Adaptive Task Planning for Large-Scale Robotized Warehouses

Dingyuan Shi*, Yongxin Tong*, Zimu Zhou†, Ke Xu*, Wenzhe Tan‡, Hongbo Li‡
* *SKLSDE Lab, BDBC and IRI, Beihang University, Beijing, China*
† *Singapore Management University*
‡ *Geekplus*
*{chnsdy, yxtong, kexu}@buaa.edu.cn
†zimuzhou@smu.edu.sg ‡{wenzhe.tan, jason.li}@geekplus.com

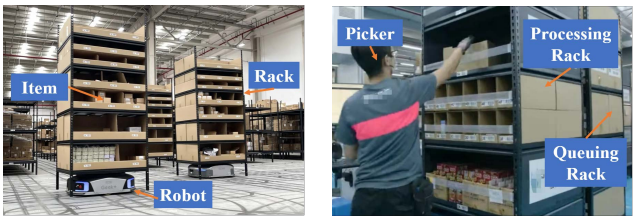
***Abstract*—Robotized warehouses are deployed to automatically distribute millions of items brought by the massive logistic orders from e-commerce. A key to automated item distribution is to plan paths for robots, also known as task planning, where each task is to deliver racks with items to pickers for processing and then return the rack back. Prior solutions are unfit for large-scale robotized warehouses due to the inflexibility to time-varying item arrivals and the low efficiency for high throughput. In this paper, we propose a new task planning problem called TPRW, which aims to minimize the end-to-end makespan that incorporates the entire item distribution pipeline, known as a fulfilment cycle. Direct extensions from state-of-the-art path finding methods are ineffective to solve the TPRW problem because they fail to adapt to the bottleneck variations of fulfillment cycles. In response, we propose Efficient Adaptive Task Planning, a framework for large-scale robotized warehouses with time-varying item arrivals. It adaptively selects racks to fulfill at each timestamp via reinforcement learning, accounting for the time-varying bottleneck of the fulfillment cycles. Then it finds paths for robots to transport the selected racks. The framework adopts a series of efficient optimizations on both time and memory to handle large-scale item throughput. Evaluations on both synthesized and real data show an improvement of 37.1% in effectiveness and 75.5% in efficiency over the state-of-the-arts.**

I. Introduction

The boom of e-commerce has stimulated enormous logistic demands. Over 2 billion logistic orders (worth over 115 billion dollars) were created during the online shopping carnival of 2020 in China[1]. Such huge amounts of orders often emerge dynamically over time. For example, there can be a sharp surge within a short time when the carnival begins at midnight. The massive, time-varying arrival of orders in unit time *i.e.*, throughput, urges highly efficient and effective operations of the warehouses that store and distribute the corresponding items to buyers [1].

Robotized warehouses are expected to improve the effectiveness and efficiency of warehouse operations by automating the fulfillment cycle of item distributions [2]. In these warehouses, multi-robot systems are installed for item distribution. Of our particular interest is the rack-to-picker mode, a popular robotized warehouse operational mode where robots pick up and deliver racks containing items from the storage area to pickers in the human picker area for processing (see Fig. 1). In this mode, a fulfill cycle for item distribution consists of five steps: rack pickup, delivery, queuing, processing, and return (see Fig. 2). From the algorithmic perspective, a central problem is to plan tasks (*i.e.*, items) for these robots, *i.e.*, determine racks (containing tasks) to fulfill, and plan paths for robots to complete fulfill cycles at each timestamp.

Such task planning problems have been extensively studied in the context of multi-agent path finding [3]–[9]. These multi-agent path finding algorithms search conflict-free paths for multiple agents (*i.e.*, no robot will collision with each other), typically with the objective to find the shortest paths [3], [4], or to minimize the makespan [5]–[9], *i.e.*, the total delay to fulfill all items [10]. Most research efforts perform *offline* task planning assuming the arrival of items is known a prior [3]–



(a) (b)

Fig. 1. A snapshot of a robotized warehouse showing entities in (a) the storage area and (b) the processing area.

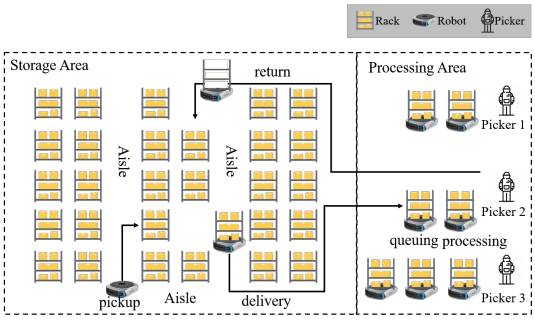


Fig. 2. The 2D layout of a rack-to-picker warehouse, where racks are shipped back and forth between storage area and picking area. Pickers located at picking area processing items (tasks) on the racks. A complete fulfilling cycle contains five steps: pickup, delivery, queuing, processing and return.

[1]https://www.cnbc.com/2020/11/12/singles-day-2020-alibaba-and-jd-rack-up-record-115-billion-of-sales.html

[6]. A few [7]–[9] investigate the more realistic *online* task planning problem, where tasks come continually as times goes on. We also focus on online task planning that minimizes the makespan. Yet we argue that prior studies [7]–[9] are unfit for online task planning in large-scale robotized warehouses due to the following limitations.

- *Limitation 1: inflexible planning to time-varying item arrival.* As mentioned, a fulfillment cycle contains multiple stages. Previous studies [7]–[9] assume a fixed makespan bottleneck, *e.g.*, delivery, which is reasonable with low, constant throughput. This assumption breaks with high, varying throughput, where the makespan bottleneck may turn into queuing or processing. It is ineffective to apply a time-invariant planning strategy to cope with the dynamic makespan bottleneck.
- *Limitation 2: inefficient planning for massive robots and items.* Many path finding algorithms [7]–[9] adopt A* search [11] from source to destination *completely*, often resulting in a time complexity of $O(I(HW)^2)$, where I, H, W is the number of items, height and width of warehouse. However, modern warehouses for e-commerce are confronted with million-scale processing workload and thousands of robots[2]. With over 10^6 items flushing in, the total time complexity will be up to 10^{14}, which is unacceptable for execution in practice. Therefore, more efficient planning algorithms are compulsory.

In response, we take a holistic problem formulation. First, we define an end-to-end makespan incorporating the entire fulfillment cycle. Such a formulation captures the bottleneck changes in fulfillment cycles due to the time-varying item arrivals. Then we propose Efficient Adaptive Task Planning (EATP), an effective and efficient task planning framework for large-scale robotized warehouses with time-varying item arrivals. Instead of start fulfilling once items emerge on racks, EATP adopts reinforcement learning to adaptively select racks to fulfill at each timestamp according to the current throughput, where the makespan may be dominated by the delay of rack transport (pickup, delivery, and return), processing, or queuing. It also incorporates a series of efficient designs such as flip requesting side, conflict detection table and cache aiding for both time and memory consumption reduction in terms of rack selection and path finding. Evaluations on both synthesized and real data show an improvement of 37.1% in effectiveness and 75.5% in efficiency over the state-of-the-art online multi-agent path finding scheme [7].

Our contributions are summarized as follows.

- We formulate the Task Planning in Robotized Warehouse (TPRW) problem, which aims to minimize the end-to-end makespan and highlights the challenges with massive, time-varying item arrivals.
- We design Efficient Adaptive Task Planning (EATP), an effective and efficient task planning framework to solve the TPRW problem. It offers adaptability to item arrivals with reinforcement learning based rack selection, and adopts a series of optimizations for fast and scalable multi-agent path finding.
- We conduct experiments on both synthesized and real datasets. The results demonstrate notable gains over the state-of-the-art [7] in both effectiveness and efficiency.

[2]https://www.dhl.com/nl-en/home/press/press-archive/2019/dhl-opens-largest-and-greenest-e-commerce-sorting-center-for-the-dutch-market.html

The rest of this paper is organized as follows. We first formulate our problem in Sec. II and propose a naive task planning algorithm in Sec. III. We then provide an overview of our efficient adaptive task planning method in Sec. IV, and elaborate on its learning based rack selection and robot path finding in Sec. V. The efficient design are detailed in Sec. VI. We present the evaluations in Sec. VII, review related work in Sec. VIII, and finally conclude in Sec. IX.

II. Problem Statement

In this section, we define the Task Planning in Robotized Warehouses (TPRW) problem. We formulate the problem in the context of the rack-to-picker mode, a prevailing operational mode in robotized warehouses [12]. In this operation mode, *robots* ship *racks* containing multiple items from the storage area to *pickers* in the processing area. As in prior studies [7]–[9], we partition the warehouse into grids whose side length is the same as a robot's side length (about 1 meter). The grid partition is reasonable because the layout of a warehouse is often regular. We build grid index for the warehouse. Next, we formally define racks, pickers, and robots.

Definition 1 (Rack)**.** *A rack r is represented as $\langle l_r, \tau_r, p_r \rangle$, which locates at l_r in the storage area and is associated with picker p_r. τ_r is the set of items' processing time unit consumption on r to be delivered to p_r.*

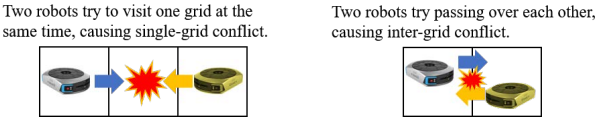
In the rack-to-picker mode, each rack is associated with a fixed picker. This is because certain pickers and racks may be dedicated to serve items destined to specific cities. Item processing time usage set τ_r specifies each item's processing time at the picker. Items arrive and processed in an online manner thus elements in τ_r emerge and disappear as time goes on. We use R to denote the set of all racks.

Definition 2 (Picker)**.** *A picker p is represented as $\langle l_p, q_p, e_p \rangle$, where l_p is its fixed location and e_p is the estimated remaining processing time of the currently picked item. q_p is the queue of racks waiting to be processed.*

We assume a picker processes the items on the racks in the queue in the "first-come-first-serve" manner. This is reasonable since the picking area is often confined, making it difficult for robots that carry racks to cut in line. We use P to denote the set of all pickers.

Definition 3 (Robot)**.** *A robot a is represented as $\langle l_a, s_a \rangle$ with location l_a and state s_a.*

Since the robot is mobile, its location and state change over time. The state s_a can be either busy or idle. A robot is busy if it is in the stage of rack pickup, delivery, queuing, processing, or return. We use A to denote the set of all idle robots.



(a) Single-grid (b) Inter-grid

Fig. 3. Examples of conflicts in path planning.

Now we define the objectives of our task planning problem. In short, we aim to minimize the *makespan*.

Definition 4 (Makespan)**.** *The makespan M is the time from the emergence of the first item till the return of the last rack.*

Assuming the first item appears at time 0, M equals to the time when the last rack is returned:

$$M = \max_{r \in R} f_r \tag{1}$$

where f_r denotes the latest time at which rack r is returned. It can be calculated as follows.

$$\begin{aligned} f_r = {} & t_k + d(l_a, l_r) + d(l_r, l_{p_r}) \\ & + \max\{d(l_a, l_r) + d(l_r, l_{p_r}) - f_p, 0\} + \sum_{i \in \tau_r} i + d(l_{p_r}, l_r) \end{aligned} \tag{2}$$

where t_k is the last time the rack is selected. The remaining five terms correspond to the delays for rack pickup, delivery, queuing, processing, and return, respectively. $d(\cdot, \cdot)$ is the path length between two locations. Assuming that robots move at unit velocity, $d(\cdot, \cdot)$ equals to the delay. f_p is the delay of picker p to process the items on all the racks in queue, which can be computed as follows.

$$f_p = e_p + \sum_{r \in q_p} \sum_{i \in \tau_r} i \tag{3}$$

Makespan is a widely used metric in prior studies [3], [4], [7], [8], but we redefine it in an end-to-end manner. From Eq.(1) to Eq.(3), the makespan is determined after deciding planning schemes $U = \{U_t\}$ for the robots at each timestamp t, where U_t is the planning scheme generated at timestamp t. Finally, we define the problem below.

Definition 5 (Task Planning in Robotized Warehouses (TPRW))**.** *Given sets of racks R, idle robots A and pickers P at every timestamp, the problem is to generate planning schemes $U_t = \{u_a | s_a = idle\}$ correspondingly at each timestamp in which element u_a is a path for an idle robot a, such that*

$$\min M$$
$$\{U_t | \forall t\} \textit{ is sufficient and conflict-free}$$

The planning schemes $\{U_t|\forall t\}$ are sufficient if all racks are assigned to a robot after the arrival of its items, *i.e.*, $\forall i_t \in I_R$, $\exists t' \geq t$ s.t. $\exists a s.t. u_a \in U_{t'}$, where item i_t emerges at time t and I_R is the item set of rack set R.

The planning schemes are conflict-free if there is neither *single-grid* conflict nor *inter-grid* conflict among all paths.

TABLE I
SUMMARY OF IMPORTANT NOTATIONS.

Notation	Description
t	current timestamp
R, P, A	set of racks, pickers, idle robots
l_r, τ_r, p_r	location, item processing time usage set, corresponding picker of rack r
l_p, q_p, e_p	location, rack queue and estimated remaining time of current item of picker p
l_a / s_a	location/state of robot a
M	makespan of all tasks
$d(\cdot, \cdot)$	distance between two locations
$\mathcal{A}$	designed algorithm
f_r / f_p	finish time of rack r/picker p
U_t, u_a	path planning scheme at t and a single path for robot a
S_t	rack selection scheme at timestamp t
H/W	height/width of the warehouse
k, ξ	number of tasks, processing time of each task in Sec. III-B's example
o_i / v_j	one task for p_1/p_2 in Sec. III-B's example.
D / D_j	summation of pickup, delivery and return time of all o_i/each v_j in Sec. III-B's example
M	time usage of moving between p_1's rack and p_2's first rack in Sec. III-B's example
$s, \alpha, \gamma, c, \beta, \epsilon$	state, action, discount factor, reward, learning rate and policy derivation parameter
ap_r / ar_r	picker p_r/rack r's accumulative processing time
δ, K, L	parameters of bootstrap, robot requesting and distance threshold

The two conflicts are defined below (see Fig. 3).

- *Single-grid conflict.* Two paths visit the same location at the same time.
- *Inter-grid conflict.* For two paths u_1 and u_2, there exist $\langle t_i, x_i, y_i \rangle$, $\langle t_{i+1}, x_{i+1}, y_{i+1} \rangle$ in u_1 and $\langle t_j, x_j, y_j \rangle$, $\langle t_{j+1}, x_{j+1}, y_{j+1} \rangle$ in u_2 such that $t_i = t_j \wedge t_{i+1} = t_{j+1} \wedge x_i = x_{j+1} \wedge y_i = y_{j+1} \wedge x_{i+1} = x_j \wedge y_{i+1} = y_j$.

Table I summarizes the important notations.

Remarks. The TPRW problem is a variant of the online multi-agent path finding problem [7] with an end-to-end makespan definition that accounts for the entire fulfillment cycle (pickup, delivery, queuing, processing, and return). The TPRW problem is challenging for large-scale warehouses with highly varied item throughput (*i.e.*, amount of item arrivals in unit time) due to the complex composition of the makespan.

- With low item throughput, the makespan is dominated by the rack transport delay (pickup, delivery, and return) [1], as is optimized by mainstream online multi-agent path finding literature [7]–[9].
- With high item throughput, large queues may build up at pickers, turning the queuing time the bottleneck.

In Sec. VII-C we will show that the bottleneck changes under different throughput. The state-of-the-art solution [7] fails to adapt to such bottleneck changes in the makespan, while other studies [8], [9] assume tasks and robots emerge in a binding manner, which is unfit for our problem. Neither do they offer efficient design for warehouses with hundreds of

Algorithm 1: Naive Task Planning

Input : t: timestamp, P: pickers, A: robots, R: racks
Output: U_t: planning scheme at t

$U_t \leftarrow \emptyset$
Sort P in ascending order based on finishing time f_p
for $p \in P$ **do**
  $R \leftarrow \{r | \tau_r \neq \emptyset \wedge p_r = p\}$
  for $r \in R$ **do**
    find the closet idle robot a from A
    $u_a \leftarrow$ plan path for robot avia A* algorithm
    $U_t \leftarrow U_t \cup \{u_a\}$
return U_t

robots processing thousands or even millions of items daily. Next we will show the extension of state-of-the-art solution to our problem and how it may lead to bad results.

III. Naive Task Planning

In this section, we adapt the state-of-the-art online multi-agent path finding algorithm [7] to solve the TPRW problem (Sec. III-A), and analyze why it is ineffective (Sec. III-B).

A. Extension from State-of-the-Art

The state-of-the-art online multi-agent path finding algorithm [7] plans conflict-free paths for each robot one at a time following certain order. The order of planning is decided by the distance from robots to its closest rack. That is, this algorithm greedily plans paths for robots with the least pickup time.

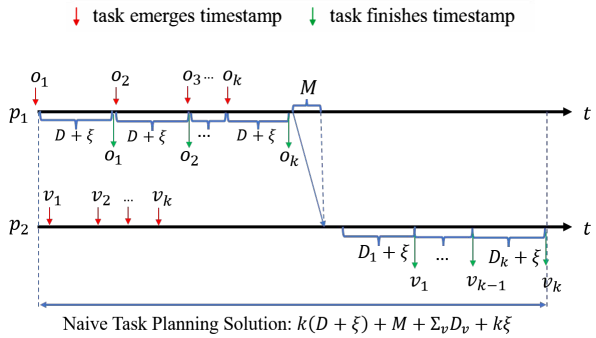
This algorithm is inapplicable to the TPRW problem since it only accounts for pickup and delivery time. We extend the algorithm to our problem as follows. Instead of planning paths for robots with the least pickup time, we plan paths for robots associated with the most slack picker. Specifically, the most slack picker p has the smallest finish time f_p as in Eq.(3). This is because a slack picker indicates a smaller queuing time. Together with optimization on rack transport time, the algorithm tend to minimize the makespan defined in Eq.(1).

Algorithm 1 illustrates the naive path planning algorithm. It first finds all pickers whose corresponding racks that still have items for processing. Then, it sorts pickers based on their degree of slack (*i.e.*, the current finish time). For each picker, it finds the corresponding racks that require processing. Finally, it choose the closet idle robot and finds a path via A* algorithm, a classical yet prevailing algorithm in multi-agent path finding studies [3], [7]–[9] (detailed in Sec. V-C).

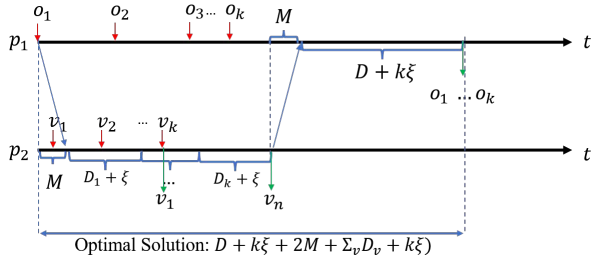
B. Limitations of Naive Task Planning

We illustrate the limitations of the naive task planning algorithm via the following example.

Consider two pickers p_1, p_2 and one robot a. For p_1, there is only one associated rack r. For p_2, k racks on which tasks will emerge. Robot a's initial location is the same as rack r (right under the rack). Both p_1 and p_2 have k items (tasks) to be processed. For simplicity, we assume all items of p_1 and p_2 have the same processing time ξ.



(a) Naive



(b) Optimal

Fig. 4. A bad case for the naive path planning algorithm.

Picker p_1's all k tasks are $o_1, o_2, ...o_k$ which appear on this rack with the same time interval $D+\xi$, where D is the sum of pickup, delivery and return time from r to p_1. For p_2, there are k tasks $v_1, v_2, ...v_k$ and each belongs to a different rack. Let D_j be the sum of pickup, delivery and return time for item v_j. These k items emerge in an online manner and the span between adjacent items v_i and v_{i+1} is shorter than all D_i. The emerging time of v_1 is later than o_1.

In this case, the greedy algorithm will first move rack r to p_1 right after the emergence of item o_1. Then when the rack is returned, o_2 just shows up and the robot will deliver the rack to p_1 again. The cycle repeats until o_k is finished. Meanwhile, all the items of p_2 have emerged. The robot then moves from r to the rack of v_1, taking M time units. Then it delivers all racks of p_2 one by one, which takes $\sum_v D_v + k\xi$ time units. The makespan is $k(D+\xi)+M+\sum_v D_v+k\xi$, as shown in Fig. 4(a). The optimal solution, however, will not greedily deliver racks of p_1. It will first deliver all racks of p_2. Meanwhile, all items of p_1 will emerge, and it delivers rack r to p_1 only once. The makespan is $D+k\xi+2M+\sum_v D_v+k\xi$, as shown in Fig. 4(b). Thus, the competitive ratio will be $O(\frac{k(D+\xi)+\sum_v D_v+k\xi}{D+k\xi+2M+\sum_v D_v+k\xi})$. With sufficiently large D, the bound is approximately $O(k)$, which is not constant.

The ineffectiveness of the naive path planning algorithm can be explained intuitively as follows.

- From picker p_1's perspective, all items emerge on a single rack. Hence a smart strategy should be batching the

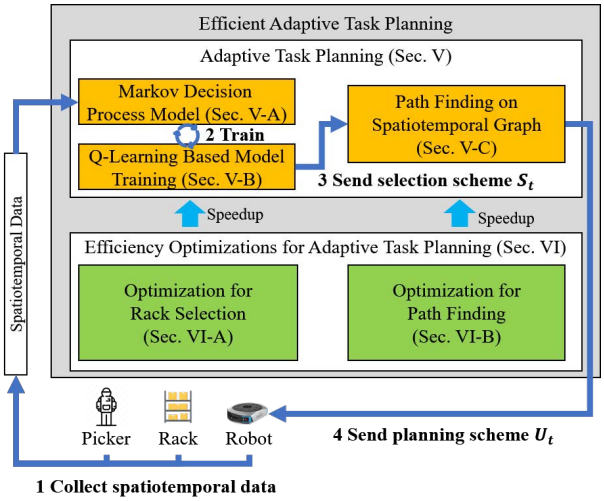


Fig. 5. Efficient Adaptive Task Planning (EATP).

delivery of all items in one time rather than moving racks as soon as one item emerges.

- From picker p_2's perspective, all items emerge on different racks. So, the bottleneck is the rack transport time.

In summary, the difference in throughput *i.e.*, the number of items emerged on a rack in unit time, shifts the dominating factor in the makespan, which affects the decision on whether to deliver a rack once an item appears or wait for more items to emerge (experimental results in Sec. VII-C also validate the observation). Naive task planing fails to incorporate such decisions for its greedy strategy. This limitation motivates us to develop a solution that adapts its decisions (*i.e.*, immediately deliver a rack or wait for more items to arrive) according to the dynamic throughput, as explained next.

IV. Efficient Adaptive Task Planning Overview

This section presents an overview of our Efficient Adaptive Task Planning (EATP) solution (see Fig. 5). From the analysis of naive task planning in Sec. III-B, its poor performance is due to lack of adaptability, *i.e.*, it considers only *how* to plan shortest paths without *when* to plan it.

Idea. Instead of immediately processing all the racks with item arrival, our EATP framework only selects a subset of racks for robots to pick up and deliver. To make this selection be adaptive, EATP reformulates the problem as a Markov decision process and incorporates a reinforcement learning based rack selection according to the dynamic item throughput. It also involves a set of efficiency optimizations for both selection and path finding in large-scale warehouse applications.

Workflow. The workflow of EATP can be summarized as four steps: *(i)* collect spatiotemporal data from pickers, racks and robots, then *(ii)* use Q-learning to train the model and *(iii)* derive the rack selection scheme S_t containing racks for which *(iv)* it plans paths.

Next, we introduce our adaptive task planning (Sec. V) and explain how to improve its efficiency (Sec. VI).

Algorithm 2: Adaptive Task Planning

Input : t: timestamp, A idle robots, R: racks , P: pickers, δ: bootstrap degree, β: learning rate, ϵ: policy derivation
Output: U_t: planning scheme at t

Initialize
initialize q
initialize spatiotemporal graph G
Rack Selection Step
approximate $\leftarrow$ Sample from Bernoulli(δ)
if *approximate* $= 1$ **then**
  $S_t \leftarrow$ Select same as Naive Task Planning
  for $r \in S_t$ **do**
    update q by Eq.(5)
else
  $S_t \leftarrow \emptyset$
  sort R in descending order based on $q(s_r, 0)$
  for $r \in R$ **do**
    action $\leftarrow \epsilon$-greedy
    if *action* $= 1$ **then**
      $S_t \leftarrow S_t \cup \{\langle r\}$
      update q by Eq.(5), where c is calculated by Eq.(4)
    if $|S_t| = |A|$ **then**
      break
Path Finding Step
$U_t \leftarrow \emptyset$
for $r \in S_t$ **do**
  $a \leftarrow$ find the closest robot of r
  $u_a \leftarrow$ find the path on spatiotemporal graph
  $U_t \leftarrow U_t \cup \{u_a\}$
  insert the u_a into G
return U_t

V. Adaptive Task Planning

In this section, we present adaptive task planning. We model rack selection from the Markov decision process perspective (Sec. V-A), and exploit reinforcement learning for model training and rack selection (Sec. V-B). For each selected rack, we find conflict-free paths (Sec. V-C). At last we integrate rack selection and path finding in Sec. V-D.

A. Markov Decision Process Model

The rack selection decisions are made sequentially every timestamp. It requires considering the rack and its corresponding picker's status (*i.e.*, the rack's containing items and the picker's queue and workload and so on). The selection decisions only depend on the *current* status of racks and pickers instead of *historical* ones, which implies that rack selection decision is sequential decision with Markov property. Hence, we can derive the definition of rack selection Markov decision process as below.

State. Since our selection decisions are rack-centric, we define the state of each rack as $\langle ap_r, ar_r \rangle$, where ap_r is the accumulative processing time of the picker associated with rack r, *i.e.*, $ap_r = \sum_{i=1}^{t} \mathbb{I}_{p_r \text{ is processing at } i}$, ar_r is the accumulative processing time of rack r, *i.e.*, $ar_r = \sum_{i=1}^{t} \mathbb{I}_{r \text{ is being processed at } i}$, where $\mathbb{I}_w$ is the indicator function, which is 1 if w is true.

Action. The action is also defined from the rack's perspective. The action α_r becomes requesting pickup, delivery and processing, which is 1 if r asks for a robot and 0 otherwise. The definition dramatically reduces the actions space as binary. If we define the action from meta view and the action is to directly select racks, the action space would be combinatorial and difficult to cope with.

State Transition. Based on the definition of action α, the transitions are as below. If $\alpha_r = 0$, the state remain the same. If $\alpha_r = 1$, state $\langle ap_r, ar_r \rangle$ will transit to $\langle ap_r + \sum_{i \in \tau_r} i, ar_r + \sum_{i \in \tau_r} i \rangle$. That is, the accumulative processing time of both the picker and the rack will increase by the total processing time of the current items on rack r, *i.e.*, $\sum_{i \in \tau_r} i$.

Though a single transition always changes ap_r and ar_r in the same way, updates from different racks can make ap_r and ar_r different since one picker can be associated with multiple racks. The fact that one picker is responsible for multiple racks also implies the dependence among racks, which motivates the joint state modeling of the rack and its picker.

Reward. We use negation of the increment in a picker's finish time f_p after selecting certain rack as the reward. However, it is difficult to derive the increment of picker's finish time. This is because the rack selection decision is performed before path planning, and thus the delays for pickup, delivery, queuing and return are unknown. Thus we estimate the reward as follows.

$$c = -\left(\max\{f_p, d(l_r, l_{p_r})\} + \sum_{i \in \tau_r} i \right) \tag{4}$$

where the max term denotes the increase in waiting time and the sum term is the increment of ap_r. The negation is because our goal is to minimize makespan while reinforcement learning maximizes the sum of rewards.

Note that our reward design considers the end-to-end delay for pickup, delivery, queuing, processing, and return, and thus is aligned with our makespan definition.

Optimizations. Based on the above Markov decision process definition, optimizing makespan requires us to find *policy* that accounts for the current states and make actions for all racks. The policy is derived from *value function*, which is represented as $q(\langle ap_r, ar_r \rangle, \alpha)$ in our problem. It maps the state-action to the expected accumulative rewards. Based on the definition of reward, the value function indicates the expected finishing time of rack r considering both its delivery time and the picker's finish time. According to the value function q, the best action would be $\arg\max_{\alpha'} q(s, \alpha')$. However, this policy may be trapped into sub-optimal solutions because $q(s, \alpha)$ can be inaccurate especially in the early stage of training. Instead, we adopt the ϵ-greedy policy [13]. It chooses the current best action with $1 - \epsilon$ probability and a random action with ϵ probability to balance exploration and exploitation.

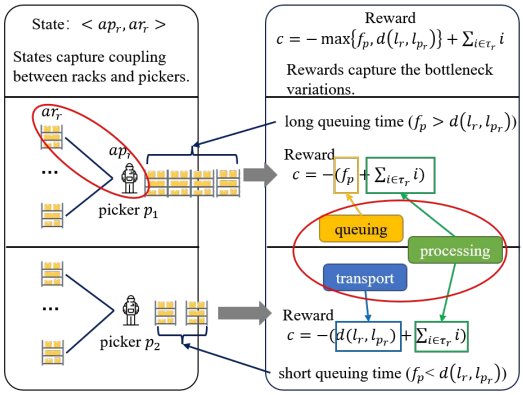


Fig. 6. Illustration of our Markov decision process model

Remarks. Fig. 6 illustrates why our model is fit for the TPRW problem. The state definition jointly implies both the rack and its picker, the reward will change while the fulfillment bottleneck varies.

We can derive a policy only if the value function can be effectively trained, as explained below.

B. Q-Learning Based Model Training

We apply Q-learning [14], a classic temporal difference based bootstrap method to train the value function, since it is better fit for online learning and highly self-adaptive [13]. The Q-learning trains value function as below.

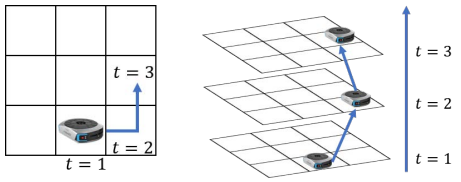
$$q(s, \alpha) \leftarrow q(s, \alpha) + \beta \cdot (c + \gamma \max_{\alpha'} q(s', \alpha') - q(s, \alpha)) \tag{5}$$

where s/s' are the current/next state after taking action α, β is the learning rate, c is the reward and γ is the discount factor.

Directly applying Q-learning as above will bootstrap *unexplored* states. Recall that state $\langle ap_r, ar_r \rangle$ is time-dependent. When updating the value function by Eq.(5), s' is $\langle ap_r + \sum_{i \in \tau_r} i, ar_r + \sum_{i \in \tau_r} i \rangle$ (see state definition in Sec. V-A). The new state is unexplored because both ap_r and ar_r always increase, preventing the value function from converging. As a remedy, we integrate a greedy method into the training. Specifically, at each timestamp, we choose the greedy method with probability δ and the original bootstrap with $1 - \delta$ probability. This way, the greedy method will provide solutions, explore some states and update the value function approximately. Then based on the approximation, bootstrap is able to train the value function more precisely. The greedy method adapts the "most slack picker first" strategy. That is, it greedily chooses those racks whose associated picker has the smallest f_p.

C. Path Finding on Spatiotemporal Graph

Given the selection schemes, a path finding algorithm plans conflict-free paths. We adopt A* algorithm [11] for finding conflict-free paths. Instead of searching on the spatial graph, the algorithm searches on the spatiotemporal graph to avoid



(a) Spatial graph (b) Spatiotemporal graph

Fig. 7. Illustration of a spatial graph and a spatiotemporal graph.

conflicts. Intuitively, the space graph is duplicated in every time step. Each vertex represents a location with a certain timestamp, while each edge represents two vertex are adjacent both spatially and temporally (see Fig. 7) [15]. On spatiotempoal graph, the algorithm starts with the source vertex (*i.e.*, grid with certain timestamp) and then maintain a open set which stores vertices explored. Based on the current cost and heuristic value (h-value), we choose a vertex from the open set for the next round search. When searching on the grid based space, the h-value is usually defined as the Manhattan distance from current vertex to the destination [16].

D. Put it Together

By integrating rack selection and path finding, we propose Adaptive Task Planning (ATP), as shown in Algorithm 2.

Lines 4 to 19 are rack selection step while lines 20 to 26 are path finding step. In rack selection step, we first randomly decides whether to approximate or bootstrap (line 5). If approximate, we use the greedy method to derive the selection scheme (lines 6 to 9). If bootstrap, we sort racks based on the value function in line 12. It will then preferentially select racks with the largest expected finish time till no robot is available (line 13 to 19). Both steps adopt Q-learning to update the value function (line 9 and 17). Based on selection scheme calculated from selection step, path finding step will assign the closest robot of each selected racks (line 23) and finds path for it (line 24). The spatiotemporal graph will maintain all prior planed path for conflict avoidance (line 26). Note that δ controls the degree of bootstrap. A larger δ means smaller bootstrap. From empirical results (see Sec. VII-B), a δ smaller than 0.4 contributes to effective training.

VI. Efficiency Optimizations for Adaptive Task Planning

In this section, we present our efficient design for adaptive task planning. For rack selection step, we flip the requesting side from rack to robot to reduce selection time consumption (Sec. VI-A). For path finding step, we optimize both the time and memory consumption. We replace the spatiotemporal graph with conflict detection table which has less space complexity while support for quick conflict detection and use cache aiding the finding step (Sec. VI-B). At last we elaborate integrate these efficient design into ATP (Sec. VI-C).

A. Optimization for Rack Selection

From the time complexity analysis above, the bottleneck of ATP's rack selection step lies in sorting racks. Instead of traversing racks then requesting robot for delivery, we accelerate the process by flip requesting side to robot. Specifically, we traverse robots and then finding a rack among its closest K racks. Since all racks' locations in the storage area are fixed, recording closest K racks of different grids is static and easy to maintain. Then for each robot, we can easily find its closest racks according to its located grid and find the closest selected rack among those racks (See Fig. 8).

B. Optimization for Path Finding

Memory Compression via Conflict Detection Table. Searching on a spatiotemporal graph incurs large space cost due to the ever-increasing temporal dimension. In worst case, the space complexity of the spatiotemporal graph is $O((HW)^2)$ because the space complexities of the spatial graph and the path lengths are both $O(HW)$. Next we introduce our conflict detection table to reduce the space complexity while maintaining efficient conflict detection.

In conflict detection table, an array is built for all grids, and each entry contains a set recording the passing time. The set can either be implemented based on balanced binary search tree or hash set for quick search. When planning path, ATP will search for a grid and by checking the grid's corresponding entry contains the timestamp or not, it will quickly judge whether conflict will happen. This conflict detection table removes the ever growing temporal dimension. The space complexity is decreased to $O(HW)$.

In addition to path finding, the table also supports update and insertion. The update operation deletes all passed timestamp. This operation reduces the space cost of the table and is executed periodically. The insertion operation inserts a path to the table. It will insert the passing time to the corresponding grid entry for each point of trajectory.

Cache-aided Path Finding. We can further accelerate the path finding algorithm by caching certain shortest paths without considering conflict, and then deriving the conflict-free paths based on the shortest paths.

As mentioned in Sec. V-C, the path finding algorithm will maintain a open set and pick vertex from it for path finding. Specifically, when picking vertex from the open set, if the distance between the current vertex and the destination is within a threshold L, we directly extract the shortest path from the cache and derive the conflict-free path. The corresponding policy is to let the robot wait till there is no conflict to move next steps along the shortest path. The rationale is that, when the current vertex is close to the destination (*i.e.*, within the threshold), instead of searching for the shortest conflict-free path, directly moving along the shortest path with some wait may be effective, since it is already close to the destination. Such cache can notably reduce the size of open set and thus the search cost.

C. Integrate Efficient Design into ATP

Integrating all efficient designs, we get our final Efficient Adaptive Task Planning (EATP), as in Algorithm 3 and Fig. 8.

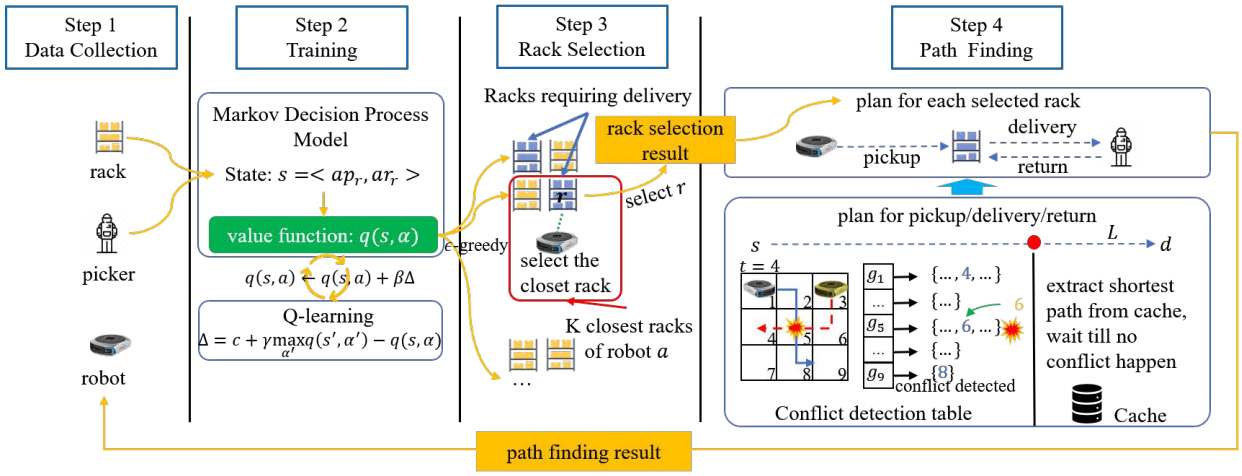


Fig. 8. Workflow of Efficient Adaptive Task Planning (EATP).

It first initializes the cache (line 3) and conflict detection table (line 4). The cache contains shortest paths whose Manhattan distances are within threshold L. For rack selection, lines 10 to 13 are flip requesting and line 18 is cache-aided path finding. It will find paths by CDT and when the current vertex is close to the destination (l_r or l_p depended on different steps of pickup, delivery or return) less than L, it will derive the last segment of path by waiting till no conflict occurs.

For the hyperparameter distance threshold L, it controls the degree of cache-aiding, a larger value encourages using cache for less computation consumption.

VII. Evaluation

This section presents the evaluations together with case study of our proposed methods.

A. Experimental Setup

Datasets. We use both synthesized and real datasets. The two synthesized datasets Syn-A and Syn-B are generated on two warehouse layouts with different numbers of items. All items emerge following Poisson distribution and each racks picking time is distributed uniformly between 20 and 40 seconds, which is close to the real situation. Two real datasets are derived based on historical records from Geekplus, one of the world's leading smart logistics companies [3]. Two real datasets are named as Real-Normal and Real-Large considering the scalability of data. Table II lists the dataset details.

Validation system. To test algorithms' performances on these datasets, We build a virtual warehouse which simulates the movement of robots and the processing of pickers. At each timestamp, it collects all idle robots and racks containing remaining items as well as pickers' working status, then executes the algorithm for path planning. Then it converts the path planning scheme to instructions on robots' motion. It also records the performance of task planning algorithms in terms of effectiveness and efficiency. A snapshot of the validation system is shown in Fig. 9(a). A real warehouse is also used for deployment demonstration (See Fig. 9(b)).

[3]https://geekplusrobotics.borealtech.com/en/

Algorithm 3: Efficient Adaptive Task Planning

Input : t: current timestamp, A: idle robots, P: pickers, R: racks, L: distance threshold
Output: U_t: planning scheme at time t

1 **Initialize**
2 initialize q
3 initialize $Cache$ containing all shortest paths with length $\leq L$.
4 initialize conflict detection table CDT
5 **Rack Selection Step**
6 approximate $\leftarrow$ Sample from Bernoulli(δ)
7 **if** *approximate* $= 1$ **then**
8 $S_t \leftarrow$ Same as lines 7 to 9 in Algorithm 2
9 **else**
10 **for** $a \in A$ **do**
11 **for** $r \in \{K$ *racks closet to* $l_a\}$ **do**
12 Update S_t, q same as line 14 to 17 in Algorithm 2
13 break inner loop when a rack is selected
14 **Path Finding Step**
15 $U_t \leftarrow \emptyset$
16 **for** $r \in S_t$ **do**
17 $a \leftarrow$ find closet robot to r
18 $u_a \leftarrow$ path finding on CDT and derive via $Cache$
19 $U_t \leftarrow U_t \cup \{u_a\}$
20 $CDT.insert(u_a)$
21 **return** U_t

As for implementation, we set the default value of δ, ϵ, learning rate β and distance threshold L to be 0.2, 0.1, 0.1 and 50, respectively. All algorithms as well as the validation system are implemented in Java 8 and the experiments are run on 4 cores CPU Intel(R) Xeon(R) Platinum 8269CY CPU T

TABLE II
SUMMARY OF DATASETS.

Name	$H \times W$	#Item	#Robot	#Rack
Syn-1	233×104	10^5	0.5k	5.0k
Syn-2	426×146	5×10^5	1.0k	1.3k
Real-Norm[3]	240×206	5.6×10^5	1.0k	10k
Real-Large[3]	541×302	10^6	3.0k	34k

3.10GHz with 20 GiB Java virtual machine memory.

(a) Virtual system (b) Real warehouse

Fig. 9. Snapshots of virtual (a) and real (b) warehouses.

Baselines. We compare EATP with the following methods.

- **Naive Task Planning (NTP) [7]**. This method is the directly extension of state-of-the-art path planning algorithm [7]. It assigns robots to racks whose corresponding picker has the earliest finish time f_p (see Sec. III).
- **Least Expiration First planning (LEF) [17]**. This spatiotemporal task selection algorithm selects tasks with least expiration time [17]. Though our items emerge without expiration, by assuming all items with the same degree of tolerance of delay, this algorithm will select racks whose items are emerged earliest.
- **Integer Linear Programming planning (ILP) [12]**. This method proposes an integer linear programming based approach to handle orders composed of items from different racks [12]. We extend their method to our problem by adding the pickers' status in the linear programming model.
- **Adaptive Task Planning (ATP)**. This algorithm incorporates reinforcement learning for rack selection and A* algorithm for path finding as introduced in Sec. V.
- **Efficient Adaptive Task Planning (EATP)**. This algorithm incorporates all efficient design for both selection and planning (Algorithm 3).

Evaluation Metrics. We use three metrics to evaluate the algorithms in terms of *effectiveness*.

- **Makespan (M)**. It is the objective of TPRW problem as defined in Eq.(1). A smaller makespan indicates a higher processing efficiency of a warehouse.
- **Picker's Processing Rate (PPR)**. This metric is defined as follows.

$$PPR = \frac{1}{|P|} \sum_{p \in P} \frac{\sum_t \mathbb{I}_{p \text{ is processing at } t}}{M} \tag{6}$$

where $\mathbb{I}_{p \text{ is working at } t}$ is 1 if picker p is processing or 0 otherwise, so the summation term in numerator is the picker's total processing time. A larger PPR means that all pickers are working sufficiently, leading to a higher processing efficiency.

- **Robot's Working Rate (RWR)**. Similar to PPR, RWR can be defined as follows.

$$RWR = \frac{1}{|A|} \sum_{a \in A} \frac{\sum_t \mathbb{I}_{a \text{ is working at } t}}{M} \tag{7}$$

where the summation term in numerator is the robot's working time. A large RWR means that the assignment algorithm using robot in a more effective way, that is less delivering time and more picking time.

TABLE III
MAKESPAN COMPARISON ON ALL DATASETS.

Method	Syn-A	Syn-B	Real-Norm	Real-Large
NTP [7]	95,713	229,865	222,044	264,139
LEF [17]	68,736	225,484	176,317	–
ILP [12]	72,423	219,555	173,446	–
ATP (Ours)	**60,193**	**209,531**	**165,438**	**220,257**
EATP (Ours)	**60,753**	**209,866**	**164,628**	**220,263**

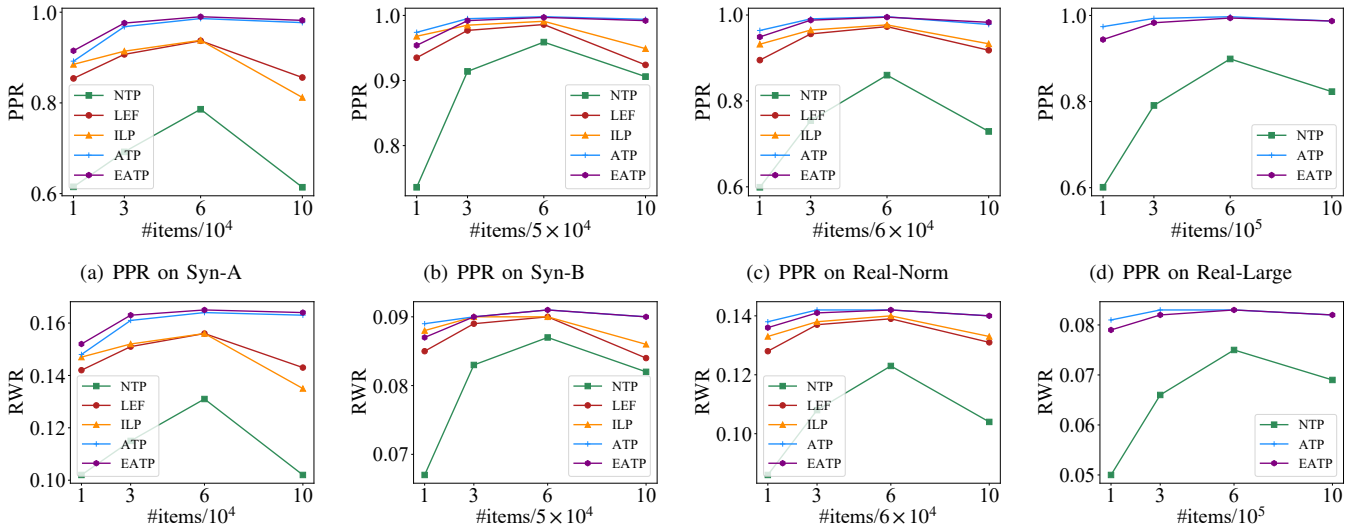
Apart from effectiveness metrics, we adopt three metrics to quantify the time and memory *efficiency*.

- **Selection Time Consumption (STC)**. The selection time consumption is the total time usage when executing the algorithms for making rack selection decisions. A smaller STC means a higher executing efficiency.
- **Planning Time Consumption (PTC)**. Similar to STC, the planning time consumption is the total time usage of path planning scheme generation. A smaller PTC means a higher executing efficiency.
- **Memory Consumption (MC)**. It measures the memory consumption when executing an algorithm. A smaller memory consumption indicates a higher space efficiency.

B. Experimental Results

Overall Performance. The makespan results of all datasets are shown in Table III. ILP and LEF are too slow to execute on dataset Real-Large so we only compare our methods with NTP. Our ATP and EATP reduce makespan by 4.5% ˜37.1% than the baselines. Specifically, our EATP outperforms LEF, ILP by 8.5% and 8.7% on average, respectively.

Fig. 10 illustrates the comparison results for other effectiveness metrics. The x-axis (y-axis) indicates the task planning procedure (the value of PPR or RWR). Our EATP and ATP achieves the highest PPR and RWR, which is 4.6% ˜59.1% higher and 3.5% ˜59.8% than baselines, respectively. In particular, our EATP outperforms LEF by 9.4% and 9.3% on average in terms of PPR and RWR, respectively. As for ILP, EATP gains an average improvement of 9.9% and 10.2% in



(a) PPR on Syn-A (b) PPR on Syn-B (c) PPR on Real-Norm (d) PPR on Real-Large

(e) RWR on Syn-A (f) RWR on Syn-B (g) RWR on Real-Norm (h) RWR on Real-Large

Fig. 10. Picker's Processing Rate (PPR) and Robot's Working Rate (RWR) comparisons.

terms of PPR and RWR, respectively. Also, EATP consistently outperforms the baselines in all the three datasets.

We also find an interesting fact that PPR has nearly the same variations as RWR. This is reasonable because all pickers' processing workload is delivered by robots thus robots are as busy as pickers. Also note that EATP has a slightly performance loss than ATP. This is because the adoption of efficient design introduced in Sec. VI trades some precision for acceleration, which leads to a slightly performance loss. However, the performance loss is less than 1% and it still significantly outperforms other baselines, so the trade-off is worthwhile. We will show that EATP has a large reduction on both time and space consumption.

Adaptibility. Our ATP and EATP shows a strongly adaptive property mainly because: *(i)* our algorithms are steadily outperforms other baselines overall datasets that have different layouts, levels of throughput and so on, indicating our algorithm are capable of multiple situations. *(ii)* PPR and RWR of our algorithms steadily remain high during the task planning procedure, unlike other baselines vary largely (See Fig. 10), which implies that our algorithms can adaptively make task planning decisions and keep robots and pickers working steadily despite of the time-variant throughput.

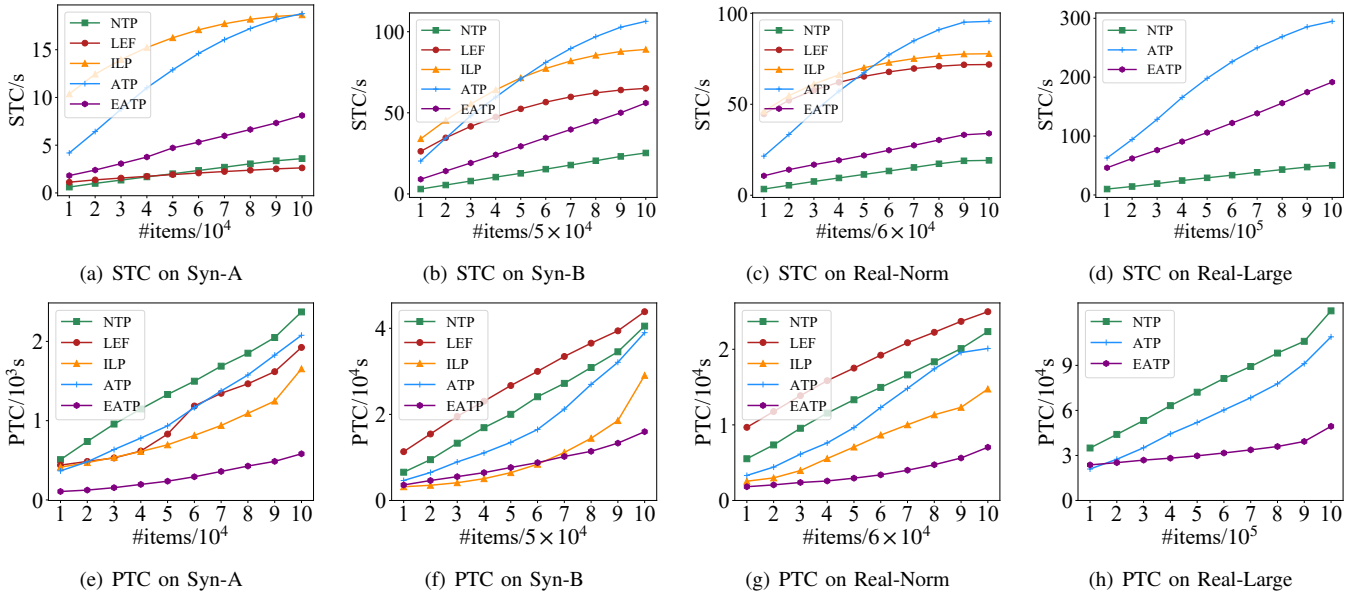
Scalability. Fig. 11 illustrates the selection and planning time consumption results. For selection time consumption, without adoption of efficient design, ATP does not perform well and it is even worse than ILP. With the selection optimization (Sec. VI-A), our EATP's efficiency improves significantly by 153.8% ~280.9% and close to most naive Greedy methods, whose complexity is only $O(|P|\log|P|+|A|)$. Its time usage is less than LEF and ILP by 52.7% and 56.5% at most, respectively. As for planning time consumption, our efficient design introduced in Sec. VI-B improves EATP planning efficiency largely. Specifically, our EATP has a reduction of 75.5%, 60.5%, 71.8% and 60.8% at most compared with other algorithms on dataset Syn-A, Syn-B Real-Normal and Real-Large respectively. This results shows that our algorithm has a strong potential for large-scale processing requirement. Especially on Real-Large, our EATP's total execution time are less than NTP for over 7000 seconds.

Note that even though ATP adopts the same path planning algorithm as other baselines, it still has a smaller PTC because its adaptibility helps reduces the planning frequency.

As for memory cost, Fig. 12 illustrates the comparison results. All algorithms except EATP have nearly the same memory cost due to the memory cost bottleneck lies on A*-based planning which these algorithm both adopts. Besides, all algorithm has a steadily usage of memory as the task planning procedure goes on, this is because we eliminate passed saptiotemporal graph or timestamps timely and maintain the memory consumption relatively stable. The comparison results shows our EATP is also memory efficient. With the help of conflict detection table (Sec. VI-B), we can largely reduces the memory usage by 16.4%, 68.4%, 89.2% and 58.1% on Syn-A, Syn-B, Real-Normal and Real-Large, respectively.

Summary of Experimental Results. The experimental results are summarized as below.

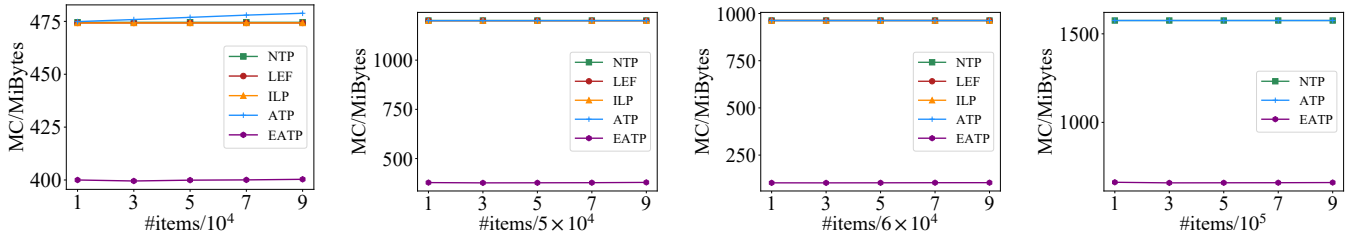
- Our ATP and EATP achieve state-of-the-art effectiveness performances. With an reduction on makespan by mostly 37.1% than other baselines.
- The steady superiority over other baselines on all datasets and the steadily high PPR and RWR during the whole task planning procedure indicates a high adaptability of our algorithm.
- The efficiency design on both selection and planning improves efficiency by up to 280.9%, which enables our EATP to overcome scalability challenge.



(a) STC on Syn-A (b) STC on Syn-B (c) STC on Real-Norm (d) STC on Real-Large

(e) PTC on Syn-A (f) PTC on Syn-B (g) PTC on Real-Norm (h) PTC on Real-Large

Fig. 11. Selection Time Consumption (STC) and Planning Time Consumption (PTC) comparisons.



(a) Syn-A (b) Syn-B (c) Real-Norm (d) Real-Large

Fig. 12. Memory Consumption comparisons.

C. Case Study of Bottleneck Variations on Geekplus

We conduct case study on demonstrative warehouse with over 1 thousand robots and 50 thousand items which is built by Geekplus[3], a leading smart logistics company. We validate the phenomenon of bottleneck variations and how our method can adaptively batching items. The bottleneck variations among processing, queuing and transport (*i.e.*, summation of pickup, delivery and return) are shown in Fig. 13. The x-axis is picking time while y-axis is cost summation of all racks under different fulfillment steps. At the beginning, when the number of items is small, the bottleneck lies in transport. As times goes on, the number of items is continually growing, bottleneck convert to queuing. Meanwhile the processing time grows and then it remains static.

Next we choose a single rack to illustrate how our ATP accounts for the situation and make decisions adaptively. Different items emerge on this rack at different time, where the bottleneck is different. Our ATP decides to batch items instead of delivery immediately. When the bottleneck lies in transport, ATP tends to batch less items while it will tends to batch more orders as queuing time becomes larger.

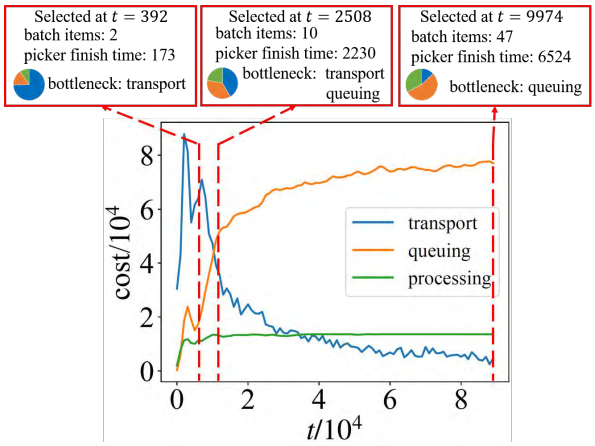


Fig. 13. Bottleneck variation over time.

VIII. Related Work

Our study is related to two threads of studies.

A. Multi-Agent Path Finding

Multi-agent path finding is inspired by the need for coordinating hundreds of robots in a robotized warehouse [2]. The problem is about searching for shortest paths while meeting up with conflict constraints [10]. Although this problem has been proved to be NP-hard [18], earlier studies still try to design branch-bound search algorithms hoping to find the optimal solution in an offline setting [3]–[6]. Yet these solutions are unfit for large-scale applications. For example, the empirical validations in these studies typically consider fewer than 200 robots and delivery tasks, while their running time can be up to 5 minutes, which fails to support online planning with over thousands robots and million-scale daily throughput.

More recently, online multi-agent path finding has been explored [7]–[9]. In [7], the authors study the online multi-agent path finding problem where tasks emerge in an online manner while robots remain static, which is same as our settings. Other efforts [8], [9] consider a different setting where robots and tasks are tied and emerge together. The setting ignores the process to assign robots to tasks, which is not aligned with the rack-to-picker setting in our problem.

In this work, we study online multi-agent path finding in settings more aligned with large-scale applications. On the one hand, we aim to minimize the end-to-end makespan that covers the entire fulfilment cycle (pickup, delivery, queuing, processing, and return). On the other hand, we account for the high-volume, high-variation item arrivals. Existing solutions fail to deliver satisfactory effectiveness and efficiency in these two settings.

B. Task Assignment and Planning in Spatial Crowdsourcing

Task assignment and planning are two core issues in spatial crowdsourcing [19], a popular topic in database research.

Task assignment is typically modeled as a bipartite matching problem, where vertices from two sides represented tasks and workers, respectively [20]. Various optimization goals have been investigated, including maximizing the matching edge weights [21], minimizing the detour distance [22] and load balancing [23], [24]. Alternatively, task assignment can be viewed as a selection problem where the objective is to maximize the number of fulfilled tasks [17], [25].

Although these formulations are aligned with different spatial crowdsourcing applications such as ride hailing [21], [24], geographical data generation [17], [22], [25] and workload distribution [23], [24], they cannot be trivially extended into our problem setting. This is because spatial crowdsourcing confines the assignment decision into a batch or assume an expiration of each task. After expiration, a task is discarded. In our scenario, all tasks must be fulfilled and assignment decisions cover the whole time horizon.

Task planning, or route planning, as another important issue in spatial crowdsourcing [26]–[30]. These studies optimize different objectives [26], [27], [29] and also adopt data-driven approaches [28], [30] to improve the planning performance. Notice that [30] is also related to logistics and uses reinforcement learning. Their methods cannot be adapted to our problem because they focus on planning *among* warehouses and factories on road networks without considering conflicts, and the reinforcement learning are used to avoid myopic optimization of the travel distance, rather than the bottleneck variations in our problem. Furthermore, the number of tasks and robots we plan is ten times larger than their orders and vehicles, making us confronting stricter efficiency requirement. Since works in spatial crowdsourcing are humans, their route planning does not consider issues such as conflicts. In contrast, route planning for robotized warehouses is more challenging for the coordination among robots to avoid conflicts.

An orthogonal research on path planning focuses on indoor spaces, where the positioning data may be uncertain and noisy [31]–[33]. We assume the location data of robots are accurate, which is common in real-world robotized warehouses. Task planning under location errors is out of our scope.

IX. Conclusions

In this paper, we propose the TPRW problem, an extension from online multi-agent path finding problem. It defines an end-to-end makespan incorporating all steps of item fulfillment, which is suitable for large-scale and highly varied throughput in modern robotized warehouses. Direct extension of state-of-the-art methods is inflexible to solve the problem. In response, we propose the framework of efficient adaptive task planning (EATP). EATP exploits reinforcement learning to adaptively decide and plan paths for robots. It also adopts a set of acceleration techniques to optimize both time and memory efficiency. Experimental results show that EATP achieves 37.1% and 75.5% improvement in effectiveness and efficiency over the state-of-the-art online multi-agent path finding algorithms.

Acknowledgments

We are grateful to anonymous reviewers for their constructive comments. This work is partially supported by the National Key Research and Development Program of China under Grant No. 2018AAA0101100, the National Science Foundation of China (NSFC) under Grant No. U21A20516, U1811463 and 62076017, and the State Key Laboratory of Software Development Environment Open Funding No. SKLSDE-2020ZX-07. This research was supported by the Lee Kong Chian Fellowship awarded to Zimu Zhou by Singapore Management University. Yongxin Tong is the corresponding author in this paper.

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] P. R. Wurman, R. D'Andrea, and M. Mountz, "Coordinating hundreds of cooperative, autonomous vehicles in warehouses," *AI Mag.*, vol. 29, no. 1, pp. 9–20, 2008.

[3] T. Standley, "Finding optimal solutions to cooperative pathfinding problems," in *AAAI*, vol. 24, no. 1, 2010.

[4] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, "The increasing cost tree search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 195, pp. 470–495, 2013.

[5] 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.

[6] E. Boyarski, A. Felner, R. Stern, G. Sharon, D. Tolpin, O. Betzalel, and E. Shimony, "Icbs: Improved conflict-based search algorithm for multi-agent pathfinding," in *IJCAI*, 2015.

[7] H. Ma, J. Li, T. K. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding for online pickup and delivery tasks," in *AAMAS*. ACM, 2017, pp. 837–845.

[8] 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.

[9] 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*. AAAI, 2021, pp. 11 272–11 281.

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

[11] 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.

[12] N. Boysen, D. Briskorn, and S. Emde, "Parts-to-picker based order processing in a rack-moving mobile robots environment," *European Journal of Operational Research*, vol. 262, no. 2, pp. 550–562, 2017.

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge: MIT press, 2018.

[14] C. Watkins, "Learning from delayed rewards," *PhD thesis, Cambridge University*, 1989.

[15] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," in *AAMAS*. ACM, 2016, pp. 1144–1152.

[16] F. Grenouilleau, W. van Hoeve, and J. N. Hooker, "A multi-label a* algorithm for multi-agent pathfinding," in *ICAPS*. AAAI, 2019, pp. 181–185.

[17] D. Deng, C. Shahabi, U. Demiryurek, and L. Zhu, "Task selection in spatial crowdsourcing from worker's perspective," *GeoInformatica*, vol. 20, no. 3, pp. 529–568, 2016.

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

[19] Y. Tong, Z. Zhou, Y. Zeng, L. Chen, and C. Shahabi, "Spatial crowdsourcing: a survey," *VLDBJ*, vol. 29, no. 1, pp. 217–250, 2020.

[20] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing," in *ICDE*. IEEE, 2016, pp. 49–60.

[21] Z. Xu, Z. Li, Q. Guan, D. Zhang, Q. Li, J. Nan, and et al, "Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach," in *KDD*. ACM, 2018, pp. 905–913.

[22] C. F. Costa and M. A. Nascimento, "Online in-route task selection in spatial crowdsourcing," in *SIGSPATIAL*. ACM, 2020, pp. 239–250.

[23] Y. Zhao, K. Zheng, J. Guo, B. Yang, T. B. Pedersen, and C. S. Jensen, "Fairness-aware task assignment in spatial crowdsourcing: Game-theoretic approaches," in *ICDE*. IEEE, 2021, pp. 265–276.

[24] Z. Chen, P. Cheng, L. Chen, X. Lin, and C. Shahabi, "Fair task assignment in spatial crowdsourcing," *PVLDB*, vol. 13, no. 11, pp. 2479–2492, 2020.

[25] D. Deng, C. Shahabi, and U. Demiryurek, "Maximizing the number of worker's self-selected tasks in spatial crowdsourcing," in *SIGSPATIAL*. ACM, 2013, pp. 314–323.

[26] H. Kriegel, M. Renz, and M. Schubert, "Route skyline queries: A multi-preference path planning approach," in *ICDE*. IEEE, 2010, pp. 261–272.

[27] 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.

[28] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *ICDE*. IEEE, 2013, pp. 410–421.

[29] 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.

[30] 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*. IEEE, 2021, pp. 2511–2522.

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

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

[33] 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*. IEEE, 2020, pp. 2014–2017.