Hybrid Multi-scale Search for Dynamic Planning of Multi-agent Drone Traffic

Jun Xiang * and Jun Chen †
San Diego State University, San Diego, CA 92182

Yanchao Liu‡
Wayne State University, Detroit, Michigan, 48202

Unmanned aerial vehicles or drones are widely used or proposed to carry out various tasks in low-altitude airspace. In order to safely integrate drone traffic into congested airspace, the current concept of operations for drone traffic management will reserve a static traffic volume for the whole planned trajectory, which is safe but inefficient. In this paper, we propose a dynamic traffic volume reservation method for the drone traffic management system based on a multi-scale A* algorithm. The planning airspace is represented as a multi-resolution grid world, where the resolution will be coarse for the area on the far side. Therefore, each drone only needs to reserve a temporary traffic volume along the finest flight path in its local area, which helps release the airspace back to others. Moreover, the multi-scale A* can run nearly in real-time due to a much smaller search space, which enables dynamically rolling planning to consider updated information. To handle the infeasible corner cases of the multi-scale algorithm, a hybrid strategy is further developed, which can maintain a similar optimal level to the classic A* algorithm while still running nearly in real-time. The presented numerical results support the advantages of the proposed approach.

Nomenclature

b	=	branching factor of the graph of the A* task
C_a	=	arrival cost
$comt_j$	=	computing time of the jth cycle
d	=	solution depth of the A* task
d_h	=	a parameter set by the user used to decide the search algorithm indicator
d_t	=	destination of agents

*Ph.D. Student, Department of Aerospace Engineering, 5500 Campanile Dr, San Diego, CA 92182, jxiang9143@sdsu.edu , AIAA Student Member

†Assistant Professor, Department of Aerospace Engineering, 5500 Campanile Dr, San Diego, CA 92182, Jun.Chen@sdsu.edu, AIAA Senior Member (Corresponding Author)

‡Assistant Professor, Department of Industrial and Systems Engineering, 42 W Warren Ave, Detroit, MI 48202, yanchaoliu@wayne.edu

dt_j = actual processing time for the jth cycle

$d\alpha$ = increment of α

$d\beta$ = increment of β

e = error of the A* tas

$\mathbf{E}$ = edge set connects the node

Ed = Euclidean distance

e_{msa*} = edge cost

G = grid graph

g^s_{xy} = unit grid cell, located at x in the x direction, y in the y direction, with status s

$\mathbf{N}$ = full quadtree node set

$n_{l,o}$ = the node is in the full quadtree node set at the level l and its center position is o

p = current location of the agent

R = risk value function

s = search algorithm indicator

t_h = a parameter set by the user used to decide the search algorithm indicator

T_k = flight plan for agent k

x_r = a parameter set by the user controls how often the α and β are decremented

α = a parameter set by the user used to determine a node should be divided

α_h = a parameter set by the user used to decide the search algorithm indicator

β = a parameter set by the user used to determine a node should be divided

τ = time step

ω = arrival cost coefficient

I. Introduction

Airspace is predicted to be much more crowded in the next decade with the progress of aircraft technology and autonomy, [1]. Many flying cab projects are under development [1], which may become a major component of public urban transit service. At the same time, research on drones is spreading. Drones are expected to deliver packages, respond to emergencies, inspect infrastructure, and complete other tasks [2]. Unmanned aircraft technology will also bring more flights to the limited airspace since fewer pilots are required. Therefore, it is important to develop a more efficient UTM to increase the efficiency of airspace utilization. However, the current concept of operations for unmanned aircraft system (UAS) traffic management (UTM) proposed in [3] requires the reservation of a static traffic volume for the whole planned trajectory before the departure of each flight. Fig. 1 shows an example of a drone reserving airspace

in the current UTM. One of the reasons the UTM requires a massive reservation is the current trajectory planning and reservation algorithm has very high computing complexity [4]. It is difficult to modify the planned trajectory and reservation during the operation, so massive reservation is required to lower the chance of re-planning requesting

Safety is the most important factor in a UTM. In the UTM, the UAS operator is the control center collecting all the information and navigating all the participants in the system. The UAS operator is responsible for preventing all participants from entering the area with high risk unless it's necessary and keeping both UAS participants and non-participants separated. In addition, the UAS operator should be able to respond to dynamic information such as change of intent information of an aircraft, weather, and temporal constraint. The operator also needs to give priority to some flights such as public safety flights. Therefore, the trajectory planning and reservation algorithm used by the UAS operator (or the onboard computer in the near future) must meet the following requirements: 1) the flight following the planned trajectory can not conflict with any other flight; 2) the planned trajectory should avoid high-risk areas and reserved areas; 3) the algorithm must be very fast to respond to real-time incidence such as weather influences[5]; 4) each flight should reserve as little airspace as possible; 5) the planned trajectory should be as short as possible.

To cope with the five requirements above, a multi-scale A*(MSA*) search algorithm-based hybrid strategy is proposed in this paper to plan the trajectory and reserve airspace for the UTM flights. This hybrid strategy can run nearly in real-time due to a much smaller search space with the multi-scale framework. Therefore, this algorithm can respond fast to real-time incidence in a dynamic way. Meanwhile, the hybrid strategy can maintain a similar optimal level to the classic A* algorithm such that it can guarantee flight safety and reduce flight delays. The major contributions of this paper are summarized as follows:

- This paper developed a multi-scale A* based searching algorithm and applied it in the UTM. The customized MSA* has better adoption to corner cases and better time efficiency than the original A* algorithm.
- This paper proposed a hybrid strategy based on MSA* and A*. The hybrid strategy can take the advantage of both MSA* and A* and avoid problems caused by MSA* and A*. The experiences show that the hybrid strategy costs less computational time than A* while handling the corner cases MSA* can not handle.
- Based on the proposed hybrid strategy, this paper introduced a dynamic reservation strategy to replace the current static reservation strategy in the UTM to improve airspace utility efficiency. The experiments show that the dynamic reservation strategy greatly increases the efficiency of airspace utilization while completing the same tasks.

The paper's structure is as follows. Section II discusses related work; Section III explains how the airspace is modeled and how the UTM cycle works; Section IV explains the idea of A* and MSA* algorithm. Section V analyzes the problem with A* and MSA*, and introduces the hybrid search strategy; Section VI shows the simulation results by comparing the hybrid search strategy and baseline method; Section VII concludes this paper.

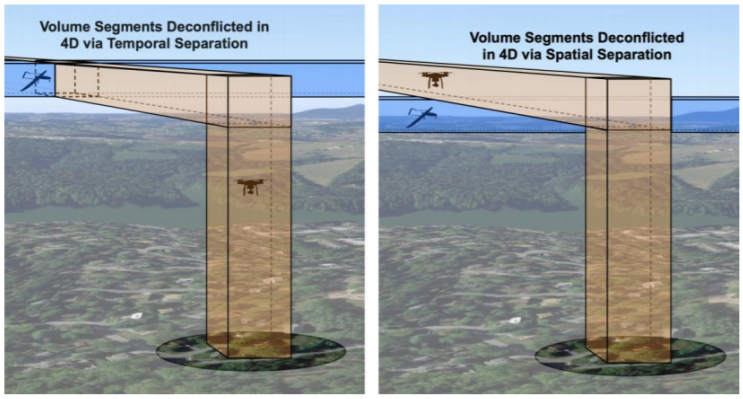


Fig. 1 Pre-departure Static Airspace Volume Reservation[3]

II. Literature Review

In this section, we will discuss prior related work on non-graph-based airspace reservations, graph-based search algorithm airspace reservations, fast/inaccurate search, robust planning, and learning-based trajectory planning for robots. The main challenge is to find the intersection of safety, low computational load, and high airspace utilization efficiency.

Right now, most airspace volume reservation framework in the current airspace system is static because the airspace capacity constraint is ignored and the current planning method is not fast enough for online replanning. Stevens, M. proposes geofence [6] to temporally and spatially organize the airspace. The concept of geofence has been widely utilized to reserve airspace volume. Zhu, G.[7] proposed a linear programming-based algorithm to generate and reserve conflict-free trajectory. The fastMDP-GPU algorithm can generate conflict-free flight plans with parallelized implementation [8]. A chance-constrained RRT planning algorithm is proposed to generate a path under location uncertainty[9]. Bertram, J, and Zambreno, J [8] employed the use of an algorithm known as FastMDP to assist in the de-conflicting of aircraft in a dense airspace environment and reserving the airspace for an aircraft prior to its departure. However, those methods can only be applied before the departure of the flight. Reinforcement learning [10] also is applied to search for the trajectory, however, it will take thousands of episodes to find the trajectory. As airspace becomes more and more crowded, a dynamic airspace volume reservation strategy that reserves much less airspace should be introduced to increase airspace efficiency [1].

The graph-based search method [11–16] is commonly used to search for the optimal path in many areas. Many papers model the airspace into a graph and generate the path with a graph-based search algorithm such as A*. Li, J. [17] proposed a graph-based algorithm for dynamic airspace configuration. However, this algorithm is designed for interstate airspace and hour-level duration. It will not work when participants in UTM are much close to each other. Conflict-Free A* (CFA*) [18] can generate paths to prevent a significant number of airspace conflicts. However, since CFA* searches in a four-dimensional graph, its search space is larger, and its computation load is worse than A*.

Long computation time is a major drawback of the optimal search algorithm. Local searches then are proposed to reduce the computation time. Zhang, N. [19] created a real-time urban environment that is divided into a 3D voxel world to run an efficient path-finding algorithm using Jump Point Search (JPS) [20]. JPS can greatly reduce the computing time for the path-finding algorithm by reducing the search points calculated in the 3D environment to form the optimal path while the A* algorithm computes more points that are ultimately ignored by the algorithm. A similar solution to reduce computing time while using the A* algorithm is introduced in Kambhampati and Davis [21]. A hierarchical refinement scheme of an environment's search space is abstracted in a quad-tree, where the risk region is given the coarsest resolution and excluded from the refinement to reduce the search space. This is done to obtain a multiscale version of the A* algorithm which allows for preprocessing of the environment in a top-to-bottom dyadic decomposition. This method divides the problem so that it can be solved at a much smaller and quicker rate and then be merged, thus creating a much faster and more efficient A* algorithm. A similar idea was enhanced by Lim,J. and Tsiotras, P. [22] by creating a multi-scale grid to process an optimal path across a risk map efficiently. Nevertheless, all of those methods can not guarantee collision-free or sacrifice optimality. Some corner cases exist for most of those local search methods, which may cause critical safety issues.

Recently, there are many robust trajectory planning introduced to navigate robots including unmanned airplanes. The sequential path planning method can generate paths for multiple participants under dynamic disturbance and imperfect knowledge [23]. Contraction theory and convex optimization also are applied to online motion planning for robotic systems with nonlinear dynamics [24]. FaSTrack is a modular proposed to fix errors caused by trajectory planners [25]. However, all of those methods can not reach enough robustness for UAS.

Besides the graph-based search method, learning-based methods are proposed to navigate the robot system and avoid a collision, which is a very similar system to UAS. BADGR is a self-supervised learning-based navigation system that can navigate robots in environments with geometrically distracting obstacles [26]. Julian, K used a deep neural network to approximate the giant decision table of Airborne Collision Avoidance System X(ACAS Xu) and it outputs the correct direction to flight much faster [27]. Although Learning-based methods are proven to accomplish certain tasks, their robustness still needs a lot of work to improve.

III. Problem Modeling

A. Grid World

In this paper, we model airspace as a dynamic grid world as shown in Fig. 2. The UTM system interacts with agents on a $m \times m$ grid graph G that is defined as follows:

$$G = \{g^s_{xy}\}_{m\times m} \quad where: x \in \{1, 2, ..., m\}, y \in \{1, 2, ..., m\}, and, s \in \{1, 2, 3, 4, 5\} \tag{1}$$

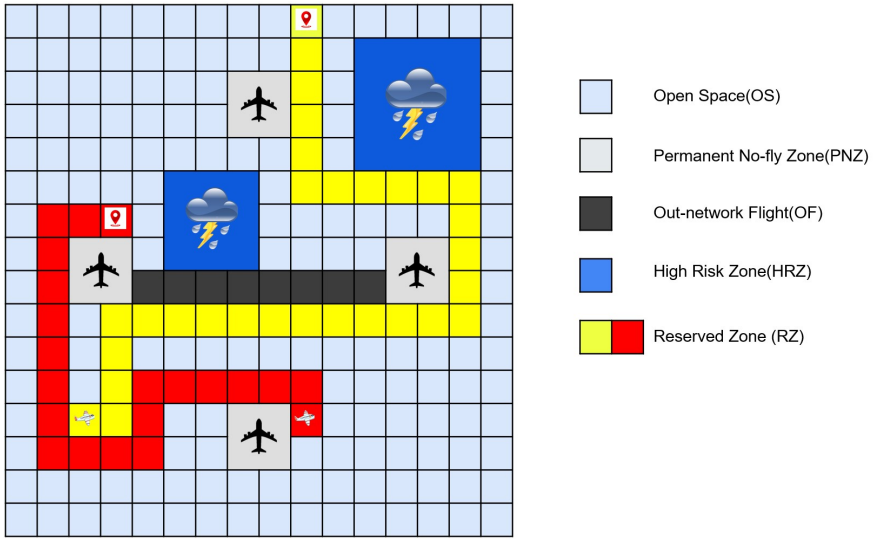


Fig. 2 Dynamic Grid World

In the dynamic graph, each grid cell g^s_{xy} represents a 1 unit × 1 unit area (the actual size of 1 unit is defined by the need of different missions), x, y represents the position of the grid in a Cartesian coordinate system, and s denotes the status of the grid cell. We assume that each cell g in the airspace has a risk level $R(g)$ depending on the situation in that area. We classify airspace into five different statuses and each of them has an associated risk value. The detailed definition is given as follows: 1) Open Space (OS): it represents grids that all the agents can pass freely, so the risk value of OS is 0; 2) Permanent No-fly Zone (PNZ): it represents grids that are permanently occupied, and agents strictly cannot pass them (e.g., airports). The risk value of PNZ is infinity; 3) Out-network Flight (OF): it represents grids that are temporarily occupied or reserved by flights that are not controlled by the system. The rule of agents passing those grids is the same as the PNZ. The risk value of OF is also infinity; 4) High-Risk Zone (HRZ): it represents grids that are temporarily impacted by weather, and agents must avoid them unless there is no other feasible way. The risk value of HRZ is 1; 5) Reserved Zone (RZ): it represents grids that all agents reserved. Like the HRZ, other agents will not pass the RZ unless it's necessary, so the risk value is also 1. To further distinguish the RZ reserved by different agents, we define a function $ID(g) = f_k$ to identify that the grid g is reserved by flight f_k. In summary, HRZ and RZ are soft obstacles, and the agent can still pass them when it is safe and there is no other feasible solution. On the other hand, PNZ and OF are hard obstacles. It is strictly applied to block flight agents. The risk value function for each grid is defined as follows:

$$R(g^s_{xy}) = \begin{cases} 0 & \text{for} \quad s = 1 \text{ (OS)} \\ \infty & \text{for} \quad s = 2 \text{ (PNZ) or } 3 \text{ (OF)} \\ 1 & \text{for} \quad s = 4 \text{ (HRZ) or } 5 \text{ (RZ)} \end{cases} \tag{2}$$

Like similar works [18], this paper assumes if a portion of a grid is occupied, the whole grid is occupied. The shape of obstacles or aircraft will not affect the occupancy. Each grid cell will only have one status. One flight agent occupies one grid cell at each time step, and it can only fly grid cell by grid cell in four directions, i.e., up, left, right, and down and the flight speed is one grid cell per unit of time.

B. Airspace Management Cycle for UTM

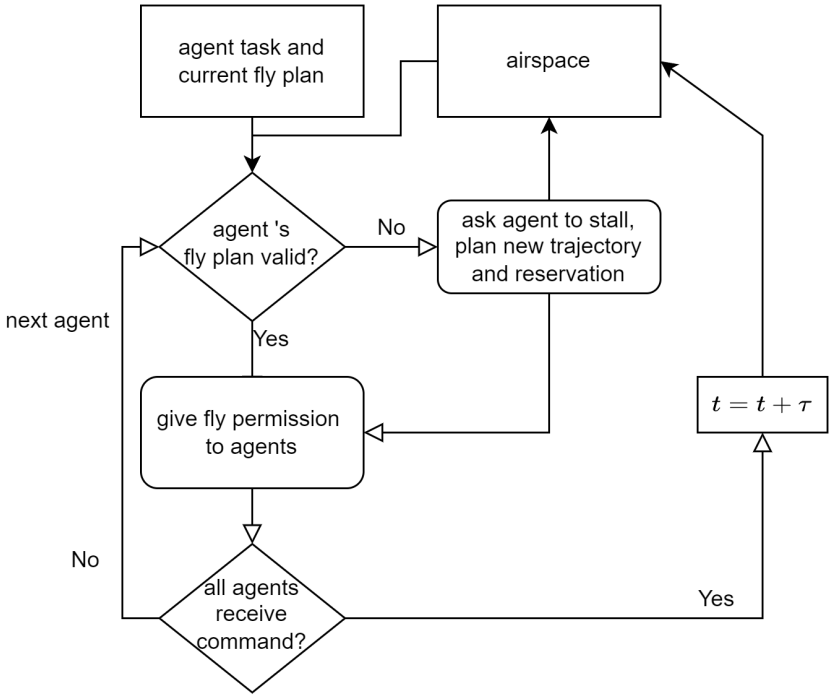


Fig. 3 Airspace Management Process

The UTM system will be responsible to navigate all agents to their corresponding destinations d_t. We consider the UTM system managing K flight agents, where each agent f_k is initialized with a flight task associated with an origin-destination (O-D) grid pair according to the real flight plan. The management cycle is summarized in Fig. 3. For each cycle, the agent f_k reports its task and the current flight plan to the UTM system. If the current flight plan is missing, the UTM system will generate a flight plan for the agent f_k with a search algorithm. Then, the UTM system updates the reservation for the agent f_k, and all the grids in the flight plan will become RZ. If the UTM system finds that the flight plan is not valid anymore, the UTM will reject the current flight plan and require re-planning. For safety reasons, the flight plan becomes invalid under two situations: 1) if the status of any grid in the flight plan changes to PNZ, HRZ, or OF in the new cycle; or 2) if the first grid of the flight plan is occupied by another agent. A valid flight plan T_k for flight agent f_k is defined as follows:

$$T_k = \{g^{(0)}, g^{(1)}, ..., g^i, g^{i+1}, ..., g^{(k_e)}\} \quad where : \forall g, s(g) \neq 2, 3, 4, and, ID(g^{(1)}) = f_k \tag{3}$$

where $g^{(0)}$ is the current position of the agent, and $g^{(1)}$ is the start of the planned trajectory, $g^{(k_e)}$ is the end of the planned trajectory and should be the destination, and any successive g^i and g^{i+1} must be neighbors on the graph G.

If the flight plan passes the evaluation, the agent will get permission to fly to the next planned grid, and the UTM will start to process the next agent. After every agent is iterated, this cycle is finished and the next cycle will begin. When a new cycle begins, the map and time t are updated. The HRZ and the OF will also be updated based on the current scenario. We assume there is a minimal processing time τ for the UTM system to update in each cycle because flights need time to move. So the new cycle will start at t + τ.

IV. Multi-Scale A*

A. Navigating by A*

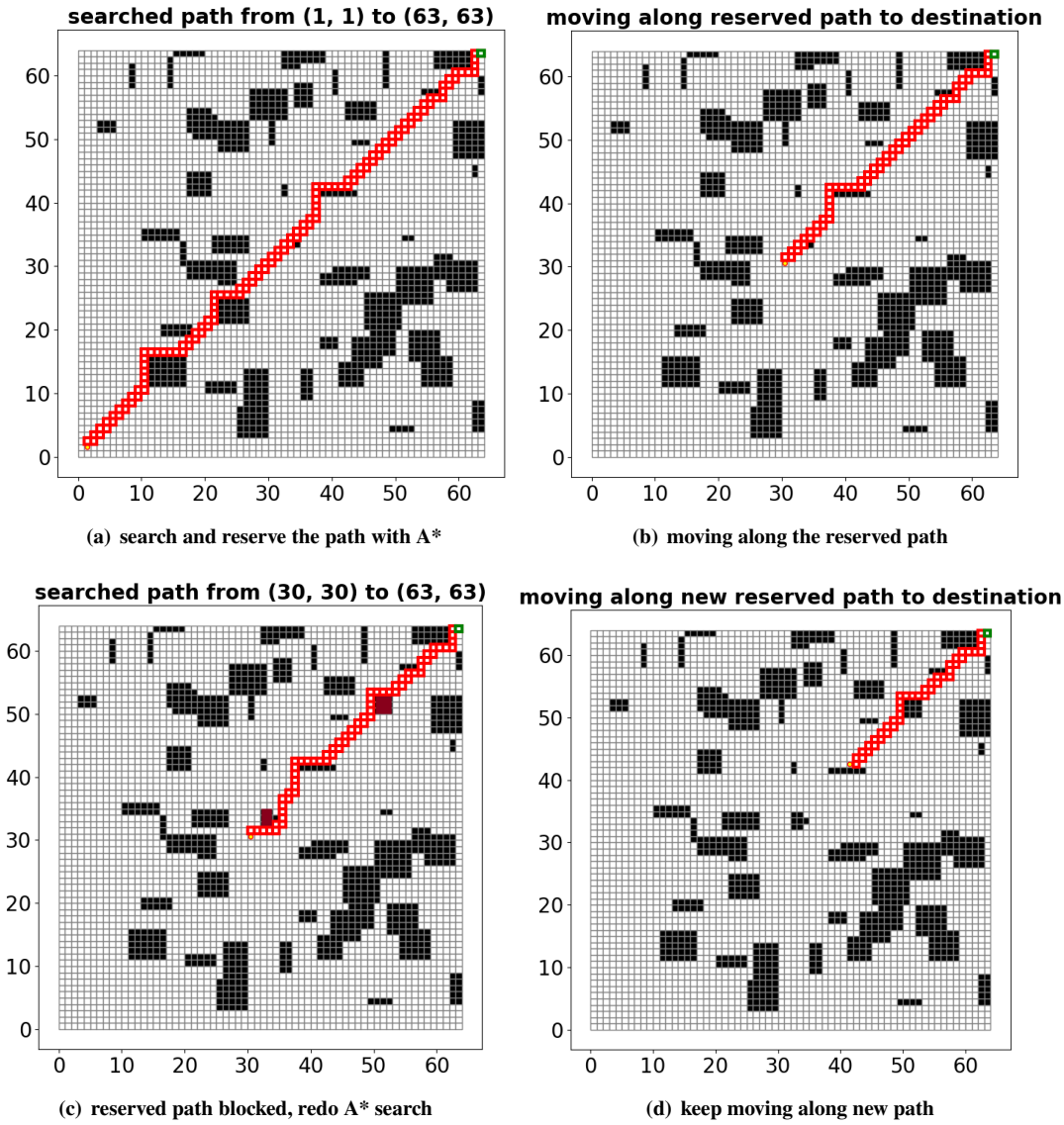
The A* algorithm is a widely used graph-based search strategy for finding traversable paths between nodes due to its performance and accuracy [12]. In this paper, the search space will include all the available grids (i.e., status=1, 4, 5) in the grid world and exclude all the blocked grids (i.e., status=2, 3). The edge cost of A* is the Euclidean distance between the center of grids. The heuristic function is Manhattan, which is equal to the sum of the distance between the current grid and the destination on two axes. Arrival cost c_a of the grid is applied to prevent the flight from passing risky grids. The arrival cost is based on the grid's risk value $R(g)$ (see equation (2)) and is given as follows:

$$c_a(g^s_{xy}) = \omega * R(g^s_{xy}), \forall s = \{1, 4, 5\} \tag{4}$$

where ω is a coefficient depending on the map size. Therefore, if the grid is high risk or reserved (i.e., s= 4, 5), the risk value of the grid will be counted as a cost to prevent agents from passing the grid. Fig. 4 shows an example of a flight agent being navigated with the A* in a 64×64 sample airspace. As shown in the figure, the A* algorithm finds a path that starts from the current position of the agent to the destination, and the UTM reserves the path for the agent. Then the agent moves along the reserved path. If the current path is blocked, the UTM must redo the search and provide another path. Although the A* algorithm can find the optimal solution, the computational complexity could be high, especially when the grid world is large [4].

B. Multi-resolution Graph

To improve computational efficiency, this paper develops a Multi-Scale A* (MSA*) method that is inspired by the multi-resolution framework in [22]. The idea of the multi-resolution graph is that a fine-resolution graph representation



(a) search and reserve the path with A*

(b) moving along the reserved path

(c) reserved path blocked, redo A* search

(d) keep moving along new path

Fig. 4 Agent Navigated by A*, where black grids are No-fly Zone, white grids are Open Space, red grids are Reserved Zone, brown grids are new No-fly Zone and the green grid is the destination.

is constructed only around the agents to focus the processing power on the locally optimal path it will take. Meanwhile, a coarse resolution representation is kept farther away from the agents to reduce the size of the search space, which greatly reduces the computational efforts needed to run the algorithm. For most popular grid search algorithms, the search depth and the search space are the main factors that affect the computational complexity. Multi-Scale A* can dramatically decrease the search depth and the search space, then lower the computational complexity.

Before the search, the grid map will be transformed into a multi-scale grid, which is constructed based on the agent's current location, destination, and map information. Therefore, the multi-scale grid map is varying for different agents and for the same agent at different locations. The purpose of using a multi-scale grid is that a lot more simulations can

be run at a quicker succession due to the fact that the MSA* can run in near real-time because of its smaller search space. This allows the collection of new environmental data more efficiently. In this work, the type of environmental data includes how the flight path of the UAS is affected as it changes its reserved airspace volume when encountering any dynamic or static obstacles, and the likelihood of collisions occurring as air traffic density increases.

In this paper, we abstract the full-resolution graph G as a full quadtree with node set $\mathbf{N} = \{n_{l,o}\}$. The quadtree node $n_{l,o} \in \mathbf{N}$ represents a grid set $G_n \subset G$, where $l \in \mathbf{Z}^+$ is the node level in a quadtree, $o(n) = (x, y)$ is the center position of the grid set the node $n_{l,o}$ represents. The size of the grid set represented by $n_{l,o}$ is $size(n_{l,o}) = 2^l \times 2^l$. The root node of the quadtree, which is the parent for all other nodes, represents all the grids in the grid world. Each leaf node ($l = 0$) represents all the information in a 1x1 grid(unit grid) such as position, status, and risk value. Each non-leaf node($l > 0$) represents its 4 direct children at level $l - 1$, and the risk value will be the average of its 4 direct children. Each child represents 1/4 of the grid set from its direct parent. For example, in Fig. 5(c), the red node represents four unit grids and the green node only represents one unit grid.

In practice, some nodes in the full quadtree are not necessary. For agent k at the time i, we create a partial tree for search based on the current position of the agent and current map information. The partial tree builds a non-empty multi-resolution graph $G_k(\mathbf{N}_i, \mathbf{E}_i)$ where $\mathbf{N}_i$ is the set of the selected nodes and $\mathbf{E}_i$ is the corresponding edge set. The multi-resolution graph $G_k(\mathbf{N}_i, \mathbf{E}_i)$ represents the same airspace as the full-resolution G. The current agent selects nodes following the hierarchical system and goes down the set nodes in each depth layer. When a node with children is selected by the agent, then its children node and parent node will be excluded from $\mathbf{N}_i$. This ensures that the selected node in $\mathbf{N}_i$ completely covers the grid world G without overlap. $\mathbf{N}_i$ is a subset of the full node set $\mathbf{N}$. The node $n_{l,o}$ will be selected if the Euclidean distance between the center point o of the node, and the current location p of the agent, is smaller than the $\alpha * 2^l$, where α is a parameter set by the user. Otherwise, its children will be selected instead. The larger the α is, the more nodes will be selected.

$$||o(n) - p|| < \alpha * 2^l \tag{5}$$

Meanwhile, some non-leaf nodes may be too crowded and the possibility of the agent must pass those nodes are small. so, those nodes can also be excluded from $\mathbf{N}_i$. In this paper, all the nodes whose average capacity rate (CR) is less than β will be excluded unless their level is higher than 3.

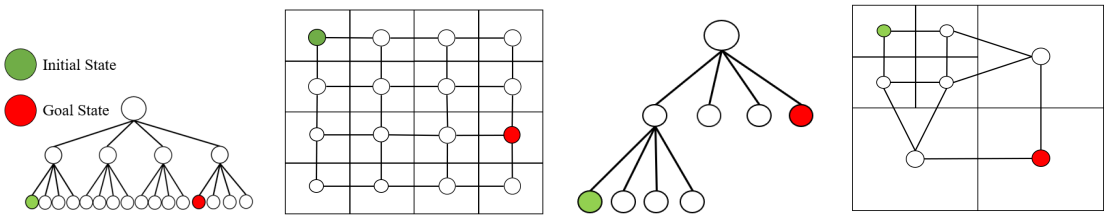
$$\mathbf{N}_i =: \{\mathbf{N} | CR(n) > \beta \quad \text{or} \quad l(n) > 3\} \tag{6}$$

where CR is the percentage of the Open Space grid in the represented grid sets of node n, and $l(n)$ is the level of node n. After nodes are selected, the edge will be defined by the distance between nodes. We consider there is an edge if the

Euclidean distance between any two nodes (e.g., n_i and n_j) is smaller than the average size of those two nodes.

$$\mathbf{E}_i =: \{E| \quad ||o(n_i) - o(n_j)|| < 0.5 * (size(n_i) + size(n_j)), \quad \forall n_i, n_j \in \mathbf{N}_i\} \tag{7}$$

where E is all the possible edges connected between nodes in $\mathbf{N}$.



(a) full quadtree **(b) full-resolution map** **(c) partial tree** **(d) multi-resolution map**

Fig. 5 Illustration of The Multi-scale Graph Construction

In Fig. 5, the formulation for multi-scale graph $G_k(\mathbf{N}_i, \mathbf{E}_i)$ is illustrated: a) The grid-based world encoded in a full quadtree, which shows the parent nodes and children nodes with the initial state, shown as the green node, and the goal state, shown as the red node. b) From the full tree, the full-resolution graph G is constructed using all the leaf nodes from the created tree. c) $\mathbf{N}_i$ are selected from the full tree based on the rule introduced above. d) The new graph $G_k(\mathbf{N}_i, \mathbf{E}_i)$ is now created where the initial state still locates the grid represented by the leaf node (shown as the green node), while some other grids are represented by the higher-level nodes, such as the grid the goal state (shown as the red node) located at.

C. Cost Function

In contrast with A*, the MSA* will do the search on the multi-resolution graph. The heuristic function is still Manhattan. For safety reasons, we want agents to avoid crowded areas. The base edge cost function is the Euclidean distance between the center of nodes. An additional edge cost is applied if a node has less open space. The cost is high enough to force agents to avoid the crowded grids unless no other path is available. The total edge cost to reach a node is defined as

$$e_{msa*}(n_i, n_j) = Ed(n_i, n_j) * 1/(CR(n_j)), \quad \forall n_i, n_j \in \mathbf{N}_i \tag{8}$$

where e_{msa*} is the edge cost between any nodes n_i and n_j, Ed is the Euclidean distance, CR(n_j) is the capacity rate of the node n_j, which is equal to the ratio of the open space in the node n_j.

D. Adaptive Parameter Controller

The size of search space and search depth highly rely on α and β in equations (5) and (6). If the α is larger, the results of the search algorithm will be much more reliable because the graph is more accurate. However, the search will

cost much more time as α increases. If β is larger, more crowded nodes are dropped, then the agent has a larger chance to avoid a crowded area. However, the feasible search space will be much smaller, which may lead to infeasibility. To balance the time cost and reliability of the results, adaptive parameters are introduced, which is shown in Algorithm 1. The adaptive parameter controller (APC) will modify the parameters when it finds that the previous trajectory is a dead loop(agents keep passing the same grids or wandering around an area). If a dead loop is detected, the algorithm will increase α temporarily to jump out of the loop. In this paper, the dead loop is defined by two situations: 1) the agent revisits the previously visited position for 4 times; 2) the agent returns to the previous position. The APC has three parameters, $d\alpha$ and $d\beta$ are the basic increments in each adjustment, and x_r controls how often the α and β are decremented.

Algorithm 1: ADAPTIVE PARAMETER CONTROLLER

if *a dead loop is detected* **then**
    $\alpha \leftarrow \alpha$ + dα
    $\beta \leftarrow \beta$ + dβ
if *no dead loop for x_r iterations* **then**
    $\alpha \leftarrow \alpha$ - dα
    $\beta \leftarrow \beta$ - dβ

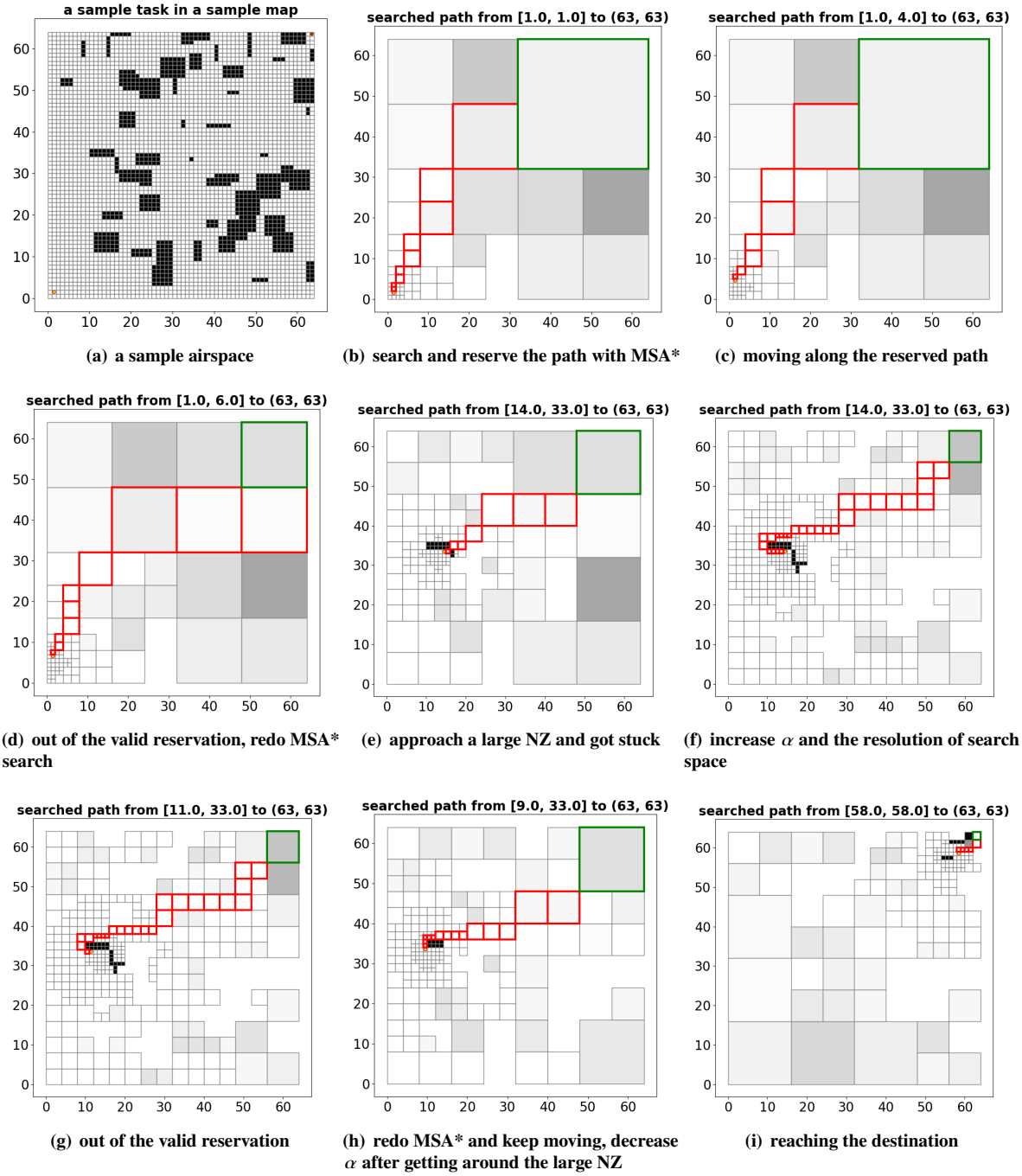
E. Navigating by MSA*

Fig. 6 shows an example of an agent being navigated by MSA*. Each cell represents a node in the search space, the darker the node is, the percentage of OS in the node is lower. Some nodes are dropped from the search space because they are far away from the agent and have a low percentage of OS. MSA* can quickly find an approximate path on the multi-resolution graph as Fig. 6(b). Among all cells in the path, only those with size 1×1 are reserved because only the grids of airspace with the highest resolution can be directly used in practice according to Section III. After the path is planned and reserved, the agent will move along the reserved node as Figure. 6(c). After all the reserved nodes are passed, the UTM system will redo the search with the new location and the updated map as Fig. 6(d). It will repeat this process until it researches the destination. Fig. 6(e) shows an example of APC stepping in when the agent approached a large No-fly Zone (NZ) and MSA* can not find a way to pass around the NZ. The APC increases the α so that the search space is more accurate, then it is feasible to find a path around the NZ.

V. Hybrid A* Search Strategy

A. Time and Optimality Analysis

According to [4], the computational complexity of A* is $O(b^e d)$, where b is the branching factor of the graph, e is the error, and d is the solution depth. The computational complexity increases as the heuristic value are more inaccurate



(a) a sample airspace **(b) search and reserve the path with MSA*** **(c) moving along the reserved path**

(d) out of the valid reservation, redo MSA* search **(e) approach a large NZ and got stuck** **(f) increase α and the resolution of search space**

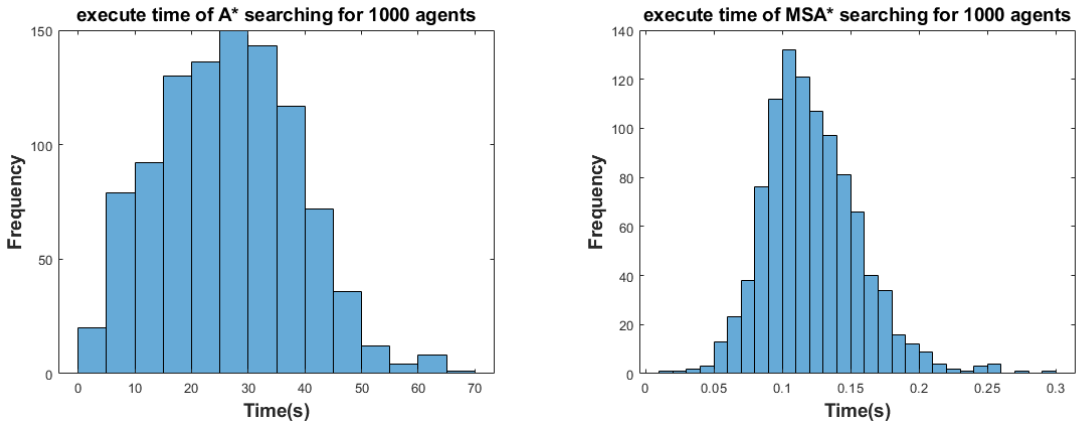
(g) out of the valid reservation **(h) redo MSA* and keep moving, decrease α after getting around the large NZ** **(i) reaching the destination**

Fig. 6 Agent Navigated by MSA*, where the destination is framed by green and the planned path is framed by red.

and solution depth increases. In this paper, we use the same heuristic value for both searches and the error is not the interest factor we try to minimize. Because of the smaller search space and special node selection strategy, the solution depth of MSA* is very small compared to A*. The computational complexity of edge and node selection is small enough as O(N), so it can be ignored. When the solution depth is large, the computational complexity of MSA* with α

smaller than 3, will be much smaller than that of A* [28]. Fig. 7(a) shows the computing time distribution of searching a path with A* for 1000 different tasks in the 512*512 simulated grid map. In almost all cases, A* will cost more than one second, which may delay the online updating cycle. In contrast, when we search the path with MSA* for the same tasks, all of the tasks cost less than 0.5 seconds, as shown in Fig. 7(b). Meanwhile, since A* search the path all the way from the current position to the destination, A* reserve more airspace. Although airspace usually is reserved with a time window, that information increases the complexity of the scenario and the difficulty and computing complexity of the search algorithm.

However, MSA* cannot always find the optimal path. In some cases, MSA* may find a path that's much longer than the optimal path or stuck in some places. For example, in Fig. 6(e), the MAS* navigates the flight agent into a dead-end and is unaware of it until the No-fly Zone is very close because the dead-end is not shown in the multi-scale graph. The MAS* needs to increase the size of the search space to find a way out. Nevertheless, if the search space is too large, the computational complexity advantage will disappear. Besides theoretical analysis, experiences in section VI also prove A* has high computational complexity and MSA* has more infeasible situations.



(a) A* computational time of 1000 random tasks **(b) MSA* computational time of 1000 random tasks**

Fig. 7 Computational Time Distribution

B. Hybrid A* Search Strategy

To overcome the limitations of applying any individual algorithm with either A* or MSA*, a hybrid A* search strategy is proposed to choose the more suitable search algorithm in a dynamic environment. The search algorithm should first be decided by the Euclidean distance between the O-D pair. Because search depth is at least proportional to the Euclidean distance between the O-D pair, MSA* will be preferred when the Euclidean distance is large. Otherwise, A* has the advantage. When the search algorithm is MSA* with APC, if APC increases the α too much, the search space size and search depth of MSA* will be close to the A*, then the search algorithm should switch to A*. On the other hand, if A* is too slow and delays the online updating, the search algorithm should switch to MSA*.

The hybrid A* search strategy is summarized as Algorithm 2. Search algorithm indicator s indicates the current search algorithm, where s is initialized as A*. There are three adjustable parameters, d_h, a_h, and t_h in the hybrid A* strategy. d_h is a threshold distance, if the Euclidean distance between the current node p and destination node d_t is larger than d_h, MSA* is chosen, otherwise, A* is chosen. a_h is the upper threshold for α of MSA*, the search algorithm should switch to A* if α is larger than a_h. t_h is the maximum allowed computational time, the search algorithm will switch to MSA* if the computational time of A* exceeds t_h.

Algorithm 2: HYBRID A* SEARCH STRATEGY

```
Input: current position p, destination d_t
Parameter: threshold distance d_h, upper threshold a_h, allowed computational time t_h
while p ≠ d_t do
    if s = MSA* then
        adaptive parameter control
        MSA* search and move
        if Ed(p, d_t) < d_h or α > a_h then
            s ← A*
    if s = A* then
        A* search and move
        if search time > t_h then
            s ← MSA*
```

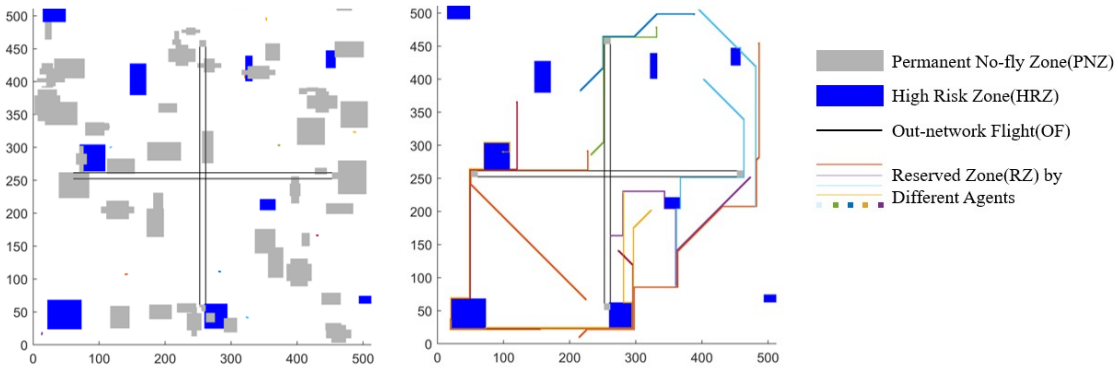
VI. Results

In this section, a series of numerical studies are conducted to demonstrate the feasibility and efficiency of the proposed hybrid strategy in the UTM system.

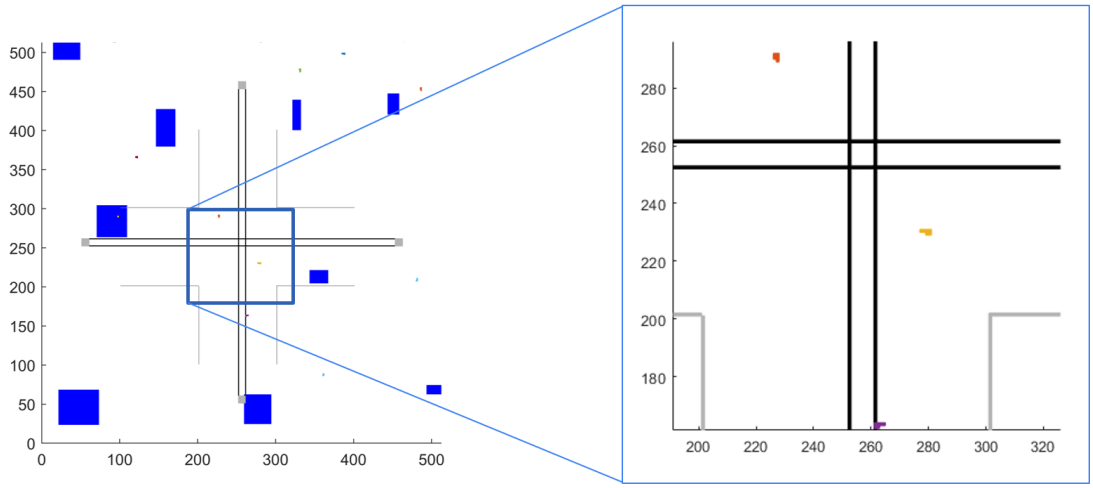
A. Experiment Setting

To evaluate the proposed search strategy, A*, MSA*, and the hybrid A* search strategy will be separately tested in a simulated dynamic flight environmental grid map with size 512*512 as shown in Fig. 8. The strategy will be first tested in airspace containing a certain amount of random obstacles (Fig. 8(a)). Then, there are two different designed airspaces, one is easy to represent a common flight situation (Fig. 8(b)) and one is hard to represent stress flight situations (Fig. 8(c)). In Fig. 8, the gray area is the PNZ. The Out-network Flight, which is shown as the black line area, represents the service flight route with a fixed schedule, similar to the bus route on the ground. Those areas will be blocked following a fixed schedule. Moreover, some HRZs, shown as blue rectangle areas, represent areas with adverse weather conditions. To make simulations more realistic, all HRZs are randomly generated on the map. In specific, a random number (ranges from 0 to 12) of HRZs are generated with arbitrary locations on the map. Each HRZ is a rectangle with a random size with the side length ranging from 10 to 50, and the duration ranges from 15 to 45-time steps. In the hard

map, some static obstacles (shown as gray PNZs) are intentionally placed to increase the difficulty for A* and MSA* but also form a situation that occasionally happens in the real world, such as areas with high buildings.



(a) MSA* testing in a random map **(b) A* testing in an easy map**



(c) MSA* testing in a hard map

Fig. 8 Agents under UTM moving in the airspace, where gray lines and areas are Permanent No-fly Zone, blue areas are High-Risk Zone, black lines are the areas reserved by out-network flight, colorful lines, and squares are agents and their reservation

In this section, various numbers of flight tasks are generated randomly with at least 200 grids away from each other. The UTM system introduced in section III will navigate the agents to complete the generated tasks. The current position of a flight agent is shown as a small colored rectangle. The colored rectangle is larger than the actual grid the flight agent occupies for display purposes. The path reservation of that agent is shown as the same color (as shown in Fig. 8(b)). The reservation by A* starts from the current position all the way to the destination. In contrast, the reservation of MSA* only takes a few grids as shown in the zoomed area of Fig. 8(c).

The search strategy is evaluated by the task success rate, the total process time, and the ratio of flight distance to the Euclidean distance between the task origin and destination of all agents. If an agent can not complete its task in 2500

iterations, the agent fails to complete the task. Because the search algorithm can not always finish the path planning task within the minimum airspace update interval τ, there will be delays for the UTM cycle. To measure the delays due to slow computational time, the experiment will consider a processing time dt_j as

$$dt_j = \max(\tau, comt_j) \tag{9}$$

where dt_j is the actual processing time for the j th cycle, $comt_j$ is the computing time of the j th cycle. Because of possible delay caused by high computing complexity, dt could be larger than the regular updating time τ. so the total process time is the sum of dt_j in this section.

All experiments were done at least 10 times with a different set of tasks. All the tests in the designed airspace were implemented in Python 3.8 and were run on an Intel CPU i7-8700k @3.70GHz with 16GB RAM. All the tests in the random airspace were run on an AMD Ryzen 9 3900X 12-Core Processor and 32GB RAM. The parameter of MSA* with APC is shown as table 2.

Table 2 Experience Parameter

parameter	value
ω	1000
α	2.0
β	0.75
$d\alpha$	0.25
$d\beta$	0.25
x_r	15
d_h	30
a_h	2.5
t_h	5s
τ	1s

B. Reliability

Table 3 shows all three methods can successfully navigate the agents to the destination in the easy map despite MSA* taking limited map information. However, MSA* performs badly in the stress test on the hard map. After 2500 iterations, MSA* can only finish around 64% of tasks. In the random map, the reliability of MSA* is also not guaranteed. The success rate is lower than 98%. This result supports the previous feasible analysis in Section V. Basically, MSA* is a local search algorithm in which only partial information is considered as a multi-scale graph, there are many dead ends caused by very few obstacles that were not shown in the multi-scale graph. Therefore, it may navigate the flight agent into a dead-end and is unaware of it until the No-fly Zone is very close. Our hard map is specially designed to include many of those dead ends to demonstrate the performance of the MSA* algorithm under those corner cases, therefore the

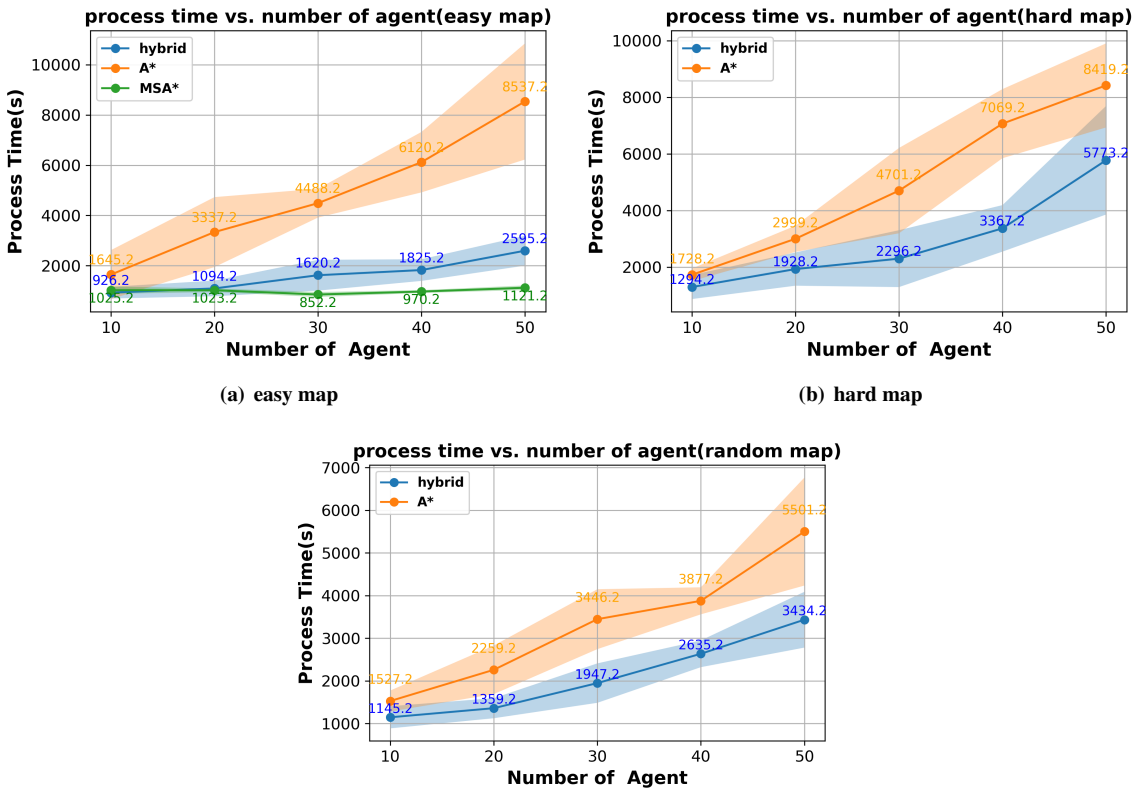
Table 3 Successful Rate

map	number of agents	A*	MSA*	hybrid strategy
random	10	100%	98%	100%
	20	100%	97.5%	100%
	30	100%	96.7%	100%
	40	100%	96%	100%
	50	100%	95.6%	100%
easy	10	100%	100%	100%
	20	100%	100%	100%
	30	100%	100%	100%
	40	100%	100%	100%
	50	100%	100%	100%
hard	10	100%	64.6%	100%
	20	100%	64.6%	100%
	30	100%	64.2%	100%
	40	100%	64.1%	100%
	50	100%	63.7%	100%

drop in the success rate is significant. However, as shown in the random map, those corner cases are only a very small percentage of all cases. In contrast, both A* and hybrid searching can complete the tasks within 2500 iterations and pass the stress test. The result proves that the hybrid search strategy can handle all corner cases that MSA* can not handle.

C. Time Efficiency

As shown in Fig. 9, in all maps, A* costs the most process time and causes the most delays while MSA* seldom causes delays. Meanwhile, the processing time of A* increases dramatically as the increase of the number of agents. The process time difference will be larger if there are more agents in the UTM. The results respond to the previous computational complexity analysis that the A* algorithm delays the UTM cycle most time. The proposed hybrid search strategy also has a longer process time than MSA* but shorter than A*. In the easy map, When the number of agents is 10, the hybrid strategy only takes 57% of the processing time of A*. With the number of agents increasing, the hybrid strategy saves more process time. When the number of agents reaches 50, the hybrid strategy takes only 30 % of the processing time of A*. In the hard map, the difference between the processing time of hybrid and A* is smaller, though, hybrid only costs around the half time A* costs when the number of agents is larger than 10. In the random map, the hybrid also saves 40% of time compared to the A*. Note that the results of MSA* are not shown here because not all MSA* cases can be finished on the hard map and random map as shown in Table 3. The experience proves that the hybrid strategy has much better time efficiency than A* and can reduce delay. Therefore, the hybrid strategy is suitable for online planning with a dynamic short reservation, while the A* has to reserve a static long reservation due to the slow updating rate.



(a) easy map

(b) hard map

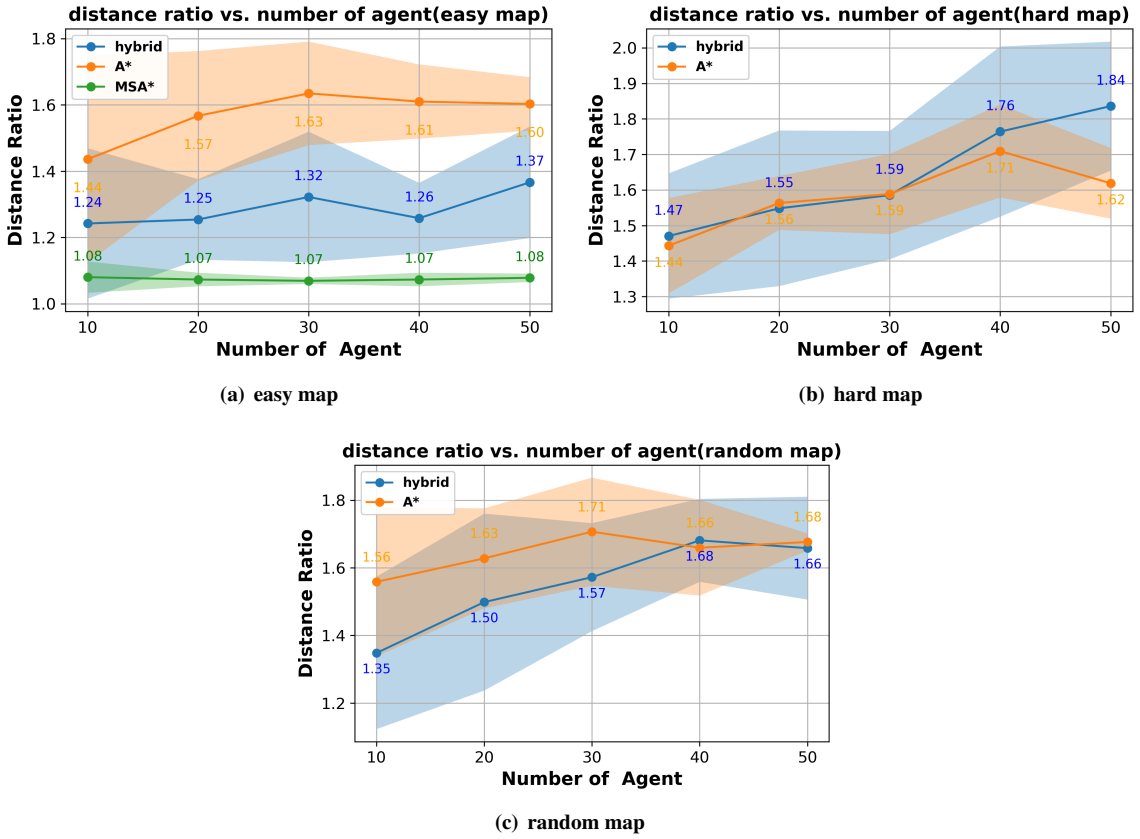
(c) random map

Fig. 9 Process Time Comparison

D. Route Efficiency

As shown in Fig. 10(a), when the map is easy, the distance ratio (DR) of A* is the largest, and the hybrid has a lower distance ratio than A* but higher than MSA*. Because all the test tasks of the same number of agents are the same for each search strategy, the average flight length of agents navigating with A* is the longest due to congestion which proves that MSA* based methods have better route efficiency than A*. The agent navigated by the A* reserves much more airspace than the agent navigated by the MSA* in the same iteration of the UTM cycle. Those extra reservations may force other agents to take a detour. At the same time, the hybrid strategy has a lower distance ratio than A*. It proves that the hybrid strategy can lower the average travel distance of the agents and use airspace more efficiently than A* in the easy map. Meanwhile, MSA* has a low variance. In contrast, A* is very sensitive to the locations of the O-D pair. Therefore, in most regular cases, as the easy map shows, the performance of the MSA*-based method is more reliable. In the hard map, as shown in Fig. 10(b), the DR result of A* and hybrid is very close. As mentioned in section V.A, there are many dead ends caused by very few obstacles that were not shown in the multi-scale graph in the hard map, and the agent guided by the MSA* can not avoid those dead ends and get stuck. Therefore MSA* performs worse than A* in hard maps and some random maps with corner cases. According to the result from Table 3, the MSA* can not

even finish the task in many cases and most agents under the hybrid strategy will switch to A* eventually. This is the main reason we need a hybrid strategy when MSA* is extremely fast and efficient in the easy map. Hybrid is a strategy that has good time efficiency and a low distance ratio while guaranteeing finishing all the tasks. However, in those hard cases, the A* may have a lower distance ratio. The reason is that the agent with a hybrid strategy tries MSA* first and makes some incorrect moves, then switches to A* after finding out MSA* is not feasible. In the random map, as shown in Fig. 10(c), when the number of agents is lower than 30, the DR of the hybrid strategy is better than A*. However, when the number of agents is larger, the DR of the A* and hybrid is very close.



(a) easy map

(b) hard map

(c) random map

Fig. 10 Ratio Between Travel Distance and Euclidean Distance

E. Discussion

Although MSA* performs very well in the easy map, there are many corner cases MSA* fails to complete the task. We designed a map that is very difficult for the MSA* in purpose, which may not often be seen in practice. The most difficult part is that some obstacles completely block some tunnels but occupy very few grids. For MSA*, the search agent can not be aware of those obstacles until the flight approaches them. MSA* also can not finish all the tasks in the random map. On the other hand, A* can successfully finish all the tasks because it takes all the information and

searches for the optimal solution. However, A* requires a large computation capacity and can not plan the path in a short time. The agents must wait an unacceptable time for A*. Both MSA* and A* have disadvantages and advantages. The proposed hybrid search strategy can switch the search algorithm automatically based on the scenario and avoid the disadvantages of the algorithm. According to our experiences, the proposed hybrid search strategy has a better time and route efficiency than A* while having better reliability than MSA*. The proposed search strategy is a more suitable algorithm than the two previous pure search algorithms in the UTM system.

VII. Conclusion

This paper introduces a dynamic trajectory planning and traffic volume reservation method for the drone traffic management system based on a hybrid multi-scale A* algorithm. The major advantage of the multi-scale A* algorithm is the ability to effectively decrease the size of the search space for the A* algorithm while keeping essential map information. To overcome the corner cases of the multi-scale A*, we propose a hybrid strategy that combines full-resolution A* to help agents jump out of the dead-end. This hybrid planning strategy is shown to be significantly faster than pure A* planning while maintaining a 100% success rate in various test cases. The dynamic trajectory planning and traffic volume reservation method based on this hybrid planning strategy also helps improve route efficiency in our comprehensive experiments. The hybrid planning strategy with high reliability and time efficiency will be beneficial to high-dense drone traffic management.

Acknowledgment

We would like to thank the National Science Foundation (NSF) under Grants CMMI-2138612 for the support of this work. Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not reflect the views of NSF.

References

[1] Thipphavong, D. P., Apaza, R., Barmore, B., Battiste, V., Burian, B., Dao, Q., Feary, M., Go, S., Goodrich, K. H., Homola, J., et al., "Urban air mobility airspace integration concepts and considerations," *2018 Aviation Technology, Integration, and Operations Conference*, 2018, p. 3676. https://doi.org/10.2514/6.2018-3676.

[2] Musavi, N., Onural, D., Gunes, K., and Yildiz, Y., "Unmanned aircraft systems airspace integration: A game theoretical framework for concept evaluations," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 1, 2017, pp. 96–109. https://doi.org/10.2514/1.G000426.

[3] Kopardekar, P., Rios, J., Prevot, T., Johnson, M., Jung, J., and Robinson, J. E., "Unmanned aircraft system traffic management (UTM) concept of operations," *AIAA Aviation and Aeronautics Forum (Aviation 2016)*, 2016.

[4] Russell, S. J., *Artificial intelligence a modern approach*, 3rd ed., Pearson Education, Inc., Hoboken, NJ, 2010, pp. 93–98.

[5] Balakrishnan, H., "Control and optimization algorithms for air transportation systems," *Annual Reviews in Control*, Vol. 41, 2016, pp. 39–46. https://doi.org/10.1016/j.arcontrol.2016.04.019.

[6] Stevens, M., and Atkins, E., "Geofence definition and deconfliction for UAS traffic management," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 22, No. 9, 2020, pp. 5880–5889. https://doi.org/10.1109/TITS.2020.3040595.

[7] Zhu, G., and Wei, P., "Pre-departure planning for urban air mobility flights with dynamic airspace reservation," *AIAA Aviation 2019 Forum*, 2019, p. 3519. https://doi.org/10.2514/6.2019-3519.

[8] Bertram, J., Wei, P., and Zambreno, J., "Scalable FastMDP for Pre-departure Airspace Reservation and Strategic De-conflict," *AIAA Scitech 2021 Forum*, 2021, p. 0779. https://doi.org/10.2514/6.2021-0779.

[9] Wu, P., Xie, J., and Chen, J., "Safe path planning for unmanned aerial vehicle under location uncertainty," *2020 IEEE 16th International Conference on Control & Automation (ICCA)*, IEEE, 2020, pp. 342–347. https://doi.org/10.1109/ICCA51439.2020.9264542.

[10] Hu, J., and Liu, Y., "Uas conflict resolution integrating a risk-based operational safety bound as airspace reservation with reinforcement learning," *AIAA Scitech 2020 Forum*, 2020, p. 1372. https://doi.org/10.2514/6.2020-1372.

[11] Dijkstra, E. W., "A note on two problems in connexion with graphs," *Numerische mathematik*, Vol. 1, No. 1, 1959, pp. 269–271. https://doi.org/10.1007/BF01386390.

[12] Hart, P. E., Nilsson, N. J., and Raphael, B., "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, 1968, pp. 100–107. https://doi.org/10.1109/TSSC.1968.300136.

[13] LaValle, S. M., and Kuffner, J. J., *Rapidly-Exploring Random Trees: Progress and Prospects*, AK Peters/CRC Press, Natick, MA, 2001, pp. 303–307.

[14] Yang, X., and Wei, P., "Scalable multi-agent computational guidance with separation assurance for autonomous urban air mobility," *Journal of Guidance, Control, and Dynamics*, Vol. 43, No. 8, 2020, pp. 1473–1486. https://doi.org/10.2514/1.G005000.

[15] Liu, Z., Zhang, J., Zhu, Y., and Cai, K., "Hierarchical Four-Dimensional Trajectories Planning Method for Manned and Unmanned Aircraft Integrated Airspace," *Journal of Guidance, Control, and Dynamics*, Vol. 45, No. 6, 2022, pp. 1017–1032. https://doi.org/10.2514/1.G006206.

[16] Xiang, J., Amaya, V., and Chen, J., "Dynamic Unmanned Aircraft System Traffic Volume Reservation Based on Multi-Scale A* Algorithm," *AIAA SCITECH 2022 Forum*, 2022, p. 2236. https://doi.org/10.2514/6.2022-2236.

[17] Li, J., Wang, T., Savai, M., and Hwang, I., "Graph-based algorithm for dynamic airspace configuration," *Journal of guidance, control, and dynamics*, Vol. 33, No. 4, 2010, pp. 1082–1094. https://doi.org/10.2514/1.47720.

[18] Dai, W., Pang, B., and Low, K. H., "Conflict-free four-dimensional path planning for urban air mobility considering airspace occupancy," *Aerospace Science and Technology*, Vol. 119, 2021, p. 107154. https://doi.org/10.1016/j.ast.2021.107154.

[19] Zhang, N., Zhang, M., and Low, K. H., "3D path planning and real-time collision resolution of multirotor drone operations in complex urban low-altitude airspace," *Transportation Research Part C: Emerging Technologies*, Vol. 129, 2021, p. 103123. https://doi.org/10.1016/j.trc.2021.103123.

[20] Ma, L., Gao, X., Fu, Y., and Ma, D., "An improved jump point search algorithm for home service robot path planning," *2019 Chinese Control And Decision Conference (CCDC)*, IEEE, 2019, pp. 2477–2482. https://doi.org/10.1109/CCDC.2019.8833422.

[21] Kambhampati, S., and Davis, L., "Multiresolution path planning for mobile robots," *IEEE Journal on Robotics and Automation*, Vol. 2, No. 3, 1986, pp. 135–145. https://doi.org/10.1109/JRA.1986.1087051.

[22] Lim, J., and Tsiotras, P., "MAMS-A*: Multi-Agent Multi-Scale A," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 5583–5589. https://doi.org/10.1109/ICRA40945.2020.9197045.

[23] Bansal, S., Chen, M., Fisac, J. F., and Tomlin, C. J., "Safe sequential path planning under disturbances and imperfect information," *2017 American Control Conference (ACC)*, IEEE, 2017, pp. 5550–5555. https://doi.org/10.23919/ACC.2017.7963818.

[24] Singh, S., Majumdar, A., Slotine, J.-J., and Pavone, M., "Robust online motion planning via contraction theory and convex optimization," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 5883–5890. https://doi.org/10.1109/ICRA.2017.7989693.

[25] Herbert, S. L., Chen, M., Han, S., Bansal, S., Fisac, J. F., and Tomlin, C. J., "FaSTrack: A modular framework for fast and guaranteed safe motion planning," *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, IEEE, 2017, pp. 1517–1522. https://doi.org/10.1109/CDC.2017.8263867.

[26] Kahn, G., Abbeel, P., and Levine, S., "Badgr: An autonomous self-supervised learning-based navigation system," *IEEE Robotics and Automation Letters*, Vol. 6, No. 2, 2021, pp. 1312–1319. https://doi.org/10.1109/LRA.2021.3057023.

[27] Julian, K. D., Kochenderfer, M. J., and Owen, M. P., "Deep neural network compression for aircraft collision avoidance systems," *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 3, 2019, pp. 598–608. https://doi.org/10.2514/1.G003724.

[28] Hauer, F., Kundu, A., Rehg, J. M., and Tsiotras, P., "Multi-scale perception and path planning on probabilistic obstacle maps," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 4210–4215. https://doi.org/10.1109/ICRA.2015.7139779.