

WAP: Wavefront Arithmetic Propagation Algorithm and Implementation for Motion Planning Among Dynamic Obstacles

Shijie Bao¹[0009–0008–3258–1117], Shizhe Zhao¹, and Zhongqiang Ren¹

Shanghai Jiao Tong University
thetoror@sjtu.edu.cn

Abstract. This paper considers Motion Planning among Dynamic Obstacle (MPDO), which seeks collision-free path for a point robot with maximum speed limit from given start to goal position in 2D continuous space among dynamic obstacles along known trajectories while minimizing the arrival time. Existing algorithms can either find feasible solutions for MPDO without solution optimality guarantees, or solve special cases of MPDO to optimality using continuous Dijkstra paradigm. A major difficulty in MPDO is to compute the reachable space of the robot among dynamic obstacles which involves boolean operation over complex point sets. This paper proposes a new method Wavefront Arithmetic Propagation (WAP) to address this difficulty by converting these complex boolean operations to efficient arithmetic operations on a new data structure which we refer to as the wavelet graph. Additionally, we implemented the proposed WAP, which is rare in the field of continuous Dijkstra, and compare against mixed integer programming (MIP) baselines. Results show that, WAP can find optimal paths in complex environments among tens of dynamic obstacles in less than a second while achieving earlier arrival time than the baseline methods.

Keywords: Motion and Path Planning, Dynamic Obstacles, Continuous Dijkstra

1 Introduction

Motion planning is of fundamental importance in robotics [4, 9, 13, 14]. This paper considers Motion Planning among Dynamic Obstacle (MPDO), which seeks collision-free path for a point robot with maximum speed limit from given start to goal position in 2D continuous space while minimizing the arrival time. In particular, the obstacles in MPDO can be static, transient (i.e., appear or disappear at any time), or dynamic (i.e., moving along known trajectories). When the obstacles are static, there are algorithms for finding an optimal or bounded sub-optimal solutions in polynomial time [4, 13, 14]. In the presence of dynamic obstacles, this motion planning problem is known to be NP-Hard [3].

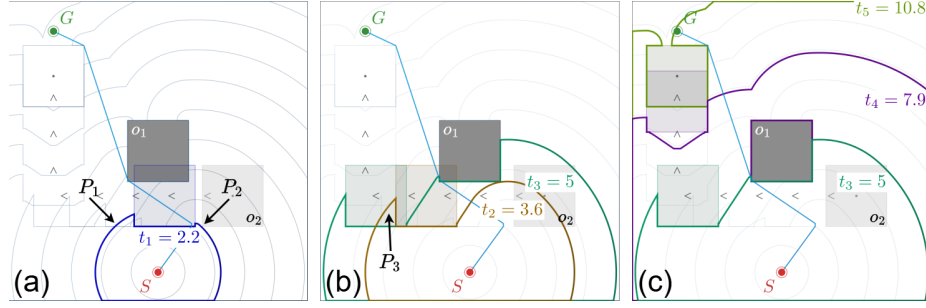


Fig. 1: An MPDO instance and the earliest-arrival solution produced by WAP. There is a static obstacle o_1 (dark square) and a dynamic obstacle o_2 , with light gray squares indicating the start and gray triangular waypoints indicating its trajectory. The time axis is discretized with $t_\Delta = 0.1$. The light blue curves show the boundaries of the reachable sets for every 1 second. The blue line from S to G is the computed earliest-arrival path. (a) The deep blue curve shows the boundary of the reachable set at time $t_1 = 2.2$. For boundary points P_1 and P_2 , the robot cannot reach them earlier than t_1 (b) The brown and green curves show the boundary of the reachable set at times $t_2 = 3.6$ and $t_3 = 5$ respectively. Due to dynamic obstacles, reachable points may become inaccessible at later times. For example, point P_3 is reachable at time t_2 but blocked by the obstacle o_2 at time t_3 . (c) The violet and olive green curves show the boundary of the reachable set at times $t_4 = 7.5$ and $t_5 = 10$ respectively. WAP terminates when the reachable set contains the goal point G at time t_5 .

While there are many approaches to find a feasible (sub-optimal) solution for MPDO, we are not aware of any algorithm in the literature that can solve MPDO to optimality. For special cases of MPDO, there are methods to find an optimal solution. When obstacles are convex polygons moving along fixed directions with constant speeds, optimal solution can be found by constructing an “accessibility graph” [7] in 3D where time is added as the third dimension. When obstacles are polygons, static or transient, Continuous Dijkstra (CD) paradigms [6, 10] can be leveraged to compute an optimal solution by alternating between propagating the wavefront and detect events where the wavefront intersects with each other or the obstacles. When both obstacles and robot path are rectilinear, an improved version of the CD with reduced time complexity has recently been developed in [15]. While these methods find optimal solutions in theory, we are not aware of any implementations or any numerical results of these algorithms.

CD [6, 10, 15–17] is of close relevance to our work. However, extending CD to address MPDO is challenging. The idea in CD is to propagate wavelets from the start in the free space while detecting “events” when wavelets intersect each other or obstacles, which thereafter generate new or modify the existing wavelets. The MPDO problem considered here generalizes the notion of transient obstacles in [6] in a sense that the obstacles can not only appear or disappear at any time at any position, but also move along known trajectories, which leads to difficulty in detecting those events and can possibly lead to infinitely many events.

Furthermore, existing CD only considers the newly swept region by the wavelet during propagation without worrying about obstacles entering the previously reachable sets. Here, we have to consider obstacles re-entering the previously reachable sets, which may become unreachable due to the moving obstacles.

To address the challenge, we propose a new CD-based algorithm which we call Wavefront Arithmetic Propagation (WAP) which is fundamentally different from existing ones. Instead of relying on the notion of events, WAP discretizes the time-axis into time steps, and propagates the wavelets between time steps. Then, at each time step, WAP computes the resulting wavelets based on boolean operations over the previous wavelets, the newly swept areas and the obstacles. There is no obvious way to directly conduct these boolean operations as it is hard to represent arbitrary set of points. We bypass this difficulty by (i) introducing a data structure, which we call wavelet graphs (WGs), to capture the topology of the boundary of point sets, (ii) converting the boolean operations over sets to arithmetic operations in WGs, and (iii) developing efficient algorithms to conduct these arithmetic operations.

Besides the proposed algorithm, another contribution is the implementation and numerical evaluation, which is rare in the field of CD for motion planning. We compare our WAP against mixed-integer programming (MIP) approaches [20, 25] that also relies on discrete time step representation. The results show that WAP runs 2.93x to 4.37x faster than baselines, provides optimality guarantees in arrival time, and achieves near-optimal path lengths. Our video shows a complex environment with 30 dynamic obstacles where WAP finds an optimal solution in less than a second.

2 Other Related Work

While finding the optimum for MPDO is challenging, there are many algorithms to compute a *feasible* solution. By discretizing the workspace into a graph, search-based planners [19, 22] can find an optimal solution within the graph, whose quality is determined by the resolution of the discretization. Path-velocity decomposition methods [12] can also be leveraged to find a feasible solution by first finding a path among static obstacles and then finding the speeds along the path to avoid the dynamic obstacles. Sampling-based methods [5, 11] have been applied for generalizations of MPDO with motion constraints and can be used to find feasible solutions. To quantify the deviation of these feasible solutions from the optimum, one can find tight lower bounds [21] to provide posteriori sub-optimality bound of the obtained solution.

By discretizing the time dimension, one can also formulate the MPDO as a mixed-integer linear program (MILP) [1, 20, 23] or mixed-integer conic program (MICP) [25, 26], which uses big-M constraints to encode obstacle collision constraints at each time step, and invoke solvers such as Gurobi to solve the formulation to optimality. We compare our WAP against this type of methods.

3 Problem Description

Let $\mathcal{W} \subset \mathbb{R}^2$ denote a bounded 2D workspace. Let $t_\Delta > 0$ denote a fixed time interval and the time dimension is discretized into time steps $t_i = i \cdot t_\Delta, i = 0, 1, 2, \dots$. For each time t_i , let $\mathbb{O}_i = \{o_i^1, o_i^2, \dots\}$ denote a set of time-dependent transient obstacles. We assume all obstacles are polygons and their edges do not overlap¹. The boundaries of obstacles are traversable, i.e., each $o \in \mathbb{O}_i$ is an open point set. Let O_i denote the unified non-traversable space, i.e., $O_i = \bigcup_{o \in \mathbb{O}_i} o$.

The robot is a point that can move in any direction with a speed limit of $v_{max} \in \mathbb{R}^+$. Let $\vec{s}, \vec{g} \in \mathcal{W}$ represent the start and goal location of the robot. Let $\pi : \langle \vec{p}_0, \vec{p}_1, \dots, \vec{p}_n \rangle$, where $\vec{p}_0 = \vec{s}, \vec{p}_n = \vec{g}$, denote a path of the robot that reach \vec{g} at time t_n , and let \vec{p}_i denote the i -th waypoint at time t_i . MPDO seeks a path π from s to g with the minimum t_n that satisfies: (i) speed constraint: $\forall i \leq n, |\vec{p}_i - \vec{p}_{i+1}| \leq v_{max} t_\Delta$, and; (ii) collision free: $\forall i \leq n, \vec{p}_i \notin O_i$.

Remark 1. In this problem, we only consider transient obstacles that appear or disappear at each time step. The trajectory of a moving obstacle can be converted to transient obstacles by sampling its position at each time step. This problem setting does not explicitly consider collision avoidance between two adjacent time steps during the transition. In practice, this can be avoided by applying obstacle inflation that considers $v_{max} t_\Delta$.

4 Methods

4.1 Basic Concepts

We employ some basic concepts from topology that will be used in this work.

Definition 1. For a given point set $S \subset \mathcal{W}$, let $Int(S)$ and ∂S denote the interior and boundary respectively. S is a regular set (RS) if $\partial(Int(S)) = \partial S$ [24] (Fig 2(a) illustrates these concepts). Let ‘ \neg ’ denote the complement operator of a point set, i.e., $\neg S = \mathcal{W} \setminus S$. Let $B(\vec{p}, d)$ denote a closed RS in circle centered at \vec{p} with radius d , and let $B_\epsilon(\vec{p}) := \lim_{\epsilon \rightarrow 0} B(\vec{p}, \epsilon)$.

Let R_n denote the set of points that the robot can reach at time t_n , starting from t_0 at position \vec{s} . Then R_n can be represented recursively as follows:

$$\begin{cases} R_0 = \{\vec{s}\}, & \text{(base case)} \\ U_n = R_{n-1} \oplus B_\Delta, & \text{(propagation)} \\ R_n = U_n \cap \neg O_n, & \text{(restriction)} \end{cases} \quad (1)$$

where $B_\Delta = B(\vec{0}, v_{max} t_\Delta)$ and \oplus is the Minkowski sum of two sets, i.e., $X \oplus Y = \{\vec{x} + \vec{y} | \vec{x} \in X, \vec{y} \in Y, X, Y \subset \mathcal{W}\}$. In Eq. (1), U_n denotes the point set that all points in R_{n-1} moves in all directions in \mathcal{W} , and we refer to this process as *propagation*. Then U_n is clipped by O_n to ensure the reachable set is collision-free at t_n , and we refer to this process as *restriction*.

¹ We can always merge polygons with overlapping edges into one.

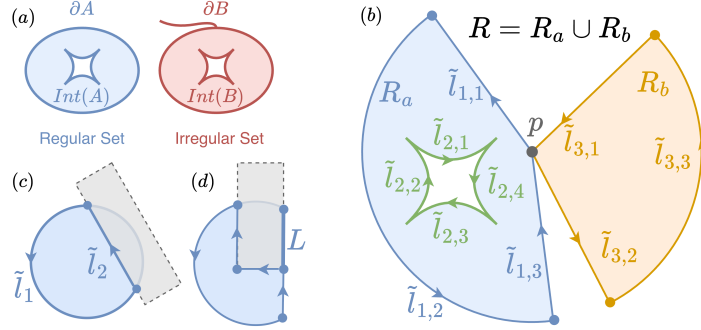


Fig. 2: (a) Examples of regular sets and irregular sets. (b) A reachable set R with two components R_a, R_b and a hole bounded by wavelets $\tilde{l}_{2,1}, \dots, \tilde{l}_{2,4}$. (c) A full-circle wavelet is split into a circular arc (blue) and a segment (green) by the boundary of an obstacle. (d) The restriction introduces the component L , which is not a regular set.

Definition 2. (*Wavefront and Wavelets*) The boundary of the reachable set at t_n , ∂R_n , is called wavefront. ∂R_n can be expressed as a set of directional curves $\{\tilde{l}_1, \tilde{l}_2, \dots\}$, where each $\tilde{l} \in \partial R_n$ is called a wavelet. The direction of wavelet will be defined in Def. 3. A wavelet \tilde{l} is a closed set consisting of three parts l^s, l^e, \tilde{l}^o . Here, l^s and l^e are start and end points of the curve, respectively, while \tilde{l}^o is the open set of \tilde{l} , excluding l^s and l^e . This means that \tilde{l}^o never intersects with other wavelets. Specifically, when the wavelet \tilde{l} is a full circle, we can choose an arbitrary point $p \in \tilde{l}$ and set $l^s = l^e = p$. All wavelets form multiple loops without any self-intersection (See in Exp 1).

Definition 3. (*Direction of Wavelets*) The direction of \tilde{l} at a point $\vec{p} \in \tilde{l}$ is defined by the unit vector tangent to \tilde{l} at \vec{p} , pointing from l^s to l^e , denoted as $\vec{p}(\tilde{l})$. For a reachable set R , the directions of wavelets determine the interior and exterior of R . Let $n_{\tilde{l}}^l(\vec{p})$ and $n_{\tilde{l}}^r(\vec{p})$ denote the unit normal vector of $\vec{p}(\tilde{l})$ pointing to left and to right w.r.t. \tilde{l} , respectively. Let $N(\tilde{l}) = \{\vec{p} \oplus B_\epsilon(\vec{p}) | \vec{p} \in \tilde{l}\}$ denote all nearby points of \tilde{l} . A point $\vec{q} \in N(\tilde{l})$ is left (or right)² to \tilde{l} if $\exists \vec{p} \in \tilde{l}$ that $n_{\tilde{l}}^l(\vec{p}) \cdot (\vec{q} - \vec{p}) > 0$ (or $n_{\tilde{l}}^r(\vec{p}) \cdot (\vec{q} - \vec{p}) > 0$ for right). For a point p that is not on the boundary ($p \notin \partial R$), p is in interior (exterior) of R if (i) $\exists \vec{q}$ that is left (right) to any wavelet of ∂R ; and (ii) there exists a continuous path that connects \vec{q} and \vec{p} without intersecting any wavelet. Jordan Curve Theorem [8] ensures that a point $\vec{p} \in R$ cannot be both in the interior and exterior of R .

Example 1. (Components of Reachable Set) In Fig. 2(b), the regular set R can be represented by two regular sets $\{R_a, R_b\}$. The boundary of R consists of three loops \tilde{l}_1, \dots (blue), \tilde{l}_2, \dots (green) and \tilde{l}_3, \dots (orange). Their directions define the interior and exterior of R . Specifically, all points inside the region defined

² We define the left/right relationship only for nearby points of a wavelet.

by green wavelets are exterior points, creating a hole in R_a . Also note that $\partial R_a \cap \partial R_b = \{p\}$, while $\text{Int}(R_a) \cap \text{Int}(R_b) = \emptyset$.

Exp. 1 shows that, due to time-dependent transient obstacles, R_n may contains multiple reachable regions with holes, and ∂R may form multiple loops. In the remaining content, we refer the different reachable regions and loops as components of R and ∂R respectively. For a given wavelet \tilde{l} , let $R(\tilde{l})$ and $\partial R(\tilde{l})$ denote the corresponding component that \tilde{l} is part of.

Remark 2. At any time $t_n > 0$, each wavelet can be represented by either a circular arc or segment. Intuitively, during the propagation from t_{n-1} to t_n , for a point $\vec{p} \in \partial R_{n-1}$, $\vec{p} \oplus B_\Delta$ is a subset of U_n , and $\partial(\vec{p} \oplus B_\Delta)$ has a single wavelet \tilde{l} (i.e., a full circle). Then the restriction may introduce a segment from the border of an obstacle that splits \tilde{l} into a circular arc \tilde{l}_1 and a segment \tilde{l}_2 (see in Fig 2(c)). We will show that wavelets of a propagation can still be represented by arcs and segments (see in Exp 2).

Note that R_n guarantees that a collision-free path exists for all points $\vec{p} \in R_n$. However, the lengths of these paths are not necessarily the same. Since our objective is to minimize arrival time, we never split a reachable set based on the path length metric.

Remark 3. R_n may contains regular set (e.g., U_n), segments L and points P (e.g., R_0), where L and P are not part of RS. Fig. 2(d) shows an scenario of generating a segmental reachable set that not be part of any regular set. To simplify discussion, for the remainder of the paper, we will refer to R_n by its regular set component and disregard L, P . In Sec. 4.4, we will show that the proposed method still works when L, P are considered.

Now we demonstrate the result of propagating a wavelet \tilde{l} . Let $S_w(\tilde{l}) \subset U_n$ denote the newly swept region of a wavelet \tilde{l} propagates from t_{n-1} to t_n , i.e.:

$$S_w(\tilde{l}) = S_w(l^s) \cup S_w(l^e) \cup S_w(\tilde{l}^o) \supseteq (\tilde{l} \oplus B_\Delta) \setminus R_{n-1}, \quad (2)$$

where $S_w(\tilde{l}^o)$ consists of points generated by moving all points in \tilde{l} to the right direction of \tilde{l} , i.e., $\{\vec{p} + v_{max} t_\Delta n_\wedge^l(\vec{p}) | \vec{p} \in \tilde{l}^o\}$; and $S_w(l^s)$ (or $S_w(l^e)$) is a quarter circle centered at l^s (or l^e) and propagates in direction $n_\wedge^l(l^s)$ (or $n_\wedge^l(l^e)$) without overlapping with $S_w(\tilde{l}^o)$. Exp. 2 demonstrates the propagation.

Example 2. (Wavelet Propagation) Fig. 3(a,b) shows the result of propagating a single wavelet \tilde{l} . $S_w(\tilde{l}^o)$ is shown in blue, dashed arrows at $v \in \tilde{l}$ are in direction $n_\wedge^l(\vec{v})$ with length $v_{max} t_\Delta$. $S_w(l^s), S_w(l^e)$ are shown in green. We can see that using quarter circles is enough to represent the propagated region such that $S_w(\tilde{l}) \supseteq (\tilde{l} \oplus B_\Delta) \setminus R_{n-1}$. Figure 3(c,d) demonstrates the result of propagating a wavefront. We can see that all wavelets expand towards the exterior region, resulting in a larger reachable region (represented by the outer blue and green wavelets) and a smaller hole (represented by the inner blue wavelets).

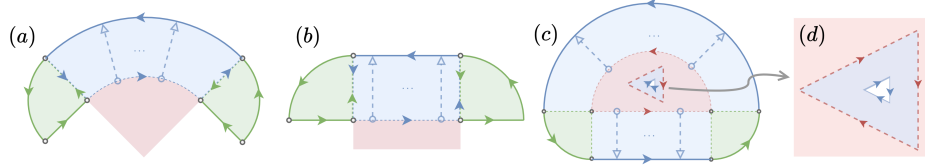


Fig. 3: Red areas represent a subset of the reachable region from the previous iteration. Green and blue areas represent the swept regions generated by propagation. Solid arrows indicate the direction of wavelets, while hollow arrows show the movement direction of a point during propagation. (a) Propagation of a single circular wavelet; (b) Propagation of a single segment wavelet; (c) Propagation on a wavefront; (d) Propagation of a hole in the wavefront;

U_n in Eq (1) can be expressed as an union of multiple regular sets (RSs)

$$U_n = R_{n-1} \cup \left(\bigcup_{\tilde{l} \in \partial R_{n-1}} S_w(\tilde{l}) \right). \quad (3)$$

Eq (3) suggests that we can compute U_n and R_n via boolean operators on finite number of RSs. For this purpose we introduce the concept, *indicator scalar field* (ISF) of a regular set $X \subset \mathcal{W}$; and we define boolean operators to compute ISF of X from subsets of it.

Definition 4. (*Indicator Scalar Field*) For an RS, $X \subset \mathcal{W}$, $\mathcal{F}(\partial X)(\cdot)$ defines an indicator scalar field (ISF), for a point $\vec{p} \in \mathcal{W}$:

$$\mathcal{F}(\partial X)(\vec{p}) = \begin{cases} 0, & \text{if } \vec{p} \notin (\partial X \cup \text{Int}(X)) & (\text{exterior point}) \\ 1, & \text{if } \vec{p} \in \partial X & (\text{boundary point}) \\ 2, & \text{if } \vec{p} \in \text{Int}(X) & (\text{interior point}), \end{cases} \quad (4)$$

Next we define the boolean operators of $\mathcal{F}(\cdot)$ for two RSs $X_1, X_2 \subset \mathcal{W}$.

Union ISF of $X = X_1 \cup X_2$ is expressed as follows:

$$\mathcal{F}(\partial X)(\vec{p}) = \begin{cases} f(\mathcal{F}(\partial X_1) + \mathcal{F}(\partial X_2)) & \text{for } \vec{p} \notin \partial X_1 \cap \partial X_2 \\ r(X_1 \cup X_2, \partial X_1 \cap \partial X_2, \vec{p}) & \text{for } \vec{p} \in \partial X_1 \cap \partial X_2, \end{cases} \quad (5)$$

where $f(x) = \min(x, 2)$ is a saturation function indicates that if \vec{p} is an interior point any of X_1, X_2 then it must be an interior point of $X_1 \cup X_2$; and $r(X, D, \vec{p})$ is a repairing function that considers $B_\epsilon(\vec{p})$ to ensure $\mathcal{F}(\partial X)$ is defined when \vec{p} locates at both boundary of X_1 and X_2 , that

$$r(X, D, \vec{p}) = \begin{cases} 0, & \text{if } \forall \vec{q} \in B_\epsilon(\vec{p}) \setminus D, \text{ that } \mathcal{F}(\partial X)(\vec{q}) = 0 \\ 2, & \text{if } \forall \vec{q} \in B_\epsilon(\vec{p}) \setminus D, \text{ that } \mathcal{F}(\partial X)(\vec{q}) = 2 \\ 1, & \text{otherwise.} \end{cases} \quad (7)$$

Eq (7) indicates that a \vec{p} is an exterior (or interior) point of X if all nearby points $B_\epsilon(\vec{p}) \setminus D$ are exterior (or interior) points, while the point \vec{p} is on the boundary if there exist both interior and exterior points in $B_\epsilon(\vec{p}) \setminus D$.

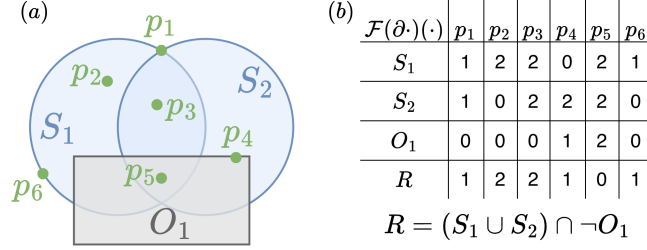


Fig. 4: (a) A regular set $R = (S_1 \cup S_2) \cap \neg O_1$ is defined by boolean operations on multiple regular sets. (b) The ISF value of points p_1, \dots, p_6 in each regular set.

Complement ISF of the complement set is:

$$\mathcal{F}(\partial(\neg X)) = 2 - \mathcal{F}(\partial X), \quad (8)$$

which turns all interior points to exterior (and vice-versa) while leaving all boundary points unchanged.

Intersection ISF of $X = X_1 \cap X_2$ is expressed as follows:

$$\mathcal{F}(\partial X)(\vec{p}) = \begin{cases} 2 - f(4 - (\mathcal{F}(\partial X_1) + \mathcal{F}(\partial X_2))), & \text{for } \vec{p} \notin \partial X_1 \cap \partial X_2, \\ r(X_1 \cap X_2, \partial X_1 \cap \partial X_2, \vec{p}) \text{ (Eq. (7))}, & \text{for } \vec{p} \in \partial X_1 \cap \partial X_2, \end{cases} \quad (9)$$

where Eq. (9) comes from De Morgan's laws that $X_1 \cap X_2 = \neg(\neg X_1 \cup \neg X_2)$ with Eq. (5) and (8). Exp. 3 shows how to leverage the boolean operators to compute the ISF value of a point.

Example 3. Fig. 4(a) shows a regular set $R = (S_1 \cup S_2) \cap \neg O_1$ that consists of multiple subsets, and Fig. 4(b) shows the ISF value of points p_1, \dots, p_6 in S_1, S_2, O_1 and R . We now demonstrate how to compute the ISF value of these points. Let $R_{12} = S_1 \cup S_2$, for p_2 we have

$$\mathcal{F}(\partial R_{12})(p_2) = f(\mathcal{F}(\partial S_1)(p_2) + \mathcal{F}(\partial S_2)(p_2)) = f(2 + 0) = 2;$$

and $\mathcal{F}(\partial \neg O_1)(p_2) = 2 - 0 = 2$. Substituting $\mathcal{F}(\partial R_{12})(p_2)$ and $\mathcal{F}(\partial \neg O_1)(p_2)$ into Eq. (9), we have $\mathcal{F}(\partial R)(p_2) = 2 - f(4 - (2 + 2)) = 2$. Using the same procedure, we have $\mathcal{F}(\partial R)(p_4) = 2 - f(4 - (2 + 1)) = 1$, and $\mathcal{F}(\partial R)(p_5) = 2 - f(4 - (2 + 0)) = 0$. ISF value for other points can be obtained in a similar way. Here we intentionally skip the case for p_1 , who is located at the boundary of both S_1 and S_2 (the repairing function (Eq (7)) is required). In Sec. 4.2, we will show that our method from the main paper does not need to address such case. A detailed discussion about this case can be found in Appendix. A.

We should strictly distinguish between a set of regular sets and their composition via boolean operators. For example, in Fig. 4, $\mathbb{R} = \{S_1, S_2, O_1\}$ is a set of regular

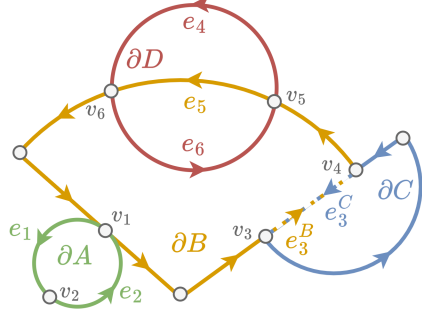


Fig. 5: Wavelet Graph, $G^w(A, B, C, D)$, is built from regular sets A, B, C, D . Each regular set forms a loop in WG. Vertex v_1 is a tangent point between ∂A and ∂B , splitting ∂A into e_1, e_2 . Here, v_2 is an arbitrarily chosen endpoint for the full-circle wavelet of ∂A . Two wavefronts, $\partial B, \partial C$ have a wavelet overlapping at v_3 and v_4 , forming a duplicated edge $e_3 = e_3^B = e_3^C$, which is indicated by the dashed line. Edges e_4, e_5, e_6 connect vertices v_5, v_6 , where e_4, e_5 are in the same direction, indicating that G^w can be a multi-graph.

sets, while R is a new regular set composed by boolean operators on \mathbb{R} . \mathbb{R} and R represent different set of wavefronts. Next, we introduce *Wavelet Graph*, which allows us to compute R_n within finite steps by leveraging the concept of ISF.

Definition 5. (*Wavelet Graph*) For a set of regular sets, $\mathbb{P} = \{P_1, \dots, P_n\}$, let $\tilde{l}_{a,i}$ denote the i -th wavelet of $P_a \in \mathbb{P}$. A *Wavelet Graph* (WG), denoted as $G^w(\mathbb{P}) = (V, E)$ is a directed graph constructed from all wavelets in \mathbb{P} . In short, V consists of the ending points of all wavelets in \mathbb{P} and the points located on multiple wavelets. Each $e \in E$ is a subset of one or more wavelets. Formally speaking:

Vertices For a single wavelet \tilde{l} , its start and end points are graph vertices, i.e., $l^s, l^e \in V$. For a pair of wavelets $\tilde{l}_{a,i} \in P_a, \tilde{l}_{b,j} \in P_b$, where $a \neq b$: (1) if $\tilde{l}_{a,i}$ and $\tilde{l}_{b,j}$ intersect at \vec{p} , then $\vec{p} \in V$; (2) if $\tilde{l}_{a,i}$ and $\tilde{l}_{b,j}$ overlap along a curve whose endpoints are \vec{p} and \vec{q} , then $\vec{p}, \vec{q} \in V$.

Edges A wavelet \tilde{l} is split into multiple edges by $v \in V \cap \tilde{l}$, for an edge $e = (u, v)$, let $A(e)$ denote the set of wavelets intersect or overlap with e : (1) if u, v are from ending points of overlapped wavelets \tilde{l}_1, \tilde{l}_2 , then e has both directions of \tilde{l}_1, \tilde{l}_2 , and $A(e) \supseteq \{\tilde{l}_1, \tilde{l}_2\}$; (2) otherwise e is subset of a single \tilde{l} and has the same direction as \tilde{l} , and $A(e) = \{\tilde{l}\}$. We refer to an edge e for which $|A(e)| > 1$ as a duplicated edge. All edges are open set excluding their end points, i.e., $V \cap E = \emptyset$.

WG is a directed cyclic graph where each cycle represents a loop on the wavefront of $P_i \in \mathbb{P}$. WG is also a multi-graph, meaning there can be multiple edges between the same pair of vertices. For example, in Fig. 5, $e_4 = (v_5, v_6)$ and $e_5 = (v_5, v_6)$ are different edges. Next, we show an important property of WG that enables us to compute the reachable set R_n within finite steps.

Lemma 1. Consider a WG, $G^w(\mathbb{P}) = (V, E)$, where $\mathbb{P} = \{P_1, \dots, P_n\}$. Then for any points \vec{p}, \vec{q} on the same edge $e \in E$, for all $P_i \in \mathbb{P}$, we have:

$$\mathcal{F}(\partial P_i)(\vec{p}) = \mathcal{F}(\partial P_i)(\vec{q}). \quad (10)$$

Furthermore, for any \vec{p}, \vec{q} on a continuous curve $c \in \mathcal{W}$, if $c \cap G^w(\mathbb{P}) = \emptyset$ then Eq. (10) also holds.

Algorithm 1 WAP

```

1: procedure WAP( $\vec{s}, \vec{g}, v_{max}, t_\Delta, \mathcal{W}$ )
2:    $n, R_0 \leftarrow 0, \{\vec{s}\}$ 
3:   while  $g \notin R_n$  do ▷ Check whether  $\mathcal{F}(\partial R_n)(g) = 0$ 
4:      $n \leftarrow n + 1$ 
5:      $\mathbb{P}_n \leftarrow \text{PROPAGATION}(R_{n-1}, v_{max}, t_\Delta)$  ▷ Eq. (1) and (2)
6:      $G^w \leftarrow \text{BUILDWG}(\mathbb{P}_n, \mathbb{O}_n)$  ▷ Def. 5
7:      $\lambda_1(\cdot), \lambda_2(\cdot) \leftarrow \text{CALCALLLAMBDA}(G^w, \mathbb{P}_n, \mathbb{O}_n)$  ▷ Alg. 2
8:      $\partial R_n \leftarrow \{e \in G^w \mid \gamma(\lambda_1(e), \lambda_2(e)) = 1\}$  ▷ Eq. (13)
9:     if  $R_n = \emptyset$  then
10:       return No Solution
11:   return  $n$  ▷  $g$  is reachable at  $t_n$ 

```

Sketch Proof. Assume there are an edge $e \in E$, a pair of points $\vec{p}, \vec{q} \in e$ and a RS $P_i \in \mathbb{P}$, such that $\mathcal{F}(P_i)(\vec{p}) \neq \mathcal{F}(P_i)(\vec{q})$. Then, moving from \vec{p} to \vec{q} along e must cross the boundary of a RS $P_j \in \mathbb{P}$ (where i, j might be the same), which contradicts the assumption that $\vec{p}, \vec{q} \in e$. \square

4.2 WAP Algorithm

We solve MPDO by finding the minimum n such that $g \in R_n$. Computing R_n is equivalent to finding all wavelets of ∂R , which can be achieved by identifying all points $\vec{p} \in \mathcal{W}$ such that $\mathcal{F}(\partial R_n)(\vec{p}) = 1$. For this purpose, we can construct a WG (Def. 5) from all regular sets (RSs) involved in propagation and restriction. According to Lemma 1, we only need to evaluate a single point on each edge of the WG.

Next we establish the expressions to compute $\mathcal{F}(\partial R_n)(\vec{p})$ when \vec{p} is not on a duplicated edge. In Appendix. C, we demonstrate how to compute $\mathcal{F}(\partial R_n)(\vec{p})$ when \vec{p} is on a duplicated edge. For simplicity, in the reminder of the paper, we assume that duplicated edges do not exist unless explicitly mentioned.

Let $\mathbb{P}_n = \{R_{n-1}\} \cup \{S_w(\vec{l}) \mid \vec{l} \in \partial R_{n-1}\}$ denote all RSs involved in propagation, and let $G^w : (E, V)$ denote the WG constructed from \mathbb{P}_n and \mathbb{O}_n (obstacle set at t_n). For a point \vec{p} on an edge of G^w , let

$$\begin{cases} \lambda_1(\vec{p}) &= \sum_{P \in \mathbb{P}_n} \mathcal{F}(\partial P)(\vec{p}), \end{cases} \quad (11)$$

$$\begin{cases} \lambda_2(\vec{p}) &= \sum_{O \in \mathbb{O}_n} \mathcal{F}(\partial O)(\vec{p}), \end{cases} \quad (12)$$

$$\gamma(a, b) = 2 - f(2 - f(a) + f(b)), \quad (13)$$

Lemma 2. $\mathcal{F}(\partial R_n)(\vec{p}) = \gamma(\lambda_1(\vec{p}), \lambda_2(\vec{p}))$ holds when \vec{p} is not on a duplicated edge of G^w . (We provide a proof for Lemma 2 in Appendix B.)

Alg. 1 demonstrates the key procedures of WAP. In each iteration, we first propagate all wavelets from the previous iteration according to Eq. (2) and obtain

Algorithm 2 Compute λ_1, λ_2 of All Edges

```

1: procedure CALCALLLAMBDA( $G^w, \mathbb{P}_n, \mathbb{O}_n$ )
2:    $E_{todo} \leftarrow E(G^w)$  ▷ Edge set to be evaluated
3:   for  $e \in E_{todo}$  do
4:      $m \leftarrow$  middle point of  $e$ 
5:      $\lambda_1(e) \leftarrow \sum_{P \in \mathbb{P}_n} \mathcal{F}(\partial P)(m)$  ▷ Eq. (11)
6:      $\lambda_2(e) \leftarrow \sum_{O \in \mathbb{O}_n} \mathcal{F}(\partial O)(m)$  ▷ Eq. (12)
7:     if use spread then ▷ Improvement by spread, discussed in Sec. 4.3
8:        $E_{spread} \leftarrow \text{SPREAD}(e, \lambda_1, \lambda_2, \emptyset, G^w, \mathbb{P}_n, \mathbb{O}_n)$  ▷ Alg. 3
9:        $E_{todo} \leftarrow E_{todo} \setminus E_{spread}$ 
10:    return  $\lambda_1, \lambda_2$ 
11: procedure  $\mathcal{F}(\partial R)(p)$ 
12:    $q, cnt \leftarrow (\infty, \infty), 0$  ▷  $q$  is far enough that outside  $R$ 
13:    $\tilde{q}p \leftarrow \text{RAY}(q, p)$ 
14:   for  $\tilde{l} \in \partial R$  do
15:     for  $v \in \text{INTERSECT}(\tilde{q}p, \tilde{l})$  do ▷ Compute all intersection points
16:       if  $|vp| = 0$  then
17:         return 1 ▷ On boundary
18:        $\Delta \leftarrow \text{SIGN}(\tilde{q}p \cdot n_{\tilde{l}}^l(\tilde{v}))$  ▷  $\text{SIGN}(x) \Rightarrow -1 : x < 0, 0 : x = 0, 1 : x > 0$ 
19:        $cnt \leftarrow cnt + \Delta$ 
20:   return  $2 \times cnt$ 

```

\mathbb{P}_n (line 5). Then we build the Wavelet Graph, $G^w(\mathbb{P}_n, \mathbb{O}_n)$, based on Def. 5 (line 6). Next, we compute the value of λ_1, λ_2 of all edges (line 7), which will be discussed next. Finally, according to Eq. (13), we can identify all wavelets of ∂R_n by selecting those edges for which $\gamma(\lambda_1(e), \lambda_2(e)) = 1$ (line 8). The algorithm terminates at the first time R_n contains g (line 3), which can be determined by evaluating $\mathcal{F}(\partial R_n)(g)$. There are two cases when there is no solution: (i) R_n becomes an empty set, meaning that there is no reachable point from t_n , then the algorithm terminates and confirms that there is no solution (line 10); (ii) The algorithm keeps iterating and never terminate, e.g., when \vec{s}, \vec{g} are in different connected components.

Alg. 2 computes λ_1, λ_2 for all edges in WG (line 3 to 10). Lines from 7 to 9 are enhancements that will be discussed in Sec. 4.3. To compute $\lambda_1(e), \lambda_2(e)$, we firstly select a point $m \in e$ (line 4) then evaluate the summation of $\mathcal{F}(\cdot)(m)$ according to Eq. (12) and (11) (line 5 to 6). Here m can be chosen arbitrarily (Lemma 1), in our case, it is the middle point of e .

Procedure $\mathcal{F}(\partial R)(\vec{p})$ (Alg 2, line 12 to 20) computes the ISF value at \vec{p} of a regular set R according to Def 4. Line 12 to 20 can be viewed as an adaption of the well-known *Ray Crossing* algorithm [18], which tests whether a point is in a polygon. The main difference is that, we compute intersections between a ray $\tilde{q}p$ and a wavelet $\tilde{l} \in \partial R$ (line 15), where \tilde{l} can be either a circular arc or a segment (Remark 2). For each intersection point v , if $|vp| = 0$, we can confirm \vec{p} is on boundary and directly return (line 17). Otherwise, we use the dot product between $\tilde{q}p$ and $n_{\tilde{l}}^l(\tilde{v})$ to determine whether $\tilde{q}p$ is entering the

Algorithm 3 Spread λ_1 and λ_2 to Other Edges

```

1: procedure SPREAD( $e : (u, v), \lambda_1, \lambda_2, E_{done}, G^w, \mathbb{P}_n, \mathbb{O}_n$ )
2:    $E_{done} \leftarrow E_{done} \cup \{e\}$  ▷ Record  $e$  has been done
3:   ▷ All edges in  $G^w$  at  $v$  are arranged by polar angle in CCW
4:    $next \leftarrow \text{NEXTADJCW}(v, e)$  ▷ Pick the edge next to  $e$ 
5:   if  $next \in E_{done}$  then ▷ Each edge only be evaluated once.
6:     return  $E_{done}$ 
7:    $p_e, p_n := p_e \in e, p_n \in next$  ▷ Pick a point on  $e$  and  $n$ .
8:    $\Delta\lambda_1(e \rightarrow next) \leftarrow 0$ 
9:    $\Delta\lambda_2(e \rightarrow next) \leftarrow 0$ 
10:  if  $R(next) \in \mathbb{P}_n$  then
11:     $\Delta\lambda_1(e \rightarrow next) \leftarrow \int_{p_e}^{p_n} \nabla \mathcal{F}(\partial P)(x) \cdot dx$  ▷ Integration according to Eq. (14)
12:  else
13:     $\Delta\lambda_2(e \rightarrow next) \leftarrow \int_{p_e}^{p_n} \nabla \mathcal{F}(\partial P)(x) \cdot dx$ 
14:   $\lambda_1(next) \leftarrow \lambda_1(e) + \Delta\lambda(e \rightarrow next)$ 
15:   $\lambda_2(next) \leftarrow \lambda_2(e) + \Delta\lambda(e \rightarrow next)$ 
16:   $E_{done} \leftarrow E_{done} \cup \text{SPREAD}(next, \lambda_1, \lambda_2, E_{done}, G^w, \mathbb{P}_n, \mathbb{O}_n)$  ▷ Spread recursively
17:  return  $E_{done}$ 

```

interior (or exterior), and we then increase (or decrease) the counter accordingly (line 18). Specially, if $\vec{q}\vec{p} \cdot n_{\curvearrowright}^l(\tilde{v}) = 0$, it implies that $\vec{q}\vec{p}$ is tangent to a circular wavelet or collinear with a segment wavelet. Both cases should not affect the counter. Note that lines 15 to 18 might be affected by numeric errors, leading to an incorrect ISF value. A detailed discussion on numerical errors can be found in Appendix D.

4.3 Improvement Through Gradient Integration

Computing $\lambda_1(e), \lambda_2(e)$ for all $e \in E$ can be expensive, we now present a method to speed up this procedure. Since $\mathcal{F}(\partial R)(\cdot)$ is defined in a continuous space, we can construct its gradient field, $\nabla \mathcal{F}(\partial R)$. This enables us to define $\mathcal{F}(\partial R)(\vec{p})$ as a result of gradient integration. Specifically, consider a regular set R , a wavelet $\tilde{l} \in \partial R$, and any point $\vec{p} \in \tilde{l}$. Let $\vec{q}_l, \vec{q}_r \in B_\epsilon(\vec{p})$ denote nearby points of \vec{p} left and right to \tilde{l} respectively. The gradient field can be constructed as follows:

$$\begin{cases} \nabla \mathcal{F}(\partial R)(\vec{p}) &= 0, \quad \forall \vec{p} \notin \partial R \\ \int_{\vec{p}}^{\vec{q}} \nabla \mathcal{F}(\partial R)(\vec{r}) \cdot d\vec{r} &= 0, \quad \forall \vec{q} \in \tilde{l}^o \\ \int_{\vec{q}_r}^{\vec{p}} \nabla \mathcal{F}(\partial R)(\vec{r}) \cdot d\vec{r} &= 1 \\ \int_{\vec{q}_l}^{\vec{p}} \nabla \mathcal{F}(\partial R)(\vec{r}) \cdot d\vec{r} &= -1 \end{cases} \quad (14)$$

Eq. (14) indicates that when a point \vec{q} is not on boundary, the gradient of \vec{q} is $\vec{0}$. For any $\vec{q}, \vec{p} \in \tilde{l}$, the gradient integration result from \vec{q} to \vec{p} is always 0. If a path enters the boundary from the exterior (or interior), the gradient integration increases by 1 (or decreases by 1), and if a path exits the boundary to the exterior (or interior), the gradient integration decreases by 1 (or increases by 1).

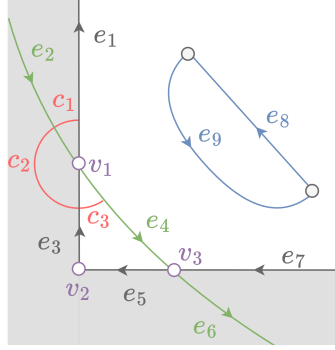


Fig. 6: A subgraph of $G^w(\mathbb{P}_n, \mathbb{O}_n)$ is shown, where shaded area represents a non-traversable region due to obstacles. The dark edges $\{e_1, e_3, e_5, e_7\}$ are on the boundary of obstacles from \mathbb{O}_n ; green and blue edges are on the boundary of regular set from \mathbb{P}_n . Assume we have $\lambda_1(e_1) = 2, \lambda_2(e_1) = 1$. By using *spread*, and following the curves c_1, c_2, c_3 (red), we can compute $\lambda_1(\cdot), \lambda_2(\cdot)$ for other edges that are in the same connected component as e_1 .

The concept of gradient integration allows us to compute the ISF value at \vec{p} from any point whose ISF value is known, rather than always from a far away point \vec{q} (Alg. 2, line 12). Specially, we have

$$\mathcal{F}(\partial R)(\vec{p}) = \mathcal{F}(\partial R)(\vec{q}) + \int_{\vec{q}}^{\vec{p}} \nabla \mathcal{F}(\partial R)(x) \cdot dx \quad (15)$$

This means that, once the values $\lambda_1(e), \lambda_2(e)$ have been evaluated (Alg. 2 line 5,6), we can use them to compute λ_1, λ_2 of all adjacent edges on G^w without expensive computation on line 5 and 6. We refer to this procedure as *spread* (Alg. 3). Exp. 4 illustrates this procedure.

Example 4. Consider a subgraph of WG, $G^w(\mathbb{P}_n, \mathbb{O}_n)$ as shown in Fig. 6. The dark edges $\{e_1, e_3, e_5, e_7\}$ are on wavelets from \mathbb{O}_n , green and blue edges are on wavelets from \mathbb{P}_n . Assume we have $\lambda_1(e_1) = 2, \lambda_2(e_1) = 1$, and we begin the spreading from e_1 . First, we sort all edges connected to v_1 in a counter-clockwise (CCW) direction, resulting in the edge list $[e_1, e_2, e_3, e_4]$. Next, we integrate along the curve c_1 to compute $\lambda_1(e_2)$ and $\lambda_2(e_2)$. Recall that $\mathbb{P}_n, \mathbb{O}_n$ only contribute to λ_1, λ_2 respectively (Eq. (11), (12)). For $\lambda_1(e_2)$ we are integrating from the interior of $R(e_2) \in \mathbb{P}_n$ to the boundary of $R(e_2)$, so $\lambda_1(e_2) = \lambda_1(e_1) - 1 = 1$; for $\lambda_2(e_2)$, we are integrating from the boundary of $R(e_1) \in \mathbb{O}_n$ to the interior of $R(e_1)$, so $\lambda_2(e_2) = \lambda_2(e_1) + 1 = 2$. Then, by following the curve c_2 , we can compute $\lambda_1(e_3) = \lambda_1(e_2) - 1 = 0$ and $\lambda_2(e_3) = \lambda_2(e_2) - 1 = 1$. Similarly, we can compute the values on e_4 via e_3 by following the curve c_3 . The same spreading process is applied on the edges connected to v_2 and v_3 . Note that, e_5 can be spread via either e_3 or e_4 . The property of gradient integration ensures that the results will be consistent. In practice, we evaluate each edge only once. Blue edges e_8, e_9 cannot be spread because they are in a different connected component from e_1 .

Spread (Alg. 3) explores G^w in a depth-first manner. For an edge $e : u \rightarrow v$ who has been evaluated, we sort all adjacent edges of e at v by polar angle in CCW, and visit the next *next* immediately after e (Line 4). This guarantees that there is no other wavelets between e and *next*, allowing us to directly apply Eq. (14)

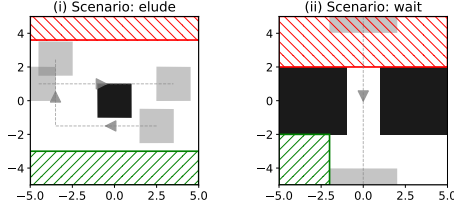


Fig. 7: Test scenarios: (a) *elude* and (b) *wait*. Black polygons are static obstacles. Gray polygons indicate the start and end positions of dynamic obstacles, with their trajectories indicated by gray dashed lines and arrows. In each scenario, the start and goal locations are randomly sampled from the green and red regions, respectively.

(Line 11). After computing the $\lambda_1(nxt), \lambda_2(nxt)$, we visit adjacent edges of nxt (Line 16) recursively. The recursion guarantees that each edge is visited only once (Line 5), ensuring that it terminates in a finite number of steps.

4.4 Discussion

Irregular Sets in R_n As discussed in Remark 3, the reachable point set R_n may contain irregular sets, which are intentionally ignored in our method. The reason is that, the operation $R_{n-1} \oplus B_\Delta$ in propagation (Eq. (1)) makes all irregular sets to regular sets. Hence, when we build WG and run procedures in Alg. 2, the theoretical foundations (Lemma 1 and 2) still hold.

Runtime Complexity Let N_i denote the number of wavelets at the i -th iteration. The complexity of a single iteration at t_i is $O(N_i^2 \cdot \log(N_i^2))$, where N_i^2 arises from the number of edges in WG, and $\log(N_i^2)$ is due to computing the intersections between wavelets using a K-D tree [2]. Existing work [6] suggests that, in the worst case, n wavelets can generate $O(n^2)$ new wavelets in a single propagation when the time dimension is continuous. As time increases, the number of wavelets can grow exponentially in the worst case, and the overall runtime complexity of our method is non-polynomial. A more precise complexity bound that takes into account discretized time could be an interesting direction for future work.

5 Experimental Results

We compare our WAP with MIP methods CP-MILP [20] and MICP [25]. All code is implemented in C++ and compiled by g++-11 with the -O2 flag. All experiments run on an Intel® Core™ i7-14700KF CPU. The code and datasets will be publicly available after acceptance.

Baselines Both CP-MILP [20] and MICP [25] work in continuous space under discretized time and max speed constraint. Additionally, they have a maximum time step constraint, T_{max} , and only consider solutions with arrival times before

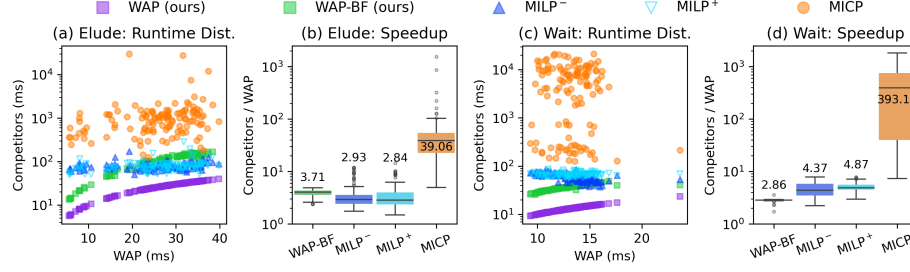


Fig. 8: Runtime comparison. (a) and (c) show the runtime of all instances in the scenarios *elude* and *wait*. Each point (x, y) represents the result of an instance, where x is the runtime of WAP, and y represents the runtime of the corresponding method; smaller values are better. (b) and (d) show the speedup ratio of WAP compared to competitors; a value greater than 1 indicates that WAP is faster.

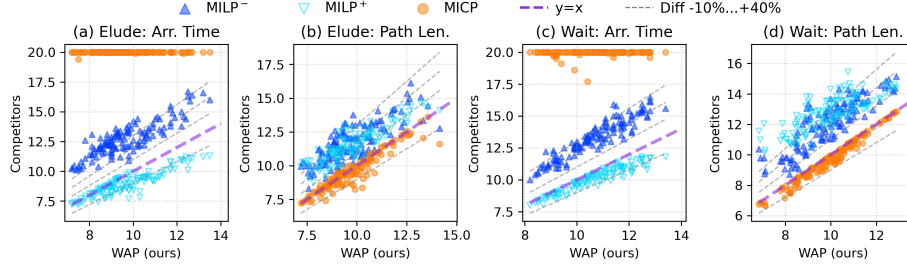


Fig. 9: Solution quality comparison. (a) and (c) show the arrival times (Arr. Time). (b) and (d) show the path lengths (Path Len.). The purple dashed line, $y = x$, represents the path quality of WAP. A point of a competitor above this line indicates that the competitor has worse quality than WAP in the corresponding instance.

T_{max} . In this experiment, we set the $t_{\Delta} = 0.1$, $v_{max} = 1$ and $T_{max} = 20$ (i.e., 200 time steps). Specifically, CP-MILP can be configured to minimize the arrival time, but its decision variables are control effort on x, y separately. This means that the reachable set of a point p per time step in CP-MILP is a square with a side length of $2v_{max}t_{\Delta}$, instead of B_{Δ} . Hence, we evaluate CP-MILP under two configurations: (1) $v_{max,x} = v_{max,y} = \frac{1}{\sqrt{2}}$, which indicates that the total maximum speed is 1, referred to as “MILP⁻”; and (2) $v_{max,x} = v_{max,y} = 1$, which indicates that the maximum speed per axis is 1, while the total max speed can be $\sqrt{2}$, this configuration is referred to as “MILP⁺”. MICP has the same reachable set per time step as WAP, but it minimizes travel distance instead of arrival time. Hence, we use the result of MICP as a reference for path length. In the experiments, we allow MICP to terminate when it reaches a 5% optimality gap. We also compare WAP with the brute-force version, i.e., without *spread*, referred to as *WAP-BF*.

Dataset We test on two representative scenarios *elude* and *wait* (Fig. 7). Each scenario includes 128 instances. The *elude* shows a situation with a static obstacle in the middle and two dynamic obstacles moving in workspace. The *wait* shows a situation where a dynamic obstacle moves across the corridor, resulting in the robot waiting for the dynamic obstacle to pass. WAP-BF has a 68% success rate in *elude* and 96% in *wait*, due to numerical errors. Other methods have a 100% success rate. To focus on the main conclusion, we have excluded failed instances from the results of WAP-BF. More experiments and discussion regarding numerical errors can be found in Appendix D.

Result Fig. 8 shows the runtime comparison. We can see that, in both scenarios, WAP is significantly faster than the others. Specifically, consider the median of speedup ratio compared to MILP⁻ (Fig. 8(b,d)), WAP is 2.93 times faster in *elude* and 4.37 times faster in *wait*. We can also see that *spread* in WAP achieves a speed-up of 3.71 times in the *elude* scenario and 2.86 times in the *wait* scenario compared to brute force. Fig. 9 compares the solution quality. We can see that WAP has a smaller arrival time compared to MILP⁻ in both scenarios, as WAP guarantees optimality in arrival time. MILP⁺ has $\approx 10\%$ smaller arrival time (Fig. 9(a,c)) because it is configured with a larger maximum speed. The arrival times of MICP are mostly around 20 (T_{max}) because MICP only optimizes the path length. Fig. 9(b,d) show that, the path length of WAP is within $\pm 10\%$ to MICP, indicating that WAP also achieves near-optimal path length.

6 Conclusion and Future Work

This work considers Motion Planning Among Dynamic Obstacles (MPDO) in continuous space and aims to minimize the arrival time. We propose an arithmetic-based wavefront propagation, WAP, to solve the problem in discretized time steps. Experiment shows that, our method can be 2.93 to 4.37 times faster than baselines and achieves a near-optimal path length.

Limitations and Future Work There are some limitations in our current method and can be addressed in future work. First, WAP may never terminate if the instance is unsolvable. This issue can be resolved by conducting connectivity analysis in each iteration. Second, Alg. 2 may suffer from numerical errors, which impact the result of reachable set R_n . Future work includes developing methods to identify and repair numerical errors during planning.

References

1. Afonso, R.J., Maximo, M.R., Galvao, R.K.: Task allocation and trajectory planning for multiple agents in the presence of obstacle and connectivity constraints with mixed-integer linear programming. *International Journal of Robust and Nonlinear Control* **30**(14), 5464–5491 (2020)
2. Bentley, J.L.: K-d trees for semidynamic point sets. In: Seidel, R. (ed.) *Proceedings of the Sixth Annual Symposium on Computational Geometry*, Berkeley, CA, USA, June 6–8, 1990. pp. 187–197. ACM (1990)
3. Canny, J., Reif, J.: New lower bound techniques for robot motion planning problems. In: *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. pp. 49–60. IEEE (1987)
4. Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G.A., Burgard, W.: *Principles of robot motion: theory, algorithms, and implementations*. MIT press (2005)
5. Frazzoli, E., Dahleh, M.A., Feron, E.: Real-time motion planning for agile autonomous vehicles. *Journal of guidance, control, and dynamics* **25**(1), 116–129 (2002)
6. Fujimura, K.: Motion planning amid transient obstacles. *The International journal of robotics research* **13**(5), 395–407 (1994)
7. Fujimura, K., Samet, H.: Planning a time-minimal motion among moving obstacles. *Algorithmica* **10**(1), 41–63 (1993)
8. Hales, T.C.: The jordan curve theorem, formally and informally. *The American Mathematical Monthly* **114**(10), 882–894 (2007)
9. Halperin, D., Salzman, O., Sharir, M.: Algorithmic motion planning. In: *Handbook of Discrete and Computational Geometry*, pp. 1311–1342. Chapman and Hall/CRC (2017)
10. Hershberger, J., Suri, S.: An optimal algorithm for euclidean shortest paths in the plane. *SIAM Journal on Computing* **28**(6), 2215–2256 (1999)
11. Hsu, D., Kindel, R., Latombe, J.C., Rock, S.: Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research* **21**(3), 233–255 (2002)
12. Kant, K., Zucker, S.W.: Toward efficient trajectory planning: The path-velocity decomposition. *The international journal of robotics research* **5**(3), 72–89 (1986)
13. Latombe, J.C.: *Robot motion planning*, vol. 124. Springer Science & Business Media (2012)
14. LaValle, S.M.: *Planning algorithms*. Cambridge university press (2006)
15. Maheshwari, A., Nouri, A., Sack, J.R.: Shortest paths among transient obstacles. *Journal of Combinatorial Optimization* pp. 1–39 (2020)
16. Mitchell, J.S.: An optimal algorithm for shortest rectilinear paths among obstacles in the plane. In: *Abstracts of the First Canadian Conference on Computational Geometry*, vol. 22 (1989)
17. Mitchell, J.S.: Shortest paths among obstacles in the plane. In: *Proceedings of the ninth annual symposium on Computational geometry*. pp. 308–317 (1993)
18. o'Rourke, J.: *Computational geometry in C*. Cambridge university press (1998)
19. Phillips, M., Likhachev, M.: Sipp: Safe interval path planning for dynamic environments. In: *2011 IEEE International Conference on Robotics and Automation*. pp. 5628–5635. IEEE (2011)
20. Ren, Z., Philip, A.G., Zhao, S., Rathinam, S., Choset, H.: Cp-milp: Mixed integer linear programming for multi-agent motion planning with linear dynamics. *IEEE Robotics and Automation Letters* **10**(12), 12573–12579 (2025)

21. Ren, Z., Rathinam, S., Choset, H.: A lower bounding framework for motion planning amid dynamic obstacles in 2d. In: *Algorithmic Foundations of Robotics XV*. pp. 540–556. Springer International Publishing, Cham (Dec 2022)
22. Ren, Z., Rathinam, S., Likhachev, M., Choset, H.: Multi-objective safe-interval path planning with dynamic obstacles. *IEEE Robotics and Automation Letters* **7**(3), 8154–8161 (2022)
23. Richards, A., How, J.P.: Robust variable horizon model predictive control for vehicle maneuvering. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal* **16**(7), 333–351 (2006)
24. Steen, L.A., Seebach, J.A., Steen, L.A.: *Counterexamples in topology*, vol. 18. Springer (1978)
25. Zhao, S., Liu, Y., Choset, H., Ren, Z.: Mixed integer conic programming for multi-agent motion planning in continuous space. In: *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 10540–10547 (2025)
26. Zhao, S., Philip, A.G., Rathinam, S., Choset, H., Ren, Z.: Cb-gcs: Conflict-based search on the graph of convex sets for multi-agent motion planning. In: *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*. pp. 2208–2214 (2025)

A Revisit the Repairing Function

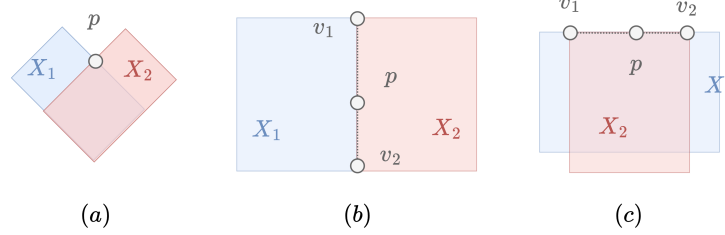


Fig. 10: Scenarios where a point is on the boundary of two regular sets X_1, X_2 when union them. The blue and red regions represent two regular sets, X_1 and X_2 , respectively, with point p located on the boundary of both sets. In scenario (a), point p is at the intersection of the two boundaries. In scenarios (b) and (c), the boundaries ∂X_1 and ∂X_2 overlap on the segment between points v_1 and v_2 , with point p situated on this segment.

Recall that when defining union operation between two regular sets $X = X_1 \cup X_2$ (Eq. (5)), a repairing function is introduced to handle the edge case for point p on the boundary of both sets ($p \in \partial X_1 \cap \partial X_2$). We now show that our algorithm never requires the use of the repairing function. Fig. 10 illustrates three scenarios. In Fig. 10(a), point p is at the intersection of the two boundaries, which will be a vertex in WG. Since we only evaluate the ISF value for each edge in WG in Alg. 2 and WG guarantees vertices never in the point set of edges, the repairing function is never be called. In Fig. 10(b) and (c), the pair v_1, v_2 forms a duplicated edge in WG, and will be handled in Appendix C. For the same reason, the repairing function is also never be called for the intersection operation (Eq. (9)).

B Proof of Lemma 2 in Sec. 4

Recall that $\mathbb{P}_n = \{R_{n-1}\} \cup \{S_w(\tilde{l}) | \tilde{l} \in \partial R_{n-1}\}$ denote all regular sets involved in propagation, and let $G^w : (E, V)$ denote the WG constructed from \mathbb{P}_n and \mathbb{O}_n (obstacle set at t_n). For a point \vec{p} on an edge of G^w , let

$$\begin{cases} \lambda_1(\vec{p}) &= \sum_{P \in \mathbb{P}_n} \mathcal{F}(\partial P)(\vec{p}), \end{cases} \quad (16)$$

$$\begin{cases} \lambda_2(\vec{p}) &= \sum_{O \in \mathbb{O}_n} \mathcal{F}(\partial O)(\vec{p}), \end{cases} \quad (17)$$

$$\begin{cases} \gamma(a, b) &= 2 - f(2 - f(a) + f(b)), \end{cases} \quad (18)$$

Lemma 2. $\mathcal{F}(\partial R_n)(\vec{p}) = \gamma(\lambda_1(\vec{p}), \lambda_2(\vec{p}))$ holds when \vec{p} is not on a duplicated edge of G^w .

Proof. Since p is not on a duplicated edge, the repairing function (Eq. (6)) is never be used for the union operator, this means that we can reformulate the saturation function $f(\cdot)$ (Eq. (5)) to union multiple sets:

$$f(a + f(b + c)) = f(a + b + c), \text{ when } a, b, c > 0.$$

Then, applying the union operator (Eq. (5)) on \mathbb{P}_n and \mathbb{O}_n , we have:

$$\begin{cases} \mathcal{F}(\partial U_n)(\vec{p}) &= f(\sum_{P \in \mathbb{P}_n} \mathcal{F}(\partial P)(\vec{p})) = f(\lambda_1(\vec{p})), \\ \mathcal{F}(\partial O_n)(\vec{p}) &= f(\sum_{O \in \mathbb{O}_n} \mathcal{F}(\partial O)(\vec{p})) = f(\lambda_2(\vec{p})). \end{cases} \quad (19)$$

With complement operator (Eq. (8)) on $\mathcal{F}(\partial O_n)$ and intersection operator (Eq. (9)) on $\mathcal{F}(\partial(U_n \cap \neg O_n))(\vec{p})$ we have

$$\mathcal{F}(\partial(U_n \cap \neg O_n))(\vec{p}) = \gamma(\lambda_1(\vec{p}), \lambda_2(\vec{p})). \quad \square$$

C Consider Duplicated Edges on Wavelet Graph (WG)

Recall that the WG, $G^w(\mathbb{P}, \mathbb{O})$, is constructed from regular sets \mathbb{P} and \mathbb{O} , where \mathbb{P} denote set of reachable regions and \mathbb{O} denote set of obstacles. For an edge $e \in G^w$, $A(e)$ is at set of wavelets that contain e , and e is a duplicated edge if $|A(e)| > 1$. Our problem is to compute $\mathcal{F}(\partial X)(p)$ for point $p \in e$ when e is a duplicated edge, where:

$$\begin{cases} X_P &= \bigcup_{P \in \mathbb{P}} P \\ X_O &= \bigcup_{O \in \mathbb{O}} O \\ X &= X_P \cap \neg X_O; \end{cases} \quad (20)$$

We start with a basic scenario that $|\mathbb{P}| + |\mathbb{O}| = 2$. Then, we demonstrate that when $|\mathbb{P}| + |\mathbb{O}| > 2$, we can treat e as a non-duplicated edge in certain cases and compute $\mathcal{F}(\partial X)(e)$ according to Eq. (18). In the remaining cases, we can reduce the problem to the basic scenario.

Basic Scenario Assume $|\mathbb{P}| + |\mathbb{O}| = 2$ and $A(e) = \{\tilde{l}_1, \tilde{l}_2\}$, where \tilde{l}_1 and \tilde{l}_2 are wavelets from regular sets X_1 and X_2 respectively. Fig. 11 illustrates all possible cases. In Fig. 11(a), all points $p \in e$ are interior points ($\mathcal{F}(\partial X)(p) = 2$), we referred to this case as *merge case*; in other cases, all points $p \in e$ are boundary points ($\mathcal{F}(\partial X)(p) = 1$).

General Scenario Let e denote a duplicated edge in $G_n = G^w(\mathbb{P}_n, \mathbb{O}_n)$, which constructed from \mathbb{P}_n and \mathbb{O}_n . Here, \mathbb{P}_n denote all regular sets involved in propagation at t_n , and \mathbb{O}_n denote all obstacles at t_n . We can start with $\mathbb{P} = \mathbb{O} = A(e) = \emptyset$, and iteratively add new regular set X^i from \mathbb{P}_n (or \mathbb{O}_n) to \mathbb{P} (or \mathbb{O}). The fact that $e \in G_n$ ensures that adding a new regular set X^i will

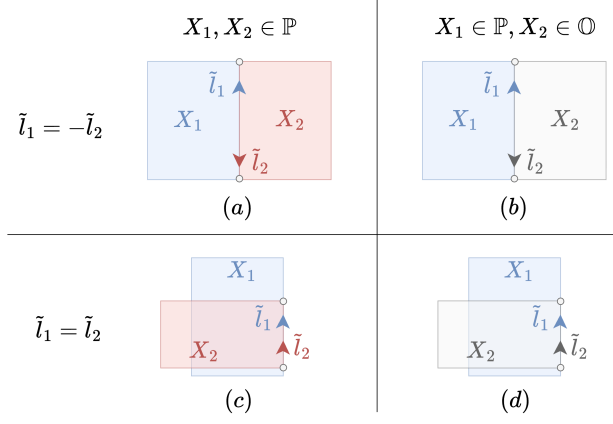


Fig. 11: All cases of a duplicated edge between two regular sets X_1 and X_2 in the basic scenario ($|\mathbb{P}| + |\mathbb{O}| = 2$). Blue and red regions represent regular sets from \mathbb{P} , grey region represent regular set from \mathbb{O} . Arrows indicate the direction of the associated wavelets to the edge.

never introduce new vertices $v \in e$. Let $A(e) = A_P(e) \cup A_O(e)$, where $A_P(e)$ and $A_O(e)$ denote wavelets from \mathbb{P} and \mathbb{O} respectively. In each iteration, we update $A(e)$ when e is on the boundary of X^i ($e \in \partial X^i$). Additionally, if we can confirm that $e \in \text{Int}(X_P)$ (or $e \in \text{Int}(X_O)$), then e can be regarded as an interior curve of X_P (or X_O), so that we can let $A_P(e) = \emptyset$ (or $A_O(e) = \emptyset$) and never update it in the remaining iterations. Determining $e \in \text{Int}(X_P)$ (or $e \in \text{Int}(X_O)$) can be done by checking (i) $e \in \text{Int}(X^i)$; or (ii) $X^i \in \mathbb{P}$ and falls into *merge case* (Fig 11(a)). After iterating all regular sets, there are following cases that we can treat e as a non-duplicated edge, so that we can compute $\mathcal{F}(\partial X)(e)$ according to Eq. (18):

- $A(e) = \emptyset$, meaning that e can be represented by interior curve of both X_P and X_O ;
- $A_P(e) = \emptyset \wedge A_O(e) \neq \emptyset$, meaning that e is on the boundary of X_O , and since overlapping edges between obstacles are not allowed (Sec. 3), we have $|A_O(e)| \leq 1$;
- $A_P(e) \neq \emptyset \wedge A_O(e) = \emptyset$, meaning that e is only on the boundary of X_P , then we have $\mathcal{F}(\partial X)(e) = \gamma(1, \lambda_2(e))$;

If $A_P(e) \neq \emptyset \wedge A_O(e) \neq \emptyset$, then, by letting $X_1 = X_P, X_2 = X_O$, we can confirm that $\mathcal{F}(\partial X)(e) = 1$ according to the basic scenario in Fig. 11(b) or (d),

D Numerical Robustness

Rounding errors in floating-point arithmetic can impact the result of our algorithm:

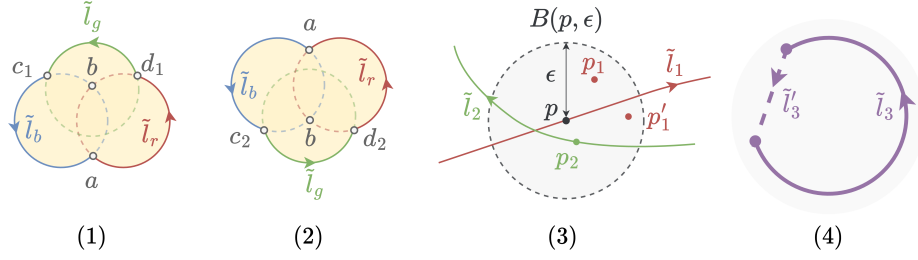


Fig. 12: (1) and (2): Solid curves represent wavefronts and yellow shaded areas represent reachable regions. In both scenarios, \tilde{l}_b and \tilde{l}_r intersect at points b, a , and \tilde{l}_g intersect with \tilde{l}_b, \tilde{l}_r at c, d respectively; (3) A point p might be treated as any point in the circle $B(p, \epsilon)$ due to numerical errors; (4) The dashed segment \tilde{l}_3 is a missing wavelet segment due to numerical errors.

- (i) when building the WG (Alg. 1 line 6). For example, in Fig. 12(1), intersection points on \tilde{l}_b in CCW is $a - b - c_1$, while in Fig. 12(2), it is $a - c_2 - b$. Different orders lead to different wavelet segments in WG, representing different reachable regions.
- (ii) when computing ISF value of a point, which involves intersection tests and dot products between rays and wavelet segments (Alg. 2 line 12 to 20). For example, in Fig. 12(3), p is supposed to be on the boundary of \tilde{l}_1 , but when computing with \tilde{l}_1 , p might be treated as located at p_1 ($p_1 \in R(\tilde{l}_1)$) or p'_1 ($p'_1 \notin R(\tilde{l}_1)$); while p might also be treated as located at p_2 when computing with \tilde{l}_2 .

The consequence of rounding errors might be accumulated along propagation and may significantly alter the topology of the reachable region. For example Fig. 12(4) demonstrates that when rounding errors cause missing of wavelet segment \tilde{l}_3 , the boundary of reachable region will become a non-closed curve. This causes all interior points of $R(\tilde{l}_3)$ to be incorrectly treated as exterior points. Mitigating such numerical issues is non-trivial. Our implementation simply uses an ϵ -distance test to determine point coincidence, and restart an iteration whenever detect unclosed boundaries that might due to rounding error. In each restarted iteration, we change the starting edge of *spread*. Experiment results in Appendix E show that this simple strategy is effective.

E Experiments on Synthetic Datasets

Dataset Description We generate the synthetic datasets in following ways:

1. Polygonal obstacles are randomly generated independently at each timestep.
2. Static obstacles combined with moving obstacles, specifically:
 - (a) Obstacle speed is similar to the agent's.

Scenario	Time (ms)	SR(%)	#W
1	78 ± 14	100%	97.5
2.a	69 ± 13	99.9%	77.3
2.b	72 ± 14	100%	85.7
2.c	83 ± 22	100%	88.5

Table 1: Performance summary in different scenarios. The column Time shows the median value with standard deviation; SR shows the success rate; and #W shows the average number of wavelets per iteration.

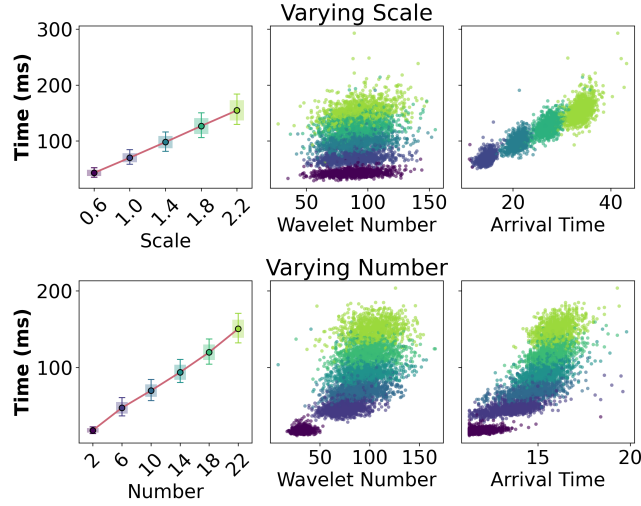


Fig. 13: Impact of workspace size (referred to as ‘Scale’) and number of wavelets (referred to as ‘Number’): The color gradient from dark to light indicates an increase in scale or the number.

- (b) Obstacle speed is significantly lower than the agent’s.
- (c) Obstacle speed is significantly higher than the agent’s.

The workspace is 10×10 . The start and goal positions are randomly sampled from the workspace, and the average euclidean distance between them is 12. There are 1024 instances for each scenario. The size of an obstacle is defined as the bounding box of the polygon. At each timestep, there are 10 obstacles of average size 2. Due to the numerical issues discussed in Appendix D, we inspect the result of all instances and ensures the path is valid. Invalid paths are treated as failures.

Table 1 summarizes the performance of WAP (with *spread*) in different scenarios. We can see that WAP can solve all instances in millisecond-level time, and rarely fails due to numerical errors.

Next, we take all 1,024 instances from the scenario 2.b, and vary the workspace size and obstacle density to analyze their impact on WAP. To inspect the impact of workspace size, we set a scale factor $f \in \{0.6, 1.0, \dots, 2.2\}$. For each f , we map a point (x, y) in the original workspace to $(f \cdot x, f \cdot y)$ in the new workspace ($f = 1.0$ means no change). To inspect the impact of obstacle density, we set the average number of obstacles per timestep $n_o \in \{2, 6, \dots, 22\}$ ($n_o = 2$ is the default setting). For both experiments, we record the runtime, maximum number of wavelets per iteration (referred to as ‘wavelet number’), and arrival time.

Impact of Workspace Size. Fig. 13(a) shows that both the runtime and arrival time increases linearly with the workspace size, while the wavelet number remains stable across different scales. This is because scaling the workspace size has minor impact on the topology of the reachable region, resulting in the wavelet number remains stable, while the number of iterations increases linearly with the scale.

Impact of Obstacle Density. Fig. 13(b) shows that the runtime increases linearly with the obstacle number. However, the wavelet number and arrival time initially increases with the obstacle number but then saturates when the obstacle number becomes sufficiently large. This is because runtime is impacted by two factors: the number of iterations (i.e., arrival time) and the computational cost per iteration, with only the latter increasing linearly with the obstacle number. Specifically, consider when obstacle number increases from 2 to 6. The wavelet number significantly increases as more obstacles appear in the reachable region, and more wavelets be generated per iteration. However, these obstacles which do not block the optimal path, can not impact the minimum arrival time. When the obstacle number increases further, the reachable region per iteration becomes smaller due to more obstacles block the way, and wavelets won’t be generated for obstacles that appear in non-reachable regions. Therefore, the wavelet number trends to saturate. This observation suggests that we can further improve the runtime by reducing the number obstacles to process in each iteration, for example, pruning these obstacle which cannot generate new wavelets.