F. Teichteil-Koenigsbuch,
G. Poveda
(Airbus AI Research)

E-mail: florent.teichteil-koenigsbuch@airbus.com

DOI: 10.12762/2020.AL15-04

Collaborative Common Path Planning in Large Graphs

This paper studies two-agent path planning algorithms in graphs, where the two agents are assigned independent initial and goal states but are incentivized to share some parts of their travel glued together by scaling down the duet cost function when they move in formation. Applications range from ride sharing to formation flights. After presenting an optimal but unscalable algorithm, we propose a decoupled approach that separates spatial and temporal reasoning by first geometrically finding formation and breaking nodes in the graph then temporally synchronizing the agents on the formation node by adapting their speeds along their paths in the graph.
We also introduce an original heuristic function, which accounts for the potential formation paths in the graph and that is used to guide A* search on a cross-product graph representing the coordinated moves of the agents. We finally experiment our framework and compare its variants on grid-like and aircraft formation flight problems.

Introduction

Path planning has a long history of research dating back to the early days of Artificial Intelligence. Many variants have been studied, from continuous motion planning [2] to discrete optimization in graphs [5] including sampling approaches [16], investigating both single-agent [9] and multi-agent [23] settings. Multi-agent frameworks have mainly looked at optimizing trajectories for a set of agents while avoiding collisions [13][11], or at coordinating the trajectories of a set of agents to make them accomplish a common group objective; e.g., building formation patterns [7][4][1]. A specific case consists in incentivizing two or more agents to execute paths that share common moves; i.e., moves with the same current and next positions simultaneously, by reasoning about a cost function, which is lower when the agents move in formation side by side than when they follow different routes. In this paper we refer to this problem as "Collaborative Common Path Planning" (CCPP), depicted in Figure 1.

Applications of CCPP range from ride sharing planning, where two or more traveling people save money if they share the same vehicle along a common portion of their routes, to freight dispatch and routing where operational costs can be significantly reduced by transiting goods via intermediate common hubs, including comprehension of behaviors observed in nature, such as formation flights of migratory birds. Actual research works on CCPP, which have mainly originated

Figure 1 – Collaborative common paths example. Agent 1 (resp. 2)'s path duration is 1.7 (resp. 2.0). The agents move together in formation from t = 0.6 where they meet at the same navigation point until t = 1.4 where they break the formation. The global cost of their paths is 12.25.

from this latter example to the best of our knowledge [17][22][21], have recently studied how aircraft routes can be planned so that two given aircraft whose geodesic routes from their departure airports to their destination airports are spatially close can fly one behind the other along a common portion of their flight routes to save global fuel burn. Indeed, the lift of the follower aircraft can be increased if it flies in the aerodynamic vortex created by the leader aircraft wings, allowing it to reduce the thrust and thus reduce fuel burn. Some works also studied leader-follower path planning for mobile robots [4], but as CCPP for aircraft; all of these works assume the vehicles to be moving in continuous spaces using geometrical equations.

However, in reality, aircraft routes are rather defined over worldwide navigation graphs. The other aforementioned applications of CCPP also involve moving in discrete graphs rather than in continuous spaces. In this paper we study what we believe to be the first attempt to solve the CCPP problem in graphs, also paying special attention to making our approach scalable. We assume two agents to be moving in their own graphs – which can potentially be the same – in order to minimize the overall cost of reaching their goal vertices for each of them, while having the opportunity to significantly cut down atomic moving costs when they travel along specific pairs of (Agent 1's edge, Agent 2's edge) at the same time. Figure 1 represents the specific case where the graphs of each agent are identical. Importantly, our model assumes that the agents can control their speeds in order to synchronize at potential joining points from where they can move in formation, and that these speeds (or similarly, edge durations) can only be chosen from a discrete set – as is the case in aircraft formation flight, where edge cost evaluation is based on very time-consuming aerodynamic models that prevent a continuum of edge duration values from being explored. We present an optimal but non-scalable algorithm based on a transformation of the CCPP problem to a single agent path planning problem in the cross product of the agents' graphs, which can then be solved by heuristic search methods like A*, as well as an efficient but suboptimal spatio-temporal decoupling algorithm. We also discuss generic but non informative heuristics and propose efficient geometric heuristics specific to graphs defined on Euclidean spaces. We finally experiment with our framework and compare its variants on grid-like problems, as well as world-scale formation-flight experiments using real aircraft models and world flight networks.

Related work

There is an abundant literature on multi-agent path planning that may take a close look at the research that we investigate in this paper. The graph search community has investigated for many years multi-agent collision-free path planning [6]. Even though the objective of this research consists in minimizing the team cost-to-go summed over all of the agents, it sensibly differs from our work in the sense that avoiding collisions is the opposite objective to incentivizing the agents to meet and share common paths. Another line of research is aimed at planning dynamic formations for many moving agents [8], [19]. However, formations have a different meaning to ours: while we want the agents to move along the same graph edges to reduce their moving costs, like bird formation flights, those works dynamically assign predefined geometrical formation patterns to a group of agents that do not necessarily travel along the same edges. Ridesharing trip planning [18] is closer to our research, since the objective is to share vehicles among different passengers starting and ending at different locations. Since they share the same car, those passengers necessarily travel along the same edges on a portion of their trip. However, most works on ridesharing that we are aware of consider predefined meeting points, whereas we notably consider meeting points as part of the optimization problem itself. In ridesharing with passenger transfer optimization [3], the path from the passengers' starting location to the meeting point and from the breaking point to their ending location is additionally optimized, but the meeting and breaking points are still predefined and fixed. Along the same line of research, multi-modal path planning [15][12] look at optimizing the global flow of passengers between different sources and targets, while partly travelling by using common transportation means. However, multi-modal stations that serve as meeting and breaking points are also fixed, as for the meeting and breaking time tables. To the best of our knowledge, aircraft formation flight optimization in continuous airspace [22][17][21] is the closest problem to ours. Indeed, the specificity of the cost function for this problem, which reduces the sum of agent costs when they travel along the same edges, makes this line of research quite singular in the multi-agent landscape. We believe that our work is one of the first to investigate formation flight planning in discrete structured airspaces.

Background

There has been a long history of research on path planning in graphs. Many single-agent algorithms are based on the famous A* algorithm [9][10] which allows for lazy and partial exploration of the agent's navigation graph by guiding the search with heuristic estimates of the distance from the current node to the goal. Since we will use single-agent path planning as a generic tool to solve CCPP via a transformation to single-agent path planning, we only present here the plain A* algorithm. The interested reader is invited to look at [20] for state-of-the-art alternatives to A* for solving the single-agent transformation of our problem.

A* reasons about graph $\mathcal{G}=(V,E)$, which represents the feasible moves of the agent, with the edges being labelled by a cost function $c:E\to\mathbb{R}_+$. This iteratively expands the vertices of the graph from the starting vertex $v_s\in V$ up to reaching the goal vertex $v_g\in V$ via a minimum-cost path. Vertex expansion is guided by a heuristic function $h:V^2\to\mathbb{R}_+$ which gives numerical under-estimates of the distance from the current vertex to the goal. In order for A* to be optimal, the heuristic function needs to be both *admissible*, i.e., by noting as $C^*(v_1,v_2)$ the optimal path between any distant vertices $v_1\in V$ and $v_2\in V$, we have $h(v_1,v_2)\leqslant C^*(v_1,v_2)$, and *monotonic*, i.e., for any edge $e\in E$ and vertex $v\in V$ we have $h(e.\text{in}(),v)\leqslant c(e)+h(e.\text{out}(),v)$ by noting as $e.\text{in}()$ (resp. $e.\text{out}()$) the incoming (resp., outgoing) vertex of e.

CCPP problem formulation

Let $\mathcal{G}^{(i)}=(V^{(i)},E^{(i)})$ be the labeled graph of Agent $i\in\{1,2\}$, which represents its possible moves in its own navigation network. Both agents' navigation networks do not need to be identical, nor semantically equivalent. Graph vertices (resp., edge ends) stand for positions (resp., moves). Edges are equipped with label pairs (c,d) representing the cost and the duration of each move. Whereas standard single-agent path planning algorithms do not reason about moving duration, collaborative common path planning needs move duration information in order to temporally synchronize the two agents on vertices where they can initiate a formation. We add exponent notations

(i) to labels c and d, and edge flows *in* and *out* to indicate to which agent the concept is related.

We note as $\mathcal{F}^{\otimes} \subset E^{(1)} \times E^{(2)}$ the set of edge pairs for both agents on which formations are possible. Elements of $\mathcal{F}^{\otimes}$ must be temporally coherent, meaning that both agents' moves must have the same duration when in formation; i.e., $\forall \left(e^{(1)}, e^{(2)}\right) \in \mathcal{F}^{\otimes}, d^{(1)}\left(e^{(1)}\right) = d^{(2)}\left(e^{(2)}\right)$. In Figure 1, the dashed yellow lines are elements of $\mathcal{F}^{\otimes}$. Depending on the application, other properties might be required typically that the start and end positions of the agents always be the same in the case where their navigation graphs are identical (e.g., vehicles moving in a same route network, as in Figure 1). It is also required that the two agents visit candidate formation vertices at the same time; i.e., $\delta^{(1)}\left(e^{(1)}\right) = \delta^{(2)}\left(e^{(2)}\right)$ for $\left(e^{(1)}, e^{(2)}\right) \in \mathcal{F}^{\otimes}$, where $\delta(i) : E^{(i)} \rightarrow \mathbb{R}_{+}$ represents the time when the incoming vertex of an edge is visited by Agent i in its own graph (implicitly dependent on the previously visited vertices from the start up to the edge's incoming vertex).

Let $\Pi_{v_s^{(i)}}^{v_g^{(i)}}$ be the set of all possible *timed* sequences of Agenti i's adjacent edges, which represent all of its possible paths in its own graph from its start position to its end position, all such edges being labelled by the duration (and cost) of the move along the edge – which controls the agents' speeds so that they can synchronize their formation rendezvous point. The set of all possible collaborative paths is $\Pi_{v_s^{(1)}}^{v_g^{(1)}} \times \Pi_{v_s^{(2)}}^{v_g^{(2)}}$, where sections of each agent's path can correspond to common formation moves. Note that both agents' paths separately extracted from a given collaborative path do not need to have the same length, but they can contain common edges where the agents move in formation. The objective of the collaborative common formation path planning problem is to find a path for each agent with common formation sections where they can move side by side. The global cost of the pair of paths for the two agents is split into two parts: (1) cost of individual sections where no formation moves are possible, corresponding to the sum of each agent's individual moves; (2) cost of common sections where both agents are moving together in formation, corresponding to the formation cost $c^{\otimes}$. Thus, the cost function of a pair of collaborative formation paths $\left(\sigma^{(1)}, \sigma^{(2)}\right) \in \Pi_{v_s^{(1)}}^{v_g^{(1)}} \times \Pi_{v_s^{(2)}}^{v_g^{(2)}}$ is defined as $C\left(\sigma^{(1)}, \sigma^{(2)}\right) =$

$$\underbrace{\sum_{e^{(1)} \in \sigma^{(1)} \,:\, \nexists e^{(2)} \in \sigma^{(2)} : \left(e^{(1)}, e^{(2)}\right) \in \mathcal{F}^{\otimes}, \delta^{(1)}\left(e^{(1)}\right) = \delta^{(2)}\left(e^{(2)}\right)} c^{(1)}\left(e^{(1)}\right)}_{\text{agent (1)'s edges not in formation with agent (2)' edges (red in Figure 1)}} +$$

$$\underbrace{\sum_{e^{(2)} \in \sigma^{(2)} \,:\, \nexists e^{(1)} \in \sigma^{(1)} : \left(e^{(1)}, e^{(2)}\right) \in \mathcal{F}^{\otimes}, \delta^{(1)}\left(e^{(1)}\right) = \delta^{(2)}\left(e^{(2)}\right)} c^{(2)}\left(e^{(2)}\right)}_{\text{agent (2)'s edges not in formation with agent (1)' edges (green in Figure 1)}} +$$

$$\underbrace{\sum_{e^{(1)} \in \sigma^{(1)}, e^{(2)} \in \sigma^{(2)} \,:\, \left(e^{(1)}, e^{(2)}\right) \in \mathcal{F}^{\otimes}, \delta^{(1)}\left(e^{(1)}\right) = \delta^{(2)}\left(e^{(2)}\right)} c^{\otimes}\left(e^{(1)}, e^{(2)}\right)}_{\text{agent (1)'s edges and agent (2)' edges in formation (yellow in Figure 1)}}$$

The collaborative formation path planning problem consists in finding collaborative paths that minimize the global cost of the paths by taking account for common formation moves:

$$(\mathcal{P}) \quad min_{\left(\sigma^{(1)}, \sigma^{(2)}\right) \in \Pi_{v_s^{(1)}}^{v_g^{(1)}} \times \Pi_{v_s^{(2)}}^{v_g^{(2)}}} C\left(\sigma^{(1)}, \sigma^{(2)}\right)$$

Since graph edges are weighted by duration (and cost), the variables of the minimization problem above are not only vertex positions but also edge durations, which is required to synchronize the agents at their formation rendezvous point.

Algorithms

Problem $(\mathcal{P})$ consists in finding the agents' paths in their graphs and synchronizing them both temporally and spatially, which can be viewed as a single path finding problem in a cross-product graph that aggregates the positions of each agent. Figure 2 represents how agents' positions are aggregated and tracked together within a single cross-position vertex (depicted in yellow in the figure) that can lie at intermediate positions respective to each agent's graph. In fact, the figure shows that it is not sufficient to synchronize each agent's atomic move on the vertices of their respective graphs: in order to make a formation at $t = 0.6$, Agent 1 has to perform 1 atomic move whereas Agent 2 must execute 2 atomic moves. Therefore, we need to represent an intermediate position for Agent 1 between its first and second vertices, which coincides with Agent 2's move through its second vertex. Thus, Agent 1 has to control its speed in order to meet Agent 2 at the same time point ($t = 0.6$ in the figure) at the rendezvous point. This example shows that edges must contain duration information about which we have to reason in order to temporally and spatially synchronize agents.

By casting the collaborative common path planning problem into a single path planning problem, we can run an A* algorithm on the cross-product graph. To this end, additional information must be embedded in the nodes of the product graph in order for A* to expand the product graph properly. Said differently, rules to expand the product graph on nodes closed by A* need specific information explained below.

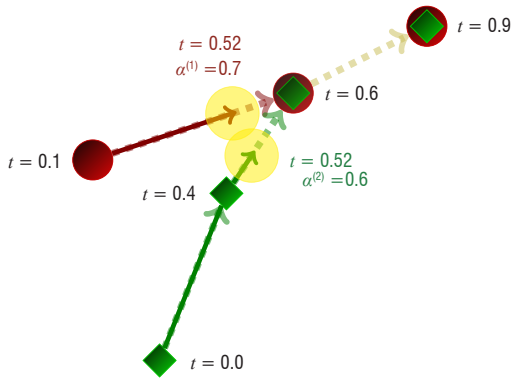


Figure 2 – Cross-product graph: at $t = 0.52$, Agent 1 is at the $\alpha^{(1)}$ portion of the edge linking the first and second vertices of its graph, while Agent 2 is at the $\alpha^{(2)}$ portion of the edge linking the second and third vertices of its graph.

Optimal algorithm

The optimal algorithm reasons in the combined spatial and temporal spaces without decoupling spatial and temporal synchronization logics of the agents. To this end, we define the following cross-product graph.

Cross-product labelled graph

The coupled cross-product labelled graph is defined as $\mathcal{G}^{\otimes} = \left(V^{\otimes}, E^{\otimes}\right)$ with:

- $V^{\otimes} = E^{(1)} \times [0;1] \times E^{(2)} \times [0;1]$, such that $v^{\otimes} = \left(e^{(1)}, \alpha^{(1)}, e^{(2)}, \alpha^{(2)}\right) \in V^{\otimes}$ represents the $\alpha^{(i)}$ percentage of completion of each agent i's move along edge $e^{(i)}$ in its own graph. We recall that $e^{(1)}$ and $e^{(2)}$ are each labelled with the cost and duration of the move of the agent in its own graph, meaning that they each take the form

of a tuple in $V^{(i)} \times V^{(i)} \times \mathbb{R}_+ \times \mathbb{R}_+$. It is especially important for the understanding of the next paragraphs to note that *there can exist in each agent's graph many different edges with the same incoming vertex and outgoing vertex but with different durations (and costs)*.

- $E^{\otimes} \subseteq V^{\otimes} \times V^{\otimes} \times \mathbb{R}_+^4$, such that $e^{\otimes} = \left(v_1^{\otimes}, v_2^{\otimes}, c^{(1)}, d^{(1)}, c^{(2)}, d^{(2)}\right)$ represents a labelled cross-product transition of both agents from a cross-product position $v_1^{\otimes}$ to another cross-product position $v_2^{\otimes}$, with labels $\left(c^{(1)}, d^{(1)}, c^{(2)}, d^{(2)}\right)$, representing the cost and the duration of the original individual graph edges that are being travelled along by each agent. These labels are needed when constructing the edges outgoing from a given cross-product position, in order to know how much time and cost remain to be consumed and paid along the current edge of each agent's original graph, in case the cross-product position may correspond to an intermediate position of one of the agents in its own graph (meaning that the agent has not yet finished travelling along its current edge in its own graph when transitioning to another cross-product vertex).

For instance, consider Figure 2 and assume that Agent 1 (resp., Agent 2)'s path as it appears from left to right in this figure is noted as $\sigma^{(1)}$ (resp. $\sigma^{(2)}$). The current positions of both agents depicted in Figure 2 is encoded in a product graph vertex $v_1^{\otimes} = \left(\sigma^{(1)}(1), 0.7, \sigma^{(2)}(2), 0.6\right)$, which means that Agent 1 (resp. Agent 2) is at 70% (resp., 60%) of completion in transitioning to the in-vertex to the out-vertex of its first (resp., second) transition along its path. Now, assume that the next event corresponds to Agents 1 and 2 reaching the out-vertices of their current respective edges (resp., $\sigma^{(1)}(1)$ and $\sigma^{(2)}(2)$) simultaneously at $t = 0.6$, which happens to be a rendezvous point, the next cross-product vertex will be $v_2^{\otimes} = \left(\sigma^{(1)}(2), 0, \sigma^{(2)}(3), 0\right)$[1] and the corresponding transition will be $e^{\otimes} = \left(v_1^{\otimes}, v_2^{\otimes}, c^{(1)}, 0.5, c^{(2)}, 0.2\right)$, since Agent 1 (resp., Agent 2) was traveling along its current edge for $d = 0.6 - 0.1 = 0.5$ (resp., $d = 0.6 - 0.4 = 0.2$) time units.

Graph expansion

Once the cross-product graph is constructed, we can run a single-agent path planning algorithm like A* on it, then retrieve the individual moves from the cross-product solution plan. Thus, we first need to instantiate the cross-product graph: we generate it lazily from the starting cross-product vertex – which is simply the pair of starting vertices of each agent – and expand it on-demand (e.g., when A* looks at successor vertices of the current closed vertex) by constructing outgoing cross-product edges from a given current cross-product vertex. Algorithm 1 formalizes this lazy cross-product graph expansion.

The algorithm tests different cases corresponding to whether each agent i is located at a vertex of its own graph (i.e., $\alpha^{(i)} = 1$), where a decision regarding the next successor vertex to target has to be taken, or if it is located between two successive vertices of its own graph (i.e., $0 < \alpha^{(i)} < 1$), where it can only continue to execute the current move to the next targeted successor vertex. When an agent is at a decision point, we iterate over the edges of its own graph whose incoming vertex is equal to the outgoing vertex of the previously executed edge (e.g., Lines 3 and 19). Then, we compare the remaining durations of the edges of each agent in their own graphs (e.g., Lines 4 and 20), in order to know which agent will reach its next targeted successor vertex first and update the completion ratio of the other agent – which may not reach its next targeted successor vertex in the same move – accordingly (e.g., Lines 7, 23 and 27). Note that the case $\left(1-\alpha^{(1)}\right) \cdot d^{(1)}\left(e^{(1)}\right) < \left(1-\alpha^{(2)}\right) \cdot d^{(2)}\left(e^{(2)}\right)$ is impossible, since at least one agent reaches its next targeted successor vertex at each graph expansion. When both agents are at a decision point (i.e., Line 2) we check whether a formation is feasible, in which case we add cross-product edges corresponding to the two agents traveling along the same common path with appropriate formation costs and durations (Lines 13 to 17).

Conditions on initial and goal vertices

Special attention specific to the CCPP problem must be paid to the initial and goal vertices. If we do not allow an agent that has reached its goal vertex to wait for the other agent to reach its own goal, it significantly reduces the chance of finding a solution because it means that we would force the two agents to reach their goal vertices at the same time. However, such a condition is not desired in practice, so we allow each agent to wait for the other agent at its goal vertices by adding in its own graph self-transitions from its goal vertices labelled with the durations of all of the edges of the other agent. Doing so, whatever the location of the other agent in its own graph, the agent at the goal will always transit again to its goal for each atomic move of the other agent. Since this "goal holding" property might be undesirable in very specific applications, we activate or deactivate it according to a test function named `can_hold`. This logic, which is implemented in Lines 4 to 6 of Algorithm 2, is also valid for initial vertices since holding the initial vertex of an agent while the other has begun to move increases the chance of finding a (spatial and temporal) synchronization point where both agents can begin to move in formation. Indeed, meeting at a given rendezvous point at a given time can be achieved by controlling each agent's edge speed or initial vertex starting time. Even if better formation plans can be found by allowing one agent to hold its initial vertex for an amount of time dictated by the other agent's initial transition duration, it is of course less optimal than an approach that would allow a free continuous holding duration irrespective of the other agent's initial transition duration. However, such an approach would be much harder computationally, since it would require continuous optimization variables to be considered, whereas our approach allows us to keep reasoning about only discrete optimization variables.

Moreover, some applications might require the two agents to start at predefined time points, or, in other words, that the difference of their starting times be constrained to a predefined value. This is, for instance, the case of two flights that might want to fly somewhere in formation in order to save fuel burn given the current day's weather conditions but whose takeoff times are set several months in advance and cannot be significantly changed at the last minute. To do this, we test for a condition `must_shift` that, if true, adds a predefined amount of time to the duration of the first transition of each agent from their initial vertices (see Lines 7 to 9 of Algorithm 2). Note that this option is incompatible with the "holding" one, since an agent cannot start within a predefined time interval from the other agent's start while holding its starting vertex.

1 In this example $\sigma^{(1)}(2) = \sigma^{(2)}(3)$ because the agents travel in formation from their previous vertices, and their graphs are the same. However, note that in general their graphs do not need to be the same, so their edges can be different even in formation.

Algorithm 1 – Cross-product graph expansion

Data: $v^{\otimes} = (e^{(1)}, \alpha^{(1)}, e^{(2)}, \alpha^{(2)}) \in V^{\otimes}$: cross product vertex
Result: $[e^{\otimes}]_{e^{\otimes} \in E^{\otimes}}$: list of cross product edges outgoing from $(e^{(1)}, \alpha^{(1)}, e^{(2)}, \alpha^{(2)})$

1 *successors* ← `list()`
2 **if** $\alpha^{(1)} = 1$ **and** $\alpha^{(2)} = 1$ **then**
3 **for** $(\overline{e}^{(1)}, \overline{e}^{(2)}) \in E^{(1)} \times E^{(2)} : e^{(1)}.\texttt{out}() = \overline{e}^{(1)}.\texttt{in}()$ **and** $e^{(2)}.\texttt{out}() = \overline{e}^{(2)}.\texttt{in}()$ **do**
4 **if** $d^{(1)}(\overline{e}^{(1)}) < d^{(2)}(\overline{e}^{(2)})$ **then**
5 $\overline{c} \leftarrow c^{(1)}(\overline{e}^{(1)}) + c^{(2)}(\overline{e}^{(2)}) \cdot \frac{d^{(1)}(\overline{e}^{(1)})}{d^{(2)}(\overline{e}^{(2)})}$;
6 $\overline{d} \leftarrow d^{(1)}(\overline{e}^{(1)})$;
7 $\overline{\alpha}^{(1)} \leftarrow 1$; $\overline{\alpha}^{(2)} \leftarrow \frac{d^{(1)}(\overline{e}^{(1)})}{d^{(2)}(\overline{e}^{(2)})}$;
8 **else**
9 Symmetry of Lines 5 to 7
10 $\overline{v}^{\otimes} \leftarrow V^{\otimes}.\texttt{add}(\overline{e}^{(1)}, \overline{\alpha}^{(1)}, \overline{e}^{(2)}, \overline{\alpha}^{(2)})$;
11 $\overline{e}^{\otimes} \leftarrow E^{\otimes}.\texttt{add}(v^{\otimes}, \overline{v}^{\otimes}, \overline{c}, \overline{d})$;
12 *successors*.`add`$(\overline{e}^{\otimes})$;
13 **if** `can_make_formation`$(\overline{e}^{(1)}, \overline{e}^{(2)})$ **then**
14 $\overline{c} \leftarrow c^{\otimes}(\overline{e}^{(1)}, \overline{e}^{(2)})$; $\overline{d} \leftarrow d^{\otimes}(\overline{e}^{(1)}, \overline{e}^{(2)})$;
15 $\overline{v}^{\otimes} \leftarrow V^{\otimes}.\texttt{add}(\overline{e}^{(1)}, 1, \overline{e}^{(2)}, 1)$;
16 $\overline{e}^{\otimes} \leftarrow E^{\otimes}.\texttt{add}(v^{\otimes}, \overline{v}^{\otimes}, \overline{c}, \overline{d})$;
17 *successors*.`add`$(\overline{e}^{\otimes})$;
18 **else if** $\alpha^{(1)} = 1$ **and** $\alpha^{(2)} < 1$ **then**
19 **for** $\overline{e}^{(1)} \in E^{(1)} : e^{(1)}.\texttt{out}() = \overline{e}^{(1)}.\texttt{in}()$ **do**
20 **if** $(1-\alpha^{(2)}) \cdot d^{(2)}(e^{(2)}) < d^{(1)}(\overline{e}^{(1)})$
21 $\overline{c} \leftarrow (1-\alpha^{(2)}) \cdot \left(c^{(1)}(\overline{e}^{(1)}) \cdot \frac{d^{(2)}(e^{(2)})}{d^{(1)}(\overline{e}^{(1)})} + c^{(2)}(e^{(2)}) \right)$
22 $\overline{d} \leftarrow (1-\alpha^{(2)}) \cdot d^{(2)}(e^{(2)})$
23 $\overline{\alpha}^{(1)} \leftarrow \frac{(1-\alpha^{(2)}) \cdot d^{(2)}(e^{(2)})}{d^{(1)}(\overline{e}^{(1)})}$; $\overline{\alpha}^{(2)} \leftarrow 1$
24 **else**
25 $\overline{c} \leftarrow c^{(1)}(\overline{e}^{(1)}) + c^{(2)}(e^{(2)}) \cdot \frac{d^{(1)}(\overline{e}^{(1)})}{d^{(2)}(e^{(2)})}$
26 $\overline{d} \leftarrow d^{(1)}(\overline{e}^{(1)})$
27 $\overline{\alpha}^{(1)} \leftarrow 1$; $\overline{\alpha}^{(2)} \leftarrow \overline{\alpha}^{(2)} + \frac{d^{(1)}(\overline{e}^{(1)})}{d^{(2)}(e^{(2)})}$
28 $\overline{v}^{\otimes} \leftarrow V^{\otimes}.\texttt{add}(\overline{e}^{(1)}, \overline{\alpha}^{(1)}, e^{(2)}, \overline{\alpha}^{(2)})$;
29 $\overline{e}^{\otimes} \leftarrow E^{\otimes}.\texttt{add}(v^{\otimes}, \overline{v}^{\otimes}, \overline{c}, \overline{d})$;
30 *successors*.`add`$(\overline{e}^{\otimes})$;
31 **else if** $\alpha^{(1)} < 1$ **and** $\alpha^{(2)} = 1$ **then**
32 Symmetry of Lines 19 to 27
33 **return** *successors*;

Algorithm 2 – Make initial and goal conditions

Data: $v_s^{(i)}, i \in \{1,2\}$: Agent i's starting vertex; $v_g^{(i)}, i \in \{1,2\}$: Agent i's goal vertex
1 **for** $i \in \{1,2\}$ **do**
2 $j \leftarrow i$ **mod** 2+1;
3 **for** $v^{(i)} \in \{v_s^{(i)}, v_g^{(i)}\}$ **do**
4 **if** `can_hold(`$v^{(i)}$`)` **then**
5 **for** $e^{(j)} \in E^{(j)}$ **do**
6 $E^{(i)}$`.add(`$v^{(i)}, v^{(i)}, 0, d^{(j)}(e^{(j)})$`)`;
7 **if** `must_shift(`$v_s^{(i)}$`)` **then**
8 **for** $e^{(i)} \in E^{(i)} : e^{(i)}$`.in()` $= v_s^{(i)}$
9 $d^{(i)}[e^{(i)}] \leftarrow d^{(i)}[e^{(i)}] +$ `shift` $(v_s^{(i)})$;

Global algorithm

The optimal CCPP solving procedure is presented in Algorithm 3. It proceeds in three logical steps:

1. Establish initial and goal conditions using Algorithm 3 (Line 1);
2. Construct the cross-product initial and goal vertices and running A* by lazily expanding the cross-product graph on closed vertices using Algorithm 2 (Lines 2 to 4);
3. Extract the individual plans of each agent from the cross-product plan by looking at cross-product vertices whose completion ratios are equal to 1, meaning that the corresponding agent is located at a decision vertex but not in-between two successive vertices of its own graph (Lines 5 to 9).

Decoupled spatial/temporal algorithm

The complexity of the optimal algorithm lies in the coupled spatio-temporal space: in order to synchronize the agents at potential formation points, both spatially and temporally, it has to memorize cross-product vertices containing the agents' original graph vertices and completion ratios of traveling along these edges. By noting as $D^{(i)}$ an upper bound on the number of different durations of each travel between two successive vertices of Agent i's own graph (i.e., the maximum number of edges with the same given incoming and outgoing vertices) the potential number of cross-product vertices explored by A* is $|E^{(1)}| \cdot D^{(2)} \cdot |E^{(2)}| \cdot D^{(1)} = |V^{(1)}|^2 \cdot D^{(2)} \cdot |V^{(2)}|^2 \cdot D^{(1)}$. In the case where both agents share the same graph and are identical (implying same possible edge durations) it amounts to visiting at most $|V|^4 \cdot D^2$ vertices, which is intractable even for very small instances.

A simple idea consists in decoupling the problems of synchronizing the agents spatially and temporally. First, we aim at finding a pair of vertices for each agent where they can begin a formation by discarding all temporal information on edge durations. It is achieved by running lazy A* with a vertex expansion procedure presented in Algorithm 4, which is composed of two phases: (1) filter out durations from each agent's graph edges by replacing each set of edges with the same incoming and outgoing vertices by a single edge without duration and whose cost is averaged on-the-fly over all of the costs of the replaced edges (Lines 1 to 10); (2) generate successor cross-product vertices from the filtered edges while checking for the possibility of beginning a formation, in which case the formation cost is appropriately set (Lines 11 to 15). Crucially, a self-transition from each agent's current vertex is added, in order to simulate the fact that, in reality, an agent might not have reached its next successor vertex when the other has indeed reached its own next vertex (Line 10). This trick actually allows the agents to reach a common formation point even if their individual numbers of atomic moves to reach it are different (see Figure 2).

Algorithm 3 – Optimal CCPP algorithm

Data: $v_s^{(i)}$: Agent i's start node; $v_g^{(i)}$: Agent i's goal node; $\mathcal{G}^{(i)} = (V^{(i)}, E^{(i)})$: Agent i's move graph; $c^{\otimes} : E^{\otimes} \to \mathbb{R}_+$: cross-product cost function; $h^{\otimes} : V^{\otimes} \times V^{\otimes} \to \mathbb{R}_+$: cross-product heuristic function
Result: $\pi^{(i)}, i \in \{1,2\}$: optimal path from $v_s^{(i)}$ to $v_g^{(i)}$
1 `make_initial_and_goal_conditions()`;
2 $v_s^{\otimes} \leftarrow ((v_s^{(1)}, v_s^{(1)}), 1, (v_s^{(2)}, v_s^{(2)}), 1)$;
3 $v_g^{\otimes} \leftarrow ((v_g^{(1)}, v_g^{(1)}), 1, (v_g^{(2)}, v_g^{(2)}), 1)$;
4 $\pi^{\otimes} \leftarrow$ `lazy_astar(`$v_s^{\otimes}, v_g^{\otimes}, h^{\otimes}$`)` using `cross_product_graph_expansion`;
5 $\pi^{(1)} \leftarrow$; `list()`; $\pi^{(2)} \leftarrow$ `list()`;
6 **for** $e^{\otimes} \in \pi^{\otimes}$ **do**
7 $(e_{out}^{(1)}, \alpha_{out}^{(1)}, e_{out}^{(2)}, \alpha_{out}^{(2)}) \leftarrow e^{\otimes}$ `.out()`;
8 **if** $\alpha_{out}^{(1)} = 1$ **then** $\pi^{(1)}$`.add(`$e_{out}^{(1)}$`)`;
9 **if** $\alpha_{out}^{(2)} = 1$ **then** $\pi^{(2)}$`.add(`$e_{out}^{(2)}$`)`;
10 **return** $(\pi^{(1)}, \pi^{(2)})$;

Algorithm 4 – Spatial graph expansion

Data: $v^{\otimes} = (v^{(1)}, v^{(2)}) \in V^{(1)} \times V^{(2)}$: cross-product vertex without time info
Result: $[e^{\otimes}]_{e^{\otimes} \in \left(V^{(1)} \times V^{(2)}\right)^2 \times \mathbb{R}_+}$: list of cross-product edges outgoing from $v^{\otimes}$

1 **for** $i \in \{1,2\}$ **do**
2 $\quad \tilde{E}^{(i)} \leftarrow$ `list()`; $n \leftarrow$ `map()`;
3 $\quad$ **for** $e^{(i)} \in E^{(i)} : e^{(i)}.\texttt{in}() = v^{(i)}$ **do**
4 $\quad\quad$ **if** $\forall e \in \tilde{E}^{(i)}, e^{(i)}.\texttt{out}() \neq e.\texttt{out}()$ **then**
5 $\quad\quad\quad \tilde{E}^{(i)}.\texttt{add}(e^{(i)}.\texttt{in}(), e^{(i)}.\texttt{out}(), c^{(i)}(e^{(i)}))$;
6 $\quad\quad\quad n\left[e^{(i)}.\texttt{out}()\right] \leftarrow 1$;
7 $\quad\quad$ **else**
8 $\quad\quad\quad \tilde{c}^{(i)}\left(e^{(i)}\right) \leftarrow \frac{1}{n\left[e^{(i)}.\texttt{out}()\right]+1}\left(n\left[e^{(i)}.\texttt{out}()\right] \cdot \tilde{c}^{(i)}\left(e^{(i)}\right) + c^{(i)}\left(e^{(i)}\right)\right)$;
9 $\quad\quad\quad n\left[e^{(i)}.\texttt{out}()\right] \leftarrow n\left[e^{(i)}.\texttt{out}()\right]+1$;
10 $\quad \tilde{E}^{(i)}.\texttt{add}(v^{(i)}, v^{(i)}, 0)$;
11 *successors* $\leftarrow$ `list()`;
12 **for** $\tilde{e}^{(1)} \in \tilde{E}^{(1)}$ **and** $\tilde{e}^{(2)} \in \tilde{E}^{(2)}$ **do**
13 $\quad$ *successors* `.add(` $\tilde{e}^{(1)}, \tilde{e}^{(2)}, \tilde{c}^{(1)}\left(\tilde{e}^{(1)}\right) + \tilde{c}^{(2)}\left(\tilde{e}^{(2)}\right)$ `)`;
14 $\quad$ **if** `can_make_formation(` $\tilde{e}^{(1)}, \tilde{e}^{(2)}$ `)` **then**
15 $\quad\quad$ *successors* `.add(` $\tilde{e}^{(1)}, \tilde{e}^{(2)}, c^{\otimes}\left(\tilde{e}^{(1)}, \tilde{e}^{(2)}\right)$ `)`;
16 **return** *successors*

The second phase consists in planning for the agents' coordination in the temporal space only by running lazy A* with a temporal graph expansion schema, which constrains the spatial vertex transitions to the plans found during the first spatial planning phase, as formalized in Algorithm 5. Thus, the cross-product vertex is composed of the pair of indices of the moves in the spatial plans currently executed by the agents, which is sufficient to track each agent's plan execution and to obtain the spatial edge corresponding to the current plan indices (Lines 2 to 6). Once this edge has been extracted, we obtain all of the original time-labelled edges with the same incoming and outgoing vertices, in order to retrieve timing information that was lost during the first spatial synchronization phase (Lines 7 to 9). Finally, we generate successor cross-product vertices from the reconstructed time-labelled edges while checking for the possibility of beginning a formation, in which case the formation cost is appropriately set (Lines 10 to 14). Note that, at this stage, time labels are part of the reconstructed vertices so we can check whether it is possible for the two agents to simultaneously reach a potential formation vertex.

Algorithm 5 – Temporal graph expansion

Data: $\{\tilde{\pi}^{(i)}\}_{i \in \{1,2\}}$: Agent i's spatial plan w/o timing information; $v^{\otimes} = (k_1, k_2)$: cross-product vertex containing indexes of current spatial vertices in Agents 1 and 2's plans

1 **for** $i \in \{1,2\}$ **do**
2 $\quad v^{(i)} \leftarrow \tilde{\pi}^{(i)}[k_i]$;
3 $\quad$ **if** $k_i \neq \tilde{\pi}^{(i)}.\texttt{length}() - 1$ **then**
4 $\quad\quad \overline{v}^{(i)} \leftarrow \tilde{\pi}^{(i)}[k_i + 1]$;
5 $\quad$ **else**
6 $\quad\quad \overline{v}^{(i)} \leftarrow v^{(i)}$;
7 $\quad \tilde{E}^{(i)} \leftarrow$ `list()`;
8 $\quad$ **for** $e^{(i)} \in E^{(i)} : e^{(i)}.\texttt{in}() = v^{(i)}$ **and** $e^{(i)}.\texttt{out}() = \overline{v}^{(i)}$ **do**
9 $\quad\quad \tilde{E}^{(i)}.\texttt{add}(e^{(i)})$;
10 *successors* $\leftarrow$ `list()`;
11 **for** $\tilde{e}^{(1)} \in \tilde{E}^{(1)}$ **and** $\tilde{e}^{(2)} \in \tilde{E}^{(2)}$ **do**
12 $\quad$ *successors* `.add(` $\tilde{e}^{(1)}, \tilde{e}^{(2)}, \tilde{c}^{(1)}\left(\tilde{e}^{(1)}\right) + \tilde{c}^{(2)}\left(\tilde{e}^{(2)}\right)$ `)`;
13 $\quad$ **if** `can_make_formation(` $\tilde{e}^{(1)}, \tilde{e}^{(2)}$ `)` **then**
14 $\quad\quad$ *successors* `.add(` $\tilde{e}^{(1)}, \tilde{e}^{(2)}, c^{\otimes}\left(\tilde{e}^{(1)}, \tilde{e}^{(2)}\right)$ `)`;
15 **return** *successors*

Algorithm 6 – Spatio-temporal decoupled CCPP

Data: $v_s^{(i)}$: Agent i's start node; $v_g^{(i)}$: Agent i's goal node; $\mathcal{G}^{(i)} = \left(V^{(i)}, E^{(i)}\right)$: Agent i's move graph; $c^{\otimes} : E^{\otimes} \to \mathbb{R}_+$: cross-product cost function; $h^{\otimes} : V^{\otimes} \times V^{\otimes} \to \mathbb{R}_+$: cross-product heuristic function
Result: $\pi^{(i)}, i \in \{1,2\}$: optimal path from $v_s^{(i)}$ to $v_g^{(i)}$

```
make_initial_and_goal_conditions();
v_s^⊗ ← (v_s^(1), v_s^(2))
v_g^⊗ ← (v_g^(1), v_g^(2))
π_spatial^⊗ ← lazy_astar(v_s^⊗, v_g^⊗, h^⊗) using spatial_graph_expansion;
{π̃^(i)}_{i∈{1,2}} ← extract_individual_plans(π_spatial^⊗);
ṽ_s^⊗ ← (0,0);
ṽ_g^⊗ ← (π̃^(1).length()−1, π̃^(2).length()−1);
π_temporal^⊗ ← lazy_astar(ṽ_s^⊗, ṽ_g^⊗, h^⊗) using temporal_graph_expansion({π̃^(i)}_{i∈{1,2}});
return extract_individual_plans(π_temporal^⊗);
```

The global decoupled spatio-temporal procedure is presented in Algorithm 6. The pseudo-code is quite obvious given the previous explanations of the successive spatial and temporal synchronization phases. Whereas this decoupled schema certainly does not bring optimal solutions to the CCPP problem, it allows us to solve large problems with satisfactory quality in a very small fraction of the time spent by the optimal coupled algorithm.

Heuristics

Given that our algorithms rely on A*, we need a heuristic function to guide the search in the cross-product graph. Designing a generic admissible and informative heuristic is especially challenging because the cost function changes when the agents move in formation and we do not know in advance when and where they will begin or end a formation pattern. A brute-force method would consist in iterating over all possible pairs of each agent's vertices representing joining and breaking formation points, in order to ease heuristic estimate computations. However, we would spend too much time iterating over too numerous pairs, which would ruin the benefit of using heuristic functions.

Admissible heuristic

A simple admissible but non-informative heuristic estimate consists in summing individual heuristic estimates for each agent in their own graphs using formation costs $c^{\otimes}$, even if they do not travel in formation.

Any admissible heuristic for single-agent path planning (e.g., Euclidean distance, Manhattan distance, etc.) can be used to calculate the individual heuristic estimates but considering formation costs instead of standard single-agent costs (i.e., like in the case of agents travelling in formation from their starting to ending locations). This allows us to obviate the difficult question of when and where they join together or break the formation, but we lose the crucial information that their individual moving costs are actually generally much higher when they do not travel in formation. Formally, we note as $h^{(i)} : V^{(i)} \times V^{(i)} \to \mathbb{R}_+$ a heuristic function defined on Agent i's vertices in its own graph, such that $h^{(i)}\left(v_A^{(i)}, v_B^{(i)}\right) \leqslant C^{\otimes,i}\left(v_A^{(i)}, v_B^{(i)}\right)$ where $C^{\otimes,i}\left(v_A^{(i)}, v_B^{(i)}\right)$ is Agent i's contribution to the duet cost-to-go function if it travels in formation with the other agent from the starting vertex $v_A^{(i)}$ to the target vertex $v_B^{(i)}$. Said differently, $h^{(i)}$ is computed with edge costs assumed to be all equal to Agent i's contribution to $c^{\otimes}$, even if it does not travel in formation. Therefore, an admissible heuristic is $h_{adm}^{\otimes} = h^{(1)} + h^{(2)}$.

Informative heuristic

In order to use the important information that the cost function is higher when the agents do not move in formation, we try to approximately guess with simple geometric reasoning where the agents will make a formation and further break it. This simple geometric method, depicted in Figure 3 and formalized in Algorithm 7, assumes the vertices of the agents' own graphs to be elements of a Euclidean space. Given a pair of starting vertices $(v_A^{(1)}, v_A^{(2)})$ and target vertices $(v_B^{(1)}, v_B^{(2)})$ we compute their respective barycenters $\tilde{v}_A$

Algorithm 7 – CCPP heuristic on Euclidean space

Data: $\{\mathrm{v}_A^{(i)}\}_{i\in\{1,2\}}$: Agent i's starting point (vector); $\{\mathrm{v}_B^{(i)}\}_{i\in\{1,2\}}$: Agent i's target point (vector); $\varepsilon > 0$: spatial precision between possible joining or breaking points
Result: Heuristic estimate of the CCPP problem from joint starting point to joint target point

$\tilde{\mathrm{v}}_A \leftarrow \frac{1}{2}\left(\mathrm{v}_A^{(1)} + \mathrm{v}_A^{(2)}\right)$; $\tilde{\mathrm{v}}_B \leftarrow \frac{1}{2}\left(\mathrm{v}_B^{(1)} + \mathrm{v}_B^{(2)}\right)$;

$N \leftarrow argmin\left\{n \in \mathbb{N}_+ : n \geqslant \frac{\|\tilde{\mathrm{v}}_B - \tilde{\mathrm{v}}_A\|}{\varepsilon}\right\}$;

for $i \leftarrow 0$ **to** N **do**

  $\tilde{\mathrm{v}}_i \leftarrow \tilde{\mathrm{v}}_s + \frac{i}{N}\left(\tilde{\mathrm{v}}_B - \tilde{\mathrm{v}}_A\right)$;

$h^{\otimes}\left((\mathrm{v}_A^{(1)}, \mathrm{v}_A^{(2)}), (\mathrm{v}_B^{(1)}, \mathrm{v}_B^{(2)})\right) \leftarrow min_{0\leqslant i\leqslant N/2}\left\{\omega^{\otimes}\left\|\tilde{\mathrm{v}}_{N-i} - \tilde{\mathrm{v}}_i\right\| + \omega^{(1)}\left(\left\|\tilde{\mathrm{v}}_i - \mathrm{v}_A^{(1)}\right\| + \left\|\tilde{\mathrm{v}}_{N-i} - \mathrm{v}_B^{(1)}\right\|\right) + \omega^{(2)}\left(\left\|\tilde{\mathrm{v}}_i - \mathrm{v}_A^{(2)}\right\| + \left\|\tilde{\mathrm{v}}_{N-i} - \mathrm{v}_B^{(2)}\right\|\right)\right\}$;

return $h^{\otimes}\left((\mathrm{v}_A^{(1)}, \mathrm{v}_A^{(2)}), (\mathrm{v}_B^{(1)}, \mathrm{v}_B^{(2)})\right)$;

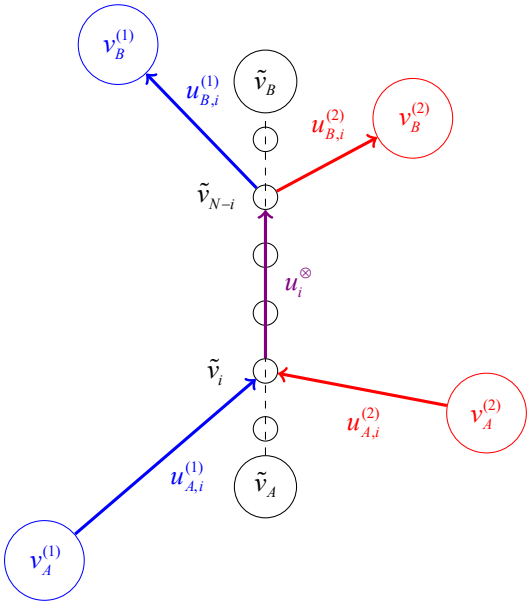


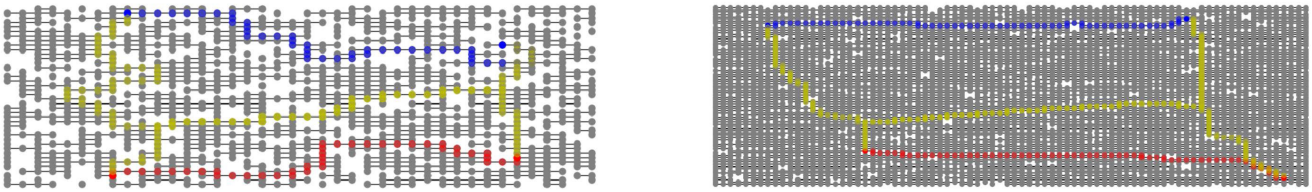
Figure 3 – Heuristic estimate of the CCPP problem in Euclidean spaces from a joint position $v_A^{\otimes} = (v_A^{(1)}, v_A^{(2)})$ to $v_B^{\otimes} = (v_B^{(1)}, v_B^{(2)})$: $h^{\otimes}(v_A^{\otimes}, v_B^{\otimes}) = min_{0 \leqslant i \leqslant N/2} \left\{ \omega^{\otimes} \left\| u_i^{\otimes} \right\| + \omega^{(1)} \left(\left\| u_{A,i}^{(1)} \right\| + \left\| u_{B,i}^{(1)} \right\| \right) + \omega^{(2)} \left(\left\| u_{A,i}^{(2)} \right\| + \left\| u_{B,i}^{(2)} \right\| \right) \right\}$, where ω's represent speed-dependent moving costs per length unit as functions of duration-labelled edges

and $\tilde{v}_B$ (Line 1) and assume that the agents will travel in formation somewhere along the segment joining these barycenters – which is obviously not guaranteed, but hopefully close to the optimal formation segment, preventing us from proving the admissibility of this heuristic estimate. Then we iterate over possible joining and breaking points along the median segment $[\tilde{v}_A; \tilde{v}_B]$, assumed to be symmetric for simplicity (Lines 2 to 4). Finally (Line 5), we search for the minimum estimate over these possible joining/breaking patterns of the sum of the individual contributions of the agents when not in formation from their initial starting vertices to the joining point and from the breaking point to their target vertices (blue and red segments in Figure 3), and of the heuristic formation cost-to-go from the joining point to the breaking point (violet segment in Figure 3).

Experiments

Navigation grids

In this section we propose to experimentally compare the different variants of our CCPP solving algorithms on navigation grids that are Euclidean spaces, so that we can test our informative geometric heuristic. As shown in Figure 1, we experimented with various random grids having various sizes, obstacle densities and formation moving costs. Table 2, whose caption defines shortcut names for the tested variants, summarizes the results. We implemented the algorithms in pure Python and run the tests on Intel's core i7 with 2.80 GHz CPUs and 16 Go of RAM (the tests used at most 1.5 Go of RAM). For each



(a) Small sparse grid

(b) Large dense grid

Figure 4 – Formation path planning experimented on randomly generated grids with various sizes and obstacle densities. Individual optimal agents' paths regardless of the CCPP problem are in red and blue, while CCPP agent paths are in yellow.

	PATHS TEAM COST				NUMBER OF EXPLORED A* NODES				CPU TIME (seconds)			
Problem	CA	CI	DA	DI	CA	CI	DA	DI	CA	CI	DA	DI
NG-5-5-75	**1.8**	2.18	**1.8**	**1.8**	22525	11577	422	**296**	2.67	0.888	0.0377	**0.0328**
NG-5-5-50	**1.1**	**1.1**	**1.1**	**1.1**	6819	2456	164	**86**	0.319	0.0549	0.0169	**0.013**
NG-10-10-75	**2.94**	2.99	4.4	2.99	582997	33464	2669	**784**	85.5	1.92	0.658	**0.263**
NG-10-10-50	**3.1**	3.2	3.43	3.5	512555	159521	2107	**1620**	77	16	0.6	**0.47**
NG-20-20-75	—	**7.03**	9.7	8.23	—	757991	17775	**8004**	—	99.8	6.35	**3.59**
NG-20-20-50	—	**7.13**	8.05	7.58	—	1139752	11859	**4066**	—	151	11.2	**5.83**
NG-40-40-75	—	—	17	**16.7**	—	—	47172	**14167**	—	—	153	**49.8**
NG-40-40-50	—	—	16.9	**14.3**	—	—	21410	**18504**	—	—	184	**98.7**
NG-80-80-75	—	—	—	**34.1**	—	—	—	**97244**	—	—	—	**732**
NG-80-80-50	—	—	—	**40.6**	—	—	—	**110855**	—	—	—	**1140**

Table 1 – Navigation grid experiments: comparison of solution path team costs, numbers of nodes explored by A* in the cross-product graph and CPU times, for the coupling algorithm using the admissible generic heuristic (CA) or the informative geometric one (CI), and similarly for the decoupling algorithm (DA and DI). Problems are noted as NG-X-Y-P, where X and Y stand for the x and y dimensions of the grid, and P represents the cost reduction percentage of edges where the two agents move in formation side by side, in comparison with non-formation edges (i.e., $c^{\otimes} = (1-P)/100 \times (c^{(1)} + c^{(2)})$).

algorithm we time out the search at 3 minutes of computation. Note that the only optimal algorithm is CA, i.e., the coupling algorithm equipped with the admissible heuristic.

Coupling algorithm vs. decoupling algorithm

The coupling algorithms can only solve the first problems, since they explore significantly more nodes during A* graph search than the decoupling algorithms do: up to 280 times more for the same heuristics. The best version of the coupling algorithm times out after the 7th problem. The decoupling algorithms' solution quality is at most at 12% of the optimal solution, even finding optimal ones on small problems.

Admissible heuristic vs. informative heuristic

As expected, the informative heuristic allows the coupling or decoupling algorithms to solve more problems, since it provides more accurate estimates of the team cost-to-go. However, it is not admissible, which degrades the solution quality of the coupling algorithm in comparison with the admissible heuristic. However, interestingly, we observe the inverse behavior for the decoupling algorithm, which seems to indicate that the negative impact on the solution quality of decoupling is higher than when using a non-admissible heuristic.

Impact of formation edge cost reduction

Since higher cost reductions tend to incentivize agents more to make formations, we expect the CCPP problem to be more difficult to solve with lower cost reductions, since more cross-product nodes should be explored before finding potential joining points that can provide some benefits. For reasons that we do not fully understand yet, this is partially observed with the informative heuristic only. More nodes are systematically explored by A* with 75% formation cost reduction using the admissible heuristic for both algorithms.

Formation Flight

We ran the DI algorithm, i.e., the only scalable one, on formation flight problems for hundreds of flights constrained to fly over official airway graphs published by aviation authorities, which contain more than 30000 of vertices at different altitudes. Each aircraft is assigned a route, which is defined by a pair of departure and destination airports, but the flight paths have to be optimized in the 4D space composed of the aircraft's 3D waypoint positions and the times when it flies through each waypoint. In addition to optimizing the flight routes for a given pair of leader-follower aircraft, we must choose the best leader-follower pairs among all of the possible pairs. To this end, we note as $N \in \mathbb{N}_+$ the number of aircraft to pair and as $\mathcal{F} \subseteq [1;N]^2$ the possible leader-follower pairs for which the CCPP problem returns a feasible solution. For $i \in [1;N]$, we note as $\mathcal{F}^F(i) = \{j \in [1;N], (i,j) \in \mathcal{F}\}$ the set of possible followers that can pair with i. Symmetrically, for all $j \in [1;N]$, $\mathcal{F}^L(j) = \{i \in [1;N], (i,j) \in \mathcal{F}\}$ is the set of possible leaders that can pair with j. Finally, we note c_{ij} the global cost of the flights of a leader i and follower j flying in formation on a subset of their routes (i.e., solution to the CCPP problem), and c_i and c_j the costs of their solo flights if they had flown without pairing at some point.

The solution is computed in two successive phases. First, we compute all of the possible leader-follower formation routes by running the CCPP algorithm on all possible pairs of aircraft, as well as all of the solo flight routes. This phase allows us to compute the set $\mathcal{F}$ of feasible formation flights, to fill in the formation costs c_{ij} for all $(i,j) \in \mathcal{F}$, and the solo flight costs c_i for all $i \in [1;N]$. Second, we compute the best possible pair assignments by running the following integer linear program whose optimization variables are $f_{ij} \in \{0,1\}$:

$$maximize: \sum_{(i,j)\in\mathcal{F}} f_{ij}\left(c_i + c_j - c_{ij}\right) \quad (1)$$

$$subject\ to: \forall i \in [1;N], \sum_{j\in\mathcal{F}^F(i)} f_{ij} \leqslant 1 \quad (2)$$

$$\forall j \in [1;N], \sum_{i\in\mathcal{F}^L(j)} f_{ij} \leqslant 1 \quad (3)$$

$$\forall (i,j) \in \mathcal{F}, f_{ij} + \sum_{k\in\mathcal{F}^L(i)} f_{ki} \leqslant 1 \quad (4)$$

$$\forall (i,j) \in \mathcal{F}, f_{ij} + \sum_{k\in\mathcal{F}^F(j)} f_{jk} \leqslant 1 \quad (5)$$

Decision variables f_{ij} are equal to 1 whenever Leader i is paired with Follower j. Constraint 2 (resp. 3) means that each possible leader i (resp. follower j) can be paired with a single follower j (resp. leader i). Constraint 4 (resp. 5) means that a given leader (resp. follower) cannot be the follower (resp. leader) of another aircraft. The objective function is to maximize the gain of flying in formations, i.e., the difference between the sum of individual solo flight costs $c_i + c_j$ and the formation cost c_{ij} whenever a formation f_{ij} is selected. An excerpt of the resulting formation flights over the Atlantic Ocean is shown in Figure 5. Note that aircraft have to fly though so-called *North Atlantic Tracks,* which are very beneficial to formation flights since solo flights take those tracks anyway.

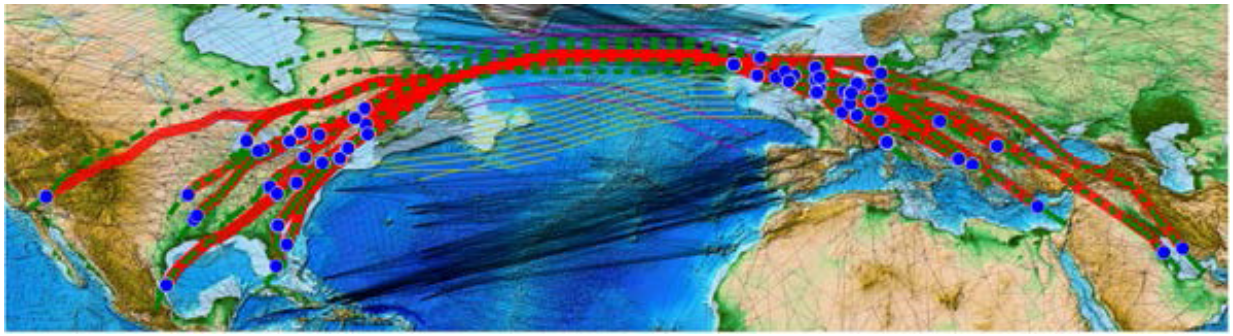


Figure 5 – Formation Flight test bench: common formation paths are shown in solid red, solo (non-cooperative) paths appear in dashed green, the airway graph is shown in solid gray, and airports appear as blue dots.

Discussion

Our experimental results show that only the decoupling algorithm equipped with informative heuristics like the geometric one that we proposed can reasonably scale. However, we also observed a loss in quality, which can be of up to 12 % of the optimal team cost on some benchmarks even if the loss is in general much lower. These results encourage us to conduct further research on improving the decoupling algorithm by trading a little computation time off for quality regaining. Possible research directions would consist in alternatively iterating between the spatial and temporal synchronization phases instead of a single iteration that freezes the spatial phase independently from the temporal phase's solution constraints as we currently do.

Another important line of research would look at improving heuristics: making the informative geometric heuristic admissible, as well as designing a generic informative heuristic that differentiates the costs of the pre- and post-formation paths of the agent from the formation common path – like our geometric heuristic, but in general state spaces that are not necessarily Euclidean. Finally, extension of the CCPP problem to continuous edge duration spaces would require the mixing of A* search in the spatial space, followed by scheduling techniques in the temporal space to control the speed of frozen spatial moves as a true continuum of values instead of continuous values chosen in a discrete set [14]. However, note that such settings are not always desirable, especially not in formation flight, as explained in the introduction ■

References

[1] Y. CHEN, J. YU, X. SU, G. LUO - *Path Planning for Multi-UAV Formation.* Journal of Intelligent & Robotic Systems, 77(1):229-246, 2015.

[2] H. CHOSET, K. M. LYNCH, S. HUTCHINSON, G. A. KANTOR, W. BURGARD, L. E. KAVRAKI, S. THRUN - *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, Pittsburgh, PA, 2005.

[3] B. COLTIN, M. M. VELOSO - *Ridesharing with Passenger Transfers.* 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014, pages 3278-3283. IEEE.

[4] L. DENG, X. MA, J. J. GU, Y. LI - *Multi-Robot Dynamic Formation Path Planning with Improved Polyclonal Artificial Immune Algorithm*. Control and Intelligent Systems, 42, 2014.

[5] D. FERGUSON, M. LIKHACHEV, A. T. STENTZ - *A Guide to Heuristic-Based Path Planning.* Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS), Pittsburgh, PA, 2005.

[6] M. GOLDENBERG, A. FELNER, R. STERN, J. SCHAEFFER - *A* Variants for Optimal Multi-Agent Pathfinding.* D. Borrajo, A. Felner, R. E. Korf, M. Likhachev, C. L. López, W. Ruml, N. R. Sturtevant, editors, Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012. AAAI Press.

[7] W. GUANGHUA, L. DEYI, G.WENYAN, J. PENG - *Study on Formation Control of Multi-Robot Systems.* 2013 Third International Conference on Intelligent System Design and Engineering Applications, pages 1335-1339, 2013.

[8] Y. HAO, S. K. AGRAWAL - *Planning and Control of UGV Formations in a Dynamic Environment: A Practical Framework with Experiments.* Robotics Auton. Syst., 51(2-3):101-110, 2005.

[9] P. E. HART, N. J. NILSSON, B. RAPHAEL - *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Trans. Systems Science and Cybernetics, 4(2):100-107, 1968.

[10] P. E. HART, N. J. NILSSON, B. RAPHAEL - *Correction to "a Formal Basis for the Heuristic Determination of Minimum Cost Paths".* SIGART Newsletter, 37:28-29, 1972.

[11] W. HÖNIG, T. K. S. KUMAR, L. COHEN, H. MA, H. XU, N. AYANIAN, S. KOENIG - *Multi-Agent Path Finding with Kinematic Constraints.* Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016., pages 477-485, 2016.

[12] J. HRNCIR, M. JAKOB - *Generalised Time-Dependent Graphs for Fully Multimodal Journey Planning.* 16th International IEEE Conference on Intelligent Transportation Systems, ITSC 2013, The Hague, The Netherlands, October 6-9, 2013, pages 2138-2145. IEEE.

[13] M. KATSEV, J. YU, S. M. LAVALLE - *Efficient Formation Path Planning on Large Graphs.*2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 6-10, 2013, pages 3606-3611.

[14] E. KELAREVA, K. TIERNEY, P. KILBY - *CP Methods for Scheduling and Routing with Time-Dependent Task Costs*.Springer, Berlin, Heidelberg, pp. 111-127 2013.

[15] D. KIRCHLER - *Efficient Routing on Multi-Modal Transportation Networks.* (Routage efficace sur réseaux de transport multimodaux). PhD thesis, École Polytechnique, Palaiseau, France, 2013.

[16] S. M. LAVALLE, J. J. KUFFNER - *Rapidly-Exploring Random Trees: Progress and Prospects.* Algorithmic and Computational Robotics: New Directions, pp. 293-308, 2000.

[17] R. L. RAFFARD, C. J. TOMLIN, S. P. BOYD - *Distributed Optimization for Cooperative Agents: Application to Formation Flight.* 2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601), volume 3, pages 2453-2459 Vol.3, 2004.

[18] M. SCHREIECK, H. SAFETLI, S. A. SIDDIQUI, C. PFLÜGLER, M. WIESCHE, H. KRCMAR - *A Matching Algorithm for Dynamic Ridesharing.* Transportation Research Procedia, 19:272-285, 2016.

[19] R. SILVEIRA, E. P. E SILVA JR., L. P. NEDEL - *Managing Coherent Groups.* Comp. Anim. Virt. Worlds, 19(3-4):295-305, 2008.

[20] R. TIWARI, A. SHUKLA, R. KALA - *Intelligent Planning for Mobile Robotics: Algorithmic Approaches.* Information Science Reference, 2012.

[21] M. VAN HELLENBERG HUBAR - *Multiple-Phase Trajectory Optimization for Formation Flight in Civil Aviation*. 2016.

[22] J. XU, S. A. NING, G. BOWER, I. KROO - *Aircraft Route Optimization for Formation Flight.* Journal of Aircraft, 51, 2014.

[23] J. YU, S. M. LAVALLE - *Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics.* IEEE Transactions on Robotics, 32(5):1163-1177, 2016.

Florent Teichteil-Koenigsbuch graduated in 2002 from the French aerospace engineering school SUPAERO. He then obtained a PhD in Artificial Intelligence from the University of Toulouse in 2005. After having worked at ONERA as a research scientist in Robotics and Artificial Intelligence from 2005 to 2015, he moved to the Airbus Central Technology Office where has been working as a data scientist and research project leader. He has published several conference and journal papers on AI decision making and autonomous robotics.

Guillaume Poveda has been an AI research engineer in Airbus R&T since 2018. He graduated from ISAE-SUPAERO in 2016 with a Master's degree in engineering along with a Master's degree in operational research. Since then, Guillaume has been working on optimization topics for aerospace. When he's not in front of his computer, Guillaume is on his bike seat or somewhere in the Pyrénées.