Trajectory planning for UAV navigation in dynamic environments with matrix alignment Dijkstra

Jinyang Wang · Yuhua Li · Ruixuan Li · Hao Chen · Kejing Chu

Received: 23 May 2022 / Published online: 25 June 2022 © The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2022

Abstract

The trajectory planning for Unmanned aerial vehicles (UAVs) in the dynamic environments is a challenging task. Many restrictions should be taken into consideration, including dynamic terrain collision, no-fly zone criteria, power and fuel criteria and so on. However, some methods treat dynamic restrictions as static in order to reduce cost and obtain efficient and acceptable paths. To achieve optimal, efficient and acceptable paths, we first use a high-dimension matrix, extended hierarchical graph, to model the unexplored dynamic grid environment and to convert weather conditions to passable coefficient. The original shortest path planning task is then translated to plan the safest path. Therefore, we propose a new forward exploration and backward navigation algorithm, matrix alignment Dijkstra (MAD), to pilot UAVs. We use matrix alignment operation to simulate the parallel exploration process of all cells from time $t$ to $t + 1$. This exploration process can be accelerated on a GPU. Finally, we recall the optimal path according to the navigator matrix. In addition, we can achieve UAV path planning from one source cell to multiple target cells within one single run. We validate our method on a real dynamic weather dataset and get competitive results in both efficiency and accuracy. We analyse the performance of MAD on a grid-based benchmark dataset and artificial maze data. Simulated results show MAD is especially helpful when the grid-based environment is large-scale and dynamic.

Keywords

Trajectory planning · Unmanned aerial vehicles · Matrix alignment Dijkstra · GPU acceleration

1 Introduction

Unmanned aerial vehicles (UAVs) are gradually becoming part of everyday work and life. UAVs are used in a wide range of outdoor applications varying from military applications (Chen et al. 2020), express delivery industry, oilfield inspections (Ge, Fawei, et al 2020), to search and rescue tasks. The most important requirement for these tasks is the navigation control system which can provide precise, robust and effective trajectory planning. Trajectory planning is to plan vehicle motion from point A to point B without collisions in a period of time (Zammit and Erik-Jan 2018). With the introduction of a complex dynamic environment and model constraints, the UAV trajectory planning problem shifts from a simple 2D trajectory planning problem to a 3D trajectory planning problem, which requires it to automatically generate feasible and optimal paths to target points under numerous restrictions.

Traditional graph search path planning algorithms are divided into two major categories, Dijkstra’s algorithm and A* algorithm and their variants. The biggest weakness of Dijkstra’s algorithm is that it maintains an ordered queue of unexplored nodes. The search space and computational cost expand rapidly in large-scale or dynamic environments. This deeply troubled phenomenon is called sequential bottleneck. Most improved acceleration algorithms address the sequential bottleneck using different parallelization strategies or data structures. However, in dynamic UAV navigation scenarios, the shrinking speed of the search space does not
match the growth rate. Depending on the flexibility of heuristic function, A* can modify its own heuristic functions to adapt to different tasks. Adaptive A* algorithms are used in autonomous surface vessel navigation (Liu 2019) and evolutionary robotics (Prabhu and Jodie 2018). Since A* is limited by the heuristic function, the generated path always follows the edges of danger zones, which means that even the edges of danger zones are also dangerous. Moreover, A* can only explore between one source node and one target node in one run. It cannot handle the single-source and multi-target UAV navigation task. Due to the expansion of search space in a dynamic grid environment, it is hard to achieve both accuracy and efficiency with traditional exploration methods. There are also many restrictions, including the typical limited flight mileage and real-time weather conditions, that should be taken into consideration when we model the UAV’s environment. By directly utilizing the character of the grid environment to incorporate these dynamic elements and explore the environment along the time dimension, we can handle the expanding search space in dynamic environments.

Based on the discussion above, we construct an extended hierarchical graph (EHG) to model dynamic grid environment and design a novel matrix alignment Dijkstra (MAD) algorithm to explore the dynamic environment for UAVs. EHG is a high-dimension matrix whose shape is similar to the unexplored grid environment. EHG combines all information into passability coefficients using a security measurement function. Starting from the constructed EHG data structure, MAD uses matrix alignment to simulate forward exploration in all passable directions. The forward exploration is along the dimension of the advance of time. In this way, we can avoid the explosion of the search space and reduce the subsequent cost. In the exploration process from time t to time t+1, MAD first obtains the transmission matrix of at time t+1 from EHG. It performs matrix alignment and Hadamard product operations between the exploration matrix at time t and the transmission matrix at time t+1 in all passable directions. MAD then uses the navigator matrix to record information from the precursor cell in each cell at time t+1. Benefiting from GPU’s powerful parallel computing capabilities, the matrix computation takes little time. After reaching the target cells, MAD recalls and generates the final optimal paths with the help of the navigator matrix. MAD is specifically a deterministic, map-marked algorithm. The number of its exploration steps increases linearly as the number of search space increases. MAD can complete the path planning process from one source cell to multiple target cells in the full space in one single run.

Our main contributions are summarized as follows:

- We propose EHG as a universal structure to model the dynamic of grid environments, which is constructed by converting raw environmental data to quantified safety indicators for MAD. We use association kernels to gain more information from neighbouring cells to improve EHG’s accuracy.
- We design a forward exploration and backward navigation trajectory planning algorithm, matrix alignment Dijkstra (MAD), to explore the full search space and generate optimal paths from one source cell to multiple target cells by using matrix operations to simulate inner exploration. We use a GPU to accelerate the exploration process.
- We verify the validity of the proposed EHG structure and MAD algorithm in a real dynamic weather dataset and get competitive simulation results. We do in-depth analysis of the MAD in a benchmark dataset for grid-based path planning and an artificial maze dataset.

The rest of this paper is organized as follows. In Sect. 2, we give a brief overview of relevant previous work. Section 3 describes the proposed EHG structure and MAD algorithm in detail. Simulation results and the analysis of the algorithm are presented in Sect. 4. Finally, conclusions are made in Sect. 5.

2 Related work

The UAV trajectory planning task aims to generate an optimal path between the source point and the target point. Over the past two decades, a range of trajectory planning methods have been developed. Savuran and Murat (2016) proposed novel variants of vehicle routing problem for the efficient route planning of an unmanned air vehicle deployed on a moving carrier, in which depot mobility and range capacity constraints were introduced. Yao and Honglun (2017) proposed a novel Dynamic Adaptive Ant Lion Optimizer (DAALO) for route planning of unmanned aerial vehicle. Ramirez-Atenciano et al. (2017) designed and implemented a multi-objective genetic algorithm (MOGA) for solving complex mission planning problems involving a team of Unmanned Air Vehicles (UAVs) and a set of Ground Control Stations (GCSs). These methods can be divided into two kinds according to the research parts, world representation and path planning methods.

In order to reduce search space, many methods have been used in the world presentation. Knoll Knoll, Aaron (2006) proposed an extension of quadtrees to 3D, called octree, which is also widespread in image processing. A novel 3D visibility graph is proposed in You and Yan (2019) and applied in motion and control. Song Song, Chanyoung, et al (2019) introduced dynamic Voronoi diagrams for moving disks. Debarn Debnath et al. (2019) compared the visibility graphs with the navigation meshes and Voronoi diagrams.
Debnath et al. (2019) also gave detailed explanations of these methods. But these data structures cannot reflect the dynamics of the grid environment over a period of time. We proposed the EHG structure to model the grid environment and provide a convenient running structure for the MAD algorithm.

Many path planning methods were proposed. These methods can be divided into three main categories: graph-searched, sampling-based (Short 2016; Wang 2020) and evolutionary approaches (Ram Kishan et al. 2019; Zhang 2018; Cui 2018; Yao and Honglun 2017; Ramirez et al. 2017). Our work is the most relevant to graph-searched approaches, so we focus on this category. Graph-based searching algorithms define the state space as an occupancy cell and define obstacles as inaccessible grids. Then algorithms search the available cells and give a solution if a path exists (González et al. 2015). Graph-based algorithms guarantee a solution if the grid resolution is adequate. Dijkstra’s Johnson, Donald (1973) and A* are typical graph search algorithms. For different application scenarios, the variants of Dijkstra and A* (Swathi and Hemamalini 2016; Daniel 2010; Przybylski 2018; Gia Luan 2020) have been proposed. Many researchers have introduced some parallel algorithms (Jasika et al. 2012; Ortega-Arranz 2015) with improved efficiency. The differential evolution (DE) algorithm is another efficient random search algorithm to deal with navigation task. Some improved methods are proposed, SFSADE (Pan et al. 2021) for example. These methods sacrifice some accuracy for greater efficiency. But both accuracy and efficiency are important when the application scenarios become more complicated.

### 3 Methodology

#### 3.1 Definition of problem

We first give the definition of the UAV navigation problem in a dynamic environment. The navigation scene has been divided into cells with the shape $X \times Y$. Additionally, detailed weather information for the cells over a period time is provided by 10 different weather forecast models. $T$ is the length of this period time and time $t$ is the specific time. The navigation task aims to generate the safest paths by avoiding dangerous weather areas between one source cell $(S_x, S_y)$ and multiple target cells within the limited flight mileage. The list of target cells is noted as $[(T_{x1}, T_{y1}), \ldots, (T_{xn}, T_{yn})]$. UAVs have a constant power reserve and a limited flight duration. We assume that the flight speed of UAV remains the same under all weather conditions, and that the flight time in each area block is fixed, limited to flying over an area block in 2 min. UAVs can only fly up, down, left and right from the current cell to the next cell, or stay in the current cell.

#### 3.2 The framework of matrix alignment Dijkstra algorithm

We provide a brief introduction of the workflow of MAD, which is shown in Fig. 1. The workflow can be divided into two stages, the environment representation stage and trajectory planning stage. In the environment representation stage, which is shown in the left, the unexplored environment is converted into a high-dimensional matrix, extended hierarchical graph (EHG). In this process, we first pre-process the environment information. Then the machine learning method is used to extract valid associations of features between the past and the present to improve the accuracy of the environmental representation. Finally a high-dimensional extended hierarchical graph is output. In the trajectory planning stage, the matrix alignment Dijkstra carries out trajectory planning under the EHG data structure. MAD can be divided into forward exploration and backward navigation process. MAD uses matrix alignment operation to achieve parallel exploration in the forward exploration process. MAD uses exploration matrix to record the state of exploration. The transmission matrix represents the environmental information which is obtained from EHG. Then MAD does Hadamard product between the exploration matrix and transmission matrix to obtain the optimal precursor direction and saves it into navigator matrix. In the backward navigation process, MAD uses navigator matrix to generate the optimal paths. The parallel exploration of the search space realizes the avoidance of the sequential bottleneck problem in Dijkstra’s algorithm. In this way, MAD can complete single-source multi-target trajectory planning problems at the same time.

#### 3.3 The construction of extended hierarchical graph

EHG is a multi-dimensional matrix structure to model a dynamic environment over a period of time. It has the same shape as the unexplored grid environment. We take for example the real weather dataset, which shows the weather conditions of an area over a period of time. The example
The learning process of a typical representative of time-dependent elements, weather conditions.

In light of this characteristic, we can apply useful machine learning methods to extract valid associations of features between the past and the present. Inspired by convolutional neural network (CNN), we design an association kernel which is responding to the surrounding cells for a portion of time and space coverage. The association kernel is actually a convolution kernel of 3D-CNN Ji et al. (2012), which is used to extract the influence of the surrounding grids on the central grid. We can obtain more accurate weather data from multiple weather forecasts. A schematic diagram of the learning process of weather conditions is shown in Fig. 3. We distill the features from neighbouring cells and stitch the feathers into one feature vector. Then we can split the association kernels and splice them into the association kernel weights. The process of extracting features by convolution is equivalent to applying linear regression to the stitched features. By applying association kernels, we improve the accuracy of the passable probability.

After a comprehensive study of available elements, we build the EHG with the shape $X \times Y \times T$ and provide the running carrier for the MAD algorithm.

### 3.4 Matrix alignment Dijkstra algorithm

With the explosion of search space in a dynamic environment, the sequential bottleneck of Dijkstra’s algorithm will lead to inefficient planning. To overcome the inefficiency, we propose matrix alignment Dijkstra algorithm to accelerate planning. Traditional acceleration methods of Dijkstra’s algorithm focus on handling the ordered queue of unexplored nodes, including multiply thread, efficient data structure and better equipment. And these methods are serial in nature. But if we simulate all possible paths and finally select the best path among them, we can avoid the sequential bottleneck and realize real parallel exploration. Due to the fixed movement direction in the grid environment, we can use matrices to model the grid environment and shift matrix to simulate...
Matrix alignment Dijkstra algorithm can be divided into two stages: forward exploration process and backward navigation process. In MAD, we will use an exploration matrix, a transmission matrix and a global navigator matrix. Value in each cell of the exploration matrix $M_{\text{exp}}$ means the lowest crash probability of the path between the source cell and the current cell. The exploration matrix is a matrix whose values change over time. In order to help better understand this changing matrix, we add an superscript $t$ to the exploration matrix which means the crash probability after the $t$-round of exploration, $M_{\text{exp}}^t$. The transmission matrix $M_{\text{trans}}$ is obtained from the constructed EHG structure, and value in the $M_{\text{trans}}$ represents the passable coefficient of corresponding cell in the specific time. The path navigator matrix maintains pointers from the precursor to the current cell, which can be used to recall the shortest path. The algorithm’s outputs are the task-specific optimal paths. In the forward exploration process, MAD first uses function GetLayerTransmission to get the transmission matrix at the specific time. The function GetLayerTransmission is used to get the transmission matrix from the prebuilt multi-dimensional matrix structure, EHG. Then MAD uses function exploration to shift the exploration matrix by one unit in the given passable direction. Afterwards, MAD executes exploration actions according to task-specific rules. In the example of weather condition, we initialize the $M_{\text{exp}}[s_i][s_j] = 0$ for a starting crash probability of zero. In each iteration, the algorithm first gets the transmission matrix of the current time. Then the exploration matrix is adjusted in the all four passable directions and is readied for exploration. Because of the alignment operation, the blank needs to be filled by border. For example, value 1 indicates a crash. Due to the target task for the safest path, we take exploration according to line 21 in Algorithm 1. This inner exploration process from $t$-th time to $t+1$-th time is clearly displayed in Fig. 4 according to Formula 1.

$$P(t) = I - (I - P(t)) * (1 - P(t+1))$$

where $P(t)$, obtained from $M_{\text{exp}}$, represents the crash probability of the current path and $p(t)$, obtained from $M_{\text{trans}}$, represents the crash probability in the $t$ time, and $I$ represents a matrix whose elements are all 1. On the left of the green Hadamard product symbol is the alignment process of the exploration matrix $M_{\text{exp}}$. The centre matrix $M_{\text{exp}}$ is the exploration matrix at the $t$-th time. It also represents the situation of staying still. Each cell stores the lowest crash probability of the trajectory from the starting cell to the current cell. The four arrows represent the given four passable directions. For example, the upward arrow represents upward exploration process. In the upward alignment process, we move the exploration matrix $M_{\text{exp}}$ upward by one cell. We then fill value 1 in the vacancies which comes as a result of alignment operations and represents unreachable area. In order to show it more clearly, we pick the cell with value

![Exploration Matrix of Time t](image1)

![Transmission Matrix of Time t+1](image2)

**Fig. 4** The example of inner-layer exploration movement. More we can use GPU to further accelerate the above process.
0.12 marked in green to demonstrate. Alignment upward, left, right and downward are marked with red, yellow, purple and blue, respectively. But in fact, the objects of the alignment operation are all the cells on the map.

On the right of Hadamard product is the transmission matrix at the unexplored time $t+1$. In the upward exploration process, we operate Hadamard product between the transmission matrix $M_{\text{trans}}^{t+1}$ and the upaligned exploration matrix $M_{\exp}$ according to modified Formula 2.

$$M_{\exp}^{t+1} = I - (1 - M_{\exp}^t) * (I - M_{\trans}^{t+1})$$

where $M_{\exp}^t$ represents the crash probability of UAVs, $I$ represents a matrix whose elements are all 1, so $I - M_{\exp}^t$ represents the pass probability of UAVs.

An overview of the entire inner-layer exploration process is shown in the right of the green arrow. The different colours represent the four different passable direction, respectively. After inner exploration process, we can get one $M_{\exp}^{t+1}$ from one passable direction. Then we can take the minimum value corresponding to each cell from all $M_{\exp}^t$ which is called dimension reduction operation. The minimum matrix is treated as the exploration matrix for the time $t+1$. We can get the pointer to indicate the exploration direction of the minimum value of the previous step. We use a navigator matrix to save off the pointer from the previous step so that we can rebuild the optimal path. Values 1, 2, 3, 4 correspond to up, right, down, left, respectively. In this way, we finish the alignment upward, right, down, left, respectively. In this way, we finish the upward exploration, right, down, left, respectively. In this way, we finish the upward exploration.

Notably, MAD can be used not only by dynamic 3D environments but also by large-scale static 2D environments to reduce the search space. The difference between these two use cases is in the construction of their respective EHG. We can extend 2D into 3D with a simple copy. We replace the time dimension with the path length increasing dimension. After we reach target cells, MAD terminates. With this, we can analyse the performance of MAD. Our results are surprisingly good. We propose the following lemmas.

**Algorithm 1** Matrix Alignment Dijkstra’s algorithm

**Input:**
- The constructed EHG: $M$ with shape of $X \times Y \times T$
- The source cell: $(S_x, S_y)$
- The list of target cells: $\{ (T_{x_1}, T_{y_1}), \ldots, (T_{x_n}, T_{y_n}) \}$

**Output**
- The Global Navigator Matrix: $M_{\text{navi}}$

**Initialize:**
- The Exploration Matrix: $M_{\exp}$
- The Global Navigator Matrix: $M_{\text{navi}}$
- The Transmission Matrix: $M_{\trans}$
- The Temp Matrix: $\text{Temp}$ with size $X \times Y \times T$

1: Initialize $M_{\exp}[S_x][S_y]$
2: while targets aren’t all reached do
3: $M_{\trans} = \text{GetLayerTransmission}(t)$
4: for each direction $i$ do
5: $M_{\text{align}} = \text{Alignment}(M_{\exp}, \text{direction}_i)$
6: $\text{Temp}[\ldots, i] = \text{Exploration}(M_{\text{align}}, M_{\trans}^{t=1})$
7: end for
8: $M_{\exp}, M_{\text{navi}} = \text{Temp.Dimension_Reduction}$
9: $t = t + 1$
10: end while
11: return $t$ and $M_{\text{navi}}$
12: function ALIGNMENT($M$, direction)
13: alignment $M$ in the passable direction
14: for each target in the list
15: return $M_{\text{alignment}}$
16: end function
17: function EXPLORATION($M_{\text{align}}, M_{\trans}$)
18: if task is numeric then
19: return $M_{\text{align}} + M_{\trans}$
20: else task is probabilistic
21: return $I - (1 - M_{\text{align}}) * (I - M_{\trans})$
22: end if
23: end function
24: function GETLAYERTRANSMISSION$(t)$
25: return $M_{\trans}^t$
26: end function

**Algorithm 2** Recall & Create Path

**Input:**
- The source cell: $(S_x, S_y)$
- The list of target cells: $\{ (T_{x_1}, T_{y_1}), \ldots, (T_{x_n}, T_{y_n}) \}$
- Termination time list: $\{ t(S_1, S_y), \ldots, t(S_n, S_y) \}$
- The Global Navigator Matrix: $M_{\text{navi}}$

**Output**
- The optimal paths: $\text{Path}((S_x, S_y), (T_{x_1}, T_{y_1})), \ldots, \text{Path}((S_x, S_y), (T_{x_n}, T_{y_n}))$

1: for each target cell in the list do
2: $t = t(S_x, S_y)$
3: while $t \geq 0$ and source cell is not reached do
4: $\text{direction} = M_{\text{navi}}[T_x, T_y, t]$
5: TempCell = $\text{Find precursor}(T_x, T_y, \text{direction})$
6: AddInPath(TempCell)
7: $t = t - 1$
8: end while
9: end for
10: return Path
Lemma 1  The runtime has a strict linear correlation with the length of the optimal path.

This lemma is related to the termination situations of MAD. From the algorithm, we can see that the length of the path is strictly equal to the number of exploration iterations. Hence, the runtime is linearly related to the scale of the environment and the length of the optimal path.

Lemma 2  Matrix alignment Dijkstra algorithm is the lower bound of classical Dijkstra’s algorithm.

The best case scenario of classical Dijkstra is that the explored cells are all existing in the optimal path between source and target. Consequently, the search iterations must be the length of the existing path. This is precisely the general case of MAD algorithm according to Lemma 2.

We provide sufficient experimental simulation results to prove Lemmas 1 and 2. Details are given in Sect. 4.

4 Simulation results

In this section, we first describe our hardware configuration. Then we present the simulation on the real dynamic weather condition environments. Finally, we analyse the performance of MAD on an artificial dataset and the benchmark grid maps of Sturtevant (2012).

4.1 Hardware configuration

The simulations are conducted in the IDE Spyder with Python 3.7.1 on an AMD Ryzen 7 2700X (eight-core processor) with 3.7GHZ CPU and with CUDA 9.0 on an NVIDIA GeForce GTX 1080 Ti GPU.

4.2 Simulation on the weather dataset

4.2.1 Weather map data

Our weather map data are a real-world dataset which can be obtained from the Tianchi competition platform. Weather data are provided by the Met Office. The dataset contains real data for 12 h per day for 5 days. A weather forecast model predicts weather data each hour. They divide the full weather map into cells which can be represented by an unique pair of coordinates \((x, y)\). The whole map \(W\) is modelled as a grid map with size \((548 \times 421 \times 12 \times 5)\). The passability conditions of each cell can be predicted by the given value of wind speed. The motion of UAV is limited to the vertical and horizontal directions. Staying still is also allowed. The task is to plan an effective and the safest path for UAVs to traverse the airspace and evade hazardous weather areas.

\[ (x, y, z) \]

4.2.2 Implementation details

- Data preprocessing Raw data \(w(x, y, t)\) describe the wind speed of a specific cell at time \(t\). After processing outliers, we apply the sigmoid function to convert real wind speed values to the crash probabilities. Then we apply the learning methods mentioned earlier to obtain the association kernels. The learning process is equivalent to linear regression. We split and stitch features according to their spatial and temporal order and feed them into a linear regression model. This completes the construction of the EHG structure.

- Analysis of Generated Trajectories Since different application environments have different influencing factors, and different planning tasks are corresponding to different target tasks, the corresponding path planning algorithms are basically designed for specific application areas, but they lack generality. Besides, Dijkstra algorithm belongs to the path planning algorithm based on graph search category in path planning field, so we choose the Improved 3D-A* and Dijkstra algorithm as comparison algorithms. The Improved-A* algorithm (Liu et al. 2019) is proposed by Chenguang Liu and applied to surface ships for path planning. In the Improved-A* algorithm, factors such as water velocity and obstacle collisions are considered dangerous for the navigation of ships. Finally the safest path is generated. This target optimal path under this task is equivalent to the trajectory planning target under dynamic weather environment. The difference is that the surface navigation scenario is a 2D static environment, but the UAV’s navigation scenario is 3D and dynamic. Based on this Improved-A* algorithm, we add the time dimension and implement the Improved 3D-A* algorithm. In order to reduce the search space as the dimension increases, we improve the heuristic cost function. Only the grids with an absolute distance within 60 units near the currently explored grid point are calculated, which saves nearly half of the exploration time. At the same time, we increase the safety factor of safety grid. Then we use the Improved 3D-A* algorithm as a comparison method. Based on the original Dijkstra algorithm, Dijkstra algorithm uses the sorting optimization technology of the priority queue to optimize the serialization bottleneck as another comparison algorithm. In the scenario of UAV navigation, the target path is the safest path within the flight mileage, and the safety degree

| Table 1 Comparison between different navigation algorithms in the dynamic weather data |
|---------------------------------|----------------|------------------|
| Average time                   | MAD            | Improved 3D-A*   | Dijkstra        |
| Safety                         | 0.8981         | 0.7908           | 0.8981          |
| Average time                   | 11.4757(s)     | 41.6019(s)       | 22654.5865(s)   |

1 https://tianchi.aliyun.com/competition/entrance/231622.
refers to the ability of the planned path to avoid risks. Given a planned path \( \text{Path} \), its safety degree is defined as shown in Formula 3. When the Improved-A* algorithm is extended from 2D to 3D, a similar EHG structure is also used. Therefore, the Improved 3D-A* algorithm calculates safety degree of its path using the same method as Formula 3.

\[
\text{Safety(\text{Path})} = \sum_{t=1}^{\text{|Path|}} (1 - p(t)) / \text{|Path|}
\]  

(3)

where \( \text{Path} \) is the generated target path, \( |\text{Path}| \) is the length of the target path \( \text{Path} \), \( p(t) \) represents the safe passing probability of the grid corresponding to time \( t \) on the target path. The specific value of \( p(t) \) is obtained from the constructed extended hierarchy diagram according to the specific grid position and specific time. Based on real weather map data, we perform a verification experiment on MAD algorithm, the Improved 3D-A* algorithm and Dijkstra algorithm for the safety degree and the time spent to complete the planning task. Specific experiments were carried out to obtain the average value of the performance of each algorithm in the real dynamic meteorological dataset. The verification results are shown in Table 1. As can be seen from Table 1, in the weather data within 12 h of one day to complete the UAV navigation task from the starting city to 10 destination cities, MAD algorithm only takes 11.4757 s to complete the global search in 13,842,480 search states and plans the safest path between the starting city and 10 destination cities, and the safety degree of the path is as high as 0.8981. Because of serialization bottleneck due to the exponential growth of search space brought by dynamic scenes, Dijkstra algorithm’s running time is as long as 22,654.5865 s to generate target path. As we know, Dijkstra algorithm is also a path planning algorithm for global search. Therefore, paths generated by Dijkstra algorithm and MAD algorithm as well as the safety degrees of these paths remain the same. In order to improve the safety degree of the path, the Improved 3D-A* algorithm increases the security threshold that can pass through the grid, and in order to improve the efficiency of the algorithm, the Improved 3D-A* algorithm only calculates the grids with an absolute distance within 60 units near the currently explored grid point, which effectively saves nearly half of the exploration time, compared with global computation. Even so, it takes 41.6019 s. Since the A* algorithm can only solve the path planning problem from a single starting point to a single target point in one run, it has to repeat 10 times to complete the navigation task from the starting point city to 10 destination cities, which reduces the efficiency of the algorithm. At the same time, because of the characteristics of heuristic algorithm, the grid selected after cost function calculation is often close to the edge of the dangerous region, so safety degree of the path generated by the Improved 3D-A* algorithm is lower than the other two methods, which is 0.7908. To show the advantage of MAD, we pick one path out of all generated paths from the source to different targets and present it in Fig. 5. Because of the dynamical character of weather map, we cannot present the global dynamic meteorogram in a 2D graph. Therefore, we choose some special time periods to describe the trajectories. The selected trajectory is divided into multiple segments. The selected trajectory starts from red square (142, 328). Between S and A, we find that the weather conditions are fair, so the two paths extend in an approximately the same direction. At point A, the two algorithms generate different navigation directions under the influence of the extreme weather condition. The dotted line shows the trajectory of the extreme weather conditions. The weather forecast can only update once an hour, and UAV can move 30 cells in one hour. During the hour when the extreme weather event is marked purple, the improved 3D A* algorithm prefers to move to point B along the edge of the extreme weather event in the vertical direction. The path from point A to point D is close to the extreme weather event in this period of time. Meanwhile, the path from point A to C generated by MAD moves away from danger at the same time. In this way, the cells in the path are low risk. In the next hour, when the extreme weather event is marked azure, it moves down. MAD still chooses lower-risk cells to expand and generates the segment from C to D. Then the extreme weather event moves to the left, shown in brown, while MAD moves right and upward to E after reaching point D. Improved-3D A* has good efficiency but the path generated by it is sub-optimal and less safe. Moreover, it can only compute a path from the source cell to one single target cell in one single run. With the MAD algorithm, we can even compute several days at the same time by adding an extra dimension of date to matrices. MAD can compute all possible paths from source to the list of target cells in one run. It takes an average of 11.4757 s to compute the safest path for one day from one source cell to ten target cells, averaged over 100 runs.

4.3 Performance analysis of MAD

The limitations of a real weather dataset preclude it from being suitable for data scalability experiments. Instead, experiments are conducted on static artificial data and grid benchmark datasets to test the performance of the MAD algorithm on different scales and different datasets, including accuracy and time cost.
Trajectory planning for UAV navigation in dynamic environments with matrix alignment Dijkstra

Table 2  Comparison of the length of the optimal paths with different algorithms in different maps

| Map size     | Edges    | MAD   | Dijkstra | Bidirection-Dij | MT-Dij | Frequency |
|--------------|----------|-------|----------|-----------------|--------|-----------|
| 10 × 10      |          | 16    | 16       | 16              | 16     | 1         |
| 100 × 100    | 5 KB     | 107.70 | 107.70   | 107.70          | 107.70 | 1000      |
| 500 × 500    | 17.1 MB  | 651.40 | 651.40   | 651.40          | 651.40 | 1000      |
| 1000 × 1000  | 70.9 MB  | 1576.44| 1576.44  | 1576.44         | 1576.44| 500       |
| 2000 × 2000  | 288.3 MB | 3206.80| 3206.80  | 3206.80         | 3206.80| 200       |
| Mazes with 1 corridor | 15.0 MB | 2231.57| 2231.57  | 2231.57        | 2231.57| 17863     |
| Mazes with 8 corridor | 27.9 MB | 1725.64| 1725.64  | 1725.64        | 1725.64| 8561      |
| Mazes with 16 corridor | 29.5 MB | 1200.3848| 1200.3848| 1200.3848     | 1200.3848| 5796     |
| Mazes with 32 corridor | 30.4 MB | 1032.7663| 1032.7663| 1032.7663    | 1032.7663| 5998     |
| Percentage of optimal paths | - | 100.0% | 100.0%   | 100.0%          | 100.0% | -         |

Table 3  Comparison of execution time with different algorithms in different maps

| Map size     | Edges    | MAD(s) | Dijkstra(s) | Bidirection-Dij(s) | MT-Dij(s) | Frequency |
|--------------|----------|--------|------------|-------------------|-----------|-----------|
| 10 × 10      |          | 0.0086 | 0.0002     | 0.0002            | 0.0103    | 1         |
| 100 × 100    | 5 KB     | 0.0566 | 4.3520     | 2.7321            | 2.6147    | 1000      |
| 500 × 500    | 17.1 MB  | 1.3912 | 45.7404    | 21.2403           | 27.4493   | 500       |
| 1000 × 1000  | 70.9 MB  | 11.9212| 353.1257   | 193.2405          | 215.9336  | 200       |
| 2000 × 2000  | 288.3 MB | 105.9964| 45.7404   | 21.2403           | 27.4493   | 500       |
| Mazes with 1 corridor | 15.7 MB | 4.9206  | 0.7642     | 0.0405            | 0.5897    | 17863     |
| Mazes with 8 corridor | 27.9 MB | 4.2242  | 2.6310     | 1.4082            | 2.5356    | 8561      |
| Mazes with 16 corridor | 29.5 MB | 3.8768  | 3.6185     | 1.9093            | 2.3234    | 5796      |
| Mazes with 32 corridor | 30.4 MB | 2.4330  | 5.5028     | 2.8632            | 3.2979    | 5998      |

Boldface values represent the least execution time of each row among different comparison algorithms in a certain map

4.3.1 Datasets

- Artificial Maze Data The maps used for testing the MAD algorithm and classic Dijkstra’s algorithm are generated randomly by randomized Prim’s algorithm (Jeong et al. 2018). Maps are generated with more branches, so they are more complicated and harder. Maps of different sizes (x × y) of (10 × 10), (100 × 100), (500 × 500), (1000 × 1000), (2000 × 2000) contain edges of 5 KB, 654 KB, 17.1 MB, 70.9 MB, 288.3 MB, respectively.
- Mazes Mazes from the benchmarks for grid-based path-finding (Sturtevant 2012) are formed randomly, with a number of parameters influencing the layout of the map. Initially the whole map is blocked except for a single square in the centre. The algorithm attempts to extend the current maze corridor, but has a (3%) chance of randomly selecting a different corridor to extend. All maps (x × y) in this set are 512 × 512 and are parameterized by the corridor size in the passages, which can be 1, 2, 4, 8, 16, or 32.

4.3.2 Analysis of simulation

The simulation in Table 2 shows that, when facing G4-grid maps, MAD algorithm always shows the same optimal path as Dijkstra’s algorithm, which is the optimal path. We choose a random start vertex and a target vertex in the given grid map at each run and compute the shortest length path. The statistical result is an average value from 1000 times in the size (100 × 100), 1000 times in the size (500 × 500), 500 times in the size (1000 × 1000), 200 times in the size (2000 × 2000). In the experiments, we select Dijkstra’s algorithm, bi-direction Dijkstra’s algorithm and multi-thread Dijkstra’s algorithm (Bazan et al. 2016) as the contrast algorithms.

Table 3 shows the statistical information about execution time to find an optimal path for each map size of random maps, for Dijkstra, multi-thread Dijkstra and MAD algorithms. We can see that the improvement is very exaggerated. When dealing with a small scale, below size (100 × 100) for example, Dijkstra’s shows a better performance than our MAD algorithm because of the powerful computation of CPU. However, when it comes to much larger-scale size maps, the difference between Dijkstra and MAD is some orders of magnitude.

Figure 6 shows a scatter plot of the execution time (for finding the optimal path) for different map sizes of random maps, for Dijkstra, multi-thread Dijkstra and MAD algorithms. We can see that the improvement is very exaggerated. When dealing with a small scale, below size (100 × 100) for example, Dijkstra’s shows a better performance than our MAD algorithm because of the powerful computation of CPU. However, when it comes to much larger-scale size maps, the difference between Dijkstra and MAD is some orders of magnitude.

Figure 6 shows a scatter plot of the execution time (for finding the optimal path) for different map sizes of random maps, for Dijkstra, multi-thread Dijkstra and MAD. The trend of MAD’s algorithm’s execution time provides a clear
Fig. 5 Comparison between two generated trajectories of improved 3D-A* and MAD. The imaginary line indicates the storm’s direction. The entire trajectory is cut into several segments according to the corresponding time period, and the colour of the end point of each segment is consistent with the colour used to identify storms in different time periods.

and strong evidence of Lemma 1. We find that as the length of the optimal path increases, the execution time increases at a low linear rate and highlights its advantages with the increase of the map size. In other words, the execution time will increase linearly as the searching space becomes larger.

As is mentioned above, mazes in the benchmarks have different corridor sizes (1, 2, 4, 8, 16, 32) in their passages. We choose mazes with corridor sizes of 1, 8, 16, 32 to verify the performance of MAD algorithm. Figure 7 shows the experiment results of different corridor sizes. Namely, in the mazes with a smaller corridor size, the number of passable cells is less than that of mazes with a larger corridor size.

At the same time, the number of passable edges indicates the upper bound of Dijkstra algorithm. So when we handle smaller corridor size mazes, the set of unexplored edges is smaller and the execution time of Dijkstra will decrease correspondingly. But for the larger sizes of corridors and unexplored edge sets, the execution time of MAD remains stable and shows performance linear to path’s length. Figure 7 displays it clearly. This result can be explained as follows: the upper bound of Dijkstra is the size of the unexplored edges set and the lower bound condition is when one path exists between source and target and the path length is equal to the size of the unexplored edges set. However, the efficiency of MAD is strictly and linearly dependent on path length. Therefore, the corridor size can significantly influence the efficiency of Dijkstra but has little influence on the MAD. In other words, MAD will be the lower bound of Dijkstra when the speed of GPU iteration becomes as fast as CPU itera-

Fig. 6 The scatter plot of execution time versus the optimal path for different size of random map.
5 Conclusion

In this work, we first present a new approach, extended hierarchical graph, to model the dynamic grid-based environments. We convert raw information to passable coefficients that are used to build the EHG structure. Then we design a trajectory planning algorithm, matrix alignment Dijkstra, and search directly in the EHG to decrease the search space explicitly. MAD can work in the large-scale and dynamic environment by using an exploration matrix to explore forward and a navigator matrix to recall the optimal path backward. Benefiting from Dijkstra’s optimality and GPU’s powerful matrix computing capabilities, MAD obtains lower planning time cost and achieves optimal global path. MAD achieves remarkable improvements both in efficiency and practicality over heuristic-based A* and Dijkstra’s variants. Due to the limitations of the real weather dataset, we test MAD on different scale maps to analyse the performance of the algorithm itself. The simulation result provides strong evidence for the lemmas. It proves that the running time of the algorithm is linearly related to the size of the environment. Limited by the computation speed of GPU, MAD does not perform as well as Dijkstra’s variants on small-scale environments in the analysis of MAD’s performance. But when the environment becomes larger, MAD shows excellent performance. And we do not take collision avoidance (Tang et al. 2021) into consideration when multiple aircraft plan navigation routes at the same time.

In future work, we plan to combine more elements into experiments and expand the current MAD in more passable
directions. In terms of algorithm parallelism, we would like to
study how to plan the trajectories of multiple aircraft simulta-
necessarily. We are also interested in the full UAV aerodynamic
constraints, such as fuel reserve and maximum yaw rate, and
in researching the relationship between risk probability and
fuel or power capability. Finally, for the comparison between
UAV trajectory planning algorithms, it is very important to
create a benchmark testing dataset.

**Funding** This work is supported by National Natural Science
Foundation of China Under Grants U1936108, U1836204, U1401258 and
61572222.

**Data availability** Enquiries about data availability should be directed to
the authors.

**Declarations**

**Conflict of interest** The authors have not disclosed any competing inter-
ests.

**References**

Bazan M, et al. (2016) Multithreaded enhancements of the Dijkstra algo-
rithm for route optimization in urban networks. Arch Transp Syst
Telemat 9

Chen X, Tang J, Lao S (2020) Review of unmanned aerial vehicle
swarm communication architectures and routing protocols. Appl
Sci 10(10):3661

Cui J et al. (2018) UAV motion strategies in uncertain dynamic environ-
ments: a path planning method based on Q-learning strategy. Appl
Sci 8(11):2169

Daniel K et al. (2010) Theta*: any-angle path planning on grids. J Artif
Intell Res 39:533–579

Debnath SK, Omar R, Latip NBA (2019) Comparison of different con-
figuration space representations for path planning under combina-
torial method. Indones J Electr Eng Comput Sci 14(1):1–8

Ge F et al. (2020) Path planning of UAV for oilfield inspections in a three-
dimensional dynamic environment with moving obstacles based on
an improved pigeon-inspired optimization algorithm. Appl Intell
50:2800

Gia Luan P (2020) Real-time hybrid navigation system-based path
planning and obstacle avoidance for mobile robots. Appl Sci
10(10):3355

González et al. D (2015) A review of motion planning techniques for
automated vehicles. IEEE Trans Intell Transp Syst 17(4):1135–
1145

Jasika N et al. (2012) Dijkstra’s shortest path algorithm serial and parallel
execution performance analysis. In: 2012 proceedings of the 35th
international convention MIPRO. IEEE

Jeong K, Lee J, Kim J (2018) A study on new virtual reality system in
maze terrain. Int J Hum Comput Integ 34(2):129–145

Ji S, Xu W, Yang M et al. (2012) 3D convolutional neural networks for
human action recognition. IEEE Trans Pattern Anal Mach Intell
35(1):221–231

Johnson, Donald B (1973) A note on Dijkstra’s shortest path algorithm.
J ACM 20(3):385–388

Knoll A (2006) A survey of octree volume rendering methods. GI, the
Gesellschaft für Informatik. 87

Liu C et al. (2019) An improved A-star algorithm considering water
current, traffic separation and berthing for vessel path planning.
Appl Sci 9(6):1057

Liu C, Mao Q, Chu X et al. (2019) An improved A-star algorithm con-
considering water current, traffic separation and berthing for vessel
path planning. Appl Sci 9(6):1057

Ortega-Arranz H et al. (2015) Comprehensive evaluation of a new GPU-
based approach to the shortest path problem. Int J Parallel Prog
43(5):918–938

Pan Q, Tang J, Wang H et al. (2021) SFSADE: an improved self-adaptive
differential evolution algorithm with a shuffled frog-leaping strat-
egy. Artif Intell Rev. https://doi.org/10.1007/s10462-021-10099-9

Prabhu SGR, Peter K, Jodie W (2018) Investigating an A-star algorithm-
based fitness function for mobile robot evolution. In: 2018 22nd
international conference on system theory, control and computing
(ICSTCC). IEEE

Przybylski M (2018) AD*-Cut: a search-tree cutting anytime dynamic
A* algorithm. In: Twenty-Eighth international conference on auto-
mated planning and scheduling

Ram Kishan D, Shukla A, Wilfred Godfrey W (2019) Three dimensional
path planning using Grey wolf optimizer for UAVs. Appl Intell
49(6):2201–2217

Ramirezatencia C et al. (2017) Solving complex multi-UAV mission
planning problems using multi-objective genetic algorithms. Soft
Comput 21:4883–4900

Savurun H, Murat K (2016) Efficient route planning for an unmanned air
vehicle deployed on a moving carrier. Soft Comput 20:2905–2920

Short A et al. (2016) Recent progress on sampling based dynamic motion
planning algorithms. In: 2016 IEEE International conference on
advanced intelligent mechatronics (AIM). IEEE

Song C et al. (2019) Dynamic voronoi diagram for moving disks. Trans
Vis Comput Graph 27:2923

Sturtevant NR (2012) Benchmarks for grid-based pathfinding. IEEE
Trans Comput Intell Al Games 4(2):144–148

Swathiha OVG, Hemamalini S (2016) Prims-aided dijkstra algorithm
for adaptive protection in microgrids. IEEE J Emerg Sel Top Power
Electron 4(4):1279–1286

Tang J, Lao S, Wan Y (2021) Systematic review of collision-avoidance
approaches for unmanned aerial vehicles. IEEE Syst J. https://doi.
org/10.1109/JSYST.2021.3101283

Wang X et al. (2020) Collision-free path planning method for robots
based on an improved rapidly-exploring random tree algorithm.
Appl Sci 10(4):1381

Yao P, Honglun W (2017) Dynamic adaptive ant lion optimizer applied
to route planning for unmanned aerial vehicle. Soft Comput
21:5475–5488

You Y, Caixia C, Yan W (2019) 3D visibility graph based motion plan-
ing and control. In: International conference robotics and artificial
intelligence

Zammit C, Erik-Jan Van K (2018) Comparison between A* and RRT
algorithms for UAV path planning. In: 2018 AIAA guidance, nav-
igation, and control conference

Zhang Q et al. (2018) Modified collective decision optimization algo-
rithm with application in trajectory planning of UAV. Appl Intell
48(8):2328–2354

**Publisher’s Note** Springer Nature remains neutral with regard to juris-
dictional claims in published maps and institutional affiliations.