Peer-to-Peer Networking and Applications (2025) 18:160
https://doi.org/10.1007/s12083-025-01968-z

OSIS: Obstacle-Sensitive and Initial-Solution-first path planning

Kaibin Zhang[1] · Liang Liu[1] · Wenbin Zhai[1] · Youwei Ding[2] · Jun Hu[1]

Received: 30 November 2023 / Accepted: 25 March 2025

Abstract
Informed path planning is a type of algorithm that uses problem-specific knowledge, expressed as heuristics, to efficiently discover an optimal path between a start and a goal state. The primary challenge in optimizing informed planning algorithms lies in quickly finding the initial solution while minimizing collision checking costs. In this paper, an Obstacle-Sensitive and Initial-Solution-first path planning algorithm (OSIS) is proposed. OSIS leverages historical collision check results to construct an asymptotically accurate distribution of obstacles in space. Based on this distribution, OSIS employs a reusable, inadmissible, yet more accurate heuristic that applies to the entire problem domain. Additionally, an initial-solution-first path optimization strategy is proposed to eliminate unnecessary path optimization. It ensures that OSIS prioritizes exploring uncharted spaces, leading to faster initial solution discovery. Experiments have demonstrated that OSIS outperforms existing algorithms in navigating around obstacles, and achieving convergence in solution cost. Experimental data show that OSIS can even improve the success rate of collision checking to more than 90% in the planning problems studied in this paper, which far exceeds the performance of other algorithms.

Keywords Sampling-based path planning · Optimal path planning · Informed search · Collision check · Obstacle avoidance

1 Introduction

A path planning algorithm is a computational method designed to determine a sequence of valid states from an initial start location to a desired goal within a defined state space, while avoiding any obstructions present within the space. Path planning algorithms have been applied in various fields such as robotics [1], unmanned aerial vehicles (UAVs) [2], autonomous vehicles [3], and video games [4]. The objective of the planner is to promptly find an initial solution and progressively converge it towards an optimal solution.

Existing algorithms can be classified into two primary categories: search-based path planning and sampling-based path planning. Search-based path planning algorithms, such as Dijkstra's algorithm [5], A* [6], D* Lite[7], and the latest work like MOPBD* [8] use dynamic programming techniques [9] to find solutions in discrete state spaces.

The solution quality of search-based path planning algorithms depends on the granularity of space discretization. Finer granularity improves solution quality but increases computational overhead. In high-dimensional state spaces, this overhead grows exponentially with the number of states, a phenomenon known as the *curse of dimensionality* [10].

To overcome the limitations of the search-based algorithm in continuous space, the sampling-based path planning algorithm has been developed.

The RRT [11] algorithm is one of the most classical sampling-based path planning algorithms, and RRT* [12] is an asymptotically optimal improvement of it.

This article is part of the Topical Collection: *1- Track on Networking and Applications*
Guest Editors: Vojislav B. Misic

✉ Kaibin Zhang
zhangkaibin@nuaa.edu.cn

✉ Liang Liu
liangliu@nuaa.edu.cn

✉ Wenbin Zhai
wenbinzhai@nuaa.edu.cn

✉ Jun Hu
hujun@nuaa.edu.cn

Youwei Ding
ywding@njucm.edu.cn

[1] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China

[2] School of Artificial Intelligence and Information Technology, Nanjing University of Chinese Medicine, Nanjing, China

Published online: 24 April 2025

Informed RRT* [13] introduces the informed search, which uses known solutions to limit the optimal solution's range, thereby enhancing algorithm performance by reducing the sampling and search space.

Modern algorithms for informed path planning include Batch Informed Trees (BIT*) [14], Advanced BIT* (ABIT*) [15], Adaptively Informed Trees (AIT*) [16], and Effort Informed Trees (EIT*) [17].

Path planning algorithms often incur significant collision check overhead [18, 19], which should be minimized. However, existing informed algorithms do not effectively use collision information, leading to redundant checks.

Meanwhile, most existing informed algorithms prioritize path optimization through rewiring before finding the initial solution. This slows the discovery of the initial solution, wastes computational resources, and increases the risk of becoming trapped in local optima.

In response to the limitations of existing informed planning algorithms, this paper introduces the Obstacle-Sensitive and Initial-Solution-first (OSIS) path planning algorithm. OSIS leverages collision check results from previous search processes to estimate the distribution of obstacles in the environment. This information helps OSIS maneuver around obstacles and minimizes unnecessary collision checks, thereby enhancing search efficiency. Furthermore, OSIS prioritizes the discovery of the initial solution as its primary objective. To achieve this, OSIS defers rewiring processes that do not directly contribute to finding the initial solution until after the initial solution is found. This strategic approach allows OSIS to quickly locate the initial solution and ignore extraneous paths.

The contribution of this paper is as follows.

- An Obstacle-Sensitive and Initial-Solution-first informed path planning algorithm (OSIS[1]) is proposed to cope with the disadvantages of existing algorithms.
- OSIS estimates the distribution of obstacles in space by analyzing historical collision check data. The planner then employs this information to navigate the environment efficiently, evading obstacles when possible and minimizing the incidence of collisions.
- OSIS optimizes the existing rewiring strategy to prioritize exploration of uncharted regions within the space. It enables the planner to navigate rapidly away from potential local optima, thereby accelerating the search for the initial solution.

The remainder of this paper is organized as follows: Section 2 provides an overview of the related work in the field of path planning algorithms. Section 3 presents the foundational knowledge relevant to informed planning algorithms, along with the mathematical model of the planning problem. Section 4 details the implementation of OSIS. The experimental results are presented in Section 5. Finally, Section 6 concludes this paper, summarizing the key findings.

[1] This paper is an extended version of the poster paper accepted by The 29th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2023).

2 Related work

2.1 Classical sampling-based path planning algorithms

The RRT [11] algorithm is considered one of the pioneers of sampling-based path planning. It works by randomly sampling in space and growing a spanning tree based on these random samples to find solutions. Since RRT growth is random and disordered, it is not guaranteed that RRT will find the optimal solution.

RRT-Connect [20] improves upon RRT by growing two trees from the start and goal, finding a path when the trees connect. RRT-Connect significantly outperforms RRT, but remains non-asymptotically optimal. However, its bidirectional search strategy has influenced many algorithms, such as AIT* [16] and EIT* [17], and inspired asymptotically optimal variants [21–24].

RRT* [12] uses a rewiring technique to ensure asymptotic optimality by updating the cost-to-come for tree nodes. However, its random growth makes it inefficient in finding the optimal solution.

RRT and RRT* utilize random geometric graph (RGG) [25] theory to approximate the planning space by generating random samples in the space, which form graphs with implicit edges that are used to find solutions. Since RRT and RRT* generate only one sample in each iteration, the construction and search of the RGG graph are performed simultaneously, resulting in a randomized and unordered anytime search.

Fast marching trees (FMT*) [26] also employ RGG theory by generating a fixed number of samples at a time to construct an RGG, which is then searched for solutions. FMT*'s construction and search are not simultaneous, resulting in an ordered but non-anytime search, which means that it cannot return a solution at any point during the search.

2.2 Heuristic-based path planning algorithms

Sampling-based planning algorithms can use heuristics to guide the planner's search and improve the planner's performance, such as Sampling-based A* (SBA*) [27]. Heuristics improve search efficiency by utilizing problem-specific information, often in the form of a heuristic function that estimates the cost of connecting any pair of vertices in the graph. Those states that have lower heuristic cost-to-go and

cost-to-come are generally more likely to optimize the solution.

A heuristic is considered admissible if it never overestimates the true cost, such as the Euclidean distance. Conversely, an inadmissible heuristic is one that may overestimate the true cost. A suitable heuristic should be both computationally efficient and as accurate as possible in estimating the true cost, and some planners can update the heuristic to improve its accuracy.

Heuristically-Guided RRT (hRRT) [28] is a heuristic-based variation of RRT, which employs heuristics to direct the growth of the tree. Although these algorithms improve RRT's performance, they lack the ability to constrain the search space based on existing solutions.

FMT* does not use heuristics, and the Motion Planning using Lower Bounds (MPLB) [29] is an anytime adaption of FMT* that incorporates heuristics, but it couldn't update the heuristics dynamically.

2.3 Informed path planning algorithms

Gammell et al. introduced Informed RRT* [13] as an extension of RRT*, which led to the development of the concept of informed planning. In informed planning, the optimal path length can be bounded by an ellipse (ellipsoid) defined by the length of the current path and the Euclidean distance between start and goal. This approach allows the planner to narrow the range of optimal solutions and eliminate states and edges that cannot improve the solution, resulting in improved search efficiency. However, Informed RRT* does not use heuristics to guide the search.

Depending on the sampling pattern, the planner can be either a single-sampling planner, which generates one sample at a time, or a batch-sampling planner, which generates a batch of samples at once. The batch sampling planning algorithm can build an approximation of the planning space faster, so as to find the initial solution faster. FMT* can also be viewed as a batch sampling algorithm, but FMT* only generates a batch of samples in a planning process. BIT * [14] can generate samples in multiple batches, so BIT * can dynamically build a more and more accurate RGG approximation.

ABIT* improves the performance of BIT* by introducing advanced graph search techniques. ABIT* incorporates two factors, namely the inflation factor (ε_{infl}) and the truncation factor (ε_{trunc}), to guide its search. The ε_{infl} factor inflates the heuristic cost-to-go of edges, prioritizing the exploration of edges with lower heuristic cost-to-go, so edges closer to goal will be added to the tree first, which greatly improves the efficiency of the search. On the other hand, the ε_{trunc} factor truncates the search, allowing the planner to initiate the subsequent round of more precise search at the earliest opportunity.

However, the heuristics of BIT* and ABIT* do not update when a collision is detected, which makes the accuracy of their heuristics plummet in some cases. Figure 5 (a) and (b) illustrate a scenario where BIT* and ABIT* exhibit poor performance. Additionally, BIT* and ABIT* adopt a "greedy" strategy to select the edge that is most likely to improve the solution in the current situation for addition to the tree, which makes them susceptible to getting trapped in a local optimum.

AIT* leverages bidirectional search to compute a more precise heuristic. It constructs a forward tree rooted at the start and a reverse tree rooted at the goal, with the latter ignoring collisions with obstacles during its growth. The cost-to-go heuristic of a state in the forward tree is equal to its cost-to-come in the reverse tree. At each iteration, AIT* tries to add the edge in the reverse tree that connects to the forward tree and is most likely to improve the solution to the forward tree, and if this edge collides with an obstacle, AIT* updates the reverse tree using LPA* [30] . LPA* is an enhancement of the A* algorithm that is designed to handle dynamic scenes, where the positions of obstacles may dynamically change. Unlike A*, LPA* does not require a complete re-search after detecting a change in the environment, instead, it updates only the relevant local path to quickly recompute the optimal path.

However, the reverse tree of AIT* does not take into account the collision with obstacles during its growth process, leading to frequent updates of the reverse tree in complex scenarios, as shown in Fig. 5 (c). As a result, while AIT* can compute more accurate heuristics, the cost of achieving such accuracy may be substantial.

EIT* is an extension of AIT* that addresses its limitations through the use of effort heuristics and sparse collision checks. The effort heuristic is an inadmissible heuristic that estimates the computational effort needed to verify a path from a state to a goal, based on the path length and collision check resolution. By incorporating effort heuristics and sparse collision check, EIT* alleviates the problem of frequent updates of the reverse tree that may result from AIT* not considering collisions at all. The performance of EIT* has shown a significant improvement compared to AIT*. However, as illustrated in Fig. 5 (d), EIT* does not effectively address the frequent "bumping into the wall" issue.

Although AIT* and EIT* can compute more accurate heuristics, the computational overhead of such heuristics may be huge in some complex scenarios. In this case, these algorithms actually transfer the overhead of the forward search to the reverse search, but the overall search overhead is not significantly reduced.

Existing informed algorithms do not effectively use past collision data to guide the search. Moreover, these algorithms apply the rewiring strategy for path optimization before locating the initial solution, which not only hinders initial solution discovery but also leads to unnecessary overhead by optimiz-

ing numerous irrelevant paths. This can potentially prolong the process and trap the planner in local optima.

Based on these problems that exist in existing informed programming algorithms, in this paper OSIS, a novel informed planning algorithm that can efficiently identify and bypass obstacles in space, find an initial solution and converge to it, is proposed.

3 Problem modeling and preliminaries

3.1 Problem modeling

The definition of the path planning problem in this paper is similar to that in [12]. Let $X \subseteq \mathbb{R}^n$ represent the state space, $X_{obs} \subset X$ represent the space occupied by obstacles, and $X_{free} \subset X \backslash X_{obs}$ represent the free space without obstacles. In the path planning problem definition, it is generally assumed that the planner does not know the specific distribution of X_{obs} and X_{free}. The planner usually needs to confirm whether a state belongs to X_{obs} or X_{free} through expensive collision checks.

Let $x_{start} \in X_{free}$ be the start state and $X_{goal} \subset X_{free}$ be the set of goal states. Assume $\sigma : [0, 1] \mapsto X_{free}$ is a continuous function that possesses finite total variation, which is equivalent to a valid path. The set of all valid paths is denoted by Σ. Assuming an optimization objective, the cost function $s : \Sigma \mapsto [0, \infty)$ is defined to map each path to a non-negative real number.

The task of the optimal path planning problem is to either find the path with minimum cost, $\sigma^* \in \Sigma$, from the x_{start} to any $x_{goal} \in X_{goal}$, or report failure if there is no such path. The cost of the optimal path is denoted s^*. σ^* is defines as:

$$\begin{aligned} \sigma^* := \underset{\sigma \in \Sigma}{\arg\min} \{ & s(\sigma) | \sigma(0) = x_{start}, \sigma(1) \in X_{goal}, \\ & \forall t \in [0, 1], \sigma(t) \in X_{free} \} \end{aligned} \tag{1}$$

3.2 Preliminaries

Informed planning algorithms, such as BIT* [14] , generate an increasingly dense RGG [25] by batch sampling. The RGG is composed of a set of states, $X_{samples} \subset X$, which are randomly generated uniformly, and these states constitute the vertexes in the graph. The edges between the vertex are implicit. After finding a solution, the informed planning algorithm can limit the range of the optimal solution to an ellipse (or ellipsoid) determined by the solution. The set of states within this ellipse (or ellipsoid) is known as the informed set. The informed planning algorithm searches for a solution by growing a tree rooted at x_{start} on this edge-implicit graph. To grow this tree, every time a new vertex is added to the tree, this new vertex is expanded (the root node x_{start} is expanded at the beginning of the algorithm). When expanding a vertex, the planner considers all the implicit edges between the neighbors of the expanded vertex and the expanded vertex in RGG, and selects the implicit edge that is most likely to improve the solution based on the heuristic. If the child state of this implicit edge is already in the tree, the edge is considered as a rewiring edge. There are two main methods for determining whether a vertex is a neighbor: *k-nearest* [31] and *r-disc* [32]. The *k-nearest* policy defines a state's neighbors as the k nearest states to that state, where k is defined as:

$$k(q) = \eta e \left(1 + \frac{1}{d}\right) log(q) \tag{2}$$

The *r-disc* policy defines the neighbors of a state as all the states within a radius of r around it, where r is defined as:

$$r(q) = 2\eta \left(1 + \frac{1}{d}\right)^{\frac{1}{d}} \left(\frac{\lambda(X_{\hat{f}})}{\zeta_d}\right)^{\frac{1}{d}} \left(\frac{log(q)}{q}\right)^{\frac{1}{d}} \tag{3}$$

In Eqs 2 and 3, q is the number of states in the informed set, d is the dimension of the planning space, $\eta \geq 1$ is the tuning parameter, $\lambda(X_{\hat{f}})$ is the Lebesgue measures of the informed set and ζ_d is an d-dimensional unit ball.

After obtaining the implicit edges between the expanded state and its neighbors, the planner needs to evaluate the potential of these implicit edges to improve the solution, in order to select the most promising edges to add to the tree. The informed planner utilizes a priority queue, or min-heap, of lexicographically ordered keywords based on heuristic costs to prioritize pending implicit edges. The planner then takes the best edge from the priority queue and performs a series of checks (e.g., collision check) on it to decide whether to add it to the tree.

4 OSIS

4.1 Basic idea

OSIS constructs an obstacle density approximation by collecting statistics on the results of collision checks and uses this information to calculate an inadmissible but more effective heuristic. This heuristic prioritizes exploration of the free space and avoids ineffective exploration, resulting in improved performance. In addition, OSIS improves the rewiring strategy to avoid inefficient path optimization prior to finding the initial solution. This approach enables the planner to focus on the search for the initial solution. Using these

strategies, OSIS effectively addresses the challenges faced by existing informed planners, resulting in faster and more efficient discovery and convergence of the initial solution.

The obstacle sensitivity of OSIS is based on the fact that if collisions are frequently detected in a certain region of the planning space, it means that this region is likely to be occupied by obstacles, and the planner should choose to avoid this region as much as possible or reduce the priority of exploration in this region. In order to measure the proportion of detected collisions occurring in a region, OSIS partitions the planning space into multiple subspaces based on user-defined parameters. Each subspace is associated with an obstacle density, denoted by ρ, where $\rho \in [0, 1]$, which represents the fraction of the space occupied by obstacles within that subspace. The density of obstacles in the subspace will be calculated dynamically according to the results of the collision check during the execution of the algorithm.

Figures 1 and 2 demonstrate how OSIS finds a solution through the obstacle-sensitive inadmissible heuristic and how it outperforms the admissible heuristic. Where the left blue square represents the start, the right red square represents the goal, the black point represents the sample, the gray rectangle represents the obstacle, the blue edge represents the search tree, the red edge represents the collision edge, and the green path represents the solution. As illustrated in Fig. 2, the planner utilizing the admissible heuristic repeatedly attempts to progress directly toward the goal but frequently collides with the obstacle. In contrast, as shown in Fig. 1, OSIS uses information from previous collisions, such as those encountered during sampling, to proactively navigate around obstacles.

OSIS achieves initial-solution-first by delaying rewiring. Before finding the initial solution, OSIS records all potential rewiring edges encountered and does not process them until the initial solution is found.

OSIS mainly uses two priority queues, sorted by heuristic cost, to delay processing potential colliding edges (PCE) and potential rewiring edges (PRE) that may not improve the solution: the PCEs queue Q_C and PREs queue Q_R. Since a PCE is likely to collide with an obstacle, it has a low probability of improving the solution. If an edge is both a PCE and a PRE, it will first be treated as a PCE. Furthermore, the normal edge queue Q of OSIS stores edges that are neither PCEs nor PREs. In general, the edges in Q will be processed preferentially compared to the edges in Q_C and Q_R.

Figure 3 shows the sequence in which OSIS processes edges. PREs are not processed until an initial solution is found, and OSIS starts processing PREs immediately after finding an initial solution. In each batch, OSIS will give priority to the normal edges, while the PCEs that may not be able to optimize the solution will be processed last. If all edges in a batch can not significantly improve the current solution, then processing the delayed PCE may not yield much benefit either. In such situations, one can consider completely discarding PCEs or discarding them before finding an initial solution to speed up the planner to start the next batch for a more refined search. After finding an initial solution, OSIS will no longer delay PREs, but it will still delay PCEs.

The utilization of PCEs queue and PREs queue enables OSIS to prioritize edges that have a higher probability of improving the solution, enables OSIS to find the initial solution faster, and thereby enhancing the search efficiency and convergence speed of the planner.

OSIS mainly has three key components: getting the best edge, trying to add the best edge to the tree, exhausting the current approximation, and they will be described in detail in Sections IV-C, IV-D, and IV-E, respectively.

Algorithm 1 $OSIS(x_{start}, X_{goal}, m)$.

1: $V \leftarrow \{x_{start}\}$; $E \leftarrow \emptyset$; $T \leftarrow (V, E)$;
2: $X_{samples} \leftarrow \{X_{goal}\}$;
3: $V_{closed} \leftarrow \emptyset$; $V_{inconsistent} \leftarrow \emptyset$;
4: $Q_R \leftarrow \emptyset$; $Q_C \leftarrow \emptyset$;
5: $Q \leftarrow Expand(\{x_{start}\})$;
6: $\varepsilon_{infl} \leftarrow \infty$; $\varepsilon_{trunc} \leftarrow 1$;
7: $is\_search\_done \leftarrow False$;
8: $is\_final\_search\_on\_batch \leftarrow False$;
9: $has\_exact\_solution\_ \leftarrow False$;
10: **repeat**
11: **if** is_search_done **or** $Q \equiv \emptyset$ **then**
12: **if** $is\_final\_search\_on\_batch$ **or**
13: **not** $has\_exact\_solution$ **then**
14: $ProcessPotentialEdges(Q_C)$;
15: $Prune()$;
16: $V_{closed} \leftarrow \emptyset$;
17: $X_{samples} \xleftarrow{+} Sample(m)$;
18: $Q \leftarrow Expand(\{x_{start}\})$;
19: $\varepsilon_{trunc} \leftarrow UpdateTruncationFactor()$;
20: $is\_final\_search\_on\_batch \leftarrow False$;
21: **else**
22: $\varepsilon_{infl} \leftarrow UpdateInflationFactor()$;
23: $Q \xleftarrow{+} Expand(V_{inconsistent})$;
24: $V_{inconsistent} \leftarrow \emptyset$;
25: $is\_final\_search\_on\_batch \leftarrow True$;
26: $is\_search\_done \leftarrow False$;
27: **else**
28: $(x_p, x_c) \leftarrow \underset{(x_i, x_j) \in Q}{\arg\min} \{key_{OSIS}(x_i, x_j)\}$;
29: **if not** $has\_exact\_solution$ **and** $x_c \in V$ **then**
30: $Q_R \xleftarrow{+} (x_p, x_c)$;
31: **else**
32: $is\_search\_done \leftarrow TryAddBestEdge((x_p, x_c))$;
33: **until** stop

4.2 Initialization

Initially, there are no edges in the tree, but only x_{start}, which represents the root node of the tree *(line 1 in Alg. 1)*. The sample set $X_{samples}$ is initialized with X_{goal} *(line 2 in Alg. 1)*, where $X_{goal} \subset X$ denotes goal states.

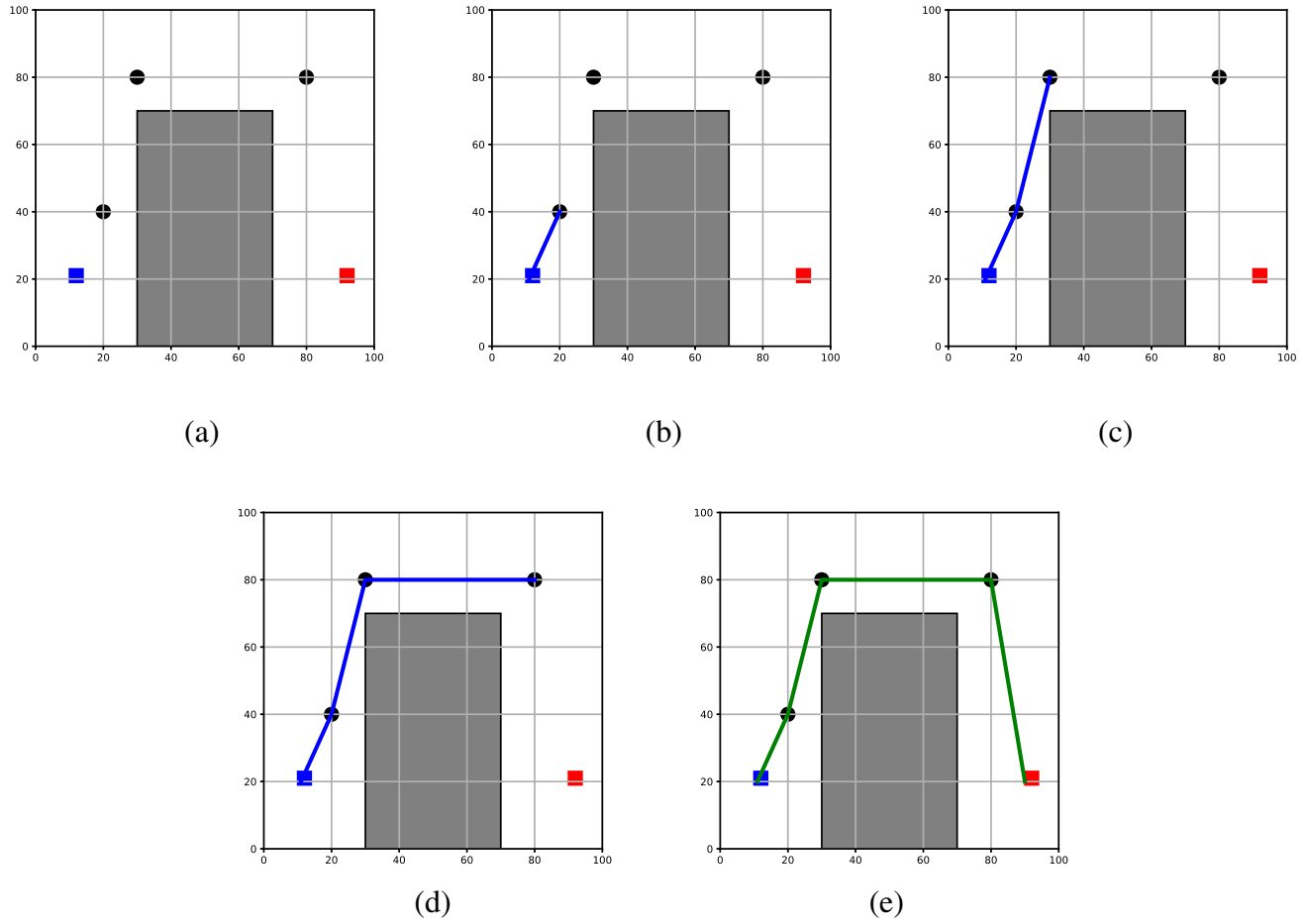


Fig. 1 Example of obstacle-sensitive inadmissible heuristic search

OSIS then takes candidate edges that can be added to the tree by expanding x_{start}. Alg. 2 shows how OSIS expands the set of states to be extended: X_e. For each state x_p in X_e, OSIS first checks whether it is possible to optimize the solution through x_p *(line 3)*, where $\hat{f}(x)$ represents an admissible estimate of the cost to reach the goal from the start, passing through state x, defined as $\hat{f}(x) := \hat{g}(x) + \hat{h}(x)$; $\hat{g}(x)$ represents the heuristic cost-to-come of state x, serving as an admissible estimate or lower bound on the cost of reaching state x from the start; $\hat{h}(x)$ represents the heuristic cost-to-go of the state x, providing an admissible estimate or lower bound on the cost of reaching the goal from state x. If it is possible to optimize the solution through x_p, OSIS obtains its neighbors based on the *k-nearest* or *r-disc* strategy *(line 4)*. For a neighbor x_c of x_p, if the solution may be optimized by edge (x_p, x_c) *(line 5-6)*, where $\hat{f}((x, y))$ represents the heuristic cost of reaching the goal from the start, while passing through the edge (x, y), which defined as $\hat{f}((x, y)) := \hat{g}(x) + \hat{c}(x, y) + \hat{h}(x)$; $\hat{c}(x, y)$ represents the heuristic cost of the edge (x, y), providing an admissible estimate of the cost associated with traversing from state x to state y. Finally, OSIS determines whether it is a PCE, if so, it is added to the PCE queue Q_C, otherwise it is added to E_{out}, which represents the set of candidate edges. *(line 7-10)*

Algorithm 2 $Expand(X_e)$.

1: $E_{out} \leftarrow \emptyset$;
2: **for all** x_p **in** X_e **do**
3: **if** $\hat{f}(x_p) \leq \min_{x \in X_{goal}} \{g_T(x)\}$ **then**
4: **for all** x_c **in** $neighbors(x_p)$ **do**
5: **if** $\hat{f}((x_p, x_c)) \leq \min_{x \in X_{goal}} \{g_T(x)\}$ **then**
6: **if** $\hat{g}(x_p) + \hat{c}(x_p, x_c) \leq g_T(x_c)$ **then**
7: **if** $IsPCE((x_p, x_c))$ **then**
8: $Q_C \xleftarrow{+} (x_p, x_c)$;
9: **else**
10: $E_{out} \xleftarrow{+} (x_p, x_c)$;
11: **return** E_{out}

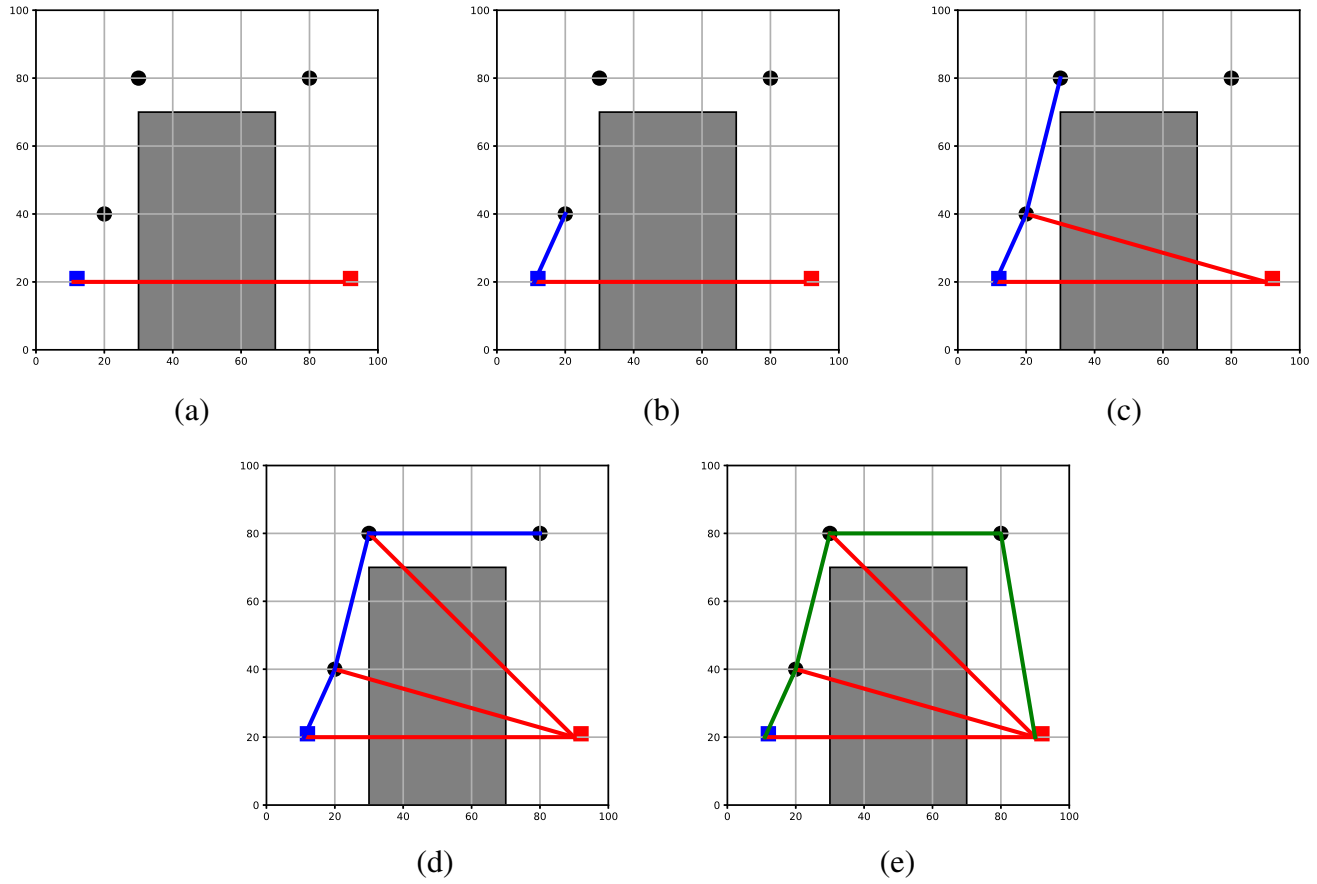


Fig. 2 Example of admissible heuristic search

OSIS, similar to ABIT*, incorporates advanced graph search techniques to enhance search efficiency. Formally, the inflation factor, $\varepsilon_{infl} \geq 1$, is initialized to infinity *(line 6 in Alg. 1)*, and it is initialized to a very large number during the actual execution of the algorithm.

The Boolean variable is_search_done indicates whether a search round is complete, $is\_final\_search\_on\_batch$ indicates whether a batch is complete.

4.3 Get the best edge

In this step, OSIS obtains the edge from the edge queue that is most likely to improve the solution. If the best edge is PRE and the initial solution has not been found, then PRE will be delayed; otherwise, OSIS will try to add it to the tree.

If the search under the current approximation is not finished, OSIS takes an edge (x_p, x_c) from the edge queue Q that is most likely to improve the solution *(line 28 in Alg. 1)*. OSIS sorts edges lexicographically in terms of sorting keys, where the edge with the smallest key value is treated as the best edge. For an edge (x_p, x_c), the sort key is defined as Eq. 4:

$$key_{OSIS}(x_p, x_c) = \Big(\rho(x_p, x_c)\big(g_T(x_p) + \hat{c}(x_p, x_c) + \varepsilon_{infl}\hat{h}(x_c)\big), \; g(x_p) + \hat{c}(x_p, x_c), \; g(x_p) \Big) \tag{4}$$

where $g_T(x)$ represents the cost-to-come of state x in a given tree T, which denotes the current cost of coming from the start to state x; $\rho(x_p, x_c)$ represents the collision factor of edge (x_p, x_c), which reflects the possibility of collision between the edge and the obstacle; the higher the value of the collision factor, the more likely the edge is to collide with the obstacle.

Suppose that for a problem domain space X, OSIS partitions it into n mutually disjoint subspaces, i.e. $\bigcup_{i=1}^{n} X_i = X$, and $X_i \cap X_j = \emptyset, i, j \in [1, n], i \neq j$. For an edge (x, y), assuming that its trajectory through space passes through m

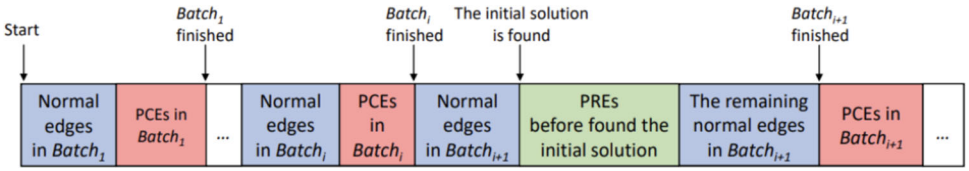


Fig. 3 The order of edges processed by OSIS

subspaces, its collision factor is calculated as follows:

$$\rho(x, y) = \prod_{i=1}^{m} \left(1 + \frac{l_{a_i}^d}{\lambda(X_{a_i})}\right)^{\alpha \rho_{a_i}} \tag{5}$$

where l_{a_i} denotes the length of the edge (x, y) in the i-th subspace that it passes through, $\lambda(X_{a_i})$ denotes the Lebesgue measure of the i-th subspace that (x, y) passes through, $\rho_{a_i} \in [0, 1]$ denotes the obstacle density of the i-th subspace that (x, y) passes through, d denotes the dimension of the planning space, α denotes the obstacle sensitivity parameter, which is used to adjust the sensitivity of OSIS to obstacles. By default, α is set to 1.

The implication of Eq. 5 is that if an edge has a longer trajectory in space or passes through a subspace with a high obstacle density, it is highly likely to collide with obstacles. For the subspace X_{a_i} that the edge (x, y) traverses, if the region traversed by (x, y) in X_{a_i} is sufficiently small or the obstacle density ρ_{a_i} tends to 0, the collision factor of (x, y) in this space tends to 1. OSIS computes the collision factor of (x, y) by multiplying the collision factors of each traversed subspace. When (x, y) passes through an empty area, the collision factor tends to 1, which has a minimal impact on the heuristic cost of (x, y). In contrast, when (x, y) passes through a "crowded" area with frequent collisions, the collision factor of (x, y) becomes significantly larger than 1, inflating its heuristic cost and reducing its processing priority.

Figure 4 builds on Figs. 1 and 2 to demonstrate how OSIS utilizes obstacle density to compute the inadmissible heuristic cost and identify the correct path. In this example, each subspace is sized 20×20. A point(x, y) is mapped to a subspace X_{ij}, where $i = \lfloor x/20 \rfloor$ and $j = \lfloor y/20 \rfloor$. The path AB passes through subspaces X_{14}, X_{24} and X_{34}, while the path AG traverses X_{31}, X_{32}, X_{22}, X_{32}, X_{31}, and X_{41}. The figure also indicates the obstacle density of these subspaces, defined as the proportion of their area occupied by obstacles.

According to Eq. 5, $\rho(A, B) = 1$ and $\rho(A, G) \approx 3.05$ are calculated [2]. Similarly, based on Eq. 4 (for simplicity, assuming $\varepsilon_{infl} = 1$), we have $g_T(A) + \hat{c}(A, B) + \hat{h}(B) \approx 174.42$ and $g_T(A) + \hat{c}(A, G) + \hat{h}(G) \approx 148.44$. Consequently, $key_{OSIS}(A, B)[0] = 1 \times 174.42 = 174.42$ and $key_{OSIS}(A, G)[0] = 3.05 \times 148.44 = 452.742$. As a result, OSIS selects AB over AG, thereby avoiding a guaranteed failed collision check.

4.4 Try to add the best edge to the tree

In this step, OSIS verifies the best edge taken from the edge queue, and if the best edge can indeed optimize the solution and has no collision with obstacles, OSIS adds it to the tree and updates the solution.

After identifying the best edge (x_p, x_c) that has the highest potential to improve the solution, OSIS proceeds to add it to the tree. However, before doing so, OSIS verifies whether x_c is already present in the tree T. If x_c is indeed in the tree, (x_p, x_c) is considered a PRE. In the case where an initial solution has not yet been found, OSIS postpones the processing of this edge. It is then placed in the queue of PREs, and the next iteration begins *(lines 29-30 in Alg. 1)*.

If the best edge does not qualify as a PRE or an initial solution has already been found, the $TryAddBestEdge$ function is invoked to try adding it to the tree. This function returns a Boolean value that indicates whether the best edge is impossible to optimize the solution. If the $TryAddBestEdge$ function returns True, it signifies that optimizing the solution is unfeasible even for the best edges, indicating even a lower potential for optimizing the solution for the remaining edges.

The function $TryAddBestEdge$ first checks if the edge (x_p, x_c) is already present in the tree. If (x_p, x_c) is found in the tree and x_c has not been expanded during the current search, OSIS includes x_c in V_{closed}, signifying that x_c has been expanded during the current approximation, then OSIS expands x_c, and the function returns False *(lines 1-7 in Alg. 3)*. If x_c has been expanded in the current search, it is added to the set $V_{inconsistent}$ as an inconsistent state and will not be expanded further.

Then OSIS checks whether the edge (x_p, x_c) is likely to improve the solution *(lines 8-9 in Alg. 3)*. In doing so, the truncation factor ε_{trunc} inflates the cost of the solution that can be obtained by (x_p, x_c) under the current approximation. This requires that the best edge (x_p, x_c) must significantly

[2] AG passes through 6 subspaces with their obstacle density $\rho_a = [0.25, 0.5, 1, 0.5, 0.5, 0]$, and the trajectory length of AG in each subspace is $\sqrt{200}$, and the Lebesgue measure of each subspace is $20 \times 20 = 400$. So $\rho(A, G) = \prod_{i=1}^{6} \left(1 + \frac{200}{400}\right)^{\rho_{a_i}} \approx 3.05$. The same goes for $\rho(A, B)$.

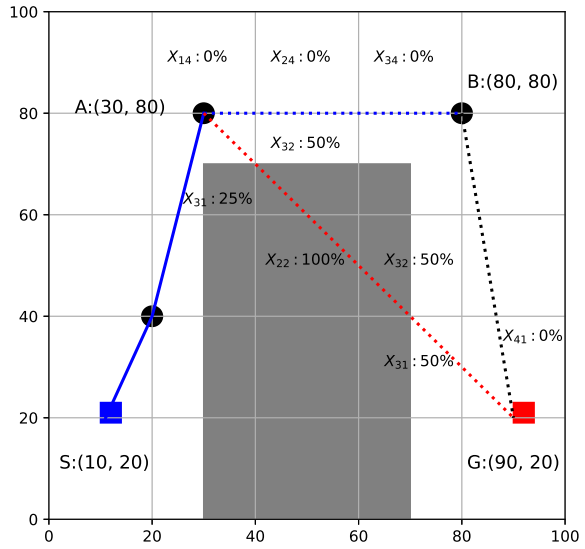


Fig. 4 How does OSIS use obstacle density to find the correct path

improve the solution for the best edge to be considered to improve the solution, otherwise $TryAddBestEdge$ returns $True$, indicating that none of the remaining edges can improve the solution.

When the initial solution has not been found yet, the value of $\min_{x \in X_{goal}} \{g_T(x)\}$ is infinite, then the inequality of *line 8* and *line 11* in *Alg. 3* will always be true, and then all the best edges (x_p, x_c) are regarded as can optimize the solution. This approach leads to two issues: (i) All PREs are regarded as can optimize the solution, however, since no new state is added to the tree, it is not possible to update the solution even by adding it to the tree. (ii) Any rewiring edge will be treated as useful, even if it is not on the optimal path. Due to these reasons, rewiring in the absence of an initial solution does not provide any apparent benefits to finding the initial solution. In contrast, it may result in an unnecessary search. Consequently, OSIS defers rewiring until an initial solution is found.

If a heuristic cost based on the edge (x_p, x_c) indicates that it may be able to improve the solution, it is collision detected to compute its true cost and verify whether it is indeed likely to improve the solution *(lines 10-12 in Alg. 3)*, where $c(x, y)$ represents the true cost of the edge (x_p, x_c).

OSIS dynamically updates the obstacle density based on the collision check results. When a collision is detected for a state, the obstacle density of the corresponding subspace is increased, indicating a higher proportion of obstacles. Conversely, if no collision is detected, the obstacle density of the subspace is decreased, reflecting a lower obstacle presence.

The best edge (x_p, x_c) is added to the tree if it does not collide with the obstacle and can improve the solution. If

Algorithm 3 $TryAddBestEdge((x_p, x_c))$.

1: **if** $(x_p, x_c) \in E$ **then**
2: **if** $x_c \in V_{closed}$ **then**
3: $V_{inconsistent} \xleftarrow{+} x_c$;
4: **else**
5: $Q \xleftarrow{+} Expand(\{x_c\})$;
6: $V_{closed} \xleftarrow{+} x_c$;
7: **return** $False$;
8: **if** $\varepsilon_{trunc}(g_T(x_p) + \hat{c}(x_p, x_c) + \hat{h}(x_c)) \leq \min_{x \in X_{goal}} \{g_T(x)\}$ **then**
9: **if** $g_t(v) + \hat{c}(x_p, x_c) < g_T(x_c)$ **then**
10: **if** $CheckEdge((x_p, x_c))$ **then**
11: **if** $g_T(x_p) + c(x_p, x_c) + \hat{h}(x_c) < \min_{x \in X_{goal}} \{g_T(x)\}$ **then**
12: **if** $g_T(x_p) + c(x_p, x_c) < g_T(x_c)$ **then**
13: **if** $x_c \in V$ **then**
14: $E \xleftarrow{-} \{(x_{prev}, x_c) \in E\}$;
15: **else**
16: $X_{samples} \xleftarrow{-} x_c$;
17: $V \xleftarrow{+} x_c$;
18: $E \xleftarrow{+} (x_p, x_c)$
19: **if** $x_c \in V_{closed}$ **then**
20: $V_{inconsistent} \xleftarrow{+} x_c$;
21: **else**
22: $Q \xleftarrow{+} Expand(\{x_c\})$;
23: $V_{closed} \xleftarrow{+} x_c$;
24: **if not** $has\_exact\_solution$ **and**
25: $x_c \in X_{goal}$ **then**
26: $has\_exact\_solution \leftarrow True$
27: $ProcessPotentialEdges(Q_R)$;
28: **return** $False$;
29: **return** $True$;

Algorithm 4 $ProcessPotentialEdges(Q_p)$.

1: $is\_processing\_done \leftarrow False$
2: **while** $Q_p \neq \emptyset$ **and not** $is\_processing\_done$ **do**
3: $(x_p, x_c) \leftarrow \arg\min_{(x_i, x_j) \in Q_p} \{key_{OSIS}(x_i, x_j)\}$;
4: $is\_processing\_done \leftarrow TryAddBestEdge((x_p, x_c))$;

x_c is already in the tree, the edge connecting x_c to its previous parent x_{prev} is also removed *(lines 13-18 in Alg. 3)*. When (x_p, x_c) is added to the tree, x_c is expanded if it has not been expanded already in the current search. An initial solution is found when x_{goal} is first added to the tree as x_c. At this point, OSIS immediately starts processing previously delayed PREs, adding those that can improve the solution to the tree *(lines 24-27 in Alg. 3)*. If the best edge fails to improve the solution after further inspection, $TryAddBestEdge$ returns False and proceeds to try the next edge.

4.5 Exhaust the current approximation

If the best edge is impossible to improve the solution, or the edge queue is already empty, OSIS will start the next round

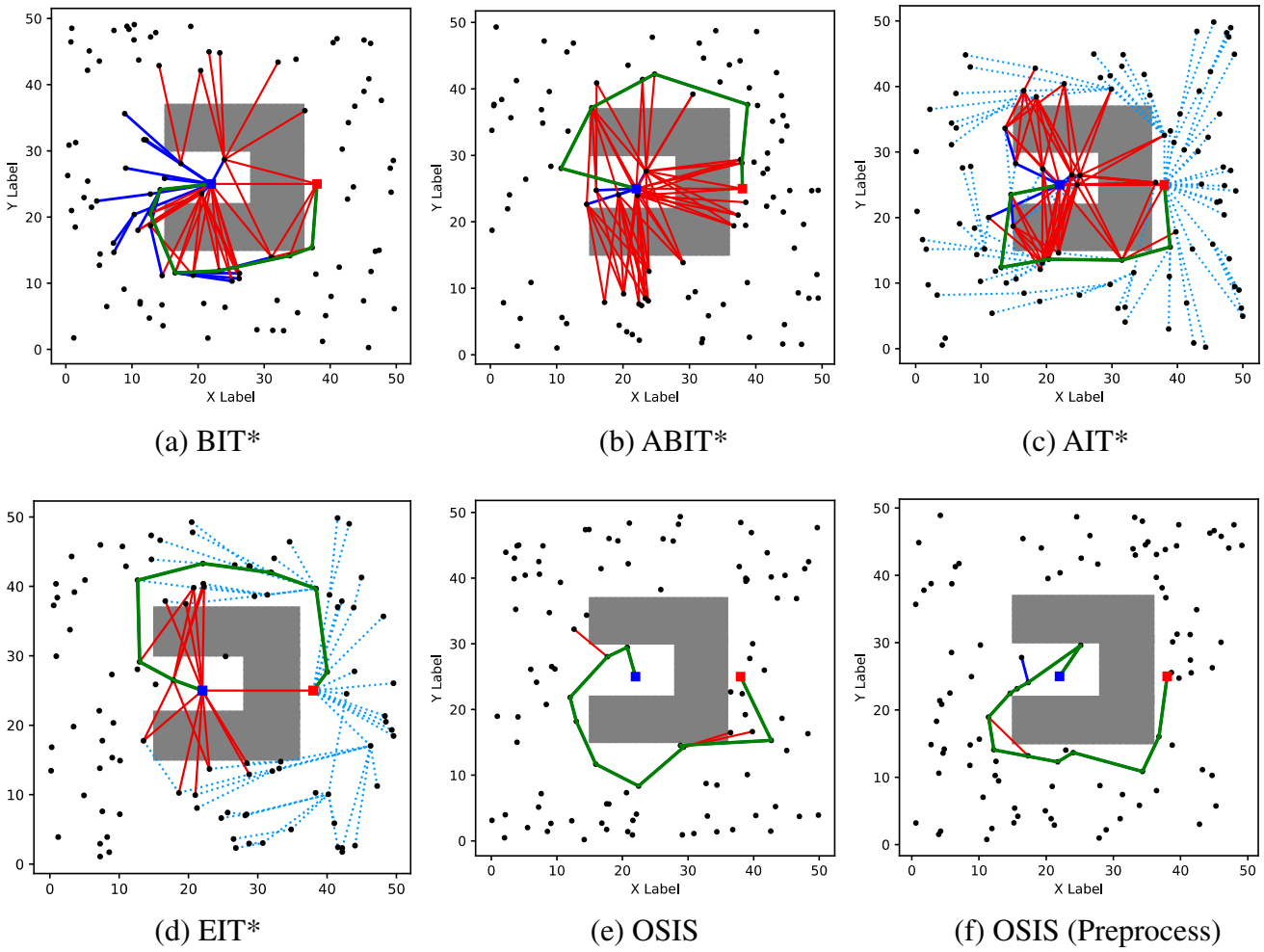


Fig. 5 The growth of the tree in Problem 1 when different planners find the initial solution. Obstacles in the space are represented in gray, while black dots represent the sampling states. The blue square represents start and the red square represents goal. The green lines represent solutions found by the planner. The solid blue lines indicate the edges added to the tree, and the dotted light blue lines in (c) and (d) indicate the edges added to the reverse tree of AIT* and EIT*. The red line represent the edge that the planner tried to add to the tree, but failed because it overlapped with the obstacle and did not pass the collision check. (e) and (f) are both the planning results of OSIS. The difference is that (f) preprocesses the scene before execution to obtain the distribution of obstacles in the space in advance, while (e) gradually collects the results of collision check in the planning process to predict the distribution of obstacles in the space

of search or approximation *(line 11 in Alg. 1)*. If the current approximation is not exhausted, OSIS updates the inflation factor, expands the set of inconsistent states, and starts the final search under the current approximation *(lines 22-25 in Alg. 1)*.

If the current approximation is exhausted or an initial solution has not been found *(lines 12-13 and Alg. 1)*, OSIS proceeds to initiate the next round of approximation. Prior to that, OSIS handles the PCEs that were postponed during the current approximation phase *(line 14 in Alg. 1; Alg. 4)*. Subsequently, the graph is pruned to eliminate redundant states and edges *(line 15 in Alg. 1)*, The definition of Prune is similar to that of research [14]; A new batch of samples is generated and x_{start} is expanded again *(lines 17-18 in Alg. 1)*.

During the sampling process, it is necessary to check whether the generated samples collide with obstacles. OSIS estimates the probability that a sample is valid based on the obstacle density. If the subspace where the sample belongs to has a high density of obstacles, OSIS can directly discard the sample without collision check. Alternatively, OSIS can utilize a priority queue to order the samples to be detected, delaying the check of potential colliding states. Similarly to the edge validation, OSIS also updates the obstacle density based on the result of the sample collision check.

Finally, the truncation factor is updated to ensure a more precise and efficient search in the subsequent round *(line 19 in*

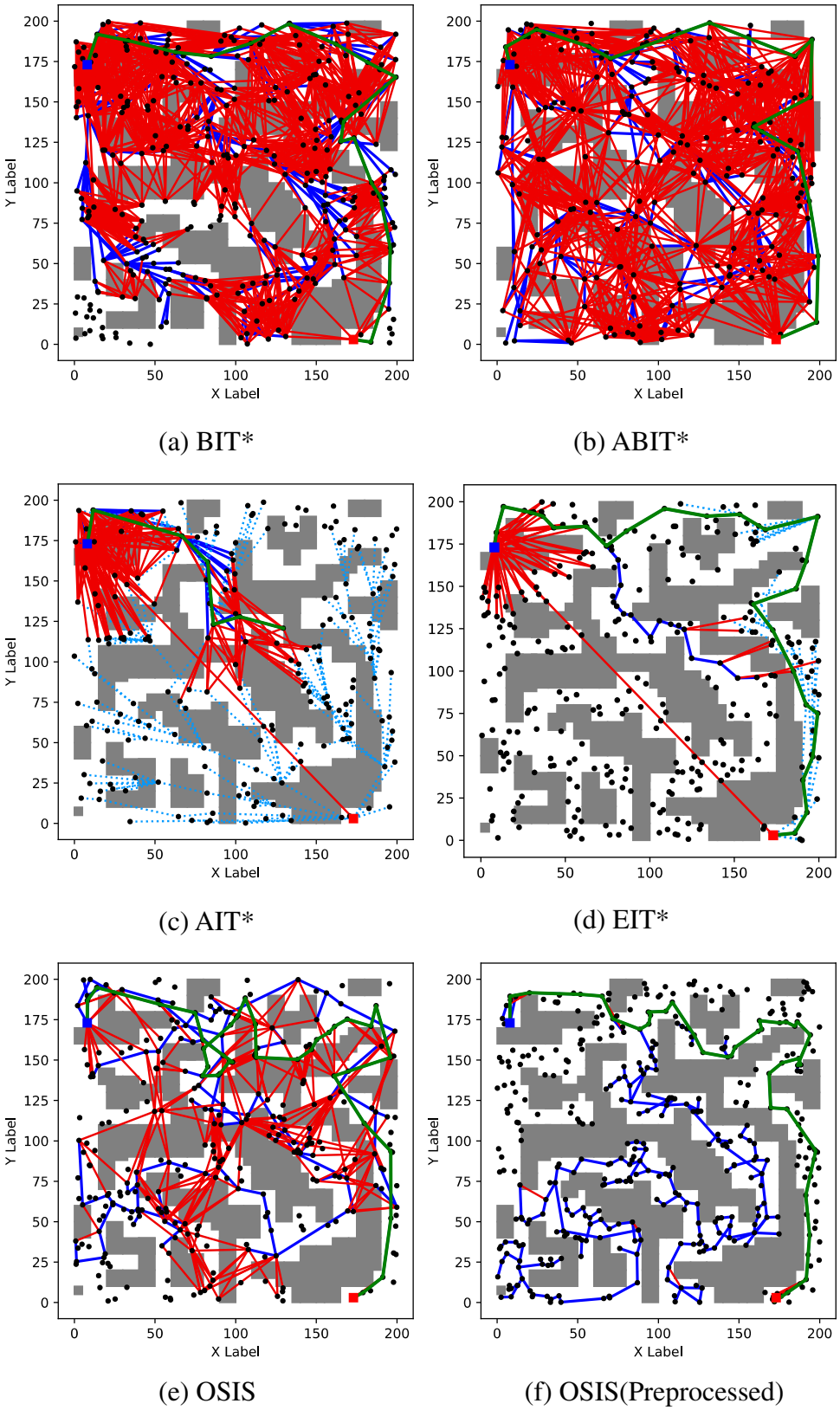
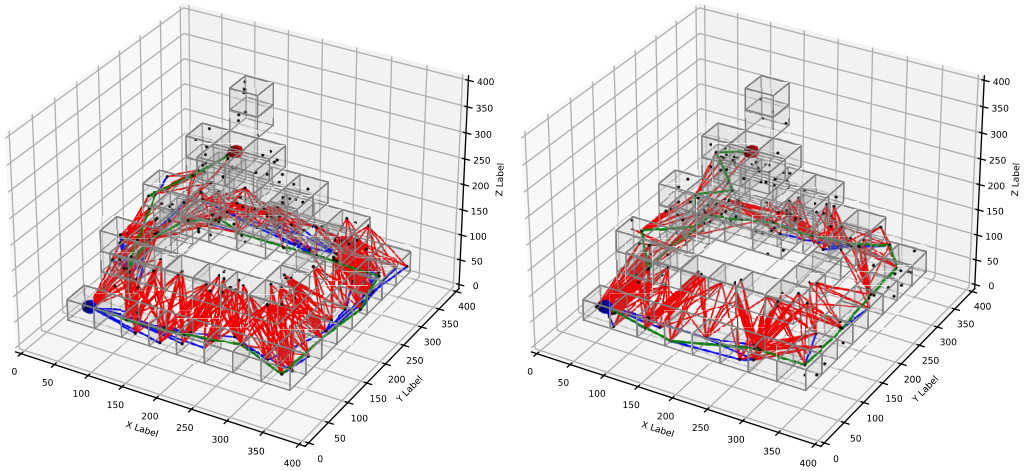
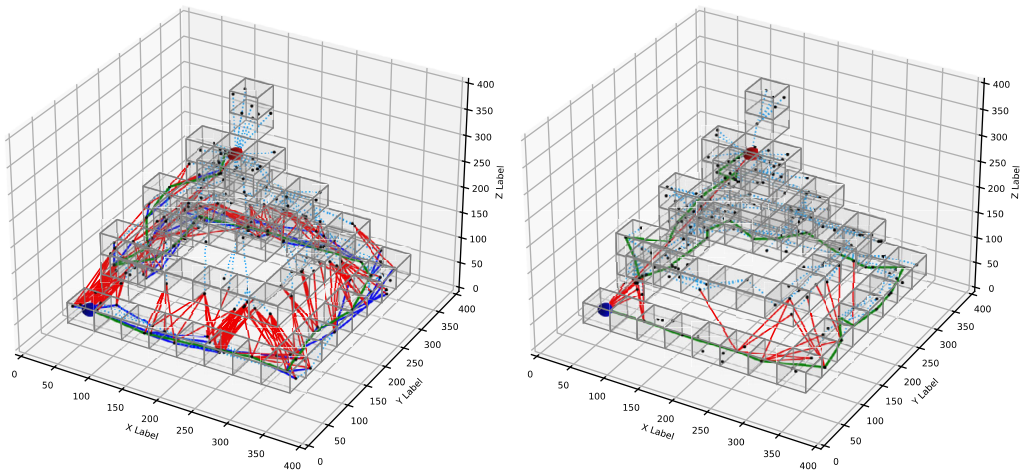


Fig. 6 The growth of the tree in Problem 2 when different planners find the initial solution. The meanings represented by the edges and points of various colors are the same as in Fig. 5



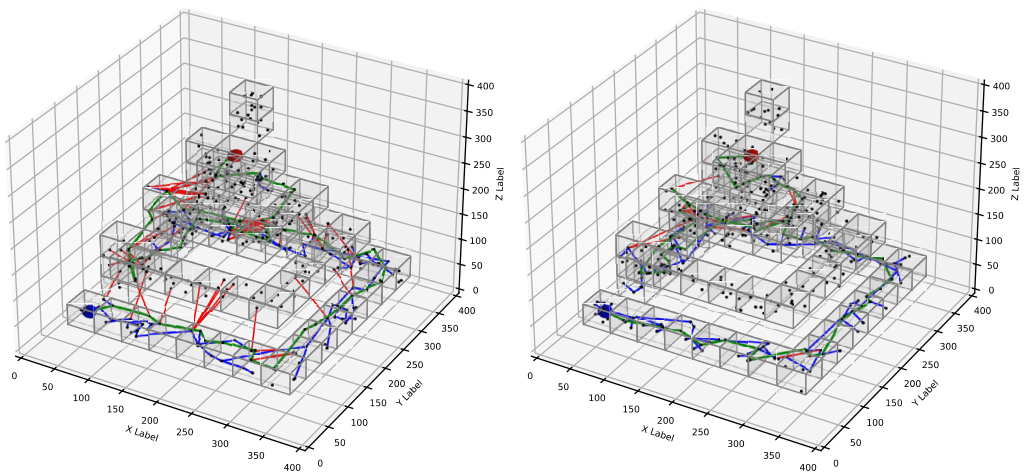
(a) BIT*

(b) ABIT*



(c) AIT*

(d) EIT*



(e) OSIS

(f) OSIS (Preprocessed)

Fig. 7 The growth of the tree in Problem 3 when different planners find the initial solution. The spiral channel is the only free space in this problem

Table 1 Experimental parameters

Problem	Samples per batch	Solve time (ms)	OSIS subspace size	Space size
1	100	1000	5	50
2	100	2000	10	200
3	200	10000	40	400

Alg. 1). The strategy for updating the inflation and truncation factors is similar to that of the study [15].

4.6 Analysis

This section will analyze why the obstacle-sensitive inadmissible heuristic used by OSIS is guaranteed to find the optimal solution. Firstly, in [14] et al., the authors have analyzed and proved that admissible heuristics can guarantee that the planner will converge the solution to the optimal.

According to Eqs. 2 and 3, as the number of samples increases, the growth steps of the search tree progressively decrease. Combined with Eq. 5, the edge collision factor gradually approaches 1, and ε_{infl} in Eq. 4 converges similarly toward 1 as the number of iterations increases. At this point, the heuristic for OSIS will gradually converge from inadmissible to admissible. So, if there is an optimal solution to the planning problem, OSIS must be able to find it as the number of iterations increases.

5 Experimental results

To evaluate the performance of OSIS, we compare it with the Open Motion Planning Library (OMPL) [33] versions of BIT*, ABIT*, AIT*, and EIT*. The performance are measured with OMPL v1.6([3]) on a laptop with 16GB of RAM and an Intel i7-8550U processor.

To intuitively demonstrate the advantages of OSIS over existing algorithms, this experiment includes three designed planning problems, as shown in Figs. 5, 6 and 7. Problem 1 is a simple scenario where the start is semi-surrounded by walls and the planner needs to bypass them to find the goal behind the walls.

Problem 2 presents a more intricate scenario characterized by numerous obstacles and the presence of local optimal paths. In this case, the planner must effectively circumvent the obstacles and quickly escape from local optima.

Problem 3 is a 3D scenario in which the spiral channel is the only free space in the whole planning space, and the traditional heuristic function will face great challenges in this space.

Table 1 shows the settings of the planner parameters for different planning problems [4]. OSIS partitions the planning space into subspaces using a coarse-grained grid. For example, in a two-dimensional space, if the planning space X has a size of n and each subspace has a size of m, the coordinate range of the planning space in each dimension is $[0, n]$. Each point (x, y) is uniquely mapped to a subspace X_{ij}, where $i = \lfloor x/m \rfloor$ and $j = \lfloor y/m \rfloor$.

In addition, for all planners, the RGG tuning parameter η is set to 1.1, the *k-nearest* strategy is used to obtain neighbors, the Euclidean distance is used as the default heuristic, and the objective of optimization is to minimize the length of the path. For ABIT* and OSIS, the initial values of the inflation factor ε_{infl} and the truncation factor ε_{trunc} are 10^6 and 1, respectively. In OSIS, the obstacle sensitivity α is set to 1. When considering any edge (x, y), if its collision factor $\rho(x, y)$ exceeds the threshold of 1.3, it is identified as a PCE.

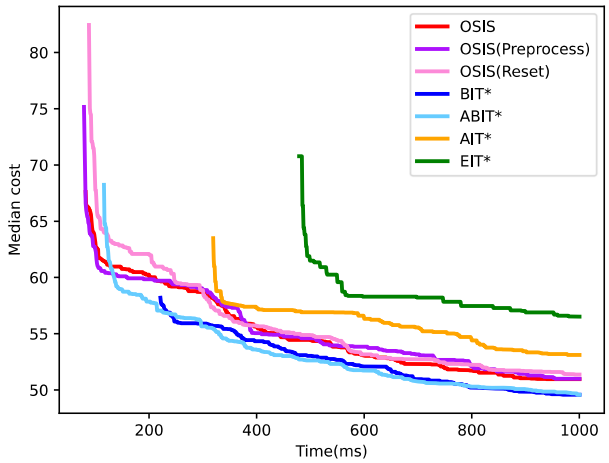
In the experiment, each planner was run 100 times per planning problem using the settings described above. Furthermore, for testing purposes, OSIS is evaluated in three modes, namely:

1. **The default mode:** initially, no obstacle density information is available, but OSIS maintains the obstacle density record after each execution;
2. **The density reset mode:** the obstacle density is reset after each planning, and the next planning is equivalent to planning in a completely new environment;
3. **The pre-processed mode:** prior to planning, the planning space is pre-scanned to acquire a more accurate obstacle density.

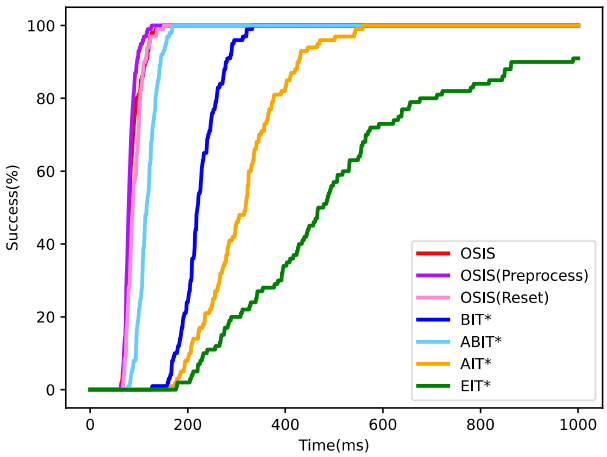
Figure 7 shows the median path cost and the success rate achieved by all test subjects in each planning problem. In Figs. 8 (a), 8 (c) and 8 (e) the lines show the median cost over time of the solution of the almost-surely asymptotically optimal planners. In Figs. 8 (b), 8 (d) and 8 (f), the lines represent the success rate of the planner in finding the initial solution over time.

[3] Due to the removal of the EIT* algorithm in OMPL v1.6, this experiment compares against the EIT* algorithm as implemented in OMPL v1.5.

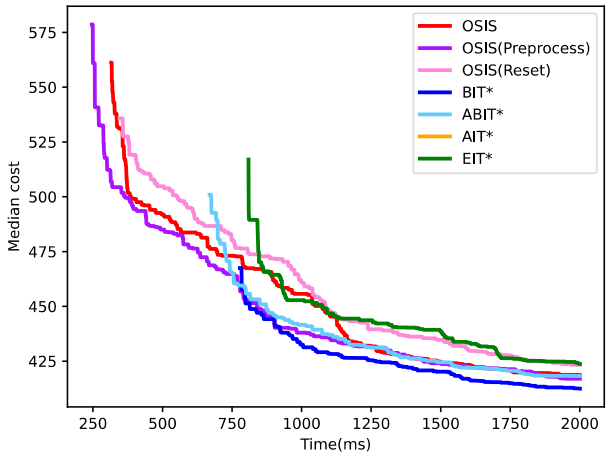
[4] In the OMPL implementation, some planners had a maximum limit on the number of sampling attempts. We adjusted these settings to ensure that each planner could collect an equal number of samples per batch.



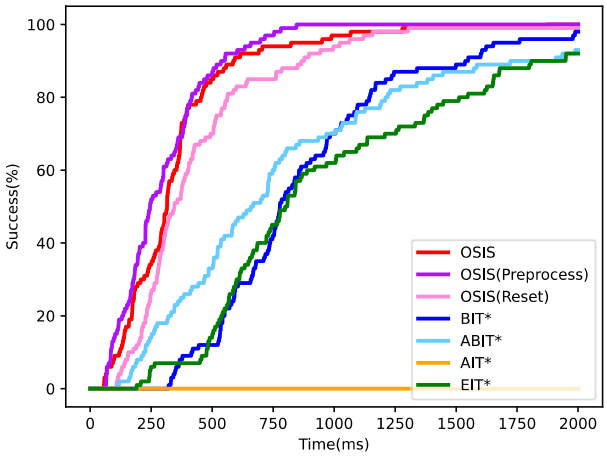
(a) Median path cost of Problem 1



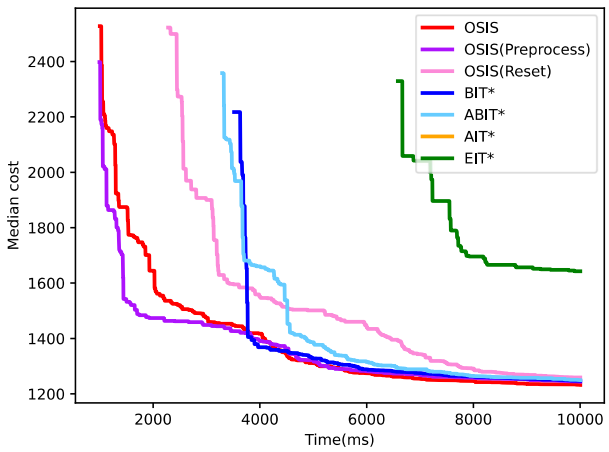
(b) Success rate of Problem 1



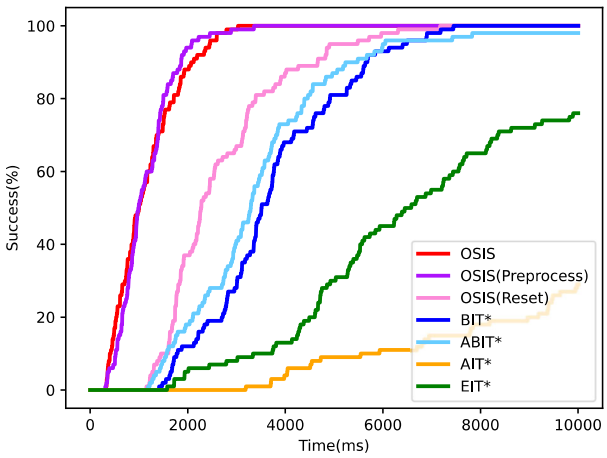
(c) Median path cost of Problem 2



(b) Success rate of Problem 2



(e) Median path cost of Problem 3



(f) Success rate of Problem 3

Fig. 8 Performance of each planner

Table 2 Average performance of each planner when finding the initial solution in Problem 1

Planner	Time (ms)	Edge collision checks	Edge valid rate	State collision checks	State valid rate
BIT*	237.68	99	39.40%	118	84.75%
ABIT*	117.74	66	19.70%	118	84.75%
AIT*	276.68	63	23.81%	116	86.21%
EIT*	471.91	21	42.86%	117	85.47%
OSIS	81.95	15	86.67%	107	93.46%
OSIS (Reset)	88.14	24	70.83%	116	87.07%
OSIS (Preprocess)	81.35	15	86.67%	104	96.15%

Note that in Problem 2, AIT* can hardly find the initial solution within 2 seconds, so there are no data for AIT* in Fig. 7. Meanwhile, Fig. 8 (f) shows that AIT* finds fewer than 50% of the initial solutions within 10 seconds, resulting in no data for AIT* in Fig. 8 (e). This is due to the fact that the scene in Problem 2 and Problem 3 is too complex and there are a large number of obstacles in the space, which causes AIT* need to update the reverse tree frequently and leads to the degradation of the search performance.

As observed in Fig. 8, regardless of the mode used, OSIS outperforms existing informed planning algorithms by achieving faster initial solution finding and convergence to the optimal solution. The advantage becomes more pronounced as the environment becomes more complex and collision check costs increase.

In the simple scenario of Problem 1, there is minimal variation in the performance of different OSIS modes when it comes to finding the initial solution. Among the other planners, only ABIT* performs better.

In the complex scenario of Problem 2 and Problem 3, it is evident that the preprocessed mode and the default mode of OSIS exhibit slightly superior performance compared to the reset density mode in terms of finding the initial solution. This advantage comes from their initial understanding of the distribution of obstacles within the planning space, allowing them to effectively navigate around obstacles from the outset.

Conversely, the reset density mode of OSIS clears the historical collision information before each planning, requiring a period of planning to gather sufficient information and identify potential obstacles. However, even the reset density mode of OSIS significantly outperforms other informed planning algorithms.

Furthermore, it is worth noting that the convergence speed of the OSIS may gradually decreases with time. This is because the search strategy of OSIS drives the planner to stay away from obstacles as much as possible, resulting in some clearance between the path and the obstacles. Considering that in practical applications, the planner is usually required to keep a certain safe distance from the obstacles, so the attenuation of convergence speed is acceptable.

For other informed planning algorithms, ABIT* performs slightly better than BIT*, while EIT* and AIT* suffer performance degradation due to frequent updates of the reverse tree.

Figures 5, 6 and 7 depict the evolution of the search tree during the process of finding the initial solution for Problem 1, Problem 2 and Problem 3, respectively. It should be noted that due to the complexity of Problem 2, AIT* struggles to find the initial solution within the allotted execution time. As a result, Fig. 6 (c) shows the growth of AIT* until the execution time limit is reached, and the green path represents the closest path to the goal found by AIT* after reaching the upper bound of the time, and it can be seen that this is a locally optimal path, which does not actually lead to goal.

Figures 5 (e), 6 (e) and 7 along with Figs. 5 (f), 6 (f) and 7 (f) present the growth of the search tree for OSIS in different settings: without any obstacle density information, and

Table 3 Average performance of each planner when finding the initial solution in Problem 2

Planner	Time (ms)	Edge collision checks	Edge valid rate	State collision checks	State valid rate
BIT*	840.40	1194	20.77%	416	59.38%
ABIT*	810.05	1135	23.70%	404	59.65%
AIT*	–	–	–	–	–
EIT*	855.14	75	40.00%	373	59.79%
OSIS	299.81	229	92.14%	451	79.16%
OSIS (Reset)	496.582	640	46.52%	713	62.41%
OSIS (Preprocess)	297.48	226	95.58%	408	87.5%

Table 4 Average performance of each planner when finding the initial solution in Problem 3

Planner	Time (ms)	Edge collision checks	Edge valid rate	State collision Checks	State valid rate
BIT*	3296.02	808	17.45%	2872	6.96%
ABIT*	3468.96	738	20.05%	3089	6.93%
AIT*	7959.62	2378	21.31%	2378	6.90%
EIT*	5561.04	61	57.38%	2914	6.90%
OSIS	1123.37	159	88.05%	568	70.77%
OSIS (Reset)	2382.98	199	87.44%	3600	12.69%
OSIS (Preprocess)	1014.11	164	88.41%	358	100.00%

with sufficient obstacle density information obtained through preprocessing or multiple planning tasks, respectively.

Based on Figs. 5, 6 and 7, it is evident that OSIS outperforms other informed planning algorithms in terms of collision check, with significantly fewer instances of collision check and a higher success rate. Furthermore, thanks to its optimized rewiring strategy, OSIS exhibits improved efficiency in initial solution search and quicker escape from local optima.

Tables 2, 3, and 4 present the average time taken by each planner to find the initial solution in Problems 1, 2, and 3, respectively, as well as the average collision checking overhead during this process. The table shows that all versions of OSIS find the initial solution more quickly. This is because the edges and states evaluated by OSIS generally have a higher probability of passing collision checks, resulting in fewer necessary collision checks and thus improved performance.

Table 3 indicates that OSIS(Reset) checks more states on average because the limited free space in Problem 3 requires OSIS(Reset) to spend additional time discovering it in each run. Planners other than OSIS do not optimize sampling and sample uniformly across the entire planning space, resulting in similar state validity rates. In contrast, OSIS optimizes the sampling based on obstacle density, leading to a higher state validity rate.

It should be noted that EIT* enhances the quality of the reverse tree by performing sparse collision checks on reverse edges, which are not included in Tables 2, 3, and 4. Figure 9 illustrates EIT* conducting sparse collision checks on edges when the initial solution is found, it shows that EIT* performs a large number of sparse collision checks, which actually makes the search overhead shift from forward search to reverse search, but the overall search overhead does not decrease significantly.

6 Conclusion

OSIS employs two key strategies that contribute to its ability to find and converge the initial solution efficiently. Firstly, it calculates the obstacle density distribution in the planning space by tracking collision check results. Based on this information, it determines the collision factor, which indicates the likelihood of collision between edges and obstacles. This obstacle-sensitive approach enables the planner to efficiently navigate around obstacles.

Secondly, OSIS adopts an initial solution-first path optimization strategy. It delays the rewiring operation, which

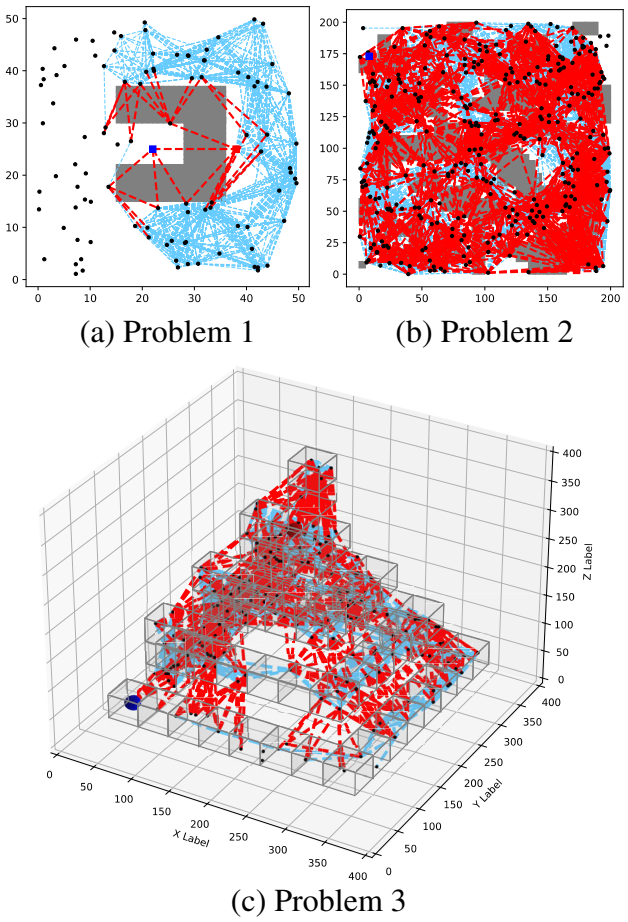


Fig. 9 Edges for sparse collision checking in EIT*. The red edges indicate the edges where the sparse collision check fails, and the blue edges indicate the edges where the sparse collision check succeeds

optimizes the existing path, until the initial solution is found. This approach prevents unnecessary path optimization before finding the initial solution, resulting in faster space exploration.

In addition, it makes OSIS jump faster from the local optimum, to speed up the convergence of the solution. By combining these two strategies, OSIS demonstrates its ability to quickly discover the initial solution and converge efficiently to the optimal solution.

Although OSIS exhibits notable advantages in improving pathfinding efficiency, it also has limitations. Its strong sensitivity to obstacles leads it to actively avoid them, which can hinder its ability to identify paths through narrow passages. Future work will focus on integrating OSIS with algorithms such as RRV [34], allowing OSIS to efficiently avoid obstacles while also effectively discovering narrow passages in the planning space.

Author Contributions Kaibin Zhang: Methodology, Algorithm design, Programming, Formal analysis, Experimental design, Writing - Original Draft, Writing - Review & Editing; Liang Liu: Problem analysis, Methodology, Validation, Funding acquisition, Writing - Review & Editing; Wenbin Zhai: Methodology, Writing - Review & Editing; Youwei Ding: Funding acquisition, Writing - Review & Editing; Jun Hu: Writing - Review & Editing;

Funding This work is supported by the National Key R&D Program of China under No. 2021YFB2700500 and 2021YFB2700502, the Open Fund of Key Laboratory of Civil Aviation Smart Airport Theory and System, Civil Aviation University of China under No. SATS202206, the Open Fund of Key Laboratory of Complex Electronic System Simulation under No. 614201002022205, the National Natural Science Foundation of China under No. U20B2050 and 82004499.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing Interests The authors declare no competing interests.

Ethics Approval Not applicable.

Consent to Publish All authors have approved the contents of this paper and have agreed to the submission policies of Peer-to-Peer Networking and Applications.

References

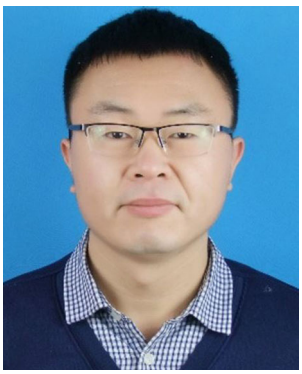
1. Costa M M, Silva M F (2019) "A survey on path planning algorithms for mobile robots," In: 2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC). IEEE, pp 1–7
2. Jones M, Djahel S, Welsh K (2023) Path-planning for unmanned aerial vehicles with environment complexity considerations: A survey. ACM Comput Surv 55(11):1–39
3. Paden B, Čáp M, Yong SZ, Yershov D, Frazzoli E (2016) A survey of motion planning and control techniques for self-driving urban vehicles. IEEE Trans Intell Veh 1(1):33–55
4. Algfoor ZA, Sunar MS, Kolivand H (2015) A comprehensive study on pathfinding techniques for robotics and video games. Int J Comput Games Technol 2015:7–7
5. Dijkstra E W (2022) "A note on two problems in connexion with graphs," In: Edsger Wybe Dijkstra: His Life, Work, and Legacy, pp 287–290
6. Hart PE, Nilsson NJ, Raphael B (1968) A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern 4(2):100–107
7. Koenig S, Likhachev M (2002) "D* lite," In: Eighteenth national conference on Artificial intelligence, pp 476–483
8. Ren Z, Rathinam S, Likhachev M, Choset H (2022) Multi-objective path-based d* lite. IEEE Robot Automat Lett 7(2):3318–3325
9. Bellman R (1954) The theory of dynamic programming. Bulletin of the American Math Soc 60(6):503–515
10. Bellman R (1957) "Dynamic programming, princeton univ," Press Princeton, New Jersey
11. LaValle SM, Kuffner JJ Jr (2001) Randomized kinodynamic planning. The Int J Robot Res 20(5):378–400
12. Karaman S, Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. The Int J Robot Res 30(7):846–894
13. Gammell J D, Srinivasa S S, Barfoot T D (2014) "Informed rrt: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, pp 2997–3004
14. Gammell J D, Srinivasa S S, Barfoot T D (2015) "Batch informed trees (bit): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," In: IEEE international conference on robotics and automation (ICRA). IEEE 2015:3067–3074
15. Strub M P, Gammell J D (2020) "Advanced bit (abit): Sampling-based planning with advanced graph-search techniques," In: 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp 130–136
16. Strub M P, Gammell J D (2020) "Adaptively informed trees (ait): Fast asymptotically optimal path planning through adaptive heuristics," In: 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp 3191–3198
17. Strub M P, Gammell J D (2022) "Adaptively informed trees (ait*) and effort informed trees (eit*): Asymmetric bidirectional sampling-based path planning," The Int J Robot Res, vol 41, no 4, pp 390–417
18. Hauser K (2015) "Lazy collision checking in asymptotically-optimal motion planning," In: IEEE international conference on robotics and automation (ICRA). IEEE 2015:2951–2957
19. Kleinbort M, Salzman O, Halperin D (2016) "Collision detection or nearest-neighbor search? on the computational bottleneck in sampling-based motion planning," arXiv:1607.04800
20. Kuffner J J, LaValle S M (2000) "Rrt-connect: An efficient approach to single-query path planning," In: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), vol 2. IEEE, pp 995–1001
21. Akgun B, Stilman M (2011) "Sampling heuristics for optimal motion planning in high dimensions," In: IEEE/RSJ international conference on intelligent robots and systems. IEEE 2011:2640–2645
22. Klemm S, Oberländer J, Hermann A, Roennau A, Schamm T, Zollner JM, Dillmann R (2015) "Rrt-connect: Faster, asymptotically optimal motion planning," In: IEEE international conference on robotics and biomimetics (ROBIO). IEEE 2015:1670–1677

23. Qureshi AH, Ayaz Y (2015) Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. Robot Autonom Syst 68:1–11
24. Burget F, Bennewitz M, Burgard W (2016) "Bi 2 rrt: An efficient sampling-based path planning framework for task-constrained mobile manipulation," In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp 3714–3721
25. Penrose M (2003) Random geometric graphs. OUP Oxford, vol 5
26. Janson L, Schmerling E, Clark A, Pavone M (2015) Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. The Int J Robot Res 34(7):883–921
27. Persson SM, Sharf I (2014) Sampling-based a* algorithm for robot path-planning. The Int J Robot Res 33(13):1683–1708
28. Urmson C, Simmons R (2003) "Approaches for heuristically biasing rrt growth," In: Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453), vol 2. IEEE, pp 1178–1183
29. Salzman O, Halperin D (2015) "Asymptotically-optimal motion planning using lower bounds on cost," In: IEEE International Conference on Robotics and Automation (ICRA) 2015:4167–4172
30. Koenig S, Likhachev M, Furcy D (2004) Lifelong planning a*. Artif Intell 155(1–2):93–146
31. Salzman O, Halperin D (2015) "Asymptotically-optimal motion planning using lower bounds on cost," In: 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp 4167–4172
32. Gilbert EN (1961) Random plane networks. J Soc Indust Appl Math 9(4):533–543
33. Sucan IA, Moll M, Kavraki LE (2012) The open motion planning library. IEEE Robot Automat Magaz 19(4):72–82
34. Tahirovic A, Ferizbegovic M (2018) "Rapidly-exploring random vines (rrv) for motion planning in configuration spaces with narrow passages," In: 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp 7055–7062

Kaibin Zhang is currently pursuing a Ph.D. degree in the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics (NUAA), Nanjing, China. He received his B.S. degree from Nanjing University of Chinese Medicine in 2021 and his M.S. degree from NUAA in 2024. His research interests include path planning algorithms, AI-based fuzz testing, and agent-based penetration testing.

Liang Liu is currently a associate professor in College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu Province, China. His research interests include distributed system, big data and system security. He received the B.S. degree in computer science from Northwestern Polytechnical University, Xi'an, Shanxi Province, China in 2005, and the Ph.D. degree in computer science from Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu Province, China in2012.

Wenbin Zhai is currently pursing the Ph.D. degree with the Department of Computing, The Hong Kong Polytechnic University. He received the B.S. degree from Nanjing University of Chinese Medicine, Nanjing, Jiangsu Province, China in 2020, and the M.S. degree in Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu Province, China in 2023. His research interests include AI security, Cybersecurity, Wireless Sensor Networks (WSNs), and Networking.



Youwei Ding received the B.S. and M.S. degrees in computer science from Yangzhou University, Yangzhou, Jiangsu, China, in 2007 and 2010, respectively, and the Ph.D. degree in computer science from the Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu, in 2016. He is currently an Associate Professor with the School of Artificial Intelligence and Information Technology, Nanjing University of Chinese Medicine, Nanjing. His research interests include energy efficient data management, big data analysis, and data security.

Jun Hu is an associate professor at the School of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics. He graduated from the Department of Computer Science and Technology, Nanjing University in 2006 with a doctorate degree in computer software and theory. His current research interests include software analysis and verification, modeling and security of complex systems, and airborne software systems.