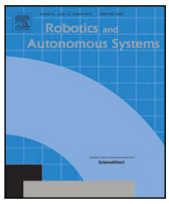
Robotics and Autonomous Systems 193 (2025) 105109

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot



Coverage Trajectory Planning Problem on 3D Terrains with safety constraints for automated lawn mower: Exact and heuristic approaches

Hang Zhou, Peng Zhang *, Zhaohui Liang, Hangyu Li, Xiaopeng Li *

Department of Civil and Environmental Engineering, University of Wisconsin-Madison, United States of America

ARTICLE INFO

Keywords:
Coverage Path Planning
Trajectory Planning
Automated lawn mower

ABSTRACT

Recent technological advancements in automation have attracted increased interest in automated lawn mowers. Developing a safe and efficient trajectory to cover entire terrains is crucial for autonomous mowing. Unlike the simplified indoor environments with flat surfaces commonly assumed in existing literature, lawn maintenance typically occurs in complex outdoor settings characterized by irregular terrains, including obstacles and slopes. These irregularities pose significant safety risks, such as the potential for the mower to tip over. This paper introduces the Coverage Trajectory Planning Problem on 3D Terrains (CTPP-3DT), which involves determining both the path and speed profile for an automated lawn mower to effectively cover a general 3D terrain with varying slopes. The objective of the CTPP-3DT is to minimize the completion time, including the time for turning, which satisfies safety constraints on the robot's speed and acceleration on various slopes. To address this challenge, we first propose a Mixed-Integer Linear Programming (MILP) model based on a graph expansion method, suitable for solving small-scale instances. For larger instances, we develop a decomposition-based heuristic algorithm using Simulated Annealing. Extensive experiments conducted on benchmark instances demonstrate the effectiveness of our proposed MILP model and heuristic algorithm for small-size and large-size instances. The comparison between the optimized strategy and the conservative strategy highlights the necessity of incorporating safety constraints in trajectory planning, resulting in an average reduction of more than 40% in completion time. Furthermore, sensitivity analyses reveal that technological advancements in mowers, such as increasing the maximum speed and acceleration and reducing turning speed, can significantly reduce the overall completion time. Our code and data are available at https://github.com/CATS-Lab/Mower-CTPP-3D.

1. Introduction

The rapid development of automation has presented numerous industries with opportunities to enhance efficiency, with automated mowing emerging as a prominent application. Traditional manual lawn maintenance has several drawbacks. Firstly, maintaining large-size lawns is labor-intensive and time-consuming, which brings substantial human costs. Secondly, human operation errors often result in irregular lawn maintenance and inefficient mowing. Lastly, lawn mowers pose certain safety risks associated with adverse weather conditions, tipping over on steep slopes, and hazards caused by the mower blades. Automated lawn mowers offer solutions to these issues. By leveraging trajectory planning and control algorithms from autonomous driving technology, automated mowers can precisely execute optimal mowing paths without human involvement, significantly reducing labor costs, improving mowing efficiency, and eliminating safety hazards. Recent advancements in autonomous driving perception technology, particularly in low-cost sensors such as RGB-D cameras, have further driven the applications and large-size production of automated lawn mowers [1,2]. This highlights the need for developing algorithms for the operational scenarios of automated lawn mowers.

In automated mowing, the path to cover the entire terrain is crucial for the efficiency of the mowing task [3,4]. In the literature, the problem of determining such a coverage path is referred to as the Coverage Path Planning (CPP) problem, which is known to be NP-hard, implying that no polynomial-time algorithm is currently known for solving it in general cases [5]. Existing CPP algorithms are effective in indoor environments on flat surfaces [6]. However, lawn mowing often involves outdoor environments with irregular terrain, including obstacles, slopes, and dips, which bring significant safety risks. Research shows that only 47% of cropland in the United States has less than 2% slopes; 48% of the cropland is on slopes between 2% and 10% [7]. Therefore, considering three-dimensional (3D) terrain in CPP is crucial for enhancing mowing operations.

* Corresponding authors.
E-mail addresses: pzhang257@wisc.edu (P. Zhang), xli2485@wisc.edu (X. Li).

https://doi.org/10.1016/j.robot.2025.105109

Available online 25 June 2025

Although some literature has considered 3D terrain in CPP [7,8], significant research gaps remain. One critical issue is that these studies focus only on path planning, without considering the robot's speed during movement. As a result, they overlook safety constraints caused by varying slopes. However, when safety constraints are taken into account—especially on irregular 3D terrains—it becomes essential to jointly optimize both the path and the speed profile in a unified problem. This process is referred to as trajectory planning. Traditional CPP methods [7,8] typically assume constant speed or ignore dynamic feasibility. On the other hand, existing speed planning methods such as MPC-based approaches [9–11] are often based on a given path. These methods are insufficient to directly address the integrated problem posed by coverage planning over complex, uneven terrains. Therefore, new algorithm designs are required to incorporate terrain-induced safety constraints while optimizing coverage trajectories in such environments.

To address the above-mentioned problems, this paper proposes the Coverage Trajectory Planning Problem on 3D Terrains (CTPP-3DT) for automated lawn mowers and other agricultural machines, aiming to solve the safety challenges caused by uneven terrain. In this problem, we introduce height onto traditional two-dimensional (2D) maps to calculate the robot's slope along its trajectory. Safety constraints are defined as the maximum allowable slopes for robot movement and the speed and acceleration limits under varying slopes. To tackle this problem, the paper presents a solution framework employing two methods. The exact algorithm determines optimal solutions for small-size scenarios using a mixed integer linear programming (MILP) model. This model incorporates the turning cost with a graph expansion method. The heuristic algorithm divides the area into cells via a tailored decomposition algorithm. Then, it generates the candidate coverage trajectories in each cell and the shortest trajectories to connect them. Finally, the optimal trajectories are selected from the candidate trajectories set to form a complete trajectory. Our problem and solutions are not only suitable for lawn mowers but can also be applied to other robots, such as agricultural machines. In general, the main contributions can be summarized as follows:

- We introduce a CTPP-3DT for automated lawn mowers and other agricultural machines. This problem aims to determine the robot's coverage trajectory in the 3D terrain that minimizes the completion time while satisfying the safety constraints.
- We design an exact MILP model for the CTPP-3DT. A graph expansion method is applied before solving the MILP model to represent turning using dummy arcs.
- We design a novel decomposition-based heuristic algorithm to solve the CTPP-3DT. Several components in this algorithm, such as the decomposition algorithm, shortest path algorithm, dynamic programming, and local search, are tailed for the 3D problem.

The remainder of this paper is organized as follows. Section 2 reviews the related literature to CPP. Section 3 defines the 3D-CTTP. Section 4 describes the exact method of the 3D-CTTP. Section 5 presents the details of our proposed heuristic algorithm. Simulation results and discussions are shown in Section 6. Section 7 concludes this paper and provides future work.

2. Related works

There are various classifications of CPP algorithms in the literature [12–14]. Depending on whether the boundary of the area to be traversed is mapped out in advance, CPP algorithms can be categorized as offline or online. The latter approach, also known as sensor-based coverage, relies on real-time sensor data from the robot to gather information about the environment [5]. Furthermore, based on the dimension of the space to be covered, CPP approaches may be classified as 2D, when the robot's motion is constrained to a plane, or 3D, with 3D coverage algorithms finding relevance in operations with unmanned aerial vehicles (UAVs) [15–17] or autonomous underwater vehicles (AUVs) [18].

From the perspective of solution methods, existing approaches for 2D and offline CPP can be broadly classified into two categories: exact and heuristic methods. Exact methods aim to find optimal solutions but often require significant computational time. Representative exact approaches include MILP [19], spanning tree coverage [20,21], and dynamic programming [22]. These methods are typically suitable for indoor scenarios where the area to be covered is relatively small. In contrast, heuristic methods aim to find feasible, though not necessarily optimal, solutions in a more computationally efficient way. One common strategy among heuristic approaches in CPP is to decompose the entire map into subregions, plan local paths within each subregion, and then connect them to form a complete coverage path. Notable decomposition-based algorithms include trapezoid decomposition [23], boustrophedon decomposition [24,25], Morse-based cellular decomposition [26], and exact cell decomposition [27].

In real-world applications, CPP must consider many practical constraints, such as 3D terrain, turning time, and speed control. Some literature addresses these constraints separately. For instance, [7] developed a 3D coverage planning approach with energy consumption models that account for terrain inclinations to minimize energy requirements using a genetic algorithm. [8] tackled energy-efficient coverage path planning on general 3D surfaces by generating geodesic Fermat spiral paths. [28] proposed two greedy algorithms for CPP in agricultural fields, where the cost function minimizes the relative efficiency, defined as the operated area divided by total time (including turns). [29] incorporated speed control into their CPP. However, their speed profile optimization is applied after obtaining the path plan, separating path optimization from speed optimization.

Overall, while the above literature addresses some problem settings or constraints in the CTPP-3DT, none of them comprehensively considers all the settings in an entire problem. Most literature on 3D terrain focuses on optimizing energy consumption for different slopes without considering the impact of slopes on robot speed, which may cause safety risks. Additionally, existing literature primarily focuses on path planning and lacks joint control of speed and path, which is essential for trajectory planning. To bridge this research gap, this paper proposes the CTPP-3DT that considers safety constraints under 3D terrain and simultaneously optimizes the trajectory's path and speed profile to minimize completion time, including the turning time. To systematically solve and investigate this problem, the MILP model and a heuristic algorithm are proposed to solve this problem, followed by experiments for the algorithms and the sensitivity analyses.

3. Problem definition

The CTPP-3DT considers a 3D space $\mathbb{R}^3$, where the X and Y axes represent the horizontal plane and the Z axis denotes the elevation. Within this space, the coverage region is defined as a closed surface $\mathcal{G}_{\text{ini}}$ embedded in $\mathbb{R}^3$ ($\mathcal{G}_{\text{ini}} \subset \mathbb{R}^3$). Denote the position of each point $p \in \mathcal{G}_{\text{ini}}$ as (p_x, p_y, p_z), this means that the vertical position (p_z) can be represented as a function f of horizontal coordinates, i.e., $p_z = f(p_x, p_y)$. Since lawn mowing is usually performed periodically in fixed locations, we assume that $\mathcal{G}_{\text{ini}}$ is a static and known environment. The area $\mathcal{G}_{\text{ini}}$ contains some service areas that need to be covered, denoted as $\mathcal{S}$, and restricted areas that cannot be traversed, denoted as $\mathcal{R}$. These areas are closed and do not touch each other, which means that $\mathcal{S} \cap \mathcal{R} = \emptyset$. By ignoring the elevation of the area $\mathcal{G}_{\text{ini}}$, we discretize it into a 2D grid consisting of square cells with side length w. The value of w can be set to the width of the robot, which ensures that each cell can be fully covered when the robot passes through it from any of the four cardinal directions. A robot starts from a start point $p_{\text{start}} \in \mathcal{S}$ at time 0 to cover the entire area and eventually returns to the end point $p_{\text{end}} \in \mathcal{S}$. We define a path as a sequence of points. A trajectory is defined as a path

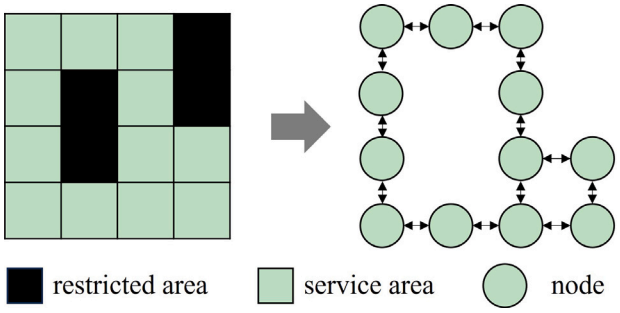


Fig. 1. Transition from the input map $\mathcal{G}_{\text{ini}}$ to the directed graph $\mathcal{G}_{\text{dir}}$.

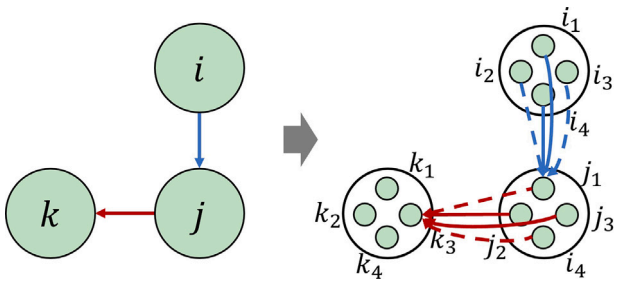


Fig. 2. Illustration of the graph expansion from $\mathcal{G}_{\text{dir}}$ to $\mathcal{G}_{\text{exp}}$. The solid lines represent the non-turning arcs, and the dashed lines represent the turning arcs.

along with the speed and acceleration on the path. The speed of the robot moving from p_i to p_j is denoted as v_{ij}, and the acceleration is denoted as a_{ij}. The slope from point p_i to p_j is denoted as $\theta_{ij} = \frac{z_j - z_i}{w}$. The safety ranges of speed and acceleration are determined by the slope θ_{ij}, denoted as $[0, v_{ij}^+]$ and $[a_{ij}^-, a_{ij}^+]$. Similar to Charitidou and Keviczky [19], this study assumes that the robot's speed and turning angle are discretized into finite sets $\mathcal{U} = \{0, \Delta u, 2\Delta u, \ldots, N\Delta u\}$ and $\mathcal{K} = \{0, \pi/2, \pi, 3\pi/2\}$, respectively. Here, $\mathcal{U}$ denotes the set of feasible speed values, $\mathcal{K}$ represents the set of discrete heading angles, and Δu is the speed increment. The value of N is chosen such that the maximum feasible speed satisfies $N\Delta u \le \max_{p_i,p_j \in S} v_{ij}^+ < (N+1)\Delta u$. We assume the robot is a zero-turn robot, which must decelerate to 0 speed before turning. We assume the time for the turn is a constant t_{turn}. The objective of the CTPP-3DT is to determine a trajectory that minimizes the completion time while satisfying the above safety constraints.

4. Exact algorithm

This section presents an exact algorithm for solving the CTPP-3DT. The method formulates the problem as a MILP model, which can be optimally solved using state-of-the-art solvers. Before describing the model, we construct a directed graph using a graph expansion method in Section 4.1 to model the robot's turns. In Section 4.2, we describe the decision variables, objective function, and constraints of the model.

4.1. Graph construction

The first step of the exact algorithm is to build the directed graph based on the input map. We construct a directed graph $\mathcal{G}_{\text{dir}} = (\mathcal{V}_{\text{dir}}, \mathcal{E}_{\text{dir}})$ for the grid map $\mathcal{G}_{\text{ini}}$, where $\mathcal{V}_{\text{dir}}$ represents the set of nodes and $\mathcal{E}_{\text{dir}}$ represents the set of arcs. Each point $p \in S$ is transferred to a node $i \in \mathcal{V}_{\text{dir}}$. Nodes i and $j \in \mathcal{V}_{\text{dir}}$ are connected with arc $(i,j) \in \mathcal{E}_{\text{dir}}$ if their corresponding points p_i and p_j are adjacent. In particular, we denote node i_{start} and i_{end} as the nodes corresponding to the start and end point p_{start} and $p_{\text{end}} \in \mathcal{G}_{\text{ini}}$, respectively. Fig. 1 shows an example of the directed graph.

To incorporate the turning time, a graph expansion method is applied to $\mathcal{G}_{\text{dir}}$. The main idea of this method is to create dummy arcs with penalty costs for turns. As shown in Fig. 2, for node $i \in \mathcal{G}_{\text{dir}}$, a dummy node set $D_i = \{i_1, i_2, i_3, i_4\}$ is generated. These nodes receive arcs entering from different directions. For example, the dummy node j_1 only receives arcs entering from dummy nodes in D_i, while dummy node k_3 only receives arcs entering from dummy nodes in D_j. Thus, we can classify arcs into turning and non-turning arcs, which are represented as dashed lines and solid lines respectively in Fig. 2. Any trajectory containing turns must have turning arcs. For instance, the trajectory $(i_4, j_1), (j_1, k_3)$ includes the dashed arc (j_1, k_3), indicating a turn at node j. Denote the expanded directed graph as $\mathcal{G}_{\text{exp}} = (\mathcal{V}_{\text{exp}}, \mathcal{E}_{\text{exp}})$, where $\mathcal{V}_{\text{exp}} = \cup_{i \in \mathcal{V}_{\text{dir}}} D_i$ and $\mathcal{E}_{\text{exp}} = \{(i', j') | i' \in D_i, j' \in D_j, (i,j) \in \mathcal{E}_{\text{dir}}\}$. In particular, we denote sets D_{start} and D_{end} as the dummy node sets for i_{start} and i_{end}, respectively. We also define the set of turning arcs as $\mathcal{E}_{\text{turn}}$. A complete trajectory must include at least one dummy node from each dummy node set in $\mathcal{G}_{\text{exp}}$.

Then we introduce two sets of parameters for the speed and acceleration decisions. Denote the time the robot takes to move from node $i \in \mathcal{V}_{\text{exp}}$ at speed $n \in \mathcal{U}$ to reach speed $m \in \mathcal{U}$ at node $j \in \mathcal{V}_{\text{exp}}$ as $t_{ijnm} = 2w/(n+m)$. The turning time t_{turn} is added to t_{ijnm} if arc (i,j) is a turning arc. We also introduce a set of binary parameters v_{ijnm} to indicate whether this movement violates safety constraints. If $m \in [v_{ij}^-, v_{ij}^+]$ or $(m^2 - n^2)/(2w) \in [a_{ij}^-, a_{ij}^+]$, $v_{ijnm} = 1$, otherwise $v_{ijnm} = 0$.

If the robot makes two consecutive turns at nodes i and j, the speed at both nodes is set to 0. To avoid infinite travel time, for this special case, we assume the robot accelerates with the maximum acceleration and then decelerates with the maximum deceleration along arc (i,j). The travel time for such an arc $(i,j) \in \mathcal{V}_{\text{exp}}$ is computed as:

$$t_{ij00} = \left(\frac{1}{a_{ij}^+} - \frac{1}{a_{ij}^-}\right)\sqrt{\frac{2wa_{ij}^+ a_{ij}^-}{a_{ij}^+ - a_{ij}^-}} \tag{1}$$

Note that this calculation does not include the turning time at nodes i or j.

4.2. A MILP model

The second step is to build the MILP model. The model contains the following variables:

- $x_{ij} \in \{0,1\}$: binary decision variable. $x_{ij} = 1$ if the robot traverses arc $(i,j) \in \mathcal{E}_{\text{exp}}$; 0 otherwise.
- $y_{in} \in \{0,1\}$: binary decision variable. $y_{in} = 1$ if the robot passes through node $i \in \mathcal{V}_{\text{exp}}$ with speed level $n \in \mathcal{U}$; 0 otherwise.
- $\tau_{ij} \in \mathbb{R}_{\ge 0}$: continuous decision variable. Represents the travel time on arc $(i,j) \in \mathcal{E}_{\text{exp}}$ if it is traversed by the robot; otherwise, it is set to zero.
- $u_i \in \mathbb{Z}_{\ge 0}$: integer decision variable. Represents the sequence number of node $i \in \mathcal{V}_{\text{exp}}$ in the tour. This variable is used to eliminate sub-tours in the routing constraints.

The model is constructed below:

$$\min \quad \sum_{(i,j)\in\mathcal{E}_{\text{exp}}} \tau_{ij} \tag{2}$$

$$\text{s.t.} \quad \sum_{(i,j)\in\mathcal{E}_{\text{exp}}} x_{ij} = \sum_{(j,i)\in\mathcal{E}_{\text{exp}}} x_{ji}, \quad \forall i \in \mathcal{V}_{\text{exp}}/(D_{\text{start}} \cup D_{\text{end}}), \tag{3}$$

$$\sum_{i' \in D_i} \sum_{j \in \mathcal{V}_{\text{exp}}} x_{i'j} \ge 1, \quad \forall i \in \mathcal{V}_{\text{dir}}/\{i_{\text{start}}, i_{\text{end}}\}, \tag{4}$$

$$\sum_{i \in D_{\text{start}}} \sum_{j \in \mathcal{V}_{\text{exp}}} x_{ij} = \sum_{i \in D_{\text{end}}} \sum_{j \in \mathcal{V}_{\text{exp}}} x_{ji} = 1, \tag{5}$$

$$u_i - u_j + 1 \le M\left(1 - x_{ij}\right), \quad \forall (i,j) \in \mathcal{E}_{\text{exp}}, \tag{6}$$

$$\sum_{n \in \mathcal{U}} y_{in} \le 1, \quad \forall i \in \mathcal{V}_{\text{exp}}, \tag{7}$$

$$\sum_{i \in D_{\text{start}}} y_{i0} = \sum_{i \in D_{\text{end}}} y_{i0} = 1, \tag{8}$$

$$\sum_{n\in\mathcal{U}} y_{in} \geq 1 - M\left(1 - \sum_{(i,j)\in\mathcal{E}_{\text{exp}}} x_{ij}\right), \quad \forall i \in \mathcal{V}_{\text{exp}}, \tag{9}$$

$$\sum_{n\in\mathcal{U}} y_{in} \geq 1 - M\left(1 - \sum_{(j,i)\in\mathcal{E}_{\text{exp}}} x_{ji}\right), \quad \forall i \in \mathcal{D}_{\text{end}}, \tag{10}$$

$$y_{i0} \geq 1 - M\left(1 - \sum_{(i,j)\in\mathcal{E}_{\text{turn}}} x_{ij}\right), \quad \forall i \in \mathcal{V}_{\text{exp}}, \tag{11}$$

$$y_{jm} \leq v_{ijnm} + M\left(2 - x_{ij} - y_{in}\right), \quad \forall (i,j) \in \mathcal{E}_{\text{exp}}, n \in \mathcal{U}, m \in \mathcal{U}, \tag{12}$$

$$\tau_{ij} \geq t_{ijnm} - M\left(3 - x_{ij} - y_{in} - y_{jm}\right), \quad \forall (i,j) \in \mathcal{E}_{\text{exp}}, n \in \mathcal{U}, m \in \mathcal{U}, \tag{13}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in \mathcal{E}_{\text{exp}}, \tag{14}$$

$$y_{in} \in \{0,1\}, \quad \forall i \in \mathcal{V}_{\text{exp}}, n \in \mathcal{U}, \tag{15}$$

$$\tau_{ij} \geq 0, \quad \forall (i,j) \in \mathcal{E}_{\text{exp}}, \tag{16}$$

$$u_i \geq 0, \quad \forall i \in \mathcal{V}_{\text{exp}} \tag{17}$$

The objective function (2) minimizes the total completion time. Constraints (3)–(5) are the arc-flow constraints. In these constraints, each dummy node can be visited at most once, but each original node in $\mathcal{V}_{\text{dir}}$ may be visited multiple times through different dummy nodes. This allows for potential backtracking and ensures that a feasible solution always exists. Constraints (6) eliminate sub-tours, where M is a large constant. Constraints (7)–(11) are the speed control constraints. Specifically, constraints (7) ensure that each node has only one speed when traversed by the robot. Constraints (8) specify that the speed at the start and end nodes is zero. Constraints (9)–(10) ensure that a dummy node i must have a speed assigned if it is traversed by the robot, where constraint (10) is specially designed for the case where it is the last node in the route. Constraints (11) enforce the speed to be zero when the robot needs to turn. Constraints (12) are the safety constraints for speed. Constraints (13) calculate the travel time for arc (i, j). Note that the parameters t_{ijnm} and v_{ijnm} used in these constraints are computed in advance based on the methods described in Section 4.1. Constraints (14)–(17) define the domain of the decision variables.

5. Heuristic algorithm

While the MILP model can solve the problem to optimal, it requires long computational time for large-size instances in real-world applications. This section introduces an efficient heuristic algorithm for the CTPP-3DT, which aims to solve the same optimization problem as formulated in Section 4. Instead of transferring each point as a node in the directed graph, the heuristic algorithm decomposes the map into cells and transfers each cell as a node, which can significantly reduce the scale of the directed map. The algorithm has three steps: map decomposition, trajectory generation, and trajectory connection. The following sections describe the details of these three steps.

5.1. Map decomposition

The first step of the algorithm is map decomposition. This step aims to decompose the map into a small number of cells to reduce the computational complexity of the next two steps. The decomposition can be solved by algorithms proposed in the literature, such as the Boustrophedon Cellular Decomposition (BCD) [24,25]. However, the BCD does not consider the safety constraints in the 3D terrain. Therefore, we adapt the BCD to a Rotated BCD (RBCD) algorithm. We first briefly review the BCD and then introduce details of the RBCD.

The BCD approach uses a slice to sweep the area in a certain direction. As shown in Fig. 3, if it encounters an obstacle with a convex boundary, the existing cell will be divided into several new cells. If it leaves an obstacle, multiple old cells will merge into a new cell. The cells generated by the algorithm can all be completely covered using a Boustrophedon path (i.e., a zig-zag path) in the direction perpendicular to the sweep direction. However, BCD cannot handle obstacles represented by non-convex polygons, as it relies on the assumption that obstacles are convex. Since the RBCD algorithm builds upon BCD, it also inherits this limitation. A common solution in the literature is to approximate non-convex obstacles using their minimum bounding convex hulls, which can be computed using standard convex hull algorithms such as the Quickhull algorithm [30].

In the CTPP-3DT, due to the influence of slopes, the completion time for a robot to cover an entire area along different path directions can vary significantly. In some directions, the zig-zag paths may be infeasible if there is a steep slope along the paths. Therefore, we modified the BCD to RBCD, which attempts to cover the map using zig-zag paths in the optimal directions. For each direction $\alpha \in \mathcal{A}$, we rotate the map by direction α. Then, replace the steep path in the direction that is perpendicular to α with dummy obstacles and apply the BCD to the map. Dummy obstacles are marked as special cells and handled separately in the following steps. Finally, evaluate the BCD results in each direction and select the optimal one α^* to apply the BCD.

There are two key aspects to applying the RBCD. 1. Implementing the BCD in different directions: Since rotating the map $\mathcal{G}_{\text{ini}}$ yields the same result as rotating the direction of the zig-zag path, we fix the sweeping direction of the slice in the BCD to the horizontal right and then rotate the map. We first construct a pixel matrix based on the input grid map. Then, we rotate the image of the pixel matrix to obtain the rotated pixel matrix, which serves as the rotated map. 2. Selecting the optimal direction α^*: We evaluate the BCD results by minimizing the number of segmented cells and the number of turns in the zig-zag path. Fewer cells indicate a better decomposition. If the number of cells is the same, fewer turns in the zig-zag path are preferable. Fig. 4 provides an example of the RBCD process with sweep directions $\alpha = 0$ and $\alpha = \frac{1}{2}\pi$. The top part of the figure illustrates the 3D terrain of the target area, which includes two rectangular obstacles. In addition, a steep slope is present at the location marked by the dashed line, making it impossible for the robot to proceed in the direction indicated by the arrow. As a result, when BCD is applied in the direction shown in Fig. 4(a) (i.e., $\alpha = 0$), a dummy obstacle is generated to account for the impassable terrain. The bottom part of the figure presents the resulting cell decomposition in the top view, where brown rectangles represent dummy obstacles and black rectangles represent physical obstacles. Since the decomposition with $\alpha = \frac{1}{2}\pi$ produces fewer cells, it outperforms the case with $\alpha = 0$. The pseudocode in Algorithm 1 illustrates the algorithm's process.

Algorithm 1 Map decomposition algorithm.

```
Input: map G_ini
Initialize α* ← 0, #Cell* ← M, #turn* ← M
for α ∈ A do
    G_α ← rotate G_ini for α degree
    G'_α ← add dummy obstacles for slopes larger than the safety slope
    #Cell, #turn ← BCD(G'_α)
    if (#Cell < #Cell*) or (#Cell == #Cell* and #turn < #turn*) then
        α* ← α, #Cell* ← #Cell, #turn* ← #turn
    end if
end for
C ← BCD(G'_α*)
Output: a set of cells C
```

5.2. Trajectory generation

The second step trajectory generation creates trajectories that cover cells in C and connect them. The generation is divided into three steps:

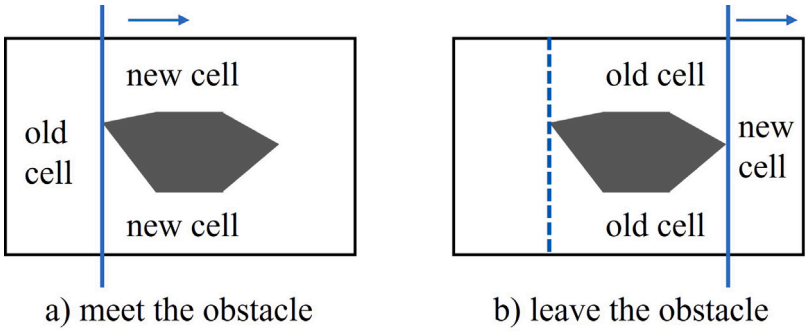


Fig. 3. Illustrations of the key process of the BCD algorithm when meeting and leaving an obstacle.

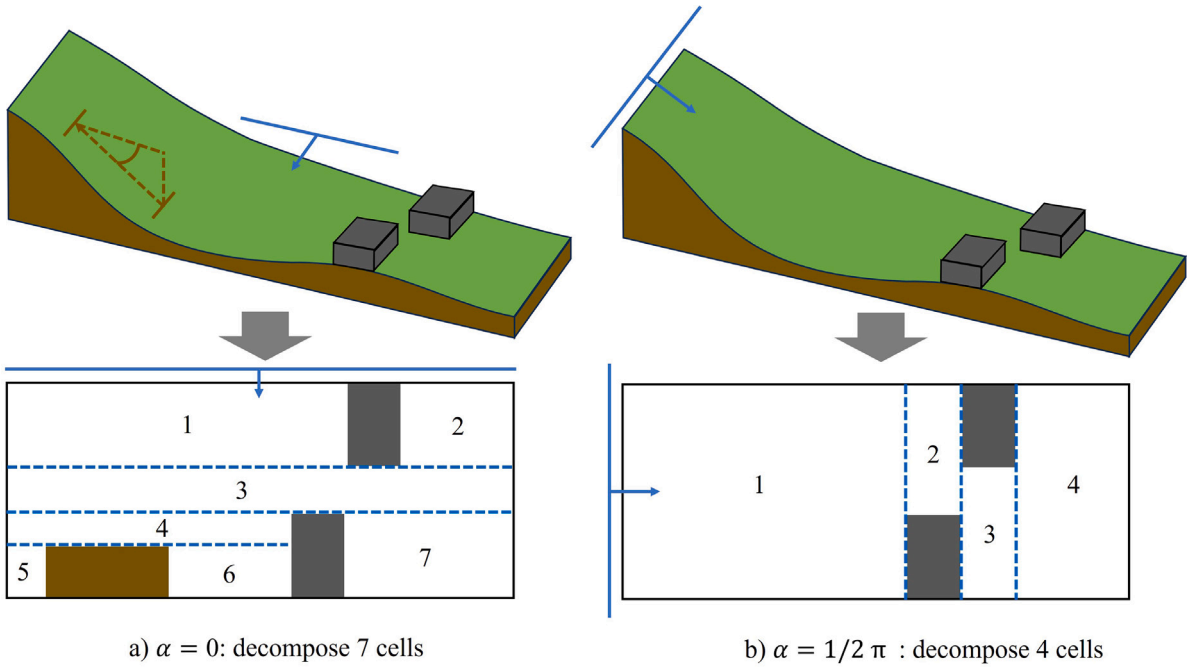


Fig. 4. Examples of the RBCD algorithm in two sweep directions.

(1) generating the path within cells, (2) generating the path to connect the cells, and (3) dynamic programming for speed control.

5.2.1. Generate paths within cells

According to Section 5.1, the cells decomposed by the BCD can be covered using zig-zag paths. Different from traditional CPP, due to the consideration of slope effects on speed, in CTPP-3DT, the completion time of zig-zag paths starting from different points within the same cell may vary. Therefore, we generate four zig-zag paths starting from the four vertices of the cell: the top-left, bottom-left, top-right, and bottom-right corners, as the candidate paths. Fig. 5 shows an example of zig-zag paths generated in one cell, where the yellow grid represents the start point and the green grid represents the end point. Specifically, for the cells generated by dummy obstacles that cannot be covered using the zig-zag path, we divide these cells into rows and generate only horizontal paths, as shown in Fig. 6. All the paths obtained in the step are denoted as $\mathcal{Z}_{\text{in}}$.

5.2.2. Generate paths to connect cells

Denote the start point and end point for path $z \in \mathcal{Z}_{\text{in}}$ as $p_{\text{start}}(z)$ and $p_{\text{end}}(z)$. To form a complete path, we need to find the shortest path connecting $p_{\text{end}}(z_1)$ and $p_{\text{start}}(z_2)$, where $z_1 \neq z_2$. We use a modified A* algorithm to search for the path between these two points.

The A* algorithm is a widely-used heuristic algorithm for the shortest path problem [31,32]. The traditional A* uses a priority queue to explore points with the minimum estimated cost. In our problem, the objective is to minimize the completion time including the turning time. To record turns, we need to check if at least three consecutive points in the path are in a straight line. Therefore, we extend the A* algorithm's priority queue to include three nodes to record turns. Since we have not yet obtained the complete path and cannot calculate the optimal speed, we use the robot's lowest speed to calculate travel time when computing the shortest path.

In numerical experiments, we found that the robot's optimal path usually does not pass through two distant cells consecutively. Therefore, it is unnecessary to connect the start and end points of all paths z when calculating the shortest path. To this end, we set a maximum span $\bar{\eta}$ for the cells when computing the shortest path. First, we build an undirected graph for the decomposed cells, where all arcs have a cost of 1. Then, we use the Floyd algorithm [33] to calculate the shortest distance between cells. If the distance between two cells c_1 and c_2 exceeds $\bar{\eta}$, we do not compute the shortest path between them.

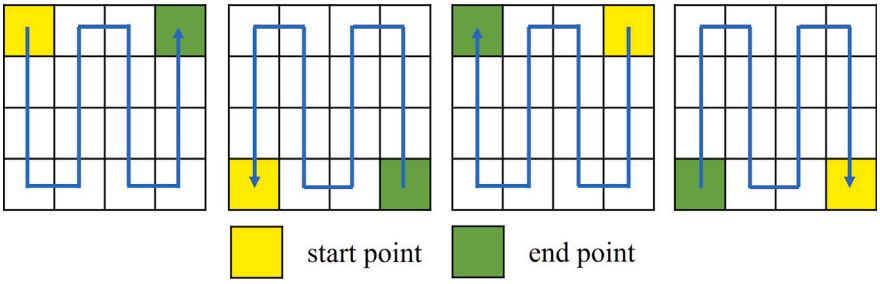


Fig. 5. Example of the four candidate zig-zag paths generated in one cell.

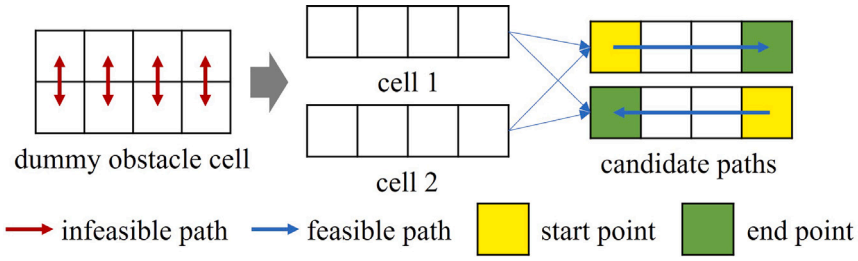


Fig. 6. Illustration of the candidate paths for dummy obstacle cells.

The parameter $\bar{\eta}$ is given as a hyperparameter before the algorithm runs. Notice that its value may need to be tuned to ensure feasibility, depending on the problem instance. All the shortest paths obtained in this step are stored in set $\mathcal{Z}_{\text{out}}$.

5.2.3. Dynamic programming for speed control

For all paths in $\mathcal{Z}_{\text{in}}$ and $\mathcal{Z}_{\text{out}}$, we use a dynamic programming algorithm to determine the optimal speed. Since the robot must slow down to zero speed at each turn, we divide each path z into several straight-line segments $\mathcal{L}(z)$ at the turning points, referred to as segments. A segment $l \in \mathcal{L}(z)$ can be represented as a sequence of points $\{p_l^0, p_l^1, \ldots, p_l^Q\}$, where Q represents the total number of points in segment l.

Intuitively, the optimal speed strategy within a segment is to accelerate to the maximum speed, maintain that speed, and then decelerate at the end. However, the slope at each point within the segment may vary, resulting in different speed and acceleration limits. Therefore, we design a dynamic programming method to obtain the optimal acceleration at each point within the segment. The key elements of dynamic programming are defined below:

- State: $S(q, v)$. The time consumption from point 0 to point p_l^q when the robot's speed is v in point p_l^q.
- Initial state: $S(0, 0) = 0$. Represent that the robot starts at the initial position with 0 speed.
- State space transition function:

$$S(q+1, v) = \min_{u,a} \left[S(q, u) + \frac{2w}{u+v} \right] \tag{18}$$

$$\text{s.t. } v^2 = u^2 + 2aw \tag{19}$$

$$0 \le v \le v^+_{p_l^q p_l^{q+1}} \tag{20}$$

$$a^-_{p_l^q p_l^{q+1}} \le a \le a^+_{p_l^q p_l^{q+1}} \tag{21}$$

where u is the current speed, v is the speed in the next step, a is the current acceleration. Functions (18) aim to minimize the time consumption by determining the optimal acceleration and speed. Constraints (19) calculate speed v. Constraints (20) and (21) are the safety constraints.

- Objective function: minimize the traveling time in segment l:

$$\min S\left(p_l^Q, 0\right) \tag{22}$$

With the state and transition equations defined, dynamic programming can solve the problem by constructing a table that represents the optimal solution at each state. By iteratively updating the table using the transition equations, we can determine the optimal solution for the speed control problem.

5.2.4. Integrate the three steps

The pseudocode in Algorithm 2 illustrates the algorithm's process.

Algorithm 2 Trajectory generation.

```
Input: the set of cells C
Initialize set of paths Z_in ← ∅, Z_out ← ∅
for cell c ∈ C do
    if (cell c is a dummy obstacle cell) then
        Divide each row in c into individual cells and add them to C
        Generate two horizontal paths for c and add them to Z_in
    else
        Generate zig-zag paths for c and add them to Z_in
    end if
end for
η ← Floyd algorithm (C)
for cell c_1 ∈ C, c_2 ∈ C/c_1 do
    if (η_{c_1 c_2} ≥ η̄) then
        z_{c_1 c_2} ← modified A* algorithm (c_1, c_2)
        Z_out ← Z_out ∪ {z_{c_1 c_2}}
    end if
end for
Initialize the set of speed commands Û ← ∅
for z ∈ Z_in ∪ Z_out do
    L(z) ← Divide z to a set of segments
    Û ← Û ∪ DP(L(z))
end for
Output: sets of paths Z_in, Z_out, and speed commands Û.
```

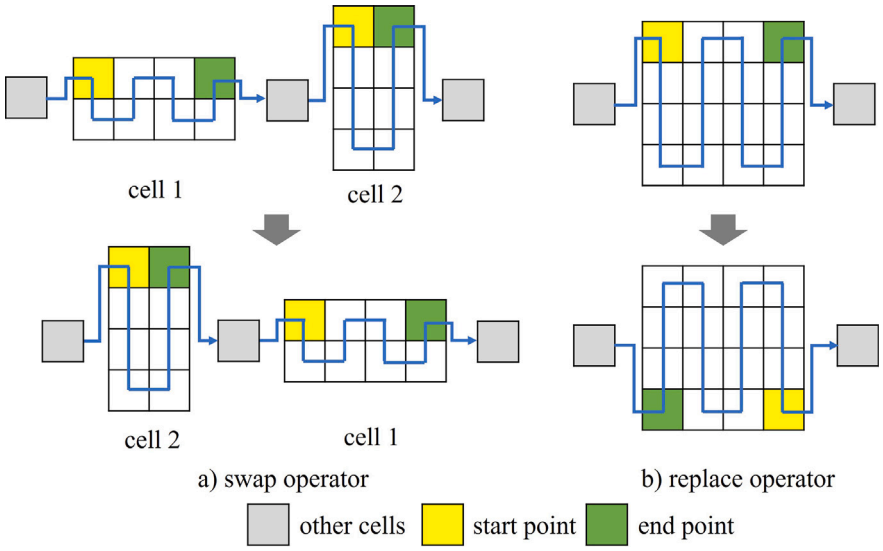


Fig. 7. Illustration of the swap operator and the replace operator.

5.3. Trajectory connection

The final step for the trajectory planning algorithm is to connect the candidate trajectories obtained from the trajectory generation. Based on these trajectories, we construct a directed graph $\mathcal{G}_{\text{path}} = (\mathcal{V}_{\text{path}}, \mathcal{E}_{\text{path}})$, where $\mathcal{V}_{\text{path}} = \{p_{\text{start}}(z), p_{\text{end}}(z) \mid z \in \mathcal{Z}_{\text{in}} \cup \mathcal{Z}_{\text{out}}\}$ and $\mathcal{E}_{\text{path}} = \{(i,j) \mid i,j \in \mathcal{V}_{\text{path}}, i \neq j\}$. The cost of the arcs is the time of the trajectories. Additionally, we set a dummy source and a dummy sink, connecting to the trajectories corresponding to point p_{start}. The costs related to the dummy source and sink are zero. This step aims to find a route with a minimal total cost that starts from the dummy source, covers all cells, and finally returns to the dummy sink. We use a greedy algorithm [34,35] and a simulated annealing (SA) algorithm to solve this TSP variant.

The greedy algorithm starts from the dummy source. It selects the nearest node as the next visited node at each current node. When a cell is covered, it is marked, and subsequent trajectories are prohibited from being selected. The solution obtained by the greedy algorithm is used as the initial solution for the SA algorithm. The principle of SA is to iteratively apply operators to the current solution to obtain its neighborhood solutions. It accepts the neighborhood solution that decreases the cost and occasionally accepts solutions that increase it, to avoid being trapped in local optima.

In our algorithm, we design two operators that are widely used in the routing problem [36] to generate the neighborhood solutions. The first operator is the swap operator, which exchanges the points where two cells are covered. The second operator is the replace operator, which replaces the zig-zag path covering a cell. The two operators are illustrated in Fig. 7.

The probability of accepting a worse solution decreases over time, allowing the algorithm to focus more on refining the best solution found. For a minimization problem, the probability of accepting a worse solution is calculated as follows:

$$P = e^{(Obj_{\text{current}} - Obj_{\text{new}})/T_{\text{current}}} \tag{23}$$

where Obj_{current} is the objective value of the current solution, Obj_{new} is the objective value of the new solution, and T is the current temperature. The temperature is gradually decreased according to a cooling schedule. A common approach is to use a geometric cooling schedule, where the temperature is updated as follows:

$$T_{\text{new}} = \beta T_{\text{current}} \tag{24}$$

where T_{new} is the updated temperature, T_{current} is the current temperature, and β is a constant factor between 0 and 1, typically close to 1. The algorithm terminates once T_{current} drops below a predefined threshold T_{min}.

Finally, Algorithm 3 presents the pseudocode of the trajectory connection algorithm. Here $swap(.)$ and $replace(.)$ represent the swap and replace operators, and $Obj(.)$ represents the objective value, i.e., the total travel time.

Algorithm 3 Trajectory connection algorithm.

Input: sets of paths $\mathcal{Z}_{\text{in}}$, $\mathcal{Z}_{\text{out}}$, and speed commands $\hat{\mathcal{U}}$.
Initialize $\mathcal{V}_{\text{path}} \leftarrow \{p_{\text{start}}(z), p_{\text{end}}(z) \mid z \in \mathcal{Z}_{\text{in}} \cup \mathcal{Z}_{\text{out}}\}$, $\mathcal{E}_{\text{path}} \leftarrow \{(i,j) \mid i,j \in \mathcal{V}_{\text{path}}, i \neq j\}$, and $weight() \leftarrow$ the travel time calculated from $\hat{\mathcal{U}}$.
Initialize a queue $\mathcal{J}$ with only the dummy sink. Denote the last node in $\mathcal{J}$ as $\mathcal{J}[0]$.
while $\mathcal{J}$ does not cover all cells **do**
    $cost_{\text{min}} \leftarrow \infty, i_{\text{next}} \leftarrow None.$
    for $(\mathcal{J}[0], j) \in \mathcal{E}_{\text{path}}$ **do**
        if The cell of node j is not visited in $\mathcal{J}$ and $weight(\mathcal{J}[0], j) \leq cost_{\text{min}}$ **then**
            $cost_{\text{min}} \leftarrow weight(\mathcal{J}[0], j), i_{\text{next}} \leftarrow j.$
        end if
    end for
    $\mathcal{J} \leftarrow \mathcal{J} \cup i_{\text{next}}.$
end while
Initialize T_{current} and $\mathcal{J}_{\text{opt}} \leftarrow \mathcal{J}$.
while $T_{\text{current}} \geq T_{\text{min}}$ **do**
    $\mathcal{J}_{\text{new}} \leftarrow swap(\mathcal{J})$ and $replace(\mathcal{J})$.
    if $e^{(Obj(\mathcal{J}) - Obj(\mathcal{J}_{\text{new}}))/T} \geq random(0,1)$ **then**
        $\mathcal{J} \leftarrow \mathcal{J}_{\text{new}}.$
    end if
    if $Obj(\mathcal{J}) \leq Obj(\mathcal{J}_{\text{opt}})$ **then**
        $\mathcal{J}_{\text{opt}} \leftarrow \mathcal{J}.$
    end if
    $T_{\text{new}} \leftarrow \beta T_{\text{current}}.$
end while
Output: path sequence $\mathcal{J}_{\text{opt}}$.

Table 1
Mower parameters used in the experiment.

Description	Value	Unit
Safe slope range	$[-30\%, 30\%]$	Grade
Maximum speed	3.5	m/s
Acceleration range (on $[0\%, 10\%]$ slope)	$[-2.5, 1.25]$	m/s^2
Acceleration range (on $[10\%, 30\%]$ slope)	$[-1.4, 0.6]$	m/s^2
Turning time	2	s

6. Experimental results

This section presents the instance sets and analyzes the performance of the MILP model and the algorithm. Our algorithm was coded in Java programming language using ILOG CPLEX 12.6.3 as the solver. The experiments were conducted on a machine equipped with a 3.2 GHz AMD Ryzen 7 7735HS with Radeon Graphics CPU and 16 GB of memory under the Windows 11 operating system.

6.1. Experimental settings

To evaluate the effectiveness of our model and algorithm, we design an algorithm to randomly generate large-size instances for the CTPP-3DT, which also serve as the benchmark instances for future study. The instance generation algorithm contains four steps:

1. Create a 2D binary matrix where each pixel is randomly set to 0 (service area) or 1 (restricted area) based on the obstacle probability σ.
2. Apply the cellular automata algorithm to smooth the binary map, ensuring obstacle connectivity and realistic terrain.
3. Create a height matrix where each pixel is assigned a random value between 0 and h if it is not an obstacle, where h is the highest height of the map.
4. Apply the Gaussian blur algorithm to the height matrix to ensure realistic terrain transitions.

Particularly, we set the start point and the end point at the same position, as in practical applications, the mower is expected to return to its initial position for retrieval.

We generated two sets of instances, small-size and large-size, to test the MILP model and the heuristic algorithm. To test the performance of the MILP model in different sizes, the small-size set includes ten instances with sizes 4×4 m^2, 6×6 m^2, ..., and 13×13 m^2, with parameters $\sigma = 0.3$ and $h = 1.0$ are manually generated. The large-size set includes three sizes, 50×50 m^2 and 100×100 m^2. For each size, four sets of instances are generated with different parameter settings: $\sigma = 0.32, 0.35$ and $h = 1.0, 1.2$. Three instances are generated by the algorithm for each parameter setting, resulting in 36 instances. The instances are named as "$length\_width\_\sigma\_h\_ID$", representing the length, width, parameter σ, parameter h, and the instance ID. Fig. 8 shows one of the 100×100 instances. Besides, the research team collaborated with an anonymous company to obtain parameters for the mower. Detailed parameters of the mower can be found in Table 1. The speed increment is set to $\Delta u = 0.35$ m/s, resulting in a maximum number of speed levels $N = 10$ in the set $\mathcal{U}$.

In the following test, the time limit on each run of the MILP model was set to 3600 s for instances smaller than 10×10, and 7200 s for larger instances. For the heuristic algorithm, we run each instance five times and report the best objective value along with the variance. In the SA, we set the initial temperature as 100, the cooling rate $\beta = 0.99$, and the maximum iteration for the algorithm as 100. $\bar{\eta} = 20$ and 40 for 50×50 and 100×100 instances, respectively. Our code and data are available at https://github.com/CATS-Lab/Mower-CTPP-3D.

Table 2
Comparison of MILP model and the heuristic algorithm on small-size instances.

Instance	MILP					Heuristic		Δ
	UB	LB	$\#Node$	$Time$	Gap	Obj	$Time$	
4_4_0.3_1.0_0	**15.67**	15.67	50,066	6	0.00	31.53	**0.16**	50.31
5_5_0.3_1.0_0	**31.42**	31.42	1058,801	65	0.00	46.93	**0.11**	33.06
6_6_0.3_1.0_0	**42.54**	36.44	22,212,925	3600	14.35	52.55	**0.00**	19.05
7_7_0.3_1.0_0	**54.09**	40.06	14,150,324	3600	25.93	76.35	**0.00**	29.16
8_8_0.3_1.0_0	**48.70**	23.44	7234,851	3600	51.87	71.99	**0.00**	32.35
9_9_0.3_1.0_0	**58.64**	36.05	7823,501	3600	38.53	108.13	**0.01**	45.77
10_10_0.3_1.0_0	**87.20**	52.43	14,007,681	7200	39.87	133.05	**0.01**	34.46
11_11_0.3_1.0_0	**76.82**	40.68	13,283,736	7200	47.04	144.22	**0.01**	46.74
12_12_0.3_1.0_0	**111.46**	38.41	16,844,730	7200	65.54	159.81	**0.11**	30.25
13_13_0.3_1.0_0	206.52	9.64	6528,749	7200	95.33	**164.09**	**0.11**	−25.86
Average	58.50	32.42	10,319,536	4327	37.85	98.86	0.05	35.68

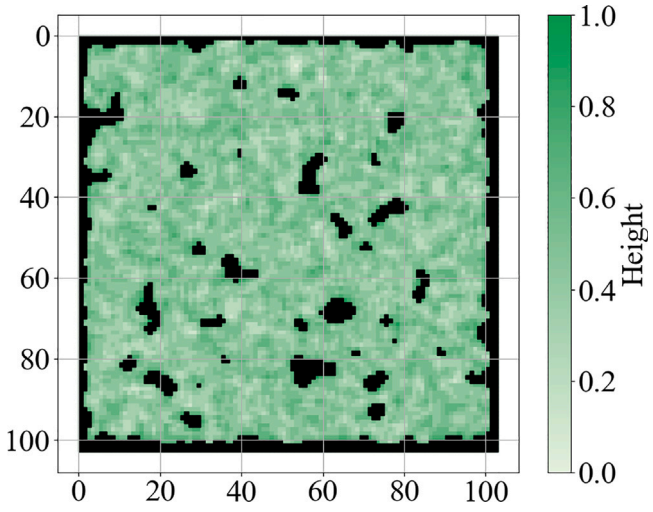


Fig. 8. The map of instance 100_100_0.35_1.0_1 generated by the instance generation algorithm (unit: m).

6.2. Performance comparison between the MILP and the heuristic algorithm on small-size instances

Table 2 reports the results of the MILP model and heuristic algorithm on small-size instances. The first column displays the instance names. Columns UB and LB represent the best upper and lower bounds for the MILP model. Column $\#Node$ is the number of nodes explored in the B&B (branch and bound) tree in the MILP model. Column Gap is the percentage difference between the best upper bounds and the lower bound. Column $Time$ reports the time in seconds consumed to solve one instance. Specifically, $Gap = (UB - LB)/UB \times 100$. Column Obj represents the objective values of the heuristic algorithm, where the objective function is equivalent to the MILP model of columns UB and LB. Column Δ is the percentage gap of the objective values between the MILP model and the heuristic algorithm, calculated as $\Delta = (Obj - UB)/Obj \times 100$.

Table 2 shows that the MILP model can only solve 4×4 and 5×5 instances to optimality within the time limit. It also obtains good solutions for instances smaller than 13×13 with better quality than the heuristic algorithm. However, for the 13×13 size instance, the upper bound solution obtained by the MILP is of lower quality than the solution found by the heuristic algorithm. Notably, CPLEX has explored a large number of B&B nodes for this problem, averaging 10319536. As the instance size increases, the time the MILP model takes to solve each B&B node also increases, leading to an exponential growth in computation time. Compared to the results of the MILP model, the heuristic algorithm's objective values increase by about 36%, but it solves all instances within 0.2 s. Therefore, in terms of

Table 3
Performance of the heuristic algorithm on 50 × 50 size instances.

Instance	*Obj*	*Var*	*#turn*	$\bar{v}$	*Time*
0.32_1.0_0	1688.0	2.8	189	2.01	2.6
0.32_1.0_1	1489.6	4.8	134	2.16	0.7
0.32_1.0_2	1566.6	9.9	151	2.12	0.6
0.32_1.2_0	1293.3	31.3	144	2.00	2.4
0.32_1.2_1	1612.3	168.8	154	2.12	1.8
0.32_1.2_2	1516.6	0.0	145	2.10	1.4
0.35_1.0_0	1706.5	100.3	192	2.00	2.9
0.35_1.0_1	1705.6	0.0	196	2.00	2.5
0.35_1.0_2	1764.7	0.0	211	1.97	3.5
0.35_1.2_0	1826.2	9.8	223	1.93	1.4
0.35_1.2_1	1726.8	3.5	195	2.01	1.5
0.35_1.2_2	1789.9	0.0	213	1.94	1.5
Average	1640.5	13.8	179	2.03	1.9

Table 4
Performance of the heuristic algorithm on 100 × 100 size instances.

Instance	*Obj*	*Var*	*#turn*	$\bar{v}$	*Time*
0.32_1.0_0	5792.5	101.5	464	2.21	85.6
0.32_1.0_1	5847.7	62.2	468	2.21	97.6
0.32_1.0_2	5690.5	10.2	427	2.23	65.0
0.32_1.2_0	5695.7	0.0	438	2.22	54.1
0.32_1.2_1	5903.1	26.1	454	2.21	61.8
0.32_1.2_2	5994.8	220.9	507	2.19	38.5
0.35_1.0_0	6099.3	0.0	544	2.16	98.8
0.35_1.0_1	6284.3	74.0	577	2.14	86.4
0.35_1.0_2	6168.8	59.0	570	2.13	71.6
0.35_1.2_0	6252.4	61.0	550	2.14	91.5
0.35_1.2_1	6443.7	0.0	625	2.09	63.9
0.35_1.2_2	6352.7	31.3	592	2.12	86.0
Average	6043.8	53.9	518	2.17	75.1

solution quality, the MILP model results are much better. However, in terms of computation time, the heuristic algorithm can solve the problems much faster. Therefore, we recommend using the MILP model for problems of approximately 13 × 13 in size and heuristic algorithms for larger-size problems.

6.3. Performance of the heuristic algorithm on large-size instances

The heuristic approach is applicable in tackling large-size instances. Tables 3–4 show the results of our heuristic algorithm on large-size instances with 50 × 50 and 100 × 100 maps. Column *Obj* represents the optimal objective values among the five test runs. Column *Var* represents the variance of objective values among the five test runs. Column *#turn* is the number of turns of the solution. Column $\bar{v}$ is the average speed of the solution. Column *Time* represents the computational time.

Tables 3 and 4 show that our algorithm has a very fast computation speed. Specifically, for 50 × 50 instances, the instances are solved within 2 s, and for 100 × 100 instances, the instances are solved within 1 min. Analyzing the instance generation parameters, the average computational time for obstacle probability $\sigma = 0.32$ and $\sigma = 0.35$ on 50 × 50 and 100 × 100 size instances are 1.6, 2.2, 67.1, and 76.4 s, respectively. This result indicates that an increase in obstacles significantly increases the complexity of the problem. The average computational time for height parameter $h = 1.0$ and $h = 1.2$ on 50 × 50 and 100 × 100 size instances are 2.1, 1.7, 85.8, and 66.0 s, respectively, showing no significant difference. Additionally, the values in column *Var* indicate that the variance across multiple runs is generally small, and in some cases, it is even zero. This suggests that the solution quality of the heuristic algorithm is relatively stable, which can be attributed to the fact that the randomness only arises from the SA component. Finally, we find a relationship between the number of turns and computational time. The Pearson correlation coefficients between column *Time* and column *#turn*, and column $\bar{v}$ are 0.97, indicating a positive correlation for these two variables. When the number of turns is higher, the robot needs to decelerate to zero more frequently, reducing the average speed.

Table 5
Performance of the heuristic algorithm on 125 × 125 size instances.

Instance	*Obj*	*Var*	*#turn*	$\bar{v}$	*Time*
0.32_1.0_0	8308.2	85.8	538	2.29	71.8
0.32_1.0_1	8335.2	78.4	545	2.29	68.6
0.32_1.0_2	8549.5	58.6	616	2.26	81.0
Average	8397.6	74.3	566	2.28	73.8

Table 6
Comparison between turning angle sets $\mathcal{K}_1$ (minimum turning angle $\frac{1}{2}\pi$) and $\mathcal{K}_2$ (minimum turning angle $\frac{1}{4}\pi$).

Instance	$\mathcal{K}_1$				$\mathcal{K}_2$			
	Obj	*#turn*	$\bar{v}$	*Time*	*Obj*	*#turn*	$\bar{v}$	*Time*
0.32_1.0_0	1688.0	189	2.01	2.6	**1624.8**	222	2.44	**2.0**
0.32_1.0_1	1489.6	134	2.16	0.7	**1319.5**	140	2.72	**0.4**
0.32_1.0_2	1566.6	151	2.12	**0.6**	**1389.7**	151	2.68	0.7
0.32_1.2_0	1293.3	144	2.00	**2.4**	**1182.5**	164	2.42	3.4
0.32_1.2_1	1612.3	154	2.12	1.8	**1446.3**	166	2.67	**1.3**
0.32_1.2_2	1516.6	145	2.10	**1.4**	**1564.2**	189	2.60	1.8
0.35_1.0_0	1706.5	192	2.00	2.9	**1643.4**	231	2.39	**1.9**
0.35_1.0_1	1705.6	196	2.00	2.5	**1593.4**	216	2.44	**1.7**
0.35_1.0_2	1764.7	211	1.97	3.5	**1713.0**	233	2.43	**3.2**
0.35_1.2_0	1826.2	223	1.93	**1.4**	**1703.0**	221	2.40	1.8
0.35_1.2_1	1726.8	195	2.01	**1.5**	**1620.6**	212	2.49	2.1
0.35_1.2_2	1789.9	213	1.94	**1.5**	**1705.5**	223	2.39	1.9
Average	1640.5	179	2.03	1.9	1542.2	197	2.51	1.8

To explore the computational limits of our algorithm, we conducted additional experiments on larger instances, including 125 × 125, 150 × 150, and 200 × 200 grids. For the 125 × 125 instances, the heuristic algorithm was still able to produce solutions within 2 min. Compared to the 100 × 100 cases, the variance of the results increased slightly but remained below 100 on average. Detailed results are provided in Table 5.

However, for instances of size 150 × 150 and above, we observed that the algorithm failed to complete due to memory limitations during the execution of the A* search in Algorithm 2. This is due to the significantly larger search space introduced by point-to-point planning with additional constraints such as turning time. We therefore recommend replacing this component with a more memory-efficient heuristic shortest path algorithm [37] when solving extremely large instances.

6.4. Impact of the turning angles

In the CTPP-3DT problem, the discretized set of feasible turning angles $\mathcal{K}$ constrains the possible trajectories of the robot. It is therefore important to explore how the granularity of turning angles affects both the objective value and the computational efficiency of the algorithm. To investigate this, we conduct experiments on 50 × 50 instances using two angle sets: $\mathcal{K}_1 = \{0, \frac{1}{2}\pi, \pi, \frac{3}{2}\pi\}$ and $\mathcal{K}_2 = \{0, \frac{1}{4}\pi, \frac{1}{2}\pi, \frac{3}{4}\pi, \pi, \frac{5}{4}\pi, \frac{3}{2}\pi, \frac{7}{4}\pi\}$, which correspond to minimum turning angles of $\frac{1}{2}\pi$ and $\frac{1}{4}\pi$, respectively.

As shown in Table 6, refining the turning angle granularity (i.e., allowing more flexible directions) leads to improvement in objective values across all instances. On average, the completion time decreases by 5.99%. Interestingly, the number of turns increases with $\mathcal{K}_2$, but the robot's average speed also increases. This suggests that more flexible turning enables longer and faster straight-line motions, which offsets the cost of additional turning. In terms of computation time, we observe no significant increase. In some cases, the runtime even decreases. This is because the turning angles primarily affect the trajectory connection phase, where a modified A* algorithm is used. While more turning options introduce more branching, the overall complexity remains low

due to the heuristic nature of A*. However, for angle sets with finer resolution (i.e., minimum turning angle smaller than $\pi/4$), a denser grid (smaller cell width w) may be required to accurately represent feasible paths. For most use cases, we recommend choosing $\pi/2$ or $\pi/4$ based on the mechanical capabilities of the robot.

6.5. Comparison between different CTPP strategies

The most important features of our CTPP-3DT are the incorporation of safety constraints and a time-aware objective function. As discussed in the introduction and literature review, many existing studies neglect the necessity of safety constraints, which may result in dangerous trajectories. Moreover, some of the literature focuses on minimizing energy consumption rather than completion time. To explore the trade-off between minimizing time and energy, we develop and compare four strategies based on variations of our proposed algorithm:

- **Original Strategy**: This strategy optimizes speed and acceleration within safety limits to minimize total completion time, as introduced in previous sections.
- **Conservative Strategy**: To ensure absolute safety during movement, this strategy uses the most conservative speed and acceleration values during trajectory planning (e.g., maximum acceleration is limited to 0.6 m/s^2).
- **Aggressive Strategy**: This strategy ignores safety constraints by removing limits related to maximum safe slope and the speed/acceleration bounds on steep terrain (e.g., maximum acceleration is allowed up to 1.25 m/s^2).
- **Energy-aware Strategy**: This strategy respects the safety constraints, but modifies the objective function to minimize energy consumption instead of the completion time. Since there is no widely accepted energy consumption model for mowers in the literature, we adopt an energy consumption model for agricultural machines proposed in [38], given that our CTPP-3DT has broad applications in agricultural domains. Specifically, the power is modeled as a function of speed:

$$P(v) = \left(p_1 + v \cdot w \cdot p_2\right) + \left(p_3 + d \cdot v^2 \cdot p_4\right) w + (0.115M \cdot v \cdot a/3600) + P_{\text{air}} + \left(g \cdot m \cdot v \cdot r_{rc}/1800\right), \quad (25)$$

where P is the required power (kW), v is the robot speed (km/h), w is the working width (m), d is the working depth (set to 0), M is the total vehicle and implement mass including the tank load (kg), m is the implement mass (kg), a is the inclination of the terrain (%), g is the gravitational acceleration (9.81 m/s^2), P_{air} is the power for air conditioning and compressors (kW), and r_{rc} is the rolling resistance coefficient (set to 0.06). Constants $p_1 = -0.2683$, $p_2 = 0.06775$, $p_3 = 4.55752$, and $p_4 = 0.03141$ are equipment-specific parameters, which we follow the example values in [7].

To convert the objective to energy minimization, the dynamic programming cost function Eq. (18) is modified as follows:

$$S(q+1, v) = \min_{u,a} \left[S(q,u) + P(u) \cdot \frac{2w}{u+v} \right]. \quad (26)$$

Following the same experimental setup as before, we test all four strategies on 50 × 50 size instances. Each instance is run five times, and we report the best objective value from the five runs. The variances observed are consistent with those reported in Section 6.3, and therefore, we do not report variance in this section for simplicity.

Fig. 9(a) compares the completion time and energy consumption of the four strategies on 50 × 50 size instances. It can be observed that both the completion time and energy consumption of the conservative strategy are significantly higher than those of the other three strategies by more than 100%. This indicates that adopting a conservative approach to ensure absolute safety severely compromises operational efficiency.

To better examine the differences among the remaining three strategies, we provide a zoomed-in view on the right side of Fig. 9(a). The results show that the performance of the energy-aware strategy is very close to that of the time-aware strategy (i.e., the original strategy). Although power consumption is positively correlated with robot speed, increasing speed also reduces the overall operation time, which in turn lowers total energy consumption. This indicates that in some cases, minimizing energy usage and minimizing completion time are not conflicting objectives but can be aligned. Specifically, when the marginal increase in power due to higher speed is offset by the reduction in operation time, both energy-aware and time-aware strategies may yield similar results.

The difference between the aggressive and original strategies is also smaller compared to the gap between the conservative strategy and others. On average, the completion time of the aggressive strategy is approximately 2.9% shorter than that of the original strategy. However, we observe that the aggressive strategy violates safety constraints an average of 69.1 times per instance, indicating that it frequently places the robot in unsafe conditions. To further investigate the differences between the aggressive and original strategies under varying terrain conditions, we generated five additional instances with steeper terrain ($h = 1.3$ m, size 50 × 50). The results are shown in Fig. 9(b). Compared to the flatter terrain in Fig. 9(a), the performance gap between the two strategies widens under steeper conditions. In this case set, the average completion time difference increases to 6.5%, and the number of safety constraint violations by the aggressive strategy rises to 91.4. Overall, the results show that in steeper terrains, the aggressive strategy achieves faster coverage at the cost of significantly higher safety risk. This demonstrates the necessity of incorporating safety constraints in trajectory optimization. The original strategy effectively balances safety and efficiency, keeping the overall completion time within an acceptable range while ensuring safe operation.

6.6. Sensitivity analyses

Since our CTPP-3DT is a new variant of the CPP, we conduct sensitivity analyses on key parameters, including the maximum safety slope θ^+, the speed and acceleration limit v^+ and a^+, and turning time t_{turn}, to investigate the characteristics of this variant. Experiments are performed on 50 × 50 size instances.

Speed and acceleration limit. Speed and acceleration limits affect the robot's moving speed, thereby influencing its operational efficiency. When the robot moves faster, its efficiency improves. We analyze the influence of these parameters by testing three settings, which represent low, medium, and high levels of robot performance. The detailed parameters for each setting are provided in Table 7. Fig. 10 shows the results under these three parameter settings. It shows that when the robot is set to the medium speed mode, the average completion time decreased from 1644.5 to 1430.8, a reduction of 13.0%. The robot's average speed increased from 2.03 to 3.34, an improvement of 33.0%. When the robot is further adjusted to the fast mode, the completion time continues to decrease by 12.5%, and the average speed increases by 24.0%. This conclusion indicates that increasing speed and acceleration can significantly enhance the robot's operational efficiency.

Turning time. Now, we study the impact of the robot's turning time. From Tables 3–4, we can observe that covering 50 × 50 and 100 × 100 size instances requires an average of 198 and 580 turns, respectively, indicating that turning time significantly affects the total time consumption. Therefore, we tested the results for $t_{\text{turn}} = 2, 1$, and 0.5. As shown in Fig. 11, the average completion time significantly reduces when t_{turn} decreases. When t_{turn} decreases from 2S to 1S, the average completion time decreases by 11.3%, i.e., 185.3. When t_{turn} decreases from 1S to 0.5S, the average completion time decreases

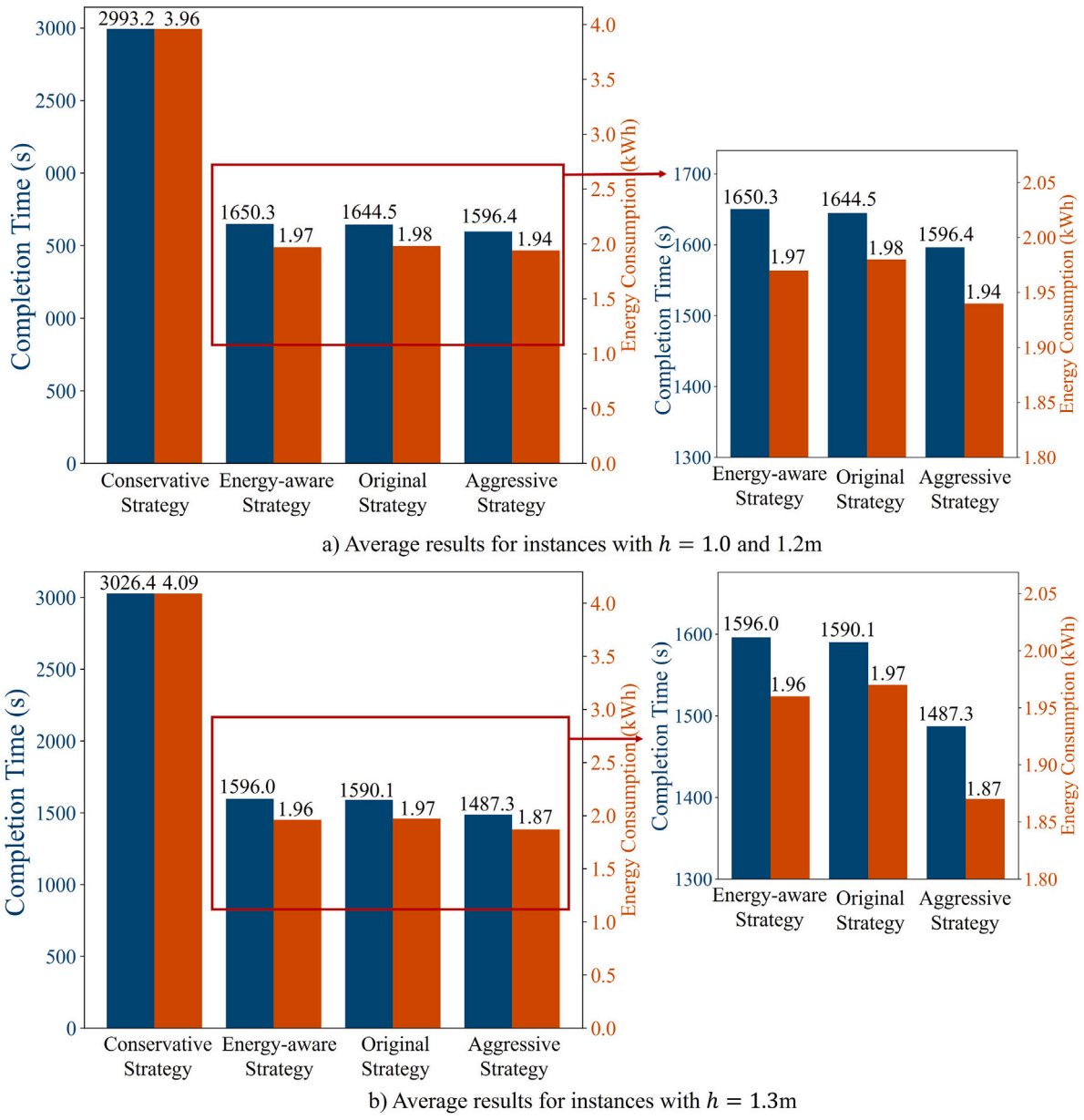


Fig. 9. Comparison between the four CTPP strategies in 50 × 50 size instances.

Table 7
Three speed and acceleration parameter settings for the sensitivity analyses.

Description	Setting 1	Setting 2	Setting 3
Maximum speed (m/s)	3.5	5.0	6.5
Acceleration range on $[-0.1, 0.1]$ slope (m/s^2)	$[-2.5, 1.25]$	$[-2.5, 1.7]$	$[-2.5, 2.2]$
Acceleration range on $[-0.3, -0.1] \cup [0.1, 0.3]$ slope (m/s^2)	$[-1.4, 0.6]$	$[-1.4, 1.0]$	$[-1.4, 1.4]$

by 3.0%, i.e., by 43.5. These reductions are slightly greater than the decrease in turning time, indicating that reducing t_{turn} allows the algorithm to find more efficient trajectories.

Maximum safety slope. In CTPP-3DT, the maximum safety slope θ^+ influences the range of the robot's movement. We analyze the influence of this parameter by changing its value from 0.3 to 0.4 and 0.5. Fig. 12 shows the results under different values of θ^+. We can observe that when $\theta^+ = 0.4$, the average objective function value is slightly lower than that for $\theta^+ = 0.3$. However, the results for $\theta^+ = 0.5$ are very close to those for $\theta^+ = 0.4$. This indicates that increasing the maximum safety slope can enhance the robot's operational efficiency, but the marginal benefits of further increasing the maximum safety slope with current technology are limited. These limited marginal benefits could be due to the smoothing of map heights in our instances.

7. Conclusion

To address the safety challenges of automated mowing on uneven terrain, this paper proposes the CTPP-3DT for automated lawn mowers and other agricultural machines. In this problem, we introduce height onto traditional 2D maps to calculate the robot's slope along its trajectory. Safety constraints are defined as the maximum allowable slopes for robot movement and the speed and acceleration limits under varying slopes. The problem aims to determine the optimal path and

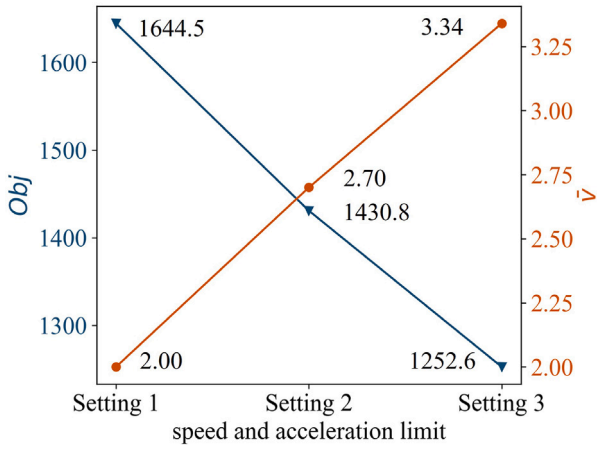


Fig. 10. The impact of the speed and acceleration limit.

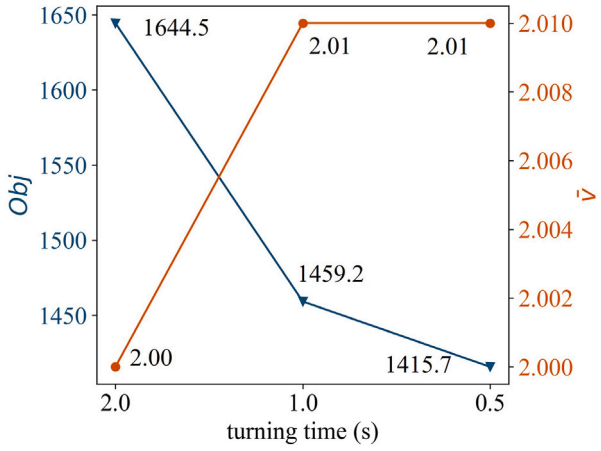


Fig. 11. The impact of the maximum turning time.

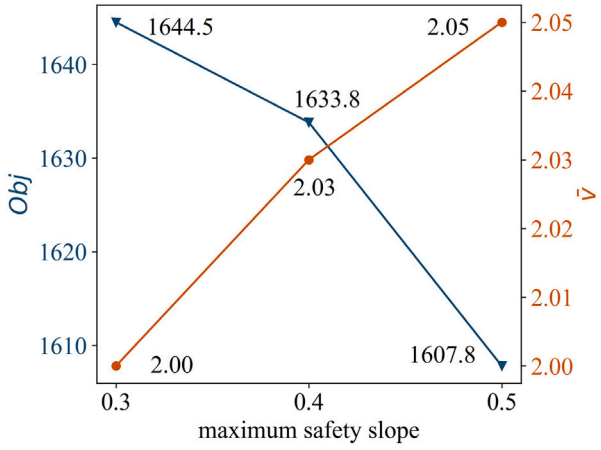


Fig. 12. The impact of the maximum safety slope.

speed that minimizes the completion time while satisfying the safety constraints.

To tackle this problem, the paper presents a solution framework employing two methods. The exact algorithm determines optimal solutions for small-size scenarios using a MILP model based on a graph expansion method to represent turning using dummy arcs. The heuristic algorithm divides the area into cells via a tailed RBCD. The candidate trajectories to cover and connect cells are generated based on a modified A* algorithm and dynamic programming. Finally, an SA algorithm is applied to select the optimal trajectories in the candidate trajectory set.

Experiments are conducted based on the instances generated by a random algorithm. The results show that: (a). The MILP model performs well for instances smaller than 13×13, while the heuristic algorithm is more effective for larger instances. (b). Incorporating safety constraints in the CTPP-3DT avoids the potentially dangerous trajectories observed in the aggressive strategy, while achieving over a 40% reduction in completion time compared to the conservative strategy. This highlights that incorporating safety constraints can significantly improve both safety and efficiency. (c). Increasing the mower's movement speed, acceleration, and turning speed can significantly reduce completion time, though the benefits of increasing the safety slope need further testing with real-world data.

As a potential direction for future work, more scalable exact methods such as dynamic programming, spanning tree decomposition, or branch-and-cut algorithms could be explored. These approaches may help improve solution quality or provide theoretical guarantees without incurring the computational cost of solving large-scale MILPs. Besides, although we conducted a preliminary comparison between the energy-aware and time-aware strategies in the experimental section, the energy consumption model used may differ from that of actual lawn mowers, both in terms of model structure and parameter settings. For example, the energy model adopted in this study does not account for the effect of acceleration, which is an important factor in vehicle dynamics and has been highlighted in literature [39]. Moreover, the energy-aware strategy considered in this study is not a fully tailored algorithm for optimizing energy consumption. Comparisons with more advanced energy optimization approaches from the literature may yield different results. On the hardware side, considering the differences between generated instances and real-world data, we plan to test the algorithm using real-world data instead of randomly generated instances. We aim to develop a perception system based on RGB-D cameras and a mapping algorithm for the localization of the mower and collect real-world data for testing.

CRediT authorship contribution statement

Hang Zhou: Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology. **Peng Zhang:** Writing – review & editing, Validation, Formal analysis, Data curation, Conceptualization. **Zhaohui Liang:** Writing – review & editing, Validation, Formal analysis, Data curation. **Hangyu Li:** Writing – review & editing, Validation, Formal analysis, Data curation. **Xiaopeng Li:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data and code will be opened after the acceptance of this paper.

References

[1] M. Skoczeń, M. Ochman, K. Spyra, M. Nikodem, D. Krata, M. Panek, A. Pawłowski, Obstacle detection system for agricultural mobile robot application using RGB-d cameras, Sensors 21 (16) (2021) 5292.

[2] A.S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, N. Roy, Visual odometry and mapping for autonomous flight using an RGB-d camera, in: Robotics Research: The 15th International Symposium ISRR, Springer, 2017, pp. 235–252.

[3] I.A. Hameed, A. la Cour-Harbo, O.L. Osen, Side-to-side 3D coverage path planning approach for agricultural robots to minimize skip/overlap areas between swaths, Robot. Auton. Syst. 76 (2016) 36–45.
[4] H. Nagar, A. Paul, R. Machavaram, P. Soni, et al., Reinforcement learning particle swarm optimization based trajectory planning of autonomous ground vehicle using 2D LiDAR point cloud, Robot. Auton. Syst. 178 (2024) 104723.
[5] S. Dogru, L. Marques, ECO-CPP: Energy constrained online coverage path planning, Robot. Auton. Syst. 157 (2022) 104242.
[6] E. Galceran, M. Carreras, A survey on coverage path planning for robotics, Robot. Auton. Syst. 61 (12) (2013) 1258–1276.
[7] I.A. Hameed, Intelligent coverage path planning for agricultural robots and autonomous machines on three-dimensional terrain, J. Intell. Robot. Syst. 74 (3) (2014) 965–983.
[8] C. Wu, C. Dai, X. Gong, Y.J. Liu, J. Wang, X.D. Gu, C.C. Wang, Energy-efficient coverage path planning for general terrain surfaces, IEEE Robot. Autom. Lett. 4 (3) (2019) 2584–2591.
[9] Y. Ding, L. Wang, Y. Li, D. Li, Model predictive control and its application in agriculture: A review, Comput. Electron. Agric. 151 (2018) 104–117.
[10] J. Backman, T. Oksanen, A. Visala, Navigation system for agricultural machines: Nonlinear model predictive path tracking, Comput. Electron. Agric. 82 (2012) 32–43.
[11] P. Zhang, H. Huang, H. Zhou, H. Shi, K. Long, X. Li, Online adaptive platoon control for connected and automated vehicles via physics enhanced residual learning, Transp. Res. C 178 (2025) 105242, http://dx.doi.org/10.1016/j.trc.2025.105242.
[12] M. Höffmann, S. Patel, C. Büskens, Optimal guidance track generation for precision agriculture: A review of coverage path planning techniques, J. Field Robot. 41 (3) (2024) 823–844.
[13] C.S. Tan, R. Mohd-Mokhtar, M.R. Arshad, A comprehensive review of coverage path planning in robotics using classical and heuristic algorithms, IEEE Access 9 (2021) 119310–119342.
[14] L.C. Santos, F.N. Santos, E.S. Pires, A. Valente, P. Costa, S. Magalhães, Path planning for ground robots in agriculture: A short review, in: 2020 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC, IEEE, 2020, pp. 61–66.
[15] E.V. Vazquez-Carmona, J.I. Vasquez-Gomez, J.C. Herrera-Lozada, M. Antonio-Cruz, Coverage path planning for spraying drones, Comput. Ind. Eng. 168 (2022) 108125.
[16] A. Garg, S.S. Jha, Learning continuous multi-UAV controls with directed explorations for flood area coverage, Robot. Auton. Syst. 180 (2024) 104774.
[17] B. Narottama, S.Y. Shin, et al., UAV coverage path planning with quantum-based recurrent deep deterministic policy gradient, IEEE Trans. Veh. Technol. (2023).
[18] G. Han, Z. Zhou, T. Zhang, H. Wang, L. Liu, Y. Peng, M. Guizani, Ant-colony-based complete-coverage path-planning algorithm for underwater gliders in ocean areas with thermoclines, IEEE Trans. Veh. Technol. 69 (8) (2020) 8959–8971.
[19] M. Charitidou, T. Keviczky, An MILP approach for persistent coverage tasks with multiple robots and performance guarantees, Eur. J. Control 64 (2022) 100610.
[20] Y. Gabriely, E. Rimon, Spanning-tree based coverage of continuous areas by a mobile robot, Ann. Math. Artif. Intell. 31 (2001) 77–98.
[21] N. Hazon, G.A. Kaminka, Redundancy, efficiency and robustness in multi-robot coverage, in: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, IEEE, 2005, pp. 735–741.
[22] P. Zhou, Z.m. Wang, Z.n. Li, Y. Li, Complete coverage path planning of mobile robot based on dynamic programming algorithm, in: 2nd International Conference on Electronic & Mechanical Engineering and Information Technology, Atlantis Press, 2012, pp. 1837–1841.
[23] R.N. De Carvalho, H. Vidal, P. Vieira, M. Ribeiro, Complete coverage path planning and guidance for cleaning robots, in: ISIE'97 Proceeding of the IEEE International Symposium on Industrial Electronics, 2, IEEE, 1997, pp. 677–682.
[24] H. Choset, P. Pignon, Coverage path planning: The boustrophedon cellular decomposition, in: Field and Service Robotics, Springer, 1998, pp. 203–209.
[25] H. Choset, Coverage of known spaces: The boustrophedon cellular decomposition, Auton. Robots 9 (2000) 247–253.
[26] E.U. Acar, H. Choset, A.A. Rizzi, P.N. Atkar, D. Hull, Morse decompositions for coverage tasks, Int. J. Robot. Res. 21 (4) (2002) 331–344.
[27] S.C. Wong, B.A. MacDonald, A topological coverage algorithm for mobile robots, in: Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453), vol. 2, IEEE, 2003, pp. 1685–1690.
[28] T. Oksanen, A. Visala, Coverage path planning algorithms for agricultural field machines, J. Field Robot. 26 (8) (2009) 651–668.
[29] A. Šelek, M. Seder, M. Brezak, I. Petrović, Smooth complete coverage trajectory planning algorithm for a nonholonomic robot, Sensors 22 (23) (2022) 9269.
[30] C.B. Barber, D.P. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Trans. Math. Softw. (TOMS) 22 (4) (1996) 469–483.
[31] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (2) (1968) 100–107.
[32] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, L. Jurišica, Path planning with modified a star algorithm for a mobile robot, Procedia Eng. 96 (2014) 59–69.
[33] R.W. Floyd, Algorithm 97: shortest path, Commun. ACM 5 (6) (1962) 345–345.
[34] H. Zhou, H. Qin, C. Cheng, L.M. Rousseau, An exact algorithm for the two-echelon vehicle routing problem with drones, Transp. Res. Part B: Methodol. 168 (2023) 124–150.
[35] H. Zhou, H. Qin, Z. Zhang, J. Li, Two-echelon vehicle routing problem with time windows and simultaneous pickup and delivery, Soft Comput. 26 (7) (2022) 3345–3360.
[36] H. Zhou, Y. Li, C. Ma, K. Long, X. Li, Modular vehicle routing problem: Applications in logistics, Transp. Res. Part E: Logist. Transp. Rev. 197 (2025) 104022.
[37] L. Fu, D. Sun, L.R. Rilett, Heuristic shortest path algorithms for transportation applications: State of the art, Comput. Oper. Res. 33 (11) (2006) 3324–3343.
[38] N. Fröba, Benötigte Traktormotornennleistung bei landwirtschaftlichen Arbeiten, KTBL, 1995.
[39] K. Ma, H. Zhou, Z. Liang, X. Li, Automated vehicle microscopic energy consumption study (AV-micro): Data collection and model development, Energy 320 (2025) 135096.

Hang Zhou is a Ph.D. candidate in the Department of Civil and Environmental Engineering at the University of Wisconsin, Madison, Madison, WI, USA. He obtained his bachelor's degree in Logistics Management from the Huazhong University of Science and Technology, Wuhan, China in 2023. His main research interests are the evaluation and control of automated vehicles.

Peng Zhang is a Ph.D. candidate in the Department of Civil and Environmental Engineering at the University of Wisconsin-Madison, Madison, WI, USA. He earned his bachelor's degree in Electrical Engineering from Fuzhou University, Fuzhou, China, in July 2017, and later pursued a master's degree in Industrial Engineering at the University of South Florida, Tampa, FL, USA, completing it in August 2019. His primary research interests include vehicle car-following stability control, AI-based vehicle control systems, autonomous vehicle testing, and robot development.

Zhaohui Liang is a Ph.D. candidate in the Department of Civil and Environmental Engineering at the University of Wisconsin, Madison, Madison, WI, USA. He obtained his bachelor's degree from the Harbin Institute of Technology, Weihai, China in June 2019. His main research interests are Eco-driving algorithms and connected autonomous vehicle testing.

Hangyu Li received his B.Eng. degree in vehicle engineering from Tsinghua University, Beijing, China in 2021, and an M.Phil. degree in intelligent transportation from the Hong Kong University of Science and Technology, Hong Kong SAR, China in 2023. He is currently working with Prof. Xiaopeng Li toward a Ph.D. degree at the University of Wisconsin-Madison, Madison, United States. His research interests focus on automation and cooperation in intelligent transportation systems.

Xiaopeng Li received the B.S. degree in civil engineering with a computer engineering minor from Tsinghua University, Beijing, China, in 2006, and the M.S. degrees in civil engineering and applied mathematics and the Ph.D. degree in civil engineering from the University of Illinois at Urban-Champaign, Champaign, IL, USA, in 2007, 2010, and 2011, respectively. He is currently a Professor with the Department of Civil and Environmental Engineering, University of Wisconsin-Madison, Madison, WI, USA. His main research interests include automated vehicle control and connected and interdependent infrastructure systems.