 1$ $\quad Y = 1$ $\quad Z = 1$

The space obtained by adding a propagator for $X \neq 4$ becomes stable immediately after this propagator has written its information into the constraint store, which then looks as follows:

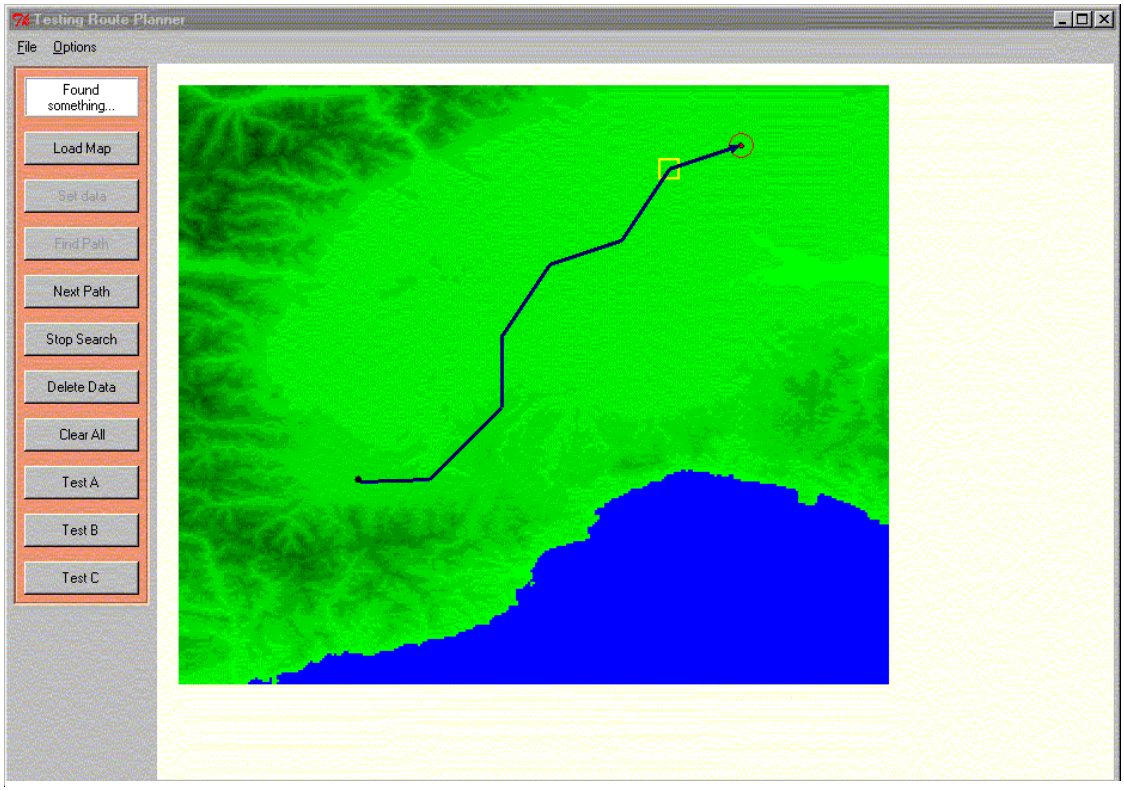

Figure 2 - An example of a path generated in a scenario without threats.

$$X \in [5\#6\ 8] \qquad Y \in [1\#2] \qquad Z \in [1\ 3\#5]$$

This time we distribute with respect to the constraint $X=5$. The space obtained by adding a propagator for $X=5$ fails since $X - Z=3 \cdot Y$ is inconsistent with the store obtained by adding $X=5$. The space obtained by adding a propagator for $X \neq 5$ becomes stable immediately after this propagator has written its information into the constraint store, which then looks as follows:

$$X \in [6\ 8] \qquad Y \in [1\#2] \qquad Z \in [1\ 3\#5]$$

Now we distribute with respect to the constraint $X=6$. The rest of the search process can be seen in Figure 1.

In order to use CCP for solving the path planning, a model of the problem as a set of variables, domains, and constraints has to be formalized. In the following section, the specification of the path planner is introduced.

2. SPECIFICATION OF THE PATH PLANNER

This section introduces a possible approach to path planning by using CCP. The specification of the problem is clearly identified. The terrain, the threats, and the fuel consumption models are described too. After that, some important technical issues are considered.

2.1. Requirements

The main objective of the planner is to find a feasible plan (i.e., a flight route) that respects all the constraints. The path planner is designed considering the UAV mission requirements, but the results can be easily transferred to other class of navigation problems. The main imposed constraints are:

(A) the path planner must trace a route from a start point to a target, flying over a given position, and avoiding threats;

(B) the flight path must keep the same direction for a minimal distance (minimal leg length);

(C) the flight path can not exceed a maximum (eventually a minimum) turn angle;

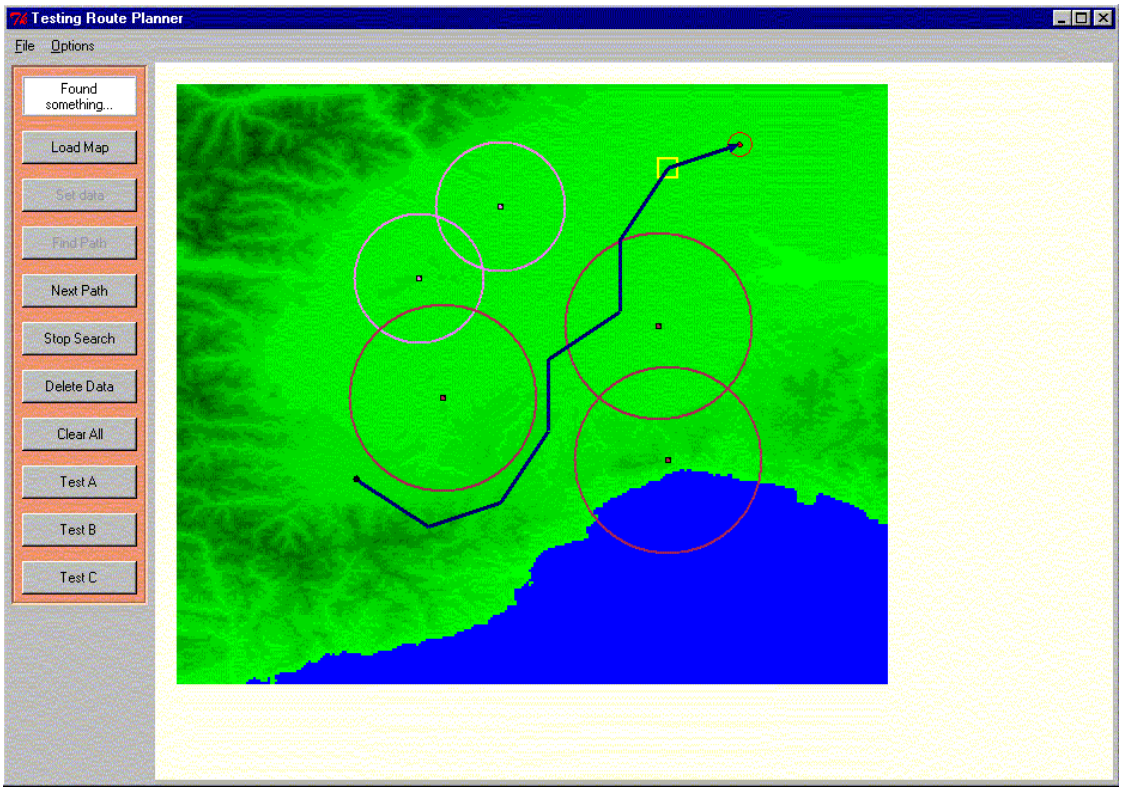

Figure 3 - Scenario with threats: a solution with a threat-cost.

(D) the fuel availability must to be respected.

Once a feasible solution is found, the search for an optimal plan starts. Two different optimal criteria might be adopted: minimized fuel consumption or minimized threat exposure. With the latter criteria, the planner is able to avoid regions where flight conditions are adverse (e.g., because of meteorological conditions). Figure 2 shows the shortest path is no obvious, when considering the constraints (B) and (C).

2.2. Digital terrain model

The DTED was adopted to model the search space. It is a geographic matrix of terrain elevation data points converted into a numerical format. The National Imagery and Mapping Agency (NIMA) manages the DTED files. The elevation points, in the form of a matrix are at precise increments of latitude and longitude. The elevation values are expressed in meters. The elaboration of the matrices that are stored in some text files draws the maps. There are three formats of DTED: DTED level 0, DTED level 1, and DTED level 2. The first one indicates the altitude in meters of a square region of 1 km x 1 km (30 arc second post spacing). The other formats have a higher precision.

The DTED level 0 format is adopted in this application. However, in order to be precise, the application is able to construct a new matrix. The new matrix is constructed by taking the maximal quote on sub-matrixes of $L \times L$ elements. For instance, if the original DTED matrix has $10 \cdot L \times 10 \cdot L$ elements, the new matrix will have a dimension of 10×10. As a consequence, if L has little values ($L=2$ or $L=3$), the path will be more precise, but more computation time will be necessary to find it.

The geographic coordinates are converted in matrix indices. Thus, the geographic coordinates are represented in the solution as couples of integers. Each area turns out to be a cell, and an aircraft is supposed to move over this grid-cell space.

2.3. Threat model

A second matrix, different from the previous one, specifies the threats. Each element of the matrix indicates a cost of an area of M km^2, where M is a fraction of L (e.g., if L=5 then M=1). This matrix is more precise because of the necessity of having an accurate description of threats. The values of corresponding elements of the cost matrix are updated, when a new threat appears in the actual scenario. Consequently, in order to specify a particular kind of threat, it is sufficient to define how it modifies the cost matrix.
Threats are described as general entities without considering their particular nature. They are modeled here as circular regions. Two types of threats are currently implemented. The first has the cost decreasing with the radius (type TA). The second has a constant cost in the whole region (type TB). Type TA threats are the smallest ones in Figure 3.
When calculating the total cost of a path, in order to determine which positions are flown over, a modified version of the Bresenham algorithm for tracing lines is applied [2].

2.4. Fuel Consumption model

The fuel consumption depends on several parameters of the air vehicle: the speed, the altitude, the drag index, and the weight. Lookup tables indexed with these parameters are used to compute the fuel consumption of the flight plan.

3. CCP-BASED APPROACH TO PATH PLANNING

The path planning is formalized as a constraint satisfaction problem and it is solved with CCP. An executable plan is a list of waypoints (Cartesian coordinates). The linear connection of these waypoints represents the flight path. The first waypoint in the list is the position of the aircraft given by the Global Positioning System when the planner is triggered. The last element of the list is the target position. The path planning is reduced to the search of a valid variable assignment to the list of coordinates in such a way that all constraints are satisfied. The elements of the list are couples of finite domain variables. Their domains depend on the dimension of the DTED map that expresses the search space. If the map is represented by a matrix of 50x50 grid cells, the couple of domains of each finite domain variables representing a waypoint will be [1#50]. The term root-variable is used below for to the list of waypoints representing the solution.
The contribution of this work is to model the constraints (A), (B), (C), and (D) directly on the root-variable, that is the list of waypoints. The advantage of having this model is the planning is not performed on an input graph as in all the related work. Reasoning on a graph is a drawback because only a limited set of positions is considered. Moreover, the graph as to be constructed before the search process starts, and thus, important positions might be eventually discarded during this phase. The model we propose searches on the geographical map, respecting all the constraints, and considering a much wider set of positions for the waypoints. The turn angle constraint is described in more detailed below, but for a complete formal definition of the model see [5].

Turn angle constraint. This constraint causes problems in the classical approaches, and it requires code modification in the implementation of the search algorithms. In the declarative style of CCP, this constraint does not introduce any trouble. It is enough to define a new propagator that implements the constraint, and to invoke the propagator in the main procedure of the CCP application.
A new propagator, called `turn_angle`, was defined and implemented. The purpose of this propagator is to perform domain filtering for the variable involved in the constraint (C). It has the following syntax:

```
{MyFD.turnAngle WP.n WP.(n+1) WP.(n+2) minAngle maxAngle}
```

Where WP_i are elements of the solution list; *minAngle* and *maxAngle* are the minimal and maximal possible turn angle α (in degree) between segment [WP_n; WP_{n+1}] and segment [WP_{n+1}; WP_{n+2}]. The `MyFD.turnAngle` is able to reduce the domain of each WP_i in such a way that the following inequality is satisfied:

$$minAngle \leq \alpha \leq maxAngle$$

This propagator is triggered when one of the three variables becomes a singleton.

Global Constraint. The `AllDifferent` global constraint was applied to the root-variable, in order to impose that all waypoints have to be different. Intuitively, if the flight path goes through the same waypoint more than once, the path cannot be minimal in the length.

Variable and value ordering. The distribution strategies selecting the variable to distribute at each search node and the value to assign to that variable are decisive. The following strategies were chosen for this application:

- *Variable ordering*: selects the variable with the smallest number of suspensions. This number indicates how many propagators are suspended on a variable waiting for a change in its domain. If several variables suspend on the minimal number of constraints, the leftmost variable whose domain is minimal is selected.
- *Value ordering*: the domain of the selected variable is divided in two intervals. The distribution algorithm tries to assign to the variable the first interval, and, if it fails, it tries the second one.

Anytime and incremental search. An on-board planner needs to be anytime and incremental. An algorithm is defined anytime if it is able to give a solution at any time during the search process; it is defined incremental, if it builds before a raw solution, and then it refines the quality of the solution trough incremental search steps. The more time it has, the better will be the solution.
The standard depth-first search strategy is used in this application. After a feasible solution is found, the search continues in a branch and bound fashion, adding the constraint that the new solution must have a lower cost than the previous one. When searching with branch and bound, the most recent found solution is called the incoming solution. Consequently, the search process is incremental, as all the branch and bound algorithms, and anytime, once the feasible solution is found.
In the current implementation, the search process terminates when a timeout is reached. The best solution found in a limited amount of time, is intended, in the following, as the optimal path. The time limit is not fixed a priori, and it is a parameter of the path planner. When the Flight Management System calls the planner, it might decide how much time the search could last. When the time is over, the planner returns the last incoming solution. In general, the further the modifications in the input data occur from the actual position of the UAV, the more time the search engine will have; the more time the planner has, the less the path should cost. In the best case, the search finishes before the time limit and the optimal solution costs nothing. Otherwise, if no solution is found, a standard flight recovery maneuver must be executed.

Implementation. The Mozart programming system, based on the Oz language, is employed for the software implementation. The Oz language supports declarative programming, object-oriented programming, constraint programming, and concurrency as part of a coherent whole. The Constraint Programming Interface (CPI) was used to define the `turn_angle` propagator directly in Ansi C++. Mozart was chosen because it has performance competitive with commercial products [11], despite of being freeware and open-source.

Several books have been written about Mozart/Oz ([20], [17], [3], and [6]), and the programming system is available for free at [11].

4. RESULTS

The planner was tested with different DTED level 0 files, changing while the resolution of grid-search space and using different scenarios. Test results presented in this section refer to a map of the Northwest Italian peninsula. The corresponding DTED file was described by a matrix of 297x257 integers that refers to square-cells of 1 km x 1 km. The altitude matrix was defined with L=10, and the threat matrix with M=2. Consequently, the grid space was a 29x25 matrix with cells of 10 km x 10 km, and threats were defined on matrix of 148x128 integers that give threat-cost with a precision of 2 km x 2 km.
Figure 2 shows a typical output of the planner that consists of a route that satisfies all the defined constraints. The constraints are:

- *departure point*: blue point on the left;
- *goal*: red circle on the right;
- *approach position*: yellow square;
- the minimal leg length is of 10 km. A maximal leg length of 40 km is defined too;
- the minimal turn angle is of 30° and maximal turn angle is of 60° (i.e., $30° \leq \alpha \leq 60°$);
- the amount of available fuel is 2500 Kg.

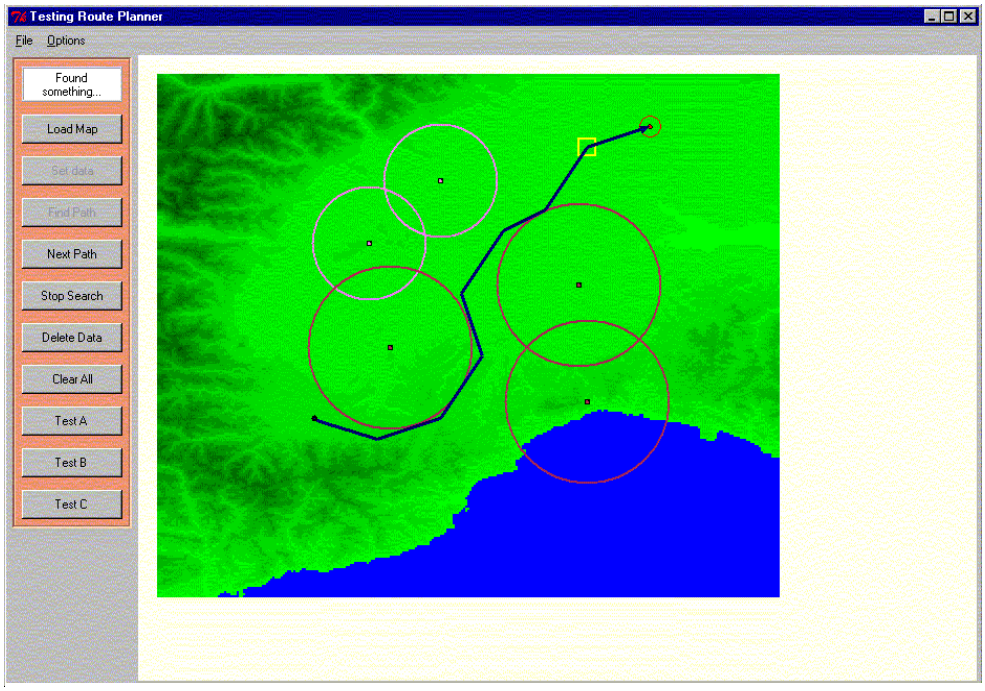

Figure 4 – A Scenario with threats: a solution with the threat-cost equal to zero.

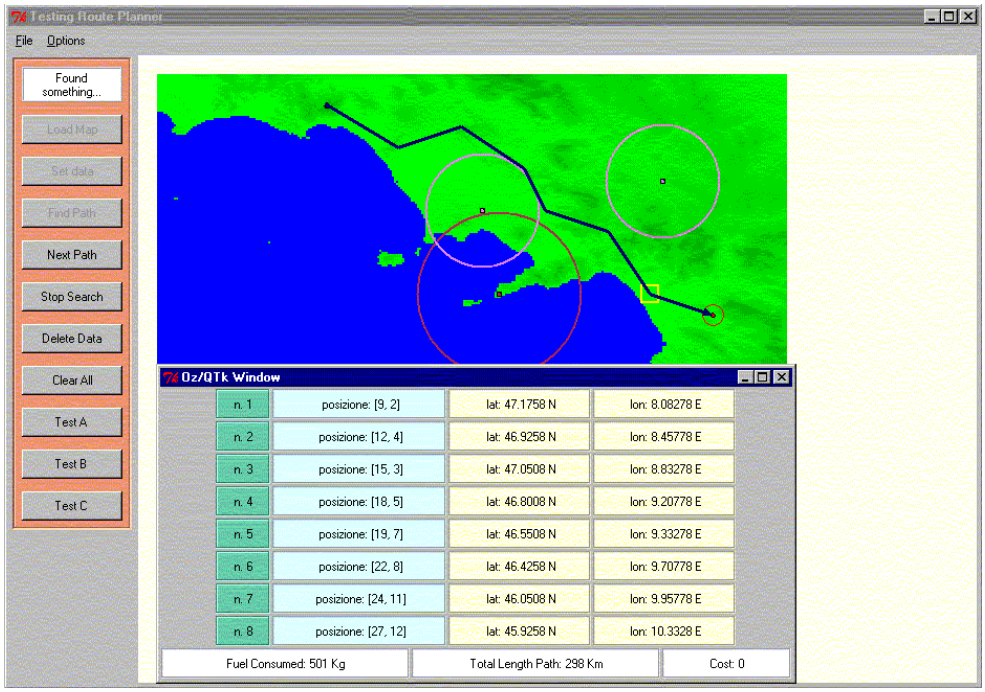

Figure 5 – The list of the coordinates identifying the waypoints.

These parameters were set by using a graphical interface, and were defined mainly in order to show that the shortest path with these types of constraints is not trivial. The planner produced this output in 0,2 sec on a Pentium III workstation. Figure 3 shows the same map, but with five threats: two of type TA and three of type TB (the type are defined in the previous paragraphs). The constraints are set as in the previous example. The flight path is the feasible solution found by

the planner, and it is computed in 2,3 sec. Then, Figure 4 shows the optimal solution (achieved in 4,3 sec). When evaluating this path, the adopted resolution (L=10 and M=2) and the parameters of constraints must be taken into account. Changing these values results in a different optimal path. It is always a trade-off between search time and optimality.

A proper configuration of the constraint parameters is believed to be dependent on both mission characteristics and path features to be favored (for instance cost, fuel consumption, time).

At the end, in Figure 5, the Naples's gulf map with three threats is shown. The flight path is described both on the map and in a textual way. It is possible to see that the solution is a list of coordinates identifying a list of waypoints that the UAV has to flight over in order to reach the goal.

5. CONCLUSIONS

The application that is presented in this paper is considered to have satisfactory performance particularly when considering the limited effort spent in the development. It is possible to state that the main advantage of approaching path planning by CCP is the possibility of modeling complex systems with minimal algorithmic effort. This application was entirely developed in a six man-months project. A similar planning problem was faced some years ago using a classic space-state graph approach, working on a dedicated Lisp machine, but the required effort was very high. Using CCP, the focus of the software designer is really shifted from the algorithm definition to the modeling of the problem as a constraint satisfaction problem, with variables, domains, and constraints. Another advantage of approaching planning by CCP is that once a basic model is designed, to develop software prototypes is easy and fast. Thus, the software development process may be carried out by iterative refinements of the first model.

The study presented here has shown interesting possibilities for further development. Essentially, two directions may be followed, one that aims at improving run-time performance, and another at elaborating more complex models. The first issue should be addressed by integrating classical operations research techniques in the CCP framework, as discussed in detailed by J.N.Hooker [7]. The second one may include possibilities of considering multiple targets, to address dynamic scenarios, and to plan routes for a constellation of air vehicles.

Acknowledgements

A special thanks to Luis Quesada for sharing his experience through daily discussion, and to Kevin Glynn and Valentin Mesaros for reviewing this paper.

References

1. Allo, B.; Guettier, C. ; Legendre, V.; Poncet, J.C. ; Strady-Lecubin, N. 2002. *Constraint Model-Based Planning and Scheduling with Multiple Resources and Complex Collaboration Schema*. AIPS 2002, Toulouses, France.
2. Bresenham, 1962. *Bresenham-based supercover line algorithm*. http://www.ese-metz.fr/~dedu/projects/bresenham
3. Duchier, D.; Gardent, C.; Niehren,J. Language. *Concurrent Constraint Programming in Oz for Natural Processing*, 1998. Programming Systems Lab, Universität des Saarlandes, Germany.
4. Duchier, D.; Kornstaedt, L.; Homik, M.; Müller, T.; Schulte C.; Van Roy, P. 2002. *Finite Domain Constraint*, in System Modules – Constraint Programming, Mozart documentation.
5. Gualandi, S. 2002. *Studio e applicabilitá del Constraint Programming a problematiche di Route Planning*. Graduation Thesis. Dept. of Computer Science Engineering, University of Pavia.
6. Henz, M. *Objects for Concurrent Constraint Programming*. International Series in Engineering and Computer Science,1997. Kluwer Academic Publishers, Boston, MA, USA.
7. Hooker, J.N. eds. 2000. *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*. John Wiley & Sons.
8. Koenig, S.; Likhachev, M.; Furcy, D. *Lifelong Planning A**. Technical Report, GIT-COGSCI-2002/2, College of Computing, Georgia Institute of Technology, Atlanta (Georgia), 2001. (Accepted for Publication in Artificial Intelligence).
9. Mata C.S., Mitchell J.S.B. 1997. *A new algorithm for computing shortest paths in weighted planar subdivisions*. Proceedings of the thirteenth annual symposium on Computational geometry: 264-273. Nice, France.

10. Mitchell J.S.B. 1988. *An algorithmic approach to some problems in terrain navigation*. Artificial Intelligence 37: 171-201.
11. Mozart Consortium. 2003. *The Mozart Programming System Version 1.2.5*. Available at http://www.mozart-oz.org/
12. Muller, T.; Wurtz, J. 1999. *Embedding propagators in a Concurrent Constraint Language*. The Journal of functional and logic programming (Special issue 1). MIT press.
13. National Imagery and Mapping Agency (NIMA): http://164.214.2.59/geospatial/products/DTED/dted.html
14. Quesada, L. 2004. *A context-dependant propagator for constrained path problems*. Technical Report RR 2004-03,INFO, Universite' Catholique de Louvain, Belgium.
15. Russell, S.J.; Norvig, P. eds. 1995. *Artificial Intelligence: a modern approach*. Englewood Cliffs, New Jersey: Prentice Hall.
16. Saraswat, V. A., and Rinard, M. 1990. *Concurrent constraint programming*. In Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, San Francisco, CA, 232-245.
17. Schulte, C. 2002. *Programming Constraint Services*, volume 2302 of Lecture Notes in Artificial Intelligence. Berlin, Germany: Springer-Verlag.
18. Szczerba, R.; Galkowski P.; Glickstein, I.; Ternullo, N. 2000. *Robust Algorithm for real-time Route Planning*. IEEE Transactions on Aerospace and Electronics Systems 36 (3): 869-878.
19. Van Blyenbyrgh, P. 1999. *UAVs – Current situation and considerations for the way forward*.
20. Van Roy, P., and Haridi, S. 2004. *Concepts, Techniques, and Models of Computer Programming*. The MIT press. Available March 2004.
21. Weatherington, D. 2002. *Unmanned aerial vehicles roadmap*. http://www.acq.osd.mil/usd/uav_roadmap.pdf