information

Article

Online Planning for Autonomous Mobile Robots with Different Objectives in Warehouse Commissioning Task

Satoshi Warita [1] and Katsuhide Fujita [1,2,*]

[1] Faculty of Engineering, Tokyo University of Agriculture and Technology, Tokyo 184-8588, Japan
[2] Institute of Global Innovation Research, Tokyo University of Agriculture and Technology, Tokyo 184-8588, Japan
* Correspondence: katfuji@cc.tuat.ac.jp; Tel.: +81-42-388-7141

Abstract: Recently, multi-agent systems have become widespread as essential technologies for various practical problems. An essential problem in multi-agent systems is collaborative automating picking and delivery operations in warehouses. The warehouse commissioning task involves finding specified items in a warehouse and moving them to a specified location using robots. This task is defined as a spatial task-allocation problem (SPATAP) based on a Markov decision process (MDP). It is considered a decentralized multi-agent system rather than a system that manages and optimizes agents in a central manner. Existing research on SPATAP involving modeling the environment as a MDP and applying Monte Carlo tree searches has shown that this approach is efficient. However, there has not been sufficient research into scenarios in which all agents are provided a common plan despite the fact that their actions are decided independently. Thus, previous studies have not considered cooperative robot behaviors with different goals, and the problem where each robot has different goals has not been studied extensively. In terms of the cooperative element, the item exchange approach has not been considered effectively in previous studies. Therefore, in this paper, we focus on the problem of each robot being assigned a different task to optimize the percentage of picking and delivering items in time in social situations . We propose an action-planning method based on the Monte Carlo tree search and an item-exchange method between agents. We also generate a simulator to evaluate the proposed methods. The results of simulations demonstrate that the achievement rate is improved in small- and medium-sized warehouses. However, the achievement rate did not improve in large warehouses because the average distance from the depot to the items increased.

Keywords: decentralized online planning; warehouse commissioning; autonomous mobile robot; multi-agent system

Citation: Warita, S.; Fujita, K. Online Planning for Autonomous Mobile Robots with Different Objectives in Warehouse Commissioning Task. *Information* **2024**, *15*, 130. https://doi.org/10.3390/info15030130

Academic Editor: Gabriel Luque

Received: 30 January 2024
Revised: 20 February 2024
Accepted: 22 February 2024
Published: 26 February 2024

1. Introduction

Multiple robot systems are becoming increasingly common in industry, and such systems are expected to revolutionize the factories of the future (Industry 4.0) through networked devices and mobile manipulation platforms. A paradigm shift from fixed robots to more flexible general-purpose mobile robots will be significant. In addition, dynamic resource allocation (DRA) is anticipated to handle uncertain demand, which is a common problem in city-scale cyber-physical systems (CPSs) [1]. In such scenarios, the decision-maker optimizes the spatial location of the resources (i.e., agents) to maximize utility by satisfying constraints specific to the target problem domain. This optimization requires designing and developing procedures to estimate how the system will evolve, as well as evaluations of the long-term value of actions. DRA is applied to many real-world problems in urban management, CPSs, and multi-agent systems. Recently, the use of multi-agent systems has become widespread as an essential technology [2].

One of the essential problems and applications in multi-agent systems is collaborative automated picking and delivery operations in warehouses. Warehouse commissioning

involves finding specified items in a warehouse environment and moving them to a specified location using robots as efficiently as possible while considering various constraints, e.g., the capacity of the robots [3]. A critical aspect of warehouse commissioning is coordinating which worker fetches and delivers which item. With human pickers, this task is allocated to a human picker using pre-computed pick lists, which are generated centrally by the warehouse management system and executed statically, meaning an allocated task cannot be reassigned to different pickers when the allocated picker is in motion [4].

Most previous studies related to warehouse commissioning focused on managing and optimizing robots in a central manner; however, to the best of our knowledge, there has not been sufficient research into methods for each robot to plan its actions autonomously. Action plans in autonomous robots are referred to as the spatial task-allocation problem (SPATAP), which is based on the Markov decision process (MDP). Here, we assume that this problem should be modeled using decentralized agents rather than focusing on central management and robot optimization. Previous studies on SPATAP modeled the environment as an MDP, and it has been reported that applying the Monte Carlo tree search (MCTS) is efficient [5]. However, there has been insufficient research into scenarios in which all agents are provided a common plan despite the fact that their actions are decided independently. In other words, previous studies have not considered cooperative robot behaviors with different goals, and the problem whereby each robot has different goals has not been investigated extensively.

Thus, in this paper, we focus on the problem where each robot is assigned a different task list in order to optimize the percentage of picking and delivering items in time in social situations . We propose an action-planning method based on MCTS and an item-exchange system between agents. We also generate a simulator for this task. Our results of simulations demonstrate that the achievement rate is improved in small- and medium-sized warehouses.

The primary contributions of this study are summarized as follows.

- We propose an environment in which each agent picks up the assigned items under constraints prohibiting collision during agent movement. We formulate the environment as a multi-agent MDP and discuss how the next state is determined from the current state and the actions of all agents;
- We propose a behavioral-planning method to transport items efficiently and a mechanism to exchange allocated items between agents. We also present an action-planning method based on MCTS;
- To improve the achievement rate, we propose a negotiation protocol and exchange negotiation strategies to facilitate the exchange of allocated items between agents;
- The results of the simulation test comprehensively demonstrate the achievement rate of the item-exchange strategies under various test settings.

The remainder of this paper is organized as follows. Section 2 describes previous studies on warehouse commissioning. Section 3 defines the target problem based on the SPATAP. Section 4 presents the proposed MCTS-based action-planning method, which we refer to as the fully decoupled UCT (FDUCT) method. Section 5 proposes the item-exchange protocol and the exchange strategy between agents. Section 6 explains the simulator used to evaluate the proposed approach, and Section 7 discusses the results of the simulations. Finally, Section 8 concludes the paper and presents future challenges.

2. Related Work

Warehouse commissioning is defined as the formal framework of SPATAPs, in which a team of agents interacts with a dynamically changing set of tasks and where the tasks are distributed over a set of locations in the given environment [6]. These models provide principled solutions under the uncertainty of action outcomes and provide defined optimality [5]. In particular, the SPATAP framework can consider moving and working on a task as actions. The SPATAP framework facilitates reasoning about all such actions to maximize the expected utility. The MCTS [7] is effective for this problem because it has a large number of states in each term and a long timeline.

Other approaches to SPATAP include auction-based methods [8], which are well suited to task-allocation problems in multi-agent system. Schneider et al. [9] stated that the *"The advantages of the best auction-based methods are much reduced when the robots are physically dispersed throughout the task space and when the tasks themselves are allocated over time."* However, they did not address the sequential and changing nature of the tasks because they assumed the tasks to be static. Generally, auctioning is also strongly dependent on communication, while the proposed method only requires the locations of the other agents. In a previous study [10], the evaluation results demonstrated that an MCTS-based approach can reduce communication costs significantly while achieving a higher coordination performance.

SPATAPs can also be considered general resource-allocation problems [11]. For example, Kartal et al. [12] used the MCTS to solve task-allocation problems. However, they focused on MRTA problems with a static task list. This differs from the current study due to the reallocations at each time step and the consideration of spatially distributed tasks and travel times. Henn et al. [4] focused on order-picking and -batching problems that only generate static pick lists, which are then executed statically. However, this study did not consider online replanning, changing the task allocations during execution, or decentralized scenarios. Claes et al. [5] proposed MCTS-based distributed-planning approaches and ad hoc approximations to solve the multi-robot warehouse-commissioning problem. This study differs from our work because they did not consider the no-collision constraints and task-exchange procedures. Haouassi et al. [13] generalized integrated order batching, batch scheduling, and picker-routing problems by allowing the orders in the order-picking operations in e-commerce warehouses to split. In addition, they proposed a route-first schedule-second heuristic to solve the problem. However, this study focused on the impact of splitting the orders, which differs significantly from the purpose of the current study.

3. Problem

3.1. Spatial Task-Allocation Problem and Markov Decision Process

The SPATAP[6] involves the following environment and is based on a multi-agent MDP [14]:

- There are N agents;
- There is a graph $G = (V, E)$, and the agent is located at one of the nodes in the graph;
- One node referred to as the depot (stop) is fixed;
- The agent has a fixed value capacity c;
- If the agent has c tasks running, it cannot run new tasks until it is reset in the depot;
- There are four types of agent actions, i.e., do nothing, move to an adjacent node, execute a task at the current node, and reset;
- All agents act simultaneously for each step;
- Node v has a fixed probability p_v, and a task appears at each step with probability p_v;
- The state changes every step, and the agent receives rewards according to its actions;
- The agent has full state observability.

The MDP [15] is an environment in which the next state and the reward received by the agent are determined by the current state and the agent's actions. The MDP is defined by the following four values:

- S: Set of agent states;
- A: Set of possible actions;
- $T(s, a, s')$: Transition function;
- $R(s, a, s')$: Reward function;

The transition function returns the probability of transitioning to state $s' \in S$ when action $a \in A$ is selected in state $s \in S$, and the reward function returns the reward when transitioning from state s to state s' by action a.

The SPATAP is defined according to the following seven values based on the MDP:

- D : Agent set;

- $G = (V, E)$: Graph;
- I : The collection of agent states;
- T : The set of node states;
- A : The set of possible actions;
- $P(s, a, s')$: Transition function;
- $R(s, a, s')$: Reward function;

The state of an agent is the combination of the agent's current position and the number of tasks it is currently executing, and the state of a node is represented by the number of tasks at that node. The transition function returns the probability of transitioning to state s' when action a is selected in state s, and the reward function returns a reward vector (where the ith element corresponds to the reward of the ith agent).

3.2. Target Problem

The target problem can be defined based on the SPATAP. Here, a warehouse is represented as a grid graph G with a single depot node. The warehouse includes N agents, and each agent selects an action for each step independently and executes it simultaneously. In addition, each agent can execute the following four actions:

- Stay in place;
- Move to an adjacent node;
- Collect the item;
- Drop the item at the depot.

Movements that cause agents to collide with other agents will fail, and agents that fail to move will remain in place. Note that a collision can cause a chain of movement failures. In addition, agents with items cannot collect new items until the current items are deposited at the depot. In each step, the items assigned to agent 1, agent 2, . . . , and agent n appear at a random vertex with probability p (each event is independent). Here, the agent's objective is to maximize the number of picking items assigned to it that are delivered to the depot within T steps. Figure 1 illustrates a case with three agents in the warehouse.

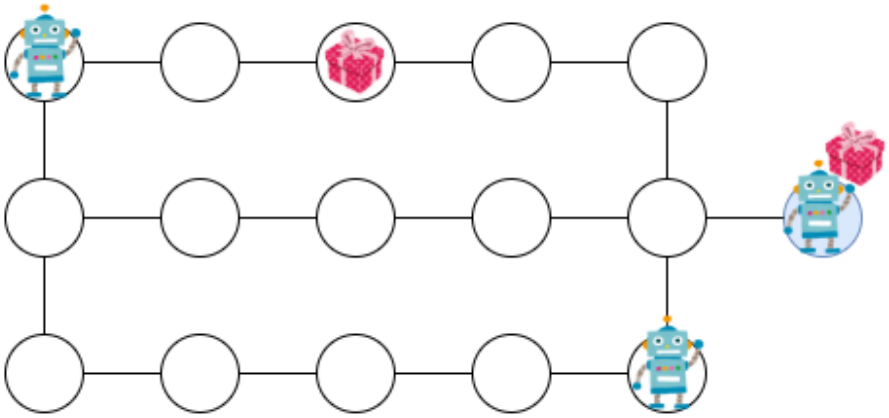

Figure 1. Example problems with three agents in the warehouse. Each agent selects an action for each step independently and executes it simultaneously to maximize the number of assigned items.

The environment of this problem is formulated with the following parameters:

- D: Agent set;
- G: Graph;
- I: The set of possible states of the agent;
- T: The set of possible states of a node;
- A: The set of possible actions for the agent;
- $P(s, a, s')$: Transition function;
- $R(s, a, s')$: Reward function;

The state of an agent is the combination of the agent's current position and the number of items it has (0 or 1), and the state of a node is the total number of items assigned to each agent at that node. The transition function returns the probability of transitioning to state s' when action a is selected in state s. The reward function returns a reward vector in which the ith element corresponds to the reward of the ith agent.

No-Collision Constraints

In the target problem's environment, movements that cause agents to collide with each other fail, and the agents that fail to move remain in place. Given the actions of all agents, we can determine the next position of each agent using Algorithms 1 and 2. Here, the $NEXT(pos, a)$ function returns the next position (ignoring collisions) when the agent whose current position is pos selects action a. $WHO_{CUR}(pos)$ represents the number of the agent whose current position is pos ($None$ if none exists), and $WHO_{NEXT}(pos)$ is the set of agents attempting to move to pos. In Algorithm 1, the $NextPos(cur, a)$ function receives each agent's current position and action, and it returns each agent's next position. First, for each agent d, we $mark_d$ and initialize $fail_d$ to record whether the movement fails, set the next position when collisions are ignored in nxt_d, and update WHO_{CUR} and WHO_{NEXT} (lines 1–7). Then, for each agent d, if $Check(d)$ has not been called, we call $Check(d)$, and if the movement fails, we set nxt_d to cur_d (lines 8–15). The $Check(d)$ function in Algorithm 2 determines whether agent d can move. First, if another agent is attempting to move to the same vertex as agent d, agent d fails to move (lines 2–6). In the following, we assume that this is not the case. If no agent is at the destination of agent d, agent d can move (lines 7–11). When agent c is at the destination of agent d, if agent c fails to move, agent d also fails to move (line 21). Thus, the decision is made recursively while tracing the agent at the destination (lines 12–14). However, if a loop occurs, the movement will fail (lines 15–19). This rule also prohibits movements that involve agents passing each other.

Algorithm 1 NextPos

Require: cur: Current position in each agent, a: Action in each agent
Ensure: nxt: Next position in each agent,
1: **for all** $d \in D$ **do**
2: $mark_d \leftarrow UNVISITED$
3: $fail_d \leftarrow False$
4: $nxt_d \leftarrow NEXT(cur_d, a_d)$
5: $WHO_{CUR}(cur_d) \leftarrow d$
6: $WHO_{NEXT}(nxt_d) \leftarrow WHO_{NEXT}(nxt_d) \cup \{d\}$
7: **end for**
8: **for all** $d \in D$ **do**
9: **if** $mark_d == UNVISITED$ **then**
10: $Check(d)$
11: **end if**
12: **if** $fail_d$ **then**
13: $nxt_d \leftarrow cur_d$
14: **end if**
15: **end for**
16: **return** nxt

Algorithm 2 Check

Require: d: Agent's Number focused on currently
1: $mark_d \leftarrow VISITING$
2: **if** $|WHO_{NEXT}(nxt_d)| > 1$ **then**
3: $mark_d \leftarrow VISITED$
4: $fail_d \leftarrow True$
5: **return**
6: **end if**
7: $c \leftarrow WHO_{CUR}(nxt_d)$
8: **if** $c == None$ **then**
9: $mark_d \leftarrow VISITED$
10: **return**
11: **end if**
12: **if** $mark_c == UNVISITED$ **then**
13: $Check(c)$
14: **end if**
15: **if** $mark_c == VISITING$ **then**
16: $mark_d \leftarrow VISITED$
17: $fail_d \leftarrow True$
18: **return**
19: **end if**
20: $mark_d \leftarrow VISITED$
21: $fail_d \leftarrow fail_c$

4. Fully Decoupled UCT

4.1. Monte Carlo Tree Search

The MCTS is a search algorithm based on a simulation that is used in game AI and other applications. A typical example of its application is AlphaGo [16]. It comprises the four steps, and each step is summarized as follows:

1. *Selection*: begin from the root node R and select actions according to an appropriate algorithm until the unexpanded node L is reached;
2. *Expansion*: Expand if the number of visits to node L is greater than or equal to the threshold. If the node is visited at the next time, it is for searched more deeply ;
3. *Simulation (Rollout, Playout)*: begin from L and select actions according to an appropriate algorithm until the end of the game (or a certain depth), and then calculate the obtained cumulative reward;
4. *Backpropagation*: propagate the reward from L to R and update the average reward obtained when each action is selected at each node.

The pseudocode for the MCTS is given in Algorithm 3. Here, the $Next(cur, a)$ function returns the next state when action a is selected at node cur, and the $MostSelected(cur)$ function returns the most selected action at cur. Moreover, $LIMIT$ represents the upper limit of the number of iterations, and $THRESHOLD$ is the threshold for expansion.

Algorithm 3 MCTS

Require: s: Current status
Ensure: a: Most selected action
$counter \leftarrow 0$
while $counter < LIMIT$ **do**
  $cur \leftarrow s$
  while $Expanded[cur]$ **do**
    $selected \leftarrow Select(cur)$
    $cur \leftarrow Next(cur, selected)$
  end while
  $VisitCount[cur] \leftarrow VisitCount[cur] + 1$
  if $VisitCount[cur] >= THRESHOLD$ **then**
    $Expanded[cur] \leftarrow True$
  end if
  $Rollout(cur)$
  $Backpropagation(cur)$
  $counter \leftarrow counter + 1$
end while
return $MostSelected(s)$

4.2. Upper Confidence Tree

In the MCTS, actions that are selected in the Selection step are searched for more intensively. It is important to select the most promising (likely to be optimal) action. However, if we simply select the action with the highest average reward obtained so far, we will miss the optimal action if it exists among actions that have not yet been fully explored. Thus, it is necessary to balance knowledge utilization (i.e., exploitation) and exploration when selecting actions. For this problem, an existing method treats the selection of actions as a multi-armed bandit problem [17] and applies the UCB algorithm [18]. In this method, the score of the behavior a is calculated as

$$score = \frac{R_a}{N_a} + c\sqrt{\frac{\ln N}{N_a}},$$

and we select high-quality actions. Here, N_a is the number of times action a was selected, N is the sum of N_a, R_a is the sum of the cumulative rewards obtained when action a is selected, and parameter c controls how many times the search is performed. The MCTS using this method is referred to as UCT [7].

4.3. Agent Behavior Prediction

When applying MCTS to a simultaneous start game, e.g., In the SPATAP, the agent needs to predict the actions that other agents will choose because the next state is not determined by a single agent's actions. Note that behavior-prediction algorithms must be fast because they are executed repeatedly. For the SPATAP, a prediction method that assumes each agent acts based on some kind of greedy method has been proposed previously [5]. The pseudocode for the iterative greedy algorithm, which is one of the proposed methods, is given in Algorithm 4. Here, $r_{d,v}$ is the value of node v for agent d, the $LIST()$function returns an empty list, and $GOTO(d, v)$ is the value of node v for agent d. This function returns the behavior when going to node v along the shortest route. The proposed iterative greedy algorithm scans the list of pairs of agent d and node v in descending order of $r_{d,v}$, and it repeats the matching operation if agent d and node v can be matched. Note that the agent d that is matched with node v moves from the current node to node v via the shortest route.

Algorithm 4 Iterative Greedy

Require: s: Current status
Ensure: a: Actions of each agent
$L \leftarrow LIST()$
for all $d \in D$ **do**
  for all $v \in V$ **do**
    $L.append((d, v))$
  end for
end for
sort L by $r_{d,v}$ in descending order
$D_{ass} \leftarrow \emptyset$
$V_{ass} \leftarrow \emptyset$
for all $(d, v) \in L$ **do**
  if $d \in D_{ass}$ or $v \in V_{ass}$ **then**
    skip
  end if
  $a_d \leftarrow GOTO(d, v)$
  $D_{ass} \leftarrow D_{ass} \cup \{d\}$
  $V_{ass} \leftarrow V_{ass} \cup \{v\}$
end for
return a

4.4. Decoupled UCT

The decoupled UCT (DUCT) algorithm is an extended version of the UCT algorithm for simultaneous-move games [19]. In the original UCT algorithm, each node of the search tree stores the number of times action a was selected and the sum of the cumulative rewards obtained by selecting action a, as shown in Table 1. In contrast, the DUCT algorithm has a table for each agent at each node of the search tree and selects the actions of each agent by executing the UCB algorithm for each table. In other words, this method predicts the behavior using the UCB algorithm rather than using a greedy-based behavior-prediction algorithm. The advantage of the DUCT algorithm is that it is easy to apply because there is no need to design a specific algorithm for each distinct problem. However, in practical applications, when the game has more than three players, the search space becomes huge; thus, it is difficult to apply the DUCT algorithm directly. Thus, we should also consider metrics to facilitate pruning.

Table 1. Table of each node in DUCT algorithm.

Action	A	B	C
Number of selections	4	3	3
Cumulative reward	10	5	40

4.5. Fully Decoupled UCT

As mentioned previously, the DUCT algorithm is an extended UCT algorithm for simultaneous-move games. Compared to using a behavior-prediction algorithm based on a greedy method, this method is easier to implement because there is no need to design different algorithms for different problems. However, as the number of agents increases, it becomes increasingly difficult to search deeply because the number of options (i.e., the set of actions for each agent) at a certain node increases exponentially relative to the number of agents. When using a behavior-prediction algorithm based on the greedy method, the choices at a given node are constant regardless of the number of agents because only one's own behavior involves degrees of freedom. If other agents also select actions using MCTS, it is difficult to predict their actions using an algorithm based on a simple greedy method.

Thus, we also propose the fully decoupled UCT (FDUCT) algorithm so that the number of actions selected at a specific node is constant regardless of the number of agents. The basic concept of the FDUCT algorithm is to execute as many UCTs as there are agents in parallel. For example, assume we have three agents (Agent 1, Agent 2, and Agent 3), and Agent 1 is attempting to plan its own actions using the FDUCT algorithm. In this case, Agent 1 employs a UCT to plan its own actions (referred to as UCT_1) and a UCT to predict the actions of Agents 2 and 3 (referred to as UCT_2 and UCT_3, respectively) in parallel. Here, each UCT_i searches for the optimal action in a single agent's MDP in which only agent i acts. As a result, the number of actions selected at each node in UCT_i is constant regardless of the number of agents because only the actions of agent i are selected. As a result, it can search more deeply than the DUCT algorithm. However, it cannot reflect the interaction between agents; thus, we cannot avoid collisions. Therefore, we also introduce a mechanism that sequentially shares the actions selected by each agent between UCTs and performs state transitions in consideration of conflicts.

The pseudocode for the proposed FDUCT algorithm is given in Algorithm 5. Here, the $Decouple(s)$ function separates the entire state into the states of each agent. First, the overall state is separated into the states of each agent (line 6), and the actions of each agent are selected using the UCB algorithm (lines 7–9). Next, state transition is performed based on the current state and each agent's actions considering the conflicts between agents (line 10). Then, the same processing as the normal UCT is performed for each agent (lines 12–23).

Algorithm 5 Fully Decoupled UCT

Require: s: Current status
Ensure: a: Actions selected at most
1: $counter \leftarrow 0$
2: **while** $counter < LIMIT$ **do**
3: $cur \leftarrow s$
4: $done \leftarrow 0$
5: **while** $done < n$ **do**
6: $cur_1, cur_2, \ldots, cur_n \leftarrow Decouple(cur)$
7: **for all** cur_i **do**
8: $selected_i \leftarrow UCT_i.Select(cur_i)$
9: **end for**
10: $cur \leftarrow Next(cur, selected)$
11: $cur_1, cur_2, \ldots, cur_n \leftarrow Decouple(cur)$
12: **for all** cur_i **do**
13: **if** $UCT_i.Expanded[cur_i]$ **then**
14: skip
15: **end if**
16: $UCT_i.VisitCount[cur_i] \leftarrow UCT_i.VisitCount[cur_i] + 1$
17: **if** $UCT_i.VisitCount[cur_i] >= THRESHOLD$ **then**
18: $UCT_i.Expanded[cur_i] \leftarrow True$
19: **end if**
20: $UCT_i.Rollout(cur_i)$
21: $UCT_i.Backpropagation(cur_i)$
22: $done \leftarrow done + 1$
23: **end for**
24: **end while**
25: $counter \leftarrow counter + 1$
26: **end while**
27: **return** $MostSelected(s)$

5. Item-Exchange Protocol and Agent Strategies

We also propose a mechanism to exchange allocated items between agents. This algorithm prevents agents from becoming overloaded or idle by distributing the load among the available agents in order to improve the efficiency of warehouse commissioning tasks.

5.1. Item-Exchange Protocol

The exchange of items between agents is performed according to the following protocol. Item exchange occurs at the beginning of each step before the agent executes an action.

1. Each agent selects at most one item from among the items assigned to them that they would like to have transported on their behalf and submits a request;
2. Each agent selects and responds to at most one request that it is willing to accept;
3. The requesting agent selects a single agent willing to accept the request. After that, the requesting agent sends the acceptance message to the agent.

5.2. Exchange Strategy

In this section, we describe the proposed item-exchange strategy. Here, a request strategy is employed to determine whether to issue a request and which item to request for transportation. An acceptance strategy is used to determine whether to accept a request and which request to accept. The strategy to determine who to select from among the agents who have responded that they will accept the request is called the nomination strategy and is distinguished from this strategy.

5.2.1. Request Strategy

We first define the load $load_i$ of agent i using the following formula. Here, V_i is the set of vertices with an item assigned to the agent, $count(v)$ is the number of bags at vertex v, and $dist(v)$ is the number of bags from the depot of vertex v. The distance cur_i is the current position of agent i. In addition, $HasItem_i$ is true when agent i has the item; otherwise, $HasItem_i$ is false.

$$load_i = \begin{cases} \sum_{v \in V_i} dist(v) \times count(v) + dist(cur_i) & (agent_i\ has\ Items) \\ \sum_{v \in V_i} dist(v) \times count(v) & (otherwise) \end{cases}$$

An agent decides whether to issue a request based on its own load and the average load. If its load exceeds the average load, it issues a request; otherwise, a request is not issued.

We propose the following three strategies to determine which item to request for transportation:

- Select the item closest to the depot (NEAREST_FROM_DEPOT);
- Select the item most distant from the depot (FARTHEST_FROM_DEPOT);
- Select a item at random (RANDOM).

However, in any case, items whose distance from the depot exceeds the difference between their own load and the average load are excluded from the available candidates.

5.2.2. Acceptance Strategy

An agent determines whether to accept a request based on its own load and the average load. If its own load is less than the average load, it will accept the request; otherwise, it will not accept the request.

We propose the following three strategies to determine which request to accept;

- Select the request closest to the depot (NEAREST_FROM_DEPOT);
- Select the request most distant from the depot (FARTHEST_FROM_DEPOT);
- Select a request at random (RANDOM).

However, items whose distance from the depot exceeds the difference between the agent's load and the average load are excluded from the available candidates.

5.2.3. Nomination Strategy

Among the agents who indicated they are willing to accept the request, the agent with the lowest load is nominated.

6. Multi-Agent Warehouse-Commissioning Simulator

6.1. Input

The configuration file input to the simulator is in a JavaScript Object Notation format. The items that can be set in the configuration file are summarized as follows, and an example of how to write the configuration file is shown in Figure 2a. In addition, Figure 2b shows the visualization of our simulator. It is a straightforward visualization because our simulator is simple to use for evaluating our proposed approach. "♯" means the wall, "." means the load, and "D" means the agent. In future work, we will improve the visualization by showing the position of each agent (different colors) in the graph.

- numAgents: Number of agents;
- lastTurn: Number of steps to perform the simulation;
- newItemProb: Probability that a new item will appear;
- numIters: Number of iterations in the FDUCT algorithm;
- maxDepth: Maximum depth to search in the FDUCT algorithm;
- expandThresh: Expansion threshold in the FDUCT algorithm;
- reward: The reward is obtained when the agent collects and unloads the item at the depot;
- penalty: The penalty (i.e., a negative reward) is given to an agent when it collides with another agent;
- discountFactor: Discount rate in cumulative reward calculation;
- randSeed: Random number seed value;
- enableExchange: Whether to exchange luggage;
- requestStrategy: Request strategy;
- acceptStrategy: Acceptance strategy;
- nominateStrategy: Nomination strategy;

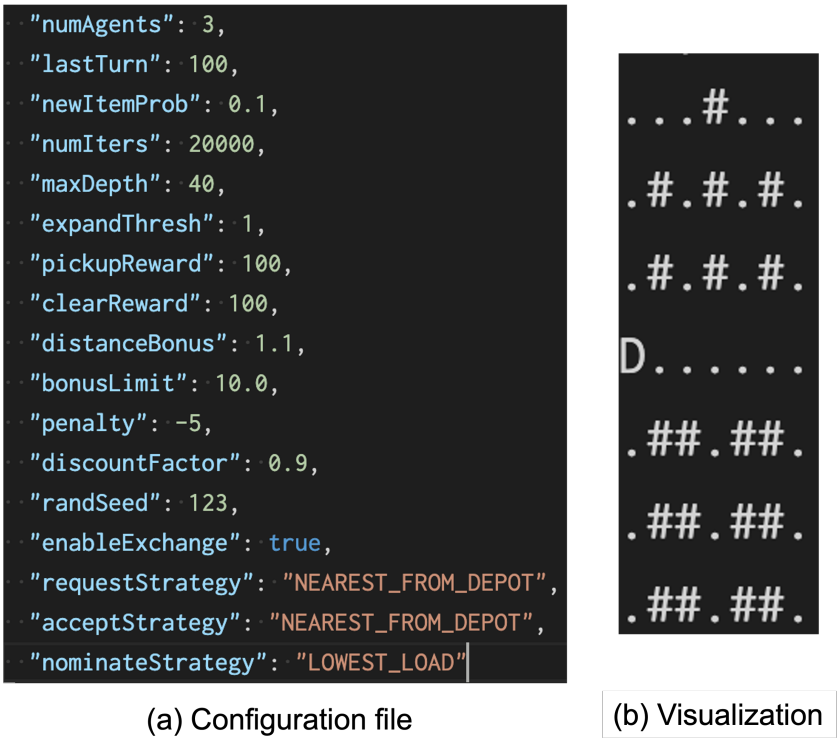

Figure 2. Example of configuration file and visualization.

6.2. Output

The following results are output when the simulation ends.

- The number of items that appeared;
- The number of items that could be carried;
- The achievement rate for each agent.

If multiple simulations are run, the average and variance of these values are output. In addition, the total number of items that have appeared, the total number of transported items, and the overall achievement rate, which is (the total number of items that have been carried)/(total number of items that have appeared), are also output.

7. Evaluation

We evaluated the effectiveness of the item-exchange system by comparing the simulation results obtained with and without the item-exchange process. In addition, we performed simulations for all pairs of request strategies and acceptance strategies, and we evaluated the exchange strategies by comparing the results. We conducted a preliminary test to determine appropriate parameter values for the FDUCT algorithm.

7.1. Common Settings

The common settings used in all simulations are shown in Table 2. In real applications, time constraints exist. In our simulations, we set the time limitations (lastTurn) and evaluated the achievement rate by 100 steps. The values of maxDepth (the maximum depth to search in the FDUCT algorithm), expandThresh (the expansion threshold in the FDUCT algorithm), and the penalty (given when collisions with other agents occur) were determined based on the parameter-tuning results. The simulation was performed 30 times, and the average overall achievement rate was compared.

Table 2. Common setting of simulation test.

Parameter	Value
numAgents	3
lastTturn	100
newItemProb	0.1
numIters	20,000
maxDepth	20
expandThresh	2
reward	100
penalty	−5
discountFactor	0.9
randSeed	123

7.2. Grid Graphs for Warehouses

Three types of grid graphs for warehouses were used in our tests: warehouse-small (30 nodes), warehouse-medium (66 nodes), and warehouse-large (214 nodes). Each graph is shown in Figures 3–5, respectively. In these figures, the blue nodes represent the depots.

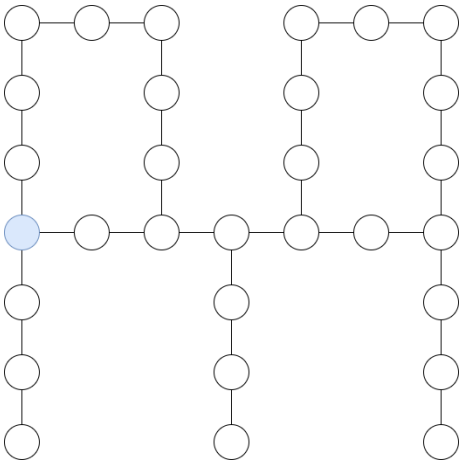

Figure 3. Warehouse-small grid graph.

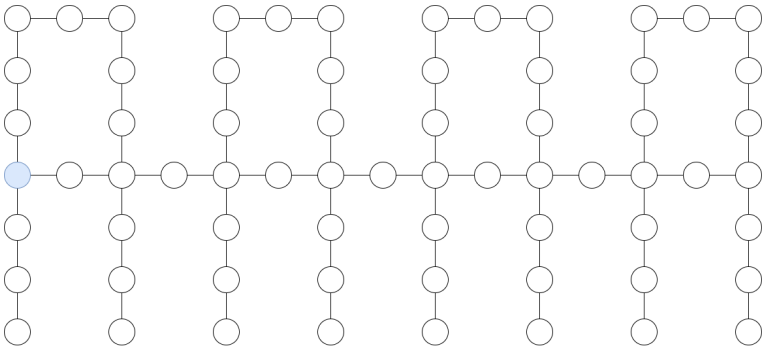

Figure 4. Warehouse-medium grid graph.

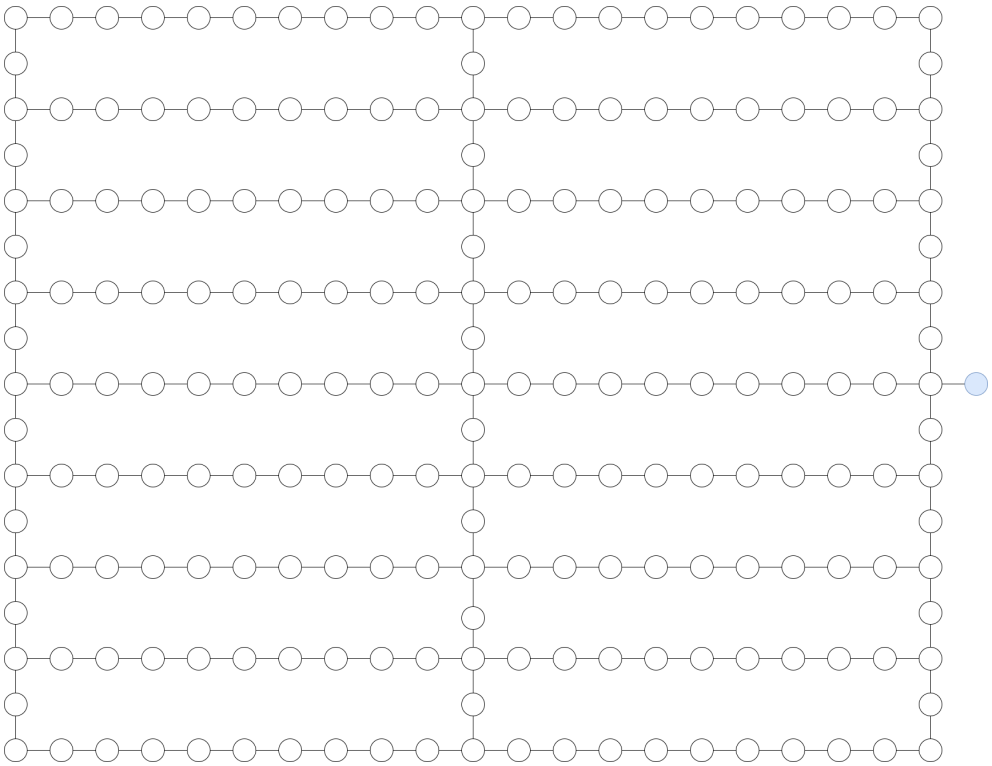

Figure 5. Warehouse-large grid graph.

7.3. Evaluation Result

In this simulation test, we evaluated the effectiveness of the item-exchange strategy by comparing the simulation results obtained with and without the item-exchange process. We also performed simulations for all pairs of request and acceptance strategies, and we evaluated the exchange strategies by comparing the obtained results.

7.3.1. Warehouse-Small

The results of the simulation tests for the warehouse-small case are shown in Figure 6. Here, BASELINE represents the result obtained without the item-exchange process, and X_Y represents the result when the request strategy was X and the acceptance strategy was Y. For example, NEAREST_RANDOM represents the result obtained when the request strategy was NEAREST_FROM_DEPOT and the acceptance strategy was RANDOM. The achievement rates for the different strategies are ranked in descending order as follows : FARTHEST_RANDOM, NEAREST_RANDOM, NEAREST_FARTHEST, RANDOM_FARTHEST, RANDOM_NEAREST, NEAREST_NEAREST, RANDOM_RANDOM, FARTHEST_FARTHEST, and FARTHEST_NEAREST, BASELINE. In summary, the achievement rate exceeded that of BASELINE, which demonstrates that the item exchange is effective. In addition, the average achievement rate obtained using NEAREST_FROM_DEPOT as the request strategy was 0.678, the average achievement rate obtained using FARTHEST_FROM_DEPOT was 0.674, and the average achievement rate obtained using RANDOM was 0.675. For the warehouse-small case, using the NEAREST_FROM_DEPOT request strategy obtained the best average results. This means that other agents with more time (i.e., a lighter load) were asked to transport items that appeared near the depot, and the agent collected items that appear further away from the depot due to the division of roles. Note that it is also possible to have a reverse division of roles, where an agent requests another agent to transport an item that appears far away, and the requesting agent collects a closer item. Nearby items tend to be collected preferentially; thus, there is a high probability that it will not be carried out in time, even if the transportation of an item that appears far away is requested. Thus, it is easier to divide the roles by requesting the transportation of nearby items.

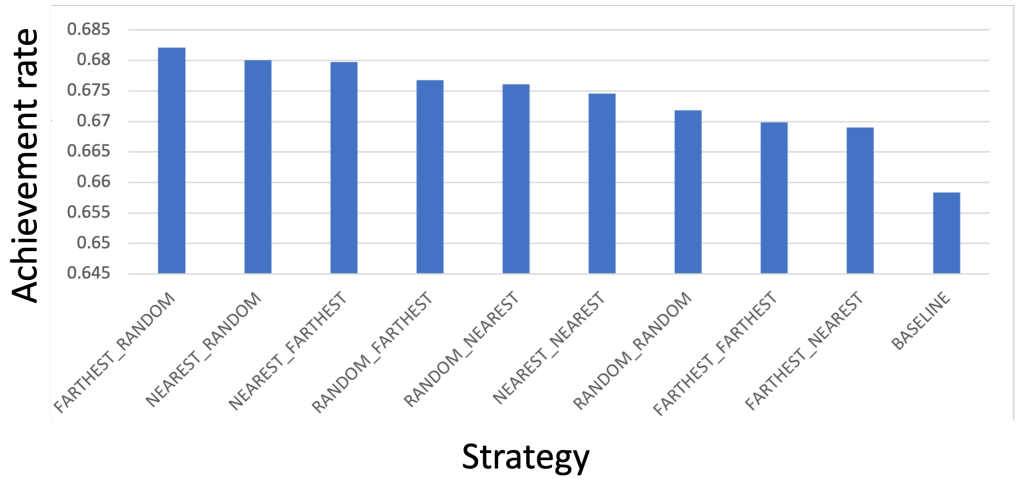

Figure 6. Achievement rate of each strategy in warehouse-small case.

7.3.2. Warehouse-Medium

The results of the simulation tests for the warehouse-medium case are shown in Figure 7. The achievement rates for the different strategies are ranked in descending order as follows: FARTHEST_NEAREST, BASELINE, RANDOM_FARTHEST, NEAREST_RANDOM, NEAREST_NEAREST, RANDOM_NEAREST, NEAREST_FARTHEST, FARTHEST_FARTHEST, RANDOM_RANDOM, and FARTHEST_RANDOM. The achieve-

ment rate of FARTHEST_NEAREST was higher than BASELINE, which demonstrates that the item-exchange process is effective. However, unlike the warehouse-small case, most strategies could not exceed the BASELINE because, as warehouses become larger, the average distance from the depot to items increases, and the time limit is reached by collecting items close to the depot. In addition, the reason the achievement rate was higher than the BASELINE only when FARTHEST_NEAREST was utilized is summarized follows. The achievement rate is increased due to the transport of far-off parcels among them relatively close to the depot that would normally not be carried within the time limit.

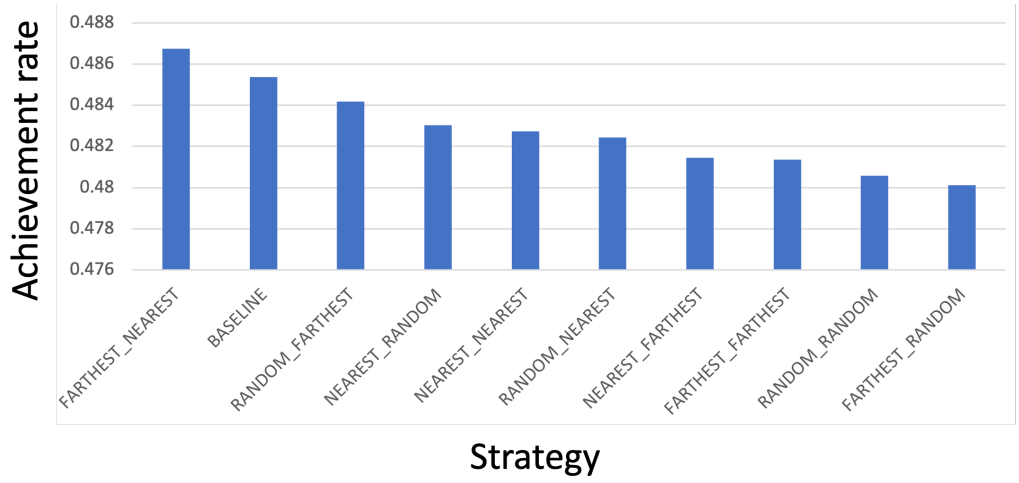

Figure 7. Achievement rate by strategy for warehouse-medium case.

7.3.3. Warehouse-Large

The results of simulation tests obtained for the warehouse-large case are shown in Figure 8. The achievement rate for the compared strategies are ranked in descending order as follows: BASELINE, FARTHEST_NEAREST, FARTHEST_RANDOM, FARTHEST_FARTHEST, RANDOM_FARTHEST, NEAREST_RANDOM, NEAREST_NEAREST, RANDOM_NEAREST, NEAREST_FARTHEST, and RANDOM_RANDOM. Note that the achievement rate of the methods with item exchange did not exceed the BASELINE because the warehouse was larger than the warehouse-medium case, and the distance from the depot to the items was longer.

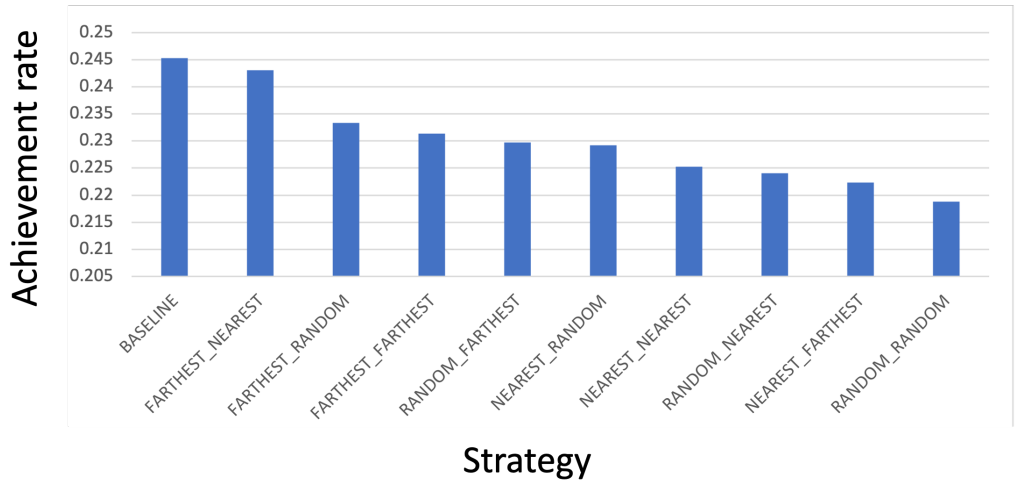

Figure 8. Achievement rate by strategy in warehouse-large case.

8. Conclusions

In this paper, we have described an environment wherein each agent transports items assigned to it under a constraint prohibiting movement that would cause agent collisions. We have proposed a behavior-planning method and an item-exchange system to transport items efficiently in such an environment. We evaluated the effectiveness of the item-exchange system and exchange strategy through tests using a simulator. As an action-planning method, we also proposed the FDUCT algorithm, which is based on the conventional DUCT algorithm. This method predicts agent behavior using the UCB algorithm. We found that this method solves the problem, whereby the number of options at a given node increases exponentially with respect to the number of agents. For the item-exchange system, we presented an exchange protocol and proposed several exchange strategies. In the warehouse-small simulation test, we found that the achievement rate exceeded the rate obtained without the item-exchange process. We also found that using the NEAREST_FROM_DEPOT-request strategy resulted in a higher achievement rate on average compared to the other compared strategies. However, in the warehouse-medium and warehouse-large cases, as the warehouse became larger, the average distance between the depot and the items also increased.

Thus, potential future work could focus on cases in which the number of agents is larger. Evaluations with a larger number of agents are the potential focus of future work. If the number of agents increases more, we can simulate traffic jams. We believe that the proposed approach can escape the traffic jams as much as possible because it considers no-collision constraints. In addition, the proposed FDUCT algorithm solves the problem where the number of options at a given node increases exponentially according to the number of agents. A possible solution is the approach of narrowing down to a certain number of agents based on the distance between agents. Other potential future work involves improving the problem settings and evaluation indicators. With the current problem setting, the achievement rate naturally decreases as the size of the warehouse increases. In the meantime, new items appear to have reached a fundamental solution. Thus, it is necessary to set an upper limit on the number of items that appear and perform evaluations based on the time required to transport all items.

Author Contributions: Conceptualization, S.W. and K.F.; methodology, S.W. and K.F.; software, S.W.; validation, S.W. and K.F.; formal analysis, S.W. and K.F.; investigation, S.W. and K.F.; resources, S.W.; data curation, S.W.; writing—original draft preparation, S.W.; writing—review and editing, K.F.; visualization, S.W.; supervision, K.F.; project administration, K.F.; funding acquisition, K.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets generated during the experiments are available from the corresponding author on reasonable request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mukhopadhyay, A.; Pettet, G.; Samal, C.; Dubey, A.; Vorobeychik, Y. An Online Decision-Theoretic Pipeline for Responder Dispatch. In Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems, New York, NY, USA, 16–18 April 2019 ; pp. 185–196. [CrossRef]
2. Pettet, G.; Mukhopadhyay, A.; Kochenderfer, M.J.; Dubey, A. Hierarchical Planning for Dynamic Resource Allocation in Smart and Connected Communities. *ACM Trans. Cyber Phys. Syst.* **2022**, *6*, 1–26. [CrossRef]
3. Wurman, P.; D'Andrea, R.; Mountz, M. Coordinating Hundreds of Cooperative, Autonomous Vehicles in Warehouses. *AI Mag.* **2008**, *29*, 9–20.

4. Henn, S.; Koch, S.; Wäscher, G., Order Batching in Order Picking Warehouses: A Survey of Solution Approaches. In *Warehousing in the Global Supply Chain: Advanced Models, Tools and Applications for Storage Systems*; Manzini, R., Ed.; Springer: London, UK, 2012; pp. 105–137.
5. Claes, D.; Oliehoek, F.; Baier, H.; Tuyls, K. Decentralised Online Planning for Multi-Robot Warehouse Commissioning. In Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, Richland, SC, USA, 8–12 May 2017; pp. 492–500.
6. Claes, D.; Robbel, P.; Oliehoek, F.A.; Tuyls, K.; Hennes, D.; van der Hoek, W. Effective Approximations for Multi-Robot Coordination in Spatially Distributed Tasks. In Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, Istanbul, Turkey , 4–8 May 2015; pp. 881–890.
7. Kocsis, L.; Szepesvári, C. Bandit Based Monte-Carlo Planning. In *Lecture Notes in Computer Science, Proceedings of the Machine Learning, ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, 18–22 September 2006*; Proceedings; Fürnkranz, J., Scheffer, T., Spiliopoulou, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4212, pp. 282–293.
8. Braquet, M.; Bakolas, E. Greedy Decentralized Auction-based Task Allocation for Multi-Agent Systems. *IFAC-PapersOnLine* **2021**, *54*, 675–680. [CrossRef]
9. Schneider, E.; Sklar, E.I.; Parsons, S.; Özgelen, A.T. Auction-Based Task Allocation for Multi-robot Teams in Dynamic Environments. In *Proceedings of the Towards Autonomous Robotic Systems*; Dixon, C., Tuyls, K., Eds.; Springer: Cham, Switzerland, 2015; pp. 246–257.
10. Li, M.; Yang, W.; Cai, Z.; Yang, S.; Wang, J. Integrating Decision Sharing with Prediction in Decentralized Planning for Multi-Agent Coordination under Uncertainty. In Proceedings of the 28th International Joint Conference on Artificial Intelligence, Macao, China, 10–16 August 2019; AAAI Press: Washington, DC, USA, 2019; pp. 450–456.
11. Wu, J.; Durfee, E.H. Resource-Driven Mission-Phasing Techniques for Constrained Agents in Stochastic Environments. *J. Artif. Int. Res.* **2010**, *38*, 415–473. [CrossRef]
12. Kartal, B.; Nunes, E.; Godoy, J.; Gini, M. Monte Carlo Tree Search for Multi-Robot Task Allocation. *Proc. AAAI Conf. Artif. Intell.* **2016**, *30*. [CrossRef]
13. Haouassi, M.; Kergosien, Y.; Mendoza, J.E.; Rousseau, L.M. The integrated orderline batching, batch scheduling, and picker routing problem with multiple pickers: The benefits of splitting customer orders. *Flex. Serv. Manuf. J.* **2021**, *34*, 614–645. [CrossRef]
14. Boutilier, C. Plannning, learning and coordination in multiagent decision processes. In Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge, Renesse, The Netherlands, 17–20 March 1996; Morgan Kaufmann: San Francisco, CA, USA, 1996; pp. 195–210.
15. Puterman, M.L. *Markov Decision Processes—Dicrete Stochastic Dynamic Programming*; John Wiley & Sons, Inc.: New York, NY, USA, 1994.
16. Silver, D.; Huang, A.; Maddison, C.; Guez, A.; Sifre, L.; Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [CrossRef] [PubMed]
17. Mahajan, A.; Teneketzis, D. Multi-Armed Bandit Problems. In *Foundations and Applications of Sensor Management*; Hero, A.O., Castañón, D.A., Cochran, D., Kastella, K., Eds.; Springer: Boston, MA, USA, 2008; pp. 121–151.
18. Auer, P.; Cesa-Bianchi, N.; Fischer, P. Finite-time Analysis of the Multiarmed Bandit Problem. *Mach. Learn.* **2002**, *47*, 235–256. [CrossRef]
19. Lanctot, M.; Wittlinger, C.; Winands, M.H.; Den Teuling, N.G. Monte Carlo tree search for simultaneous move games: A case study in the game of Tron. In Proceedings of the Benelux Conference on Artificial Intelligence (BNAIC), Delft, The Netherlands, 7–8 November 2013; Delft University of Technology: Delft, The Netherlands, 2013; Volume 2013 , pp. 104–111.