UAV Path Planning Based on GeoSOT Grid and JPS3D Optimized Algorithm

Yutian Zhang [1], Ying Nie [1], Yuanyuan Liu [1]

[1]North China Institute of Computing Technology，Beijing，China
2861089402@qq.com, nyyyh1997@sina.com, 269959041@qq.com

Keyword： JPS3D, GeoSOT, UAV pathfinding, GPJ3D-RP, Parallel Computing

Abstract

With the rapid advancement of drone technology, the demands for accuracy and real-time performance in air route planning within modern battlefield environments have significantly increased. In high-precision three-dimensional battlefield modeling, a surge in the number of targets, along with complex obstacle distributions and dynamic threat factors, greatly raises the complexity of path computation.Especially in large-scale three-dimensional complex environments, traditional A* algorithms suffer from efficiency losses due to a lack of effective pruning mechanisms, which leads to excessive redundant node expansions. To address this challenge, this paper implements a three-dimensional Jump Point Search algorithm based on the GeoSOT global discrete grid system. Leveraging the spatial partitioning characteristics of GeoSOT encoding, we propose an efficient path planning algorithm called GPJ3D-RP (GeoSOT Parallelized JPS3D for Rapid Pathfinding). This approach decomposes the global high-precision path planning task into multiple low-resolution subregions, enabling parallel processing of local path searches within each sub-block and ultimately integrating the results into a complete path. Through simulation experiments involving both fighter aircraft route planning and UAV path planning at two different scales, the GPJ3D-RP algorithm demonstrates significant improvements in search speed compared to traditional A* and basic JPS-3D algorithms, making it better suited for real-time path planning requirements in dynamic and complex battlefield environments.

1. Introduction

1.1 Military Requirements

With the increasing integration of drone technology into modern military operations, unmanned aerial systems (UAS) have emerged as pivotal components in battlefield reconnaissance, target acquisition, and precision strike missions. In complex static battlefield environments, drones must compute optimal flight paths within high-resolution three-dimensional terrain models, imposing stringent demands on path-planning algorithms. Contemporary digitized battlefields are characterized by vast operational areas, intricate topographical features, and irregular obstacle distributions, necessitating algorithms capable of processing large-scale spatial data while maintaining fine-grained environmental awareness(Wang G,2012).

Existing mainstream path-planning algorithms exhibit notable limitations when applied to large-scale combat scenarios. Grid-based search methods, while reliable in structured environments, frequently generate excessive computational overhead in open terrains, significantly degrading planning efficiency. Although jump point search (JPS) algorithms mitigate this issue through intelligent node expansion strategies, their three-dimensional variants still encounter substantial challenges in complex battlefield settings. A critical shortcoming lies in their inability to simultaneously accommodate extensive operational ranges and high-fidelity terrain data, often forcing undesirable trade-offs between computational speed and path quality—compromising mission effectiveness in real-world combat operations(Aggarwal S,2020).

To address these challenges, this study introduces an innovative three-dimensional path-planning architecture. The proposed algorithm leverages advanced spatial indexing techniques to enable hierarchical management of the search space(Jones M,2023). Specifically, it employs a multi-resolution grid representation, allowing rapid identification of navigable regions at the global level while refining trajectory details locally. Furthermore, by incorporating a parallel computing framework, the algorithm efficiently harnesses modern hardware capabilities, dramatically improving planning performance in large-scale environments. An additional obstacle distribution analysis module further optimizes computational efficiency by eliminating redundant evaluations without sacrificing path safety.

This novel methodology provides a groundbreaking technical approach to drone path planning in battlefield conditions. Comprehensive experimental validation demonstrates the algorithm's superior performance in planning speed, path optimality, and resource utilization, establishing a critical foundation for future autonomous combat operations. Beyond its immediate military applications, the underlying principles of this research hold significant potential for adaptation in civilian domains requiring robust path-planning solutions in complex environments.

2. Theoretical Basis

2.1 Jump Point Search

The Jump Point Search (JPS) algorithm fundamentally optimizes the efficiency of traditional grid-based pathfinding by leveraging the geometric regularity of the environment. Unlike the conventional A* algorithm, which examines

neighboring nodes individually, JPS intelligently skips large numbers of symmetric paths by identifying key turning points—known as jump points—where the path direction may change. This mechanism relies on two critical strategies: pruning rules and forced neighbor analysis.

During horizontal or vertical movement, the algorithm continuously jumps along the current direction until encountering an obstacle or meeting the forced neighbor condition. Forced neighbors are adjacent nodes that necessitate a directional change due to obstacle blockages, triggering the generation of jump points. In diagonal movement, the jumping process checks both horizontal and vertical pruning conditions to ensure no critical turning points are missed. This jump propagation mechanism allows JPS to drastically reduce the number of processed nodes, particularly excelling in structured environments.

Therefore, in the JPS algorithm, finding forced neighbors is very important. The following illustration demonstrates how to identify forced neighbors.

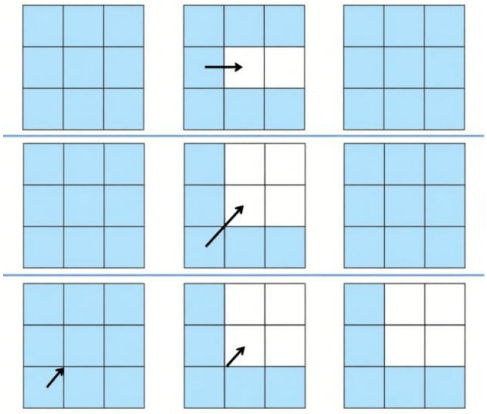


Figure 1.Neighbor situations corresponding to movement directions without nearby obstacles

We visualize a 3×3×3 voxel grid as three 3×3 two-dimensional layers, from left to right: the bottom layer, the middle layer, and the top layer. The center node, indicated by the black arrow, is currently being expanded. The natural neighbors of the current node are marked in white. Pruned neighbors are marked in light blue. The black arrow also shows the direction traveled from its parent node. The figure demonstrates all three cases from bottom to top: (1) straight line, (2) 2D diagonal, and (3) 3D diagonal.

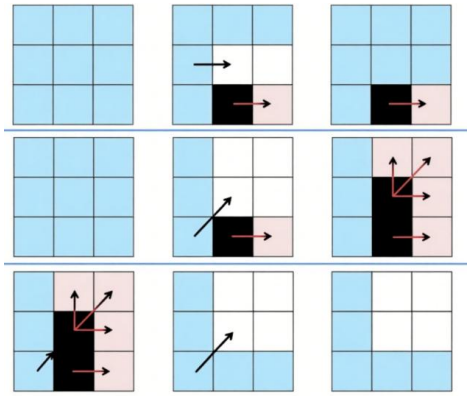


Figure 2.Neighbor situations corresponding to movement directions with nearby obstacles

When the current node is adjacent to an obstacle (black), the highlighted forced neighbors (pink) cannot be pruned. Red arrows indicate the obstacle and its corresponding forced neighbor pairs: if the tail voxel is occupied, then its head voxel is a forced neighbor. For example, in the first case (straight-line movement), if voxel (0, 1, 0) is occupied, then (1, 1, 0) is a forced neighbor. In the second case (2D diagonal movement), an occupied voxel (0, 0, 1) results in three forced neighbors, which is also the scenario in the third case.

When the current node is adjacent to an obstacle (black), the highlighted forced neighbors (pink) cannot be pruned. Red arrows indicate the obstacle and its corresponding forced neighbor pairs: if the tail voxel is occupied, then its head voxel is a forced neighbor. For example, in the first case (straight-line movement), if voxel (0, 1, 0) is occupied, then (1, 1, 0) is a forced neighbor. In the second case (2D diagonal movement), an occupied voxel (0, 0, 1) results in three forced neighbors, which is also the scenario in the third case.

The three-dimensional extension (JPS3D) further enhances the algorithm's applicability by expanding forced neighbor analysis. In 3D pathfinding, planning must account for both planar directional changes and height variations. JPS3D introduces vertical jumping logic, enabling the algorithm to identify voxels requiring directional shifts across the x, y, and z axes. For instance, when a drone ascends or descends and encounters an aerial obstacle, the algorithm marks key lateral movement nodes through forced neighbor analysis. Additionally, movement involving both horizontal and vertical components requires evaluating pruning conditions in all three dimensions to maintain path optimality(Luo Y,2022).

To preserve directional consistency, JPS3D applies traditional JPS pruning rules on each horizontal plane, while cross-layer movement triggers jumps only when height changes are unavoidable. This layered pruning strategy, combined with a 3D distance heuristic, effectively balances search efficiency and path quality.

However, standard JPS3D still faces challenges in large-scale environments. Unrestricted search depth in open areas can lead to excessive computational load—in vast, unobstructed spaces, the algorithm may degrade into a near brute-force search, diminishing its performance benefits. Additionally, inefficient voxelized obstacle representation limits practicality: dense voxel grids demand high memory, while sparse representations may sacrifice path accuracy. Another issue is the lack of hierarchical spatial organization, making it difficult to handle ultra-large-scale scenarios.

Potential improvements include dynamic jump depth limits to optimize open-area searches, octrees or sparse hash grids for flexible voxel representation, and hierarchical path planning strategies combining coarse global guidance with local refinement. These optimizations could enhance JPS3D's real-world applicability in complex environments like battlefields.

2.2 GeoSOT

GeoSOT (Geographic Spatial Organization Tool) establishes a global hierarchical grid system, providing a unified framework for organizing and indexing geospatial data. Unlike traditional latitude-longitude coordinate systems, the core concept of GeoSOT lies in recursively partitioning the Earth's surface into multi-level grid cells, enabling seamless spatial resolution from

global scale down to centimeter-level precision. This mechanism relies on two key design principles: integer encoding rules and hierarchical spatial relationships. (Hu X,2014)In planar projections, the system employs Hilbert curves or Z-order curves to convert two-dimensional coordinates into one-dimensional codes, ensuring that spatially adjacent entities maintain continuity in their encoding. The subdivision of grid cells follows a quadtree principle, where each parent cell is uniformly divided into four child cells. This recursive structure allows any region to be represented with an appropriately refined grid. During encoding, the system preserves level information through bitwise operations, enabling rapid spatial queries to locate target levels—particularly efficient for large-scale geospatial data analysis.

The three-dimensional implementation (GeoSOT3D) further extends the system's applicability by incorporating an octree-based partitioning of the vertical dimension(Zhai W,2017). In 3D scenarios, spatial objects require not only precise planar positioning but also handling the complexities of elevation data. GeoSOT3D integrates the height axis into its recursive subdivision framework, forming cubic voxel grids that uniformly represent vertical structures such as atmospheric layers and underground spaces. For instance, when planning flight paths for drones, the system can quickly retrieve obstacle distributions at different altitudes through voxel encoding. Additionally, indexing spatiotemporal dynamic data (e.g., moving object trajectories) requires combining spatial and temporal dimensions(Zhai W,2019). The system achieves efficient organization of four-dimensional data by using composite keys of timestamps and spatial codes. To maintain encoding consistency, GeoSOT3D applies the planar GeoSOT partitioning rules on horizontal planes while adjusting vertical subdivision granularity based on application requirements, ensuring rational spatial resolution. This multi-dimensional unified encoding, combined with space-filling curve techniques, effectively balances data storage and retrieval efficiency.

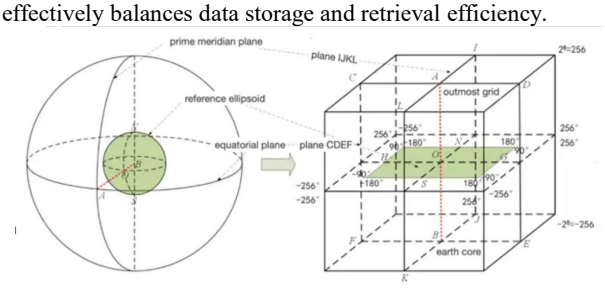


Figure 3. The partitioning pattern of GeoSOT in three-dimensional space.

The hierarchical grid structure of GeoSOT effectively addresses many of the challenges faced by JPS3D in large-scale environments. For the issue of unrestricted search depth in open areas, GeoSOT's multi-level encoding mechanism dynamically adjusts grid resolution, automatically switching to coarse-grained levels in open spaces to avoid unnecessary fine-grained traversal, thereby significantly reducing computational complexity. Moreover, GeoSOT's recursive voxel representation naturally aligns with sparse data structures such as octrees, enabling efficient memory compression while ensuring pathfinding precision through hierarchical transitions—overcoming JPS3D's trade-off between voxel representation memory usage and computational efficiency. For ultra-large-scale path planning requirements, GeoSOT's globally unified encoding system supports hierarchical path search strategies: first planning global routes on coarse-grained grids, then refining adjustments in local regions, thereby balancing efficiency and optimality. This inherent spatial hierarchical characteristic makes GeoSOT an ideal foundational framework for enhancing the practical application of JPS3D in complex scenarios such as battlefield simulations and smart city planning.

3. GPJ3D-RP Algorithm Design

3.1 Overall Algorithm Framework

This algorithm adopts a phased processing architecture to solve large-scale 3D path planning problems. The system input is a voxelized 3D environment map, and the original problem is decomposed into multiple parallelizable subtasks through spatial grid partitioning. The core processing flow consists of three main stages: the spatial grid partitioning stage uniformly divides the environment into multiple sub-grids according to GeoSOT rules; the local path computation stage uses the JPS algorithm to analyze the reachability between the eight vertices within each sub-grid; the global planning stage integrates the reachability information of all sub-grids to construct a simplified navigation graph for the final path solution.

In terms of data flow design, the system maintains three core data structures: first, a GeoSOT-encoded grid spatial index for quickly locating the sub-grid to which any spatial position belongs; second, a global reachability matrix that stores the connectivity status and path length between all adjacent vertex pairs. This hierarchical processing model retains the capability for precise path search while ensuring the algorithm's scalability through data dimensionality reduction; third, a block information table that records the GeoSOT codes, spatial ranges (which can be indirectly derived from GeoSOT codes), and obstacle rates (the proportion of obstacle voxels to the total) of the sub-grids divided during parallel computation. This table is generated during the parallel computation preparation phase and provides decision-making basis for dynamic subdivision.

3.2 GeoSOT-Based Spatial Decomposition

The spatial decomposition module employs the GeoSOT global subdivision grid system as its theoretical foundation. This system recursively divides 3D space into multi-level grids, with each grid uniquely identified by a 64-bit code. In the specific implementation, a fixed-level grid (e.g., level 7) is used as the basic processing unit to ensure all sub-grids have the same physical dimensions.

The core advantage of GeoSOT coding lies in its spatial computability: adjacency relationships can be directly derived from grid codes through bitwise operations. For example, obtaining the adjacent grid in the positive X-axis direction can be achieved by adding 1 to the code. This characteristic facilitates subsequent parallel task allocation and global graph construction. In terms of storage, linearized Z-order curves are used to arrange grid data, improving memory access locality(Zhai W,2017)..

When it is detected that the obstacle rates of all sub-grids under the same parent node are below a set threshold (e.g., 5%), these sibling nodes are merged into a parent grid for processing. After merging, only the eight vertices of the parent grid participate in the computation for this region, significantly reducing the computational load in open areas. The merged status is marked by the high-order bits of the GeoSOT code and automatically identified during path search.

3.3 Parallel Processing for Local Path Search

The local path computation phase employs the CUDA parallel architecture, with each sub-grid assigned to a thread block for processing. Within the thread block, 28 threads are used to compute all pairwise combinations (28 pairs in total) of the eight vertices of the sub-grid in parallel. Each thread independently executes the JPS algorithm, with the search scope strictly confined to the boundaries of the current sub-grid.

The implementation of the JPS algorithm includes standard 3D jump rules: straight jumps, diagonal jumps, and body diagonal jumps. To improve memory efficiency, each thread caches the obstacle information of the current sub-grid in shared memory. Reachability results are compressed and stored in bitmap form, with a single 32-bit integer capable of recording the connectivity status of all vertex pairs.

For extremely large grids, dynamic task scheduling can be implemented based on the obstacle rates in the block information table to reduce computational pressure. Fewer thread resources are allocated to grids with low obstacle rates, and the vertex reachability computation within the thread block is achieved through a combination of serial and parallel processing.
- Very low obstacle rate grids (<2%): Only 1 thread is allocated, processing all vertex reachability in parallel.
- Low obstacle rate grids (<10%): 1/7 of the threads are used, with each thread processing 7 vertex pairs (4 threads).
- Medium obstacle rate grids (10%-30%): 1/4 of the threads are allocated, with each thread processing 4 vertex pairs (7 threads).
- High obstacle rate grids (>30%): The maximum number of threads is allocated, with each thread processing 1 vertex pair (28 threads).

3.4 Global Path Integration Method

Global path planning is performed based on the constructed simplified navigation graph. The nodes of the graph correspond to the vertices of each sub-grid, and the edge weights come from two parts: the JPS computation results within the sub-grid and the Euclidean distance between vertices of adjacent sub-grids. The classic A* algorithm is used to search for the initial path on the global graph.

In terms of path optimization, the current implementation includes basic path splicing functionality: sequentially combining the JPS path segments within each sub-grid with the straight-line connection segments across grids. For advanced optimization features such as path smoothing, only the algorithm interface is currently reserved, with the specific optimization logic yet to be implemented.

The environment update mechanism currently supports a complete recalculation process: when map changes are detected, all sub-grids in the affected area are marked, and their reachability data is uniformly updated in the next computation cycle. The incremental update algorithm is still in the design phase and is not included in the current implementation.

4. Simulation Experiment Design and Results Analysis

4.1 Experimental Environment Setup

4.1.1Hardware Platform Configuration: The following hardware configuration is adopted in this experiment to ensure computational efficiency and simulation accuracy:
Processor: 13th Gen Intel(R) Core(TM) i5-13500H 2.60 GHz
GPU: NVIDIA GeForce RTX 4050 Laptop GPU
Video Memory: 6G

4.1.2Software Environment Configuration: Operating System: Windows 11 Home Chinese Edition
Programming Language: C++14
Development Tool: Visual Studio 2022
Parallel Computing Engine: CUDA 12.6

4.1.3 3D Battlefield Model Construction: Terrain Source Data:
a. OSM-format data of randomly selected streets in Hong Kong, China, from the website OpenStreetMap
b. Random obstacle maps generated based on 3D data and obstacle occupancy rates using random numbers
Data Processing: The OSM file is converted to OBJ format using the built-in data processing tool osm2World from the OSM format. The processed OBJ format is then converted to binvox format and read as the obstacle map for the experiment.
Processed Obstacle Map:

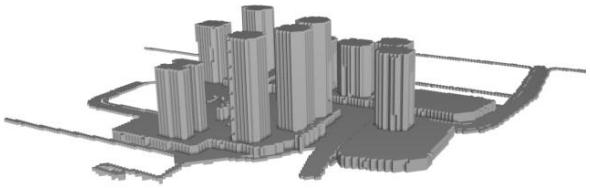


Figure 4. Obstacle Map.

4.2 Experimental Design

4.2.1 Comparison of JPS Algorithm and A* Algorithm in Small-Scale Grids:To verify the superiority of JPS and preliminarily test how to select parallel grid parameters for quickly and accurately finding paths in large-scale maps, this experiment sets the following three objectives: compare the computational efficiency (time performance) of JPS (Jump Point Search) and A* algorithms in grid maps; analyze the differences in path lengths generated by the two algorithms (solution quality); and explore the impact of map size, obstacle density, and heuristic functions on algorithm performance.

The following data are used as experimental independent variables: algorithm type (JPS/A*); map size ($16\times16\times16$, $32\times32\times32$, $64\times64\times64$, $128\times128\times128$); obstacle density (10%, 20%, 30%, 40%); heuristic function (Manhattan distance/Chebyshev distance). The search time and obtained path length are used as dependent variables to evaluate the performance superiority of the algorithm under these settings.
The calculation formula for the Manhattan distance between two points($P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$):

$$d_{Manhattan} = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2| \quad (1)$$

The calculation formula for the Chebyshev distance:

$$d_{Chebyshev} = max(|x_1 - x_2|, |y_1 - y_2|, |z_1 - z_2|) \quad (2)$$

The calculation formula for theEuclidean distance:

$$d_{Euclidean} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \quad (3)$$

The evaluation algorithm formula combining search time and path length:

$$S = A \times \frac{t}{mapsize^3} + B \times \frac{d}{mapsize} \quad (4)$$

Under the premise of ensuring the same distance between the start and end points, the credibility of the results is ensured by repeatedly conducting experiments under the same independent variables by changing the obstacle distribution of random maps. The following are the result line charts under different independent variables:

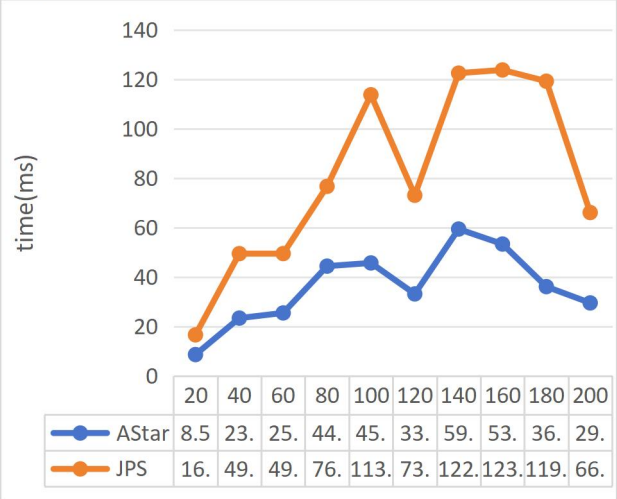


Figure 5. Evaluation Score vs. Map Size

The line chart illustrates the relationship between the evaluation score and map size under the conditions of a fixed obstacle ratio of 0.1 and using Euclidean distance as the heuristic function. The horizontal axis represents the edge length of the obstacle grid map (in grid units), while the vertical axis represents the planning evaluation score calculated according to Formula 4.

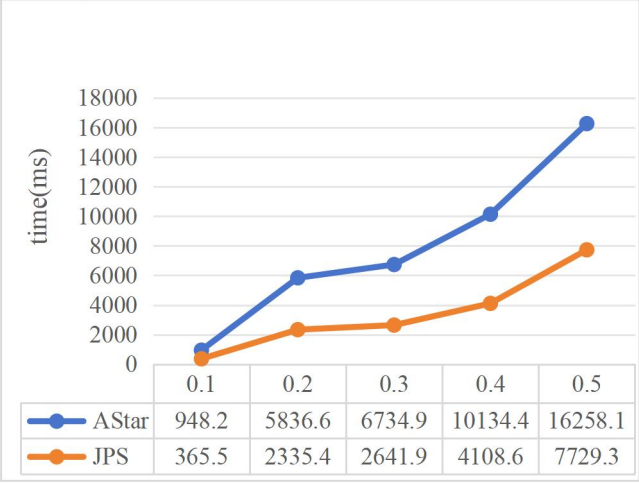


Figure 6. Planning Time vs. Obstacle Ratio

The line chart demonstrates the relationship between planning time and obstacle ratio in a 140 × 140 × 140 grid map using Euclidean distance as the heuristic function. The horizontal axis represents the obstacle ratio of the grid map, and the vertical axis represents the required planning time (in milliseconds).

4.2.2Feasibility Verification of GPJ3D-RP Algorithm and Computational Speed Test Under Different Granularities: To verify the feasibility of the algorithm, the path obtained by the algorithm was visualized, and the resulting image is as follows:

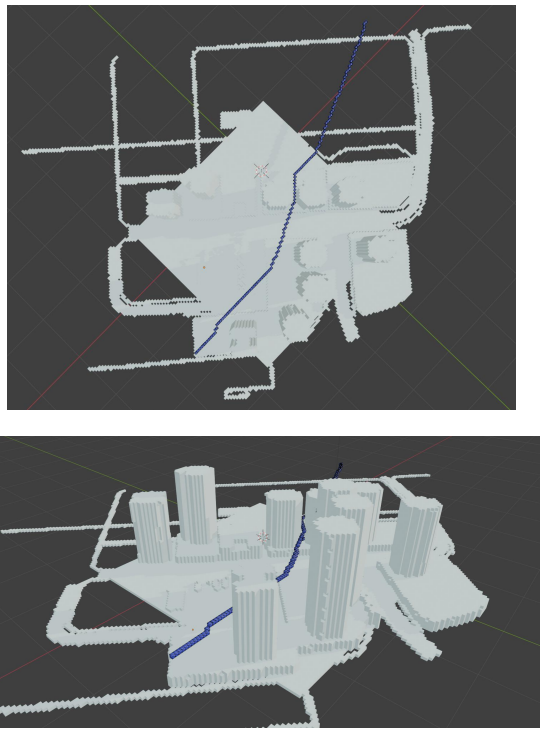


Figure 7. The path planned by the GPJ3D-RP algorithm between two randomly selected points on the map

To test whether the algorithm outperforms conventional serial algorithms in terms of time performance, the planning time of the GPJ3D-RP algorithm is compared with that of the serial JPS algorithm and the A* algorithm under different independent variable settings. The results are as follows:

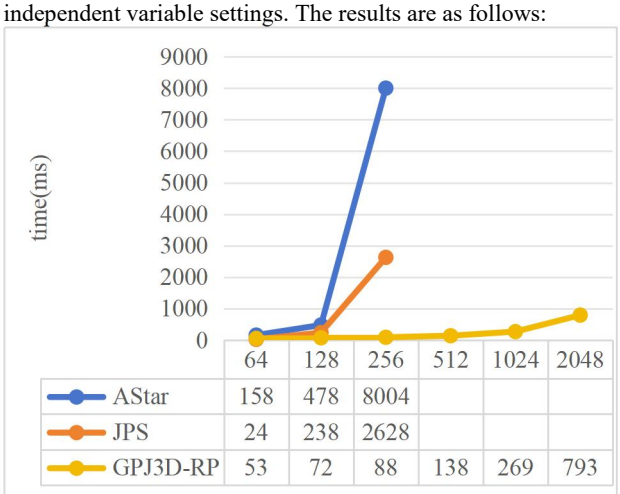
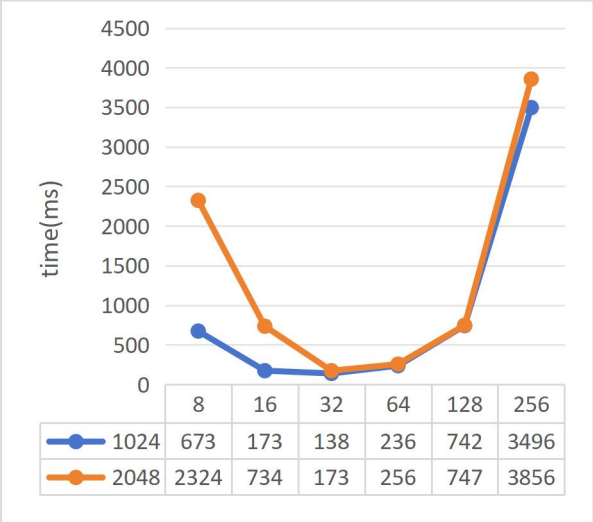


Figure 8. Planning Time Comparison: A vs. JPS vs. GPJ3D-RP.

The line chart compares the path planning time consumption of three algorithms across large-scale grid maps under a fixed obstacle ratio of 0.2 and using Euclidean distance as the heuristic function, with GPJ3D-RP employing sub-grids of size 16×16×16. The horizontal axis represents the edge length of the obstacle grid map (in grid units), and the vertical axis represents the required planning time (in milliseconds). When the map size reaches 512, the planning time of the A* algorithm reaches 131,763 ms, and the JPS algorithm's planning time reaches 45,819 ms, both significantly higher than that of GPJ3D-RP.

To further accelerate the GPJ3D-RP algorithm while ensuring path quality, the results of the GPJ3D-RP algorithm under different granularities are compared. The obtained line charts are as follows:



	8	16	32	64	128	256
1024	673	173	138	236	742	3496
2048	2324	734	173	256	747	3856

Figure 9. GPJ3D-RP Planning Time vs. Sub-grid Granularity

The line chart presents a comparison of path planning times when GPJ3D-RP adopts sub-grids of different granularities under a fixed obstacle ratio of 0.2 and using Euclidean distance as the heuristic function. The horizontal axis represents the sub-grid size, and the vertical axis represents the required planning time (in milliseconds).

4.3 Experimental Results Analysis

This experiment compared the path planning efficiency of the JPS algorithm and the A* algorithm in different grid environments and validated the performance of the GPJ3D-RP algorithm under varying block granularities. The experimental results demonstrate that the JPS algorithm significantly outperforms the A* algorithm in small-scale grids, while the GPJ3D-RP algorithm effectively reduces computation time through its blocking strategy while ensuring path feasibility.

In small grids (20 × 20 × 20 to 160 × 160 × 160), the computation time of the JPS algorithm is approximately 45%~65% shorter on average than that of the A* algorithm. Within this grid range, the advantage of the JPS algorithm gradually increases as the grid size grows. For example, in a 20×20×20 grid, the average computation time of JPS is 4.83 ms, while that of the A* algorithm is 9.34 ms, yielding a speedup ratio of 1.93. Meanwhile, the value of the evaluation function obtained from the ratio of planning time to map size continues to increase, indicating that within this range, the expansion of map size does not significantly impact the speed of the JPS algorithm. However, as the grid size further increases (e.g., 260×260×260), the preprocessing overhead of JPS rises, and the depth of jump point searches increases. Although the computation speed remains much higher than that of the A* algorithm, the operational overhead increases substantially, surpassing the rate of map size expansion. For obstacle environments of varying densities, the JPS algorithm also exhibits strong superiority, with the speedup ratio steadily improving as density increases. In terms of path length, the paths generated by both algorithms are largely consistent, with JPS only showing a 3%~5% increase in path length in a few complex obstacle layouts due to its jumping characteristics, which remains within an acceptable range.

Through the comparative experiments of the JPS and A* algorithms in small grids, it can be concluded that the JPS algorithm outperforms the A* algorithm in path planning across grids of various sizes, with speedup ratios ranging between 1.8 and 3.5. At the same time, the JPS algorithm demonstrates excellent planning speed in grid sizes ranging from 20×20×20 to 160×160×160. However, when the grid size becomes excessively large, the computation speed of the JPS algorithm declines significantly. Therefore, the sub-grid size adopted in parallel grids should not exceed 128×128×128.

The GPJ3D-RP algorithm effectively reduces the computational burden in large-scale environments through its blocking strategy. Experiments show that in a 256×256×256 grid with a 0.2 obstacle ratio, the computation time of the non-blocked JPS algorithm reaches an astonishing 36,593 ms, while the GPJ3D-RP algorithm with 32 × 32 × 32 blocks requires only 106 ms, achieving an efficiency improvement of nearly 60%. The choice of block granularity significantly impacts performance: overly small blocks (e.g., 4×4×4) lead to excessively long computation times for the merging phase and fail to leverage the computational advantages of the JPS algorithm, instead increasing computation time considerably; overly large blocks (e.g., 128×128×128) reduce the potential for parallel optimization. The experimental results indicate that block granularities between 32×32×32 and 128×128×128 achieve the best balance in most scenarios, improving runtime speed by 1-3 orders of magnitude compared to serial JPS and A* algorithms in large-scale environments, while the path length increases by only about 8%~12%.

In summary, the JPS algorithm exhibits a clear time advantage in small to medium-sized grids, while the GPJ3D-RP algorithm maintains high computational efficiency in large-scale environments through reasonable blocking. Future research could further optimize the blocking strategy to reduce the additional overhead caused by inter-block path splicing, thereby enhancing the algorithm's adaptability in dynamic environments.

5. Conclusions and Prospects

5.1 Research Conclusions

This study addresses the practical requirements of UAV path planning in complex battlefield environments by proposing the GPJ3D-RP algorithm based on the GeoSOT grid. Through theoretical analysis and experimental validation, the algorithm has demonstrated significant advancements in multiple aspects.

The research makes an innovative contribution by combining the spatial indexing capabilities of the GeoSOT grid system with the JPS3D algorithm, utilizing multi-resolution grid partitioning and a parallel computing framework to substantially improve computational efficiency in large-scale environments. Furthermore, the introduced hierarchical planning strategy successfully balances path quality with real-time performance, effectively meeting the planning demands of typical battlefield scenarios.

Experimental results confirm that the proposed algorithm outperforms conventional methods in kilometer-scale urban UAV path planning tasks, achieving remarkable gains in computational speed while consistently maintaining high path quality. These outcomes provide a viable technical approach for real-time UAV path planning in dynamic battlefield conditions, marking a meaningful step forward in the field.

5.2 Research Limitations and Future Work

While this study has yielded valuable results, certain limitations warrant further investigation. The current algorithm primarily operates in static environments, and its ability to respond to dynamic obstacles in real time requires additional refinement. The scalability of the method also needs more extensive verification, particularly in ultra-large-scale battlefield environments, to ensure robust performance across diverse operational settings.

Future research efforts will concentrate on enhancing the algorithm's dynamic adaptability, strengthening its practical implementation for seamless integration with military systems, and investigating potential synergies with emerging intelligent algorithms to further elevate planning reliability and efficiency. It is important to emphasize that the practical application of any new technology demands thorough real-world validation. Therefore, the proposed method must undergo rigorous testing in varied operational scenarios to fully assess its effectiveness. Moving forward, we remain committed to a pragmatic and progressive research approach, continuously refining and advancing UAV path planning technologies to meet evolving battlefield requirements.

Acknowledgements

This work was funded by the Natiomal Key Research and Development Program of China(2024YFF1400803).

References

Aggarwal S, Kumar N. Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges[J]. *Computer communications*, 2020, 149: 270-299.

Hu X, Cheng C. The three-dimensional data organization method based on GeoSOT-3D[C]//2014 22nd International Conference on Geoinformatics. IEEE, 2014: 1-4.

Jones M, Djahel S, Welsh K. Path-planning for unmanned aerial vehicles with environment complexity considerations: A survey[J]. *ACMComputing Surveys*, 2023, 55(11): 1-39.

Luo Y, Lu J, Zhang Y, et al. 3D JPS Path Optimization Algorithm and Dynamic-Obstacle Avoidance Design Based on Near-Ground Search Drone[J]. *Applied Sciences*, 2022, 12(14): 7333

Wang G, Guo L, Duan H, et al. A Hybrid Metaheuristic DE/CS Algorithm for UCAV Three-Dimension Path Planning[J]. *The Scientific World Journal*, 2012, 2012(1): 583973.

Zhai W, Qi C, Cheng C, et al. Spatial data management method with GeoSOT grid[C]//2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS). IEEE, 2017: 5217-5220

Zhai W, Tong X, Miao S, et al. Collision detection for UAVs based on GeoSOT-3D grids[J]. *ISPRS international journal of geo-information*, 2019, 8(7): 299.

.

.