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

Kaibin Zhang*, Liang Liu*, Wenbin Zhai*, Youwei Ding†, Jun Hu*
*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics
Nanjing, China
Emails: {zhangkaibin, liangliu, wenbinzhai, hujun}@nuaa.edu.cn
†School of Artificial Intelligence and Information Technology, Nanjing University of Chinese Medicine
Nanjing, China
Email: ywding@njucm.edu.cn

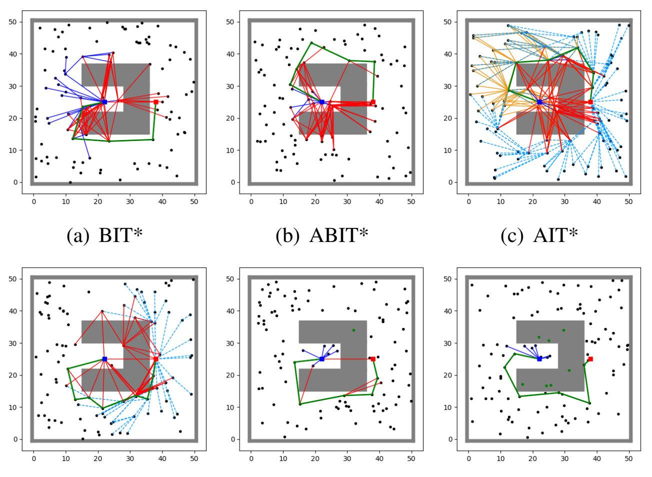
***Abstract*—Informed path planning is a type of algorithm that uses problem-specific knowledge, expressed as heuristics, to efficiently discover an optimal path between a start and goal state. The efficiency of an informed algorithm is contingent upon how quickly the planner can find the initial solution and the associated overhead involved in collision detection. Existing informed planners do not fully exploit the information contained in historical collision detection results, resulting in additional unnecessary collision detections. Furthermore, they optimize paths through rewiring before discovering an initial solution. This approach not only hampers the planner's space exploration, but also generates a superfluous amount of unproductive overhead. In this paper, an Obstacle-Sensitive and Initial-Solution-first path planning algorithm (OSIS) is proposed. OSIS leverages historical collision detection 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.**

***Index Terms*—sampling-based path planning, optimal path planning, informed search, collision detection, obstacle avoidance**

I. 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] and A* [6], utilize dynamic programming techniques [7] to find solutions in discrete state spaces.



(a) BIT* (b) ABIT* (c) AIT*

(d) EIT* (e) OSIS (f) OSIS(Preprocessed)

Fig. 1. Illustration of the search trees constructed by BIT*, ABIT*, AIT*, EIT*, and OSIS when optimizing path length. demonstrating their ability to detect obstacles in a scene and find a bypass path. 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 detection. The yellow lines in (c) represent edges that are added to the reverse tree and then removed from it after LPA* updates the reverse tree due to a collision detected. Note that the red collision edge was also added to the reverse tree and removed after a collision was detected. (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 detection in the planning process to predict the distribution of obstacles in the space. The green dots in (e) and (f) represent the sampling state in which OSIS believes that it is likely to collide with obstacles, so collision detection is not carried out and the sampling state is directly discarded.

The quality of the solution obtained by search-based path planning algorithms relies on the level of granularity in space discretization. Finer granularity discretization generally yields better solution quality, but it also introduces increased computational overhead [8]. Especially in high-dimensional state spaces, the computational overhead of search-based path planning algorithms grows exponentially with the number of states. This phenomenon is commonly referred to as the *curse of dimensionality* [9].

To overcome the limitations of the search-based algorithm in continuous space, the sampling-based path planning algorithm has been developed. It's basic idea is to randomly sample the space, where each sample is regarded as a state, and the set of all valid states (that is, they do not collide with obstacles) can be used to approximately represent the problem space, the planner then searches the path on this approximation.

The RRT [10] algorithm is one of the most classical sampling-based path planning algorithms, and RRT* [11] is an asymptotically optimal improvement of it.

Informed RRT* [12] introduces the concept of informed search. Informed search techniques empower planners to leverage existing solutions to determine the optimal solution's potential range. By doing so, the sampling and search space can be reduced, expediting solution convergence. As a result, informed planning algorithms typically exhibit improved performance.

Modern algorithms for informed path planning include Batch Informed Trees (BIT*) [13], Advanced BIT* (ABIT*) [14], Adaptively Informed Trees (AIT*) [15], and Effort Informed Trees (EIT*) [16].

In path planning algorithms, collision detection overhead is often considerable [17], [18], and planners should endeavor to minimize the frequency of collision detections. Nevertheless, existing informed algorithms do not fully capitalize on collision information during the search process, resulting in an excessive number of redundant collision detections.

Meanwhile, the majority of existing informed algorithms focus on path optimization through rewiring before locating the initial solution. This approach not only diminishes the speed of discovering the initial solution but also squanders computational resources on irrelevant paths. Furthermore, it significantly heightens the risk of the search becoming trapped in a local optimum for an extended duration.

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 detection results from previous search processes to estimate the distribution of obstacles in the environment. This information guides OSIS in maneuvering around obstacles and minimizing unnecessary collision detections, 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 don't directly contribute to finding the initial solution until after the initial solution is found. This strategic approach allows OSIS to swiftly locate the initial solution and disregard extraneous paths.

Fig. 1 illustrates the approach taken by OSIS and existing informed path planning algorithms to navigate around obstacles in space in search of solutions. As depicted in Fig. 1, OSIS detects noticeably fewer collisions compared to other informed path planning algorithms. The experimental results indicate that OSIS excels in identifying and maneuvering around obstacles, resulting in expedited initial solution searches and faster convergence.

The contribution of this paper is as follows:

- An Obstacle-Sensitive and Initial-Solution-first informed path planning algorithm (OSIS) is proposed to cope with the disadvantages of existing algorithms.
- OSIS estimates the distribution of obstacles in space by analyzing historical collision detection 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 promptly navigate away from potential local optima, thus accelerating the search for the initial solution.

The remainder of this paper is organized as follows: Section II provides an overview of the related work in the field of path planning algorithms. Section III presents the foundational knowledge relevant to informed planning algorithms, along with the mathematical model of the planning problem. Section IV details the implementation of OSIS. The experimental results are showcased in Section V. Finally, Section VI concludes this paper, summarizing the key findings.

II. RELATED WORK

A. Classical sampling-based path planning algorithms

The RRT [10] algorithm is considered to be one of the pioneers of sampling-based path planning. RRT begins by initiating a tree from the starting point. It then randomly samples one state in the planning space at each iterations and connects it to the nearest node in the tree, as long as there are no obstacles between them.

During each iteration of RRT, there's a probability of attempting to directly link the goal to the nearest node in the tree. If the resulting edge doesn't collide with obstacles, a path from start to the goal is found. As RRT samples uniformly at each iteration, it progressively explores the entire planning space with an increasing number of iterations, ultimately leading to the discovery of a solution. Since the growth of RRT is random and disordered, RRT isn't guaranteed to find the optimal solution.

RRT-Connect [19] is an improvement of RRT. RRT-Connect grows a tree at both start and goal, and grows in the direction of each other, finding a path from start to goal when the two trees are connected together. RRT-Connect has a significant performance improvement over RRT, but RRT-Connect is still not asymptotically optimal, but its proposed strategy of bidirectional search is followed by many subsequent algorithms, such as AIT* [15] and EIT* [16], and there are some algorithms that implement asymptotically optimal versions of RRT-Connect [20]–[23].

RRT* [11] introduces a rewiring technique to address the limitations of RRT in achieving optimal solutions. Through the implementation of this rewiring strategy, RRT* consistently updates the cost-to-come for nodes within the tree. This dynamic cost update process leads to a gradual convergence of the solution's cost as the number of iterations increases. Consequently, RRT* exhibits the property of being asymptotically optimal. However, the search strategy of RRT* remains random, which could result in growth of the tree in directions far from the goal.

RRT and RRT* utilize random geometric graph (RGG) [24] theory to approximate the planning space by generating random samples in the space, which form graphs with implicit edges that are used for finding 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*) [25] also employs RGG theory by generating a fixed number of samples at a time to construct an RGG, which is then searched for solutions. Unlike RRT and RRT*, FMT*'s construction and search are not simultaneous, resulting in an ordered but non-anytime search, which means that it can't return a solution at any point during the search.

B. 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*) [26]. The idea of using heuristics is derived from A* [6] . During the search process in A* algorithm, heuristics are utilized to evaluate the value of each state. The states with lower heuristic cost are more likely to contribute to the improvement of the solution. Heuristics enhance the 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. 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. Commonly used heuristics in existing planning algorithms include Euclidean or Manhattan distance, and some planners can update the heuristic estimate after detecting a collision to improve its accuracy [15], [16].

Heuristically-Guided RRT (hRRT) [27] is a heuristic-based variation of RRT, which employs heuristics to direct the growth of the tree. While this algorithms improve RRT's performance, but it 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) [28] is an anytime adaption of FMT* that incorporates heuristics. MPLB utilizes Dijkstra's algorithm to generate an admissible lower bounds heuristic that guides the search without collision detection. However, once a collision is detected, the heuristic is not updated in MPLB.

C. Informed path planning algorithms

Gammell et al. introduced Informed RRT* [12] 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 down 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. The BIT* [13] can generate samples in multiple batches, so that BIT* can dynamically build 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, enabling 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. Fig. 1 (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* [29] . 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* doesn't 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 depicted in Fig. 1 (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 detection. 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 detection resolution. By incorporating effort heuristics and sparse collision detection, 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. 1 (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 for guiding 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 optimizing numerous irrelevant paths. This can potentially prolong the process and trap the planner in local optima.

Based on these problems existing in existing informed programming algorithms, 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 in this paper.

III. Problem modeling and preliminaries

A. Problem modeling

The definition of the path planning problem in this paper is similar to that in [11]. 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. 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:

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

B. Preliminaries

Informed planning algorithms, such as BIT* [13] , generate an increasingly dense RGG [24] by batch sampling. The RGG is composed of a set of states, $X_{samples} \subset X$, that are uniformly generated at random, and these states constitute the vertexs in the graph. The edges between the vertexs 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* [30] and *r-disc* [31]. 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 Equations (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 detection) on it to decide whether to add it to the tree.

IV. OSIS

A. Basic Idea

OSIS constructs an obstacle density approximation by collecting statistics on the results of collision detections 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. Moreover, OSIS improves the rewiring strategy to avoid inefficient path optimization prior to finding the initial solution. This approach enables the planner to focus on searching for the initial solution. By employing these strategy, 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 collision detection during the execution of the algorithm.

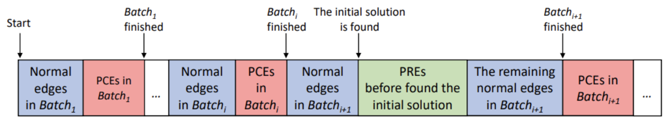

Start
$Batch_1$ finished
$Batch_i$ finished
The initial solution is found
$Batch_{i+1}$ finished
Normal edges in $Batch_1$
PCEs in $Batch_1$
...
Normal edges in $Batch_i$
PCEs in $Batch_i$
Normal edges in $Batch_{i+1}$
PREs before found the initial solution
The remaining normal edges in $Batch_{i+1}$
PCEs in $Batch_{i+1}$
...

Fig. 2. The order of edges processed by OSIS.

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 be first treated as a PCE. Furthermore, the normal edges queue Q of OSIS stores edges that are neither PCEs nor PREs. In general, edges in Q will be preferentially processed compared to edges in Q_C and Q_R.

Fig. 2 depicts 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
```

B. 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.

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 addmisable 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 as 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 $E_{out}$
```

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.

C. 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 Equation (4):

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

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 subspaces, its collision factor is calculated as follows:

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

where l_{a_i} denotes the length of the edge (x, y) in the i-th subspace that it passes through, 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 Equation (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 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 minimal impact on the heuristic cost of (x, y). Conversely, 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.

D. 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 been found yet, 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 attempt adding it to the tree. This function returns a boolean value indicating 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 lower potential for optimizing the solution for the remaining edges.

The $TryAddBestEdge$ function 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 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 for finding the initial solution. Conversely, it may result in an unnecessary search. Consequently, OSIS defers rewiring until an initial solution is found.

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:     $is\_processing\_done \leftarrow TryAddBestEdge((x_p, x_c))$;
```

If a heuristic cost based on 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 detection 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 doesn't collide with the obstacle and can improve the solution. If 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 hasn't 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.

E. 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 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)*.

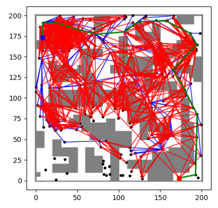
If the current approximation is exhausted or an initial solution has not been found *(lines 12-13 and Alg. 1)*, OSIS proceeds to initiate the next round of approximation. Prior to that, OSIS handles the PCEs that were postponed during the current approximation phase *(line 14 in Alg. 1; Alg. 4)*. Subsequently,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

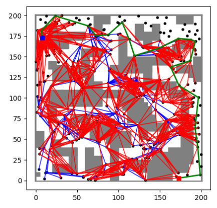
During the sampling process, it is necessary to check whether the generated samples collide with obstacles. OSIS estimates the probability that a sample is valid based on the obstacle density, If the subspace where the sample belongs to has a high density of obstacles, OSIS can directly discard the sample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

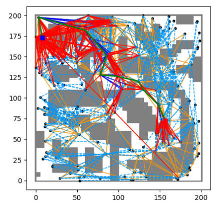
Finally, the truncation factor is updated to ensure a more precise and efficient search in the subsequent round *(line 19 in Alg. 1)*. The strategy for updating the inflation and truncation factors is similar to that of the study [14].

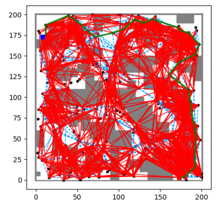
V. EXPERIMENTAL RESULTS

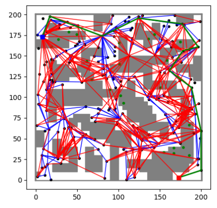
To evaluate the performance of OSIS, we compare it with the Open Motion Planning Library (OMPL) [32] versions of BIT*, ABIT*, AIT*, and EIT*. The performance are measured with OMPL v1.6.0 on a laptop with 16GB of RAM and an Intel i7-8550U processor running Ubuntu 18.04. For all planners, the RGG tuning parameter η is set to 1.1, 100 samples are generated in each batch, *k-nearest* strategy is used to obtain neighbors, Euclidean distance is used as the default heuristic, and the optimization objective is to minimize the path length. For ABIT* and OSIS, the initial values of the inflation factor ε_{infl} and truncation factor ε_{trunc} are 10^6 and 1, respectively. In OSIS, the obstacle sensitivity α is set to the default value of 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.


(a) BIT*


(b) ABIT*


(c) AIT*


(d) EIT*


(e) OSIS

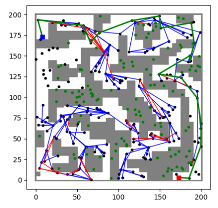
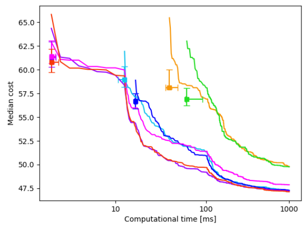
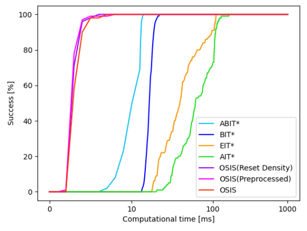

(f) OSIS(Preprocessed)

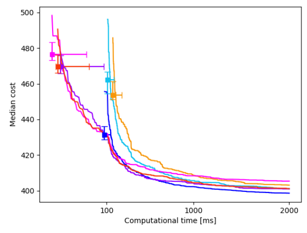
Fig. 3. The growth of the tree in Problem 2 when different planners find the initial solution. The meanings represented by the edges and points of various colors are the same as in Fig. 1.



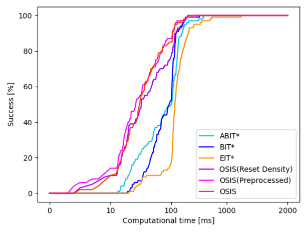
(a) Median path cost of Problem 1



(b) Success rate of Problem 1



(c) Median path cost of Problem 2



(d) Success rate of Problem 2

Fig. 4. The experimental results of Problem 1 and Problem 2.

The experiments are mainly conducted under two simulation scenarios, and Fig. 1 (Problem 1) and 3 (Problem 2) depicts the planning problem for these two scenarios. 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 portrays 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 swiftly escape from local optima.

In this paper, OSIS adopts a coarse-grained gridding approach to partition the planning space, where each subspace is represented as a square grid. Specifically, in Problem 1, the grid has a side length of 5, while in Problem 2, the grid is defined with a side length of 10. The runtime of all planners is limited to 1 second and 2 seconds in Problem 1 and Problem

2, respectively. This paper executes each planner 100 times in each problem separately based on the setup described above. In addition, 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. Simultaneously, the obstacle density will not be updated during the execution of the algorithm.

Fig. 4 shows the median path cost and success rate achieved by all test subjects in Problem 1 and Problem 2. In Fig. 4 (a) and (c) the squares represent the median cost and time of the initial solution, and the lines show the median cost over time of the solution of the almost-surely asymptotically optimal planners. Error bars show non-parametric 99% confidence intervals for solution cost and time. In Fig. 4 (b) and (d), the lines represent the success rate of the planner to find the initial solution over time.

Note that in Problem 2, AIT* can hardly find the initial solution within 2 seconds, so there is no data for AIT* in Fig. 4 (c) and (d). This is due to the fact that the scene in Problem 2 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 from Fig. 4, regardless of the mode used, OSIS outperforms existing informed planning algorithms by achieving faster initial solution finding and convergence to the optimal solution.

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. In the complex scenario of Problem 2, 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 arises from their initial understanding of the obstacle distribution within the planning space, enabling 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. Nonetheless, even the reset density mode of OSIS outperforms other informed planning algorithms significantly.

Furthermore, it is worth noting that the convergence speed of the other modes of OSIS also gradually decreases over time. This is because the search strategy of OSIS drives the planner to stay away from the obstacles as far 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 from performance degradation due to frequent updates of the reverse tree.

TABLE I
PERFORMANCE OF EACH PLANNER IN PROBLEM 1

Planner	***Avg. Time (ms)***	***Avg. Edge Collisions Detections***	***Avg. Edge Valid Rate***
BIT*	21.76	99	38.50%
ABIT*	11.42	67	20.88%
AIT*	86.99	66	24.44%
EIT*	69.18	102	10.53%
OSIS	3.86	29	47.80%
OSIS (Reset Density)	3.98	31	49.57%
OSIS (Preprocessed)	3.78	19	90.80%

TABLE II
PERFORMANCE OF EACH PLANNER IN PROBLEM 2

Planner	***Avg. Time (ms)***	***Avg. Edge Collisions Detections***	***Avg. Edge Valid Rate***
BIT*	111.43	1230	20.27%
ABIT*	116.01	1167	21.40%
AIT*	-	-	-
EIT*	178.18	1253	2.71%
OSIS	53.14	659	42.47%
OSIS (Reset Density)	61.60	728	43.97%
OSIS (Preprocessed)	46.64	353	91.56%

Fig. 1 and 3 depict the evolution of the search tree during the process of finding the initial solution for Problem 1 and Problem 2, 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. 3 (c) showcases the growth of AIT* until the execution time limit is reached, and the green path represents the closest path to 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.

Fig. 1 (e) and 3 (e) along with Fig. 1 (f) and 3 (f) present the growth of the search tree for OSIS in different settings: without any obstacle density information, and with sufficient obstacle density information obtained through preprocessing or multiple planning tasks, respectively.

Based on Fig. 1 and 3, it is evident that OSIS outperforms other informed planning algorithms in terms of collision detection, with significantly fewer instances of collision detection 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 I and II present the average time taken to find the initial solution, the average number of collision detections performed on edges, and the average proportion of edges successfully passing collision detection for each planner in Problem 1 and Problem 2, respectively. It is evident that all versions of OSIS exhibit shorter initial solution finding times, lower collision detection counts, and higher edge utilization compared to other planners.

VI. Conclusion

OSIS employs two key strategies that contribute to its ability to efficiently find the initial solution and converge it. Firstly, it calculates the obstacle density distribution in the planning space by tracking collision detection 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 navigates around obstacles. Secondly, OSIS adopts an initial solution-first path optimization strategy. It delays the rewiring operation, which 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 faster jump out of local optimum, so as to speed up the convergence of solution. By combining these two strategies, OSIS demonstrates its capability to rapidly discover the initial solution and converge efficiently to the optimal solution.

References

[1] M. M. Costa and M. F. Silva, "A survey on path planning algorithms for mobile robots," in *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*. IEEE, 2019, pp. 1–7.

[2] M. Jones, S. Djahel, and K. Welsh, "Path-planning for unmanned aerial vehicles with environment complexity considerations: A survey," *ACM Computing Surveys*, vol. 55, no. 11, pp. 1–39, 2023.

[3] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[4] Z. A. Algfoor, M. S. Sunar, and H. Kolivand, "A comprehensive study on pathfinding techniques for robotics and video games," *International Journal of Computer Games Technology*, vol. 2015, pp. 7–7, 2015.

[5] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, 2022, pp. 287–290.

[6] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[7] R. Bellman, "The theory of dynamic programming," *Bulletin of the American Mathematical Society*, vol. 60, no. 6, pp. 503–515, 1954.

[8] ——, "Dynamic programming, princeton univ," *Press Princeton, New Jersey*, 1957.

[9] T. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," *IEEE translation journal on magnetics in Japan*, vol. 2, no. 8, pp. 740–741, 1987.

[10] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[11] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[12] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "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, 2014, pp. 2997–3004.

[13] ——, "Batch informed trees (bit): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 3067–3074.

[14] M. P. Strub and J. D. Gammell, "Advanced bit (abit): Sampling-based planning with advanced graph-search techniques," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 130–136.

[15] ——, "Adaptively informed trees (ait): Fast asymptotically optimal path planning through adaptive heuristics," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 3191–3198.

[16] ——, "Adaptively informed trees (ait*) and effort informed trees (eit*): Asymmetric bidirectional sampling-based path planning," *The International Journal of Robotics Research*, vol. 41, no. 4, pp. 390–417, 2022.

[17] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 2951–2957.

[18] M. Kleinbort, O. Salzman, and D. Halperin, "Collision detection or nearest-neighbor search? on the computational bottleneck in sampling-based motion planning," *arXiv preprint arXiv:1607.04800*, 2016.

[19] J. J. Kuffner and S. M. LaValle, "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, 2000, pp. 995–1001.

[20] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2011, pp. 2640–2645.

[21] S. Klemm, J. Oberländer, A. Hermann, A. Roennau, T. Schamm, J. M. Zollner, and R. Dillmann, "Rrt-connect: Faster, asymptotically optimal motion planning," in *2015 IEEE international conference on robotics and biomimetics (ROBIO)*. IEEE, 2015, pp. 1670–1677.

[22] A. H. Qureshi and Y. Ayaz, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments," *Robotics and Autonomous Systems*, vol. 68, pp. 1–11, 2015.

[23] F. Burget, M. Bennewitz, and W. Burgard, "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, 2016, pp. 3714–3721.

[24] M. Penrose, *Random geometric graphs*. OUP Oxford, 2003, vol. 5.

[25] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015.

[26] S. M. Persson and I. Sharf, "Sampling-based a* algorithm for robot path-planning," *The International Journal of Robotics Research*, vol. 33, no. 13, pp. 1683–1708, 2014.

[27] C. Urmson and R. Simmons, "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, 2003, pp. 1178–1183.

[28] O. Salzman and D. Halperin, "Asymptotically-optimal motion planning using lower bounds on cost," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 4167–4172.

[29] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning a*," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.

[30] O. Salzman and D. Halperin, "Asymptotically-optimal motion planning using lower bounds on cost," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 4167–4172.

[31] E. N. Gilbert, "Random plane networks," *Journal of the society for industrial and applied mathematics*, vol. 9, no. 4, pp. 533–543, 1961.

[32] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.