Proceedings of the ASME 2022
International Mechanical Engineering Congress and Exposition
IMECE2022
October 30-November 3, 2022, Columbus, Ohio

IMECE2022-95364

AN INTEGRATED TASK AND PATH PLANNING APPROACH FOR MOBILE ROBOTS IN SMART FACTORY

Shuo Liu[1,2] **Bohan Feng**[1] **Dan Yu**[2] **Youyi Bi**[1*]

[1] University of Michigan – Shanghai Jiao Tong University Joint Institute
Shanghai Jiao Tong University
Shanghai, China

[2] College of Astronautics
Nanjing University of Aeronautics and Astronautics
Nanjing, China

ABSTRACT

Mobile robots are being widely used in smart manufacturing, and efficient task assignment and path planning for these robots is an area of high interest. In previous studies, task assignment and path planning are usually solved as separate problems, which can result in optimal solutions in their respective fields, but not necessarily optimal as an integrated problem. Meanwhile, precedence constraints exist between sequential processing operations and material delivery tasks in the manufacturing environment. Thus, those planning methods developed for warehousing and logistics may not simply apply to the environment of smart factories. In this paper, we propose an integrated task and path planning approach based on Looking-backward Search Strategy (LSS) and Regret-based Search Strategy (RSS). In the stage of task assignment, the real paths for mobile robots are identified based on the Cooperative A (CA*) algorithm and the time and energy consumed by mobile robots and machining centers are calculated. Then a greedy strategy working with LSS or RSS is used to search reasonable task assignments in time-series, which can generate a joint optimal solution for both task assignment and path planning. We verify the validity of the proposed approach in a simulated smart factory and the results show that our approach can improve the operation efficiency of the smart factory and save the time and energy consumption effectively.*

Keywords: Task assignment, Path planning, Integrated planning, Energy consumption, Mobile robot

NOMENCLATURE

$T = \{1, \dots, n\}$	a set of n tasks
$type_i$	the type of task $i \in T$
sp_i	the starting position of task $i \in T$
gp_i	the goal position of task $i \in T$
st_i	the starting time of task $i \in T$
gt_i	the finishing time of task $i \in T$
$parent_i$	the parent task of task $i \in T$
$M_i = \{1, \dots, m\}$	a set of m optional machining centers for task $i \in T$
$R = \{1, \dots, a\}$	a set of a robots
s_j	the starting position of robot $j \in R$
$P = \{sp_1, \cdots, sp_n, gp_1, \cdots, gp_n\}$	all possible positions that robot j can arrive at
$TA_{jp}: R \times P$	the task assignment mapping table to describe whether robot j need to arrive at position p or not
$MT_{kt}: M \times Time$	the processing mapping table to describe whether machining center k is processing task i at time t or not
c_j^t	the number of loading tasks of robot j at time t
$loc_j(t)$	the location of robot j at time t
t_j	the time spent by robot j to transport all tasks assigned to it

[*] Corresponding author, Assistant Professor in Mechanical Engineering, Shanghai Jiao Tong University
Email: youyi.bi@sjtu.edu.cn

t_k	the time spent and energy consumed by machining center k after completing all tasks assigned to it
e_k	the energy consumed by machining center k after completing all tasks assigned to it
$l\ (l \geq 1)$	the lth layer of the task queue
PP	the maximum transportation time
PT	the maximum processing time
PE	the total processing energy
t_j^i	the time when robot j finishes transporting task i
t_{cur_max}	the total time consumed by machining center k to complete the last task that has been assigned up to now
Δt_i	the search space of inserting task $i \in T$
TC	total time and energy consumption

1. INTRODUCTION

A smart factory usually owns a series of machining centers, industrial robots, storage racks and mobile robots. The mobile robots can assist industrial robots and machining centers with complex manufacturing jobs by delivering raw materials and parts, and inspecting the status of the production lines [1]. As the number of robots and tasks grows, the scheduling and planning of these robots can be complicated. Therefore, it is necessary to investigate how to efficiently assign and schedule the mobile robots to transport the materials between machining centers and storage racks with the goal of minimizing the makespan (i.e., the time consumed in transportation and processing) and/or the amount of consumed energy. Moreover, the constraints of temporal precedence may exist between sequential manufacturing processes and material delivery operations (e.g., picking up, delivering, processing, and storing). Thus, we define this problem as the precedence constrained multi-agent task assignment and path-finding (PC-TAPF) problem [2].

In a PC-TAPF problem, a set of tasks and a team of mobile robots are usually given. We first need to assign each task to a suitable robot [3]. Then we need to find a set of conflict-free paths for robots to ensure that the assigned tasks can be successfully completed [4]. Note that precedence constraints can exist between tasks in a PC-TAPF problem. For example, both task A and task B must be completed before task C is started, and the initial position of task C can only be determined once the target positions of task A and task B have been determined in the scenario of flowline manufacturing [2]. Therefore, it is not appropriate to simply apply the task assignment and path planning algorithms developed for warehousing and logistics into smart factories considering the precedence constraints of transportation tasks are different in these environments.

Various approaches have been developed to solve PC-TAPF problems [5–8]. These approaches often resort to solve the task assignment and path planning separately. The common procedure is to generate all possible assignments and then find a feasible path for each assigned task. However, many of these approaches either suffer from high complexity in computation which leads to failed deployment in practice, or over simply assume the solved paths will not conflict with each other no matter how the tasks are assigned [9–11].

In recent years, approaches that jointly solve task assignment and path planning are emerging. For instance, the CBM (Conflict-Based Min-Cost-Flow) and CBS-TA (Conflict-Based Search with Optimal Task Assignment) algorithms [12,13] can find makespan-optimal solutions to task assignment and path planning. Brown et al. [2] proposed a four-level hierarchical algorithm for computing makespan-optimal solutions to PC-TAPF problems. However, many of these algorithms are limited by poor scalability and timeout failures can happen when the number of agents and tasks becomes relatively large. In addition, they usually only consider makespan as the single optimization objective.

Other new methods have been proposed recently to solve TAPF centrally and adopted a sequential two-stage method which performs task assignment first then followed by path planning in an integrated way. For instance, Chen et.al. [14] designed an integrated method where task assignment choices are chosen by actual delivery costs. The actual path cost is considered when assigning tasks to agents for improving the quality of the task assignment. However, these methods do not consider the precedence constraints between tasks and may not apply to the scenario of smart factories.

To make up for these shortcomings, we propose an integrated task and path planning approach for mobile robots in smart factory in this paper. This approach can solve task assignment and path planning in a joint way while considering the precedence constraints of tasks and conflict-free constraints of paths. The core idea of the approach is the Looking-backward Search Strategy (LSS) and Regret-based Search Strategy (RSS) proposed in the process of task assignment, which can help with reducing the total operating time and the consumed energy of machining centers as much as possible. Our approach preliminarily solves the coupling of task assignment and path planning with improving the operational efficiency and saving energy consumption of the smart factory. It can contribute to the practical deployment of multi-mobile robot systems in smart factories and promote the development of more advanced high-efficient and energy-saving manufacturing modes.

This paper is structured as follows. Section 2 introduces our integrated task and path planning approach. Section 3 presents a simulation experiment validating the proposed approach and discusses the experiment results. Section 4 provides a summary of our work and suggests the future research directions.

2. METHODS

2.1 Problem Formulation

The PC-TAPF problem in a smart factory environment is defined as how to optimally assign tasks with precedence constraints and generate conflict-free paths for each moving robot. In our study, the environment is represented as a grid map consisting of cells with unit length, and an index incrementally numbered from left to right and top to down is used as the

position coordinate of each cell. For example, in a 5×10 cells grid, the upper left corner cell is indexed with 1, while the lower right corner one is with 50. Thus, in the path finding, graph-based search methods such as CA* [15] can be adopted.

We use $T = \{1, \dots, n\}$ to represent a set of tasks in the smart factory in which the cargo (e.g., materials or parts) are transported from starting positions to designated positions. Each task $i \in T$ has a given tuple with seven attributes, $(type_i, sp_i, gp_i, st_i, gt_i, parent_i, M_i)$. $type_i$ is the type of task i, and $type_i = 0$ represents that in task i the cargo will be transported from a storage zone or machining center to another machining center, while $type_i = 1$ represents that the cargo will be transported from a machining center to a storage zone. sp_i and gp_i are the starting and goal positions of task i, respectively, which equals -1 when the starting or goal position is unknown. st_i and gt_i are the starting and finishing time of task i, respectively, which equals -1 when either one is unknown. $parent_i$ is the parent task of task i, and it must be completed before starting task i. $M_i = \{1, \dots, m\}$ is a set of optional machining centers for task i when operating task i. $R = \{1, \dots, a\}$ represents the set of robots for transporting tasks. s_j denotes the starting position for each robot $j \in R$. The starting positions of all robots are randomly assigned in the docking zone at the beginning. In this study, the robots are assumed to be able to turn around in place, thus robot heading is not considered in task assignment.

In order to transport task i, robot j should first move to the starting position sp_i of the task i and then transport the cargo to the goal position gp_i. During this period, time is set to be discretized into unit time steps, and a robot can move over one cell in one time step. In the path finding process, two types of collisions need to be avoided: vertex collision and edge collision. The former refers to that two robots should not occupy the same cell at the same time, and the latter means that two robots should not move along adjacent cells in opposite directions at the same time.

Let $P = \{sp_1, \cdots, sp_n, gp_1, \cdots, gp_n\}$ be all possible positions that robot j can arrive at. $TA_{jp}: R \times P \to \{0, 0.5\}, j \in R, p \in P$ represents the task assignment mapping table that maps the indices of robot j and the loading or unloading position p to a fixed value, which equals 0.5 if and only if robot $j \in R$ has to reach the position $p \in P$. $TA_{jsp_i} + TA_{jgp_i} = 1$ means that task i is assigned to robot j. c_j^t denotes the number of loading tasks of robot j at time t. $loc_j(t)$ denotes the location of robot j at time t. t_j denotes the time spent by robot j to transport all the tasks assigned to it. t_k and e_k denote the time spent and energy consumed by machining center k after completing all tasks assigned to it, respectively.

The problem formulation is given in Equations (1) – (14). In Equation (1), PP is the maximum time spent by all robots transporting all tasks, i.e., the maximum transportation time. In Equation (2), PT is the maximum time spent by all machining centers to process all tasks, i.e., the maximum processing time. In Equation (3), PE is the total energy consumed when machining centers are operating, i.e., the total processing energy. The overall optimization goal is to simultaneously minimize PP, PT and PE subject to the constraints in Equations (4) - (14).

$$PP = \max_{j \in R} t_j \tag{1}$$

$$PT = \max_{k \in M} t_k \tag{2}$$

$$PE = \sum_{k=1}^{m} e_k \tag{3}$$

subject to

$$TA_{jsp_i} + TA_{jgp_i} \in \{0,1\}, \forall i \in T, \forall j \in R \tag{4}$$

$$\sum_{j=1}^{a} TA_{jsp_i} + TA_{jgp_i} \in \{0,1\}, \forall i \in T \tag{5}$$

$$c_j^t \in \{0,1\}, \forall t, \forall j \in R \tag{6}$$

$$loc_j(t) \neq loc_{j'}(t), \forall j、 j' \in R, j \neq j', \forall t \tag{7}$$

$$\{loc_j(t), loc_j(t+1)\} \neq \{loc_{j'}(t+1), loc_{j'}(t)\}$$
$$\forall j、 j' \in R, j \neq j', \forall t \tag{8}$$

$$gp_i = -1, \forall i \in layer_1 \cap T \; s.t. \; type_i = 0 \tag{9}$$

$$gp_i \geq 0, \forall i \in layer_1 \cap T \; s.t. \; type_i = 1 \tag{10}$$

$$sp_i \geq 0, st_i = 0, gt_i = -1, parent_i = \phi$$
$$\forall i \in layer_l, l = 1 \tag{11}$$

$$sp_w \geq 0, sp_{w'} = -1$$
$$\exists w, w' \in layer_l, l \geq 2, w \in T, w \neq w', \forall type_{parent_{w'}} = 0 \tag{12}$$

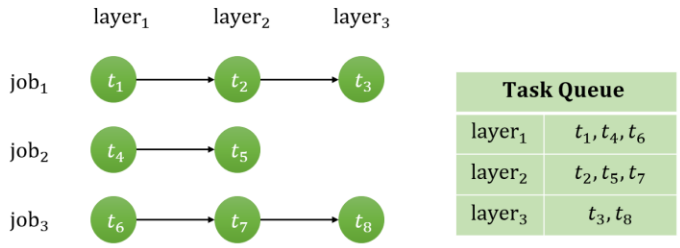
$$st_w = -1, \forall w \in layer_l, l \geq 2, w \in T \tag{13}$$

$$parent_w = i, \exists i \in layer_{l-1}, \forall w \in layer_l, l \geq 2, w \in T \tag{14}$$

Equation (4) indicates that the loading and unloading process of a task is done by the same robot; (5) means that a task can only be transported by exactly one robot; (6) implies that each robot is capable of transporting at most one task at a time, i.e., a robot cannot transport multiple tasks simultaneously; (7) implies there is no vertex collisions between robots; (8) implies there is no edge collision between robots. Equations (9) – (14) describe the constraints of certain task attributes when considering the precedence of tasks, and the details are described below.

Figure 1 shows a task queue organized in three layers consisting of eight tasks with precedence constraints. We use l $(l \geq 1)$ to describe the lth layer of the task queue, and each layer is a set of tasks in which tasks in the same layer do not need to follow specific sequences, while tasks in two consecutive layers need to be carried out one after another. For example, task t_2 must be carried out after t_1 is completed, while task t_1 and

t_4 can be started at the same time if their needed resources (e.g., robots) are available. Obviously, tasks in layer$_1$ (base layer, $l = 1$) do not own any parent tasks, while tasks in other layers ($l \geq 2$) should have parents.



Task Queue	
layer$_1$	t_1, t_4, t_6
layer$_2$	t_2, t_5, t_7
layer$_3$	t_3, t_8

FIGURE 1: A TASK QUEUE ORGANIZED IN THREE LAYERS CONSISTING OF EIGHT TASKS.

Equations (9) and (10) indicate that the goal position is unknown for task i when $type_i = 0$ and is known for task i when $type_i = 1$. Equation (11) indicates that for all tasks in the base layer ($l = 1$), their starting positions should be known, their starting time are set to 0, their finishing time are unknown initially, which can only be calculated when these tasks are assigned, and all tasks in the base layer are pioneers without parent tasks.

Tasks in layer l ($l \geq 2$) need to be assigned after the completion of their corresponding parent tasks in layer $l-1$. This means that the attributes of tasks in layer l are not exactly the same as those in layer $l-1$ due to the precedence constraints. Equation (12) describes that the starting positions of some tasks are known, while others are unknown. In the latter case, the starting position of the current task depends on the goal position of its corresponding parent task. That is, the starting position of this current task will be known only after its parent task i with $type_i = 0$ is assigned. Likewise, Equation (13) requires that task can be started only after its corresponding parent task is completed. Equation (14) represents a task in layer l ($l \geq 2$) has one parent task when the jobs have not been completed at the layer $l-1$.

2.2 Integrated Task and Path Planning

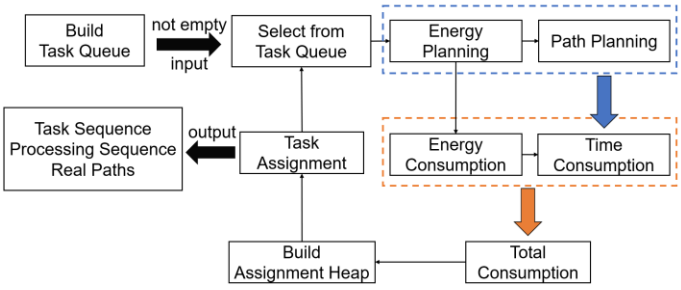


FIGURE 2: THE OVERALL FRAMEWORK OF INTEGRATED TASK AND PATH PLANNING APPROACH.

Figure 2 shows the overall framework of the proposed approach. The two thick black arrows indicate the input and output of our approach. The blue dashed box represents the energy planning and path planning for the selected task, from which we can get the time consumed by a certain robot for transportation and the time and energy consumed by a certain machining center for processing task. The orange dashed box represents the weighted sum of the time and energy consumption. The tasks are first organized into a queue by layers according to their precedence constraints as shown in Fig. 1. Then tasks are selected from the built task queue in increasing order of precedence layers, i.e., tasks from base layer are extracted first. For each selected task i, energy planning is first performed by traversing all possible machining centers that are capable of processing task i. For each possible machining center k, its position is set as the goal position of task i. The energy consumed by the machining center k and the machining time can be obtained from initial settings. Then the CA* [15] is used to generate a reasonable path for robot j ensuring that no collision occurs with other planned paths.

The priority order for path planning is determined by the task assignment sequence. The time consumption and the energy consumption for task processing and transportation are taken as the total consumption and stored in the Assignment Heap H in increasing order (i.e., the assignment with least total consumption is at the top of the heap). Assignment Heap H contains all potential assignments of task i to each available robot and machining center when $type_i = 0$. The greedy algorithm is then used to select the optimal task assignment from the Assignment Heap H and the loop cycle keeps continuing until all tasks are successfully assigned. In order to reduce the unnecessary waiting time for machining centers in task assignment, we propose the Looking-backward Search Strategy (LSS) and Regret-based Search Strategy (RSS), which are explained in the following subsections.

2.2.1 Looking-backward Search Strategy (LSS)

Traditionally, a task will be assigned to a machining center at the time point right after the last task assigned to this machining center. This treatment can lead to a waste of time and energy consumption due to unnecessary waiting time since the time periods before the last assigned task might be free for inserting the current task. Thus, we propose the Looking-backward Search Strategy (LSS) to reduce the operating time. The basic idea is to search the available time period (i.e., looking backward) before the last assigned task and try to identify whether the current task can fit in the identified time period.

Algorithm 1 shows the pseudo-code for updating processing sequence via LSS. Let $MT_{kt}: M \times Time \rightarrow \{0, i\}, k \in M, t \in Time, i \in T$ represents the processing mapping table, which equals i if and only if the machining center k is processing task i at time t. $MT_{kt} = 0$ implies that machining center k is idle at time t. Let t_j^i denote the time when robot j finishes transporting task i to the machining center k, and t_{cur_max} represent the current total time consumed by machining center

k to complete the last task that has been assigned up to now. Equation (15) describes Δt_i, the search space of inserting task i (i.e., the feasible time periods), when the looking-backward search is performed.

$$\Delta t_i = \left(t_j^i, t_{cur\_max}\right) \tag{15}$$

Algorithm 1 Update Processing Sequence via LSS

Input: current task i, robot j, machining center k
Output: processing mapping table MT_{kt}

Initialize $count = 0$
for all $t \in \left(t_j^i, t_{cur\_max}\right)$ **do**
  if $MT_{kt} == 0$ **then**
    $count++$
    if $count == t_k^i$ **then**
      for all $t' \in \left(t, t + t_k^i - 1\right)$ **do**
        $MT_{kt'} = i$
      end for
    end if
  end if
  if $MT_{kt}! = 0$ **then**
    $count = 0$
  end if
end for

Specifically, a counter function $count$ is to keep track of the time periods when the machining center k is idle. Here the greedy search strategy is used, which means that once we find the first feasible time period $insert$, the search will stop and the identified period will be the time period for machining center k to process task i.

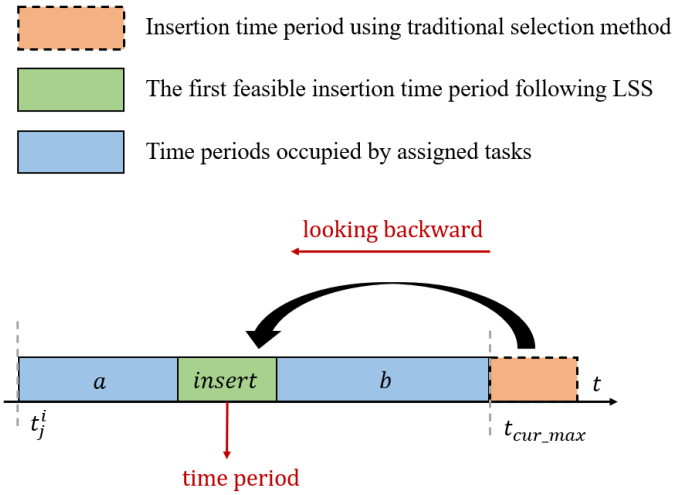


FIGURE 3: AN ILLUSTRATION ABOUT THE LOOKING-BACKWARD SEARCH STRATEGY.

Figure 3 shows an example case of updating the task assignment based on LSS. The blue boxes represent the time periods occupied by assigned tasks (e.g., tasks $a, b \in T$) and they cannot be inserted with new tasks. The orange box represents the insertion time period using traditional selection method, while the green box represents the first feasible insertion time period following LSS. The search space of the insertion time is $\left(t_j^i, t_{cur\_max}\right)$. Obviously, the new insertion strategy can reduce the idle time for the machining center and the total time and energy consumption can be saved.

2.2.2 Regret-based Search Strategy (RSS)

Note that LSS will stop searching after finding the first feasible time period for current task i. This greedy strategy may prevent the next task i' from inserting backward to machining center k because the identified time period to process task i may partially interfere with the time period to process next task i'. To address this issue, we propose a Regret-based Search Strategy (RSS). The basic idea is that when we perform the task insertion of current task i, the algorithm leaves enough space for inserting the next task i', i.e., think one more step.

Algorithm 2 Update Processing Sequence via RSS

Input: current task i, next task i', robot j, machining center k
Output: processing mapping table MT_{kt}

Initialize $inserts = \phi$
for all $t \in \left(t_j^i, t_{cur\_max}\right)$ **do**
  $inserts \leftarrow$Find all feasible time periods of task i
end for
//regret
$inserts_{regret} \leftarrow$ Filter time periods
if $inserts_{regret} = \phi$ **then**
  $inserts_{final} \leftarrow \max(inserts)$
end if
if $inserts_{regret}! = \phi$ **then**
  $inserts_{final} \leftarrow \min\left(inserts_{regret}\right)$
end if
//insert
for all $t \in \left(inserts_{final}, inserts_{final} + t_k^i - 1\right)$ **do**
  $MT_{kt} = i$
end for

Algorithm 2 shows the pseudo-code for updating task assignment sequence via RSS. The search space is the same as Equation (16). The core ideas are as follows.

(1) We define an array $inserts = \{inserts_0, inserts_1, \ldots\}$ to save all feasible time periods that allow the insertion of task

i. Note that the time period $insert$ mentioned in Fig.3 is the same as $inserts$ when the size of $inserts$ is 1.

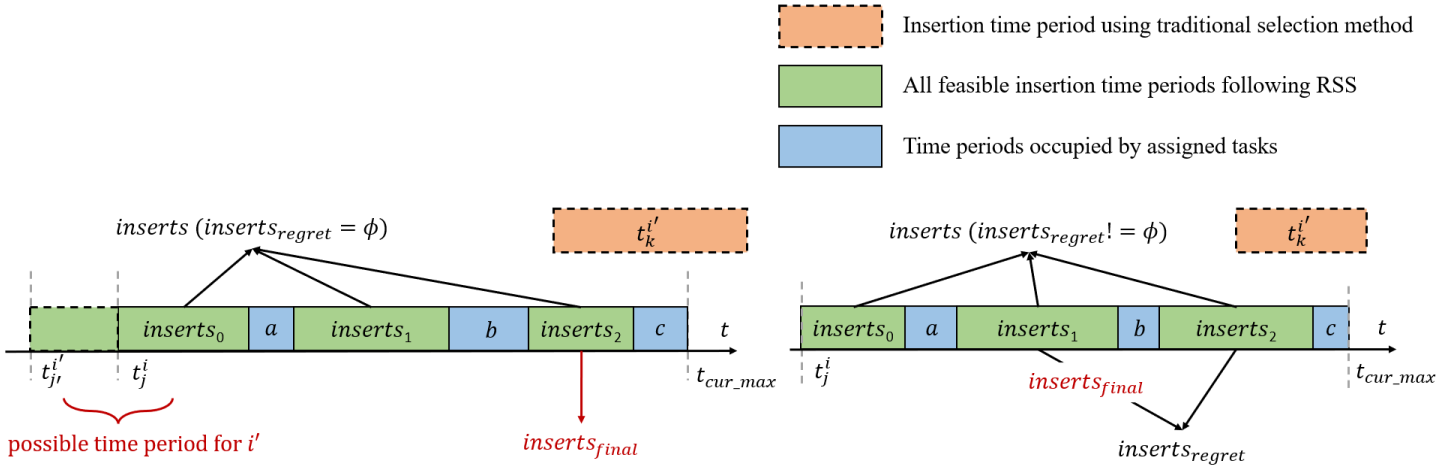
(2) Then the insertion of the next task i' is considered by filtering out the time periods that allow the insertion of task i, which can be accessed by the index $regret$.

(3) Finally, the final time period $inserts_{final}$ of task i is determinized depending on which of the two cases shown in Fig. 4 applies. The details of the two cases are provided as follows.

Figures 4 (a) and 4 (b) depict the cases when the identified time periods for insertion of task i allow and not allow the insertion of next task i', respectively. The blue boxes represent the time periods occupied by assigned tasks (e.g., tasks $a, b, c \in T$), i.e., they cannot be inserted with new tasks. The orange box represents the time of the next task i' processed on machining center k, while the green boxes represent all feasible insertion time periods following RSS. The search space of the insertion time for current task i and the next task i' is $\left(t_j^i, t_{cur\_max}\right)$ and $\left(t_j^{i'}, t_{cur\_max}\right)$, respectively.

In Fig. 4 (a), when we find time periods that allow the insertion of both task i and i' (i.e., $inserts_1$ and $inserts_2$), then we let machining center k process task i as soon as possible to free this machining resource (i.e., the final selected time period for task i is $inserts_1$).

In Fig. 4 (b), obviously the time of the next task i' needed (i.e., the orange box) is larger than any of the identified time periods that allow the insertion of task i, i.e., $\{inserts_0, inserts_1, inserts_2\}$. In this case, we choose to insert task i as far from t_j^i as possible, because if the next task i' takes a shorter transportation time, then the search space of task i' will expand to $\Delta t_{i'} = \left(t_{j'}^{i'}, t_{cur\_max}\right)$. The newly expanded search space (i.e., the left-side dashed green box) and the original search space $inserts_0$ are likely to form as a larger space allowing the insertion of the next task i', which will greatly save the time consumption.



(a) (b)

FIGURE 4: (A) A CASE WHEN THE IDENTIFIED TIME PERIODS FOR INSERTION OF TASK i ALLOW THE INSERTION OF NEXT TASK i'. (B) A CASE WHEN THE IDENTIFIED TIME PERIODS FOR INSERTION OF TASK i DO NOT ALLOW THE INSERTION OF NEXT TASK i'.

3. EXPERIMENT AND DISCUSSION

3.1 Experiment settings

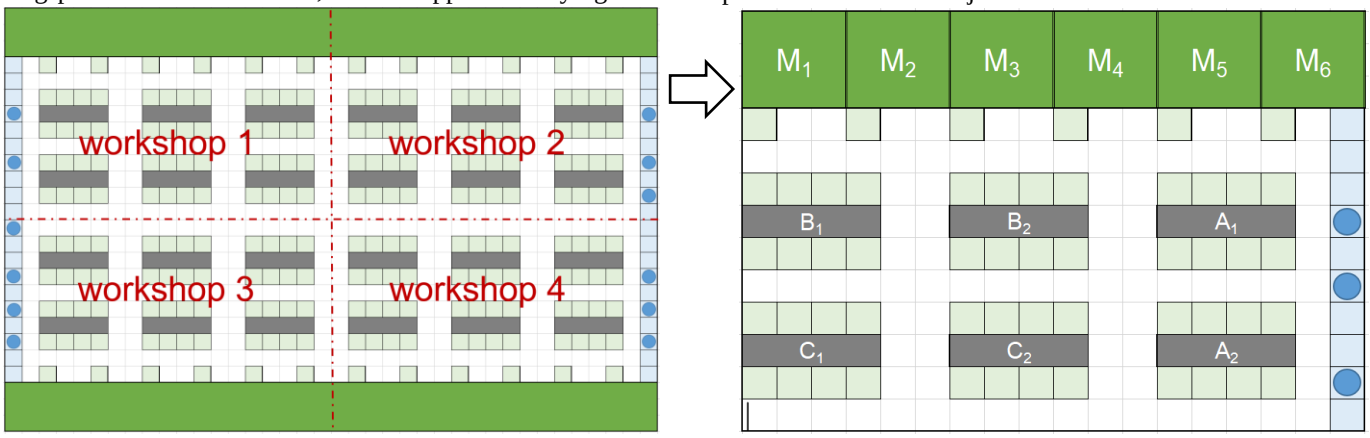
Figure 5 (a) shows the sketch of a simulated smart factory including four workshops placed with machining centers (dark green), corridors (white), docking zones for mobile robots (light blue), and storage zones (dark gray). Comparing to warehousing and logistics where mobile robots undertake similar transportation tasks, the mobile robots equipped for each workshop in a smart factory can be different since the workshops usually serve at different stages in the whole manufacturing process. Thus, it is more often to see a group of robots work within a workshop rather than across various workshops, which can reduce the complexity of control and maintain balanced loadings for these robots. Based on this consideration, we divide the whole robot team into sub-teams each responsible for a specific workshop and select one of these workshops as the experiment environment in this study. Figure 5 (b) presents a workshop of the smart factory represented with a grid map (13×18 cells). The storage zones are specified as raw material areas (A_1, A_2), semi-finished product areas (B_1, B_2), and finished product areas (C_1, C_2). The mobile robots can perform loading or unloading tasks in the light green cells. The working scenario in this workshop is provided as follows.

After receiving the starting signal, a mobile robot leaves from the docking area, goes to the raw material area to load and transports a task to a machining center. Then, the machining center starts to process it. After that, this robot will be assigned to transport other tasks. When the machining center finishes the

assigned task, the output (e.g., processed parts) will then be picked up by one of the available robots and transported to another machining center or a shelf in the storage zone. After completing all tasks assigned to it, the robot will stop at the docking area to avoid collisions with other robots that are still in working status. This process will iterate until all tasks are completed.

We examined the performance of our approach in this environment and working scenario with different number of robots and tasks. Moreover, we developed a simulation platform based on MATLAB. This platform can visualize the machining centers, storage zones and docking zones in a workshop of smart factory, and mark out the starting positions and goal positions of all the transportation tasks. It can also dynamically display the moving paths of the robot team, which supports verifying the feasibility of the proposed approach (e.g., we can observe whether two robots conflict with each other in performing tasks).

Usually, the quantity of transportation tasks to complete per work shift in a workshop of smart factory is limited by the processing capability of the machining centers. Therefore, we tested 200 to 1000 tasks in this experiment. The initial positions of some mobile robots and the starting and goal positions of some tasks are randomly assigned, while others are unknown considering the precedence constraints of tasks. The weighted sum method is used to convert the multiple objective functions into a single objective function. We use w_t to represent the weight of transportation time and processing time, and w_e to represent the weight of energy consumed by machining centers. The assignment of weights is related to the decision maker's preference for these objectives.



(a) (b)

FIGURE 5: (A) SKETCH OF THE SMART FACTORY; (B) SKETCH OF A WORKSHOP IN THE SMART FACTORY.

3.2 Experiment results

We validate the feasibility of our proposed approach by comparing the total time and energy consumption following three strategies as shown in Figure 6. Here the total time and energy consumption (TC) are normalized following Equation (16):

$$\text{normalized } TC = \frac{c - c_{min}}{c_{max} - c_{min}} \tag{16}$$

where c is the time and energy consumed for transportation and processing, c_{max} and c_{min} are the maximum/minimum sum of time or energy consumed for transporting and processing. Here the baseline strategy means a task will be assigned to a machining center at the time point right after the last task assigned to this machining center. The results in Fig. 6 show that the total time and energy consumption grows with the number of tasks, and the proposed Regret-based Search Strategy (RSS) performs better than the Looking-backward Search Strategy (LSS), and both RSS and LSS outperform the baseline strategy for a varying number of robots and tasks.

By comparing (a) vs. (b) and (c) vs. (d) in Fig. 6, we can observe that when the baseline strategy is adopted, increasing the number of robots will not significantly reduce the total consumption. For example, when using the baseline strategy and $n = 1000$, the normalized TC in (a) and (b) is 0.921 and 0.909, respectively. However, the total consumption decreases markedly if the LSS or RSS is used. For example, when using LSS and $n = 1000$, the normalized TC in (a) and (b) is 0.816 and 0.751, respectively. One possible explanation is that when the processing time and energy consumption of machining centers are not considered (e.g., in warehousing and logistics scenarios), more robots will support faster completion of tasks. However, when the processing time and energy consumption of machining centers are considered (e.g., in a smart factory environment), even if the number of robots is increased and the tasks can be transported to machining centers faster, it still takes a certain amount of time for machining centers to process these tasks, which leads to the queuing of tasks and waiting of robots, and the total time consumption will not be significantly reduced. However, the LSS or RSS can reorganize the processing sequence of tasks in real time, which can alleviate the queuing issue and reduce the waiting time for robots. Thus, the decrease of TC will be more obvious when LSS or RSS is adopted.

In addition, by comparing (a) vs. (c) and (b) vs. (d) in Fig. 6, we can observe the influence of different weights of time and energy on the final performance. For example, if using RSS, when robots $= 5$, $w_t = 0.6$, $w_e = 0.4$, $n = 1000$, the normalized $TC = 0.743$, while when robots $= 5$, $w_t = 0.4$,

$w_e = 0.6, n = 1000$, the normalized $TC = 0.881$. This result indicates that the advantage of RSS is more pronounced when the weight of time (w_t) is larger. A possible explanation is that the search of the machining sequence is performed only after each determination of which machining center is assigned, i.e., the energy consumption does not vary with the insertion position since we only consider the time consumed by the machining center without considering the energy consumption during its idle time.

Figure 7(a) indicates that the number of search iterations increases along with the increasing number of tasks. The number of search iterations using RSS is greater than that using LSS, which can lead to slightly longer computing time as shown in Figure 7(b). We also check the generated paths of each robot in a simulation environment developed with MATLAB as shown in Fig. 8, and no conflicting paths are found.

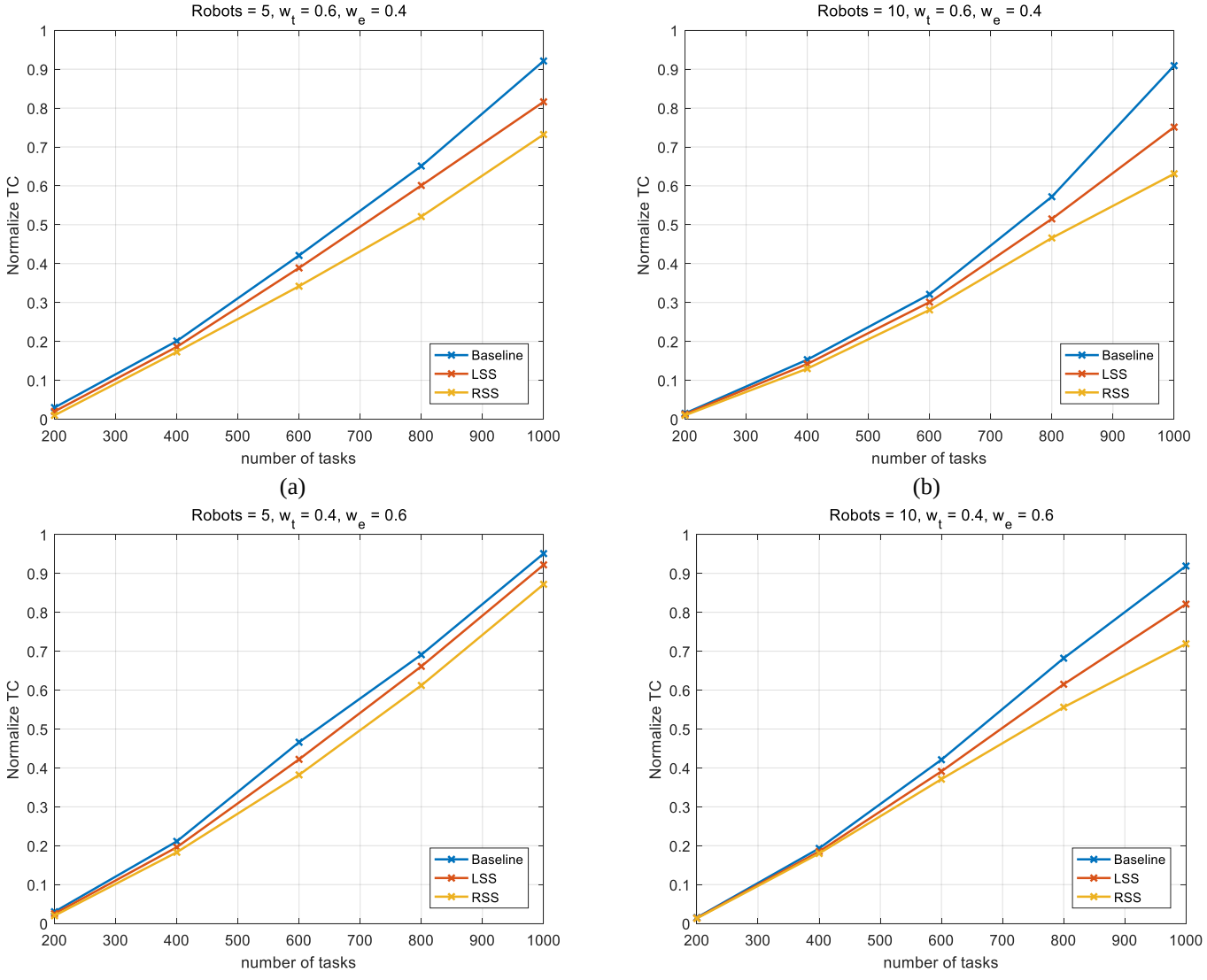


FIGURE 6: NORMALIZED TOTAL CONSUMPTION (TC) VERSUS NUMBER OF TASKS WITH DIFFERENT NUMBER OF ROBOTS AND WEIGHTS.

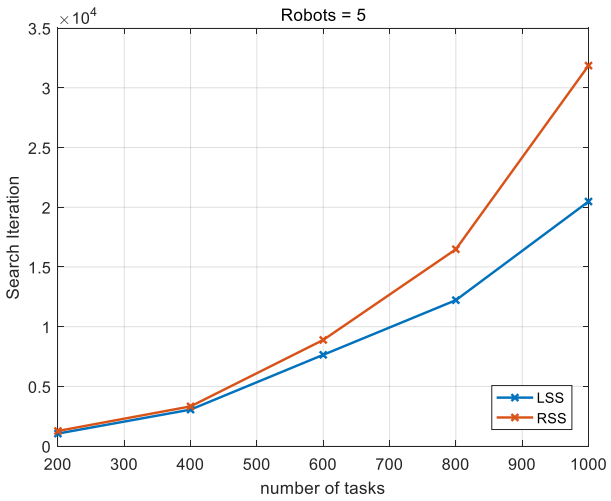


FIGURE 7: (A) THE NUMBER OF SEARCHING ITERATIONS VERSUS NUMBER OF TASKS.

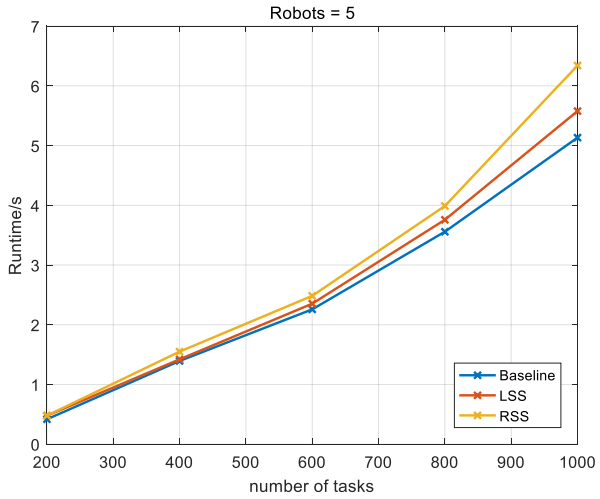


FIGURE 7: (B) RUNTIME VERSUS NUMBER OF TASKS.

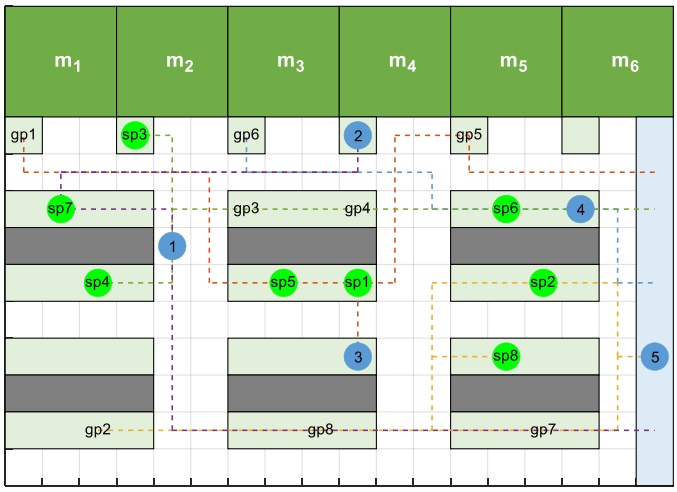


FIGURE 8: SIMULATION ENVIRONMENT DEVELOPED IN MATLAB TO VERIFY THE FEASIBILITY OF GENERATED PATHS FOR MOBILE ROBOTS.

4. CONCLUSION

In this paper, we present an integrated task and path planning approach for mobile robots in smart factory. The basic idea is that in the stage of task assignment, the real paths for mobile robots are identified and the time and energy consumed by mobile robots and machining centers are calculated. Then a greedy strategy working with the Looking-backward Search strategy or Regret-based Search Strategy is used to obtain task assignments in time-series that achieve the proposed objectives, which enables a joint optimal solution for both task assignment and path planning. The real time consumed on the planned paths is used as the basis to adjust and improve the selection of mobile robots and task assignments, and the precedence constraints between sequential manufacturing tasks are considered simultaneously.

We compare the performance of different searching strategies in a simulated factory environment. The results show that the proposed LSS and RSS are better than the traditional strategy when the number of tasks or robots increases, especially when the number of tasks is large. In addition, when the weight of time consumed increases, the advantage of our approach becomes more noticeable. The proposed approach generally will not take long computing time, and it can help reduce the idle time of machining centers and make full use of these resources to improve the overall operational efficiency of smart factory.

One limitation of this paper is that the size of the studied workshop is relatively small. We will examine the reliability and computational efficiency of the proposed approach in a large-scale factory environment where the planning and scheduling of transportation tasks for mobile robots can be more difficult. The complex relationship between manufacturing time and consumed energy will also be considered in future work.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the financial support from the National Natural Science Foundation of China (52005328) and Shanghai Science and Technology Commission "Yangfan" Program (20YF1419300).

REFERENCES

[1] Yadav, Anupma, Jayswal, S., and C., 2018, "Modelling of Flexible Manufacturing System: A Review," Int. J. Prod. Res., **56**(7–8), pp. 2464–2487.

[2] Brown, K., Peltzer, O., Sehr, M. A., Schwager, M., and Kochenderfer, M. J., 2020, "Optimal Sequential Task Assignment and Path Finding for Multi-Agent Robotic Assembly Planning," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 441–447.

[3] Korsah, G. A., Stentz, A., and Dias, M. B., 2013, "A Comprehensive Taxonomy for Multi-Robot Task Allocation," Int. J. Rob. Res., **32**(12), pp. 1495–1512.

[4] Stern, R., 2019, “Multi-Agent Path Finding--an Overview,” Artif. Intell., pp. 96–115.
[5] Yu, J., and LaValle, S. M., 2013, “Multi-Agent Path Planning and Network Flow,” *Algorithmic Foundations of Robotics X*, Springer, pp. 157–173.
[6] Ma, H., Wagner, G., Felner, A., Li, J., Kumar, T. K., and Koenig, S., 2018, “Multi-Agent Path Finding with Deadlines,” arXiv Prepr. arXiv1806.04216.
[7] Bredstrom, D., and Rönnqvist, M., 2007, “A Branch and Price Algorithm for the Combined Vehicle Routing and Scheduling Problem with Synchronization Constraints,” NHH Dept. Financ. \& Manag. Sci. Discuss. Pap., (2007/7).
[8] Yu, J., and LaValle, S. M., 2015, “Optimal Multi-Robot Path Planning on Graphs: Structure and Computational Complexity,” arXiv Prepr. arXiv1507.03289.
[9] Bennewitz, M., Burgard, W., and Thrun, S., 2002, “Finding and Optimizing Solvable Priority Schemes for Decoupled Path Planning Techniques for Teams of Mobile Robots,” Rob. Auton. Syst., **41**(2–3), pp. 89–99.
[10] Erdem, E., Kisa, D. G., Oztok, U., and Schüller, P., 2013, “A General Formal Framework for Pathfinding Problems with Multiple Agents,” *Twenty-Seventh AAAI Conference on Artificial Intelligence*.
[11] Dai, M., Tang, D., Giret, A., and Salido, M. A., 2019, “Multi-Objective Optimization for Energy-Efficient Flexible Job Shop Scheduling Problem with Transportation Constraints,” Robot. Comput. Integr. Manuf., **59**, pp. 143–157.
[12] Hönig, W., Kiesel, S., Tinka, A., Durham, J., and Ayanian, N., 2018, “Conflict-Based Search with Optimal Task Assignment,” *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*.
[13] Sharon, G., Stern, R., Felner, A., and Sturtevant, N. R., 2015, “Conflict-Based Search for Optimal Multi-Agent Pathfinding,” Artif. Intell., **219**, pp. 40–66.
[14] Chen, Z., Alonso-Mora, J., Bai, X., Harabor, D. D., and Stuckey, P. J., 2021, “Integrated Task Assignment and Path Planning for Capacitated Multi-Agent Pickup and Delivery,” IEEE Robot. Autom. Lett., **6**(3), pp. 5816–5823.
[15] Silver, D., 2005, “Cooperative Pathfinding,” *Proceedings of the Aaai Conference on Artificial Intelligence and Interactive Digital Entertainment*, pp. 117–122.