

Article

Semantic Cost-Based Task Allocation for Heterogeneous Multi-Robot Systems: Load-Dependent Route and Payload-Margin Costs

Gungyo In ¹, Kwanghee Lee ², Yechan An ¹, Kyeongjin Joo ¹, Gihyeon Kwon ³ and Taeyong Kuc ^{1,*}

- ¹ Department of Electrical and Computer Engineering, College of Information and Communication Engineering, Sungkyunkwan University, Suwon 16419, Republic of Korea; ingungyo@skku.edu (G.I.); ycan@skku.edu (Y.A.); jookj96@skku.edu (K.J.)
- ² Human-Centric Robotics R&D Department, Korea Institute of Industrial Technology (KITECH), Ansan 15588, Republic of Korea; leekh@kitech.re.kr
- ³ Department of Intelligent Robotics, College of Engineering, Sungkyunkwan University, Suwon 16419, Republic of Korea; brian1398@skku.edu
- * Correspondence: tykuc@skku.edu

Abstract

Heterogeneous robot fleets transport items that differ in shape and weight, and robots differ in what they can handle and where they can pass. Conventional task allocation methods often compute allocation costs and routes on a single load-invariant spatial model, which may overlook changes in place traversability after loading. Payload capacity typically enters allocation only as a feasibility or ranking condition, so light tasks may be allocated to high-capacity robots also needed for heavier ones. Taking the feasibility information from semantic reasoning as input, this study combines a route cost on load-state-dependent traversability graphs with a normalized payload-margin cost in a makespan-oriented sequential allocation procedure. In ablation experiments, load-invariant route planning produced non-traversable loaded routes for approximately one quarter of the tasks, whereas load-state-dependent planning and validation produced none, as expected by construction. Against a violation-free variant with load-invariant costs and no payload term, the proposed configuration shortened the makespan by 6.7–20.1%, mainly through the payload-margin term; its route cost made the predicted completion times consistent with the planned routes. The reduction persisted on a larger layout and with congested loading and unloading places. In ROS 2 and Gazebo, eight robots followed the planned place sequences in both completed runs.

Keywords: multi-robot task allocation; heterogeneous robots; semantic knowledge; load-dependent traversability; payload margin; makespan



Academic Editor: Subhas Mukhopadhyay

Received: 25 August 2026

Revised: 8 October 2026

Accepted: 8 October 2026

Published: 12 October 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

In logistics warehouses and manufacturing sites, multiple mobile robots with different payload capacities and dimensions share a single workspace and perform transport tasks. In such settings, deciding which task to allocate to which robot governs the overall completion time and resource utilization, and the range of that decision widens as the number of robot classes increases.

Approaches that assign a cost or utility to robot–task pairs and allocate on that basis have been widely used [1–3], but the spatial model used to compute the cost generally remains a single model independent of what a robot is carrying. While the places a robot

can pass through change with the size of the object being carried, a distance computed without the load state can omit the detour required after loading, and the allocation cost then diverges from the actual travel conditions. In route planning as well, when the load state is not reflected, routes are generated that include aisles that cannot be traversed after loading. In addition, the queue of a robot with a large payload margin relative to the required weight is occupied by light tasks, and less room remains for allocating heavier tasks later.

The information required to address these problems has already reached the allocation layer. From the dimensions of a robot, the shape of the loaded object, and the traversable cross-section of a place, semantic feasibility reasoning derives which robots can perform which tasks and which places are then reachable. The fact that the load state changes traversability has also been reflected in formulations that treat allocation and route planning together, but that condition was generally a single rule common to all robots and tasks. An earlier study by the authors produced these feasibility decisions for each robot–task pair and, to show that allocation methods can use them, applied four one-to-one assignment methods with a common cost, namely a route distance computed separately for the empty-travel and loaded segments [4]. This paper addresses the objective and cost of allocation in which each robot receives an ordered queue of tasks.

A method is proposed that converts the information produced by semantic feasibility reasoning into the allocation cost and integrates it into a makespan-oriented sequential allocation procedure. Load-dependent traversability conditions are reflected in the travel distance cost, candidate robots are compared using the payload margin relative to the required weight as well, and allocation is performed to reduce the completion time of the robot that finishes last. The scope is open-loop batch allocation, in which all tasks are allocated before execution in a static environment. The main contributions of this study are as follows:

- The output of semantic feasibility reasoning is represented as allocation costs for queue-dependent allocation: the loaded segment of each robot–task pair is evaluated, planned, and validated on the same pair-specific load-state graph, and the payload margin, normalized by the maximum capacity in the eligible robot set of the task, forms a cost term that functions as a capacity reservation.
- These costs are combined with the expected completion time of each robot into a single makespan-oriented selection criterion. The allocation procedure itself uses a standard sequential scheme as a controlled variable: without the payload term, it is the Min-Min rule, equivalently a sequential single-item (SSI) auction with the MiniMax bidding rule.
- The two terms act together in the proposed criterion, and their roles are distinguished by configurations that switch them independently and by configurations in which the load state is reflected only in the route planning: the payload term accounts for most of the makespan reduction, whereas the route term keeps the completion times computed in the allocation consistent with the planned routes. The sensitivity to the payload weight and to the task mixture is also reported, and the conclusions are examined on a larger layout, under other operating conditions and congestion, with another bidding rule of the same auction, and against a single loaded rule common to all robots and tasks. The consistency between the planned place sequences and the executed motion is confirmed in multi-robot simulation.

The remainder of this paper is organized as follows. Section 2 reviews related work, and Section 3 describes the problem formulation and the proposed method. Sections 4 and 5 present the experimental setup and the results, and Sections 6 and 7 present the discussion and the conclusion, respectively.

2. Related Work

2.1. Cost Models for Multi-Robot Task Allocation

Many approaches to heterogeneous multi-robot task allocation have defined the cost incurred or utility obtained when a task is allocated to a robot and determined the allocation and order of execution on that basis [5]. These values have reflected not only travel distance and time but also waiting time, deadlines, and energy consumption, among others. In auction-based approaches, each robot bids the marginal score of adding a task to its own bundle [1], and in patrolling with repeated visits, the idleness of a target point and travel cost are combined as a weighted sum [6]. In learning-based approaches, temporospatial constraints and the current schedule are represented as a graph, and a sequential allocation policy is learned using the change in the objective function as the reward [2]. An approach that sets energy, completion time, and time-window violation as an explicit composite objective function and generates the allocation and order together has also been presented [7].

In optimization-based approaches as well, the metaheuristic family has used both a single-objective formulation that minimizes travel distance [3] and a multi-objective formulation that considers completion time and energy together [8]. The specialization of a robot has also been encoded as a constraint, with energy consumption treated as the optimization objective [9]. The computational complexity and approximation bounds of related MRTA variants have also been analyzed [10], and the setting has been extended to multi-tasking, in which a single robot performs several tasks simultaneously [11]. A recent review presents deep reinforcement learning and large language models as emerging research directions [12].

Among the ways of combining these values into a single objective function, two are directly relevant to this study. One minimizes the sum of the per-robot costs [3], and the other reduces the completion time of the robot that finishes last [2,13]. The latter connects to the treatment of the maximum completion time in parallel machine scheduling [14], and directly bounds the time at which the entire fleet finishes all tasks. Instead of directly optimizing such a global objective, sequential allocation methods that build the schedule step by step using marginal costs have also been used. Bundle construction in auctions and sequential greedy allocation share a common structure in which tasks are chosen one at a time by evaluating the marginal cost of undertaking a single task in the current state [1,6]. For target visiting problems with precedence constraints, a method that evaluates the possible insertion positions in each robot's schedule to minimize the time at which the last target is visited has been proposed [13]. A knowledge-graph-based auction has also used a scheme that evaluates insertion positions [15].

The processing set restriction problem in scheduling, in which the set of machines that can process each job differs, has long been studied [16]. For the inclusive structure in which those sets are nested, the allocation of cargo to loading and unloading cranes with different weight capacities has been presented as one application [17]. However, in this family each job carries a processing time that is given as an input, whereas in transport tasks that value is derived from the traversability graph specific to each robot-task pair and from the current state of the queue.

The cost of a transport task has been computed by dividing the travel into a segment that reaches the target and a segment that carries it to the destination. In pickup and delivery problems, the task cost is the sum of two distances: from the current robot position to the pickup place, and between the pickup and drop places [18]. For a task that the same robot performs next, the cost is computed in the same way starting from the place where the previous task ended. In a study that optimizes production scheduling and transport together, the distinction between the two segments is expressed through

speed, and the model reflects that a loaded robot moves more slowly than an unloaded one [19]. Therefore, dividing the cost into an empty-travel segment and a loaded segment is an already established structure, and evaluating the two segments differently is not in itself new.

The spatial model on which that value is computed has also been treated in various forms. Straight-line distance is simple to compute, but in environments such as warehouses that contain racks and restricted zones, it does not reflect the actual travel. One learning-based study reports this limitation of its own spatial encoder and notes that path-based distances could be substituted without changing the architecture [7]. In recent learning-based allocation, robot-to-task travel distances precomputed offline with A* are used as edge attributes of the graph [20]. In a centralized coordination framework, the drivable shortest path distance returned by the navigation stack is used directly as the allocation cost [21]. Some studies also place static access constraints in the model, and a setting in which the workshop layout limits the region each robotic arm can reach is reflected in schedule optimization [22].

The effect of the load state on travel has been studied at several levels. In addition to the distinction in speed, the travel time between adjacent nodes has been estimated from a load-aware motion profile that includes acceleration and deceleration [23]. Furthermore, at the route planning level, models in which the load state changes traversability itself have been presented. One model considers the sizes of the robot and the carried material together and limits the available orientations and actions according to path width and node size [24]. Another model separates the movement space by state so that a robot can pass beneath a rack only while it is not carrying one [25].

However, the state-dependent traversability models described above have mainly been applied to the layer that generates routes. In the rack rearrangement problem, for example, a stage that pairs robots with racks does exist, but the cost of that stage is computed from the shortest distance the robot travels to reach the rack [25]. The route of the loaded segment is already fixed by the preceding plan. In work that reflects material dimensions, which robot takes which task does not follow from that determination [24]. In contrast, in formulations that solve allocation and route planning together, the allocation is decided using the actual cost produced by the planner [26], and the allocation and routes of heterogeneous vehicles are handled in the same framework [27]. The restriction that a loaded AGV cannot enter rack grids has also been included as an allocation constraint [28]. However, that restriction is a single rule common to all vehicles and tasks, and the fleet is homogeneous. Even where heterogeneity is treated, the traversable space is left as a uniform grid independent of vehicle shape [27]. Because these formulations are tied to grid maps and their own path planning models, a configuration in Section 4.2 emulates such a single loaded rule within the present procedure.

In summary, the elements included in the allocation cost have been treated in various ways, and the spatial models underlying those values have varied; the effect of the load state on travel has also been discussed at several levels. In queue-dependent allocation, in which the empty-travel segment depends on the preceding task in the same queue, the following two cases were not found. One is a case in which the set of reachable places differs for each robot–task pair according to the combined dimensions of the robot and the carried object, and the cost of the loaded segment is evaluated on that pair-specific graph. The other is a case in which the payload capacity, rather than remaining a condition that determines feasibility or a rule that orders candidates, enters the selection criterion as a real-valued term normalized over the candidate set and traded on the same axis as travel distance.

2.2. Semantic Knowledge in Task Allocation

Ontology-based knowledge representation has been widely used to enable robots to handle the environment and tasks in a machine-interpretable form. Semantic maps and knowledge processing infrastructures have been presented as means of connecting high-level task descriptions to low-level execution, and ontologies that formalize resource skills and process steps have been proposed for manufacturing processes [29–32].

One use of this knowledge is to narrow the set of feasible candidates. The capabilities of assembly resources are formalized in an ontology, and resources that meet the required capabilities are retrieved by a query and passed to the allocation stage [33]. Semantic edges have also been established only between compatible robot and task types, with the utility computed only on the resulting subgraph [15]. In learning-based allocation, an approach has also been presented that excludes pairs failing to meet capacity and energy conditions at the decoder stage, thereby structurally enforcing the constraints [7]. On the spatial side, traversability has been inferred by rules that compare the footprint size of a robot with the entrance size of a doorway; however, this result is used to constrain navigation [34]. An earlier study by the authors produced traversability determinations as task-dependent eligible robot sets and reachable place sets [4].

In one framework, a knowledge base supplies the target object weight, robot speed, and energy consumption coefficients, and the task cost is the sum of the motion, action, and recharging times [35]. In a study that combines an ontology knowledge graph with auctions, robots whose capability level falls short of the task difficulty are excluded from bidding [36]. The allocation cost is then a weighted sum of the normalized travel distance, capability gap, and task count. This capability gap is a real-valued margin term, but it is defined on capability levels rather than on payload capacity and is normalized by a maximum capability gap rather than by the largest capacity in the eligible robot set of each task. In another study that uses a knowledge graph, a utility that combines a time-discounted task value, distance, travel time, urgency, value density, and fuel consumption in a weighted sum is used as the bid [15]. In capability matching, an ordering that prefers resources close to the required capacity has been used, but this is a rule that determines the order of query results and is not treated as a real-valued term in the selection criterion [33]. Therefore, converting semantic information into an allocation cost is not new in itself, and the question of which semantic information is carried into the cost remains.

Semantic knowledge has also been transferred to the planner. Places are divided into upper and lower levels, planning is performed first at the upper level, and only the lower places under the upper places in that result are queried for a second planning pass, which reduces the environmental information the planner handles [37]. In another approach, only the places reachable within a budget set by the remaining energy of the robot are retained, and the metric of the planner then determines the allocation [38]. In a further approach, an ontology is converted into a planning language, and the results of several planners are compared by a weighted sum of execution time, number of steps, and resource use to select one [39]. In these cases, semantic knowledge is used to reduce the problem the planner must handle or to choose among its results, and which robot takes which task is determined within the search or the metric of the planner.

In summary, semantic knowledge has been used as a condition that restricts candidates, a numerical value that enters the allocation, and a state space that the planner handles. Two questions remain. One is which semantic information to convert into a cost, and the other is whether the value was computed on the same spatial model as the one on which routes are actually generated.

2.3. Comparative Positioning

Table 1 organizes the points left by the preceding two subsections along six axes. Among these, the task-dependent traversability model and load-conditioned route cost are separated because the two axes can come apart, and the rack rearrangement case is one example [25]. Cost-plan graph consistency is distinct from these, and it asks whether the model on which the cost was evaluated and the model that generates routes are the same, and whether that model depends on the load state. Work that uses the drivable shortest path distance as the allocation cost refers to the same type of spatial model at both layers, but that model does not depend on the load state [21].

Table 1. Positioning of the proposed method with respect to feasibility representation, cost formulation, and route consistency.

Study	Semantic Source	Task-Dependent Traversability	Load-Conditioned Route Cost	Capacity-Margin Preference	Queue-Dependent Cost	Cost-Plan Graph Consistency
Kluge-Wilkes [33]	OWL	×	×	△	×	—
Li [36]	OWL KG	×	×	△	△	—
Yamauchi [24]	—	△	×	×	×	×
Li [25]	—	△	×	×	×	×
Velhal [18]	—	×	×	×	✓	—
Hu [20]	—	×	△	×	✓	—
Varghese T. [21]	—	×	△	×	△	△
Han [28]	—	×	✓	×	×	✓
In [4]	TOSM/SWRL	✓	✓	×	×	△
Ours	TOSM/SWRL	✓	✓	✓	✓	✓

The axes are derived from the problem definition of this study. For the earlier study, the △ mark denotes paths that were generated on the same load-state graphs as the cost but only for visualization, not as routes to be validated and executed. ✓ = addressed; △ = partially addressed or static; × = not addressed; — = not applicable.

The semantic feasibility reasoning, including the semantic models of robots, tasks, and places, is taken from the earlier study [4], as are the loaded cross-section rule, the payload capacities of four robot classes, and the traversable cross-sections of the main aisle and the 1.0 m high shortcut of testbed_A (Section 4.1). The load-state route cost in queue-dependent allocation, the payload-margin cost, the makespan-oriented selection criterion, the validation of the planned loaded routes, and the evaluation in Sections 4 and 5 are new in this paper.

3. Problem Formulation and Proposed Method

3.1. Problem Setting and Semantic Feasibility Information

The problem follows the standard definition of multi-robot task allocation, in which tasks are allocated to robots to optimize a fleet-level objective [40,41]. With the makespan as the objective, it is related to scheduling on unrelated parallel machines with processing set restrictions [16] and sequence-dependent setup times [42]. Here, the processing sets are the eligible robot sets $A(t)$ defined below, and the setup time corresponds to the empty travel from the drop place of the preceding task.

A heterogeneous robot fleet performs item transport tasks, and the environment in which the tasks are performed consists of places, which are semantic spatial units such as areas and aisles. Each place has a unique identifier. This environment is represented as a weighted graph $G = (V, E, \ell)$, which is undirected except for the one-way connections of testbed_C (Section 4.1 and Appendix A). V is the set of places that correspond to the movement and working spaces, E is the set of traversable connections between adjacent places, and $\ell : E \rightarrow \mathbb{R}_{>0}$ gives the travel distance of each connection. The robot fleet

$R = \{r_1, \dots, r_m\}$ consists of heterogeneous robots that differ in external dimensions, provided capability, object handling method, and payload capacity. The set of transport tasks is $T = \{t_1, \dots, t_n\}$, and each task t is defined by a pickup place p_t , a drop place d_t , and a target object o_t that has dimensions and a weight.

The semantic feasibility information for this problem is given as the ReasonerOutput produced by a semantic reasoner that implements the reasoning of the earlier study [4]. The reasoner operates on an ontology based on the Triplet Ontological Semantic Model (TOSM) [34,43] with rules written in the Semantic Web Rule Language (SWRL) [44], which has also been used for ontology-based reasoning about context events [45]. The eligibility verdict combines conditions on capability, form factor, payload, and loaded-state reachability, as defined in that study. The payload condition, whose quantities are also used in the payload-margin cost of Section 3.2, holds if $\text{payloadCapacity}(r, t) \geq \text{requiredPayload}(t)$, where $\text{payloadCapacity}(r, t)$ is the effective capacity of the handling method shared by the robot and the task, and $\text{requiredPayload}(t)$ is the object weight. Each robot in this study provides a single handling method, so this value is uniquely determined. The ReasonerOutput does not contain distances or routes, so this paper computes them on reachable graphs.

The ReasonerOutput consists of two main result fields, and this paper defines their contents with the following notation. RobotReachability contains, for each robot r , the empty-state reachable place set $P_r(\text{empty}) \subseteq V$, while TaskFeasibility contains, for each task t , the eligible robot set $A(t) \subseteq R$ and, for each $r \in A(t)$, the reachable place set $P_r(o_t) \subseteq V$ in the state in which the object o_t is loaded. Figure 1 shows the flow in which the ReasonerOutput is passed to load-state graph construction, semantic cost generation, allocation, and pre-execution validation.

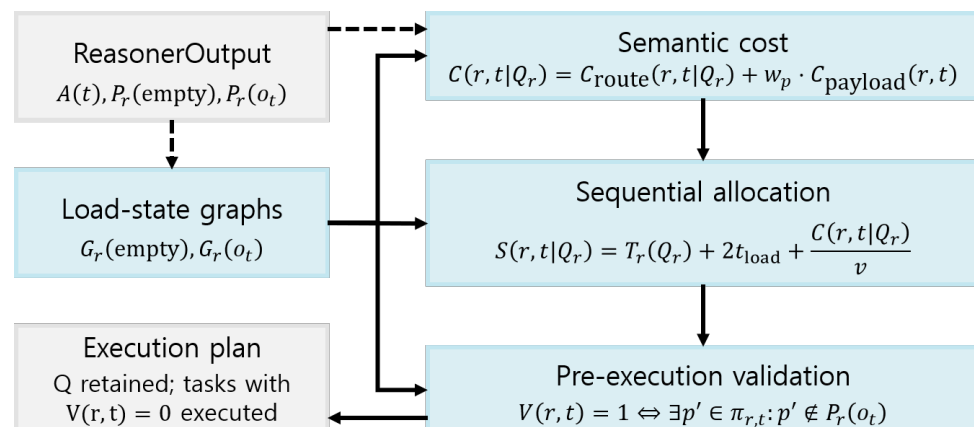


Figure 1. Workflow of the proposed semantic cost-based task allocation. Dashed arrows indicate inputs, and solid arrows the flow of the proposed method.

The allocation problem is formulated as follows. The input is the feasibility information of the ReasonerOutput together with the state of the robot fleet at the time of allocation, where the robot state comprises the position $s_r \in P_r(\text{empty})$ and the payload capacity of each robot. The output is an ordered task queue for each robot:

$$Q = \{Q_r\}_{r \in R}, \quad Q_r = (q_{r,1}, \dots, q_{r,|Q_r|}), \quad q_{r,j} \in T \quad (1)$$

In Equation (1), $q_{r,j}$ is the j -th task performed by robot r . Each robot departs from s_r and performs the tasks in the order of its queue. Allocation is carried out in a batch over all of T before execution begins, so no robot carries a load at this point. Each task also ends after unloading is completed at the drop place, so the segment between two consecutive tasks likewise begins in the empty state. Allocation follows two conditions:

- (i) Each task for which an eligible robot exists ($A(t) \neq \emptyset$) is allocated exactly once to exactly one queue.
- (ii) Every allocated pair (r, t) satisfies $r \in A(t)$.

A task with $A(t) = \emptyset$ remains unallocated because no robot can perform it, and this is not an outcome produced by the allocation procedure but a fact fixed by the feasibility information. This paper reports it separately from the allocation results. The cost measure is defined in Section 3.2 and the objective function in Section 3.3.

The load-state reachable place sets induce a family of weighted subgraphs of the environment graph G :

$$G_r(\lambda) = G[P_r(\lambda)], \quad \lambda \in \{\text{empty}\} \cup \{o_t : t \in T, r \in A(t)\} \quad (2)$$

Here, $G[S]$ is the weighted subgraph induced by a place subset $S \subseteq V$. In the implementation, distances and routes are computed on a route graph that represents each place by its center and by further nodes, and on this graph, $G[S]$ consists of the nodes and connections that lie only in places of S (Appendix A). $\lambda = \text{empty}$ is the state in which no load is carried and reflects only the task-independent static constraints of the robot itself (the empty-state graph). $\lambda = o_t$ is the state in which the object of task t is loaded, and it is defined for $r \in A(t)$, for which $P_r(o_t)$ is given (the task-dependent graph; hereafter also the loaded graph). The ReasonerOutput guarantees two properties [4]. First, $P_r(o_t) \subseteq P_r(\text{empty})$, and, therefore, $G_r(o_t)$ is a subgraph of $G_r(\text{empty})$. Second, if $r \in A(t)$, the pickup and drop places belong to $P_r(o_t)$ and the two places are connected in $G_r(o_t)$. For the directed graph of testbed_C, it was also verified that the drop place can be reached from the pickup place in $G_r(o_t)$ for every eligible robot–task pair. This paper additionally assumes that, in the empty-state graph $G_r(\text{empty})$ of each robot, every pickup place can be reached from the initial position and from every drop place.

According to the taxonomy of Korsah et al., this problem falls into ID [ST-SR-TA] [41]. Each robot performs only one task at a time (ST), each task is performed by one robot (SR), and multiple tasks are allocated to each robot as an ordered queue (TA). In addition, the cost of a candidate task depends on the preceding tasks in the same queue, that is, on the position of the queue end, so the problem belongs to the in-schedule dependencies (ID) class. Such a queue-dependent allocation structure is a representative form of the in-schedule dependency that also appears in the consensus-based bundle algorithm (CBBA) family [1]. This study addresses the same problem structure.

The system models open-loop dispatch. The task queue of each robot is dispatched after allocation, route planning, and pre-execution validation are completed. Updating the graph G in response to obstacles newly observed during execution, replanning the task queues, and reallocation belong to the closed-loop problem and are not addressed in the open-loop model of this study. The congestion-aware replay at the end of Sections 5.3 and 5.4 examine whether the open-loop results hold under congestion and in simulated execution.

3.2. Semantic Cost Modeling

The semantic reasoning input provides the eligible robot set $A(t)$ with infeasible pairs filtered out, together with the load-state reachable place sets, but it does not determine the relative preference among feasible candidates. Therefore, allocating tasks efficiently requires a real-valued cost by which the candidates can be compared, and this study defines it as the sum of a travel distance cost and a payload-margin cost as follows:

$$C(r, t \mid Q_r) = C_{\text{route}}(r, t \mid Q_r) + w_p \cdot C_{\text{payload}}(r, t) \quad (3)$$

C_{route} in Equation (3) is a travel distance cost computed on the load-state graph $G_r(\lambda)$, and it uses the actual distance directly, so no separate weight is applied to it. C_{payload} is a cost that penalizes allocations with excessive payload margin. Because it is dimensionless, it has a weight $w_p \geq 0$ that converts it into a distance-equivalent quantity. The cost is computed only for feasible pairs, that is, pairs with $r \in A(t)$.

3.2.1. Load-State-Dependent Route Cost

When task t is allocated to robot r , the travel is divided into two segments. One is the empty-travel (deadhead) segment toward the pickup place, and the other is the loaded segment that carries the object. The routes of the two segments can differ depending on the sizes of the robot and the target object and on the traversability conditions of the places. Reflecting this constraint, C_{route} is defined as the sum of the two segments:

$$C_{\text{route}}(r, t \mid Q_r) = \text{sp}_{G_r(\text{empty})}(\text{pos}_r, p_t) + \text{sp}_{G_r(o_t)}(p_t, d_t),$$

$$\text{pos}_r = \begin{cases} s_r & Q_r = \langle \rangle \\ d_{q_r, |Q_r|} & \text{otherwise} \end{cases} \quad (4)$$

In Equation (4), $\text{sp}_G(a, b)$ denotes the shortest-path distance from a to b in graph G . The first term is therefore the distance from pos_r to p_t in $G_r(\text{empty})$, and the second is that from p_t to d_t in $G_r(o_t)$. Here, p_t and d_t are the pickup and drop places of task t , and pos_r is the planned position at which robot r starts its next task. It is the initial position when the queue is empty and the drop place of the last task in the queue otherwise. Traversability is looked up directly from the load-state graphs defined above without a separate decision rule. The two segments are computed on different graphs because the reachable place set changes with the load state, and the inclusion $P_r(o_t) \subseteq P_r(\text{empty})$ determines the direction of the misestimation that arises when the cost is computed on only one of the two graphs. If the empty-travel segment is computed on $G_r(o_t)$, places that an empty robot can pass through are excluded under the loaded criterion, so the cost can be overestimated. Conversely, if the loaded segment is computed on $G_r(\text{empty})$ (load-invariant route cost), places that drop out in the loaded state can remain on the shortest path, and the cost can then be underestimated. The latter yields a distance estimate based on a route that passes through places that are not traversable in the loaded state.

Load dependence enters the graph definition rather than a separate penalty term, so blocked places and the resulting detours are reflected in the travel distance without an additional coefficient. However, whether the cost uses this graph and whether the route actually returned by the planner belongs to $G_r(o_t)$ are separate matters. The latter is determined by the validation $V(r, t)$ in Section 3.4. C_{route} is not a term that judges feasibility again, but a term that compares distances among the pairs already admitted by $A(t)$.

3.2.2. Payload-Margin Cost

The second term reflects the payload margin as a cost:

$$C_{\text{payload}}(r, t) = \frac{\text{payloadCapacity}(r, t) - \text{requiredPayload}(t)}{\text{cap}_{\max}(t)},$$

$$\text{cap}_{\max}(t) = \max_{r' \in A(t)} \text{payloadCapacity}(r', t) \quad (5)$$

Here, $\text{payloadCapacity}(r, t)$ and $\text{requiredPayload}(t)$ are the values defined in Section 3.1, and $\text{cap}_{\max}(t)$ is the largest payloadCapacity among the robots in $A(t)$. The numerator, the payload margin, is in kg and cannot be added to the travel distance as it is, so it is divided by $\text{cap}_{\max}(t)$ to give a dimensionless value $C_{\text{payload}} \in [0, 1]$. The normalization

reference is the maximum within $A(t)$ rather than the maximum over the entire fleet, so that the magnitude of the payload term matches the capacity scale of the candidates that can actually perform the task. If the maximum over the entire fleet were used, the capacity of high-capacity robots would dominate the denominator, and the margin differences among medium- and low-capacity candidates would shrink accordingly and have little effect on the selection.

C_{payload} is defined to increase as the payload margin grows among the eligible robots of a task, so the value of this term is smaller for a capacity-fit robot whose capacity is closer to the required load. In particular, the pair remains feasible even when $\text{payloadCapacity}(r, t) = \text{requiredPayload}(t)$. Because the payload margin is zero, the pair is the most preferred among the feasible candidates in this term. Among tasks whose requirements other than payload are identical, the eligible robot set of a task with a larger required load is contained in the eligible robot set of a lighter task. That is, a high-capacity robot can perform both light and heavy tasks, whereas a low-capacity robot cannot perform heavy tasks. Therefore, the capacity-fit preference discourages allocating a task that a low-capacity robot can handle to a high-capacity robot first, and it leaves room to allocate still-unallocated heavy tasks to high-capacity robots. This effect is called capacity reservation, where reservation does not mean preserving physical payload capacity but that the queues of high-capacity robots are not occupied unnecessarily by light tasks.

The weight w_p is the conversion rate between payload margin and travel distance. Because C_{route} is in meters and C_{payload} is a dimensionless value in $[0, 1]$, w_p determines how many meters of additional travel are regarded as equivalent to a normalized payload margin of 1.0. The larger w_p is, the more additional travel is accepted to choose a capacity-fit candidate.

3.3. Makespan-Oriented Allocation Procedure

The cost in Section 3.2 expresses the relative preference among feasible robot–task pairs, but the allocation performance of the entire fleet is determined by the time at which all task queues of the robots are completed. Therefore, even when the pair cost of a task is low, adding the task to that robot can delay the overall completion time if the execution time accumulated in the current queue of that robot is large. To reflect this queue workload, this section defines the completion time of each robot and the makespan of the fleet, and combines the semantic cost of Section 3.2 into the selection criterion of the sequential allocation.

The payload margin is not a physical time quantity that is added directly to the actual task completion time, so it is not used as a component term of the makespan. Instead, it is reflected only in the selection criterion defined below, as the payload-margin cost of Section 3.2. Let $v > 0$ be the nominal travel speed, and let t_{load} be the loading and unloading time required at the pickup and the drop, respectively. Let τ be the incremental execution time required when robot r adds task t to the end of its current queue Q_r and performs it, T_r the completion time of the entire queue, and M the makespan of allocation Q .

$$\tau(r, t \mid Q_r) = \frac{C_{\text{route}}(r, t \mid Q_r)}{v} + 2t_{\text{load}}, \quad T_r(Q_r) = \sum_{j=1}^{|Q_r|} \tau(r, q_{r,j} \mid Q_{r,<j}), \quad (6)$$

$$M(Q) = \max_{r \in R} T_r(Q_r)$$

In Equation (6), $Q_{r,<j}$ is the subsequence consisting of the first $j - 1$ tasks of Q_r , and when $j = 1$ the robot departs from its initial position s_r . The empty-travel segment is computed from the drop place of the preceding task to the pickup place of the next task, so each τ depends on the queue end position determined by the preceding tasks. Changes in

deadhead travel due to the task order are reflected in T_r and M . In the main experiments, v and t_{load} are constants common to all robots and tasks, and their values are given in Section 4.2.

The target allocation problem is defined as minimizing $M(Q)$ in Equation (6) subject to the eligibility constraints given by $A(t)$ defined in Section 3.1. Global optimization that searches the entire allocation and the queue order simultaneously is combinatorially complex, so this study uses a sequential heuristic that adds tasks one at a time to the end of a queue.

The one-step increase in the makespan is not used directly as the selection criterion at each step. That increase is defined as $\Delta M(r, t | Q) = \max\{0, T_r(Q_r) + \tau(r, t | Q_r) - M(Q)\}$. When the expected completion time $T_r + \tau$ of a candidate is at most the current makespan, ΔM becomes zero and such candidates are not distinguished from one another. To retain selection information about the queue workload in this region as well, this procedure includes the expected completion time in the criterion instead of ΔM .

The selection criterion S for comparing candidate pairs (r, t) at each step is defined by adding the payload-margin cost to the expected completion time of robot r after task t is added:

$$\begin{aligned} S(r, t | Q_r) &= T_r(Q_r) + \tau(r, t | Q_r) + \frac{w_p}{v} C_{\text{payload}}(r, t) \\ &= T_r(Q_r) + 2t_{\text{load}} + \frac{C(r, t | Q_r)}{v} \end{aligned} \quad (7)$$

In Equation (7), $T_r + \tau$ is the expected completion time of robot r when task t is added to the end of the current queue. T_r reflects the accumulated workload of the queue already constructed, and τ reflects the travel time and the loading and unloading time added by the new task. The last term is the payload-margin cost defined above. Because w_p is a coefficient that converts the normalized payload margin into travel distance, it is divided by v to convert it into time units. Once the units are matched in this way, the pair cost C can be used directly, as in the second line of Equation (7).

When $w_p = 0$, S reduces to the expected completion time of the candidate robot, and at each step the procedure selects the robot–task pair that gives the earliest completion time. When $w_p > 0$, a weighted trade-off between the expected completion time and the payload margin is added, and the smaller the difference in completion time is, the more a capacity-fit candidate with a small payload margin is preferred. The normalized payload margin does not imply a comparison of absolute physical quantities across tasks. However, because it is included in the common selection criterion S , it can affect not only the choice of robot but also the order in which robot–task pairs are selected. Whether this capacity-fit preference improves the makespan is not guaranteed directly by the definition; it is evaluated experimentally in Section 5.1, and its dependence on conditions is analyzed in Section 5.2.

Let U be the set of unallocated tasks for which an eligible robot exists; the selection and the completion time update at each iteration are given in Equation (8).

$$(r^*, t^*) = \underset{t \in U, r \in A(t)}{\operatorname{argmin}} S(r, t | Q_r), \quad T_{r^*} \leftarrow T_{r^*} + \tau(r^*, t^* | Q_{r^*}) \quad (8)$$

According to the conditions in Section 3.1, the initial position s_r , the pickup place of each task, and the drop place at the tail of the queue belong to the empty-state reachable place set of the corresponding robot. In addition, by the second property of the Reasoner-Output and the reachability assumed and verified in Section 3.1, a finite route exists for both the empty-travel segment and the loaded segment for an eligible candidate. Therefore, while U is not empty, at least one candidate with a finite S exists, and the allocation conditions are satisfied. If several candidates have the minimum value, one of them is selected;

the main experiments select one at random (Section 4.2), and a deterministic capacity-fit tie-breaking rule is examined as a separate configuration.

The selected task t^* is appended to the end of Q_{r^*} , the planned reference position of robot r^* is updated to the drop place of task t^* , and t^* is removed from U . This procedure reflects the deadhead due to the preceding tasks in the selection criterion, but it is an append-only scheme that does not change the order of an already constructed queue or move tasks between robots. Therefore, no route search of the TSP/VRP form over the entire task permutation is performed.

The whole procedure is given in Algorithm 1.

Algorithm 1 Completion-time-based sequential allocation (append-only)

Input: task set T (with the pickup and drop places and the object o_t of each task), robot set R with the initial position s_r and payload capacity of each robot, eligible robot sets $A(\cdot)$, load-state graphs $G_r(\text{empty})$ and $G_r(o_t)$, weight w_p , travel speed v , loading and unloading time t_{load}

Output: allocation $Q = \{Q_r\}_{r \in R}$, list of unallocated tasks

- 1: for all r , $Q_r \leftarrow \langle \rangle$, $\text{pos}_r \leftarrow s_r$, $T_r \leftarrow 0$
- 2: $U \leftarrow \{t \in T : A(t) \neq \emptyset\}$; record tasks with $A(t) = \emptyset$ as unallocated
- 3: **while** $U \neq \emptyset$ **do**
- 4: $(r^*, t^*) \leftarrow \text{argmin}_{t \in U, r \in A(t)} S(r, t \mid Q_r)$ ▷ Equation (7)
- 5: $T_{r^*} \leftarrow T_{r^*} + \tau(r^*, t^* \mid Q_{r^*})$
- 6: append t^* to Q_{r^*} ; $\text{pos}_{r^*} \leftarrow d_{t^*}$; $U \leftarrow U \setminus \{t^*\}$
- 7: **end while**
- 8: **return** Q , list of unallocated tasks

Algorithm 1 is not an optimization algorithm that guarantees the global minimum of Equation (6) but a sequential heuristic that determines the allocation at each step using the expected completion time. With $w_p = 0$, it is the Min-Min rule for mapping independent tasks onto heterogeneous machines [46,47], applied to completion times that depend on the queue end position. Equivalently, it is an SSI auction with the MiniMax bidding rule [48] in which each robot $r \in A(t)$ bids the completion time of its queue after appending t ; the proposed criterion adds the payload-margin term to this bid.

For any $w_p \geq 0$, Algorithm 1 is also the sequential greedy algorithm of Choi et al. [1] with the positive pair score $\bar{S} - S(r, t \mid Q_r)$, where $\bar{S} > \max S$ is a constant. This score has their diminishing marginal gain property on all three testbeds (Appendix B). By their Theorem 1, CBBA with this append-only score, a bundle limit of at least $|T|$, and a common deterministic tie-breaking rule therefore converges to the allocation of Algorithm 1 under the same rule, over any connected static communication network with synchronized conflict resolution. The equivalence does not cover the random tie-breaking of the main experiments or the standard CBBA, which scores a task at its best insertion position in the queue.

3.4. Plan Validation

For the empty-travel and loaded segments of each allocation, a shortest path is generated on the route planning graph, and the sequence of places visited is obtained together with the distance. When several shortest paths exist, the path whose sequence of place identifiers is lexicographically smallest is selected deterministically, and this rule is applied identically to all compared configurations. In the proposed method, this route planning graph is the same as the load-state graph used to compute the route cost in Equation (4). If $r \in A(t)$, the existence of a pickup–drop route that is traversable in the loaded state is guaranteed, but whether the specific route actually returned by the planner lies within that graph must be checked separately.

Let $\pi_{r,t}$ be the sequence of places visited by the loaded-segment route planned for task t by the allocated robot r . A violation is determined as follows:

$$V(r, t) = 1 \iff \exists p' \in \pi_{r,t} : p' \notin P_r(o_t) \quad (9)$$

In Equation (9), a violation occurs if the planned loaded route contains even one place outside the reachable place set $P_r(o_t)$, and $V(r, t) = 0$ if all places belong to it. Because the planned route is a route in the environment graph G , this determination is equivalent to whether $\pi_{r,t}$ is a route in the induced subgraph $G_r(o_t)$.

Route planning that does not reflect the load state can include in its shortest path a place that is not traversable in the loaded state, and in that case, $V(r, t) = 1$. In contrast, the proposed method generates the loaded-segment route on $G_r(o_t)$, so all places of the returned route belong to $P_r(o_t)$, and $V(r, t) = 0$ by construction. Computing C_{route} for the loaded segment on the same graph is intended to match the distance evaluated in the allocation with the state condition of the route that is subsequently planned and validated. The validation target is limited to the pickup–drop loaded segment, and the empty-travel segment is planned on $G_r(\text{empty})$, so there is no graph change due to loading.

The validation determines $V(r, t)$ for each allocation (r, t) in the output Q of the allocation procedure. This procedure is not limited to the proposed method but is applied identically to all configurations compared in Section 4.2. $V(r, t) = 1$ does not mean that the robot–task pair itself cannot be performed but that the planned loaded route deviates from the condition. Therefore, the task is not executed with the current route and is not counted as completed in the makespan calculation, while the robot–task allocation itself is retained and is not reallocated. This handling takes place before dispatch, not as replanning during execution. The recorded violations become the task-level observations of the violation rate metric defined in Section 4.2. The same section also defines configurations that keep the allocation of a configuration with violations but plan its loaded routes on $G_r(o_t)$, so that all tasks are completed. Pseudocode for building the load-state distance tables and for the validation is given in Appendix A.

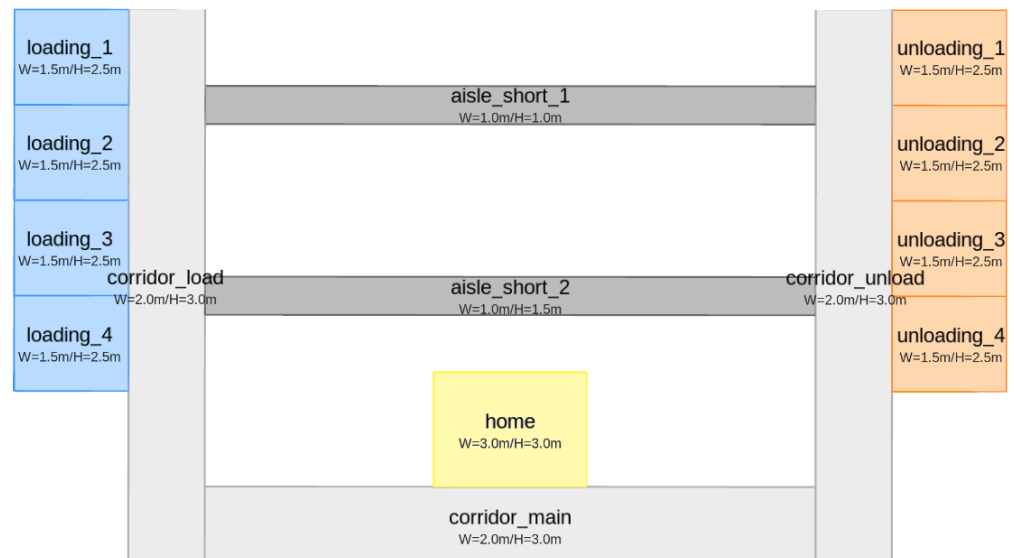
4. Experimental Setup

4.1. Testbeds, Robot Fleet, and Tasks

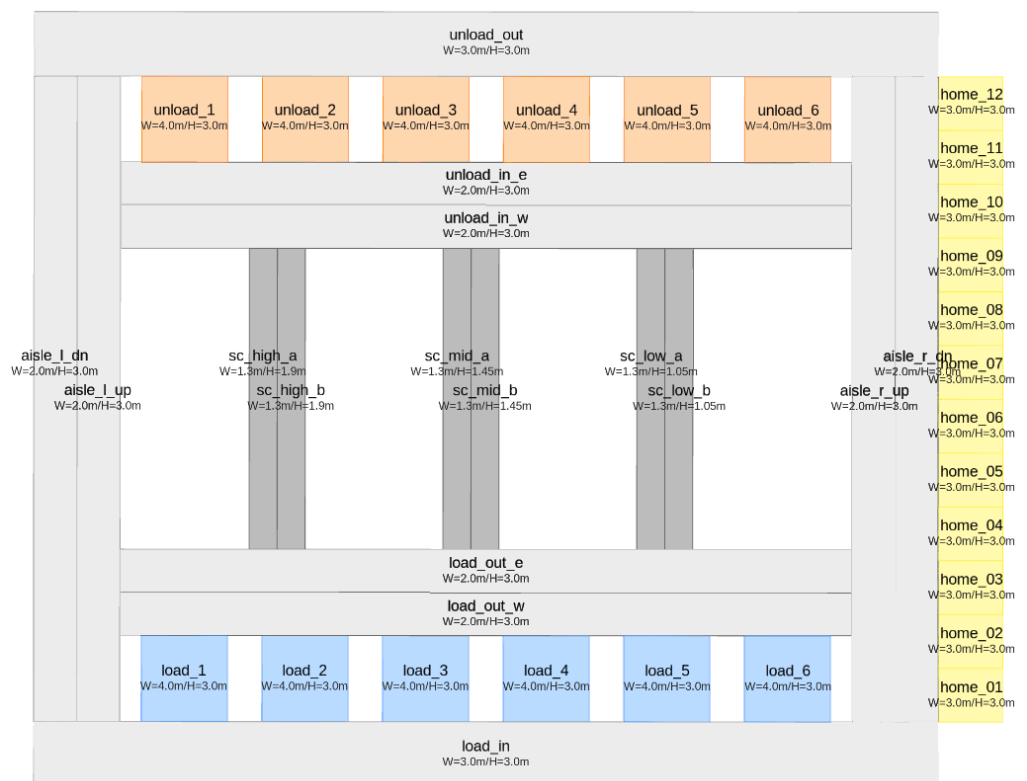
The experimental environment was constructed such that traversability varies with the load state, while a detour route that is traversable in the loaded state remains for every eligible robot–task pair. Therefore, the violations observed below do not indicate that the pair itself cannot be performed; rather, they indicate a mismatch between route planning that does not reflect the load state and the loaded graph. The robots and tasks were constructed such that the number of candidate robots differs across tasks, while factors that are not comparison axes of this study, such as travel performance, were set identically for all robot classes in the main experiments and thereby controlled.

Testbeds. The main experimental environment, testbed_A, is a warehouse-type layout of approximately 26 m × 15 m consisting of 14 places, as shown in Figure 2a. The home at the bottom center is a robot standby area that is connected only to the main aisle. Four loading places for pickup on the left and four unloading places for drop on the right are arranged side by side from north to south. Each group is bounded by a vertical access aisle, and the two access aisles are linked by three cross aisles. The main aisle at the bottom has a traversable cross-section of 2.0 m in width and 3.0 m in height, and the pickup and drop places are 1.5 m in width and 2.5 m in height. This cross-section was set such that every eligible robot–task pair generated in the experiments can pass through even in the loaded state. The two aisles above it are shortcuts that cross from left to right, with a

common traversable width of 1.0 m and heights of 1.0 m and 1.5 m, respectively. Therefore, whether a shortcut is traversable depends on the cross-section formed by the robot and its load, and even for blocked combinations a route via the main aisle remains. The auxiliary environment testbed_B has the same place composition, connections, and edge lengths as testbed_A, but the traversable cross-sections of the shortcuts were enlarged such that the reachable place sets in the loaded and empty states coincide for every eligible robot–task pair.



(a)



(b)

loading unloading home aisle shortcut

Figure 2. Experimental testbed layouts. (a) testbed_A place topology. (b) testbed_C place topology. Each place is labeled with its name and its traversable width (W) and height (H).

The third environment, testbed_C, is a warehouse-type layout of approximately 45 m × 36 m consisting of 40 places (Figure 2b). Whereas the small testbed_A keeps every violation traceable to its cause, testbed_C is used to check the conclusions on a larger layout with a different topology (Section 5.3) and for the Gazebo simulation (Section 5.4). It has six loading places and six unloading places, two-lane arterial aisles along its left and right sides, and three two-lane shortcuts across the center with a common traversable width of 1.3 m and heights of 1.9 m, 1.45 m, and 1.05 m, so fewer shortcuts remain usable as the loaded height increases. All other places are at least 2.0 m wide and 3.0 m high and admit every loaded cross-section. Each lane is one-way, so the route graph is directed; every pickup place can be reached from every home place and every drop place. Twelve home places along the outer lane of the right arterial aisle each hold four robots of one class, and the fleet, the task types, and the task generator are the same as on testbed_A.

Robot Fleet. The robot fleet consists of seven classes, as shown in Table 2. The payload capacities of H_load, LIFT_M, convey, and mani are taken from the earlier study [4], where these classes are named HEAVYLOAD_AMR, LIFT_AMR, CONVEYOR_AMR, and MANIPULATOR_AMR, respectively. F_lift, LIFT_L, and LIFT_S were added so that the candidate robots for the rack and pallet types decrease stepwise as the object weight increases (Table 3). The class composition ratio is fixed at one each for H_load, F_lift, and LIFT_L, two each for LIFT_M, convey, and mani, and three for LIFT_S. The main experiments are conducted with a 48-robot fleet that scales this by four. This ratio and the capacities of the three added classes are design choices; among the LIFT classes, lower-capacity classes are more numerous because lighter rack types are more frequent. Because all compared configurations share the same class composition ratio, the fleet composition does not vary between the comparison conditions. Each robot provides only one handling method, and in the main experiments, the travel speed is the same for all classes. Robot length is reported for completeness; only the width and the height are used to determine the traversability of places. Even in the empty state, traversability can be limited by the dimensions of the robot alone. On testbed_A, H_load, with a width of 1.2 m, cannot pass through either shortcut, and mani, with a height of 1.2 m, cannot pass through the shortcut whose height is 1.0 m, while the remaining five classes traverse all places. On testbed_C, whose shortcuts are 1.3 m wide, H_load and mani are blocked only by the 1.05 m shortcut. The planned initial position s_r of every robot is set to the home place on testbed_A and testbed_B and to one of the twelve home places described above on testbed_C. This is a condition of the allocation model in which several robots can depart from the same place, and the analytical evaluation does not model the physical arrangement that the robots simultaneously occupy within it. The physical initial pose is specified only for the robots actually spawned in Gazebo.

Table 2. Robot fleet specifications.

Class	W × L × H (m)	Payload Capacity (kg)	Handling Method	Composition Ratio
H_load	1.2 × 1.5 × 1.2	2000	MANUAL_LOAD	1
F_lift	0.9 × 1.2 × 0.4	1500	FORK	1
LIFT_L	0.8 × 1.0 × 0.5	1000	LIFT	1
LIFT_M	0.6 × 0.8 × 0.3	500	LIFT	2
LIFT_S	0.5 × 0.6 × 0.3	100	LIFT	3
convey	0.5 × 0.7 × 0.5	100	CONVEY	2
mani	0.5 × 0.7 × 1.2	20	MANIPULATE	2

Tasks. The tasks were designed as eight types, as shown in Table 3. All tasks have the same taskType, TRANSPORT, and all seven classes in the fleet provide the TRANSPORT capability, so the capability axis is uniform over all robot–task pairs. The types are

distinguished by the dimensions and weight of the object and the set of required handling methods. Types with the same set of required handling methods form one series, and within a series, the weights were placed at the boundaries of the robot payload capacities and in the intervals between them so that the candidate robots decrease stepwise.

Table 3. Task type specifications.

Type	Object W × L × H (m)	Weight (kg)	Required Object Handling Method	Frequency (%)
box_5	0.5 × 0.5 × 0.4	5	MANIPULATE, CONVEY, MANUAL_LOAD	20
box_20	0.5 × 0.5 × 0.4	20	MANIPULATE, CONVEY, MANUAL_LOAD	10
box_30	0.5 × 0.5 × 0.4	30	MANIPULATE, CONVEY, MANUAL_LOAD	10
rack_80	0.6 × 0.6 × 0.4	80	LIFT	25
rack_350	0.8 × 0.8 × 1.1	350	LIFT	15
rack_750	1.1 × 1.1 × 1.8	750	LIFT	5
pallet_1400	1.4 × 1.2 × 0.2	1400	FORK, MANUAL_LOAD	10
pallet_1800	1.4 × 1.2 × 0.2	1800	FORK, MANUAL_LOAD	5

Along the weight axis, the candidate robots for the rack series decrease from three classes to one, and those for the pallet series from two classes to one. In the box series, the weight of box_20 equals the capacity of mani, so the candidates are retained, and mani drops out at box_30. The traversable cross-section in the loaded state is computed from the dimensions of the robot and the object, with the width taken as the larger of the two and the height as the sum of the two. For the MANIPULATE method, which grasps the object within the external form of the robot, the height is also taken as the larger of the two. Along the dimension axis, the traversability condition in the loaded state differs between series. For rack, the object dimensions grow together with the weight, so the payload requirement and the traversal difficulty increase in the same direction. For pallet, the loaded width exceeds the shortcut width, but the empty-body width of F_lift does not, so the traversability conditions of the empty-travel and loaded segments differ even within the same task. For box, the traversability condition in the loaded state is the same as that in the empty state for any robot on testbed_A; on testbed_C, H_load carrying a box (loaded height 1.6 m) can no longer use the 1.45 m shortcut.

The occurrence frequency of each series was fixed at 40% for box, 45% for rack, and 15% for pallet, as shown in Table 3. This was set to approximate the ratio 5:6:2 of the numbers of robots whose handling method matches each series, so that the workload does not become heavily skewed between series. Within each series, the frequency does not increase with the weight. The frequencies are a design assumption rather than a measured order profile; Section 5.2 varies the mixture across series (Figure 3). The task type is drawn independently for each task from a distribution whose probabilities are these frequencies. The pickup place and the drop place are drawn independently and uniformly from the loading places and the unloading places, respectively, with duplication between tasks allowed. The number of tasks generated at once follows the load level, defined as the number of tasks per robot, and is swept over {0.5, 1, 2, 3, 4}, giving 24, 48, 96, 144, and 192 tasks for the 48-robot fleet, respectively.

Under the weight arrangement described above, every type has at least one candidate robot, and that robot can reach the pickup and drop places via the main aisle on testbed_A and testbed_B and via the arterial aisles on testbed_C, so no task with $A(t) = \emptyset$ arises. This is a setting intended to keep tasks that no robot can perform from dominating the comparison between configurations. The robot and task attribute values above and the traversable cross-sections of the places are input to the semantic reasoner of Section 3.1, which produces the eligible robot set $A(t)$ and the load-state reachable place sets.

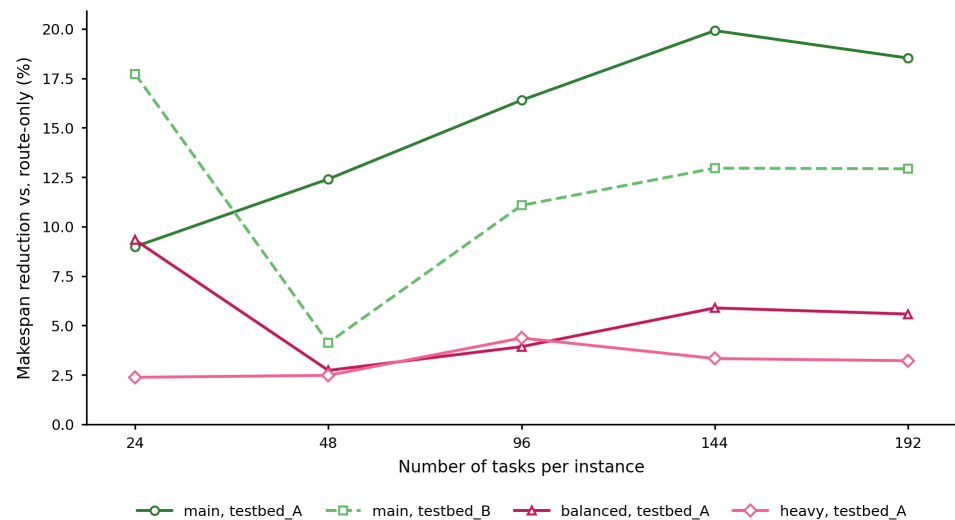


Figure 3. Makespan effect of the payload term under testbed and task-mixture variations. The values are the makespan reduction of the proposed method relative to route-only on the common completed-task set. The task mixtures are 40/45/15, 34/33/33, and 20/30/50 for main, balanced, and heavy, respectively, in terms of box/rack/pallet, and the proportions within each series were kept the same. On testbed_B, baseline, route-only, and dispatch-aware yielded the same result, so these values are the same as those relative to baseline. The five comparisons on testbed_B are listed with their statistics in Section 5.1.

4.2. Compared Configurations and Evaluation Protocol

Based on the environment and tasks described in Section 4.1, this section defines the configurations to be compared and the evaluation protocol. It describes the compared configurations, instance generation rules, time model and metrics, and evaluation layers in turn.

Compared Configurations. The main experiments compare five configurations on testbed_A: a 2×2 ablation in which the two axes of the cost in Equation (3) are switched independently, and one attribution control configuration. The settings for each configuration are listed in Table 4. The route axis switches whether the loaded-segment distance of C_{route} is computed on the loaded graph $G_r(o_t)$ or on the empty-state graph $G_r(\text{empty})$, and the payload axis switches whether w_p is positive or zero. These experiments use $w_p = 20$, a value calibrated in the sensitivity analysis of w_p in Section 5.2. The empty-travel segment route is generated on $G_r(\text{empty})$ in all configurations, and in the four configurations of the 2×2 the route planning graph for the loaded segment is matched to the graph used by the route cost. The fifth configuration, dispatch-aware, is a deliberate exception to this matching and performs only route planning for the loaded segment on the loaded graph.

The single difference of dispatch-aware relative to baseline in Table 4 makes it possible to separate whether the removal of violations results from reflecting the loaded distance in the allocation cost or from generating the route on the loaded graph. This configuration is analyzed in Sections 5.1 and 5.2, where the auxiliary experiments are also analyzed. The five configurations share the eligible robot set $A(t)$, the allocation procedure of Algorithm 1, and the pre-execution validation; they differ only in the two graphs and w_p .

Three further configurations are compared (lower part of Table 4). Capacity-fit (CF) tie-break is route-only with deterministic tie-breaking: among the pairs with the minimum S , it selects the one with the smallest C_{payload} and then the smallest robot and task indices. It corresponds to the limit $w_p \rightarrow 0^+$, so its difference from the proposed method shows the effect of the payload term beyond tie-breaking, apart from how the remaining ties are resolved. dispatch-aware + payload keeps the allocation of payload-only and plans its loaded routes on $G_r(o_t)$, as dispatch-aware does for baseline, so it differs from the proposed

method only in the cost graph for the loaded segment. The uniform-rule configuration emulates a single loaded rule common to all robots and tasks [28]: the places that are blocked in the loaded state for at least one eligible robot–object combination form a set V_{res} , and the loaded segment of every robot uses $G_r^U = G[P_r(\text{empty}) \setminus V_{\text{res}}]$ for the cost and the route planning, with $w_p = 20$. Because $P_r(\text{empty}) \setminus V_{\text{res}} \subseteq P_r(o_t)$, this configuration has no violations, and no loaded route is shorter than on $G_r(o_t)$. It is run on testbed_A only.

Table 4. Compared configurations.

Configuration	Cost Graph for the Loaded Segment	w_p	Route Planning Graph for the Loaded Segment	Role
baseline	$G_r(\text{empty})$	0	$G_r(\text{empty})$	Load-state-agnostic reference
route-only	$G_r(o_t)$	0	$G_r(o_t)$	Route axis alone
payload-only	$G_r(\text{empty})$	20	$G_r(\text{empty})$	Payload axis alone
proposed method	$G_r(o_t)$	20	$G_r(o_t)$	Both axes combined
dispatch-aware	$G_r(\text{empty})$	0	$G_r(o_t)$	Planning graph decoupled from cost
CF tie-break	$G_r(o_t)$	0	$G_r(o_t)$	Deterministic capacity-fit tie-breaking of route-only
dispatch-aware + payload	$G_r(\text{empty})$	20	$G_r(o_t)$	Payload-only with loaded-graph planning
uniform-rule	G_r^U	20	G_r^U	Single loaded rule for all pairs

A fixed testbed and fleet combined with a randomly generated task set constitute one instance. $N = 200$ instances are generated at each load level, with the random seed given as a base value plus a serial number. The results of both the main and the auxiliary experiments were obtained from this series, and a series with a different base value was used to calibrate w_p . In the selection step of Algorithm 1, several candidates with the minimum value appear in most cases, so one of them is chosen randomly, except in CF tie-break. The seed used here is separated from the task set generation, and the results are aggregated over 25 repetitions on the same task set (10 repetitions in the w_p calibration series). The generation rules, the base values of all series, and the tie-breaking seeds are given in the Supplementary Materials. Because all configurations are evaluated on the same fixed task set and initial conditions, the comparison between configurations is a paired comparison that shares the same conditions. The tie-breaking seed is also shared by all configurations that break ties at random, but what is shared is the starting point of the random sequence, so for configuration pairs with different allocation costs the consumption diverges from the point at which the tie sets differ. The 25 repetitions are intended to reduce the randomness that remains in this way. In the auxiliary experiments, testbed_B has no shortcut constraint, so $G_r(o_t) = G_r(\text{empty})$ holds for every evaluated pair. Therefore, two configurations in which only the payload term is switched, excluding the route axis, are compared using the same protocol, together with CF tie-break.

Time Model and Metrics. The travel time is the planned route distance, that is, the sum of the empty-travel and loaded segment distances, divided by the travel speed $v = 1$ m/s, which is common to all classes in the main experiments. Each task has a loading and unloading time of $t_{\text{load}} = 5$ s at the pickup and drop, set as a single constant regardless of the object or the handling method; Section 5.3 varies the speed by class and t_{load} by object series. The makespan reported in the evaluation follows the same definition as the objective $M(Q)$ of the allocation problem. This model is an analytical completion-time model shared by the allocation and the evaluation, and it does not include acceleration and deceleration, rotational dynamics, interference between robots, local avoidance delay, or charging. These elements are excluded from all configurations alike, so the comparisons in Sections 5.1–5.3

are made under the same interference-free execution assumption, except for the congestion-aware replay at the end of Section 5.3.

The first metric, the loaded-route violation rate, is the proportion of allocations judged as $V(r, t) = 1$ by Equation (9) in one instance, and the denominator is the total number of tasks in that instance. This metric counts violations of the planned loaded route, not the infeasibility of a robot–task pair.

The second metric, the makespan, is compared as the difference between configuration pairs that share the same instance. Tasks judged as violations in the pre-execution validation are not counted as completed, so Equation (6) is applied to the subsequence that skips those tasks while the relative order of each Q_r is maintained. The empty-travel segment of a subsequent task starts from the drop place of the last retained task, or from s_r if there is no such task. This recomputation does not minimize $M(Q)$ again but reconstructs the completion times for evaluation by reflecting the validation results. Accordingly, the makespan of a configuration with violations is computed only on the tasks it completes and, therefore, has a bias in the direction favorable to that configuration. In the paired comparison, the intersection of the tasks completed by both configurations, that is, the common completed-task set, is taken. Tasks outside this set are removed from the queue of each configuration while the relative order of the remaining tasks is maintained, and the makespan is recomputed by recalculating the empty travel with the removed tasks skipped. Because the empty travel is chained along the queue, this bias is not removed simply by subtracting some tasks from the completion time records. The common completed-task set still does not count the work that a configuration with violations leaves undone; the configurations without violations complete all tasks, and their makespans are compared on the full task set.

The third metric, the excess loaded-route distance Δd^L , is the amount by which the loaded-segment distance of the allocated robot exceeds the minimum within the eligible robot set:

$$\Delta d^L(t) = \text{sp}_{G_{r^*}(o_t)}(p_t, d_t) - \min_{r \in A(t)} \text{sp}_{G_r(o_t)}(p_t, d_t) \quad (10)$$

In Equation (10), r^* is the robot that performs task t in allocation Q . This metric does not use the route actually planned by each configuration but the shortest distance of the allocated robot evaluated on the loaded graph, so it isolates only the excess distance that the robot selection carries from the perspective of the loaded route. Δd^L is not an excess with respect to the selection criterion as a whole. The selection criterion includes the accumulated workload of the queue, the travel time and the loading and unloading time of the new task, and the payload margin. Therefore, the selection of a robot whose loaded segment is not the shortest does not in itself mean an inferior selection under that criterion. This metric is computed per task, and Section 5.2 reports the proportion of tasks with no excess and the mean excess distance over the tasks in which an excess occurred. The same section also reports, from the allocation before validation, the mean C_{payload} of the allocated pairs; the share of capacity-fit selections, in which the allocated robot has the smallest payload capacity in $A(t)$; and the mean payload margin in kg. The mean excess distance is pooled over all tasks with an excess at each load level; the other indicators are averaged over the 25 repetitions of an instance and then over the instances.

The fourth metric, the prediction error, is the completion time of a robot recomputed on its planned routes minus that computed in the allocation by Equation (6). It is zero by construction when the cost and the route of the loaded segment use the same graph, and it is reported per robot and for the makespan.

The statistical unit is the instance, not the individual task or repetition. For each instance, a metric is first averaged over the 25 tie-breaking repetitions (CF tie-break is

deterministic and run once), and the paired differences, confidence intervals, and tests are computed over these instance means because the repetitions of an instance are not independent; the variability due to tie-breaking therefore enters the uncertainty estimates through these means. At each load level, the per-instance difference is obtained for the configuration pairs specified in advance. Together with the mean difference, a 95% confidence interval (CI) is obtained by a percentile bootstrap that resamples instances with replacement and recomputes the mean of each configuration on the same resampled instances [49]. A comparison with a difference of exactly zero over all instances is regarded as a tie and is excluded from the tests. The significance of the remaining comparisons is evaluated by the Wilcoxon signed-rank test on the per-instance differences, computed with the normal approximation after excluding instances with zero difference [50]. The tested comparisons among the five configurations in the upper part of Table 4 on testbed_A and testbed_B form one family, to which the Holm correction is applied [51]; the comparisons of each further analysis form separate families and, for each measure of the queues in Section 5.2, the comparisons on testbed_A and testbed_C form one family, all corrected in the same way. Because the number of instances with a nonzero difference can be small, exact signed-rank p -values, which assume no ties among the absolute differences, and sign-flip permutation p -values of the mean difference (100,000 permutations) were also computed and Holm-corrected within the same families; where they lead to a different conclusion, this is stated with the result. The effect size is reported as the matched-pairs rank-biserial correlation r [52].

Evaluation Layers. The quantitative evaluation in Sections 5.1–5.3 is carried out on a statistical harness that integrates the semantic reasoning results, load-state shortest distances, allocation, pre-execution validation, and time model. At the end of Section 5.3, the allocations and routes of the harness are also replayed in a discrete-event simulation that adds waiting at the loading and unloading places. Section 5.4 executes the queues produced by the same semantic input and by the same cost, allocation, and route generation pipeline in a ROS 2 and Gazebo simulation environment. It verifies the consistency between the load-state route model and the simulated travel. The Gazebo results are used not as a performance comparison between configurations but as a pipeline-level consistency check. The experiments were performed on an Ubuntu 24.04 machine with an Intel Core i9-9900K and 32 GB of memory.

5. Experimental Results and Analysis

5.1. Ablation Results

This section uses the 2×2 ablation and the dispatch-aware control configuration to compare whether loaded-route violations arise in route planning that does not consider the load state and how the two axes affect the makespan. It first evaluates whether the loaded-segment route generated by each configuration is consistent with the loaded graph using the loaded-route violation rate and then reports the makespan differences by paired comparison over the same instances; finally, it compares the proposed method on the full task set with the three configurations in the lower part of Table 4. Section 5.2 analyzes the selection mechanisms in the allocation step that are related to the observed differences, together with their dependence on conditions.

Loaded-Route Violation Rate. Table 5 shows the loaded-route violation rate by load level. In baseline and payload-only, which generate the loaded-segment route on $G_r(\text{empty})$, the planned route was not contained in $G_r(o_t)$ for approximately 25% and 23% of the tasks, respectively. In route-only, the proposed method, and dispatch-aware, which generate the loaded-segment route on $G_r(o_t)$, no violation occurred at any of the five load levels. The configuration with only the payload term added showed a violation rate only

1.6 to 2.0 percentage points lower than baseline, and most of the violations remained when the loaded-segment route was generated on $G_r(\text{empty})$.

Table 5. Loaded-route violation rate of the configurations that plan loaded routes on the empty-state graph, by load level, in percent (testbed_A).

Configuration	24	48	96	144	192
baseline	25.33	24.66	24.48	24.54	24.53
payload-only	23.71	22.66	22.61	22.71	22.59

The column headers indicate the number of tasks per instance. Each value is the proportion of violating tasks, averaged over the 25 tie-breaking seeds and then over the 200 instances at each load level. The configurations that plan the loaded route on $G_r(o_t)$ or G_r^U had no violations at any load and are omitted.

The violation rate remained essentially unchanged across load levels. Even when the number of tasks increased eightfold from 24 to 192, the violation rate of baseline stayed between 24.5% and 25.3%. Whether an individual allocation is a violation is independent of the queue length and the preceding tasks once the combination of robot, object, pickup, and drop is fixed, and it is determined by the loaded cross-section and the places through which the empty-state shortest route passes. For the same combination of robot, object, pickup, and drop, the same determination recurs under the fixed route selection rule.

Baseline violations occurred in only four robot–object combinations (Table 6). In three of them, the loaded width or height exceeds the traversable dimensions of both shortcuts, and they produced violations in all 16 allocated pickup–drop combinations. When F_lift carries pallet_1400, the loaded width of 1.4 m exceeds the shortcut width of 1.0 m, and when LIFT_L carries rack_750 or rack_350, the loaded height is 2.3 m and 1.6 m, respectively, and exceeds the heights of both shortcuts. In the remaining combination, LIFT_M–rack_350, the loaded height of 1.4 m lies between the heights of the two shortcuts, so the outcome depends on which shortcut the empty-state shortest route passes through. Violations occurred in every instance for 10 of the 16 pickup–drop pairs and in none for the other six, and no pair showed a split outcome. The set of combinations was maintained even when the load increased eightfold, and the share of each combination in the total violations remained within a similar range.

Table 6. Violation breakdown by robot–object combination (testbed_A, baseline).

Robot	Object	Loaded Cross-Section $W \times H$ (m)	Violation Rate (%)	Share of Total Violations (%)
F_lift	pallet_1400	1.4×0.6	100.0	35.4–38.6
LIFT_M	rack_350	0.8×1.4	65.1–66.1	26.1–30.6
LIFT_L	rack_750	1.1×2.3	100.0	19.0–20.5
LIFT_L	rack_350	0.8×1.6	100.0	13.4–16.3

The ranges are the minimum and maximum over the loads 24–192. The denominator of the violation rate is the number of tasks allocated to that combination, and the denominator of the share is the total number of violations. The traversable cross-sections of the shortcuts are 1.0×1.0 m and 1.0×1.5 m.

The absence of violations in the configurations that plan the loaded route on $G_r(o_t)$ is a property expected by construction, and the nonzero rates of baseline and payload-only are the consequence of planning on a single load-invariant graph. Avoiding these violations requires only replacing the input graph of the shortest path algorithm, without additional sensing or re-inference, because the load-state traversability is already contained in the output of the semantic reasoner.

dispatch-aware shows that this property is obtained without changing the allocation. The route planning graph does not enter the input of the allocation step, so dispatch-aware and baseline produce the same robot–task mapping and queue order for the same task

set and the same tie-breaking seed. Although the allocation was the same, the violation rate dropped from 24.5–25.3% to zero. The removal of violations does not come from the allocation cost but from generating the route on the loaded graph and validating it on the same graph.

Paired Makespan Comparison. Table 7 shows the makespan differences recomputed on the tasks completed by both configurations of each pair sharing the same instance, by load level. This section covers the four pairs on testbed_A, and the last pair on testbed_B, where the route axis is neutralized, is examined together in Section 5.2. The effect of the payload term is most directly revealed in the comparison with route-only. The makespan of the proposed method was shorter, and the mean difference was 9.0 to 19.9%. The confidence intervals excluded zero at all five load levels, and the difference was larger at medium and high loads than at low loads. The comparison remained significant at all five load levels after the Holm correction, and the effect size was 0.96 or greater.

The comparison with baseline differs in both the route axis and the payload axis, but on the common completed-task set it does not count the work that baseline leaves undone (Section 4.2). At load 24, the mean difference was +0.95% with a confidence interval of [+0.61, +1.33]%, so the direction was stable but the magnitude was small, and at loads of 96 and above the lower bound of the confidence interval exceeded +16%. The difference from the configuration that uses only the payload term was smaller at the same loads than in the two preceding comparisons. At load 24, the difference was exactly zero over all 200 instances, so no test was applied, and at the other four loads the confidence intervals excluded zero but their upper bounds all remained below +6%. At load 48, only 31 instances had a nonzero difference, because in the other 169 instances the two configurations produced exactly the same makespan in all 25 repetitions. This comparison and the other 23 tested comparisons in Table 7 also remained significant under the exact signed-rank and sign-flip permutation tests. dispatch-aware produced no violations and completed all tasks, so the comparison with it evaluates the proposed method as a whole: the makespan of the proposed method was 6.7 to 20.1% shorter. The comparison with payload-only differs in both the cost and the route planning graph for the loaded segment, so the independent effect of the route term is not attributed on the basis of this result alone. The comparison with dispatch-aware has the same route planning graph and differs only in the allocation cost. The two mechanisms are diagnosed in Section 5.2.

A configuration with tasks judged as violations in the pre-execution validation has its makespan computed over fewer tasks, so it is difficult to compare directly. This is particularly the case for baseline, whose violations reach 25%. As described in Section 4.2, the paired comparison therefore retains only the tasks completed by both configurations and recalculates the empty travel while the relative order is maintained. In the raw makespan before this adjustment, payload-only was 1.6 to 9.0% shorter than the proposed method at all five load levels, and baseline was 7.9% shorter at load 24. These values include the effect of tasks excluded as violations dropping out of the completed set, so they cannot be used to compare efficiency between configurations directly. This advantage does not persist when the makespan is recomputed on the common completed-task set. The values in Table 7 are recomputed on the set of tasks completed by both configurations. The robot allocation and queue order of each configuration remain unchanged, and no reallocation or queue reconstruction is performed.

The four configurations of the 2×2 ablation showed different patterns on the two metrics. baseline and payload-only generated the route on a graph that does not reflect the load state and left violations of approximately 25% and 23%, respectively. route-only removed the violations, but its makespan was longer than that of the proposed method at

all five load levels. The proposed method, which has no violations, was also shorter than dispatch-aware at all five load levels; on the common completed-task set, it was shorter than baseline at all five load levels and than payload-only at loads 48–192, with a tie at load 24. However, these differences depend on the environment and the task mixture, as analyzed in Sections 5.2 and 5.3.

Table 7. Paired makespan comparison.

Comparison Pair (Switched Axis)	Load	$\Delta\%$	Diff (s)	95% CI	r	n_{eff}	p
vs. route-only (payload)	24	+8.99	+7.42	[+7.06, +11.00]	+0.996	121	1.8×10^{-20}
	48	+12.41	+17.72	[+10.97, +13.82]	+0.961	197	1.9×10^{-30}
	96	+16.40	+38.32	[+15.13, +17.71]	+0.993	200	7.4×10^{-33}
	144	+19.92	+65.99	[+18.86, +20.95]	+1.000	200	3.5×10^{-33}
	192	+18.53	+77.21	[+17.57, +19.47]	+0.998	200	3.9×10^{-33}
vs. baseline (route + payload)	24	+0.95	+0.66	[+0.61, +1.33]	+1.000	52	1.8×10^{-9}
	48	+8.60	+10.80	[+6.61, +10.58]	+0.557	177	9.2×10^{-10}
	96	+17.83	+39.61	[+16.32, +19.26]	+0.938	198	2.9×10^{-29}
	144	+20.51	+65.24	[+19.28, +21.72]	+0.994	200	6.9×10^{-33}
	192	+18.29	+73.44	[+17.16, +19.39]	+0.992	200	7.5×10^{-33}
vs. payload-only (route)	24	0.00	0.00	—	—	0	tie
	48	+2.70	+3.18	[+1.56, +3.98]	+0.974	31	4.4×10^{-6}
	96	+4.10	+7.81	[+2.75, +5.48]	+0.635	88	7.0×10^{-7}
	144	+1.61	+4.13	[+0.80, +2.47]	+0.525	141	2.6×10^{-7}
	192	+1.55	+5.18	[+0.71, +2.39]	+0.282	173	1.3×10^{-3}
vs. dispatch-aware (allocation cost)	24	+6.69	+5.38	[+5.12, +8.38]	+0.997	115	1.4×10^{-19}
	48	+12.68	+18.17	[+11.29, +14.05]	+0.976	189	3.7×10^{-30}
	96	+16.73	+39.24	[+15.44, +18.02]	+0.994	200	6.9×10^{-33}
	144	+20.06	+66.56	[+19.05, +21.06]	+1.000	200	3.5×10^{-33}
	192	+18.26	+75.83	[+17.34, +19.16]	+0.999	200	3.6×10^{-33}
vs. route-only, testbed_B (payload)	24	+17.73	+13.43	[+15.44, +20.00]	+1.000	144	2.2×10^{-24}
	48	+4.13	+4.91	[+2.86, +5.51]	+0.531	191	1.2×10^{-9}
	96	+11.08	+21.39	[+9.65, +12.46]	+0.903	199	2.6×10^{-27}
	144	+12.96	+34.76	[+11.96, +13.96]	+0.990	199	1.5×10^{-32}
	192	+12.93	+43.73	[+12.10, +13.76]	+0.996	199	7.0×10^{-33}

The parentheses indicate the axes that were switched between the proposed method and the comparison configuration, and the settings of each axis are listed in Table 4. The aggregation and tests follow Section 4.2, and only the last pair is on testbed_B. diff is the comparison configuration minus the proposed method in seconds, $\Delta\%$ is $100 \times \text{diff}$ divided by the mean makespan of the comparison configuration, and a positive value indicates that the proposed method is shorter. Both values are computed from the mean of each configuration and are not obtained by averaging per-instance ratios. The CI is a percentile bootstrap for $\Delta\%$ ($B = 10,000$). n_{eff} is the number of instances with a nonzero difference (total $N = 200$). p is the Holm-corrected value of the normal approximation with the 24 comparisons in this table, excluding the tie, as the family.

Comparisons on the Full Task Set. Table 8 compares the proposed method with the three configurations in the lower part of Table 4, which complete all tasks without violations. The makespan of the proposed method differed from that of dispatch-aware + payload by less than 1.2% at every load level, with one significant difference in each direction. The proposed method was 4.3 to 11.4% shorter than uniform-rule at all five load levels. CF tie-break was 1.8 to 9.0% shorter than route-only, but the proposed method was 8.7 to 17.9% shorter than CF tie-break at loads 48–192, and the two gave the same makespan in all instances at load 24. Relative to route-only, the gain of CF tie-break therefore amounted to 10 to 32% of that of the proposed method at loads 48–192. On testbed_B, CF tie-break

came within 0.34% of the proposed method at every load level, so its gain over route-only amounted to 97 to 100% of that of the proposed method, consistent with the plateau of testbed_B from $w_p = 1 \times 10^{-3}$ in Figure 4.

Table 8. Paired makespan comparison on the full task set with dispatch-aware + payload, uniform-rule, and CF tie-break.

Comparison	24	48	96	144	192
proposed method vs. dispatch-aware + payload	−0.03	−0.72 [†]	+0.36	+1.14 *	+0.68
proposed method vs. uniform-rule	+4.25 *	+11.38 *	+8.85 *	+9.34 *	+9.19 *
proposed method vs. CF tie-break	0.00 (tie)	+8.73 *	+14.26 *	+17.94 *	+17.03 *
CF tie-break vs. route-only	+8.99 *	+4.03 *	+2.50 *	+2.42 *	+1.81 *
proposed method vs. CF tie-break, testbed_B	0.00 (tie)	+0.06	+0.34 [‡]	+0.29	+0.07
CF tie-break vs. route-only, testbed_B	+17.73 *	+4.07 *	+10.78 *	+12.71 *	+12.87 *

The column headers indicate the number of tasks per instance. The values are $\Delta\%$ as defined in Table 7, relative to the mean makespan of the second configuration, on the full task set of testbed_A unless stated otherwise; a positive value indicates that the first configuration is shorter. * and [†]: significant after the Holm correction (normal approximation) with the first and the second configuration shorter, respectively. The family consists of the comparisons that involve the configurations in the lower part of Table 4, on testbed_A and testbed_B. [‡]: significant under the normal approximation and the exact test but not under the permutation test; more instances favored CF tie-break, although the mean favored the proposed method.

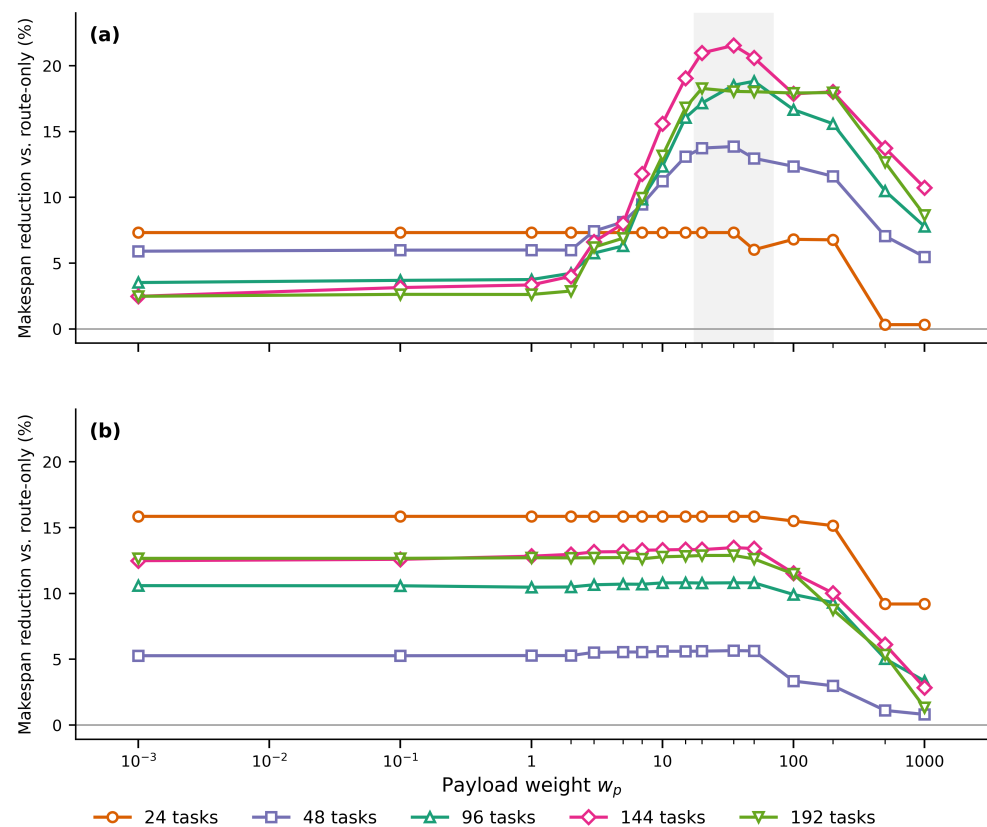


Figure 4. Sensitivity of the makespan improvement to the payload weight w_p . (a) testbed_A. (b) testbed_B. The values represent the makespan reduction of the proposed method relative to route-only. The x-axis is logarithmic and starts at $w_p = 1 \times 10^{-3}$: the values at 1×10^{-6} were identical to those at 1×10^{-3} at every load, and $w_p = 0$, where the difference is zero by definition, cannot be placed on this axis. In the testbed_A panel, the values {20, 35, 50}, whose mean over the five loads is within 1 percentage point of the maximum, are shaded. Because this figure was obtained from the instance series used for calibrating w_p , it does not exactly match the values in Section 5.1 even at the same w_p .

5.2. Attribution and Sensitivity Analysis

This section diagnoses the selection mechanisms in the allocation step that are related to the differences observed in Section 5.1, and examines the conditions under which the effects are maintained and the computational cost. Among the five configurations there are pairs that differ in only one setting, so the effect of each term is separated using those pairs.

Allocation-Side Mechanisms. Comparing dispatch-aware and route-only distinguishes what the route term of the allocation cost does. The two configurations have the same route planning and the same payload term, and differ only in the cost graph for the loaded segment. In Table 9, the proportion of tasks whose loaded-segment distance is the minimum among the eligible robots rose from 89.4–94.2% for dispatch-aware to 92.6–98.3% for route-only. However, this alignment did not appear in the makespan. The mean makespan difference between route-only and dispatch-aware, which have the same route planning conditions, was within 0.5% at loads 48–192, and at load 24, dispatch-aware was approximately 2.5% shorter. This comparison is reported only as descriptive statistics and was not included in the test family. In these experiments, the observed role of the route term was not the removal of violations but the matching of the loaded-segment distance used by the allocation to the graph on which the route is generated.

Table 9. Excess loaded-route distance and robot-selection indicators (testbed_A).

Configuration	$\Delta d^L = 0$ (%)	Mean over Tasks with an Excess (m)	Mean C_{payload}	Capacity-Fit Selection (%)	Mean Payload Margin (kg)
baseline = dispatch-aware	89.4–94.2	7.18–9.10	0.145–0.157	65.5–73.6	179–211
route-only	92.6–98.3	6.25–7.73	0.146–0.166	64.9–72.9	186–226
payload-only	95.9–99.6	5.42–8.96	0.062–0.092	82.0–88.0	85–132
proposed method	96.7–99.6	4.37–8.92	0.062–0.094	81.6–88.0	85–133

The ranges are the minimum and maximum over the loads 24–192. The indicators and their aggregation are defined in Section 4.2. The differences between configurations in the text were computed from the per-load values.

Excesses occur infrequently, but their individual magnitudes are not small. In baseline and dispatch-aware, tasks with an excess accounted for 11% or less, but the mean excess distance over those tasks corresponds to between one quarter and one third of 28.3 m, the mean loaded-segment shortest distance of the allocated robot. The route term lowers the frequency of excesses.

The payload term also reduces Δd^L . Comparing baseline and payload-only, tasks with $\Delta d^L = 0$ increased by 5.4 to 7.4 percentage points, which is larger than the 2.7 to 4.0 percentage points for switching the route-cost graph alone. Therefore, Δd^L is not a metric that responds only to the route term, and the preceding attribution does not rest on the metric itself but on the paired comparison in which the route planning conditions are the same. The direction in which the payload term shifts the robot selection is shown in the last three columns of Table 9. Moving from the two configurations with $w_p = 0$ to the two with $w_p = 20$, the mean C_{payload} decreased from 0.145–0.166 to 0.062–0.094 and the mean payload margin decreased from 179–226 kg to 85–133 kg. The proportion of capacity-fit selections increased. Because capacity-fit selections reach 56.2% even when each task is allocated to a robot drawn uniformly at random from $A(t)$, the shift produced by the payload term must be read as the difference between the two configurations. When only the route term was switched, these three metrics barely changed.

None of the metrics in Table 9 substitutes for the makespan. It is difficult to explain the makespan improvement in Section 5.1 based on the difference in the loaded-segment distance alone. Because the loading and unloading time is the same in all configurations, the remaining difference in the time model is interpreted as arising jointly from the loaded-segment distances of the selected robots, the deadhead chaining along the queue,

and the distribution of the workload across robots. The mean over the tasks with an excess was not used for size comparison between configurations because its population differs by configuration.

The role of the route term identified above appears in the prediction error defined in Section 4.2, which is zero by construction for route-only and the proposed method. For dispatch-aware, the allocation underestimated the completion time of a robot by 2.3 s on average at load 24 and by 8.9 s at load 192, and the makespan by 2.7 to 8.0%; dispatch-aware + payload showed similar errors. Because the proposed method and dispatch-aware + payload reached almost the same makespan (Table 8), the route term of the allocation cost, used together with the payload term, did not consistently shorten the makespan but made the completion times computed in the allocation match those of the planned routes; without the payload term, it lengthened the makespan on testbed_C (Section 5.3).

Sensitivity to Environment and Task Mixture. The magnitude of the improvement strongly depends on the environment and task mixture. Figure 3 shows the results obtained by varying these two parameters. The mean difference was positive in all twenty conditions tested, but the improvement narrowed from 9.0 to 19.9% for main (testbed_A) to 2.4 to 4.4% for heavy. For example, at load 192, it decreased from 18.5% to 3.2%, which is approximately one-sixth. testbed_B was 4.1 to 17.7%, and balanced was 2.7 to 9.3%.

The conditions under which the improvement narrows are related to the room for capacity reservation to become a gain. Not filling the queues of high-capacity robots with light tasks becomes a gain when heavy tasks that have not yet been allocated remain. In heavy, the share of the pallet series is large and those tasks are concentrated in a few robot classes, so fewer light tasks remain for which reserving the high-capacity robots would pay off.

Capacity reservation was checked by measuring the queues of the main experiments directly. A task allocated to a robot is counted as downgradable if a robot class with a smaller payload capacity is also in $A(t)$, that is, if the selection is not capacity-fit (Table 9). Under route-only, the share of downgradable tasks was 35.1% at load 24 and 27.1% at load 192, whereas under the proposed method it was 12.0% and 17.5%, and the difference was significant at all five load levels after the Holm correction. At load 192, the share of the tasks allocated to H_load, F_lift, LIFT_L, and LIFT_M fell from 48.1% to 38.4%, and the coefficient of variation of T_r over the robots fell from 0.33 to 0.20. The utilization of a class at this load, its busy time per robot divided by the makespan, fell for H_load, LIFT_L, and LIFT_M from 88.5, 77.7, and 65.1% to 87.2, 69.3, and 64.3% and rose for the other classes. At this load, the tasks that only one class can carry were allocated on average after 79% of the allocation steps under route-only and after 51% under the proposed method, and such a task was the final task of the robot that finished last in 91.6% and 18.6% of the instances, respectively. The planned departure times for these tasks were significantly earlier under the proposed method at all five load levels, at 76 s instead of 186 s on average at load 192. Without the payload term, these tasks therefore tend to be allocated later and to occupy later positions in the queues of the few robots that can carry them.

The results of Figure 3 show that the improvement from the payload term depends strongly on the task mixture. Within the range tested, the sign did not reverse, but there are conditions under which the improvement narrows to single-digit percentages, and whether the sign is maintained outside this range was not verified.

Sensitivity to the Payload Weight w_p . Figure 4 shows the improvement measured by varying w_p . On both testbeds, the improvement is largest at $w_p = 35$ and decreases at larger values, but the level and the width of the small- w_p region differ. On testbed_A, the mean over the five loads is almost unchanged at 4.3 to 4.9% in the region $0 < w_p \leq 2$

and then rises from 3 and reaches 15.5% at 20 and 15.8% at 35. On testbed_B, it already reaches 11.4% at $w_p = 1 \times 10^{-3}$, which is 97% of the maximum of 11.7%, stays in the range 11.4 to 11.7% up to 50, and decreases thereafter.

The fact that a difference appears in both environments merely by changing w_p from 0 to 1×10^{-3} is due to the tie structure described in Section 4.2. In the selection step, several candidates share the minimum value in most cases, so even a payload term of negligible magnitude resolves those ties toward the smaller payload margin. In both environments the values are almost unchanged in the region $0 < w_p \leq 2$. The flat response in this region indicates that the improvement is dominated by the tie-breaking effect rather than by the magnitude of the payload term. The environments differ in the region above this. On testbed_A the improvement begins to rise from $w_p = 3$, which suggests that the payload preference also acts on the cost differences between candidates, whereas on testbed_B, there is almost no change up to 50.

This study used $w_p = 20$ on testbed_A and testbed_B. The values whose mean $\Delta\%$ over the five loads on testbed_A is within 1 percentage point of the grid maximum (+15.85% at $w_p = 35$) were 20, 35, and 50, and the smallest of these, 20, was chosen. The smallest value was selected to minimize the intervention in the travel distance within this range. testbed_B was checked together as a constraint, but the mean difference was positive at every grid point with $w_p > 0$, so it did not restrict the choice. As seen above, there is almost no change with magnitude, so it does not provide a basis for choosing a value. Because w_p is a coefficient that converts the dimensionless payload margin into travel distance in meters, the appropriate range depends on the distance scale of the environment, and w_p was therefore recalibrated with the same procedure on testbed_C (Section 5.3). This 1 percentage point criterion was applied to the grid points examined, and the values between grid points (the intervals 20–35 and 35–50) were not examined.

Computational Cost. The proposed method performs semantic reasoning only once before execution and does not run the inference again during the subsequent allocation. Because only the task set changes while the robot classes and the object types are fixed, caching the reasoning results as a template and projecting them onto each instance is sufficient. The load-state shortest distances are likewise computed once in preprocessing, because the place graph is static.

The preprocessing before execution, which comprises the generation of the semantic reasoner output; the construction of the environment, fleet, and templates; and the precomputation of the load-state shortest distances, took 0.25 s on testbed_A and 0.82 s on testbed_C, excluding the loading of the Python modules. The only work required for each instance thereafter is the projection and the allocation, so about 109 ms was required for 192 tasks, the heaviest condition of the 48-robot fleet, as shown in Table 10. The five configurations use the same allocation algorithm and precomputed tables of the same size, so their computational complexity is the same. A comparison of execution time between configurations is not an object of analysis in this study. What is confirmed here is not a comparison with other methods but whether the method is within a practical range at the tested scale.

The times are wall-clock times of a single Python 3.12.3 process on one core of the machine described in Section 4.2; each preprocessing time is the mean of five runs, each in a new process. The SWRL rules are evaluated in the same process by a rule interpreter implemented in Python, which gave the same output as the OWL reasoner Pellet on all three testbeds. Each iteration of Algorithm 1 evaluates S for at most $|U||R|$ pairs using the precomputed distance tables, so the allocation requires $O(|T|^2|R|)$ evaluations in the worst case. The load-state graphs depend only on the robot class and the object type, so there are at most $K(1 + J)$ distinct graphs for K classes and J object types, and the shortest distances

from every node cost $O(N_V(N_E + N_V) \log N_V)$ per graph with Dijkstra's algorithm, where N_V and N_E are the numbers of nodes and connections of the route graph (Appendix A). For fleets of 96 and 192 robots with four tasks per robot, the allocation took 0.78 s and 6.3 s, respectively; on testbed_C, with 40 places instead of 14, it took 1.10 to 1.27 times as long as on testbed_A.

These measurements assume that the robot classes, object types, and place graph are static. If a new type or traversability attribute is added, the corresponding template and the load-state distance cache must be reconstructed, which takes no longer than the preprocessing above.

Table 10. Per-instance computation time (testbed_A, 48-robot fleet).

Load	Template Projection (ms)	Allocation (ms)	Total (ms)
24	0.38	2.52	2.90
48	0.86	7.86	8.72
96	1.91	27.69	29.60
144	2.56	60.14	62.70
192	3.38	105.64	109.02

The values are averages over 30 different instances generated at each load level, with the template projection, including the expansion of the class-level results to the individual robots, and the allocation measured once for each instance. The runs for time measurement were performed separately from the statistical experiments.

5.3. Robustness Checks

This section tests whether the conclusions of Sections 5.1 and 5.2 hold when the layout, the operating conditions, or the allocation procedure changes. Each experiment changes one factor relative to the main experiments: the layout on testbed_C, for which w_p was recalibrated, and the robot speeds, the handling times, or the bidding rule on testbed_A with the same instances as in Section 5.1; the order within each queue and congestion at the loading and unloading places are additionally examined on both testbeds. Table 11 summarizes the comparisons for the layout and the operating conditions together with those of the main experiments.

Table 11. Makespan improvement of the proposed method across layouts and operating conditions.

Condition	vs. Dispatch-Aware	vs. Route-Only	vs. CF Tie-Break	Ties Between Classes (%)	Violations of Baseline (%)
Main experiments (testbed_A)	+6.7 to +20.1 (5/5)	+9.0 to +19.9 (5/5)	0.0 to +17.9 (4/5)	45.0–61.4	24.5–25.3
testbed_C	+7.1 to +17.0 (5/5)	+12.8 to +27.4 (5/5)	+12.2 to +26.8 (5/5)	0.0–0.5	26.2–27.5
Class-specific speeds	—	−0.1 to +19.3 (4/5)	−0.2 to +17.8 (4/5; 1/5 [†])	0.0	—
Object-specific handling times	—	+8.0 to +24.3 (5/5)	0.0 to +23.3 (4/5)	33.1–55.4	—

The values are the minimum and maximum of $\Delta\%$ over the loads 24–192, as defined in Table 7, on the full task set; a positive value indicates that the proposed method is shorter. The parentheses give the number of loads at which the difference was significant in favor of the proposed method after the Holm correction (normal approximation) within each family, and [†] gives the number in favor of the comparison configuration. At load 24, the proposed method and CF tie-break gave the same makespan in the main experiments and with object-specific handling times. dispatch-aware and baseline were not run in the last two conditions (—). Ties between classes: share of the selection steps of route-only in which the minimum of S was attained by robots of more than one class.

Larger Layout. On testbed_C, w_p was recalibrated in the same way as on testbed_A (Section 5.2) and set to 35; the value 20 selected on testbed_A would have given a mean

improvement of 14.7% against a maximum of 19.1% over the examined values. The five configurations of the main experiments, CF tie-break, and dispatch-aware + payload were then run with the same fleet, task generator, and protocol. As on testbed_A, baseline and payload-only produced violations for approximately one quarter of the tasks, and the configurations that plan the loaded segment on $G_r(o_t)$ produced none. baseline and dispatch-aware produced identical allocations. On the common completed-task set, which omits the work that baseline leaves undone (Section 4.2), the proposed method was 2.8 to 3.7% longer than baseline at loads 48–192, but on the full task set it was 7.1 to 17.0% shorter than dispatch-aware at all five load levels. The comparisons on testbed_C are therefore made on the full task set among the configurations without violations (Table 11). The proposed method was also shorter than route-only and CF tie-break at all five load levels and differed from dispatch-aware + payload by at most 0.5% without a significant difference. For dispatch-aware, the allocation underestimated the completion time of a robot by 2.9 to 11.3 s on average. CF tie-break was 0.6 to 1.7% shorter than route-only, and the difference was significant at three of the five load levels after the Holm correction. The share of downgradable tasks under route-only and the proposed method was 35.5% and 13.5% at load 24 and 25.2% and 18.2% at load 192, and the difference was significant at all five load levels after the Holm correction.

Unlike on testbed_A, route-only was longer than dispatch-aware at all five load levels on testbed_C, by 4.6 to 14.4%. Evaluating the loaded detours on $G_r(o_t)$ raised the expected completion times of the tasks that need them, so route-only allocated these tasks later, on average after 65% instead of 50% of the allocation steps at load 192. A similar delay on testbed_A (72% instead of 53%) left the makespans of the two configurations within 0.5% at loads 48–192 (Section 5.2), and why the delay lengthened the makespan on testbed_C was not identified. On testbed_C at load 192, the final task of the robot that finished last needed a detour in 99.4% of the instances under route-only and in 84.6% under dispatch-aware, and it was a task that only one class can carry in 80.9% and 61.8%, respectively; under the proposed method, these shares were 18.6% and 7.4%, which suggests that the gain there also arises mainly from the payload term.

Class-Specific Speeds. Instead of the common speed of 1.0 m/s, design speeds from 0.7 to 1.3 m/s were assigned so that the speed decreases as the footprint area increases (Table 2) and, at equal footprints, as the height increases: LIFT_S 1.3, convey 1.2, mani 1.1, LIFT_M 1.0, LIFT_L 0.9, F_lift 0.8, and H_load 0.7 m/s. In Equation (6), v is replaced by the speed v_r of the class of robot r , whereas v in the payload term of Equation (7) is set to a reference speed $v_{\text{ref}} = 1$ m/s so that w_p keeps its calibrated meaning. Class-specific speeds also reduce the ties in S between classes, which the payload term resolves in the main experiments (Section 5.2); in route-only, the share of selection steps with such ties fell to 0.0%. The proposed method was nevertheless significantly shorter than route-only and CF tie-break at loads 48–192, so its gain at these loads does not come from resolving ties between classes. At load 24, the gain vanished, and CF tie-break was significantly shorter (Table 11).

Object-Specific Handling Times. The handling time t_{load} of each loading and unloading in Equation (6) was set by object series to 3, 6, and 9 s for box, rack, and pallet. These are design values in the ratio 1:2:3, whose mean over the task mixture, 5.25 s, is close to the 5 s of the main experiments. The proposed method was again significantly shorter than route-only at all five load levels and than CF tie-break at loads 48–192 (Table 11).

Allocation Procedure. All configurations above share Algorithm 1, the append-only SSI auction with the MiniMax rule when $w_p = 0$ (Section 3.3). The semantic cost was also examined under the MiniSum rule of the same auction [48], in which each robot bids C_{route} , the increase in its total route distance, plus the payload term when $w_p > 0$.

Under the MiniSum rule on testbed_A, the payload term still shortened the makespan: the MiniSum variant of the proposed method was 5.1 to 15.0% shorter than that of route-only, and the difference was significant at all five load levels after the Holm correction. Because the MiniSum rule does not use completion times, both MiniMax configurations were 12.0 to 68.3% shorter than their MiniSum counterparts. To measure the loss from the order within each queue, every queue of the allocations was reordered optimally by dynamic programming over the visiting order, with the robot–task mapping fixed. This reduced the mean makespan of route-only, dispatch-aware, dispatch-aware + payload, and the proposed method by at most 0.52% on testbed_A and 4.76% on testbed_C at all five load levels.

Congestion-Aware Replay. Because the time model excludes interference between robots, the allocations and routes of the configurations without violations were also replayed in a discrete-event simulation that keeps the routes and the queue order and adds waiting at the loading and unloading places: each such place serves one robot at a time for the handling time, and robots that arrive while it is occupied wait in the preceding aisle in first-in-first-out order. A robot that only passes through such a place on testbed_C does not occupy it, and interactions in the aisles and shortcuts are not modeled. With unlimited place capacity, the replay reproduces the completion time of every robot in the time model.

Under this congestion, the proposed method remained shorter than dispatch-aware and route-only, by 4.2 to 9.5% and 5.6 to 8.1% on testbed_A and by 5.9 to 17.6% and 12.9 to 27.4% on testbed_C, respectively, and the differences were significant at all five load levels on both testbeds after the Holm correction. Relative to the time model, congestion lengthened the makespan by 23.0 to 49.5% on testbed_A and by 4.4 to 9.4% on testbed_C. On testbed_A, it lengthened the makespan of the proposed method more than that of route-only, so the gain over route-only was smaller than the 9.0 to 19.9% of the time model; on testbed_C, the gain was almost unchanged. The proposed method was also 8.4 to 15.5% shorter than uniform-rule on testbed_A and within 1% of dispatch-aware + payload at every load level on both testbeds. dispatch-aware + payload was significantly shorter, by 0.4 to 0.8%, at three of the ten testbed–load combinations, two of which, loads 144 and 192 on testbed_A, were not significant under the permutation test.

5.4. System-Level Validation

To check whether the results of the allocation and the route planning are maintained in the simulated execution, testbed_C was implemented in Gazebo Harmonic (gz-sim 8.9) [53] (Figure 5), and the travel was carried out with Nav2 1.3.10 [54] on ROS 2 Jazzy [55]. In particular, this section examines whether the executed trajectories follow the planned place sequences and remain within the load-state reachable place sets used for planning. Eight robots, two of LIFT_S and one of each other class, started from their home places, and 24 tasks generated by the task generator of Section 4.1 were allocated by the proposed method with $w_p = 35$, the value selected for testbed_C; the allocation took 1.9 ms (median of 20 repetitions), excluding the preprocessing of Section 5.2.

The queues were produced from the places, robots, and tasks of this run by the pipeline described in Section 4.2 and were dispatched to the eight robots. The robots departed one after another at intervals of 20 s and then traveled concurrently in the same environment. The task executor, which sends the queues to Nav2, has no traffic control between robots, and the local avoidance where robots meet is included in the measured trajectories. In the simulation, the object models themselves are not placed, and the clearances of the shortcuts are not enforced physically; the experiment therefore does not test whether the geometry blocks infeasible passages. Figure 6 overlays the executed

trajectories on the planned place sequences of LIFT_M and LIFT_L, which carried rack_350 from the same pickup place.

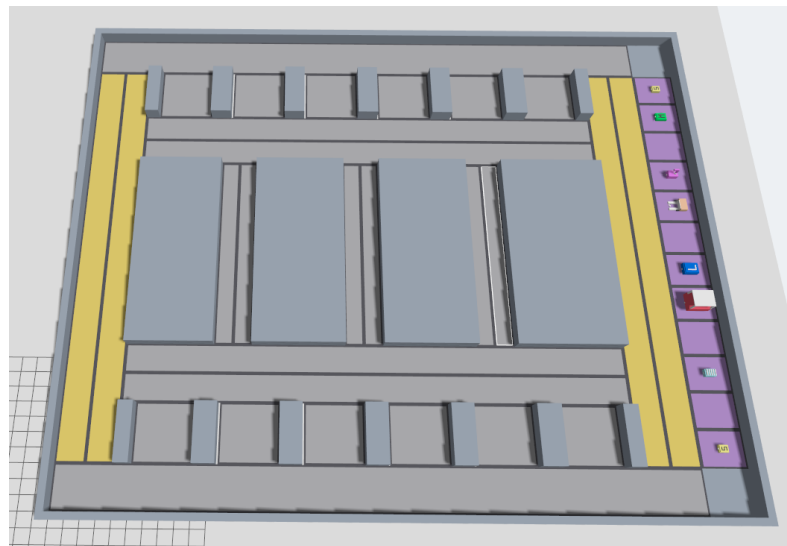


Figure 5. testbed_C in the Gazebo simulation environment.

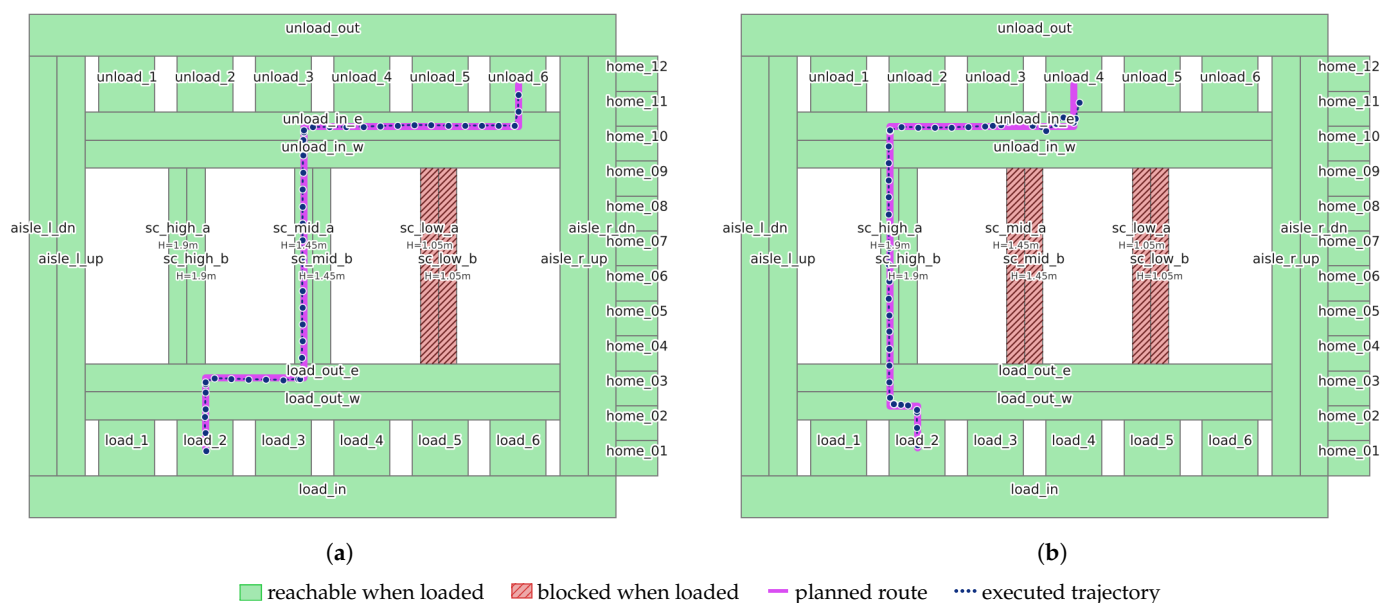


Figure 6. Planned routes and executed trajectories in Gazebo for the same object (rack_350). (a) LIFT_M (loaded height 1.4 m) through sc_mid (H 1.45 m). (b) LIFT_L (loaded height 1.6 m) through sc_high (H 1.9 m).

The planned and actual traversed place sequences agreed in all 48 segments of the 24 tasks. There was no entry into a place outside the reachable place set of that segment, and all segments reached their goal (Table 12). Nav2 executed four recovery behaviors, in two loaded segments, and no navigation goal was aborted. The planned makespan was 629.8 s, of which 50 s was handling time that the executor does not simulate, and the executed makespan, from the first departure to the completion of the last task, was 885.9 s, including the departure offsets. The robot with the longest planned completion time, which departed 100 s after the first, took 785.9 s from its departure against a planned travel time of 579.8 s, mainly because the robots traveled at 0.64 to 0.71 m/s on average instead of the 1.0 m/s of the time model; the robots finished in the order of their planned completion times.

Table 12. Planned and executed segments in the Gazebo simulation.

Segment	<i>n</i>	Reachable Places	Planned Distance (m)	Measured Distance (m)	Sequence Agreement	Entry into Blocked Places
Empty travel	24	38–40	67.58	65.51	24/24	0
Loaded, via shortcut	16	36–40	38.98	37.00	16/16	0
Loaded, via arterial aisle	8	34–40	43.62	42.03	8/8	0

n is the number of segments in each category, and the reachable places are given as the range over these segments. The distances are segment averages. The planned distance is the length of the planned route on the route graph of testbed_C (Section 4.1). The measured distance is the length of the trajectory through the recorded poses, so the two values are not on the same scale. The actually traversed place sequence was obtained by mapping the poses, recorded at most once per second, onto the place boundaries afterward.

Two further runs used the same allocation and settings. In one, all tasks were completed, every traversed place sequence again agreed with the plan, and the robots finished in the same order. In the other, four robots returning to the loading places blocked one another at the entrance of the one-way aisle through which every route from the unloading places to the loading places passes, and the run was stopped after 19 of the 24 tasks had been completed. In the 38 segments completed in this run, no place outside the reachable place set was entered; in one of them, the recorded poses crossed the boundary into the opposite lane of the same aisle by less than 0.1 m, so its traversed place sequence differed from the plan.

6. Discussion

In the proposed allocation procedure, the semantic feasibility information restricts the candidates and simultaneously forms two real-valued costs. The route term evaluates the distance of the loaded segment on the load-state graph, and the same graph is used for actual route generation and validation. Load dependence is not added to the cost as a penalty term but is reflected in the graph on which the distance is computed. In addition, the payload term prefers robots whose capacity is close to the required payload among the feasible candidates, and this discourages the queues of high-capacity robots from being occupied by light tasks. These costs are combined with the expected completion time into a single selection criterion, such that the payload preference enters the allocation together with the current queue workload. The two terms act together in this criterion.

Ablation shows that load-state route planning and validation and the payload term have distinct roles in different indicators. When route planning does not reflect the load state, routes inconsistent with the traversability conditions of the loaded state were generated for approximately one quarter of all tasks, whereas the configurations that plan and validate on the load-state graph had none, as expected by construction. The role of the route term differs from this. When only the graph of the allocation cost is switched while the route planning condition is fixed, the allocated robot more often has the shortest loaded-segment distance among the eligible robots. In the time model and without the payload term, this did not consistently shorten the makespan on testbed_A and lengthened it on testbed_C (Sections 5.2 and 5.3); with the payload term, switching the cost graph of the loaded segment changed the makespan little (Table 8 and Section 5.3). The route term instead evaluates each candidate on the graph on which its loaded route is planned and validated (Section 3.4) and therefore removes by construction the underestimation of completion times that arises when the loaded segment is evaluated on the empty-state graph but planned on the loaded graph (Sections 5.2 and 5.3). This matters whenever the completion times computed in the allocation are used beyond it, for example to time the dispatch. The shorter makespan of the proposed method compared with route-only, which

has no violations, is due to the payload term. This preference shifted the choice of feasible candidates toward robots closer to the required payload.

Because the selection criterion chooses an unallocated task and a robot together, the payload term also acts on the order in which pairs are selected. The difference in the makespan is interpreted as a combined effect of the loaded-segment distances, how the workload is spread across robots, and the deadhead chaining. However, the size of this efficiency gain was not constant. In the main task mixture of the main experiments, the gain was larger at medium and high loads, and in the heavy mixture, which increases the proportion of the pallet series, the improvement narrowed. This shows that the size of the capacity-fit effect depends on the workload and the task mixture, and suggests that the room for choice among the eligible robots is one mechanism behind this dependence.

There are several limitations on the scope within which these results can be interpreted. This study addressed an open-loop setting in which all tasks are allocated in a batch before execution in a static environment, and the completion time model did not include travel interference between robots or dynamic delays. Because the configurations generate different routes, the effect of interference could differ across them. The congestion-aware replay (Section 5.3) keeps the allocation and routes fixed and does not model the interactions in the aisles and shortcuts; where a narrow shortcut becomes a bottleneck, the gain could be smaller than in the replay or even reversed. Consistency at the execution level was confirmed in a Gazebo experiment with eight robots, which checked the planned place sequences but not the physical enforcement of clearances; because its task executor has no traffic control, robots can deadlock where routes merge, as in one of the three runs (Section 5.4). The allocation effect was evaluated with append-only sequential allocation; auctions that insert a task at the best position in a queue, which can also change the robot-task mapping, and optimal allocations were not included in the comparison. The other task mixtures, the MiniSum rule, class-specific speeds, and object-specific handling times were examined on testbed_A only, and the speed reduction in the loaded state, the fleet composition, and the distribution of weights within each series were not varied.

The payload weight w_p is a coefficient that converts the payload margin into travel distance, and its value was calibrated for each layout: 20 on testbed_A and 35 on testbed_C, where 20 would have given a smaller improvement (Section 5.3). For a new environment, fleet, or task set, w_p should be calibrated in the same way on instances separate from those used for evaluation.

Future work should extend the approach to closed-loop allocation, in which the load-state reachable place sets and graphs are updated according to observations made during execution, and the route term and queues are re-evaluated and reallocated accordingly. Such closed-loop operation can build on methods that update the assignment online as tasks arrive or conditions change [6,15] and on frameworks that couple centralized allocation with execution [21]. The effect of the semantic cost also needs to be verified in environments with interference between robots and under allocation procedures other than append-only sequential allocation.

7. Conclusions

For the task allocation of heterogeneous multi-robot systems, this study converted the feasibility information produced by semantic reasoning into real-valued costs of load-state travel distance and payload margin. A sequential allocation procedure that combines these costs with the expected completion time was presented. The route cost of the loaded segment and the actual route generation were performed on the same load-state graph, so that the travel conditions assumed by the allocation were matched to the conditions of the routes actually planned.

In the experiments, planning routes on the empty-state graph produced routes inconsistent with the loaded graph for approximately one quarter of all tasks, whereas planning and validating on the load-state graph produced none, as expected by construction. The proposed method shortened the makespan by 6.7 to 20.1% relative to the same procedure with load-invariant costs and no payload term, whose loaded routes were planned on the loaded graph before execution. This reduction came mainly from the payload term, whereas the load-state route cost kept the completion times computed in the allocation consistent with the planned routes; for the compared procedure, the allocation underestimated the makespan by 2.7 to 8.0%. The reduction was maintained on a larger warehouse layout with a different topology, for which w_p was recalibrated, and in a replay that adds congestion at the loading and unloading places. Relative to the same procedure without the payload term, the reduction depended on the task mixture and narrowed to single-digit percentages when the pallet series made up a larger share of the tasks. In ROS 2 and Gazebo with eight robots, the executed place sequences agreed with the load-state plans in the two runs that were completed, and a third run deadlocked at a merging aisle because the task executor has no traffic control.

This study targeted batch allocation before execution in a static environment, and multi-robot interference and dynamic delays were left outside the scope of the completion time model; the congestion-aware replay added only waiting at the loading and unloading places. Future work should extend this to closed-loop allocation that feeds observations made during execution back into the allocation and verify the effect of the semantic cost in environments with interference in the aisles and under allocation procedures other than append-only sequential allocation.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/app162010108/s1>. File S1: ontologies of the three testbeds in OWL format (classes, properties, and SWRL rules); File S2: place models of the three testbeds, including the route graph of testbed_C with its lane directions; File S3: robot class definitions and payload capacities by handling method; File S4: task type definitions and frequencies; File S5: generation rules and random seeds of the instances, including the tie-breaking seeds; File S6: allocation parameters and configuration files of the reported experiments; File S7: per-instance results of the main experiments on the three testbeds, with the tie-breaking repetitions of each instance averaged.

Author Contributions: Conceptualization, G.I. and T.K.; methodology, G.I., K.L., Y.A. and K.J.; software, G.I., Y.A., K.J. and G.K.; validation, G.I., K.L. and T.K.; formal analysis, G.I., K.L., Y.A. and K.J.; investigation, G.I., Y.A. and G.K.; writing—original draft preparation, G.I.; writing—review and editing, G.I. and T.K.; visualization, G.I.; supervision, T.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Ministry of Agriculture, Food and Rural Affairs (MAFRA) and the Korea Institute of Planning and Evaluation for Technology in Food, Agriculture and Forestry (IPET) (No. RS-2026-25525629, Smart Manufacturing Technology for Customized Foods).

Data Availability Statement: The original contributions presented in this study are included in the article and the Supplementary Materials. Further inquiries can be directed to the corresponding author.

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

Abbreviations

The following abbreviations are used in this manuscript:

AGV	Automated Guided Vehicle
CBBA	Consensus-Based Bundle Algorithm
CF	Capacity-Fit
MRTA	Multi-Robot Task Allocation
OWL	Web Ontology Language
ROS	Robot Operating System
SSI	Sequential Single-Item
SWRL	Semantic Web Rule Language
TOSM	Triplet Ontological Semantic Model
TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem

Appendix A. Pseudocode for Preprocessing and Validation

Algorithm A1 builds the load-state graphs and their distance tables once for a given set of robot classes, object types, and places; on testbed_C, the shortest paths are computed on the directed route graph (Section 4.1). For each task, the ReasonerOutput gives as $P_r(o_t)$ the places that admit the loaded cross-section and are connected to the pickup place. In the testbeds of this study, only the shortcuts restrict loaded passage and the remaining places are connected, so this set depends only on the class of r and the object type of o_t and is written $P_k(o)$ below. The eligible robot set $A(t)$ is applied when these class-level results are projected onto each instance (line 7 of Algorithm A1). In the route graph, a place is represented by its center and by further nodes: the midpoints of the boundaries that it shares with adjacent places on testbed_A and testbed_B, and route nodes along its travel paths on testbed_C, where the center is used only as the start or end of a route. A midpoint, like an end node of a shortcut lane on testbed_C, lies on a boundary; such a node belongs to every place that it touches, and a route that passes through it visits all of them. Algorithm A2 gives the pre-execution validation of Section 3.4.

Algorithm A1 Preprocessing of load-state graphs and distance tables

Input: environment graph $G = (V, E, \ell)$, robot classes $\mathcal{K}$, object types $\mathcal{J}$,
 reachable place sets $P_k(\text{empty})$ and $P_k(o)$ from the ReasonerOutput

Output: distance tables D_P for the reachable place sets P

- 1: **for each** class $k \in \mathcal{K}$ **do** (robots of the same class share the results)
- 2: $D_{P_k(\text{empty})} \leftarrow$ all-pairs shortest-path distances in $G[P_k(\text{empty})]$
- 3: **for each** object type $o \in \mathcal{J}$ **do**
- 4: $D_{P_k(o)} \leftarrow$ all-pairs shortest-path distances in $G[P_k(o)]$
- 5: **end for**
- 6: **end for**
- 7: **for each** instance: take $A(t)$ and $P_r(\cdot)$ of each robot r and task t
 from the class of r and the object type of o_t (template projection),
 and look up the distances in $D_{P_r(\cdot)}$

Algorithm A2 Pre-execution validation of planned loaded routes

Input: allocation $Q = \{Q_r\}_{r \in R}$ and loaded reachable place sets $P_r(o_t)$,
loaded route planning graph $H_{r,t}$ of the configuration:
 $G_r(o_t)$, $G_r(\text{empty})$, or G_r^U

Output: $V(r, t)$ for every allocated pair, completed-task set

```

1: for each robot  $r$  and each task  $t$  in  $Q_r$  do
2:    $\pi_{r,t} \leftarrow$  shortest path from  $p_t$  to  $d_t$  in  $H_{r,t}$ ; among shortest paths, the one whose
     sequence of place identifiers is lexicographically smallest
3:    $V(r, t) \leftarrow 1$  if no such path exists or  $\pi_{r,t}$  visits a place not in  $P_r(o_t)$ ,
     and  $V(r, t) \leftarrow 0$  otherwise
4: end for
5: return  $V$  and the completed-task set  $\{t : V(r, t) = 0\}$ 

```

Appendix B. Diminishing Marginal Gain of the Append-Only Score

The diminishing marginal gain property requires that the score $\bar{S} - S(r, t \mid Q_r)$ of Section 3.3 does not increase when another task is appended to Q_r , that is, that $S(r, t \mid Q_r)$ does not decrease. Because C_{payload} does not depend on the queue, appending another task t' to Q_r changes $S(r, t \mid Q_r)$ in Equation (7) by the handling time $2t_{\text{load}}$ of t' plus the travel time of the route from the end of Q_r through the pickup and drop places of t' to p_t , minus that of the shortest route from the end of Q_r to p_t . On testbed_A and testbed_B, the first route lies in $G_r(\text{empty})$, of which the loaded graph of t' is a subgraph, and is therefore no shorter than the second, so $S(r, t \mid Q_r)$ increases by at least $2t_{\text{load}}$. On testbed_C, a route cannot pass through the center of an intermediate place (Appendix A), so this argument does not apply directly. The property was instead verified for all 35,640 combinations of robot class and loaded graph, end of the queue (a home place or a drop place), pickup and drop places of t' , and a pickup place of t other than that of t' (if the two coincide, the property holds trivially): in each of them, the route through t' was at least 86.0 m longer than the shortest route from the end of Q_r to p_t .

References

- Choi, H.L.; Brunet, L.; How, J.P. Consensus-Based Decentralized Auctions for Robust Task Allocation. *IEEE Trans. Robot.* **2009**, *25*, 912–926. <https://doi.org/10.1109/TRO.2009.2022423>.
- Wang, Z.; Liu, C.; Gombolay, M. Heterogeneous Graph Attention Networks for Scalable Multi-Robot Scheduling with Temporospatial Constraints. *Auton. Robot.* **2022**, *46*, 249–268. <https://doi.org/10.1007/s10514-021-09997-2>.
- Nait Chabane, A.; Guenounou, O. An Enhanced Genetic Algorithm for Optimized Task Allocation and Planning in Heterogeneous Multi-Robot Systems. *Complex Intell. Syst.* **2025**, *11*, 435. <https://doi.org/10.1007/s40747-025-02062-w>.
- In, G.; Kwon, G.; An, Y.; Kuc, T. Semantic Feasibility Reasoning for Heterogeneous Multi-Robot Task Allocation. *Electronics* **2026**, *15*, 3562. <https://doi.org/10.3390/electronics15163562>.
- Rizk, Y.; Awad, M.; Tunstel, E.W. Cooperative Heterogeneous Multi-Robot Systems: A Survey. *ACM Comput. Surv.* **2019**, *52*, 1–31. <https://doi.org/10.1145/3303848>.
- Farinelli, A.; Iocchi, L.; Nardi, D. Distributed On-Line Dynamic Task Assignment for Multi-Robot Patrolling. *Auton. Robot.* **2017**, *41*, 1321–1345. <https://doi.org/10.1007/s10514-016-9579-8>.
- Gong, S.; Varlamov, O. CAHT: A Constraint-Aware Heterogeneous Transformer for Real-Time Multi-Robot Task Allocation in Warehouse Environments. *Algorithms* **2026**, *19*, 312. <https://doi.org/10.3390/a19040312>.
- Zhang, H.; Pan, Q.; Zhang, W.; Miao, Z.; Zhu, B. A Knowledge-Driven Collaborative Discrete Jaya Algorithm for Energy-Efficient Multi-Robot Task Allocation and Scheduling Problem. *Appl. Soft Comput.* **2026**, *201*, 115505. <https://doi.org/10.1016/j.asoc.2026.115505>.
- Notomista, G.; Mayya, S.; Emam, Y.; Kroninger, C.; Bohannon, A.; Hutchinson, S.; Egerstedt, M. A Resilient and Energy-Aware Task Allocation Framework for Heterogeneous Multirobot Systems. *IEEE Trans. Robot.* **2022**, *38*, 159–179. <https://doi.org/10.1109/TRO.2021.3102379>.
- Aziz, H.; Chan, H.; Cseh, Á.; Li, B.; Ramezani, F.; Wang, C. Multi-Robot Task Allocation—Complexity and Approximation. *arXiv* **2021**, arXiv:2103.12370. <https://doi.org/10.48550/arXiv.2103.12370>.

11. Arif, M.U.; Haider, S. A Flexible Framework for Diverse Multi-Robot Task Allocation Scenarios Including Multi-Tasking. *ACM Trans. Auton. Adapt. Syst.* **2022**, *16*, 3. <https://doi.org/10.1145/3502200>.
12. Bakhat, S.I.D.; Ullah, I.; Bai, X.; Zhang, W.; Li, J.; Ge, S.S. Advancing Multi-Robot Task Allocation: A Comprehensive Review of Emerging Trends and Challenges. *IEEE Internet Things J.* **2026**, *13*, 44021–44058. <https://doi.org/10.1109/JIOT.2026.3699696>.
13. Bai, X.; Li, B.; Liu, F.; Qiu, L.; Cao, G. Efficient Heuristic Algorithm for Precedence-Constrained Multi-Robot Task Assignment. In Proceedings of the 2024 43rd Chinese Control Conference (CCC), Kunming, China, 28–31 July 2024; pp. 4504–4509. <https://doi.org/10.23919/CCC63176.2024.10661715>.
14. Graham, R.L. Bounds on Multiprocessing Timing Anomalies. *SIAM J. Appl. Math.* **1969**, *17*, 416–429. <https://doi.org/10.1137/0117039>.
15. Wang, H.; Xue, S.; Zhang, H.; Tan, L.; Wang, C.; Fu, Y. Dynamic Multi-Robot Task Allocation for Human-in-the-Loop Space Exploration: Knowledge Graph-Guided CBBA with LLM-Assisted Fault Analysis. *Machines* **2026**, *14*, 265. <https://doi.org/10.3390/machines14030265>.
16. Leung, J.Y.T.; Li, C.L. Scheduling with Processing Set Restrictions: A Survey. *Int. J. Prod. Econ.* **2008**, *116*, 251–262. <https://doi.org/10.1016/j.ijpe.2008.09.003>.
17. Ou, J.; Leung, J.Y.T.; Li, C.L. Scheduling Parallel Machines with Inclusive Processing Set Restrictions. *Nav. Res. Logist.* **2008**, *55*, 328–338. <https://doi.org/10.1002/nav.20286>.
18. Velhal, S.; R, S.B.; Bharatheesha, M.; Sundaram, S. A Dynamic Heterogeneous Team-Based Non-Iterative Approach for Online Pick-up and Just-In-Time Delivery Problems. *arXiv* **2023**, arXiv:2304.07124. <https://doi.org/10.48550/arXiv.2304.07124>.
19. Zhao, Z.; Li, X.; Liu, S.; Zhou, M.; Yang, X. Multi-Mobile-Robot Transport and Production Integrated System Optimization. *IEEE Trans. Autom. Sci. Eng.* **2025**, *22*, 7480–7491. <https://doi.org/10.1109/TASE.2024.3421889>.
20. Hu, Y.; Liu, D.; Li, J.; Li, J.; Wu, T. Multi-Robot Task Allocation with Spatiotemporal Constraints via Edge-Enhanced Attention Networks. *Appl. Sci.* **2026**, *16*, 904. <https://doi.org/10.3390/app16020904>.
21. Varghese T., G.; Kochuvila, S.; Kumar, N.; Prasad, R.R.V. Hybrid Coordination Framework for Centralized Task Allocation and Execution in Heterogeneous Multi-Robot Systems. *IEEE Access* **2026**, *14*, 61573–61595. <https://doi.org/10.1109/ACCESS.2026.3680629>.
22. Shakeri, Z.; Benfriha, K.; Varmazyar, M.; Talhi, E.; Quenehen, A. Production Scheduling with Multi-Robot Task Allocation in a Real Industry 4.0 Setting. *Sci. Rep.* **2025**, *15*, 1795. <https://doi.org/10.1038/s41598-024-84240-3>.
23. Nguyen, T.P.; Nguyen, H.; Phan, D.M.; Ngo, H.Q.T. Dynamic Chain for Scheduling of the Multi-AGV Systems with Load-Aware Motion Profiling. *Sci. Rep.* **2026**, *16*, 6832. <https://doi.org/10.1038/s41598-026-37083-z>.
24. Yamauchi, T.; Miyashita, Y.; Sugawara, T. Efficient Path and Action Planning Method for Multi-Agent Pickup and Delivery Tasks under Environmental Constraints. *SN Comput. Sci.* **2023**, *4*, 83. <https://doi.org/10.1007/s42979-022-01475-5>.
25. Li, B.; Ma, H. Double-Deck Multi-Agent Pickup and Delivery: Multi-Robot Rearrangement in Large-Scale Warehouses. *IEEE Robot. Autom. Lett.* **2023**, *8*, 3701–3708. <https://doi.org/10.1109/LRA.2023.3272272>.
26. Chen, Z.; Alonso-Mora, J.; Bai, X.; Harabor, D.D.; Stuckey, P.J. Integrated Task Assignment and Path Planning for Capacitated Multi-Agent Pickup and Delivery. *IEEE Robot. Autom. Lett.* **2021**, *6*, 5816–5823. <https://doi.org/10.1109/LRA.2021.3074883>.
27. Li, Y.; Huang, H. Efficient Task Planning for Heterogeneous AGVs in Warehouses. *IEEE Trans. Intell. Transp. Syst.* **2024**, *25*, 10005–10019. <https://doi.org/10.1109/TITS.2024.3356514>.
28. Han, X.; Li, K.; Ram Kumar, P.N. A Branch-and-Price Algorithm for Task Allocation and Global Path Planning of Multiple AGVs in Intelligent Warehouses. *Transp. Res. Part E Logist. Transp. Rev.* **2026**, *206*, 104587. <https://doi.org/10.1016/j.tre.2025.104587>.
29. Galindo, C.; Fernández-Madrigal, J.A.; González, J.; Saffiotti, A. Robot Task Planning Using Semantic Maps. *Robot. Auton. Syst.* **2008**, *56*, 955–966. <https://doi.org/10.1016/j.robot.2008.08.007>.
30. Tenorth, M.; Beetz, M. KnowRob: A Knowledge Processing Infrastructure for Cognition-Enabled Robots. *Int. J. Robot. Res.* **2013**, *32*, 566–590. <https://doi.org/10.1177/0278364913481635>.
31. Zhang Sprenger, C.; Corrales Ramón, J.A.; Baier, N.U. ORPP—An Ontology for Skill-Based Robotic Process Planning in Agile Manufacturing. *Electronics* **2024**, *13*, 3666. <https://doi.org/10.3390/electronics13183666>.
32. Manzoor, S.; Rocha, Y.G.; Joo, S.H.; Bae, S.H.; Kim, E.J.; Joo, K.J.; Kuc, T.Y. Ontology-Based Knowledge Representation in Robotic Systems: A Survey Oriented toward Applications. *Appl. Sci.* **2021**, *11*, 4324. <https://doi.org/10.3390/app1104324>.
33. Kluge-Wilkes, A.; Gunaseelan, B.; Schmitt, R.H. Ontology-Based Task Allocation for Heterogeneous Resources in Line-Less Mobile Assembly Systems. In *Annals of Scientific Society for Assembly, Handling and Industrial Robotics 2022*; Schüppstuhl, T., Tracht, K., Fleischer, J., Eds.; Springer International Publishing: Cham, Switzerland, 2023; pp. 53–63. https://doi.org/10.1007/978-3-031-10071-0_5.
34. Joo, S.; Bae, S.; Choi, J.; Park, H.; Lee, S.; You, S.; Uhm, T.; Moon, J.; Kuc, T. A Flexible Semantic Ontological Model Framework and Its Application to Robotic Navigation in Large Dynamic Environments. *Electronics* **2022**, *11*, 2420. <https://doi.org/10.3390/electronics11152420>.

35. Sarkar, C.; Dey, S.; Agarwal, M. Semantic Knowledge Driven Utility Calculation towards Efficient Multi-Robot Task Allocation. In Proceedings of the 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), Munich, Germany, 20–24 August 2018; pp. 144–147. <https://doi.org/10.1109/COASE.2018.8560555>.
36. Li, W.; Shang, J.; Liu, G.; Liu, Z.; Tian, G. Multiagent Task Allocation for Dynamic Intelligent Space: Auction and Preemption with Ontology Knowledge Graph. *IET Cyber-Syst. Robot.* **2025**, *7*, e70013. <https://doi.org/10.1049/csy2.70013>.
37. Bae, S.; Joo, S.; Choi, J.; Pyo, J.; Park, H.; Kuc, T. Semantic Knowledge-Based Hierarchical Planning Approach for Multi-Robot Systems. *Electronics* **2023**, *12*, 2131. <https://doi.org/10.3390/electronics12092131>.
38. Choi, J.H.; Seo, D.S.; Bae, S.H.; An, Y.C.; Kim, E.J.; Pyo, J.; Kuc, T.Y. A Semantic Energy-Aware Ontological Framework for Adaptive Task Planning and Allocation in Intelligent Mobile Systems. *Electronics* **2025**, *14*, 3647. <https://doi.org/10.3390/electronics14183647>.
39. Zhao, J.; Vogel-Heuser, B.; Ao, J.; Wu, Y.; Zhang, L.; Hartl, F.; Hujo, D.; Bing, Z.; Wu, F.; Knoll, A.; et al. Ontology Based AI Planning and Scheduling for Robotic Assembly. In Proceedings of the 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Abu Dhabi, United Arab Emirates, 14–18 October 2024; pp. 9855–9862. <https://doi.org/10.1109/IROS58592.2024.10802295>.
40. Gerkey, B.P.; Mataric, M.J. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *Int. J. Robot. Res.* **2004**, *23*, 939–954. <https://doi.org/10.1177/0278364904045564>.
41. Korsah, G.A.; Stentz, A.; Dias, M.B. A Comprehensive Taxonomy for Multi-Robot Task Allocation. *Int. J. Robot. Res.* **2013**, *32*, 1495–1512. <https://doi.org/10.1177/0278364913496484>.
42. Allahverdi, A.; Ng, C.T.; Cheng, T.C.E.; Kovalyov, M.Y. A Survey of Scheduling Problems with Setup Times or Costs. *Eur. J. Oper. Res.* **2008**, *187*, 985–1032. <https://doi.org/10.1016/j.ejor.2006.06.060>.
43. Joo, S.H.; Manzoor, S.; Rocha, Y.G.; Bae, S.H.; Lee, K.H.; Kuc, T.Y.; Kim, M. Autonomous Navigation Framework for Intelligent Robots Based on a Semantic Environment Modeling. *Appl. Sci.* **2020**, *10*, 3219. <https://doi.org/10.3390/app10093219>.
44. Horrocks, I.; Patel-Schneider, P.F.; Boley, H.; Tabet, S.; Grosz, B.; Dean, M. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, 2004. Available online: <https://www.w3.org/submissions/SWRL/> (accessed on 29 September 2026).
45. Padilla-Cuevas, J.; Reyes-Ortiz, J.A.; Bravo, M. Ontology-Based Context Event Representation, Reasoning, and Enhancing in Academic Environments. *Future Internet* **2021**, *13*, 151. <https://doi.org/10.3390/fi13060151>.
46. Ibarra, O.H.; Kim, C.E. Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *J. ACM* **1977**, *24*, 280–289. <https://doi.org/10.1145/322003.322011>.
47. Braun, T.D.; Siegel, H.J.; Beck, N.; Böloni, L.L.; Maheswaran, M.; Reuther, A.I.; Robertson, J.P.; Theys, M.D.; Yao, B.; Hensgen, D.; et al. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *J. Parallel Distrib. Comput.* **2001**, *61*, 810–837. <https://doi.org/10.1006/jpdc.2000.1714>.
48. Lagoudakis, M.G.; Markakis, E.; Kempe, D.; Keskinocak, P.; Kleywegt, A.; Koenig, S.; Tovey, C.; Meyerson, A.; Jain, S. Auction-Based Multi-Robot Routing. In Proceedings of the Robotics: Science and Systems I, Cambridge, MA, USA, 8–11 June 2005. <https://doi.org/10.15607/RSS.2005.I.045>.
49. Efron, B.; Tibshirani, R.J. *An Introduction to the Bootstrap*; Chapman and Hall/CRC: New York, NY, USA, 1994. <https://doi.org/10.1201/9780429246593>.
50. Wilcoxon, F. Individual Comparisons by Ranking Methods. *Biom. Bull.* **1945**, *1*, 80–83. <https://doi.org/10.2307/3001968>.
51. Holm, S. A Simple Sequentially Rejective Multiple Test Procedure. *Scand. J. Stat.* **1979**, *6*, 65–70.
52. Kerby, D.S. The Simple Difference Formula: An Approach to Teaching Nonparametric Correlation. *Compr. Psychol.* **2014**, *3*, 11.IT.3.1. <https://doi.org/10.2466/11.IT.3.1>.
53. Koenig, N.; Howard, A. Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator. In Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sendai, Japan, 28 September–2 October 2004; Volume 3, pp. 2149–2154. <https://doi.org/10.1109/IROS.2004.1389727>.
54. Macenski, S.; Martin, F.; White, R.; Clavero, J.G. The Marathon 2: A Navigation System. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 24 October 2020–24 January 2021; pp. 2718–2725. <https://doi.org/10.1109/IROS45743.2020.9341207>.
55. Macenski, S.; Foote, T.; Gerkey, B.; Lalancette, C.; Woodall, W. Robot Operating System 2: Design, Architecture, and Uses in the Wild. *Sci. Robot.* **2022**, *7*, eabm6074. <https://doi.org/10.1126/scirobotics.abm6074>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.