Collision-Aware Route Planning in Warehouses Made Efficient: A Strip-based Framework

Dingyuan Shi*, Nan Zhou*, Yongxin Tong*, Zimu Zhou†, Yi Xu*, Ke Xu*

* *SKLSDE Lab, BDBC and IRI, Beihang University, Beijing, China*
† *City University of Hong Kong, Hong Kong, China*
*{chnsdy, nzhou, yxtong, xuy, kexu}@buaa.edu.cn
†zimuzhou@cityu.edu.hk

Abstract—**Multi-robot systems are deployed in modern warehouses to reduce operational cost. The robots are tasked to deliver items stored on racks to pickers for fast distribution. A central algorithmic problem is collision-aware route planning, which aims to plan shortest routes for robots to deliver racks while avoiding collision with racks, pickers, and other robots. Prior solutions are inefficient in real-world warehouses, where route planning requests emerge online and at large scale. In this paper, we identify collision judgement in grid-based warehouse representation as the primary efficiency bottleneck, and propose a novel <u>S</u>trip-based <u>R</u>oute <u>P</u>lanning framework (SRP). Specifically, we exploit the regularity in warehouse layouts, and aggregate grids into strips. The strip-based representation also converts collisions of 3-dimensional (2-dimensional space and 1-dimensional time) routes into 2-dimensional (1-dimensional space and 1-dimensional time) segment intersections, which can be fast checked via computational geometry. We further accelerate the collision judgement via indexing on segments within strips. Theoretical analysis shows a reduction of time complexity from square to linear-logarithmic. Experimental results on datasets collected from real-world robotized warehouses show that our SRP is up to $227\times$ faster than existing methods.**

I. INTRODUCTION

Robots are gaining increasing popularity in warehouses for cost-effective item distribution [1]. They are typically deployed in warehouses to deliver racks containing items to pickers for processing, and return empty racks back to their original locations. Such robotized warehouses significantly reduce the human labor cost and improve the distribution efficiency.

A critical issue in such warehouses is to plan collision-free routes for multiple robots, known as the <u>C</u>ollision-<u>A</u>ware <u>R</u>oute <u>P</u>lanning (CARP) problem [2]–[6]. Concretely, we need to plan shortest paths for every robot moving between racks and pickers without colliding against racks, pickers, and other moving robots. A primary challenge is to plan these routes with *high-efficiency*, because the distribution requests emerge dynamically *i.e.*, in an *online* setting [3]–[6] and at *large scale*. For example, over 2 billion logistic orders were created in a single shopping event in 2020 [6].

Despite recent solutions [3]–[6] to the online CARP problem, they hardly scale to real-world warehouses. The state-of-the-art [4] incurs a delay of over 60 seconds when planning merely 600 origin-destination pairs, which is unacceptable for large-scale warehouses that must perform route planning for $100,000$ origin-destination pairs per day [6]. We argue that

the efficiency of prior proposals is constrained by *collision judgement* on *grid*-based warehouse representations. Specifically, collisions are modeled as spatiotemporal coordinates conflicts, and A* based searching [7] is conducted on the *3-dimensional* space. Albeit various branching tricks, the 3-dimensional search space remains the efficiency bottleneck, even for mid-sized warehouses partitioned into 200×100 grids.

To break the efficiency bottleneck of collision judgement, we devise a new <u>S</u>trip-based <u>R</u>oute <u>P</u>lanning framework (SRP). The key insight is to exploit the regularity of the warehouse layout for acceleration. Instead of representing the warehouse as grids, we aggregate grids into *strips* upon which we perform inter- and intra-strip route planning. Such separation restricts the collision domain to *within strips*. Accordingly, at inter-strip level we only consider shortest path finding among strips without collision awareness while at intra-strip level we make collision judgement and plan routes within a strip. Such strip aggregation also compresses the search space of collisions from 3-dimensional (2-D space + 1-D time) to 2-dimensional (1-D space + 1-D time), which allows representing paths within a strip by 2-dimensional segments. Furthermore, collision can now be fast detected via computational geometry. Finally, we also design indexing to speed up collision detection. Theoretical analysis shows that SRP reduces time complexity from $O((HW)^2)$ to $O((HW)\log(HW))$ in a warehouse with width of W grids and length of H grids. Experiments show that SRP can be up to $227\times$ faster than prior solutions on datasets collected from real-world robotized warehouses.

Our main contributions can be summarized as follows.

- We propose a strip-based representation for fast collision awareness. It breaks the efficiency bottleneck in grid-based representations leveraging the regularity of warehouse layouts, and transforming collisions in routes into segment intersections in computational geometry.
- We devise an end-to-end collision awareness route planning solution suited for large-scale warehouses. It drastically reduces the time complexity of prior algorithms from $O((HW)^2)$ to $O((HW)\log(HW))$, where W and H are the width and length of a warehouse in grids.
- Experimental results on real-world datasets show that our solution can be up to $227\times$ faster than existing methods while maintaining high effectiveness.

TABLE I
SUMMARY OF IMPORTANT NOTATIONS.

Notation	Description
M	matrix representation for a warehouse
H/W	length/width of a warehouse
$g, \langle i,j \rangle$	a grid and its coordinate
o, d, t	origin,destination and emerging time of a planning request
st_r, G_r	start moving time and sequence of visiting grids of route r
Q_t/R_t	set of pairs (input)/routes (output) at time t
S, V, E	strip graph and its vertices and edges
ϕ, ψ	segments in our SRP
$\mathbf{s}/\mathbf{f}$	start/finish point of a segment
$C(\phi, \psi)$	whether two segments collide with each other
$CT(\phi, \psi)$	collision time of two segments

II. PROBLEM STATEMENT

In this section, we present the Collision-Aware Route Planning (CARP) problem in the context of robotized warehouses. In a robotized warehouse, there are racks to store items and pickers for item processing. Racks and pickers are installed at fixed positions. Robots are tasked to deliver racks with items to pickers and return racks back to the original locations after item processing. We aim to plan paths for robots moving between racks and pickers to minimize the finish time of all items. We formally define the relevant concepts and the problem below. Table I summarizes the important notations.

Definition 1 (Warehouse Matrix). *Following prior conventions [3], [4], [6], we consider a warehouse discretized into grids, which can be presented as a matrix M, where $M[i, j]$ denotes a grid $\langle i, j \rangle$ at the i-th row and the j-th column. Its value can either be "true" or "false", indicating whether there is a rack at the grid. The unit length is grid width.*

Definition 2 (Route). *A route r is represented as $\langle st_r, G_r \rangle$, where st_r is the start moving time and $G_r = \{g_1, \ldots, g_{|G_r|}\}$ is an ordered sequence of visiting grids.*

Following previous studies [2], [3], [6], we make the following assumptions on the movement of robots.
- A robot can only move along grids with no racks, *i.e.*, grids with values "false".
- A robot can only move along the directions of rows or columns at unit speed *i.e.*, one grid per second. Note that then $G_r[i] = g_i$ will be visited at timestamp $st_r + i$.

We consider the following route planning problem.

Definition 3 (Collision-Aware Route Planning (CARP) [5], [6]). *Given a warehouse matrix M and a set of origin-destination pairs $Q_t = \{\langle o_1, d_1 \rangle, \ldots, \langle o_{|Q_t|}, d_{|Q_t|} \rangle\}$ at every timestamp t, the problem is to output a set of routes $R_t = \{r_1, ..., r_{|R_t|}\}$ correspondingly, such that*

$$\min_{\forall t} \max_{\forall r \in R_t} st_r + |G_r|, \bigcup_{\forall t} R_t \text{ is collision-free} \quad (1)$$

where a origin-destination pair can be either robot-rack or rack-picker, which corresponds to the event of robot picking up
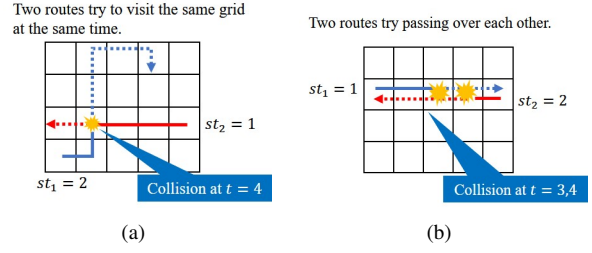

Fig. 1. Illustration of two types of collision.

racks or delivering to pickers. The goal $\min_{\forall t} \max_{\forall r \in R_t} st_r + |G_r|$ is to minimize the finish time after all routes, *i.e.*, the *makespan*, which is widely adopted in practice [2], [4]–[6]. The collision-free constraint is defined as follows [2], [6], [8]. Any two routes r_1 and r_2 should avoid the two cases below:
- Two routes visit the same grid at the same time. *i.e.*, $\exists i, j \ s.t. \ G_{r_1}[i] = G_{r_2}[j] \wedge st_{r_1} + i = st_{r_2} + j$ (Fig. 1(a)).
- Two routes try to passing over each other. *i.e.,*, $\exists i, j \ s.t. \ G_{r_1}[i] = G_{r_2}[j+1] \wedge G_{r_1}[i+1] = G_{r_2}[j] \wedge st_{r_1} + i = st_{r_2} + j$ (Fig. 1(b)).

A set of routes R is collision-free if no collision occurs between any two routes in R.

Remarks. We make two notes on the CARP problem.
- The CARP problem is often studied in the *online* setting in robotized warehouse applications, where the origin-destination pairs emerge dynamically. A practical route planning solution in this setting is expected to yield high efficiency with reasonable effectiveness [3], [5], [6]. This is because prior research has proven that it is almost impossible to achieve optimal route planning in the online setting [3]. More importantly, large-scale warehouses demand high-efficiency route planning, *e.g.*, 50 routes per second in real-world applications with a daily throughput of over $100,000$ [6].
- Existing solutions [3], [5] to the online CARP problem are low in efficiency, particularly for large-scale warehouses. They devise A*-based search algorithm [7] with various heuristics on the warehouse matrix at the *grid* level. Planning on grids results in a 3-D searching space or frequent replanning execution to avoid collision, which incurs huge delay in large-scale warehouses. For instance, the state-of-the-art method [4], takes over 60 seconds to plan only 600 origin-destination pairs. This is unacceptable if number of routes reaches $100,000$.

III. STRIP-BASED ROUTE PLANNING FRAMEWORK

This section overviews Strip-based Route Planning (SRP), an efficient solution framework for the online CARP problem.

Principles. Our key idea for high efficiency is to plan routes on *strips* (a line of grids) rather than individual grids.
- We argue that the warehouse layout is an overlooked opportunity for efficient route planning in warehouses. A real-world warehouse is usually designed with aisles and clusters of racks (see Fig. 15). Therefore, grids can
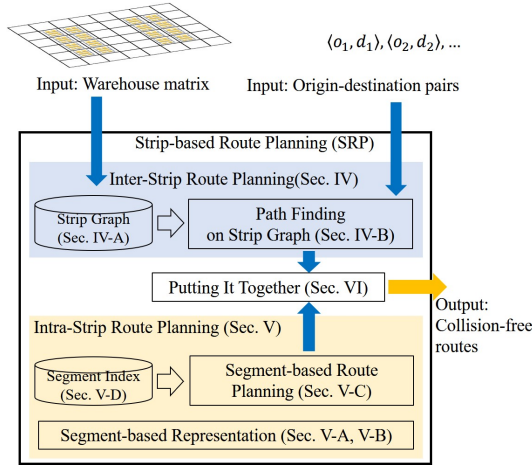
Fig. 2. Workflow of our Strip-based Route Planning framework (SRP).

be naturally aggregated into strips (rows or columns of grids) to reduce the search space in route planning.

- Concretely, we propose to plan routes (find shortest paths and detect collisions) at *inter-* and *intra-strip* levels. Such separation restricts the collision domain to within strips. That is, the time-consuming collision detection is only necessary in each strip rather than every grid in the warehouse matrix (see Sec. IV-B). We further accelerate collision detection in strips via indexing (see Sec. V-D). Theoretical analysis shows that we can reduce the time complexity of route planning from $O((HW)^2)$ to $O(HW \log(HW))$ (see Sec. VII-B).

- It is worth mentioning that shortest path finding at two levels (inter- and intra-strip) still results in effective route planning. This is because the optimal paths (with minimal makespan) are likely to take straight lines *e.g.*, aisles, which align with our strip abstraction. A proof on effectiveness of our framework is provided in Sec. VII-A.

Workflow. Fig. 2 illustrates the workflow of our SRP framework. Given a warehouse matrix, we first aggregate grids in the same row or column into strips. Then we plan routes at inter- and intra-strip levels for the input pairs of origins and destinations. Inter-strip route planning is collision-free. Thus, we view each strip as a vertex in a graph (see Fig. 3) and only perform shortest path finding on the graph. Intra-strip route planning is responsible for both shortest path finding within strips and collision detection. For efficient route planning within strips, we represent routes as *segments*, and exploit computational geometry and indexing techniques to speed up collision detection. The inter-strip module calls the intra-strip module to output the final grid-level collision-free routes.

IV. INTER-STRIP ROUTE PLANNING

This section introduces the inter-strip route planning of our SRP. Given a warehouse matrix, we construct a strip graph (Sec. IV-A), upon which a vanilla shortest path finding algorithm is executed (Sec. IV-B).
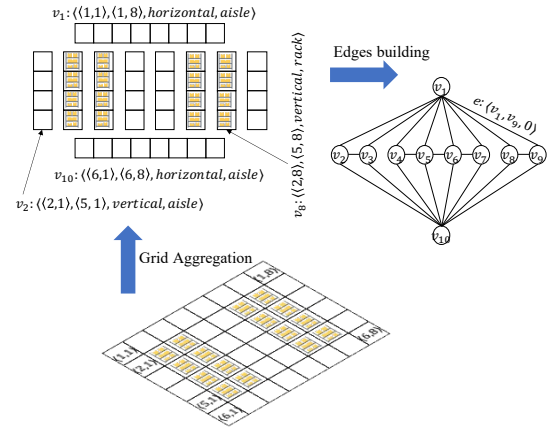

Fig. 3. A toy strip graph. We first aggregate grids in the same rows or columns into strips, then view each strip as vertex and construct strip graph. Edges exist between two adjacent strip vertices (not both are racks).

A. Strip Graph Representation for Warehouse Matrix

Definition 4 (Strip). *A strip is denoted as a vertex* $v = \langle \alpha, \beta, dir, type \rangle$, *which represents a row or column of consecutive grids with the same value (either "true" or "false") in the warehouse matrix.*

Here α_v and β_v are the westernmost and easternmost (northernmost and southernmost) grid coordinates if the direction dir_v is latitudinal (longitudinal), respectively. The type is either "ailse" or "rack". As mentioned in Sec. I, a well-designed warehouse often contains multiple aisles and clusters of racks, providing a natural opportunity for grid aggregation.

For easy route planning on grids aggregated as strips, we define the strip graph below.

Definition 5 (Strip Graph). *A strip graph S is denoted as* $\langle V, E \rangle$ *with vertices V and edges E. Each vertex $v \in V$ is a strip. Each edge $e \in E$ with weight w_e connecting u_e and n_e is denoted as* $\langle u_e, n_e, w_e \rangle$.

There exists an edge between u_e and n_e if the two strips contain adjacent grids. The edge weight indicates the time cost to move from one strip to the other, which is calculated by intra-strip route planning (see Sec. V).

Algorithm 1 illustrates how to construct a strip graph from a warehouse matrix. For vertices, we first aggregate the long latitudinal aisles as a vertex (line $4 \sim 8$), and then aggregate the remaining grids along the longitudinal direction (line $10 \sim 19$). We decide whether there is an edge between two vertices based on the coordinates of the grids they contain. Note that two vertices both have "rack" type are not adjacent because robots cannot cross over racks. The edge weights are initialized as zero (line $21 \sim 24$) which will be calculated cooperating with intra-strip level (see Sec. VI). Fig. 3 provides an example of a warehouse matrix and the corresponding strip graph.

Remarks. To simplify the grid aggregation, we assume rack clusters form the same $2 \times l$ sized rectangles, and their sides are parallel to each other. Under this assumption, most strips contain l grids, except the long aisles spanning the entire

Algorithm 1: Strip Graph Construction

Input : M: warehouse matrix
Output : S: strip graph

1 $S \leftarrow \langle \emptyset, \emptyset \rangle$
2 $visit[H, W] \leftarrow false$
3 # Aggregate grids in latitudinal direction
4 **for** $i \leftarrow 1, ..., H$ **do**
5   **if** $M[i, 1...W] = false$ **then**
6          $v \leftarrow \langle \langle i, 1 \rangle, \langle i, W \rangle, latitudinal, aisle \rangle$
7          $S.V \leftarrow S.V \cup \{v\}$
8          $visit[i, 1...W] \leftarrow true$

9 # Aggregate grids in longitudinal direction
10 **for** $i \leftarrow 1, ..., H$ **do**
11   **for** $j \leftarrow 1, ..., W$ **do**
12   **if** $visit[i, j] = false$ **then**
13             $k \leftarrow \arg\max_{k'} \{M[i...k', j] = M[i, j]\}$
14   **if** $M[i, j] = false$ **then**
15                 $v \leftarrow \langle \langle i, j \rangle, \langle k, j \rangle, longitudinal, aisle \rangle$
16   **if** $M[i, j] = true$ **then**
17                 $v \leftarrow \langle \langle i, j \rangle, \langle k, j \rangle, longitudinal, rack \rangle$
18             $S.V \leftarrow S.V \cup \{v\}$
19             $visit[i...k, j] = true$

20 # Build edges
21 **for** $v_1 \in S.V$ **do**
22   **for** $v_2 \in S.V$ **do**
23   **if** v_1, v_2 *is adjacent* **then**
24             $S.E \leftarrow S.E \cup \{\langle v_1, v_2, 0 \rangle\}$

25 **return** S

Fig. 4. Examples for segment-based route representation.

two to one, *i.e.*, along the strip direction dir_v. Accordingly, for a route within a strip, we can simplify its representation from 3-D (2-D space + 1-D time) to 2-D (1-D space + 1-D time), which also speeds up collision detection. Formally, we represent routes within strips as *segments* below.

Definition 6 (Segment). *A segment ϕ is a tuple $\langle \mathbf{s}, \mathbf{f} \rangle$, where $\mathbf{s}$ and $\mathbf{f}$ are both 2-dimensional vectors. $\mathbf{s}[0]$ and $\mathbf{s}[1]$ represent start time and grid number and $\mathbf{f}[0]$ and $\mathbf{f}[1]$ represent finish time and grid number, respectively.*

With the above definition, simple routes within a strip can be represented as follows (see Fig. 4 for an illustration).

- A route moving *forward* along the strip direction corresponds to a segment with slope 1, *e.g.*, ϕ_1 (in blue).
- A route moving *backward* along the strip direction corresponds to a segment with slope -1, *e.g.*, ϕ_2 (in black).
- As a special case, waiting time will form as a horizontal segment (with slope 0), *e.g.*, ϕ_4 (in red).

A route within a strip may contain multiple cases, resulting in segments forming polylines[1], *e.g.*, ϕ_4, ϕ_5, ϕ_6 in Fig. 4.

Remarks. The segment-based representation of routes enables easy intra-strip collision detection.

- Based on the segment representation, the collision between routes can be converted to the intersection or overlap between segments, which can be checked in constant time via computational geometry.
- Given unit moving speed [2]–[4], the slope can only be 1, -1 or 0, which simplifies collision time calculation.

B. Segment-based Collision Detection

By representing routes as segments, collisions among routes are converted into intersections among segments.

1) Collision Detection Between Two Segments: Judging an intersection and finding the intersection point of two segments are classical problems in computational geometry [10].

Two segments intersect iff two endpoints of one segment are separated by the other segment, which can be determined by the cross product of two segment vectors (see Fig. 5). Specifically, for segments ϕ and ψ, we first judge whether they

warehouse (see Fig. 3). Thus, the number of strips will roughly reduce to $\frac{1}{l}$, where l is the length of rack clusters. In fact, considering long aisles are aggregated into single vertices, the ratio is even lower (See Sec. VIII-A). This reduction accelerates the process of shortest path finding.

B. Route Planning on Strip Graph

Upon the strip graph, we can apply traditional shortest path finding algorithm such as Dijkstra's algorithm [9] without accounting for collisions. This is because the separation of inter- and intra-strip route planning restricts the collision domain to within strips. Specifically, the inter-strip route planning only calculates which aisles a route chooses to pass, rather than a concrete path *within* an aisle *i.e.*, a strip. Collision awareness of concrete paths is managed by intra-strip route planning.

V. INTRA-STRIP ROUTE PLANNING

This section presents the intra-strip route planning of SRP. The key novelty is a segment-based representation of routes within strips (Sec. V-A), which allows fast collision detection (Sec. V-B) and facilies path finding (Sec. V-C). We also propose a segment index to accelerate the collision detection of new routes within a strip (Sec. V-D).

A. Segment-based Representation of Routes within Strips

An advantage of strip-based warehouse representation is that the spatial dimension of routes within strips is reduced from
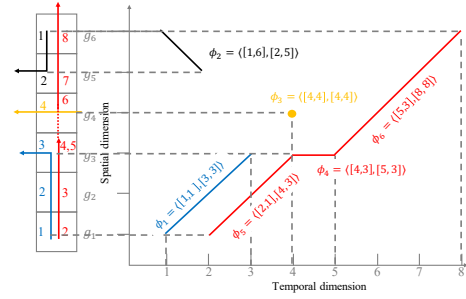
---

[1]A trivial case is that a route moving to a strip and leaving right away, forming a single point.

have overlapped ranges in one dimension. If yes, we further judge whether they intersect as below.

$$C(\phi,\psi) = ((\mathbf{s}_\phi - \mathbf{f}_\psi) \times (\mathbf{s}_\psi - \mathbf{f}_\psi))((\mathbf{f}_\phi - \mathbf{f}_\psi) \times (\mathbf{s}_\psi - \mathbf{f}_\psi)) < 0$$
$$\wedge ((\mathbf{f}_\psi - \mathbf{f}_\phi) \times (\mathbf{s}_\phi - \mathbf{f}_\phi))((\mathbf{s}_\psi - \mathbf{f}_\phi) \times (\mathbf{s}_\phi - \mathbf{f}_\phi)) < 0 \quad (2)$$

The idea is if segment ϕ and ψ intersect, then angles α_1 and α_2 rotate in different directions, which can be inferred by the cross-products of the corresponding segment vectors, and so will the two angles α_3 and α_4 (see Fig. 5).

Fig. 5. Illustration for Eq.(2).

The corresponding collision time CT is calculated as

$$CT(\phi,\psi) = \left\lfloor \frac{\mathbf{s}_\phi[0] + \mathbf{s}_\psi[0] + |\mathbf{s}_\phi[1] - \mathbf{s}_\psi[1]|}{2} \right\rfloor \quad (3)$$

Note that the segment slopes can only be 1, -1 and 0. If the slope of one segment is 0 or both segments have the same slope, the collision time can be obtained easily. So, we assume slopes of 1 and -1. Let the line equation of ϕ be $s = t + b$ and that for ψ be $s = -t + c$, where s, t are the spatial and temporal dimensions and b, c are biases. Since $\mathbf{s}_\phi[0]$ and $\mathbf{s}_\phi[1]$ are on segment ϕ, then $b = \mathbf{s}_\phi[1] - \mathbf{s}_\phi[0]$. Similarly, $c = \mathbf{s}_\psi[1] + \mathbf{s}_\psi[0]$. Since the interaction point is on both segments, we can derive the intersection time $t = \frac{c-b}{2}$ *i.e.*, $t = \frac{\mathbf{s}_\psi[1] + \mathbf{s}_\psi[0] - \mathbf{s}_\phi[1] + \mathbf{s}_\phi[0]}{2}$. If we swap the slopes of ϕ and ψ, t is $\frac{\mathbf{s}_\phi[1] + \mathbf{s}_\phi[0] - \mathbf{s}_\psi[1] + \mathbf{s}_\psi[0]}{2}$. The two cases can be unified as Eq.(3).

Fig. 6 provides an example of collision with segment-based representation. Note that for the second type of collision (see Fig. 6(b)), the floor operation in Eq.(3) is to ensure that the earlier collision time is returned, which is practical for collision detection.

2) Collision Detection of All Existing Segments: A feasible route cannot collide with *all* other segments. Therefore, we need to check collision of a new segment with all existing segments. As a feasible solution, we organize all segments in an ordered set (*e.g.*, red-black tree [11]) based on their start time $\mathbf{s}[0]$. When checking whether a new segment will collide with existing segments, we first conduct a binary search on the ordered set to find segments whose start and finish time overlap with the new segment, which indicates a potential collision. Then we judge those potential collision segments one by one using Eq.(2). If no collision occurs, we insert the new segment into the ordered set.

Remarks. This collision detection method above incurs a time complexity of $O(2 \log n + n)$, where n is the number of segments in a strip. The two log terms correspond to the binary search and insertion operation, respectively. The linear term is due to the judgement. We will further accelerate the collision detection scheme via indexing (see Sec. V-D).
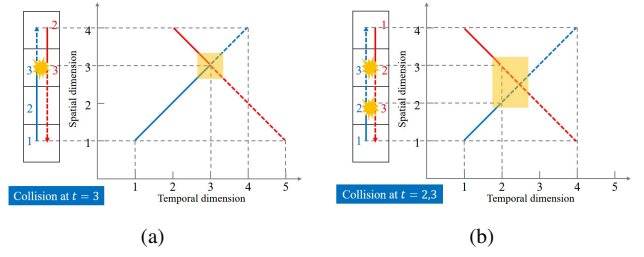

Fig. 6. Converting collisions into intersection or overlap among segments.

Algorithm 2: Segment-based Route Planning

Input : t: current time, ori: origin gird, dst: destination gird
Output : r: intra-vertex route.
1 $o, d \leftarrow$ get grid number of ori, dst
2 $segments \leftarrow$ backtracking(t, o, d)
3 **for** $segment \in segments$ **do**
4   Insert segment into ordered set
5 $r \leftarrow$ convert route from $segments$
6 **return** r
7 **backtracking**
8 $\mathbf{s} \leftarrow \langle o, t \rangle$, $\mathbf{f} \leftarrow \langle d, t + |d - o| \rangle$
9 $c \leftarrow$ Judge collision time via Eq.(3)
10 **if** $c = INF$ **then**
11   # Move forward w/o waiting cause no collision
12   **return** $\{\langle \mathbf{s}, \mathbf{f} \rangle\}$
13 **else**
14   # Move forward w/o waiting cause collision
15    $segments \leftarrow \emptyset$
16   # Try to wait and search route recursively
17   **for** $\tau = c + 1, ...$ **do**
18      $\mathbf{s} \leftarrow \langle o, t \rangle$, $\mathbf{f} \leftarrow \langle o + c - 1, t + c - 1 \rangle$
19      $segments.add(\langle \mathbf{s}, \mathbf{f} \rangle)$
20   backtracking$(t + \tau, o + c - 1, d)$
21      $segments.remove(\langle \mathbf{s}, \mathbf{f} \rangle)$
22 **return** $segments$

C. Segment-based Route Planning

To plan routes within a strip, we design a backtracking algorithm to search the shortest collision-free routes (see Algorithm 2). The idea is to store the current steps before making the next move. If collision occurs in the next move, we return to the previous step, wait one time unit and try to move again, till a collision-free route is found. Specifically, after converting the grid to the one dimensional grid number within the vertex (line 1), we begin backtracking searching (line 2). We first greedily move towards the destination (line $8 \sim 9$). If no collision occurs we directly return this segment (line $10 \sim 12$) and convert it to a route (line 5). If collision occurs, we then try to stop right before the collision occurs (line 13) and try to move forward again (line $14 \sim 21$). Note that we prohibit routes moving backward the direction from origin to destination for searching efficiency. Fig. 7 shows an example of segment-based route planning.
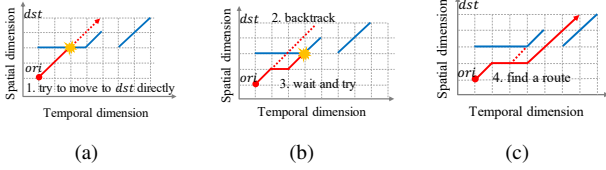
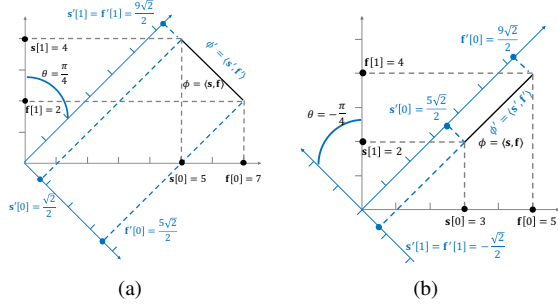Fig. 7. Illustration of collision awareness in intra-strip route planning.

Fig. 8. Illustration of coordinate rotation when θ is $\frac{\pi}{4}$ (a) or $-\frac{\pi}{4}$ (b).

D. Accelerating Collision Detection via Indexing

Recall from Sec. V-B that it takes $O(2\log n + n)$ time to check whether a candidate route collides with all existing routes, where n is the number of segments in a strip. We now accelerate the process with slope-based indexing.

Basic Idea. We can easily check whether the collision among segments with the same slope. Since the segments of the same slope are parallel to each other, a segment will collide with others only if they are at the same location in direction orthogonal to parallel direction. For horizontal segments, the orthogonal direction is the spatial dimension while for non-horizontal segments it requires an extra rotation as below.

$$\mathbf{s}' = (\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \mathbf{s})^T \qquad (4)$$

The rotation angle θ is $\frac{\pi}{4}$ for segments of slope -1, and $-\frac{\pi}{4}$ for those of slope 1. $\mathbf{s}$ is the segment endpoint. Fig. 8 shows an example of coordinate rotation.

To further exploit collision detection in case of parallel slopes, we separately store segments in different maps based on their slopes. Specifically, the key of each map is $\mathbf{s}[0]$ (or $\mathbf{s}'[0]$ after rotation if non-horizontal) and its value is an ordered set of the corresponding segments based on their start time (same as Sec. V-B to support binary search). Algorithm 3 describes the operations of insertion and collision detection.

Given a segment, we use a naive method (See Sec. V-B) to judge existing segments with different slopes. For segments with the same slope, we filter potentially intersected segment using the key of the aforementioned map and time span overlapping, and then judge intersection one by one (see examples below). By replacing line 4 and line 9 in Algorithm 2 with insertion and collision detection operation, respectively, we obtain the accelerated algorithm for intra-strip route planning.

Example. Fig. 9 illustrates how slope-based indexing acceler-

ates collision detection for a segment s of slope 0 (marked in red). The left of Fig. 9 shows the naive collision detection. We first filter existing segments (in gray) whose time spans overlap with s. In this example, s spans from $11 \sim 16$, and segments spanning from $10 \sim 12$, $10 \sim 13$ and $14 \sim 17$ overlap with s, which indicates potential intersection. Afterwards, we check the remaining segments one by one (in orange box). The right of Fig. 9 shows how we accelerate collision detection of s via slope-based indexing. We separately manage segments based on slopes. For segments with slope 0, which are parallel to s, they intersect only when they have the same spatial dimensions. In this example, only segments with spatial coordinate 13 may intersect (in orange box). We build a mapping from their spatial dimension to segments (maintained in a sorted list), then pick the segments whose spatial dimension is the same as s, and use binary search for further filtering. For segments with slope 1 or -1, the principle is the same, except an extra rotation for the mapping. For example, the spatial coordinate of the leftmost segment calculated by Eq.(4) is $4\sqrt{2}$, whose $\mathbf{s}$ and $\mathbf{f}$ are $\langle 0, 8 \rangle$ and $\langle 5, 13 \rangle$.
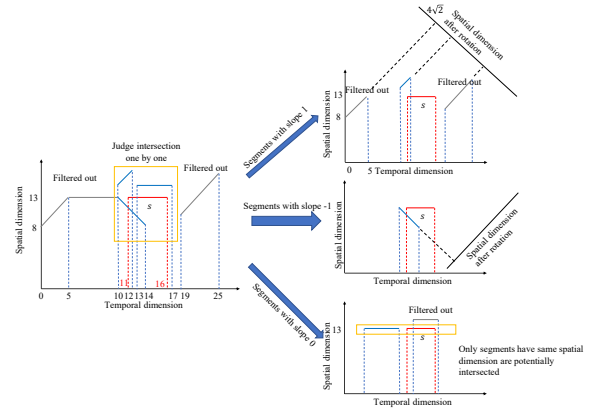

Fig. 9. Example of how naive collision detection (left) can be accelerated by slope-based indexing (right).

Remarks. The time complexity of collision detection is reduced to $O(\log m + m + \log(n - n') + n - n')$ and the insertion time complexity is reduced to $O(\log m)$, where n' is the number of segments of a certain slope and m is the number of segments with the same $\mathbf{s}'[0]$. Thus, the overall collision detection time complexity is reduced from $O(n + 2\log n)$ to $O(\log(n - n') + n - n' + m + 2\log m)$.

Moreover, the rotation of non-horizontal segments has a side-benefit. The time dimension is projected to both the rotated axes, making $\mathbf{s}'[1]$ ever-increasing. This allows two non-horizontal segments to almost always have different $\mathbf{s}'[1]$, making an almost one-to-one mapping. Consequently, m, *i.e.*, the number of segments with the same $\mathbf{s}'[0]$, is rather small, which further accelerates the overall collision detection.

VI. PUTTING IT TOGETHER

The end-to-end route planning algorithm of SRP operates in a top-down manner. The inter-strip route planning algorithm

Algorithm 3: Segment Index Operations

1 Initialize
2 $S_0, M_0 \leftarrow$ ordered set and map for 0-slope segments
3 $U_1, M_1 \leftarrow$ ordered set and map for 1-slope segments
4 $S_{-1}, M_{-1} \leftarrow$ ordered set and map for -1-slope segments
5 Insertion
   Input : $\langle \mathbf{s}, \mathbf{f} \rangle$: segment remain to be inserted
6 $k \leftarrow$ get slope of segment
7 add $\langle \mathbf{s}, \mathbf{f} \rangle$ into S_k
8 if $k \neq 0$ **then**
9 | rotate coordinate by Eq.(4)
10 | add $\langle \mathbf{s}, \mathbf{f} \rangle$ into $M_k.get(\mathbf{s}[0])$
11 Collision Judgement
   Input : $\phi = \langle \mathbf{s}, \mathbf{f} \rangle$: segment
   Output : c: earliest collision time, INF if no collision.
12 $k \leftarrow$ get slope of segment
13 if $k \neq 0$ **then**
14 | rotate coordinate by Eq.(4)
15 $c \leftarrow INF$
16 # Find all unparalleled segments
17 $S_1, S_2, \leftarrow$ set of segments whose slopes $\neq k$
18 # Find potentially overlapped segments
19 $S_1^*, S_2^* \leftarrow$ binary search from S_1, S_2 by $\mathbf{s}[0]$ and $\mathbf{f}[0]$
20 # Judge collision one by one
21 for $\psi \in M_k.get(\mathbf{s}[0]) \cup S_1^* \cup S_2^*$ **do**
22 | **if** *collide by Eq.(2)* **then**
23 | | $\tau \leftarrow$ get collision time by Eq.(3).
24 | $c \leftarrow \min(c, \tau)$
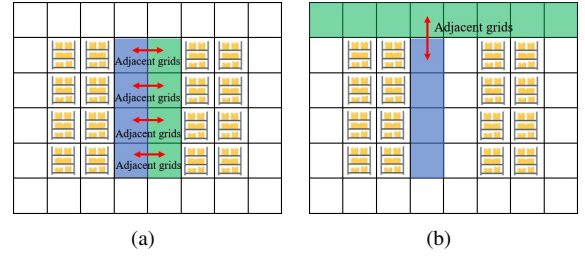**25 return** c

Fig. 10. Illustration of strip adjacency.

3), $intraRoutes[v]$ records the intra-strip level routes from $pred[v]$ to v within a strip (line 4), and q is the priority queue recording strip vertices for shortest path searching. Vertices with lower cost have higher priority to be extracted (line 8). The edge weight connects the inter- and intra-strip levels. Whenever the inter-strip level traverses an edge, intra-strip route planning is executed and provides intra-strip route duration as the edge weight for the inter-strip level (line 18).

Example. Fig. 11 shows an example to plan routes from ori grid to dst. We first find its corresponding strip vertices v_o and v_d, which are v_3 and v_9, respectively, in this example. Then we run Dijkstra's algorithm to search the path at inter-strip level. Assume a route $v_3 \rightarrow v_1 \rightarrow v_9$, is returned. The weights of each edge will be calculated by intra-strip route planning. For example, to calculate the weight of edge $\langle v_3, v_1 \rangle$, Algorithm 4 first gets the current location g (line 12), and then obtains its adjacent grid g' (line 15). Since the two grids are adjacent, the weight is 1 (see green marks and notations in Fig. 11). As for edge $\langle v_1, v_9 \rangle$, it first gets the current location from $intraRoutes$ (yellow g in Fig. 11), then finds the adjacent grid g', and assigns the edge weight to 8 (see yellow marks and notations in Fig. 11). If there are other routes, intra-strip planning may require some waiting steps to avoid collision and the edge weight will increase.
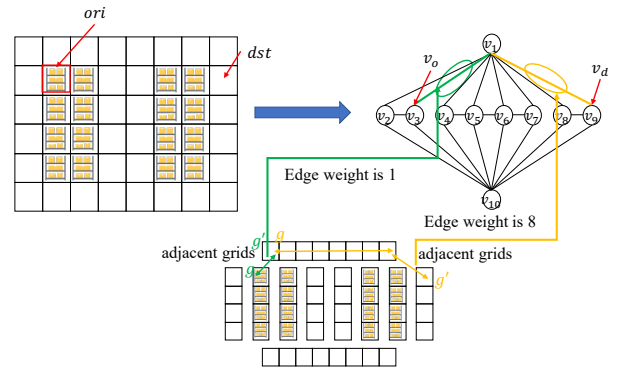
performs shortest path finding on the strip graph. As mentioned in Sec. III, there is no need to consider collision at inter-strip level because all the *concrete* routes are planned at intra-strip level. The inter-strip level only decides which strips are chosen for a route whereas how to pass a strip without collision is calculated by intra-strip route planning.

Inter-strip level calls the intra-strip route planning algorithm to retrieve the edge weights of the strip graph. The intra-strip route planning algorithm calculates the edge weights *i.e.*, the duration of routes for the two strips in two steps: *(i)* planning collision-free routes to an adjacent grid within the origin strip; and *(ii)* moving from the adjacent grid to the destination strip.

Since the directions of strips are either latitudinal or longitudinal, the adjacent grid contains only two cases.

- *Side-by-side*. Two strips have the same direction. In this case, any grids included in the origin strip is adjacent to the destination strip (see Fig. 10(a)).
- *Perpendicular*. Two strips have different directions. In this case, only one grid in the origin strip is adjacent to the destination strip (see Fig. 10(b)).

Algorithm 4 shows the overall route planning process. It is built upon the Dijkstra's algorithm for shortest path search over the strip graph, where line $10 \sim 24$ is Dijkstra's breadth-first-search and line $19 \sim 23$ is its relaxation operation [10]. Variable $cost[v]$ records the route duration from origin to a strip vertex v (line 2), $pred[v]$ records the predecessor vertex of v along the planned path at inter-strip level (line

Fig. 11. Example of edge weight calculation via intra-strip route planning.

Remarks. The high efficiency of our framework may restrict the research space. Concretely, we prohibit the route from moving backward within a strip, and constrain inter-strip movement within adjacent grids. These restrictions might lead to no feasible route, in which case we call the A* algorithm. But in practice this happens very rarely (1 out of 10^5), so this

Algorithm 4: End-to-End Route Planning in SRP

Input : t: current time, ori: origin gird, dst: destination gird, SI: segment index

Output : r: collision-free route

1 # $cost[v]$ indicates route duration from v_o to v
2 $cost[1...|SG.V|] \leftarrow \{INF, ..., INF\}$
3 $pred[1...|SG.V|] \leftarrow \{-1, ..., -1\}$
4 $intraRoutes[1...|SG.V|] \leftarrow \{\emptyset, ..., \emptyset\}$
5 # Inter-strip route planning
6 $v_o, v_d \leftarrow$ strip vertex containing ori, dst
7 # Initialize priority queue, less cost has higher priority
8 $q \leftarrow \emptyset$
9 add $(v_o, 0)$ into q
10 **while** q *not empty* **do**
11     $u \leftarrow q.extractMin()$
12     $g \leftarrow$ current location from $intraRoutes[pred[u]]$
13   # e is an edge adjacent to u
14   **for** $e = \langle u_e, v_e \rangle$ *where* $u_e = u$ **do**
15        $g' \leftarrow$ adjacent grid between u, v_e
16   # Intra-strip route planning
17        $r' \leftarrow$ Intra-vertex route planning(t, g, g')
18        $w_e \leftarrow$ duration of r'
19   **if** $w_e + cost[u] \leq cost[v_e]$ **then**
20           $cost[v_e] \leftarrow w_e + cost[u]$
21           $intraRoutes[u] \leftarrow r'$
22           $pred[v_e] \leftarrow u$
23           $q.put(v_e, cost[v_e])$

24 $r \leftarrow$ derive complete route from $intraRoutes, pred$
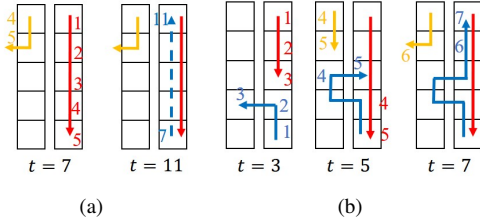25 **return** r

Fig. 12. Sub-optimality due to strip revisit omission. The numbers are the time when the routes of the same color pass the corresponding grid. The blue route is in planning route and those in other colors are existing routes.

will not influence efficiency.

VII. THEORETICAL ANALYSIS

A. Effectiveness Analysis

Since the global optimality of CARP problem in the online setting can hardly be reached and is affected by the arrival pattern of the origin-destination pairs [3], we focus on the effectiveness of a single route being planned, *i.e.*, a single call of our Algorithm 4. Specifically, we define the competitive ratio CR of a single call to be the ratio between route length of our method and optimal route length.

The sub-optimality of our method is caused by three cases.

- *Strip Revisit Omission.* The inter-strip path finding adopts shortest path finding, where each strip can only be visited at most once. However, each strip contains multiple grids and a revisit may result in to a better solution. As shown in Fig. 12, when planning the blue route, our method can only wait till t is 7 when no collision occurs. Thus,
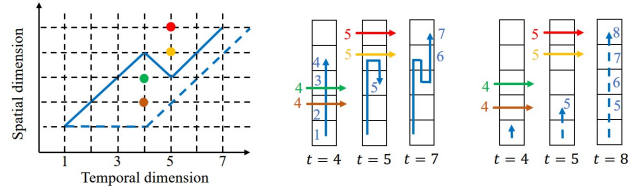

Fig. 13. Sub-optimality due to intra-strip backtracking restriction.
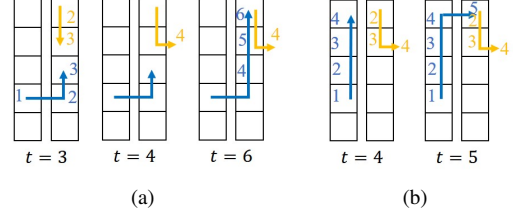

Fig. 14. Sub-optimality due to greedy inter-strip transit.

the route to destination will cost 11 time units in total. However, if we can revisit the strip, we can leave the right strip at $t = 3$ and revisit it at $t = 5$, which avoids collision with the yellow and red routes. This solution incurs only 7 time units in total.

- *Intra-Strip Backtracking Restriction.* In intra-strip backtracking, a route can only wait or move forward along the strip direction without moving backward. This restriction accelerates backtracking, but also leads to sub-optimal routes. As illustrated in Fig. 13, our backtracking algorithm can only searches the routes shown in dash lines. It waits till $t = 4$ to avoid collision with all other routes (*i.e.*, the green, brown, yellow and red ones). The optimal routes are shown as the solid lines if backward search is allowed. The route first moves towards the destination, avoids the crossing routes (green and brown ones) at $t = 4$, and then detours to avoid red and yellow crossing routes at $t = 5$. The total time cost is 7 time units.

- *Greedy Inter-Strip Transit.* When calculating the edge weight between two strips, we adopt a greed strategy that confines the transit within a pair of adjacent grids that contains the source grid. For two "side-by-side" strips, ignorance of other adjacent grid pairs may cause sub-optimality. As illustrated in Fig. 14, our method will force the transit between the two strips at $t = 2$, thus waiting the yellow routes till 4, and incurs 6 time units in total. However, if we permit transit to the right strip via other adjacent grid pairs, the optimal route will move within the left vertex till the yellow routes leave the right strip at $t = 5$, which results in 5 time units.

Theorem 1. *Given a set of routes and a pair of source and destination to be planned, the expected competitive ratio of the planned route in terms of single route length is* 1.788.

Proof. We consider the above three cases separately.

- *Strip Revisit Omission.* Revisiting a strip incurs at least two extra time units (*i.e.*, move away and back to the strip). Thus, the optimal time cost will be at least $d^* + 2$, where d^* is the length of the shortest intra-strip path.

Without revisits, the cost is $d^* + w$, where w is the waiting time. Thus, the upper bound of the competitive ratio caused by revisit omission $CR_r \leq \frac{d^*+w}{d^*+2}$.

$$\mathbb{E}[CR_r] \leq \Pr[w \leq 2] \cdot 1 + \mathbb{E}[\frac{d^*+w}{d^*+2}|w \geq 3]$$
$$\leq \Pr[w \leq 2] + \sum_{k=3} \Pr[w=k]\frac{d^*+k}{d^*+2} \quad (5)$$

A route is forced to wait because the grid right in front of it is unavailable. By assuming each grid at each time will be occupied with probability p, then $\Pr[w=k] = p^k(1-p)$. Then we have:

$$\mathbb{E}[CR_r] \leq 1 + \frac{(2p-1)p^3}{(d^*+2)(1-p)} \leq 1 + \frac{1}{3(1-p)} \quad (6)$$

As p is the probability a grid is occupied, a larger p means more congestion. Revisits may lead to a better solution.

- *Intra-Strip Backtracking Restriction.* If the optimal solution contains a route moving backward of b grids. Then, the solution will be $d^* + 2b$, where d^* is the distance from the source to destination grid. For our method, it provides a solution with cost $d^* + w$, where w is the waiting time. Thus, the competitive ratio of backtracking $CR_b \leq \frac{d^*+w}{d^*+2b}$. The same as above, we can calculate the waiting time and we have

$$\mathbb{E}[CR_b] \leq \Pr[w \leq 2b] \cdot 1 + \mathbb{E}[\frac{d^*+w}{d^*+2b}|w > 2b]$$
$$= 1 + \frac{1}{(d^*+2b)(1-p)} \leq 1 + \frac{1}{3(1-p)} \quad (7)$$

- *Greedy Inter-Strip Transit.* A transit via other adjacent grid pair adds at least one time unit cost, making CR_t upper bounded by $\frac{w}{1}$, where w is the waiting time. So,

$$\mathbb{E}[CR_t] \leq \Pr[w \leq 1] \cdot 1 + \mathbb{E}[w|w \geq 2] \leq 1 + \frac{p^2}{1-p} \quad (8)$$

To sum up, the upper bound of $\mathbb{E}[CR] = \max\{\mathbb{E}[CR_r], \mathbb{E}[CR_b], \mathbb{E}[CR_t]\} = 1 + \frac{\max\{1,3p^2\}}{3(1-p)}$. The numerator will be 1 only if $p \leq 0.577$. Since p is the probability that a grid is occupied, it is usually less than 0.5, otherwise the warehouse is too congested. Thus, $\mathbb{E}[CR] \leq 1 + \frac{1}{3(1-0.577)} \approx 1.788$. $\square$

B. Time Complexity Analysis

The inter-strip route planning adopts the Dijkstra algorithm. With the priority optimization, the time complexity is $|E|\log|V|$, where $|V|$ and $|E|$ are the numbers of strips and edges. When calculating the edge weight, the algorithm calls the intra-strip route planning which triggers backtracking (Algorithm 2). So the time complexity is $|E|T\log|V|$, where T is the time complexity of backtracking.

In our strip graph, one vertex connects with four edges, thus $|E| = 2|V|$. Since each strip contains multiple grids, thus $|V| = \frac{|HW|}{c}$, where c is the average number of grids each strip contains. As for T, we have:

$$T(t) = \begin{cases} X(n) \text{ if find routes} \\ X(n) + \sum_\tau T(t+\tau) \text{ otherwise} \end{cases} \quad (9)$$

where t is the current time and $X(n)$ is the time complexity of collision detection in Sec. V-D. Thus, $T = O(|w|^{|d|}(n' + \log n'))$, where $|d|$ is the distance between the source and destination grid and w is the waiting time. Similar to Sec. VII-A, we have $\mathbb{E}[|w|^{|d|}] = O((1-p)\sum_{k=1}^{\infty} k^d p^k)$. Therefore,

$$\mathbb{E}[|w|^{|d|}] = (1-p)\sum_{k=1}^{\infty} k^d p^{\frac{d\log k}{\log(1/p)}+1} p^{k-1-\frac{d\log k}{\log 1/p}}$$
$$\leq (1-p)\sum_{k=1}^{\infty} p^{k-1-\frac{d}{\log(1/p)}\log k} \quad (10)$$
$$\leq (1-p)\sum_{k=1}^{\infty} p^{k-1-\frac{d}{\log(1/p)}} = (\frac{1}{p})^{\frac{d}{\log(1/p)}}$$

Let $1/c$ be the ratio between the number of strips and the number of grids, *i.e.*, each strip contains $O(c)$ grids. Then we have $d = O(c)$. So, the total time complexity is $O(\frac{2}{c}(\frac{1}{p})^{\frac{c}{\log(1/p)}}|HW|\log\frac{|HW|}{c}(n' + \log n')) = O(|HW|\log|HW|(n' + \log n'))$. Because the rotation makes it nearly a one-to-one matching, the time of collision detection is almost constant. Finally, the time complexity is approximately $O(|HW|\log|HW|)$, which is lower than the A*-based algorithm with a time complexity of $O((|HW|)^2)$.

VIII. Evaluation

A. Experimental Setup

Fig. 15. Snapshot of test environment.

Datasets. We use real-world datasets for evaluations. We choose three warehouses W-1, W-2, W-3 from different cities with different throughput operated by Geekplus Technology[2], one of the world's leading smart logistic companies. For each warehouse, we extract delivery tasks with a time span of five days. Each delivery task will incur three route planning query, namely *pickup*, *transmission*, and *return*. Table II provides a summary for our datasets. The last two columns of Table II also show the scale reduction from raw grid-based representation to our strip-based representation. The numbers of vertices and edges are reduced to 16% and 23% respectively, which is aligned with the remarks in Sec. IV-A.

Test Environment. A test environment is built for algorithm performance evaluation. The test environment simulates the emergence of delivery tasks, and then sends the task information to the route planning algorithm. After receiving the results calculated by a route planning algorithm, the environment assigns those planned routes to robots for execution. The system will record all our metrics for comparison. Fig. 15 provides a snapshot of our test environment.

All planning algorithms together with the test environment are implemented in Java 8. All experiments are conducted on

[2]https://www.geekplus.com/

TABLE II
SUMMARY OF DATASETS AND THE STRIP-BASED EXTRACTION.

Name	$H \times W$	#Rack	#Robot	#Picker	#Tasks/10^3					Grid-based		Strip-based	
					Day1	Day2	Day3	Day4	Day5	#vertices	#edges	#vertices	#edges
W-1	233×104	4896	408	68	45.0	46.6	27.7	33.1	33.4	24232	48464	3997	11272
W-2	240×206	9792	952	136	41.0	45.9	34.3	79.9	63.5	49440	98880	8230	23257
W-3	292×278	15088	2208	184	34.4	35.2	26.5	134.6	103.9	81176	162352	13526	38411

(a) W1 Day1 (b) W1 Day2 (c) W1 Day3 (d) W1 Day4 (e) W1 Day5

Fig. 16. TC comparison on real dataset W-1 over all days.

(a) W2 Day1 (b) W2 Day2 (c) W2 Day3 (d) W2 Day4 (e) W2 Day5

Fig. 17. TC comparison on real dataset W-2 over all days.

(a) W3 Day1 (b) W3 Day2 (c) W3 Day3 (d) W3 Day4 (e) W3 Day5

Fig. 18. TC comparison on real dataset W-3 over all days.

Intel(R) Xeon(R) Platinum 8269CY CPU T 3.10GHz with 20 GB Java virtual machine memory.

Baselines. We compare our SRP with the following baselines.

- **Simple A*-based Planning (SAP).** It uses a simple A*-based planning algorithm by searching in a three dimensional space (2-D space and 1-D time) and planning routes one at a time. The newly planned route will avoid collisions with existing routes.
- **Replanning (RP) [3].** This algorithm adopts a replanning strategy. It first searches the shortest path for the new query ignoring collisions, and then if collision occurs, it replans all collided routes together with an offline optimal method (*e.g.*, conflict-based search [2]).
- **Time Windowed Planning (TWP) [5].** This algorithm incorporates a time window design. Instead of planning the entire routes, it confines the planning in a certain time window for acceleration.
- **Adaptive Cached Planning (ACP) [6].** This method uses a cache strategy. When the route searching is close to the destination, it will directly use the cached shortest path and simply wait till no collision will happen.

Except SAP, the other baselines (RP, TWP, and ACP) achieve competitive performances in online multi-agent path finding. Particularly, TWP achieves state-of-the-art efficiency given fewer than $1,000$ robots.

Evaluation Metrics. We use three evaluation metrics: *optimization goal* (OG), *time consumption* (TC) and *memory consumption* (MC) to evaluate both the effectiveness and efficiency of different algorithms.

- **Optimization Goal (OG).** The optimization goal of the CARP problem is defined in Eq.(1). It is also called "makespan", which is widely adopted as effectiveness evaluation for the CARP algorithm [2], [4], [5]. A smaller value indicates better effectiveness.
- **Time Consumption (TC).** It is the total time consumption of the planning algorithm executed in all rounds. A smaller value indicates higher time efficiency.
- **Memory Consumption (MC).** It records the memory consumption of data structures together with runtime space consumption during execution. A smaller value means better memory efficiency.

The unit of both OG and TC is second. Progress is the ratio between the finished tasks and all tasks of the day.
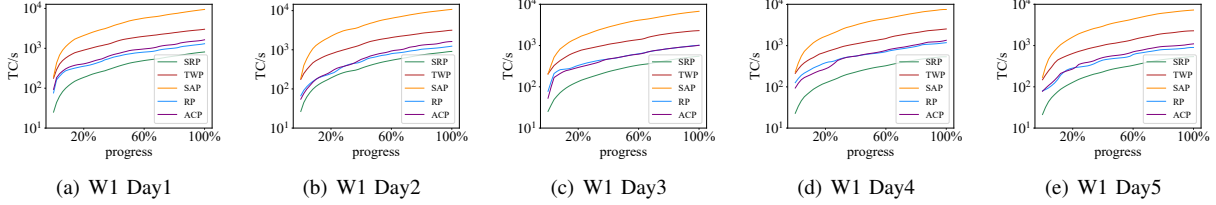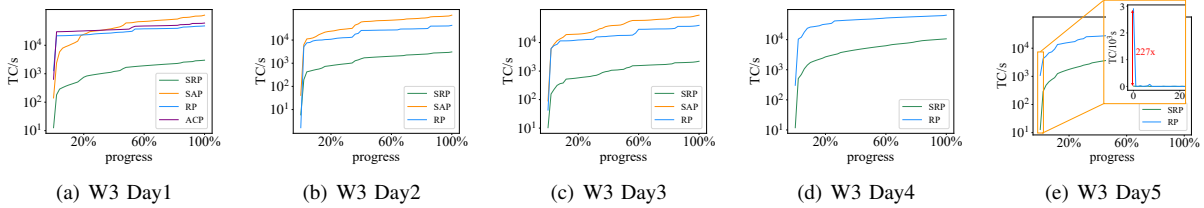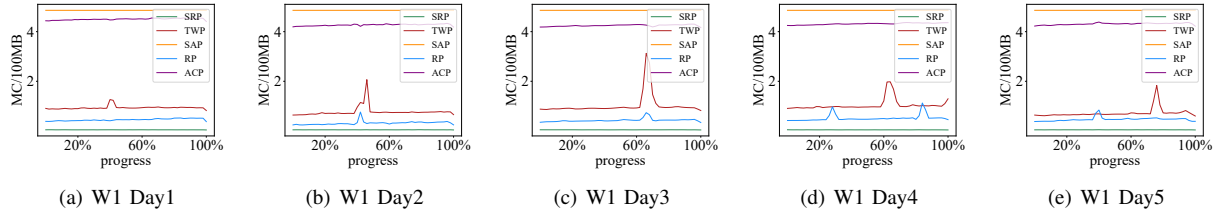
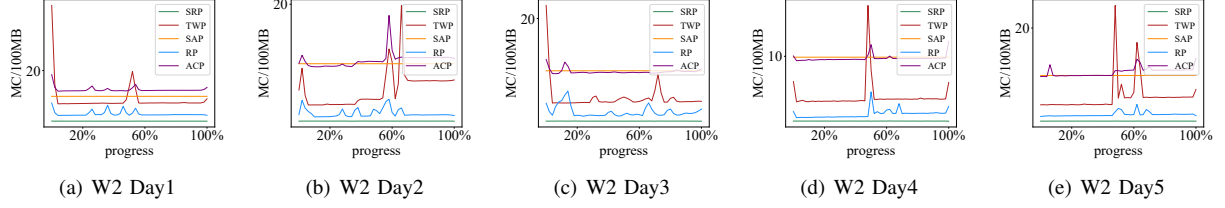Fig. 19. MC comparison on real dataset W-1 over all days.

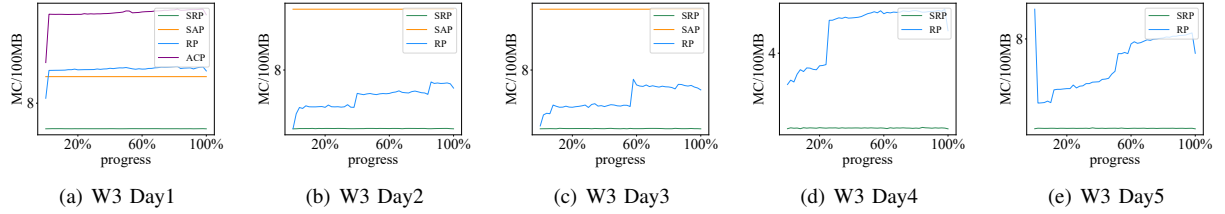Fig. 20. MC comparison on real dataset W-2 over all days.

Fig. 21. MC comparison on real dataset W-3 over all days.

B. Experimental Results

Time Efficiency Comparison. We first compare the time consumption among all baselines over all datesets. Note that the execution time consumption over some days of W-3 is too large for certain algorithms, so we omit the comparisons. From Fig. 16, Fig. 17 and Fig. 18, we can see that as picking procedure goes on, the time consumption of each algorithm steadily grows. Usually SAP is the slowest. Other algorithms such as TWP, RP and ACP show certain accelerations, but our algorithm steadily outperforms all the baselines. Concretely, our algorithm is averagely executed faster than other algorithms by 1.4x to 37.3x. If we compare the execution efficiency of a snapshot (2% of one day's procedure), the algorithm can be 227x faster than other algorithms at most. This happens at Day5 in W-3 (see Fig. 18), indicting our algorithm has a powerful ability to scalability. The time efficiency is mainly attributed to our strip-based representation, which allows fast collision detection and path finding.
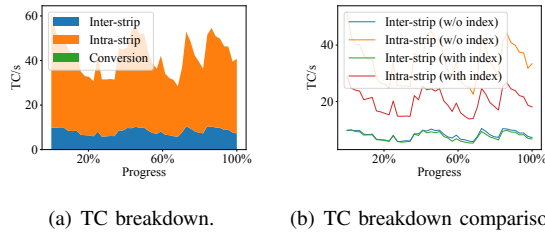

(a) TC breakdown.

(b) TC breakdown comparison.

Fig. 22. Illustration of need for slope-based indexing.

Need for Slope-based Indexing. Fig. 22(a) shows the TC breakdown of our SRP *without* slope-based indexing acceleration over a single day. The time consumption is split into three parts: inter-strip planning, intra-strip planning and conversion between strip- and grid-based representation. As is shown, the bottleneck lies in intra-strip planning, which calls for acceleration. Fig. 22(b) plots the time cost with and without slope-based indexing. With slope-based indexing, the TC of intra-strip is reduced by around 50%.

Memory Efficiency Comparison. Fig. 19, Fig. 20 and Fig. 21 illustrate the memory consumption comparisons results. In general, the memory consumption of each algorithm fluctuates as the picking process goes on. The fluctuations are caused by the dynamics of task number. We can see the spikes usually show up at the beginning or the middle, indicating the tasks floods in during morning or noon. Our SRP takes up smallest memory. Most of our memory consumption is less than 1% of the other algorithms. Specifically, SRP memory cost is 97% to 99% less than the other algorithms. This is reasonable because our strip-based representation uses only a few of segment end points to store a routes while other algorithms have to use a sequence of locations.

Effectiveness Comparison.

TABLE III
EFFECTIVENESS COMPARISONS.

Name	SAP	RP	TWP	ACP	SRP
W-1	43341	42983	43207	43282	43339
W-2	32200	32522	36958	33904	32090
W-3	41169	49809	42508	44799	34255

Table III shows the average OG over days. We use the mean value to fill blanks for the algorithms that are too slow to calculate. Our algorithm performs the best on W-2 and W-3 and slightly under-performs on W-1. However, this tiny gap is only 4 minutes and compared with a day's horizon, it can be ignored in real applications. Overall, our algorithm is reasonably effective with drastic acceleration.

Summary of Experimental Results. Our experimental results can be summarized as below.

- With the novel framework, our algorithm is significantly accelerated (at most 227 × faster).
- Due to the strip-based representation, our algorithm takes up much less memory, only 1% ~3% to that of others.
- Even with such notable acceleration, our algorithm outputs reasonably effective routes in terms of makespan.

IX. Related work

A. Route Planning

The route planning processing problem has been extensively studied in spatiotemporal research community [12]–[21].

Some research lies interests on indoor space route planning [12]–[14]. These research notices that indoor space has its own topological property (*e.g.*, two points at adjacent rooms may have a short euclidean distance, but it may require many detours to reach each other). This topological property makes indoor space more complicated and requires design for representation. [12], [22] propose representations for indoor space while other studies [13], [14] extend shortest path query as well as other spatial queries such as distance join [23] into indoor space. These solutions to indoor spaces cannot be applied mainly because the collision constraints. However, our warehouse scenario confronts with similar situation that it also requires designing new spatial representation and furthermore exploits the regular property for planning acceleration.

Other research studies route planning on road networks [15]–[18], [24]. Some studies [17], [18], [25], [26] aim at planning routes more flexible in terms of passing sequence of points of interests. Other research studies route planning under different contextual information [27]–[29] or temporal information [30], [31]. [19] aims at finding location that minimizes summation of all group members' travel distances. [15], [24] study shared mobility scenario, in which insertion operation is efficient bottleneck and some dynamic programming or index based methods are proposed for acceleration. [16] proposes a data driven method for better performance.

Though these methods using multiple manners such as indexing and dynamic programming for acceleration, they can not be adapted to our problem, mainly because the spatial layout construction upon road network is much different from that of warehouses, which leads to hardness of collision modeling and judgement.

B. Multi-Agent Path Finding

Multi-Agent Path Finding problem emerges when robotized warehouses boom [6]. Formally, it requires planning paths for multiple origin-destination pairs on a graph (usually a grid graph) to minimize the total or maximum path length while ensuring no collision happens if robots moving along these paths [8]. Though proved to be NP-hard [32], traditional methods managed to find the optimal solution using A* based searching algorithm [2]. Recent studies propose online variations of multi-agent path finding making it closer to real situation. The online situation together with hardness of reaching optimality([3]) make current works stress more on efficiency. Existing methods [3]–[5] to online multi-agent path finding mainly adopt the re-planning strategy and some heuristic tricks to branch the searching space. These tricks help A*-based search algorithms scale up to 600 pairs of origin and destination but the time consumption is over 1 minute [4]. A recent study simply uses cache for acceleration [6].

Though these methods all motivated by warehouse scenario, none of them exploits the regular layout property of warehouse. Ignoring the regular property prohibits these method from reconstructing the spatial layout. Pure grid based model forces existing methods use A* algorithm [7] to find the collision-free paths. Searching in 3-D spatiotemporal space to avoid collision results in low efficiency.

X. Conclusion

In this paper we explored high-efficiency solutions to the Collision Aware Route Planning (CARP) problem in large-scale warehouses. We observe that the regular layout of warehouses is an overlooked opportunity for acceleration. In response, we propose a strip-based framework to replace the widely adopted grid-based warehouse representation. Specifically, we aggregate grids in the same rows or columns into strips and decompose the collision aware route planning into inter- and intra-strip route planning. The decomposition makes collision judgement necessary only at intra-strip level. The strip-based representation also transforms collision judgement in 3-dimensional routes into intersection judgement among 2-dimensional segments. We further design an indexing strategy of segments within strips for acceleration. Our solution reduces the time complexity from $O((HW)^2)$ to $O((HW)\log(HW))$, in a warehouse of $H \times W$ grids. Experiments on real dataset show significant acceleration over existing schemes, which holds potentials for deployments in real-world robotized warehouses and similar applications.

REFERENCES

[1] R. Bogue, "Growth in e-commerce boosts innovation in the warehouse robot market," *Ind. Robot*, vol. 43, no. 6, pp. 583–587, 2016.

[2] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[3] J. Švancara, M. Vlk, R. Stern, D. Atzmon, and R. Barták, "Online multi-agent pathfinding," in *AAAI*, vol. 33, no. 01, 2019, pp. 7732–7739.

[4] J. Li, W. Ruml, and S. Koenig, "eecbs: A bounded-suboptimal search for multi-agent path finding," in *AAAI*, 2021, pp. 12 353–12 362.

[5] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses," in *AAAI*, 2021, pp. 11 272–11 281.

[6] D. Shi, Y. Tong, Z. Zhou, K. Xu, W. Tan, and H. Li, "Adaptive task planning for large-scale robotized warehouses," in *ICDE*, 2022.

[7] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.

[8] R. Stern, N. R. Sturtevant, A. Felner, and et al, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *SOCS*, 2019, pp. 151–159.

[9] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959. [Online]. Available: https://doi.org/10.1007/BF01386390

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[11] L. J. Guibas and R. Sedgewick, "A dichromatic framework for balanced trees," in *FOCS*, 1978, pp. 8–21.

[12] H. Lu, X. Cao, and C. S. Jensen, "A foundation for efficient indoor distance-aware query processing," in *ICDE*, 2012, pp. 438–449.

[13] X. Xie, H. Lu, and T. B. Pedersen, "Efficient distance-aware query evaluation on indoor moving objects," in *ICDE*, 2013, pp. 434–445.

[14] T. Liu, Z. Feng, H. Li, H. Lu, M. A. Cheema, H. Cheng, and J. Xu, "Shortest path queries for indoor venues with temporal variations," in *ICDE*, 2020, pp. 2014–2017.

[15] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, "A unified approach to route planning for shared mobility," *PVLDB*, vol. 11, no. 11, pp. 1633–1646, 2018.

[16] J. Wang, P. Cheng, L. Zheng, C. Feng, L. Chen, X. Lin, and Z. Wang, "Demand-aware route planning for shared mobility services," *PVLDB*, vol. 13, no. 7, pp. 979–991, 2020.

[17] M. Sharifzadeh, M. R. Kolahdouzan, and C. Shahabi, "The optimal sequenced route query," *VLDBJ*, vol. 17, no. 4, pp. 765–787, 2008.

[18] H. Liu, C. Jin, B. Yang, and A. Zhou, "Finding top-k optimal sequenced routes," in *ICDE*, 2018, pp. 569–580.

[19] T. Hashem, T. Hashem, M. E. Ali, and L. Kulik, "Group trip planning queries in spatial databases," in *SSTD*, vol. 8098. Springer, 2013, pp. 259–276.

[20] X. Li, W. Luo, M. Yuan, J. Wang, J. Lu, J. Wang, J. Lü, and J. Zeng, "Learning to optimize industry-scale dynamic pickup and delivery problems," in *ICDE*, 2021, pp. 2511–2522.

[21] H. Cheng, S. Wei, L. Zhang, Z. Zhou, and Y. Tong, "Engaging drivers in ride hailing via competition: A case study with arena," in *MDM*. IEEE, 2021, pp. 19–28.

[22] Z. Shao, M. A. Cheema, D. Taniar, and H. Lu, "Vip-tree: An effective index for indoor spatial queries," *PVLDB*, vol. 10, no. 4, pp. 325–336, 2016.

[23] X. Xie, H. Lu, and T. B. Pedersen, "Distance-aware join for indoor moving objects," *TKDE*, vol. 27, no. 2, pp. 428–442, 2015.

[24] Y. Zeng, Y. Tong, Y. Song, and L. Chen, "The simpler the better: An indexing approach for shared-route planning queries," *PVLDB*, vol. 13, no. 13, pp. 3517–3530, 2020.

[25] X. Cao, L. Chen, G. Cong, and X. Xiao, "Keyword-aware optimal route search," *PVLDB*, vol. 5, no. 11, pp. 1136–1147, 2012.

[26] J. Li, Y. D. Yang, and N. Mamoulis, "Optimal route queries with arbitrary order constraints," *TKDE*, vol. 25, no. 5, pp. 1097–1110, 2013.

[27] G. Skoumas, K. A. Schmid, G. Jossé, M. Schubert, M. A. Nascimento, A. Züfle, M. Renz, and D. Pfoser, "Knowledge-enriched route computation," in *SSTD*, vol. 9239. Springer, 2015, pp. 157–176.

[28] X. Fei, O. Gkountouna, D. Pfoser, and A. Züfle, "Spatiotemporal bus route profiling using odometer data," in *SIGSPATIAL*, 2019, pp. 369–378.

[29] B. Zheng, H. Su, W. Hua, K. Zheng, X. Zhou, and G. Li, "Efficient clue-based route search on road networks," *TKDE*, vol. 29, no. 9, pp. 1846–1859, 2017.

[30] S. Wang, W. Lin, Y. Yang, X. Xiao, and S. Zhou, "Efficient route planning on public transportation networks: A labelling approach," in *SIGMOD*, 2015, pp. 967–982.

[31] L. Li, J. Kim, J. Xu, and X. Zhou, "Time-dependent route scheduling on road networks," *SIGSPATIAL*, vol. 10, no. 1, pp. 10–14, 2018.

[32] P. Surynek, "An optimization variant of multi-robot path planning is intractable," in *AAAI*, 2010.