Zeta*-SIPP: Improved Time-Optimal Any-Angle Safe-Interval Path Planning

Yiyuan Zou[*] , **Clark Borst**
Control and Simulation, Faculty of Aerospace Engineering
Delft University of Technology, The Netherlands
y.zou@tudelft.nl, c.borst@tudelft.nl

Abstract

Any-angle path planning is an extension of traditional path-planning algorithms that aims to generate smoother and shorter paths in graphs by allowing any-angle moves between vertices, rather than being restricted by edges. Many any-angle path-planning algorithms have been proposed, such as Theta*, Block A* and Anya, but most of them are designed only for static environments, which is not applicable when dynamic obstacles are present. Time-Optimal Any-Angle Safe-Interval Path Planning (TO-AA-SIPP) was developed to fill this gap, which can find an *optimal* collision-free any-angle path that minimizes the traversal time. However, as indicated by its authors, TO-AA-SIPP may not be efficient enough to be used in multi-agent pathfinding (MAPF). Therefore, this paper presents a new algorithm Zeta*-SIPP to improve TO-AA-SIPP by means of 1) reducing useless search nodes that have no contribution to finding optimal solutions, and 2) introducing Field of View (FoV) instead of Line of Sight (LoS) to speed up visibility checks with static obstacles. Benchmark experiments showed that Zeta*-SIPP reduced the computation time of TO-AA-SIPP by around 70%-90% on average.

1 Introduction

Path planning aims to find an optimal path between two locations. A* [Hart *et al.*, 1968] is one of the most classic algorithms to solve the path-planning problem. However, the paths found by A* are usually not the true shortest because the expansion of A* is limited to adjacent neighbors and thus the shape of the path is highly affected by the graph structure. On a square grid map, A* only searches paths in 45-degree increments. Therefore, any-angle path planning has been developed, which ignores the edges of the graph and allows any-angle turns at vertices, to generate smoother and shorter paths, such as Theta* [Daniel *et al.*, 2010], Block A* [Yap *et al.*, 2011] and Anya [Harabor *et al.*, 2016]. More analysis and evaluation regarding any-angle path planning can be found in [Uras and Koenig, 2015]

[*]Corresponding Author

Most any-angle path-planning algorithms are designed only for static obstacles, which limits their applications in dynamic environments. Path planning with dynamic obstacles needs to consider the time dimension when searching for collision-free paths. A common approach to handle it is to divide the time dimension into multiple equal-length time slots, thus creating a space-time grid map [Silver, 2005]. In this way, static and dynamic obstacles can both be represented by space-time grids, and thus path-planning algorithms, like A*, can be easily extended to dynamic environments. However, this approach may aggravate the *curse of dimensionality* problem because obstacle grids could be quite sparse, especially when there are only few dynamic obstacles. Therefore, Safe Interval Path Planning (SIPP) [Phillips and Likhachev, 2011] was proposed to merge consecutive obstacle-free time slots into safe intervals, creating a *compact* space-time map to narrow the search space.

To combine any-angle path planning and SIPP, Any-Angle SIPP (AA-SIPP) was designed [Yakovlev and Andreychuk, 2017]. The basic idea of AA-SIPP is similar to Theta*. Both attempt to straighten paths by checking if the neighbors of a current node can be reached from the parent of this current node with lower cost. However, this is a greedy approach. AA-SIPP, like Theta*, is not guaranteed to find true optimal any-angle paths. Thus, Time-Optimal AA-SIPP with inverted expansion (TO-AA-SIPP is used in this paper instead of iTO-AA-SIPP) was further developed [Yakovlev and Andreychuk, 2021] by checking if the paths through the current node to its neighbors could be straightened by any other node in the search space rather than only the parent of the current node. This operation ensures optimality but also slows down the algorithm since too many nodes need to be examined during the search process. As stated in [Yakovlev and Andreychuk, 2021], TO-AA-SIPP may not be fast enough to be straightforwardly used in multi-agent pathfinding (MAPF).

With further research, we found that actually, in TO-AA-SIPP, not all nodes in the search space contributed to finding optimal paths. Some nodes were never removed from the *open* list after being inserted during initialization. Therefore, it should be possible to improve the performance of TO-AA-SIPP by reducing these useless nodes. Inspired by Informed Rapidly-exploring Random Tree* (Informed RRT*) [Gammell *et al.*, 2014] and Batch Informed Trees (BIT*) [Gammell *et al.*, 2015], we found that the "ellipse" used in Informed

RRT* and BIT* can be adapted to indicate and limit the *current* search range and thus designed an *any-angle forward expansion* to incrementally add necessary nodes to the *open* list, like the forward expansion in A*. In this case, nodes outside the "ellipse" are temporarily useless until they are expanded by this "ellipse" (search range). By implementing this idea, Zeta-SIPP was designed.

However, in the worst case, Zeta-SIPP would be as slow as TO-AA-SIPP because all nodes in the search space would be expanded by the any-angle forward expansion ("ellipse"). Therefore, to further improve TO-AA-SIPP and also Zeta-SIPP, we introduced *Field of View* (FoV) to replace Line of Sight (LoS) for collision detection (visibility checks) with static obstacles and thus developed TO-AA-FoV-SIPP and Zeta*-SIPP respectively. Zeta*-SIPP can be considered a superior version of Zeta-SIPP since their search processes are nearly identical except for the visibility check methods. According to the benchmark experiments, Zeta*-SIPP outperformed Zeta-SIPP and TO-AA-FoV-SIPP in most cases, especially when the map was large.

2 Problem Statement

Suppose there is an agent navigating from a start p_s to a target p_t in a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. Two different types of actions are allowed: *move* and *wait*. It means that the agent can *wait* at a certain vertex or *move* from a vertex to the other vertex. The movement speed is constant and the cost of an action is its duration. The agent can only turn or wait at the vertices and the inertial effects are neglected. Please note that in some practical cases, the wait action is not possible, such as fixed-wing aircraft. To simplify the problem, the radius of the agent is ignored.

A path plan is an ordered sequence of position-time pairs $\pi = \{(p_1, t_1), (p_2, t_2), ..., (p_n, t_n)\}$ where p_i represents a position and t_i denotes the waiting time at the position p_i. If the wait action is forbidden, $t_i = 0\,(i = 1, 2, ..., n)$. The cost of the path plan is the sum of the duration of the actions. The goal of this problem is to find the time-optimal plan from a given start p_s to a given target p_t.

We assume that the plans of dynamic obstacles are known: $\{\pi^1, \pi^2, ..., \pi^k\}$, and after the plans are accomplished, the dynamic obstacles will disappear. This is reasonable for flying vehicles. For instance, when drones complete their missions, they will land to reload or recharge. Some papers assume that the dynamic obstacles will stay in their target vertices forever [Yakovlev and Andreychuk, 2021] as on the ground, a robot cannot suddenly disappear and it will become a static obstacle after reaching its target.

3 Algorithm Description

3.1 Overview

Before diving into the details of the proposed algorithms, we briefly introduce the basic idea of TO-AA-SIPP [Yakovlev and Andreychuk, 2021] and illustrate how it can be improved.

To make the explanation more easily understandable, we start from the A* algorithm since A* is quite well-known and also TO-AA-SIPP is A*-based. In A*, after a node is moved from *open* to *closed*, eight neighbor nodes of this closed node

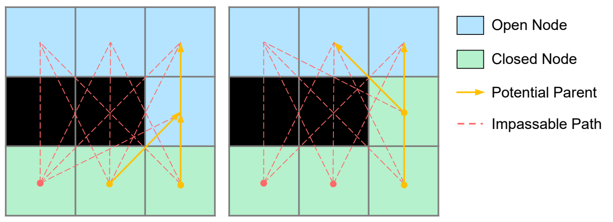

Figure 1: Inverted expansion to find potential parents of open nodes.

are generated. If a neighbor is not visited before, this closed node will directly be its parent. However, if it has been visited, which means it already has a parent, A* needs to compare this closed node with the current parent of this neighbor node to find out which is better and could be its *true parent*. These two nodes can both be viewed as *potential parents* of this neighbor node, and the metric used for comparison is $g(pp(n)) + g(pp(n), n)$ where $pp(n)$ is the potential parent of a node n, $g(pp(n))$ is the real cost from the start to $pp(n)$ and $g(pp(n), n)$ is the real cost from $pp(n)$ to n. Therefore, the current parent of a node n is also its current *best potential parent* $bpp(n)$, and thus the real cost $g(n)$ can be written as

$$g(n) = g(bpp(n)) + g(bpp(n), n) \tag{1}$$

In TO-AA-SIPP, the cost computation is similar. However, to speed up the process, it delays the SIPP-based collision detection with *dynamic obstacles* until necessary. It means that TO-AA-SIPP utilizes an estimated cost $h(bpp(n), n)$, the lower bound, to replace the real cost $g(bpp(n), n)$ in Eq. (1) when a node is inserted into $open$. After a node with the minimum cost is removed from $open$, the SIPP-based collision detection will be executed to compute the real cost $g(bpp(n), n)$ and determine if this node can be inserted into $closed$. This "lazy" evaluation is similar to Lazy Theta* [Nash *et al.*, 2010] and BIT* [Gammell *et al.*, 2015] which delay line-of-sight checks with static obstacles. Please note that in TO-AA-SIPP, only the SIPP-based collision detection with dynamic obstacles is delayed while the line-of-sight checks with static obstacles are not. Hence, the cost function of TO-AA-SIPP is

$$\begin{aligned} f(n) &= g_{low}(n) + h(n) \\ &= g(bpp(n)) + h(bpp(n), n) + h(n) \end{aligned} \tag{2}$$

where $g_{low}(n)$ is the lower bound of $g(n)$ and $h(n)$ is the estimated cost from n to the target. This approach needs to record all the potential parents of the explored nodes. Then it can conduct the SIPP-based collision detection with dynamic obstacles later from the best potential parent to the worst potential parent until one can be the true parent (the detailed process is more complicated). Only when the true parent of a node is found can this node be inserted into $closed$ and meanwhile might be a potential parent of other nodes in $open$. Since this approach discovers and maintains connections between open nodes and their potential parents at each search step, it can be called *inverted expansion* [Yakovlev and Andreychuk, 2021], as shown in Figure 1.

To find optimal any-angle paths, unlike the eight neighbors in A*, TO-AA-SIPP examines all nodes in the search space at each step. It means that in TO-AA-SIPP, when a node is closed, all the other nodes can be considered the "neighbors" of this closed node. The *extended "neighbors"* guarantee that TO-AA-SIPP is not limited to 45-degree increments and thus can find any-angle paths. However, not all search nodes may contribute to finding optimal paths. For example, if the map is large, it may be unnecessary to check the visibility connections between the closed nodes and the nodes that are very far from the start and target. In this case, many unnecessary line-of-sight checks may be conducted in TO-AA-SIPP since too many useless nodes exist in *open*, which reduces the performance of the algorithm.

To address this issue, a natural idea is to also delay line-of-sight checks, which can be performed with the SIPP-based collision detection simultaneously. In this way, only necessary line-of-sight checks will be executed. However, if there are many static obstacles, it may result in much more sorting computation in *open*. This is because after collision detection, $f(n)$ is updated. Only if $f(n)$ is still the minimum, the node n can be inserted into *closed*, otherwise, it has to be re-inserted into *open* and reordered. There is a trade-off between collision detection and sorting computation when applying the "lazy" evaluation. Since the size of *open* may be very large and grid-based line-of-sight checks can be implemented efficiently with line drawing algorithms, like Bresenham's line algorithm [Bresenham, 1965], Lazy TO-AA-SIPP could be much slower in the worst case. Therefore, this idea may not be suitable for improving TO-AA-SIPP.

Another idea is to directly reduce useless nodes in *open*, which will lead to both fewer line-of-sight checks and sorting calculations. In A*, forward expansion is applied to extend *open* at each step, and the nodes in *open* form a boundary of the A* search (search range). It is promising to develop a similar technique for TO-AA-SIPP to reduce the size of *open* as in the worst case, only the calculations for forward expansion are redundant compared to the original TO-AA-SIPP. Therefore, to implement this idea, Zeta-SIPP is proposed and the details will be illustrated in section 3.2.

Those familiar with computer graphics may recognize that line-of-sight checks may not be the best way to conduct collision detection with static obstacles in TO-AA-SIPP. The closed node can be viewed as a "light source" and the grids near the "light source" could be checked for multiple times, leading to a waste of computing resources. Therefore, *Field of View* (FoV) is introduced to replace line-of-sight checks. TO-AA-FoV-SIPP and Zeta*-SIPP are thus developed. Symmetric recursive shadowcasting [Bergström, 2001] is applied to compute the field of view efficiently.

3.2 Zeta-SIPP

The main idea of Zeta-SIPP is to reduce useless open nodes in TO-AA-SIPP using any-angle forward expansion. It means that only necessary nodes are expanded and inserted into *open* at each step instead of inserting all nodes in the search space into *open* at the beginning. Unlike A*, Zeta-SIPP aims to find any-angle paths, and thus the forward expansion of A* cannot be directly applied to Zeta-SIPP. Inspired by Informed RRT* and BIT*, we can use an expanding "ellipse" to limit

Algorithm 1 Main Loop

```
 1: while $\min_{n \in open} f(n) < \infty$ do
 2:     $n \leftarrow$ findNextClosedNode($open$)
 3:     if $n \neq null$ then
 4:         if $n = target$ then
 5:             return pathTo($n$)
 6:         end if
 7:         invertedExpansion($n, open$)
 8:     end if
 9:     forwardExpansion($open$)
10: end while
11: return $\varnothing$
```

Algorithm 2 findNextClosedNode($open$)

```
 1: $n \leftarrow \arg\min_{n \in open} f(n)$
 2: remove $n$ from $open$
 3: if $bpp(n) \in potentialParents(n)$ then
 4:     remove $bpp(n)$ from $potentialParents(n)$
 5: end if
 6: $g_{new} \leftarrow$ validateTransition($bpp(n), n$)
 7: if $g_{new} < g(n)$ then
 8:     $g(n) \leftarrow g_{new}$
 9:     $parent(n) \leftarrow bpp(n)$
10: end if
11: if newBestPotentialParentExists($n$) then
12:     insert $n$ into $open$
13:     return $null$
14: end if
15: if $g(n)+h(n) \leq \min_{n \in bound} f_{low}(n)$ and $g(n)+h(n) \leq$
    $\min_{n \in open} f(n)$ then
16:     insert $n$ into $closed$
17:     return $n$
18: else
19:     insert $n$ into $open$
20:     return $null$
21: end if
```

the range of any-angle search:

$$f_{low}(n) \leq \min_{m \in open} f(m) \quad \forall n \in S \tag{3}$$

where $f_{low}(n) = h(start, n) + h(n) \leq g(n) + h(n) = f(n)$ is the lower bound of $f(n)$ and S is the search space. The focal points of the "ellipse" are the start and target and the major axis length is $f_{low}(n)$. If all nodes satisfying the Inequality (3) are inserted into *open*, then

$$\min_{m \in open} f(m) < f_{low}(n) \leq f(n) \quad \forall n \in S_{out} \tag{4}$$

where $S_{out} = S \setminus (open \cup closed)$ is the search space outside the current search range. It indicates that the minimum cost in *open* is also the minimum in the remaining search space ($S \setminus closed$). Thus, at each step, only *open* is enough to find the next closed node, and there is no need to consider S_{out}. Therefore, we can apply the Inequality (3) to develop an any-angle forward expansion for Zeta-SIPP and the search range of TO-AA-SIPP can be limited during the search process.

The pseudocode of Zeta-SIPP is shown in Algorithms 1-5. To make the code structure clearer, we split the main loop into

Algorithm 3 `newBestPotentialParentExists`(n)

```
1: $g_{low}(n) \leftarrow g(n)$, $bpp(n) \leftarrow parent(n)$
2: $f(n) \leftarrow g_{low}(n) + h(n)$
3: $isExisting \leftarrow false$
4: for each $n' \in potentialParents(n)$ do
5:     if $g(n') + h(n', n) < g_{low}(n)$ then
6:         $g_{low}(n) \leftarrow g(n') + h(n', n)$
7:         $bpp(n) \leftarrow n'$
8:         $f(n) \leftarrow g_{low}(n) + h(n)$
9:         $isExisting \leftarrow true$
10:     end if
11: end for
12: return $isExisting$
```

Algorithm 4 `addPotentialParent`(n, n')

```
1: insert $n$ into $potentialParents(n')$
2: if $g(n) + h(n, n') < g_{low}(n')$ then
3:     $g_{low}(n') \leftarrow g(n) + h(n, n')$
4:     $bpp(n') \leftarrow n$
5:     $f(n') \leftarrow g_{low}(n') + h(n')$
6:     update $n'$ in $open$
7: end if
```

several modules, as shown in Algorithm 1. The revised part, compared to the original TO-AA-SIPP, is marked in red. The `findNextClosedNode` is to identify the next closed node n from the *open* list. Since the SIPP-based collision detection is delayed, the current node may be re-inserted into the *open* list after SIPP-based examinations (i.e., `validateTransition` in Algorithm 2). In this case, the next closed node would be *null*. The `invertedExpansion` assigns the closed node n as a potential parent to all nodes in the current *open* list, provided there are no static obstacles blocking the connection (i.e., `lineOfSight` in Algorithm 5). Please note that if the `invertedExpansion` is removed, Algorithm 1 can also represent the main loop of A*. The `forwardExpansion` utilizes an expanding "ellipse" to continuously add new nodes into the *open* list. Here, we introduce a new list called *bound* to indicate the nodes around the boundary of the search range (*open* and *closed*). The initialization and expansion of the *bound* list are important because they are related to whether the Inequality (3) can be satisfied by Line 9 in Algorithm 5. For simplicity, we initialize the *bound* list by directly inserting all the unblocked search nodes in Zeta-SIPP, similar to how the *open* list is initialized in TO-AA-SIPP. However, the nodes in the *bound* list do not need be examined by `lineOfSight`, thereby improving the performance of TO-AA-SIPP.

In `findNextClosedNode`, the SIPP-based collision detection with dynamic obstacles, namely `validateTransition`, will be executed to compute the real cost $g(bpp(n), n)$ from the best potential parent $bpp(n)$ to the current node n (Line 6). The path from $bpp(n)$ to n may be disturbed by dynamic obstacles, leading to $g(bpp(n), n) > h(bpp(n), n)$ in Eq. (2). Let the cost after the SIPP-based collision detection be

$$f_{sipp}(n) = g(bpp(n)) + g(bpp(n), n) + h(n) \qquad (5)$$

Algorithm 5 Zeta-SIPP Expansion

```
1: function invertedExpansion($n$, $open$)
2:     for each $n' \in open$ do
3:         if lightOfSight($n, n'$) $= true$ then
4:             addPotentialParent($n, n'$)
5:         end if
6:     end for
7: end function

8: function forwardExpansion($open$)
9:     while $\min_{n \in bound} f_{low}(n) \leq \min_{n \in open} f(n)$ do
10:         $n \leftarrow \arg\min_{n \in bound} f_{low}(n)$
11:         move $n$ from $bound$ to $open$
12:         invertedCheck($n$)
13:     end while
14: end function

15: function invertedCheck($n$)
16:     for each $n' \in closed$ do
17:         if lightOfSight($n', n$) $= true$ then
18:             addPotentialParent($n', n$)
19:         end if
20:     end for
21: end function
```

There maybe exists a "better" potential parent $pp(n)$ than $bpp(n)$ (Line 11, see Algorithm 3):

$$f(n) \leq f(n)' < f_{sipp}(n) \quad \exists\, pp(n) \in pps(n) \qquad (6)$$

where $f(n)' = g(pp(n)) + h(pp(n), n) + h(n)$ and $pps(n)$ is the collection of potential parents of n. The best potential parent should be changed to $pp(n)$: $bpp(n) \leftarrow pp(n)$, as perhaps $f(n)' \leq f_{sipp}(n)' < f_{sipp}(n)$. In this case, the current node should be re-inserted into *open* (Line 12) with a new cost $f(n) \leftarrow f(n)'$. Also, there maybe exists a "better" node n' than n (Line 15):

$$f(n) \leq f(n') < f_{sipp}(n) \quad \exists\, n' \in open \qquad (7)$$

The current node n is no longer the most promising node as perhaps $f(n') \leq f_{sipp}(n') < f_{sipp}(n)$. In this case, the current node n should also be re-inserted into *open* (Line 19) with a new cost $f(n) \leftarrow f_{sipp}(n)$. When the current node is not closed, `findNextClosedNode` will return *null*.

In Algorithm 4, n is a closed node while n' is an open node because only the closed node can be a potential parent of an open node. In Algorithm 5, the inverted expansion builds the connections between the current closed node and all the open nodes whereas the forward expansion inserts new nodes into *open* based on the Inequality (3) and finds the connections between the new open nodes and all the closed nodes.

3.3 TO-AA-FoV-SIPP

The idea of TO-AA-FoV-SIPP is simple. At each step of TO-AA-SIPP, after a node is inserted into *closed*, shadow-casting will be executed to find visible open nodes from this closed node and subsequently this closed node will be added to their potential parent lists using Algorithm 4. This process is similar to `invertedExpansion` in Algorithm 5, but

Algorithm 6 Zeta*-SIPP Expansion

```
1: function invertedExpansion($n$)
2:     for each $n' \in children(n)$ do
3:         if $n' \notin closed$ then
4:             addPotentialParent($n, n'$)
5:         end if
6:     end for
7: end function

8: function forwardExpansion($open$)
9:     while $\min_{n \in bound} f_{low}(n) \leq \min_{n \in open} f(n)$ do
10:        $n \leftarrow \arg\min_{n \in bound} f_{low}(n)$
11:        move $n$ from $bound$ to $open$
12:        invertedScan($n$)
13:    end while
14: end function

15: function invertedScan($n$)
16:     $N_{visible} \leftarrow$ shadowcasting($n$) $\cap$ $open$
17:     for each $n' \in N_{visible}$ do
18:         if $n' \in closed$ then
19:             addPotentialParent($n', n$)
20:         else
21:             insert $n'$ into $children(n)$
22:             insert $n$ into $children(n')$
23:         end if
24:     end for
25: end function
```

rather than `lineOfSight`, shadowcasting for computing the field of view (FoV) is applied. In addition, the size of *open* can be reduced using the field of view. In TO-AA-FoV-SIPP, only when a node is visible from one of the closed nodes can it be inserted into *open*. This is because if a node has no potential parent, the cost of this node is infinite. The inequality (4) can also be satisfied since $f(n) = \infty, \forall n \in S_{out}$ where $S_{out} = S \setminus (open \cup closed)$.

3.4 Zeta*-SIPP

The main pseudocode of Zeta*-SIPP is shown in Algorithm 6 and the major revision compared to Zeta-SIPP is marked in red. In general, the closed nodes may be the potential parents of the open nodes. Thus, the "light source" that needs to compute the field of view should be a closed node, like TO-AA-FoV-SIPP. However, in Zeta*-SIPP, since the search range is bounded, we apply *inverted scanning*, which means that the open nodes are considered as "light sources" rather than the closed nodes. The "light sources" are distributed on the boundary of the search range and only illuminate the interior. At each step of the forward expansion, when a new node is moved from *bound* to *open*, an inverted scanning is executed to compute the field of view from this new open node and obtain the corresponding visible nodes inside the search range. In practice, we treat the node n as a wall in shadowcasting if its $f_{low}(n)$ is larger than the major axis length of the current elliptical boundary plus $\sqrt{2}$ grid length (grid buffer), and thus shadowcasting is bounded. Since the applied shadowcasting is symmetric, two nodes are mutually visible if one node can be seen from the other. We record these visible connections using Line 21 and 22 of Algorithm 6. Therefore, when executing the inverted expansion in Zeta*-SIPP, there is no need to compute the field of view for the closed nodes.

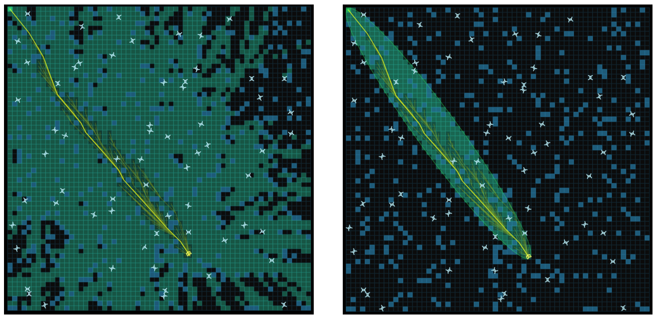
(a) TO-AA-FoV-SIPP (b) Zeta/Zeta*-SIPP

Figure 2: Screenshots of TO-AA-FoV-SIPP and Zeta/Zeta*-SIPP

3.5 Data Structure

In general, for SIPP-based planners, a node n can be represented by $(p, [t_1, t_2])$ where p is the location of the node n and $[t_1, t_2]$ is the safe interval. For the same location p, there may exist multiple nodes with different safe intervals. The node n can be regarded as a spatio-temporal point while the location p is only a spatial point. In graph-based path planning, p usually refers to a vertex or a grid. Therefore, the nodes in the proposed algorithms can be viewed as having two levels: the "node" level (space-time) and the "grid" level (space). Focusing solely on the "node" level may result in repeated visibility checks between two grids. To avoid this issue, the visibility checks can be conducted on the "grid" level and the results can then be stored in grids. If there is a need to check whether two nodes are mutually visible, the results can be generated or called from their corresponding grids.

4 Theoretical Properties

Zeta-SIPP, TO-AA-FoV-SIPP and Zeta*-SIPP have the same theoretical properties as TO-AA-SIPP. Here we mainly prove the properties affected by the *bound* list in Zeta/Zeta*-SIPP.

Lemma 1. *The bound list always contains a node with the minimum f_{low}-value in the search space outside the open and closed lists $S \setminus (open \cup closed)$.*

Proof. Since we initialize the *bound* list by inserting all the search nodes, according to Line 11 in Algorithm 6, $bound = S \setminus (open \cup closed)$. This concludes the proof. One can adapt different approaches to generate (or expand) the *bound* list, but Lemma 1 must hold to ensure optimality. □

Lemma 2. *The node extracted from the open list at each step has the minimum f-value in the search space outside the closed list $S \setminus closed$.*

Proof. The while-loop in `forwardExpansion` (Algorithm 6) guarantees $\min_{n \in open} f(n) < \min_{n \in bound} f_{low}(n)$ after expansion. Let $S_{out} = S \setminus (open \cup closed)$. Lemma 1 indicates $\min_{n \in bound} f_{low}(n) = \min_{n \in S_{out}} f_{low}(n)$. According

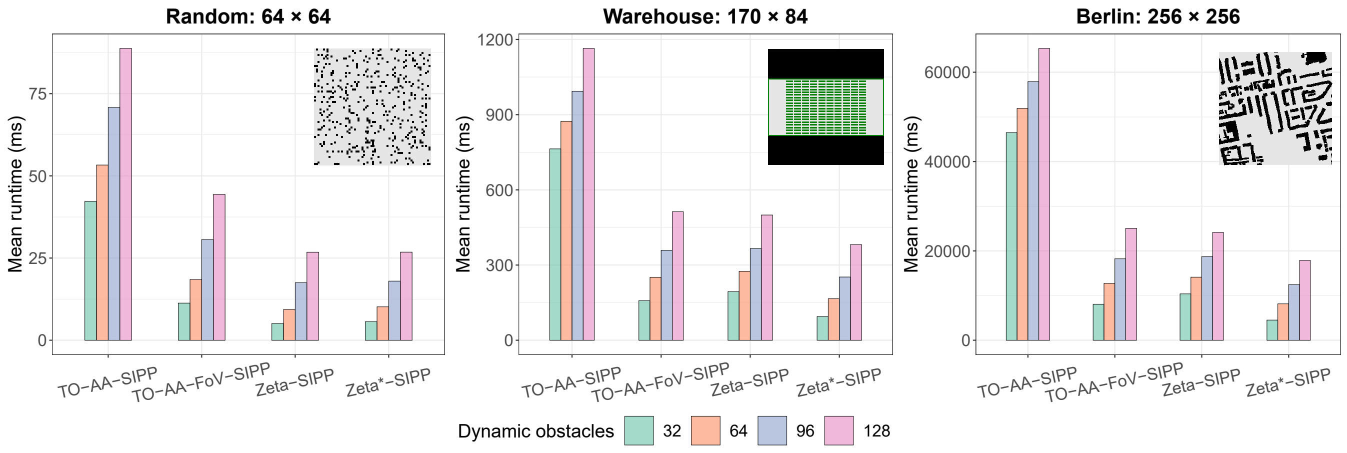

Figure 3: Mean runtime of the algorithms.

Maps	Search nodes (-SIPP)				Scanned grids (-SIPP)			
	TO-AA	TO-AA-FoV	Zeta	Zeta*	TO-AA	TO-AA-FoV	Zeta	Zeta*
Random	7.11×10^3	3.56×10^3	4.56×10^2	4.56×10^2	3.15×10^6	2.60×10^4	1.75×10^5	2.36×10^4
Warehouse	1.71×10^4	8.11×10^3	2.50×10^3	2.50×10^3	6.01×10^7	3.92×10^5	9.68×10^6	2.06×10^5
Berlin	6.20×10^4	2.53×10^4	8.53×10^3	8.53×10^3	4.31×10^9	1.46×10^7	7.07×10^8	6.94×10^6

Table 1: Mean number of the search nodes and scanned grids

to $f_{low}(n) \leq f(n)$, $\min_{n \in open} f(n) < \min_{n \in S_{out}} f(n)$. This concludes the proof. □

Theorem 1. *Zeta/Zeta*-SIPP is complete and optimal.*

Proof. Since TO-AA-SIPP has already proved to be complete and optimal, we only need to prove the main search procedure (Algorithm 2) of Zeta/Zeta*-SIPP is equivalent to that of TO-AA-SIPP. Let the open list of Zeta/Zeta*-SIPP be $open$ and the open list of TO-AA-SIPP be $open'$. In TO-AA-SIPP, since all search nodes are inserted into $open'$ at initialization, $open' = S \setminus closed$. In Zeta/Zeta*-SIPP, Lemma 2 indicates $\min_{n \in open} f(n) = \min_{n \in S \setminus closed} f(n)$. Therefore, $\min_{n \in open} f(n) = \min_{n \in open'} f(n)$. This means that the node extracted from the open list at each step in Zeta/Zeta*-SIPP is identical to that of TO-AA-SIPP. However, Lemma 2 cannot guarantee this equation still holds after extraction. Therefore, we have to add $g(n) + h(n) \leq \min_{n \in bound} f_{low}(n)$ in Line 15 of Algorithm 2. Then based on Lemma 1, the adjusted condition in Line 15 can prove to be equivalent to the original one in TO-AA-SIPP. □

5 Empirical Analysis

We implemented TO-AA-FoV-SIPP, Zeta-SIPP and Zeta*-SIPP in our web-based pathfinding visualizer[1], as shown in Figure 2. It is clear to see that Zeta/Zeta*-SIPP forms an elliptical boundary to limit the search range. The nodes outside the boundary have no contribution to finding the optimal path, and thus there is no need to insert them into $open$.

To measure the performance of the proposed algorithms, we performed experiments on three different benchmark maps [Stern *et al.*, 2019]: *Random-64-64-10*, a 64×64 map with 10% of randomly blocked grids; *Warehouse-10-20-10-2-2*, a 170×84 map from a logistics domain; *Berlin_1_256*, a 256×256 real-world city map. These three maps were chosen because they represent different types of environment and differ in size. For each map, 500 scenarios were generated by the following steps: 1) Chose 25 benchmark scenario sets (random) [Stern *et al.*, 2019]. 2) For each scenario set, we took the last 20 scenarios as tests and the top 32/64/96/128 scenarios as dynamic obstacles. 3) The trajectories of dynamic obstacles were generated successively by Zeta*-SIPP, which were collision-free and contained any-angle moves. All the algorithms were implemented in JavaScript and the experiments were performed on Node.js v18.14.2 on a laptop with 2.30GHz Intel Core i7-11800H and 16 GB RAM.

Figure 3 shows the mean runtime of different algorithms. On average, the proposed three algorithms all outperformed TO-AA-SIPP and Zeta*-SIPP was the best among them. In the Random map, Zeta-SIPP has similar performance compared with Zeta*-SIPP. This is because when the map size is small, the advantage of Field of View is rather inconspicuous. In the Warehouse and Berlin maps, the performance difference between Zeta-SIPP and TO-AA-FoV-SIPP diminishes. When the dynamic obstacles are 32 and 64, TO-AA-FoV-SIPP is even faster than Zeta-SIPP. In the Berlin map, the mean runtime of TO-AA-SIPP surpasses 40-60 seconds, exemplifying the algorithm's relative inefficiency and thereby

[1]URL: *http://dronectr.tudelft.nl/*, ID: *Zeta**. Our implementation is available at *https://github.com/yiyuanzou/zeta-sipp*.

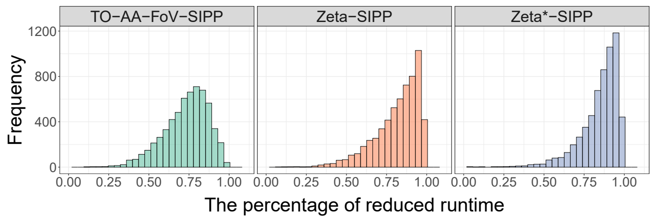

Figure 4: The frequency distributions of the percentage of reduced runtime in relation to the original TO-AA-SIPP runtime.

restricting its real-world applicability. Zeta*-SIPP substantially improves TO-AA-SIPP and the mean runtime is generally reduced by around 70%-90%.

Table 1 shows the mean search nodes and scanned grids of the algorithms regardless of the number of dynamic obstacles. The search nodes indicate the nodes visited by the algorithms whereas the scanned grids mean the grids checked by Line of Sight or Field of View. Zeta/Zeta*-SIPP effectively reduces the search nodes of TO-AA-SIPP while maintaining the capability to find time-optimal paths. The search nodes are basically reduced by about 85%-95%. It is apparent that Field of View scans much fewer grids than Line of Sight, especially when the map is large. This also illustrates why TO-AA-FoV-SIPP performs better than Zeta-SIPP in the Warehouse and Berlin maps.

In Figure 3 and Table 1, we show means rather than medians because medians may disregard some worst cases with extremely slow runtime. For example, when there are 32 dynamic obstacles in the Berlin map, relying solely on the median runtime may lead us to believe that Zeta*-SIPP is approximately 20 times faster than TO-AA-SIPP. However, in Figure 3, the mean runtime of Zeta*-SIPP is around 10% of that of TO-AA-SIPP (only 10 times speedup). The improvements should be viewed with caution.

To further evaluate the improvements, Figure 4 shows the frequency distributions of the percentage of reduced runtime in relation to the original TO-AA-SIPP runtime. The range of the x-axis is limited to $[0, 1]$ since only the positive values represent improvements. It is worth noting that in 99.4% of scenarios, all the proposed algorithms outperformed TO-AA-SIPP. Hence, omitting the part less than 0 has little effect. The distributions illustrate the superiority of Zeta*-SIPP. In majority (88.4%) of scenarios, Zeta*-SIPP reduces the runtime of TO-AA-SIPP by over 70%. Zeta-SIPP and Zeta*-SIPP both exhibit outstanding performance in the range $[0.9, 1]$ compared with TO-AA-FoV-SIPP because their search ranges are limited by the elliptical region between the start and target. If the optimal path is almost a straight line, Zeta/Zeta*-SIPP may quickly find it and could be more than 10 times faster than TO-AA-SIPP.

6 Conclusion

Optimal any-angle path planning with dynamic obstacles remains relatively underexplored, with only a handful of algorithms making significant contributions to this area of research. TO-AA-SIPP is one of the important works in this field. However, the efficiency of this algorithm is an issue, posing difficulties for its applications. Therefore, in this paper, we propose two different directions to improve TO-AA-SIPP: 1) reduce useless search nodes by any-angle forward expansion (Zeta-SIPP), and 2) replace Line of Sight with Field of View for visibility checks with static obstacles (TO-AA-FoV-SIPP). Combining these two ideas, Zeta*-SIPP is further developed. The initial experimental results of Zeta*-SIPP are promising, reducing the runtime of TO-AA-SIPP by around 70%-90% on average.

To enhance the applicability of the algorithms in real-world scenarios, future work could consider incorporating domain-specific constraints into the algorithms. For instance, the constraints on turning angles could be involved to account for maneuver restrictions of vehicles. In this case, the algorithm may be even faster as the search space is reduced. Furthermore, current algorithms only focus on 2D scenarios. In the future, 3D any-angle path planning with dynamic obstacles can also be explored (e.g., drone path planning).

Ethical Statement

There are no ethical issues.

Acknowledgments

The authors would like to thank the financial support from the China Scholarship Council (CSC) No. 202106830036.

References

[Bergström, 2001] Björn Bergström. FOV using recursive shadowcasting. http://www.roguebasin.com/index.php?title=FOV_using_recursive_shadowcasting, 2001. Accessed on Feb. 10th, 2023.

[Bresenham, 1965] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[Daniel *et al.*, 2010] Kenny Daniel, Alex Nash, Sven Koenig, and Ariel Felner. Theta*: Any-angle path planning on grids. *J. Artif. Int. Res.*, 39(1):533–579, sep 2010.

[Gammell *et al.*, 2014] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy 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*, pages 2997–3004, 2014.

[Gammell *et al.*, 2015] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. 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)*, pages 3067–3074, 2015.

[Harabor *et al.*, 2016] Daniel Harabor, Alban Grastien, Dindar Öz, and Vural Aksakalli. Optimal any-angle pathfinding in practice. *J. Artif. Int. Res.*, 56(1):89–118, may 2016.

[Hart *et al.*, 1968] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Nash *et al.*, 2010] Alex Nash, Sven Koenig, and Craig Tovey. Lazy Theta*: Any-angle path planning and path length analysis in 3d. *Proceedings of the AAAI Conference on Artificial Intelligence*, 24(1):147–154, Jul. 2010.

[Phillips and Likhachev, 2011] Mike Phillips and Maxim Likhachev. SIPP: Safe interval path planning for dynamic environments. In *2011 IEEE International Conference on Robotics and Automation*, pages 5628–5635, 2011.

[Silver, 2005] David Silver. Cooperative pathfinding. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 1(1):117–122, Sep. 2005.

[Stern *et al.*, 2019] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Eli Boyarski, and Roman Bartak. Multi-agent pathfinding: Definitions, variants, and benchmarks. *Symposium on Combinatorial Search (SoCS)*, pages 151–158, 2019.

[Uras and Koenig, 2015] Tansel Uras and Sven Koenig. An empirical comparison of any-angle path-planning algorithms. In *Proceedings of the International Symposium on Combinatorial Search*, volume 6, pages 206–210, 2015.

[Yakovlev and Andreychuk, 2017] Konstantin Yakovlev and Anton Andreychuk. Any-angle pathfinding for multiple agents based on sipp algorithm. *Proceedings of the International Conference on Automated Planning and Scheduling*, 27(1):586–594, Jun. 2017.

[Yakovlev and Andreychuk, 2021] Konstantin Yakovlev and Anton Andreychuk. Towards time-optimal any-angle path planning with dynamic obstacles. *Proceedings of the International Conference on Automated Planning and Scheduling*, 31(1):405–414, 2021.

[Yap *et al.*, 2011] Peter Yap, Neil Burch, Robert Holte, and Jonathan Schaeffer. Block A*: Database-driven search with applications in any-angle path-planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 25(1):120–125, Aug. 2011.