Time Window Based Path Planning of Multi-AGVs for Auto-Sorting in Logistics Center

Zhenyu Fan[1], Chonglin Gu[1], Xiao Yin[1],
Hongwei Du[1,2], and Hejiao Huang[1,2](✉)

[1] Harbin Institute of Technology, Shenzhen, China, 518055
[2] Shenzhen Key Laboratory of Internet Information Collaboration, Shenzhen, China, 518055
huanghejiao@hit.edu.cn

Abstract. With the booming development of online shopping, a massive number of commodities have to be delivered to customers in different regions through express mail. It brings a new challenge that modern logistics center must promote its sorting ability in a more automated way rather than artificially. In this paper, we study the problem of path planning of multi-AGVs (automated guided vehicles) for automatic sorting in logistics center. Given the warehouse map of logistics center, our goal is to transport as many cargoes as possible from different imports to the designated exports using the smallest possible number of AGVs within given period. To solve the problem, we abstract the actual warehouse map into a directed graph, based on which we give the description of our problem. We propose two heuristic algorithms to search for available free time windows of the blocks that each AGV can pass through along its selected path without collision. Experiments show that our methods are more efficient than existing work with higher sorting throughput within given time while using less vehicles.

Keywords: Multi-AGVs; Automatic Sorting; Path Planning

1 Introduction

The vigorous development of e-commerce has already changed people's shopping habits, and there can be hundreds of thousands of commodities delivered by express mail every day. As a result, the logistic centers of those express companies have to face the challenge of packages sorting with so large number that is almost impossible to be completed by traditional human labor. Therefore, AGVs (automated guided vehicles) are gaining increasing attentions in modern logistics centers, since they can automatically and efficiently transport cargoes from different imports to the corresponding sorting destinations in a collaborative way without collision. However, exploring feasible paths for multiple AGVs is not so easy, because the search space grows exponentially with the increase of the number of vehicles and the complex influence of the path choices of previous vehicles on the motions of the following ones [1, 2].

In this paper, we aim at finding effective paths of AGVs so that the sorting throughput can be maximized while using as few vehicles as possible within given time. For ease of modeling, the warehouse map is divided into square blocks, some of which are used as passable road for vehicles, while others represent imports, exports, or temporary buffers that are used to avoid traffic congestion. Thus, the actual warehouse map can be abstracted into a directed graph with vertexes representing the passable blocks. For each block, it can accommodate only one AGV in a time window. Note that, there may exist collisions when two or more vehicles have the same blocks to pass through in the same time window, which can be avoided using wait-and-go policy. Our problem is NP hard with large-scale search space that can not be solved using existing optimization tools [3]. To solve this problem, we first generate a set of potential candidate paths for each AGV using improved penalized A* algorithm, and then select the one with earliest arrival time window and make reservation for this path. Once reserved, the time windows of the blocks along this path can not be used by the any other AGVs at the same time. We propose two different heuristic algorithms F-TWS and RS-TWS to search for available free time windows of the blocks that each AGV can pass through along its selected path in global spatial-temporal space. Experiments show that our methods are more efficient than existing work with higher sorting throughput within given time while using less vehicles under different layouts of warehouse.

The rest of this paper is organized as follows. In Section 2, state-of-the-art works in literature is reviewed. We describe our problem based on the abstracted directed graph in Section 3. After that, our approaches are presented in Section 4. In Section 5, we setup experiments and make analysis of the results. Finally, we conclude the whole paper and discuss possible future work in Section 6.

2 Related Work

In general, existing work on path planning of multi-AGVs can be classified into two categories: centralized (or coupled) and decentralized approaches.

For centralized approach, all AGV are scheduled by a centralized controller that constantly gathers the environment information in real time and makes decisions on path planning with consideration of collision avoidance [4, 5]. The main advantage of the centralized methods is that the optimal schedule can be obtained when the problem scale is not so large, but its complexity will increase exponentially with scale, needing more calculating time for the controller. Wang et al. in [6] proposed a time window based A* algorithm. The basic idea is to replace the distance heuristic function to earliest arrival time, and the potential conflicts are avoided through setting the heuristic value of conflict nodes to be infinite. This method can run quickly to search path but with too many failures. Zeng et al. gave a solution using HTN based planning [7]. The algorithm searches the surrounding passable area iteratively and returns infeasibility when meeting conflicts. Though this approach can manage the conflict situations and time constrains well, it performs poorly with the increase of AGV scale. The searching space of routing using multi-AGVs can be reduced using layered structure. As done in [8], they divided the global environment into sub-regions (cells) in the lower layer within which conflicts are solved, while constructing a virtual map consisting of region cells in the upper layer and path planning are made in it.

Considering the running speed in large-scale scenario, there have been some decentralized algorithms proposed, in which each AGV determines its own path and avoids conflicts through negotiating with nearby AGVs [9, 10]. This approach does not require complete system information so that it can be applied to the unknown environment exploration or partial observation losses [11]. The searching space of decentralized methods is always incomplete and there exists an obvious gap in system throughput compared with centralized approaches due to path finding failures caused by deadlocks. However, decentralized techniques are generally faster than centralized strategies [12]. Wilde et al. proposed a rule based planning algorithm that defines a set of traffic rules to reduce computational complexity and guarantee the completeness at the cost of solution quality [13]. Another way for decentralized implementation is to add priorities to AGVs to ensure vehicles with lower priority do not generate routes that have collision to existing plans of higher priority ones. Wang et al. in [14] proposed an adaptive priority re-assignment idea in AGV routing, but the algorithm often fails to find valid routes and far from optimal. Therefore, ther later improved their algorithm by proposing a Guided Iterative Prioritized Planning strategy, but at the expense of more computing time due to algorithm complexity [15].

The above mentioned work concern only one loop of vehicles' planning. Once finished delivering the goods, the vehicles will be in idle position (or parking lot), which can not be directly applied to the real logistic sorting scenario with uninterrupted transportation. Moreover, most of the scheduling in literature are designed specifically for the predefined layout of the warehouses with no consideration of universality. In our work, we conquered the two drawbacks by proposing two strategies that can plan and coordinate a large number of AGVs back and forth for a long term, even under different warehouse layout scenarios. To optimize the path planning, we generate a set of potential candidate paths using penalizing like that in [16], which proved to be more efficient when compared with k-shortest paths [17], k-disjoint path [18], Pareto [19], and Plateau [20].

3 Problem Description

A) Modeling of Layout into A Directed Graph

For a logistics sorting center, we first need to abstract its layout into a directed graph before path planning. For ease of modeling, the layout is divided into square blocks of the same size, which are abstracted as a point in the graph. There are directed arcs connecting the adjacent points with unit moving distance of an AGV. In each time slot, one block can be occupied by only one AGV. Here is an example showing the modeling of directed graph, as shown in Fig. 1.

There are four types of blocks in the warehouse: Red blocks represent the imports (or called start blocks), where multiple AGVs are scheduled to load up

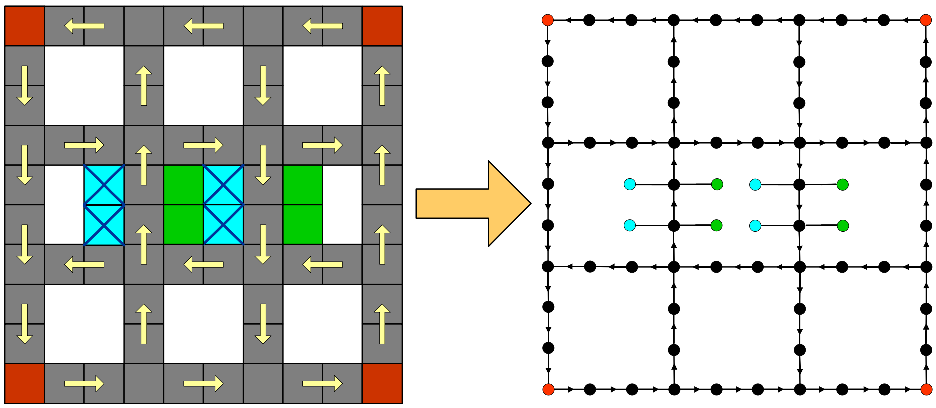

Fig. 1. The modeling of warehouse layout into a directed graph.

express packages. Blue blocks represent sorting exports. In practice, the sorting block can be a hole with a collection bag, so that the AGVs will stop on the adjacent passable block to dump the carried package into its target hole. We use grey blocks to represent passable blocks for AGVs to pass through, which can accommodate only one AGV in each time slot. Green blocks represent buffer for AGVs to temporarily stop and avoid collisions. The traveling direction of each block is predefined for AGVs to go along. Note that, the unused blocks with white color in the layout are reserved for future extension. Our scheduling focus only on the above mentioned four types of blocks.

B) Problem Description

Given the warehouse layout of logistics center, our goal is to transport as many cargoes as possible from different imports to the designated exports using the smallest possible number of AGVs within given period. We make the following assumptions for our problem of path planning of multi-AGVs:

1) All AGVs are homogeneous with constant velocity. We do not consider vehicle acceleration, deceleration or turning factor.
2) Each AGV can only carry one package in each delivery, and each passable block can be occupied by at most one AGV in a time slot.
3) It will take one time slot to dump (or offload) one package into export hole. After that, the AGV will return back for the next delivery.
4) If AGV fails to find a valid path in the loading point, it will wait until the path blocks are all available. A special case occurs when AGV have just finished package delivery and find no valid path, it shall go to buffer blocks to avoid congestion until there is valid path with all blocks available.

4 Algorithm Design

The notations to be used in this paper are listed in Table 1.

Table 1. Notations

Notation	Definition
T	The total running time slots, for each $t \in \{1, 2, ..., T\}$
K	Set of AGVs, for each $k \in K$
S_k	Starting block of AGV k in current scheduling
D_k	Destination block of AGV k in current scheduling
B_{pass}	Set of all passable blocks in the layout
$B_{succ}(i, p_k)$	Successor block of block i on path p_k
$B_{pre}(i, p_k)$	Precursor block of block i on path p_k
$w_{i,m}$	The $\boldsymbol{m}$th free time window for block i to pass through, $w_{i,m} = [t^s_{i,m}, t^e_{i,m}]$
$t^s_{i,m}, t^e_{i,m}$	The start and end time of the free time window $w_{i,m}$
$FTW_i(t)$	Set of free time windows of block i in t, $FTW_i(t) = \{w_{i,1}, w_{i,2}, ..., w_{i,m}\}$
$RFTW_{i,j}(t)$	Set of reachable free time windows from block i to block j in time slot t
$Sch(p_k)$	Schedule path of AGV k, $Sch(p_k) = ((S_k, t_{S_k}), ..., (i, t_i), (j, t_j), ..., (D_k, t_{D_k}))$
(i, t_i)	Tuple of block i and its corresponding entry time slot t_i

4.1 Overview of Our Algorithms

The objective of AGV path planing is to maximize the auto-sorting throughput, which is defined as the total number of successful deliveries of express packages into the export holes within given period. In real scheduling, there may exist collisions when more than one AGVs wanting to pass through the same block at the same time. To avoid this, the path planned for each AGV involves not only the blocks to move along, but also the corresponding passable time slot of each block in the path, which is defined in the following:

Definition 1: Schedule-Path ($Sch(p_k)$) is the routing path of AGV k when its start block and destination block are given. It can be described by a sequence of passable block-time pairs $((S_k, t_{S_k}), ..., (i, t_i), (j, t_j), ..., (D_k, t_{D_k}))$. Note that, here the *destination block* refers to the nearest passable block that can dump packages into the export hole.

To enhance sorting throughput, the AGVs will constantly be scheduled back and forth to make deliveries between start blocks and destination blocks. To simplify this problem, we decompose one cycle of AGV delivery (from start block to destination and then return back) into two equivalent single-directed schedules, each having different start blocks and destination blocks, which can be planned separately. The start block and destination block will be updated after successfully delivering the package or returning back to the original start block. Thus, we only focus on the single-directed schedules of each AGV in our path planning, which can be solved in three steps as follows:

Step 1: Generating candidate paths before scheduling each AGV.
Step 2: Calculate the *schedule-path* for each candidate path by searching the passable time windows of the blocks in the path.
Step 3: Select the earliest arrival path and make reservation.

Algorithm 1: Path Planning for Single-directed Schedule of AGV

```
Input: The layout graph G, start time t of AGV k, its start block S_k and
       destination block D_k.
Output: Schedule-path for AGV k.
1  Initination: failNum ← 0; failPaths ← ∅; SchPaths ← ∅;
2  while failNum < maxFails do
3      Paths ← generateCandidatePaths (G, S_k, D_k, failPaths);
4      foreach p_k ∈ Paths do
5          twp_k ← timeWindowSearching (p_k, t);
6          if twp_k ≠ infeasibility then
7              SchPaths ← SchPaths ∪ twp_k;
8      if SchPaths ≠ ∅ then
9          currentRoute ← selectEarliesArrival(SchPaths);
10         makeReservation(currentRoute) ;
11         return currentRoute;
12     else
13         failNum ← failNum + 1;
14         failPaths ← failPaths ∪Paths;
15 return ∅;
```

Algorithm 1 shows the path planning of one AGV in a single-directed schedule. In fact, there may exist failure in time window searching for schedule-path from the generated candidate paths, so we set a maximum failure count $maxFails$ to repeat this process until finding feasible schedule-path(s) or all loops complete with no schedule-path found (Line2∼Line15). For each loop, we first generate a set of candidate paths (Line3). Then, calculate the schedule-path for each candidate path and add it to the schedule-path set (Line4∼Line7). If there are any schedule-path(s) found in this round of search, we select the one with earliest reaching time to the destination block, and then make reservation and output the best schedule-path (Line8∼Line11). Else, all candidate paths will be added to the set of failure path, so that the same path will not be used as candidate paths in the following loops of searching. If all loops of tryings fail, the output is empty (Line 15). In this condition, the AGV with no schedule-path will stop and wait in current block in time slot t. We will continue to search the schedule-path for this AGV with the same start and destination block in time slot $t + 1$.

4.2 Candidate Path Generation

To obtain the schedule-path for AGV, we first generate multiple candidate paths for AGV, based on which to make searching of the passable time windows. The main reason is that the shortest path may not always be the best one with earliest arrival time considering the time occupation of the blocks in the path by other AGVs through reservation in advance. Therefore, we propose a penalty based A* algorithm to generate candidate paths, as can be seen in Algorithm 2.

Algorithm 2: Generating Candidate Paths for AGV

Input: Layout Graph G, number of candidate paths N, start block S_k and destination block D_k of AGV k, and failed paths set $failPaths$.

Output: Set of candidate paths $Paths$.

1 **Initination:** $Paths \leftarrow \varnothing$; $failCount \leftarrow 0$;
2 **foreach** *block* $i \in B_{pass}$ **do**
3   weight(i) $\leftarrow 1$;
4 **while** $|Paths|<N$ && $failCount<maxFailCount$ **do**
5     $p_k \leftarrow$ generatePath($G, S_k, D_k, weight$); // Using A* Search
6   **if** $p_k \notin Paths \cup failPaths$ **then**
7         $Paths \leftarrow Paths \cup p_k$;
8         $failCount \leftarrow 0$;
9   **foreach** *block* $i \in p_k$ **do**
10             $\alpha \leftarrow$ random(0,1);
11   **if** $\alpha < penaltyRatio$ **then**
12   weight(i) $\leftarrow$ weight(i) $\times$ $penalty$;
13   **else**
14         $failCount \leftarrow failCount + 1$;
15 **return** $Paths$;

In this algorithm, we try to generate N candidate paths (Line4∼Line14). Each passable block in the graph has a weight, as initialized to be 1 in (Line2∼Line3). We use penalty to change the weight of each block so as to affect the path searching in A*, which always selects the blocks with lower weight to avoid repeated selection of the same blocks among different candidate paths. In general, the weight of the blocks appearing in multiple paths will be more likely to have higher weight. For each generated candidate path, the weight of its blocks will be increased by multiplying a penalty value with probability of *penaltyRatio* between 0 and 1, as shown in (Line9∼Line12). It also means that the percentage of the number of the blocks to be updated in this path accounts for *penaltyRatio* roughly. Sometimes, A* algorithm cannot find any path not in existing path set, so our algorithm will be completed when there are $maxFailCount$ searching failures happening in a consecutive way (Line4, Line13∼Line14).

4.3 Time Window Searching: F-TWS and RS-TWS

In this section, we will introduce two algorithms of time window searching based on the generated candidate paths to build schedule-paths for AGV, including: forward time window searching and re-selection time window searching.

A) Preliminaries

Definition 2: Free Time Window. In each time slot t, the reservations (occupations) of block i during the whole period is discrete, leaving disjoint time intervals free. We call the set of such intervals as free time windows of block i, denoted as $FTW_i(t)$, for each time window $w_{i,m} \in FTW_i(t)$.

Definition 3: Reachable Free Time Window. For block i in time slot $t \in w_{i,m}$, the reachable free time windows of $w_{i,m}$ is the set of time windows belonging to $FTW_j(t)$ that should satisfy:
1) block j is the successor block of i on the path.
2) $\exists w_{j,n} \in FTW_j(t)$ such that $t+1 \in w_{j,n}$ or $t^e_{i,m}+1 \in w_{j,n}$, where $t^e_{i,m}$ is the last time slot of $w_{i,m}$.

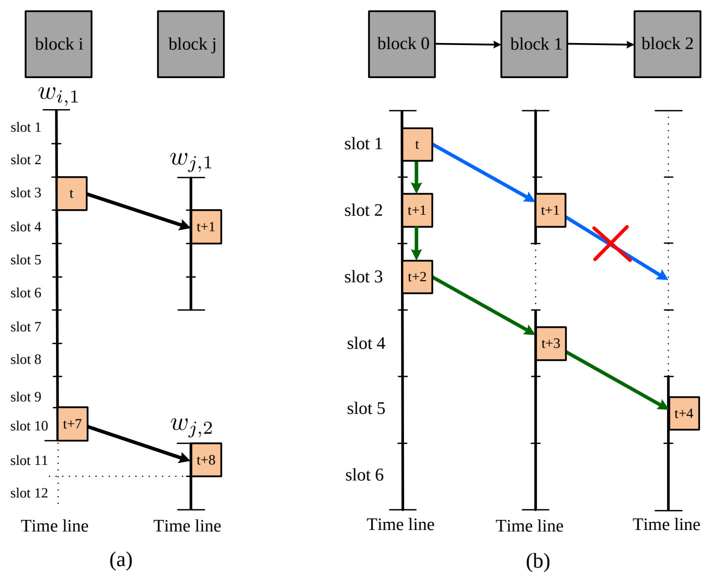

Fig. 2. (a) Reachable free time windows. (b) Example of time window search.

Fig. 2(a) is an example to show the reachable free time windows for block i in time slot t. We suppose block j is the successor of time slot i. The corresponding free time window of time slot t is $w_{i,1}$. Based on the definition of RFTW, there are totally two reachable free time windows of block i in time slot t:
1) Starting from $t \in w_{i,1}$, we can find a time window $w_{j,1}$, in which $t+1 \in w_{j,1}$.
2) Starting from $t+7 \in w_{i,1}$, we can find a time window $w_{j,2}$, in which $t+8 \in w_{j,2}$.
It's not difficult to find that the $RFTW_i(t)$ has the property of time continuity with the time window that contains t.

Fig. 2(b) is an example to show time window search process. Given the candidate path from block 0 to block 2, denoted as $p_k = (0, 1, 2)$, with start time $t = 1$. Our goal is to find a schedule-path with earliest arrival time. The dashed lines represent reserved time slots, which cannot be used in our schedule. The search fails when there is no reachable free time window to get to the next block, as shown in blue color. After waiting 2 time slots, the AGV k finds a feasible schedule-path with earliest arrival time. In this example, the schedule-path is:

$$Sch(p_k) = ((0,1),(1,4),(2,5))$$

B) Forward Time Window Searching (F-TWS)

Our forward searching algorithm is trying to get the schedule path with earliest arrival time through enumerating all combinations of reachable free time windows on the candidate path. The algorithm will always output the schedule-path once the optimum solution appears. This strategy searches from the starting block S_k with start time slot t_s. We push the tuple (S_k, t_s) to a queue, and then extend it with all its reachable free time windows on successor block in the path to make traverse in an enumerating way. We continue this process until getting to the destination block, or the queue is empty when the path is not feasible. The details of our forward searching algorithm is shown in Algorithm 3.

Algorithm 3: Forward Time Window Searching (F-TWS)

Input: Start time slot t, candidate path p_k to be scheduled
Output: Schdule-path $Sch(p_k)$
1 **Initination:** $Q \leftarrow \{(S_k, t)\}$;
2 **while** $Q \neq \varnothing$ **do**
3     $i \leftarrow \arg\min_i \{t_i \mid t_i \in Q\}$; $Q \leftarrow Q \setminus \{(i, t_i)\}$;
4   **if** $i == D_k$ **then**
5   **while** $i \neq S_k$ **do**
6             $Sch(p_k) \leftarrow Sch(p_k) \cup \{(i, t_i)\}$;
7             $(i, t_i) \leftarrow (i, t_i).parent$;
8         $Sch(p_k) \leftarrow Sch(p_k) \cup \{(i, t_i)\}$;
9   **return** $Sch(p_k)$;
10   **else**
11   **foreach** $w \in RFTW_{i,j}(t_i)$ **do**
12             $(j, t_j) \leftarrow (j, max(t_i + 1, w.t^s))$;
13             $(j, t_j).parent \leftarrow (i, t_i)$;
14             $Q \leftarrow Q \cup \{(j, t_j)\}$;
15 **return** $infeasibility$;

In this algorithm, we first initialize the queue Q with tuple of starting block S_k with start time t on candidate path p_k for this AGV k (Line1). We make searching by iteratively push and pop reachable free time windows of the time slots in current time window (Line2∼Line14). In each loop, we pop out the tuple (b_i, t_i) with earliest entry time t_i from the queue Q (Line3). To extend the free time windows on block i, all reachable free time window of this block will be added to the Q (Line14). Before this, the entry time of each reachable time window will be calculated (Line12). We record the parent block-time pair in each searching, so that the final schedule path will be output in front-to-back order (Line4∼Line9). If no schedule path is found, we output infeasibility.

C) Re-selection Time Window Searching (RS-TWS)

Definition 4: Weak Reachable Free Time Window. For block i in time slot $t \in w_{i,m}$, the weak reachable free time windows of $w_{i,m}$ is the set of time windows belonging to $FTW_j(t)$ of successor block j on the condition that: $t+1 \leq t^e_{j,n}$, denoted as $WRFTW_{i,j}(t)$.

Definition 5: Inconsistent Tuple. For schedule-path $Sch(p_k)$, two consecutive tuples $(i,t_i),(j,t_j)$ are inconsistent when $t^e_{i,m}+1 < t^s_{j,n}(t_i \in w_{i,m}, t_j \in w_{j,n})$.

The main idea of re-selection time window searching algorithm is to constantly adjust inconsistent tuple entry time on schedule-path p_k until no tuples are inconsistent or the inconsistency takes place between the first and second tuple in the schedule-path, which means the schedule-path is not feasible.

Algorithm 4: Re-selection Time Window Searching (RS-TWS)

Input: start time slot t, path p_k to be scheduled
Output: Schdule-path $Sch(p_k)$
1 $Sch(p_k) \leftarrow Sch(p_k) \cup \{(S_k, t)\}$;
2 $i \leftarrow S_k$; $j \leftarrow B_{succ}(i, p_k)$;
3 **while** $j \neq D_k$ **do**
4     $(j,t_j) \leftarrow (j, max(t_i+1, t^s_{j,n}))$; $//w_{j,n}$ is the first of $WRFTW_{i,j}(t_i)$.
5     $Sch(p_k) \leftarrow Sch(p_k) \cup \{(j,t_j)\}$;
6     $i \leftarrow j$; $j \leftarrow B_{succ}(j, p_k)$;
7 $(j,t_j) \leftarrow (j, max(t_i+1, t^s_{j,n}))$; $Sch(p_k) \leftarrow Sch(p_k) \cup \{(j,t_j)\}$;
8 **while** $\exists inconsistent\ tuple$ **do**
9     $(j,t_j) \leftarrow$ first inconsistent tuple from $Sch(p_k)$;
10   **if** $j \neq S_k$ **then**
11         $(j,t_j) \leftarrow (j, max(t_i+1, t^s_{j,n+1}))$; // $i \in B_{pre}(j,p_k)$, old $t_j \in w_{j,n}$, new $t_j \in w_{j,n+1}$ which follows $w_{j,n}$.
12   **while** $B_{succ}(j,p_k) \neq D_k$ **do**
13             $h \leftarrow B_{succ}(j,p_k)$;
14             $(h,t_h) \leftarrow (h, max(t_j+1, t^s_{h,m}))(w_{h,m} \in RFTW_{j,h}(t_j))$;
15             $j \leftarrow h$; $h \leftarrow B_{succ}(h,p_k)$;
16         $h \leftarrow B_{succ}(j,p_k)$; $(h,t_h) \leftarrow (h, max(t_j+1, t^s_{h,m}))$;
17   **else**
18   **return** $infeasibility$;
19 **return** $Sch(p_k)$;

In Algorithm 4, we initialize the schedule-path $Sch(p_k)$ by selecting the first weak reachable free time window on each block in the path and calculate its entry time (Line1∼line7). For the initialized $Sch(p_k)$, we check whether there exists inconsistent tuple (Line8). The schedule is feasible if no inconsistent tuple is found, then we output $Sch(p_k)$ (Line19). Otherwise, we retrieve the first inconsistent tuple (j,t_j) and check if it belongs to start tuple in $Sch(p_k)$ (Line9). The algorithm outputs infeasibility when j is the start block S_k. If j belongs to other blocks on p_k, we change the entry time by selecting its next weak reachable free time window and update t_j (Line11). Then start from j, we update the entry time of blocks following j through re-selection of reachable free time window till the end of path (Line12∼Line16).

5 Experiment

In this section, we compare our two approaches with HTN based method and TW-A* in different warehouse layout settings and different plant scales. We make evaluations of our methods using two merics: the sorting throughput in given time period, and the number of failures path finding.

5.1 Experiment Setup

In fact, the layout of warehouse and plant scale will grealy affect the schedule of AGVs, so that sorting throughput can quite different under different layout settings. In this paper, we designed four types of warehouse layout with scale changing from 10×10 to 64×64 blocks. Fig. 3 shows our layouts design with scale of 10×10 blocks, base on which we validate effectiveness of our approaches.

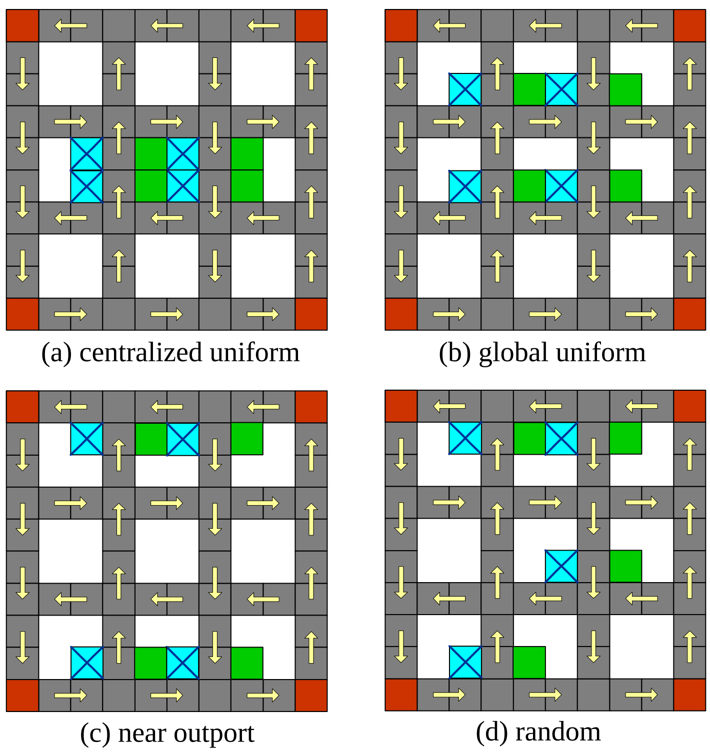

Fig. 3. Four types of warehouse layout design.

In layout (a), all sorting exports are centralized and uniformly distributed in the central part of warehouse, so that the distance from starting ports to sorting exports is basically equal. In layout (b), sorting exports are evenly distributed in the warehouse plant. For layout (c), sorting exports are set according to the principle of near starting ports. In layout (d), all sorting exports are randomly generated in the warehouse filed.

5.2 Experiment Results

Fig. 4 shows the sorting throughput under the above four different layouts of the warehouse within given period. Generally speaking, our two algorithms outperform HTN and TW-A* no matter what the given layout is degined. It can be clearly seen that with the growing of AGV number, the sorting throughput increases correspondingly and there is an obvious gap in throughput between our algorithms and HTN, TW-A*. The zoom views in the middle give us a more clear vision of the growing trend in throughput when the number of AGVs increases from 10 to 200. From the zoom views, we can conclude that the minimum number of AGVs required to achieve maximum throughput is roughly between 50 and 60. The throughput will keep fluctuating smoothly even when AGV increases to 2000. It is because that the warehouse plant scale restricts the number of active AGVs, and extra vehicles can no longer be scheduled to transport more packages in limited space like this.

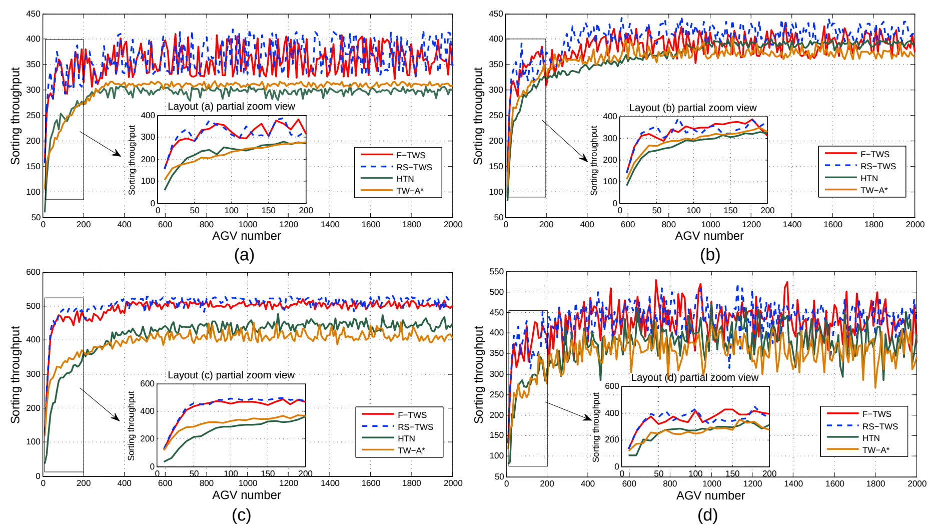

Fig. 4. The sorting throughput under various layouts.

Fig.5 depicts the number of failures in finding valid paths. It can seen that in each layout, our approaches perform better than existing methods with much fewer failures to find feasible paths. Our methods generate a set of candidate paths used to find the schedule-path with earliest arrival time window, decreasing the chance of failures in path searching. In each layout, the number of failures in our methods will reach a certain peak when AGV number increases to 50 or 60. When AGV number increases to a certain extent, the number of active ones will no longer increase due to the limited warehouse space, so that the number of failures that are caused by active AGVs will no longer change greatly.

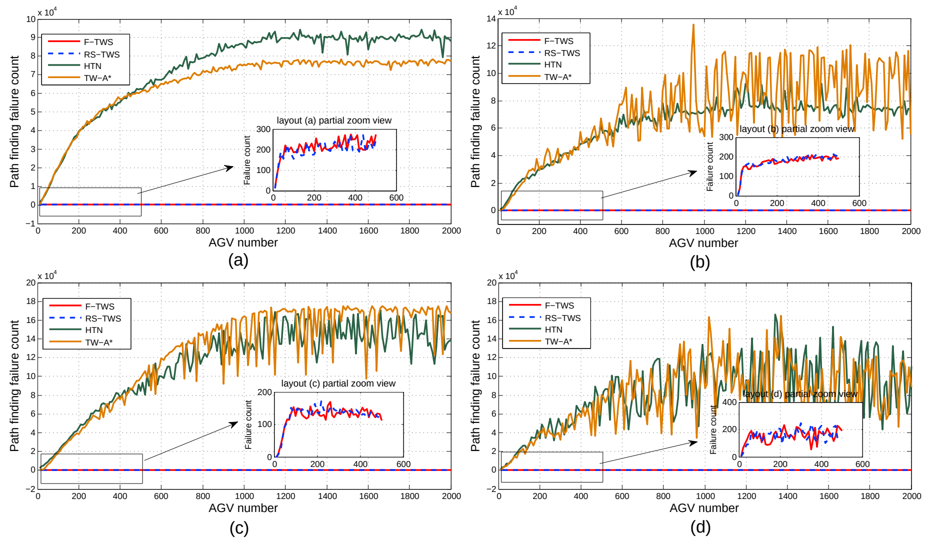

Fig. 5. The number of failures in AGV path searching.

Fig. 6(a) shows how sorting throughput changes with layout scale. In our settings, the running plant scale grows from 10×10 to 64×64 blocks, with corresponding increase of the number of starting ports and sorting exports. In this Figure, the sorting throughput increase slowly with the growing of plant scale. In larger running plant, AGVs need to travel longer distance to the sorting destinations, so that more AGVs will be scheduled to transport more packages in the same period. Fig. 6(b) shows the maximum number of AGVs that can be accommodated in different plant scale for layout (a). It can found that both the maximum number of AGVs and the maximum sorting throughput are almost in linear relationship with plant scale, as can be seen in Fig. 6(a) and Fig. 6(b).

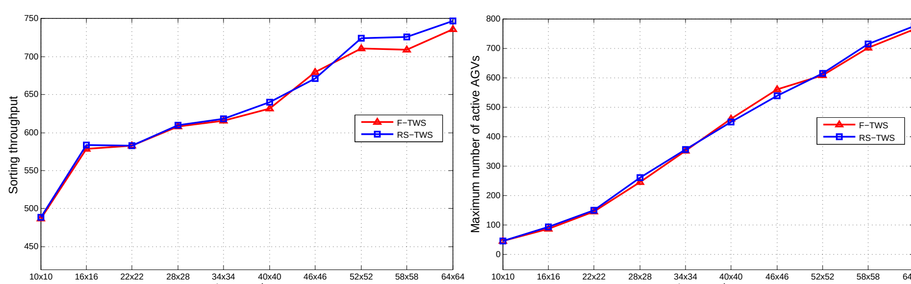

Fig. 6. (a) Throughput with plant scale. (b) Maximum number of active AGVs with plant scale.

6 Conclusion

In this paper, we proposed two heurist algorithms F-TWS and RS-TWS to search for passable time windows along candidate path, so that the sorting throughput can be maximized in logistics center of express company. Compared with existing work such as HTN and TW-A*, our approaches significantly promote sorting throughput even under different layouts of warehouse plant while with less failures in path finding. With the plant scale grows, the sorting throughput and maximum number of active AGVs will increase. In future work, the problem of multi-destination path planning should be considered, in which AGVs are capable of carrying multiple express packages, and our approaches can be extended to cope with the problem in future intelligence logistics center.

Acknowledgment

This work was financially supported by National Natural Science Foundation of China with Grant No.11371004 and No. 61672195, National Key Research and Development Program of China with Grant No. 2016YFB0800804 and No. 2017YFB0803002, and Shenzhen Science and Technology Plan with Grant No. JCYJ20160318094336513, No. JCYJ20160318094101317 and No. KQCX20150-326141251370.

References

1. Ratner, D., Warmuth, M.K.: Finding a shortest solution for the n× n extension of the 15-puzzle is intractable. In: AAAI. (1986) 168–172
2. Surynek, P.: An application of pebble motion on graphs to abstract multi-robot path planning. In: Tools with Artificial Intelligence, 2009. ICTAI'09. 21st International Conference on, IEEE (2009) 151–158
3. Digani, V., Sabattini, L., Secchi, C., Fantuzzi, C.: Ensemble coordination approach in multi-agv systems applied to industrial warehouses. IEEE Transactions on Automation Science and Engineering **12**(3) (2015) 922–934
4. Olmi, R..: Coordination of industrial agvs. International Journal of Vehicle Autonomous Systems **9**(1/2) (2011) 5–25
5. Simeon, T., Leroy, S., Lauumond, J.P.: Path coordination for multiple mobile robots: a resolution. Robotics & Automation IEEE Transactions on **18**(1) (2002) 42–49
6. Wang, C., Wang, L., Qin, J., Wu, Z., Duan, L., Li, Z., Cao, M., Ou, X., Su, X., Li, W.: Path planning of automated guided vehicles based on improved a-star algorithm. In: IEEE International Conference on Information and Automation. (2015) 2071–2076
7. Zeng, S., Zhu, Y., Qi, C.: Htn-based multi-robot path planning. In: Control and Decision Conference. (2016) 4719–4723
8. Park, B., Choi, J., Wan, K.C.: An efficient mobile robot path planning using hierarchical roadmap representation in indoor environment. In: IEEE International Conference on Robotics and Automation. (2012) 180–186

9. Habib, D., Jamal, H., Shoab, A.: Employing multiple unmanned aerial vehicles for co-operative path planning. International Journal of Advanced Robotic Systems **10**(3) (2013) 1
10. Zheng, K., Tang, D., Gu, W., Dai, M.: Distributed control of multi-agv system based on regional control model. Production Engineering **7**(4) (2013) 433–441
11. Wagner, G., Choset, H.: Subdimensional expansion for multirobot path planning . Artificial Intelligence **219**(C) (2015) 1–24
12. Zhang, W., Kamgarpour, M., Sun, D., Tomlin, C.J.: A hierarchical flight planning framework for air traffic management. Proceedings of the IEEE **100**(1) (2012) 179–194
13. Dewilde, B., Mors, A.W.T., Witteveen, C.: Push and rotate: a complete multi-agent pathfinding algorithm. Journal of Artificial Intelligence Research **51**(1) (2014) 443–492
14. Wang, W., Goh, W.B.: Spatio-temporal a* algorithms for offline multiple mobile robot path planning. In: The International Conference on Autonomous Agents and Multiagent Systems. (2011) 1091–1092
15. Wang, W., Goh, W.B.: Time optimized multi-agent path planning using guided iterative prioritized planning. In: International Conference on Autonomous Agents and Multi-Agent Systems. (2013) 1183–1184
16. Chen, Y., Bell, M.G.H., Bogenberger, K.: Reliable pretrip multipath planning and dynamic adaptation for a centralized road navigation system. IEEE Transactions on Intelligent Transportation Systems **8**(1) (2007) 14–20
17. Yen, J.Y.: Finding the k shortest loopless paths in a network. Management Science **17**(11) (1971) 712–716
18. Suurballe, J.W., Tarjan, R.E.: A quick method for finding shortest pairs of disjoint paths. Networks **14**(2) (2010) 325–336
19. Delling, D., Wagner, D.: Pareto paths with sharc. In: International Symposium on Experimental Algorithms. (2009) 125–136
20. Bader, R., Dees, J., Geisberger, R., Sanders, P.: Alternative route graphs in road networks. In: International ICST Conference on Theory and Practice of Algorithms in. (2011) 21–32