Learning Constraint-based Strategies for On-Board Dynamic Planning and Scheduling of Logistics Missions for Autonomous VTOL-UAV

No Author Given

No Institute Given

Abstract. This paper proposes a constraint programming approach for activities optimization of a specific type of unmanned aerial vehicle (UAV), able to perform vertical-take-off-and-landing (VTOL). The resource constrained optimization approach can be effective in flight, supporting the operator in all mission phases of future logistics scenarios. We propose a constraint programming model and a learning-based search algorithm approach for path-planning & scheduling and re-planning, using representative topological maps of navigation waypoints and edges. Both model and optimization algorithms, together with a constraint solver, are integrated as part of a software agent architecture that commands the UAV. Actual flight simulation experiments of the software agent on realistic scenarios are reported, showing correct performances for an embedded usage, in particular regarding the online re-planning and learning of heuristic upper bounds on path length.

Keywords: Path-planning · scheduling · re-planning · learning heuristics · application · VTOL aircraft

1 Introduction

Unmanned Aerial Vehicles (UAV) are widely used for civilian logistic, search and rescue, first responder in disaster relief, security and other defense operations. A rotorcraft based UAV can easily perform stationary observations, can land and take-off easily but has limited endurance, cannot fly fast and its energy expenditure is hard to plan. In turn, resource constrained navigation is easier with a fixed wing, that can perform long range missions but needs to orbit around a waypoint to acquire and observe a target, and cannot land. Innovative types of UAV, that have Vertical Take Off and Landing (VTOL) capabilities provide interesting agility for navigation, can fly long range and can perform several "pick and drop" logistic actions in a same mission (see Fig. 1). Usage of such VTOL-UAV can be intense, having to satisfy several requests on the fly and must adapt to its environment (traffic, obstacles, landing areas). Autonomous flight is an attractive concept for this VTOL-UAV as it limits operators interactions, and save platform costs from life support equipment. With such concept, appropriate navigation plans are needed and several scheduling constraints have

to be managed by both the platform and the remote operator, resulting from environment activities and the operational demand. Two optimization problems can occur several times during a mission:

- The vehicle endurance can be saved by minimizing energetic resource consumption while satisfying mission duration constraints.
- In case of urgent missions, the operator will try to minimize the mission duration while satisfying energetic constraints.

For VTOL-UAV, a consumption model is difficult to define and when combined with scheduling constraints, the planning problem becomes hard to solve. Simple approaches, such as finding a shortest path or using A*-like algorithms, may not lead to efficient solutions and cannot meet scheduling constraints. The core problem is also close to a Traveling Salesman Problem (TSP), for which many algorithms are efficient but cannot be directly used with complex resource constraints.

This paper proposes a hybrid constraint solving approach for VTOL-UAV activity optimization, that can be used on-board, and support the operator in all mission phases. The above planning and scheduling problem is formulated as a dedicated Constraint Satisfaction Problem (CSP), and also as a Constraint Optimization Problem (COP), developed with the *GeCode* [12] environment.

The hybridisation schema uses cost probing by learning and is two folds. Firstly, the probe algorithm learns to solve a relaxed form of the problem and estimates a heuristic upper bound to the cost function. Secondly, another probe algorithm provides an order on problem variables (aka labeling).

In both cases, the principle is to use the solution of a low-computational relaxed problem (aka probing [**?**]) to improve the current solving of the problem, as well as the performances of path planning & scheduling. A variant of the latter is used for re-planning, which is in fact is a specific case of online planning. It reuses a past solution to a close problem in order to structure the current solving process.

Such CSP/COP model is also used as a deliberation capability as part of the software architecture of the autonomous VTOL-UAV agent. It is synchronised with other capabilities (e.g., simultaneous localisation and mapping, perception) that are also integrated in this software agent used in simulation.

Experimental results on the performances of path planning & scheduling are presented, demonstrating the interest in using this re-planning scheme and in using machine learning to compute an upper bound on the cost function of a COP. These performances suit the operational needs, enabling to use CP path-planning & scheduling as a component of the software architecture of the VTOL-UAV agent. Therefore realistic flight scenarios of the VTOL-UAV agent are attainable and presented.

The paper is organized as follows: The next section presents the planning & scheduling problem formulated as a constraint-based model. Section § 3 describes the associated heuristics used for search and then section § 4 presents a constraint-based software architecture. Section § 5 provides experimental results

on realistic benchmarks. Finally, section § 6 relates our work to the state of the art, and section § 7 sums up our contributions.

Fig. 1. An example of a VTOL-UAV type fixed-wing unmanned aerial vehicle.

2 Navigation Model for VTOL-UAV

We describe a constraint-based model for solving and optimizing flight plans of the VTOL-UAV using a composite planning and scheduling model. A solution for this model is a discrete path with arrival times on waypoints, and flight durations between two successive waypoints. The model is composed of a boolean path-planning sub-model and a temporal scheduling sub-model, which are interrelated. Note that we do not consider on board storage as a resource constraints for this problem.

2.1 Boolean Model for Path-Planning

A topological map is a directed graph (V, E) where vertices $v \in V$ are locations and edges $e \in E$ are elementary paths — edge (v, v') is an elementary path from location v to location v'. Variable $\Phi_v \in \{0, 1\}$ is equal to 1 iff vertex $v \in V$ is in the path (v is a waypoint), and 0 otherwise. As for vertices, variable $\Phi_{(v,v')} \in \{0, 1\}$ is equal to 1 iff edge $(v, v') \in E$ is in the path, and 0 otherwise. Vertices $start \in V$ (resp., $end \in V$) are the starting (resp., ending) waypoints. Set $\omega^+(v) \subset V$ (resp., $\omega^-(v) \subset V$) represents the outgoing (resp., incoming) edges for a given vertex $v \in V$.

$$\sum_{u \in \omega^+(v)} \Phi_u = \sum_{u \in \omega^-(v)} \Phi_u = F_v \leq N \tag{1}$$

$$\sum_{u \in \omega^-(start)} \Phi_u = N \quad and \quad \sum_{v \in \omega^+(end)} \Phi_u = N \tag{2}$$

Since flow variables Φ are in $\{0,1\}$, equation (1) ensures conservation of the flow F_v on each vertex v, hence path connectivity and uniqueness. N is the number of agents which can pass through a vertex (flow capacity), hence is set to 1 in our case — [4] relaxes this hypothesis and builds synchronization constraints between two agents.

Equation (2) imposes limit conditions for starting and ending the path. These two constraints provide a linear chain alternating pass-by waypoints in the topological map.

$$\Phi_v = 1 \tag{3}$$
$$\Phi_v = 0 \tag{4}$$

A waypoint v can be forced to be included into (resp., excluded from) the path, with equation (3) (resp., equation (4)). *start* and *end* vertices follow equation (3). When a specific pick / drop action has to be executed on a given waypoint, equation (3) is also enforced.

The path over vertices $P = \{v \in V \ / \ \Phi_v = 1\}$ is the set of waypoints the agent passes through, including forced vertices *start* and *end*. As for vertices, the path over edges is noted $P' = \{(v,v') \in E \ / \ \Phi_{(v,v')} = 1\}$.

2.2 Temporal Model for Scheduling

In order to provide sequential temporal information, we introduce several temporal variables, defined as integer and binded to the previous model.

Variable T_v is the time at which an agent arrives at vertex v, variable A_v is the elapsed time during which it performs its action at vertex v. Variable $D_{(v,v')}$ is the duration of the agent for traversing edge $(v,v') \in E$ and variable $S_{(v,v')}$ is the average speed of the agent over edge $(v,v') \in E$.

$$T_v + A_v + D_{(v,v')} \leq T_{v'} \quad \Leftrightarrow \quad \Phi_{(v,v')} \tag{5}$$
$$T_{start} = 0 \tag{6}$$
$$||(v,v')|| = S_{(v,v')} D_{(v,v')} + R \ \ and \ \ R < D_{(v,v')} \tag{7}$$

Equation (5) ensures propagation of the arrival time T_v on vertices over edges, starting at time 0 (equation (6)). Since equation (5) holds for waypoints only and not for any pair of vertices (v,v'), this constraint is reified over variable $\Phi_{(v,v')}$ — it holds for traversed edges only. Equation (5) links the present temporal model to the previous boolean one. Equation (7) ensures that the average speed $S_{(v,v')}$ of the agent on edge $(v,v') \in E$ is the integer division of the constant length $||(v,v')||$ of this edge, by its traversal duration $D_{(v,v')}$. Variable R is the remainder in this modulo operation. This sub-model is particularly interesting to represent take off and landing actions, including picking and dropping objects or

personals. Also note that the problem also consists in finding an adequate speed that meets all constraints an may contribute to the cost optimization.

$$T_v \leq T \tag{8}$$
$$T_v \geq T \tag{9}$$

An upper (resp., lower) bound T on the passing time T_v at waypoint v can be set by equation (8) (resp., equation (9)).

2.3 Constraints Specific to VTOL-UAV Missions

Inclusion / Exclusion Spheres The application covers VTOL-UAV in logistics missions close to a no-fly zone. In a civilian context, this can be due to other traffics, while in disaster relief, defense or security, some danger could occur. Safe spheres (e.g., for refueling or emergency landing) and danger spheres (where hazards are present) are defined. Let S_i be a list of safe spheres and S_e be a list of danger spheres.

$$\forall v \in P, \exists s \in S_i \ / \ v \in s \tag{10}$$
$$\forall v \in P, \forall s \in S_e, v \notin s \tag{11}$$

Equation (10) ensures that there is always one safe zone in which the aircraft can land during path traversal (inclusion spheres). Since the 3D coordinates of the vertices are known (vertices of a topological map are locations), as well as the sphere center and radius, it is sufficient to remove from the graph (see equation (4)) vertices with no inclusion sphere. Equation (11) ensures that all danger zones (exclusion spheres) are avoided during path traversal. As above, it is sufficient to remove from the graph (see equation (4)) vertices inside any exclusion sphere. We note that spheres of S_i and S_e may include only a portion of an edge between two vertices — this is left for future refinements of the geometric model, resulting in additional equations $\Phi_{(v,v')} = 0$ for removing edges, similar to equation (4) for removing vertices.

In practice, these spheres remove a significant part of way-points to fly-by, lowering problem sizes, which is reflected in our benchmarks. Further models, more complex, may condition sphere inclusion or exclusion by VTOL-UAV speed limits or/and fly-by dates.

Temporal Windows A temporal window can be represented in the model above by using a conjunction of equations (8) and (9). That is, if x is a temporal variable (e.g., T_v in previous section) and $W = [min; max]$ is a temporal interval: (i) variable x is included in W iff $min \leq x \wedge x \leq max$; And (ii) variable x is excluded from W iff $x < min \vee max < x$.

Inclusion windows simply reduces a variable domain x with equations (8) and (9). However, exclusion windows involve an OR statement, i.e., a discrete disjunction $\vee$ over same equations, leading to different and more radical subsequent searches starting from variable x in the solver control algorithm.

2.4 Optimizing VTOL-UAV Missions

Our application supports both CSP instances as well as COP. We define a cost function f which value is computed on a solution s of the CSP model. The cost leads to further backtracking and search with the additional constraint $f < f(s)$ dynamically posted when encountering a solution s. Therefore, non-optimal solutions of decreasing cost are evaluated, until the optimal solution is found (the CSP solver proves that no other better solution exists in the search tree).

The possible cost functions f are:

None No cost function: Constraint optimization with no cost function is constraint satisfaction.

Waypoints $f = ||P||$: Constraint optimization minimizes the number of vertices in the path (shortest path in terms of waypoints). This is interesting to avoid an interaction with the operator whenever a change in direction is needed over a waypoint

Makespan $f = max_{v \in P}(T_v)$: Constraint optimization minimizes the latest arrival time on a vertex, i.e., f is the makespan of the path P (shortest path in terms of time).

Energy $f = \sum_{(v,v') \in P'} Power(v, v') D_{(v,v')}$: Constraint optimization minimizes the total energy consumed along the path (shortest path in terms of energy). To define the variable $Power$ over an edge (v, v') of the path, a simplified power consumption model is defined: The idea is to have a set of equations which takes into account in a qualitative manner the power consumption of the UAV as a function of the path which the aircraft follows. Such model can be further calibrated with real engine consumption. The variable $Power(v, v')$ over an edge (v, v') is modeled using the following types of consumption laws:

- When the VTOL-UAV is flying at constant altitude, the power consumption varies linearly as a function of its speed.
- When the VTOL-UAV is climbing, the power consumption varies (i) as a quadratic function of the angle of attack and (ii) as a linear function of its speed.
- When the vehicle is descending, the power consumption (i) is inversely proportional to the norm of angle of attack and (ii) it varies as a linear function of its speed.

Length $f = \sum_{(v,v') \in P} ||(v, v')||$: Constraint optimization minimizes the total length of the path P (shortest path in terms of length).

The cost functions on energy and path length can be dual, and a cost valuation for one type can set an upper bound over the quantity used by the other

on.e — This corresponds in fact to a low-class dominance in multi-criteria optimization.

2.5 Learning Heuristic Upper Bounds on Path Length

The cost function can be related to the length of the path, which we refer to as path cost. In such cases, introducing a heuristic upper bound on the path cost can dramatically increase search performances. The more the upper bound value is close the the actual optimal path cost, the more sub-optimal areas of the search space with higher path costs can be cut. We propose a heuristic to provide an upper bound based on learning methods and observe that search performance gets significantly improved upon with early branch cuts in the search tree (see section "Experiments"). Those cuts result from an additional constraint $f < B$, where f is the cost function and B the heuristic upper bound. We describe a machine learning approach used to build B based on iterative next-best waypoint estimation in a list of mandatory waypoints which require visiting.

Graph convolutional networks (GCNs) are generalizations of convolutional neural networks (CNNs), a popular type of neural network for image classification, using non-Euclidean graphs [1]. The input of a GCN is a graph $G = (V, E)$. Unlike CNNs, their receptive field is not based on a pixel sliding architecture, but rather on the graph structure. Edges E define the neighborhood of each node $v \in V$. This enables invariant nodes permutations, which is a strong advantage when dealing with graph-structured data. A common point with CNNs is that filtering parameters are shared over all locations in the graph, justifying the *convolutional* denomination. Since path-planning is performed in a topological map, i.e., a graph, we use GCN as a general representation to estimate path cost.

To use GCNs on a topological map, we consider the following relaxation of the original problem: $X = (v, v', M)$. Here, X refers to a problem where $v \in V$ is the start vertex (considered as the first GCN input feature), $v' \in V$ is the end vertex (second GCN input feature), and $M = (v_1, v_2, ..., v_q)$ is a list of q mandatory waypoints (third GCN input feature), i.e., waypoints forced to be visited (see equation (4) in the previous model). The input of the GCN function is the group of vertex features X and the topological map. The output is a probability distribution Π over all vertices in V: $GCN(X, (V, E)) = \Pi$. The GCN is trained so that Π assigns to each vertex in the topological map the probability of being the next best mandatory waypoint to visit, "best" in the sense of reducing the total length of the path over vertices. The training strategy for the GCN is self-supervised, i.e., training relaxed problems are randomly generated, automatically solved with a A*-based algorithm presented in [11] and each problem is labelled with the solution found.

The iterative algorithm for computing B using the trained GCN is pictured as follow:

1. The shortest paths between all pairs of vertices of V is computed off line with Dijkstra's algorithm and stored in matrix SP, i.e., $sp_{i,j} \in SP$ is the shortest path to travel from vertex v_i to vertex v_j.

2. GCN is applied to the initial path-planning problem $X_0 = (start, end, M)$ by creating the group of vertex features: $GCN(X_0, (V, E)) = \Pi_0$.
3. Let $v^0_{best} \in M$ be the mandatory waypoint with the highest probability in Π_0. This vertex is recommended by the GCN as the best next waypoint from the *start* vertex.
4. Let X_1 be $(v^0_{best}, end, M \setminus \{v^0_{best}\})$. That is, X_1 is the same path-planning problem as X_0, but (i) starting at the previously visited vertex v^0_{best}, which is now the new start vertex; And (ii) with the set of mandatory waypoints minus this new start vertex v^0_{best}.
5. GCN is applied to X_1, resulting in a new vertex $v^1_{best} \in M$ as the best mandatory waypoint, i.e., the waypoint with the highest probability in Π_1.
6. Repeat from step 4 to build $X_2, ..., X_i, ..., X_{q-1}$ and $v^2_{best}, ..., v^i_{best}, ..., v^{q-1}_{best}$ until no mandatory waypoint in M is left to visit.

The above algorithm results in sorting M as $v^0_{best}, v^1_{best}, ..., v^{q-1}_{best}$ as waypoints to travel from *start* to *end*. The upper bound B is then defined by the sum of the shortest paths found in SP according to this sort, i.e., the length of the path if we visit each waypoint in the order suggested by the GCN. The list of mandatory waypoints M is fed with the vertices of the topological map which are forced to be included (equation (3)).

3 Heuristics

Performances for solving a realistic CSP model highly depends on heuristics, either over variables (which variable to consider next?) or over values (which value of the chosen variable to consider next?). We now present these two kinds of heuristics.

3.1 Heuristics on Variables

The different variables of the model above are considered in the following static order: Φ_v, $\Phi_{(v,v')}$, $S_{(v,v')}$, $D_{(v,v')}$, $T_{(v,v')}$ and then T_v. The planning (boolean) model is solved first before the scheduling (integer) model is, since this order favors finding waypoints before assigning time to arrival at a node — it is also possible that a node is rejected from the path because it occurs too late or too early (a scheduling reason of rejection of a node from the path), but this happens more scarcely on average.

Each group of variables is considered using the labeling heuristics, i.e., along an arbitrary static sort, except for variables Φ_v, the most important variables since they define a path, which are sorted specifically: Variables Φ_v are sorted by (i) computing a constraint-free path from *start* to *end* using least-cost algorithm A* [5]; and (ii) computing a distance from any node (not only those inside the path, i.e., waypoints) to the path just found.

For this, firstly, a state/node of algorithm A* is identified to a node in the above model, i.e., this algorithm plans for locations without considering any

other potential constraint, e.g., forced included/excluded nodes (see equations (3) and (4)) or deadlines (see equations (8) and (9)). The cost function of a node is defined by $f = g + h$, where (i) g is the length of the path from *start* to the current node, i.e., the sum of the lengths of the successive edges $||(v, v')||$ involved from *start* to the current node; and (ii) h is the fly-by distance from the current node to the *end* node — A* nodes are identified to locations in a topological map and thus own 3D coordinates.

Secondly, the distance from any node v of a topological map to this path is defined by the number of edges required to travel from v to the closest node inside the above path — hence nodes on the path are at distance 0, nodes at one edge far are at distance 1, etc. This is computed by (i) initializing the distance inside the just found path to 0; And by (ii) propagating a label (i.e., the distance from the current node to the path) from the nodes of the previous path (distance 0) to the neighbors of their nodes (distance 1), and then in turn to the neighbors of these nodes (distance 2), etc, until no node remains unlabelled. Since there might be several ways to label a node (i.e., a node might be reached by following several propagation routes from the path), the label on a node is changed iff it is greater than the current propagated label (the distance is the smallest number of edges to reach the path).

These two steps result in assigning a distance d to each node v of a topological map, which is used as a static sort for the labeling heuristics on variables Φ_v. In addition to this sort, when $d = 0$ on nodes (equal rating of nodes in the sort), variables Φ_v are secondarily sorted according to the path order, from *start* to *end*, found by the previous algorithm A* — actually, variables Φ_v are sorted in reverse order, for potentially backtracking from the end of the sorted list *start* back to the beginning of this list *end*.

Since these heuristics do not change at solving time, they do not lead to processing overhead during search — as opposed to dynamic search strategies [10].

3.2 Heuristics on Values

Heuristics on values is performed by (i) searching each temporal variable T_v of the scheduling model by increasing values from 0 (i.e., T_{start}, according to equation (6)) up to a fixed time horizon (an arbitrary large constant integer) — hence searching for small values of T_v first; And by (ii) searching each boolean variable Φ_v of the planning model by first rejecting the node from the path (value 0) and then accepting it (value 1).

4 Planning and Execution

Things do not always unfold as planned for, due to environmental changes — perceived at execution time and not planned for at planning time. We now describe an on-line planning scheme, due to dynamic environments.

4.1 Software Architecture of the Autonomous Agent

On-line planning is the activity of planning the future course of actions of an autonomous agent, while it is actually executing it. Approaches for dealing with this contradiction usually involve cognitive architectures (see a survey in [6]) which set layers (e.g., deliberation, reaction) inside an autonomous agent.

On-line *path*-planning, and not action-planning, plans the future locations by which an autonomous agent should pass, while the agent is actually moving from one location to another. Path planning does not represent the logics of unfolding of actions along a path — a restricted version of action planning, i.e., action planning can represent the location where each action occurs, as a by-product of reasoning over action description models.

Now, in our case the main activities of the agent are (i) reasoning to perform path planning and scheduling (see previous sections), (ii) performing physical actions on vertices/locations and (iii) performing physical actions on edges/elementary-paths. The latter involves an action "flying" on edges, with only its duration $D_{(v,v')}$ on an edge/path (v, v') represented in the model (see previous section). The previous second point ii also involves the action's duration A_v on vertex/location v, which is the only represented characterization of the action performed on v. This can be for example temporary landing, picking or droping an object. Now, the path-planning & scheduling reasoning activity might take a very small amount of time in real-time to perform and finish, or might take a gigantic amount of time to find paths over long-distance start/end vertices with boolean and temporal constraints — a common phenomenon, the general problem with deliberative activities and the *raison d'être* of software agent architectures (and even cognitive architectures) in general. Agent activities, potentially yielding to re-planning events, are:

1. Localization and mapping (aka SLAM): Given sensor data, this consists in both locating the agent in a metric map and locating perceived obstacles on this map. Both need to be performed together, since if the agent can locate itself on a metric map (localization), then it can locate perceived obstacles on the metric map relatively to its own localisation.
2. Perception: Extracting symbols from raw sensory data, e.g., recognizing other aircraft from raw sensor data.
3. Data fusion: Augmenting the quality of perceived information by using raw data coming from several sensors instead of one sensor only.
4. Path planning: Computing a path from a given vertex/location to another one, given boolean and temporal constraints (see previous sections).
5. Control: Ground control can also provide mission and no-fly zone updates. Turning high-level commands (e.g. the next location to reach) into a series of low-level voltage commands to the actuators of the autonomous agent.

A more detailed description of those algorithms and components (except for path planning) is out of the scope of the paper. This above chain of computation, usual in the Intelligent Transportation Systems community, illustrate the data flow inside the architecture of the autonomous agent.

4.2 Path Re-Planning

Path re-planning is performed as a special case of online path-planning.

First, a path from *start* to *end* is planned for with the model of previous section. Second, execution starts, i.e., the agent moves from vertex/location to vertex/location as prescribed by the previously computed path — so far, this is the planning-then-execution paradigm. But now an unexpected event occurs, which makes the rest of the current path infeasible. The re-planning process is threefold:

1. The current path is stored;
2. Path-planning is launched again, with (2.a) the last encountered vertex/location as the new start vertex, the end vertex unchanged, as are the other boolean and temporal constraints; and with (2.b) the path in step 1 (an ordered list of vertices $\{v_i\}$) is used as static sort on variables $\{\Phi_{v_i}\}$ in the heuristics on variables, instead of using the A* algorithm (see section 3);
3. The newly computed path is executed by the agent (i.e., the agent moves from one vertex/location to the next one), starting at its current location right after the previously passed vertex/location.

In other words, re-planning is considered as using planning as a piece of advice (a heuristics) for the next planning problem.

5 Experimental Results

The model above has been implemented using the CP solver *GeCode* [12] on a rugged computer with a quad-core 64 bits x86 CPU and 32 Gb of RAM. This model with the CSP solver library has also been embedded as a software component in a graph of components (see § 4) managed by the data flow tool RTMaps [3].

For the purpose of this paper, fictitious but realistic maps have been used, supporting real life scenarios. As an example, a first earthquake scenario in *San Diego* features 81 waypoints distributed unevenly on three flight levels (i.e., ground level, 500m level and 1000m level) to match the terrain's structure and elevation. Those 81 waypoints are linked by 2628 edges. A birdview of the map is given in Fig. 2, while a 3D view showing the flight levels and the edges is given in Fig. 3. Two other scenarios have also been used, damage assessment during a flooding in the Troyes suburb (in France), and rescue for avalanches in the Alpes (in France and Italy).

Path Re-Planning In the above San Diego topological map, the vehicle flies from start to a finish point on the map with the aim of minimizing mission time under the constraint of a maximum energy consumption not to be exceeded. On top of that, three waypoints, in green in Fig. 4, are mandatory, but the planner is free to decide in which order it will visit them. During initial planning, all

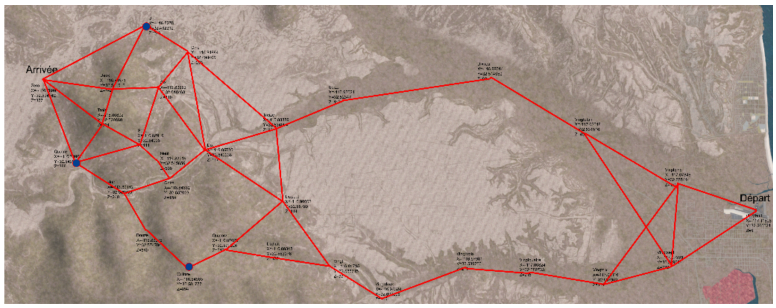

Fig. 2. Birdeye view of a map in San Diego region created as an example.

waypoints are accessible all the time. Before the start of the mission, initial path and schedule are formulated by the solving algorithm. On the *San Diego* map in fig. 4, such solution is materialized by the grey path. It fulfills mission goals, visits all three mandatory waypoints and sets the vehicle speed to the lowest value compatible with the mission's maximum time constraint, thus minimizing energy consumption (speed is proportional to the thickness of the grey line on the map).

At roughly one third into the original plan, that corresponds to the middle of the cruise flight, mission parameters change as follows :

- One part of the map, denoted by the red circle, is declared as a no-fly zone and it is now mandatory to plan around it.
- One of the mandatory waypoints, on the top left hand corner of the map is now accessible only during a short time window which happens to be different from the one used in the original plan.

The above event triggers re-planning of the mission and the new plan is shown in figure 5. As it can be seen, the planner has been able to take into account all new constraints to come up with an acceptable combination of waypoints and speed for fulfilling mission goals. The vehicle has to speed up in order to make it to the top left hand corner mandatory waypoint within the allowed time window, as shown by the thicker grey line. Once this waypoint has been visited, it can afford to slow down a little but has to maintain a higher speed than in the initial path for yet another edge. This is due to the fact that the new path is longer than the initial one and the vehicle thus has to be faster to still make

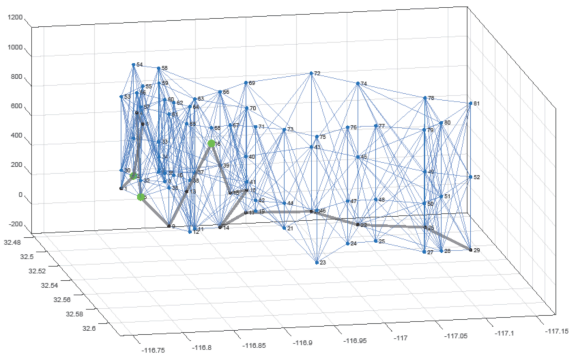

Fig. 3. 3D view of the *San Diego* map showing all nodes and edges.

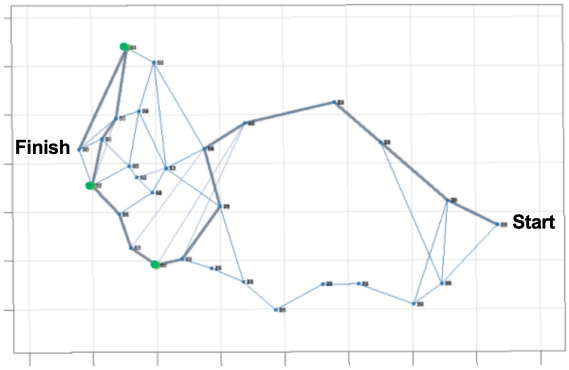

Fig. 4. Top view of the result of the initial mission plan for the San Diego area.

it in time to the finish line. Once the arrival on time is secured, the plan calls for the vehicle to slow down in order to save energy.

To measure the impact of re-planning vs. planning from scratch, several problem instances over several topological maps have been considered: First, a problem instance (v, v', m) is set in a specific topological map, where $v \in V$ is a start node, $v' \in V$ a destination node and $m \in V$ a mandatory waypoint to visit. Planning for this problem instance is performed and a solution path is found. A new problem instance is created by moving the start node v to another node further ahead on the solution path, and locally changing mandatory waypoint m. This new planning problem instance is then solved from scratch but also re-planned using the search data of the original problem, in both cases without the upper bound on path length. The mean time to the best solution is measured for both approaches in figure 6. This process is repeated for the three different problem instances on each topological map in order to measure a mean value.

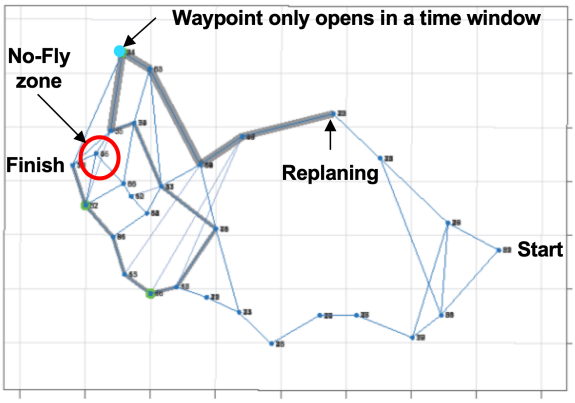

Fig. 5. Top view resulting of the re-planning process, taking into account time window and no-fly zone in the San Diego area.

Map (3 instances each)	number of nodes	Planning MT to BS (ms)	Re-planning MT to BS (ms)
San Diego	52	458	90
Troyes	48	94	44
Alpes	23	3928	923

Fig. 6. Mean time (MT) to the best solution (BS), for planning and re-planning (without upper bound on path length).

As a result, the re-planning mean time to the best solution is observed to vary between the half and the fifth of the planning-from-scratch one. (The San Diego topological map includes two levels, instead of three in the re-planning scenario of the previous section, to involve a number of nodes close to those of the other topological maps of figure 6).

Upper Bound on Path Length To measure the impact of the upper bound on path length, planning with and without upper bounds has been measured over approx. 20 problem instances over the 3 scenarios: San Diego earthquake, Troyes suburb flooding and Alpes avalanches. In each case for each region, the mean time to the first solution and to the best solution are measured (2nd et 4th column of figure 7). The mean cost (i.e., the path length) of the best solution is also provided (3rd column of figure 7). As a result,, for the map of San Diego, the best solution is found faster with upper bound than without in 100% of problem instances; Similarly, for the map of Troyes, the ratio is 83%; For the map of Alpes, the ratio is 72%. Lastly, the learning model we use for upper bound estimation is trained one hour on each graph before use. Training is done on data generated with a dropout *keep rate* of 0.9 to prevent overfitting and the *ADAM* optimizer [7] with an initial learning rate of 10^{-4}.

Map	without upper bound			with upper bound		
(20 instances)	MT to BS (ms)	MC to BS (m)	MT to FS (ms)	MT to BS (ms)	MC to BS (m)	MT to FS (ms)
San Diego	589	19928	86	485	19928	485
Troyes	88	45856	78	79	45856	79
Alpes	4415	25804	3193	2434	25804	2434

Fig. 7. Mean time (MT) in millisecond to the best solution (BS), mean cost (MC) in meter of this best solution, and mean time in millisecond to the first solution (FS), with and without the learned upper-bound on path length, measured on 20 instances for each 3 regions.

6 Related Work

The A* algorithm is an usual way to find a path of minimal cost from a start state to a potential goal state [5]. If this algorithm is useful to search for a solution in a graph of states, it is not applicable for several agents, each one searching for its solution state, and which potentially communicate (setting $N > 1$ in equation (1)). Also, it becomes difficult with this algorithm to incorporate mandatory states, except by introducing sub-solution states. Conversely, constraint programming [8] usually proposes languages or interfaces with better expressiveness (e.g., for mandatory / excluded waypoints, or mandatory / excluded edges, time windows).

[9] propose an anytime and dynamic version AD* of the algorithm A*, which can be interrupted (and return a solution the quality of which increases with the alloted time [2]) and which can cope with changing environments. If the notion of anytime path-planning in a CSP context surely is interesting, our approach regarding re-planning is more elaborated regarding largely changing environments.

[13] propose the notion of constraint network on timelines, to model discrete event dynamic systems and their properties, and also propose an application to aircraft having to perform actions on waypoints. Also defining a more elaborated model than ours regarding action representation, that approach proposes a beginning implementation whereas we quantify performances in our full implementation in a software architecture.

7 Conclusion

We have presented a CSP model for path-planning, scheduling and re-planning. This model with its CP solver library has been embedded as a component in the computation chain controlling an agent: our application involves VTOL-UAV in logistic missions close to war zones. This model has been experimented on realistic scenarios, and exhibits performances acceptable by an operational user. In addition, learning an upper bound on the cost function of this model leads to shorter performances, as intuition dictates, due to early cuts in the search tree.

References

1. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: going beyond euclidean data. IEEE Signal Processing Magazine **34**(4), 18–42 (2017)
2. Dean, T., Boddy, M.: An analysis of time-dependent planning. In: National Conference on Artificial Intelligence (AAAI) (1988)
3. DuLac, N.: Rtmaps project web site. Tech. rep. (2020), https://intempora.com/products/rtmaps.html
4. Guettier, C.: Solving planning and scheduling problems in network based operations. In: Proceedings of Constraint Programming (CP) (2007)
5. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems, Science and Cybernetics **4**(2), 100–107 (1968)
6. Ingrand, F., Ghallab, M.: Deliberation for autonomous robots: a survey. Artificial Intelligence **247**, 10–44 (2017)
7. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2015)
8. Laurière, J.L.: A language and a program for stating and solving combinatorial problems. Artificial Intelligence **10**, 29–127 (1978)
9. Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., Thrun, S.: Anytime dynamic a*: An anytime, replanning algorithm. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (2005)
10. Martinez, T., Fages, F., Soliman, S.: Search by constraint progapation. In: Proceedings of the 17th International Conference on Principles and Practice of Declarative Programming. pp. 173–183 (2015)
11. Osanlou, K., Bursuc, A., Guettier, C., Cazenave, T., Jacopin, E.: Optimal solving of constrained path-planning problems with graph convolutional networks and optimized tree search. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 3519–3525. IEEE (2019)
12. Schulte, C., Lagerkvist, M., Tack, G.: Gecode project web site. Tech. rep. (2019), https://www.gecode.org/
13. Verfaillie, G., Pralet, C., Lemaître, M.: How to model planning and scheduling problems using constraints networks in timelines. The Knowledge Engineering Review **25**, 319–336 (2010)