Thesis Report

SADCHER: Scheduling using Attention-based Dynamic Coalitions of Heterogeneous Robots in Real-Time

Authored by

Jakob Bichler

Daily Supervisor:	Andreu Matoses Gimenez (PhD candidate)
Supervising Professor:	Javier Alonso-Mora (Associate Professor)
Faculty:	Mechanical Engineering, TU Delft
Study Programm:	MSc Robotics
Lab:	Autonomous Multi-Robots Laboratory

Abstract

We present Sadcher, a real-time task assignment framework for heterogeneous multi-robot teams that incorporates dynamic coalition formation and task precedence constraints. Sadcher is trained through Imitation Learning and combines graph attention and transformers to predict assignment rewards between robots and tasks. Based on the predicted rewards, a relaxed bipartite matching step generates high-quality schedules with feasibility guarantees. We explicitly model robot and task positions, task durations, and robots' remaining processing times, enabling advanced temporal and spatial reasoning and generalization to environments with different spatiotemporal distributions compared to training. Trained on optimally solved small-scale instances, our method can scale to larger task sets and team sizes. Sadcher outperforms other learning-based and heuristic baselines on randomized, unseen problems for small and medium-sized teams with computation times suitable for real-time operation. We also explore sampling-based variants and evaluate scalability across robot and task counts.

To address performance limitations identified for large-scale problem instances, we experiment with using Reinforcement Learning (RL) to fine-tune the imitation-learned model. Both discrete and continuous RL formulations are explored, leveraging Proximal Policy Optimization (PPO). This allows for training on larger problem instances, for which optimal solutions for IL are infeasible to obtain. Our RL experiments provide insights into the comparative advantages and trade-offs of IL and RL methods.

In addition, we release our dataset of 250,000 optimal schedules to facilitate future research. We include a detailed description of the instance generation and the Mixed-Integer Linear Programming (MILP) formulation used to solve them optimally. Furthermore, this thesis contributes a lightweight simulation environment with visualization tools for benchmarking different task assignment algorithms.

Contents

1 Introduction

Preliminaries

The field of Scheduling and Multi-Robot Task Assignment includes many specialized approaches, and terms are often used differently depending on the context and paper, which might lead to confusion. This section defines the key terms used throughout the thesis to provide a consistent foundation.

Multi-Robot Task Assignment

Multi-robot task assignment (*MRTA*) aims to assign tasks to robots while meeting constraints to minimize an application-specific cost function [1]. The result is a (near-) optimal mapping from robots to tasks with corresponding starting times [2], called a schedule. Robots are assigned to tasks based on their required skills, with tasks varying in type and requirements. The value of completing each task can differ based on its finish time, the assigned robot, or other factors. Furthermore, task constraints influence the optimal schedule, since a specific order for task completion may be necessary [3].

Heterogeneous Robots

A team of robots is considered heterogeneous if the members offer different capabilities [4]. According to [5] robots can either differ in physical or behavioral aspects. Physical heterogeneity refers to differences in hardware or physical constraints. For example, a team consisting of a fast drone equipped with a camera and a slow mobile manipulator that can lift heavier objects. Behavioral heterogeneity occurs when the software that produces the behavioral model differs. Some robots might try to maximize a different objective function than others. For this work, a heterogeneous robot team is defined as physically different robots that work together to optimize for a shared goal.

Dynamic Coalition Formation

In MRTA robots might need to form coalitions (also called sub-teams) to execute a task when no single robot can provide all required capabilities [6]. Coalitions can either be formed statically or dynamically. Static formations are grouped once and remain the same over the complete task assignment horizon. Dynamic coalitions change their formation from task to task to ensure efficiency [7]. The requirement to form coalitions is closely related to tackling multi-robot (*MR*) tasks in the iTax taxonomy [8].

Motivation

Autonomous multi-robot systems (MRS) are designed for complex, real-world settings, including earthquake disaster response scenarios [9], autonomous construction [10], production assembly processes [11], or search and rescue missions [12]. Interest in MRS and multi-robot task assignment (MRTA) has grown rapidly in recent years [13]. Efficient MRTA algorithms are essential to optimize resource usage and minimize operational time [14]. MRS improve performance and enhance system robustness as a multi-robot team is more resilient against individual robot failures and performance bottlenecks [15, 16]. Using sub-teams of robots, i.e., dynamic coalition formation, enables teams to tackle complex tasks that would otherwise be infeasible for a single robot [16–18]. In practice, relying on a team of homogeneous, interchangeable robots where each robot possesses all skills can become impractical

if task requirements are numerous or highly diverse, involving different sensors and actuators [19]. Instead, using (redundant) heterogeneous robots brings practical and economic advantages. It allows leveraging existing specialized robots [20] and deploying simpler robots that are more cost-effective to implement and maintain [16] and are more robust to failures [14]. MRS operate in dynamic environments where sudden changes, new tasks, unexpected task requirements, robot malfunctions, or moving obstacles can occur [13]. Hence, the ability to replan adaptively and in real-time is essential. Modeling precedence constraints, which impose a logical temporal sequence among tasks, further enhances applicability to real-world scenarios [21] where some tasks depend on the completion of prior tasks. E.g., in autonomous construction, gathering materials must precede assembly [10].

Contributions

Motivated by these challenges, this thesis proposes Sadcher, a framework for real-time scheduling of heterogeneous multi-robot teams with dynamic coalition formation and precedence constraints, based on Imitation Learning (IL). Additionally, fine-tuning with Reinforcement Learning (RL) is explored. The main contributions are:

- A learning-based model, combining graph attention networks and transformers, inspired by [22], but extended by explicitly modeling robot and task positions, task durations, and robots' remaining processing times. This enables advanced spatiotemporal reasoning and generalization.
- A dataset containing 250,000 optimally solved small-scale problem instances (8 tasks, 3 robots, 3 skills, 1-6 precedence constraints), including detailed reporting on the dataset generation. While primarily intended for training IL models, it also enables benchmarking against optimal solutions.
- A lightweight simulation environment to test different scheduling algorithms on varying problem instances, including tools for visualization and benchmarking.
- An extensive evaluation of two Sadcher variants (deterministic and sampled outputs) against multiple baselines across problem scales.
- Quantifying the model's emergent understanding of task dependencies via a predecessor assignment ratio, showing anticipatory behavior in scheduling.
- An exploration of different RL formulations - with continuous and discrete action spaces - and experiments to research fine-tuning the IL model. We also provide a detailed discussion of the (dis-)advantages of RL and IL.

2

Related Work

This chapter provides a brief overview of related work. For a more detailed analysis, refer to the comprehensive literature review conducted at the start of this thesis project.

Referencing the taxonomy introduced in [23] and extended in [8], MRTA problems can be categorized on 4 axes: (1) single- or multi-task robots (ST/MT), (2) single- or multi-robot tasks (SR/MR), (3) instantaneous or time-extended assignment (IA/TA) where TA incorporates information about future tasks and scheduling decisions, (4) interdependence of agent-task utilities, i.e. with cross-dependencies (XD), an agent's utility for a task depends on the tasks assigned to other agents. This work addresses ST-MR-TA-XD settings

Conventional Methods

Mixed Integer Linear Programming (MILP) offers exact solutions for complex ST-MR-TA-XD problems [24], though its exponential runtime hinders real-time use. The MILP-based CTAS framework [19] explicitly models risk in agent capability for task decomposition and scheduling. Simpler heterogeneous ST-SR scenarios can be addressed with the Tercio algorithm [11], which uses an MILP-based task allocator and a polynomial runtime task scheduler. Auction algorithms like [12, 25] treat tasks as non-atomic – tasks execution can be incremental, not requiring coalition formation. [26] uses auctions to solve heterogeneous ST-MR problems with atomic tasks. Genetic Algorithms offer anytime solutions that balance exploration and exploitation: [27] tackles heterogeneous ST-SR, [28] focusses on coalition formation of homogeneous robots, while [20] can handle both heterogeneity and coalition formation. Other optimization metaheuristics applied to the heterogeneous ST-MR case include Ant Colony Optimization [18] and Particle Swarm Optimization [29]. Greedy formulations, such as [14], employ construction and improvement heuristics to balance runtime and performance.

Learning-based Methods

Deep learning methods promise fast solution generation and good scalability, by offloading most of the computation to the training phase [30]. Reinforcement Learning (RL) does not require a training dataset, but might spend a lot of time on infeasible solutions [31]. RL is used to solve ST-SR problems with mildly heterogeneous robots – robots differ in efficiency but can all perform any task – in [32] and [33]. Other RL methods solve ST-MR problems with dynamic coalition formation, but only for homogeneous robots [7, 34]. Recently, RL has been used to tackle heterogeneous ST-MR problems with dynamic coalition formation in [35]. The authors mitigate some of the problems RL faces through a flash-forward mechanism which allows for decision reordering to avoid deadlocks during training.

Instead of RL, other methods use Imitation Learning (IL) from optimal solutions during training, which requires a (computationally expensive) expert dataset, but benefits from stable and efficient training. [31] presents an IL method for mildly heterogeneous robots without coalition formation (ST-SR). Both [36] and [22] address heterogeneous ST-MR problems with a network predicting task assignment rewards and a bipartite matching algorithm yielding task assignments based on these rewards. In [36],

coalition formation is only considered if a task fails to be completed by a single robot . There are cases in which a more optimal schedule could be obtained by considering coalition formation for all tasks, e.g., two robots are faster at completing the task than one. [22] improves upon this by always considering coalition formation. Furthermore, they introduce voluntary waiting, which increases performance through enabling better future coalition formation by delaying task assignments. However, [22] omits task/robot locations and durations in the input to their network. This implicitly assumes task durations and travel times to be negligible or to match the distribution of the training data.

In this thesis, we extend previous IL methods by explicitly modeling robot/task positions, task durations, and robots' remaining time to complete the current task. This enables advanced spatiotemporal reasoning, e.g., synchronizing robot arrivals and anticipating task readiness and robot availability. Additionally, it supports generalization to environments with unseen spatiotemporal distributions.

3

Problem Statement

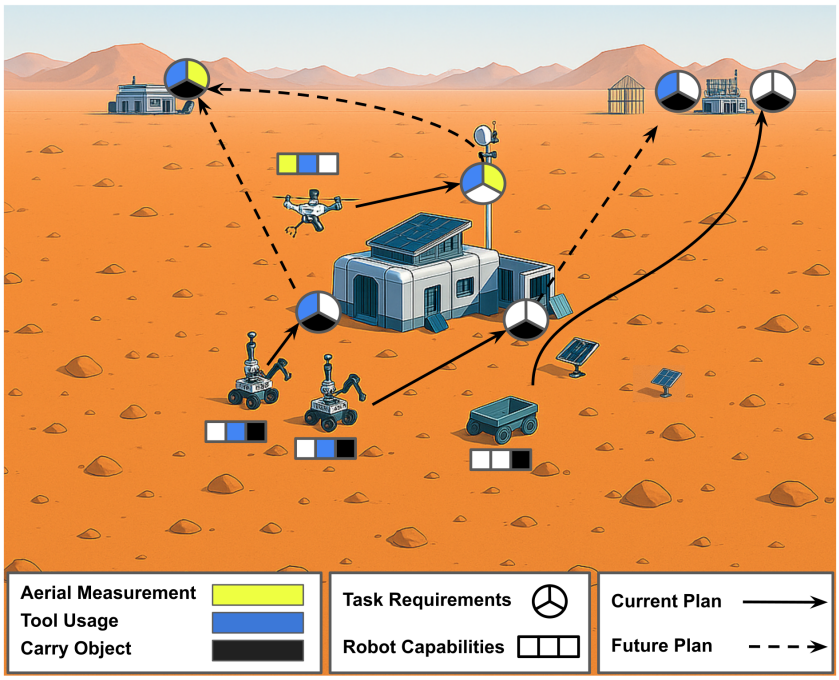

Fig. 3.1. Illustrative use case of autonomous construction on Mars. Circles represent tasks, color indicates required skills. Robot skills are shown as colored squares. Since no robot possesses all three skills, tasks requiring multiple skills (e.g., search for material in top left) must be executed by a synchronized coalition of robots.

Notation. Matrices are boldface uppercase (e.g. $\mathbf{M} \in \mathbb{R}^{\mathrm{n}\times\mathrm{m}}$), vectors are boldface lowercase (e.g. $\mathbf{v} \in \mathbb{R}^{\mathrm{d}}$), and scalars are lowercase (e.g. s).

We model a system of N heterogeneous robots, M tasks, and a set of robot skills $\mathcal{S}$. Each robot is capable of performing a subset of skills from $\mathcal{S}$; each task requires a subset of skills from $\mathcal{S}$ to be performed at its location for the given task duration.

The N heterogeneous robots with $S_i \subseteq \mathcal{S}$ distinct skills, are modeled as an undirected graph $\mathcal{G}^r = (\mathcal{R}, \mathbf{C})$, where each vertex in $\mathcal{R} = \{\mathbf{r}_i\}^N$ is a robot. Robot states $\mathbf{r}_i = [\mathbf{p_i^r}, t_i^r, a_i^r, \mathbf{c_i^r}]$ include position $\mathbf{p_i^r} \in \mathbb{R}^2$, remaining duration at the current task t_i^r, the robot's availability $a_i^r \in \{0,1\}$, and the binary capability vector over the global skill set $\mathbf{c_i^r} \in \{0,1\}^{|\mathcal{S}|}$. $\mathbf{C} \in \{0,1\}^{N\times N}$ represents the network connection among the robots. For simplicity, we assume a fully connected graph, so $C_{i,j} = 1\ \forall i,j$. However, our model is designed to accept any connected graph structure as input.

The M tasks and their respective precedence constraints are represented as a directed acyclic graph $\mathcal{G}^t = (\mathcal{T}, \mathbf{P})$. Each task is a vertex in $\mathcal{T} = \{\mathbf{t}_j\}^M$, and is described by $\mathbf{t}_j = \left[\mathbf{p}_\mathbf{j}^\mathbf{t}, t_j^t, \mathbf{r}_\mathbf{j}^\mathbf{t}, s_j^t\right]$, with position $\mathbf{p}_\mathbf{i}^\mathbf{r} \in \mathbb{R}^2$, expected duration t_j^t, required skills $\mathbf{r}_\mathbf{j}^\mathbf{t} \in \{0,1\}^{|\mathcal{S}|}$ and status $\mathbf{s}_\mathbf{j}^\mathbf{t} \in \{0,1\}^3$. The status indicates whether tasks are ready, assigned, or incomplete, e.g., $\mathbf{s}_j^t = [1,0,1]$ represents a task that is ready to be scheduled, currently not assigned, and incomplete. Precedence constraints are encoded in the edges $\mathbf{P}^{M\times M}$, where $P_{i,j} = 1$ means the i-th task is a predecessor of the j-th task. The j-th task is only ready to be scheduled if all its preceding tasks have been completed. A task can only commence when all required skills are covered by the dynamically formed coalition of robots assigned to it. This can be denoted as $\mathbf{c_C} \succeq \mathbf{r_j^t}$ where $\mathbf{c_C}$ is the element-wise sum of robot capabilities $\mathbf{c_i^r}$ of assigned robots and $\succeq$ is the element-wise greater-or-equal operator. The tasks require tightly coupled coalitions [28] - all robots have to be present at the task location for the entire execution duration. Furthermore, we introduce an idle task t_{M+1} that robots can choose to increase overall performance by delaying assignments until a better coalition can be formed. The idle task is always ready to be assigned.

Robots start at location $\mathbf{p}_i^{\text{start}}$ and end at $\mathbf{p}_i^{\text{end}}$. The cost function aims to minimize the makespan, defined as the latest arrival time of any robot at its end location after completing all its assigned tasks:

$$\min \max_{i\in\{1,\dots,N\}} \left(t_i^{\text{finish}} + \tau\left(\mathbf{p}_i^{\text{finish}}, \mathbf{p}_i^{\text{end}}\right)\right) \tag{3.1}$$

where t_i^{finish} is the time robot i finishes its final task, which is computed as the sum of its execution times, idling times, and travel times. $\tau\left(\mathbf{p}_i^{\text{finish}}, \mathbf{p}_i^{\text{end}}\right)$ is the travel time from the location of the last finished task $\mathbf{p}_i^{\text{finish}}$ to the end location $\mathbf{p}_i^{\text{end}}$. Travel times can be computed using simple Euclidean distance or more advanced path planning algorithms that account for obstacles.

There are several assumptions to be made to scope the problem:

- A1: There exists a perception/task decomposition system, that due to previously learned affordances [37] can generate tasks with according:
  - (a) Locations
  - (b) Estimated task durations (per robot)
  - (c) Required skills
  - (d) Precedence constraints relating to other tasks
- A2: The robot team composition is known, including:
  - (a) Number of robots
  - (b) Position of each robot
  - (c) Skills of each robot
  - (d) Travel speeds
- A3: The robots are equipped with a perception system, capable of:
  - (a) Locating obstacles for collision avoidance
  - (b) Assessing whether a task was successfully completed after attempted execution
- A4: The robots are equipped with a low-level control/motion planning system, capable of:
  - (a) Generating feasible/collision-free trajectories for task execution
  - (b) Locally avoiding dynamic obstacles, e.g. other robots, customers
  - (c) Execute the generated trajectories, both for traveling and task execution
- A5: All robots are connected through a stable communication network so the schedules generated by a centralized planner can be sent and executed

4

Dataset

4.1. Problem Instance Generation

To generate each problem instance and the corresponding solution, we:

1. Draw the robot skill matrix $Q \sim \text{Bernoulli}(0.5)^{n \times \ell}$ until every robot has ≥ 1 skill and every skill is present in at least one robot.
2. Draw the task requirement matrix $R \sim \text{Bernoulli}(0.5)^{m \times \ell}$ until every task requires ≥ 1 skill; prepend/append all-zero rows for the start and end locations to obtain $R \in \{0,1\}^{(m+2) \times \ell}$.
3. Draw task execution times $T^{\text{e}} \sim \mathcal{U}\{50, 100\}^m$ and pad with zeros for start and end task.
4. Place each task uniformly at random on a 100×100 grid
5. Set T^{t}_{jk} to the Euclidean distance between tasks j and k (continuous, symmetric, deterministic).
6. Sample up to n_{prec} random, acyclic precedence constraints from the internal task indices $1, \ldots, m$.
7. Solve the drawn problem instance with the MILP formulation below in Section 4.2.

To generate a dataset on the order of 10^5 solved instances within feasible computation time (around 1 week), we limit the problem size to 8 tasks, 3 robots, and 3 unique skills. The dataset was generated using the available resources of the DelftBlue High Performance Compute Cluster[1] using a dedicated CPU node equipped with 48 cores (2× Intel Xeon E5-6248R, 24C, 3.0 GHz), leveraging full multi-threading capabilities. We generate 250,000 problem instances and solutions, with a varying number of precedence constraints. To ensure full open-source reproducibility, the model is built in `PuLP`[2] and solved with the `CBC`[3] MILP solver.

[1] https://www.tudelft.nl/dhpc/system
[2] https://github.com/coin-or/pulp
[3] https://github.com/coin-or/Cbc

4.2. MILP formulation

For optimal ”ground-truth” data generation we use the following formulation. Compared to [24], we add constraint C16 to model explicit temporal precedence constraints between pairs of tasks. Furthermore, we omit the stochastic travel times, modeled as the Gaussian delay buffer T^{s}, and the associated chance-constrained derivations (Equations (17)–(19) in the paper). In our formulation, travel times are assumed to be fixed and known, since our framework handles deviations via real-time re-planning and does not benefit from conservative safety margins.

Notation

$\mathcal{R} = \{0, \dots, n-1\}$	index set of robots
$\mathcal{T} = \{0, \dots, m+1\}$	index set of tasks; 0: start depot, $m+1$: end depot
$\mathcal{S} = \{0, \dots, \ell-1\}$	index set of skills
$Q_{is} \in \{0,1\}$	1 iff robot i possesses skill s
$R_{ks} \in \{0,1\}$	1 iff task k requires skill s
$T^{\mathrm{e}}_k \in \mathbb{R}_{\geq 0}$	deterministic execution time of task k
$T^{\mathrm{t}}_{jk} \in \mathbb{R}_{\geq 0}$	deterministic travel time from task j to k (identical for all robots)
$\mathcal{P} \subseteq \{(j,k) \mid 1 \leq j < k \leq m\}$	given pairwise precedence constraints
$M = \sum_k T^{\mathrm{e}}_k + \sum_{j=0}^{m} \max_k T^{\mathrm{t}}_{jk}$	a valid “big-M” upper bound on any arrival time

Decision variables

$X_{ijk} \in \{0,1\}$	1 iff robot i performs task k *immediately after* task j
$Y_{ik} \in \mathbb{R}_{\geq 0}$	arrival time of robot i at task k
$Y^{\max}_k \in \mathbb{R}_{\geq 0}$	*start* time of task k = latest robot arrival
$Z_{ks} \in \mathbb{Z}_{\geq 0}$	# robots at task k that offer skill s
$Z^{\mathrm{b}}_{ks} \in \{0,1\}$	1 iff skill s is *excessive* (superfluous) at task k
$T^{\max} \in \mathbb{R}_{\geq 0}$	makespan (objective)

Objective

$$\min T^{\max} \tag{4.1}$$

where:

$$T^{\max} \;\geq\; Y_{i,m+1} + T^{\mathrm{e}}_{m+1} \qquad \forall i \in \mathcal{R}. \tag{C0}$$

Route-validity Constraints

- Start from the start location 0

$$\sum_{k=1}^{m+1} X_{i0k} = 1 \qquad \forall i \in \mathcal{R}. \tag{C1}$$

- End at exit location $m+1$

$$\sum_{j=0}^{m} X_{ij,m+1} = 1 \qquad \forall i \in \mathcal{R}. \tag{C2}$$

- Start location is exit only

$$\sum_{j=1}^{m+1} X_{ij0} = 0 \qquad \forall i \in \mathcal{R}. \tag{C3}$$

- End location is entry only

$$\sum_{k=0}^{m} X_{i,m+1,k} = 0 \qquad \forall i \in \mathcal{R}. \tag{C4}$$

- Enter each task k at most once

$$\sum_{j=0}^{m} X_{ijk} \leq 1 \qquad \forall i \in \mathcal{R},\ k = 1, \ldots, m. \tag{C5}$$

- Leave each task j at most once

$$\sum_{k=1}^{m+1} X_{ijk} \leq 1 \qquad \forall i \in \mathcal{R},\ j = 1, \ldots, m. \tag{C6}$$

- Flow conservation at task j

$$\sum_{k=0}^{m} X_{ikj} = \sum_{k=1}^{m+1} X_{ijk} \qquad \forall i \in \mathcal{R},\ j = 1, \ldots, m. \tag{C7}$$

- No self-loops

$$X_{ijj} = 0 \qquad \forall i \in \mathcal{R},\ j \in \mathcal{T}. \tag{C8}$$

Skill-feasibility Constraints

- Eligibility of a robot for a task (providing at least one skill)

$$\sum_{j=0}^{m} X_{ijk} \ \leq \ \sum_{s \in \mathcal{S}} Q_{is} R_{ks} \qquad \forall i,\ k \in \{1, \ldots, m\}. \tag{C9}$$

- Computation of delivered skills

$$Z_{ks} \ = \ \sum_{i \in \mathcal{R}} \sum_{j=0}^{m} X_{ijk}\, Q_{is} \qquad \forall k \in \{1, \ldots, m\},\ s \in \mathcal{S}. \tag{C10}$$

- Skill coverage

$$Z_{ks} \ \geq \ R_{ks} \qquad \forall k \in \{1, \ldots, m\},\ s \in \mathcal{S}. \tag{C11}$$

- Identifying excess skills:
  Binary Z^{b}_{ks} is 1, if skill s is provided by more robots than required

$$Z_{ks} - R_{ks} - n\, Z^{\mathrm{b}}_{ks} \leq 0 \tag{C12a}$$

$$Z_{ks} - R_{ks} - 1 + n\big(1 - Z^{\mathrm{b}}_{ks}\big) \geq 0 \tag{C12b}$$

- Rejecting robots that only provide redundant skills

$$\sum_{s \in \mathcal{S}} Q_{is} R_{ks} (1 - Z^{\mathrm{b}}_{ks}) \ \geq \ \sum_{j=0}^{m} X_{ijk} \qquad \forall i,\ k \in \{1, \ldots, m\}. \tag{C13}$$

Temporal Constraints

- Task start time

$$Y^{\max}_{k} \ \geq \ Y_{ik} \qquad \forall k \geq 1,\ i. \tag{C14}$$

- Temporal Coherence (for assigned robots: arrival time of robot i at task k is sum of arrival time at previous task j, execution time of j and travel time between j and k

$$\begin{aligned} Y_{ik} &\geq Y^{\max}_{j} + T^{\mathrm{e}}_{j} + T^{\mathrm{t}}_{jk} - M\big(1 - X_{ijk}\big) \\ Y_{ik} &\leq Y^{\max}_{j} + T^{\mathrm{e}}_{j} + T^{\mathrm{t}}_{jk} + M\big(1 - X_{ijk}\big) \end{aligned} \qquad \forall i,\ j \leq m,\ k \geq 1. \tag{C15}$$

Precedence Constraints

- Successor k can only start after completion of predecessor j

$$Y^{\max}_{k} \ \geq \ Y^{\max}_{j} + T^{\mathrm{e}}_{j}. \quad \forall (j,k) \in \mathcal{P} \tag{C16}$$

5 Method

We design the Sadcher framework, consisting of a neural network based on attention mechanisms to predict assignment rewards for robots to tasks that is agnostic to changes in the size of the input graphs, i.e. can handle arbitrary numbers of robots and tasks. A relaxed bipartite matching algorithm extracts task assignments based on the predicted reward. During runtime, the method asynchronously recomputes assignments at decision steps, i.e., when robots finish tasks or new tasks are announced.

5.1. Network Architecture

The high-level network structure is depicted in Fig. 5.1 and is similar to [22], but extended with a distance multilayer perceptron (MLP) that informs the network about relative distances between all robots and tasks and separate heads for predicting rewards for "normal" tasks and idle action. Furthermore, we feed richer features to the network by explicitly modeling robot/task positions, task durations, and robots' remaining time to complete the current task. This enables advanced spatiotemporal reasoning, e.g., synchronizing robot arrivals and anticipating task readiness and robot availability. Additionally, it supports generalization to environments with unseen spatiotemporal distributions.

The key components of the network are graph attention encoders (GAT) [38], transformer blocks [39], and reward MLPs that project latent embeddings into a reward matrix.

5.1.1. Graph Attention (GAT) Encoder Blocks

After mapping robot features $\mathbf{r}_i$ and task features $\mathbf{t}_j$ into d-dimensional embeddings, the embedded robot and task features are processed by separate GATs to capture local information-rich latent representations of the input graphs. GATs process a set of node features, incorporating information from neighboring nodes based on an adjacency matrix. While the robot features are processed as a fully connected graph, assuming all-to-all attention, the task GAT leverages the encoded precedence constraints in the adjacency matrix to understand the temporal task logic. A single head of the GAT computes attention weights α_{ij} between node i and its neighbors j (including itself), based on a projected feature vector $\mathbf{h}' = \mathbf{h}\mathbf{W}^h$:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(a([\mathbf{h}'_i || \mathbf{h}'_j])))}{\sum_{k \in \mathcal{N} i \cup i} \exp(\text{LeakyReLU}(a([\mathbf{h}'_i || \mathbf{h}'_k])))} \tag{5.1}$$

where a is a learnable linear transformation, $||$ denotes concatenation and $\mathcal{N}_i$ is the set of neighbors of node i. A Leaky ReLU [40] in combination with a softmax function over the neighbors of node i yields the final α_{ij}. The resulting α_{ij} represent the relative importance of node j to node i, enabling context-aware feature propagation. Spatiotemporally related tasks or robots with complementary skills will attend more strongly to each other.

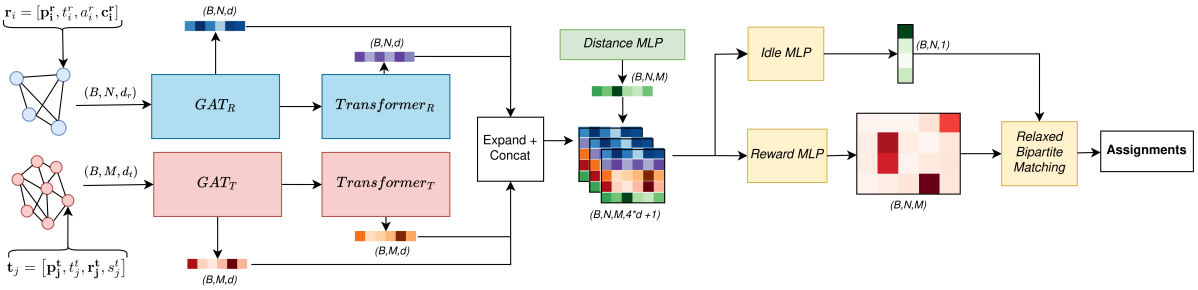

Fig. 5.1. Sadcher architecture overview. Robot and task graphs are processed by graph attention and transformer encoders and concatenated with distance features. The reward matrix is estimated by the Idle and Reward MLPs and final task assignments are extracted using relaxed bipartite matching. B: batch size, N: number of robots, M: number of tasks, d_r: robot input dimension, d_t: task input dimension, d: latent dimension.

The output $\mathbf{h}_i^{GAT}$ for a single head at node i is a sum of a self-loop contribution and the transformed neighbor contributions:

$$\mathbf{h}_i^{GAT} = \alpha_{ii}\mathbf{h}'_i + \text{LeakyReLU}\left(\sum_{j \in \mathcal{N}_i, j \neq i} \alpha_{ij}\mathbf{h}'_j\right) \tag{5.2}$$

In the GAT encoder blocks, we apply multi-head GAT, concatenating the outputs of Z_{GAT} independent heads and applying residual connections and layer normalization. The GAT encoder consist of L_{GAT} such layers and outputs $\mathbf{h}^{\text{GAT\_R}}$ for robots and $\mathbf{h}^{\text{GAT\_T}}$ for tasks respectively.

5.1.2. Transformer Encoder Blocks

Following the GAT encoders, the representations $\mathbf{h}^{\text{GAT\_R}}$ and $\mathbf{h}^{\text{GAT\_T}}$ are processed by independent transformer encoders, to build the global context of robots and tasks. Each transformer blocks applies multi-head self-attention, followed by layer normalization, MLP, and another layer normalization, with residual connections [39]. Multi-head self-attention transforms input queries, keys, and values $(\mathbf{Q}, \mathbf{K}, \mathbf{V})$ derived from the input representation $\mathbf{h}$ via linear projections $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V$:

$$\mathbf{Q} = \mathbf{W}^Q\mathbf{h}, \qquad \mathbf{K} = \mathbf{W}^K\mathbf{h}, \qquad \mathbf{V} = \mathbf{W}^V\mathbf{h} \tag{5.3}$$

$$\alpha_z = Softmax\left(\frac{\mathbf{Q}_z\mathbf{K}_z^\top}{\sqrt{d}}\right)\mathbf{V}_z \tag{5.4}$$

$$MHA(h) = \big(\alpha_1||\alpha_2||\ldots||\alpha_Z\big)\mathbf{W}^O \tag{5.5}$$

where d_k is the dimension of the keys. Multi-head attention computes this operation in parallel for Z_T heads and concatenates their outputs, followed by a linear projection $\mathbf{W}^O$ to generate the final outputs $\mathbf{h}^{\text{T\_R}}$ for robots and $\mathbf{h}^{\text{T\_T}}$ for tasks respectively.

5.1.3. Reward Prediction

The normalized relative distances between robot i and task j are passed through the distance head MLP_D to compute the distance feature d_{ij}:

$$d_{ij} = \text{MLP}_\text{D}\left(\text{Normalize}(\|\mathbf{p}_i^R - \mathbf{p}_j^T\|_2)\right) \tag{5.6}$$

While task and robot positions are part of the raw input features in $\mathcal{G}^r$ and $\mathcal{G}^t$, this explicit distance term provides the network with direct access to spatial proximity.

We construct feature vectors $\mathbf{f}_{ij}$ for each robot-task pair by concatenating the local (GAT) and global (Transformer) representation of robot i and task j with the distance term d_{ij}, so $\mathbf{f}_{ij} \in \mathbb{R}^{4\times d_k+1}$:

$$\mathbf{f}_{ij} = \mathbf{h}_i^{\text{GAT\_R}} \parallel \mathbf{h}_j^{\text{GAT\_T}} \parallel \mathbf{h}_i^{\text{T\_R}} \parallel \mathbf{h}_j^{\text{T\_T}} \parallel d_{ij} \tag{5.7}$$

This information-rich representation is then passed through the reward head MLP_R to compute the task assignment reward R_{ij} to assign robot i to task j. The idle reward R_i^{IDLE} is computed by passing $\mathbf{f}_{ij}$ to the idle head MLP_I and summing the outputs across all tasks for each robot i:

$$R_{ij}^{task} = \text{MLP}_\text{R}\left(f_{ij}\right), \qquad R_i^{\text{IDLE}} = \sum_{j=1}^{M} \text{MLP}_\text{I}(f_{ij}) \tag{5.8}$$

MLP_I can be understood as learning per-task signals that encourage a robot to wait when short-term idling is advantageous (e.g., a nearby task will become ready soon). The final predicted reward $\mathbf{R}$ contains the task rewards R_{ij}^{task} for all robot-task pairs, concatenated with the idle rewards R_i^{IDLE} for each robot, so $\mathbf{R} \in \mathbb{R}^{\text{N}\times(\text{M}+1)}$.

5.2. Task Assignment through Bipartite Matching

The final reward $\mathbf{R}$ can be interpreted as the edge rewards between robots $\mathcal{R}$ and tasks $\mathcal{T}$ at a given timestep, encoding the full complexity of the current problem state. To extract task assignments at this timestep, we employ a relaxed bipartite matching formulation (no strict one-to-one matching), which finds the optimal assignment matrix $\mathbf{A}^* \in \mathbb{R}^{\text{N}\times(\text{M}+1)}$ that maximizes the selected edge reward encoded in $\mathbf{R}$:

$$\mathbf{A}^* = \arg\max_{\mathbf{A}} \sum_{i,j} A_{i,j} R_{i,j} \tag{5.9}$$

subject to:

$$\sum_j A_{i,j} \leq 1, \quad \forall i \in \mathcal{R} \tag{5.10}$$

$$A_{i,j} = 0, \quad \forall i,j \ : \ i \notin \mathcal{R}_{\text{idle}} \vee j \notin \mathcal{T}_{\text{ready}} \tag{5.11}$$

$$\sum_i A_{i,j}\, \mathbf{c}_i^r \ \succeq \ \mathbf{c}_j^t, \qquad \forall j \tag{5.12}$$

These constraints guarantee valid assignments: (5.10) prevents robots from being assigned more than one task, (5.11) enforces that only idle robots and ready tasks are matched, and (5.12) guarantees that each task's required skills are fully covered by the assigned coalition using the element-wise inequality $\succeq$. This formulation prevents deadlocks, since no coalition can be assigned to a task that it cannot execute/finish. The above formulation allows redundant assignments, so after computing $\mathbf{A}^*$, we remove robots that do not contribute unique required skills, prioritizing those with the highest travel time to the task.

Additionally, we implement a pre-moving strategy: If robot r_i is assigned the idle task t_{M+1}, it moves towards the task with the highest reward $t^i_{\text{highest}} = \arg\max_{1\leq j\leq M} R_{i,j}$, without being formally assigned to it. This does not concatenate the tasks into a fixed schedule for the robot, since assignments are recomputed at decision steps, i.e., when robots finish tasks or new tasks are announced. The robot is likely to be assigned to t^i_{highest} at the next decision step, so pre-moving can reduce the delay to task start if r_i would have been the last coalition member to arrive at t^i_{highest}.

6

Imitation Learning

6.1. Training

We generate 250,000 small-scale problem instances (8 tasks, 3 robots, 3 skills, 3 precedence constraints) with fully randomized configurations, as detailed in Section 4.1. To solve these scenarios optimally, we use the exact MILP formulation, described in Section 4.2. A high-level overview of the training pipeline is depicted in Fig. 6.1.

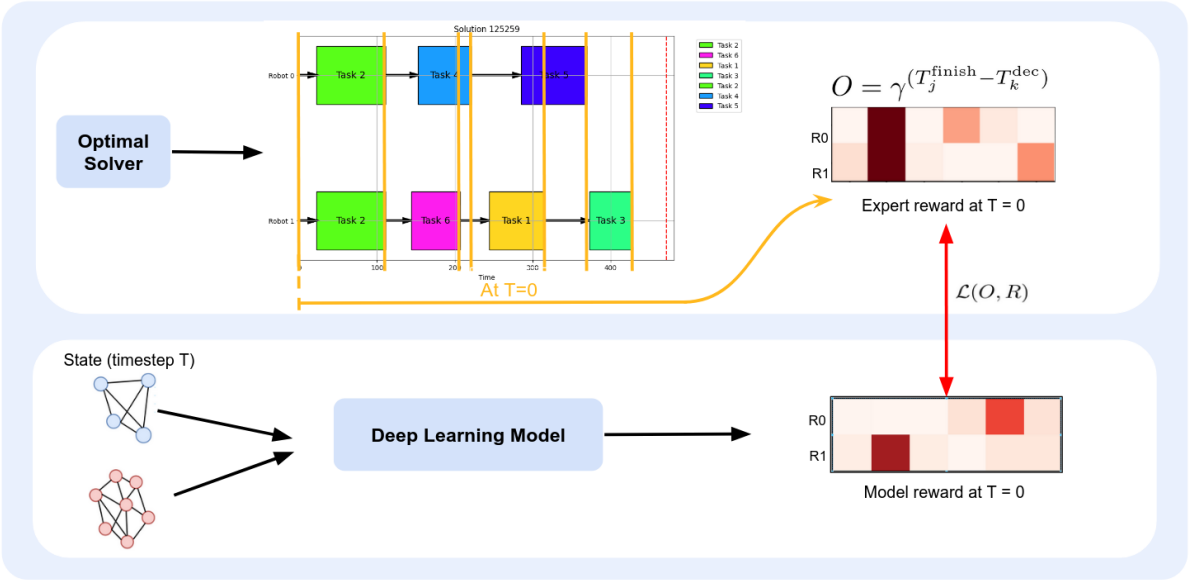

Fig. 6.1. Overview of the Imitation Learning training pipeline and concept. The optimal schedule is sliced into decision points (yellow lines), with an example showing the extraction of the expert reward at the first decision point ($T_k^{\text{dec}} = 0$). This expert reward encodes the optimal decision logic, which the model learns to imitate during training.

6.1.1. Optimal Reward Extraction

To train the network to imitate the optimal behavior, we extract "ground-truth" reward matrices $\mathbf{O_k} \in \mathbb{R}^{N \times (M+1)}$. The optimal schedules are sliced into K decision points T_k^{dec}, corresponding to timesteps when a task finishes and the robots require reassignment. At each decision point the optimal discounted reward is calculated based on the time difference between the decision point T_k^{dec} and the finish time of task j with discount factor $\gamma \in (0, 1]$ (in practice, $\gamma = 0.99$):

$$\mathbf{O_{kij}} = \gamma^{\left(T_j^{\text{finish}} - T_k^{\text{dec}}\right)} o_{kij} \tag{6.1}$$

where o_{kij} is 1 if robot i executes task j in the optimal solution and the decision point precedes the task start time:

$$o_{kij} = \begin{cases} 1, & \text{if } T_k^{\text{dec}} < T_{ij,\text{start}} \ \wedge \ r_i \text{ executes } t_j \\ 0, & \text{otherwise} \end{cases} \tag{6.2}$$

We handle the idle action t_{M+1} in the same way: whenever the time between a robot's last finish time and its next start time exceeds the travel time between the corresponding tasks, we treat this interval as an explicit idle assignment and compute its reward using the above formulas.

By design, this reward encoding captures the optimal decision logic: The next selected task will have the highest reward, with decreasing rewards for later tasks. Given the sequence of optimal rewards $\mathbf{O_k}$ over all decision steps T_k^{dec}, the bipartite matching algorithm outputs the exact solution.

6.1.2. Loss Function

We modify the loss $\mathcal{L}$ from [22] by applying the inverse mask $(1 - \mathbf{X_k})$ to the second term:

$$\mathcal{L} = \|\mathbf{X_k} \circ (\mathbf{R_k} - \mathbf{O_k})\|_1 + \lambda \|(1 - \mathbf{X_k}) \circ (\mathbf{R_k} - \mathbf{O_k})\|_1 \tag{6.3}$$

where $\circ$ denotes the element-wise product operator, $\mathbf{O_k}$ is the optimal reward, $\mathbf{R_k}$ the predicted reward and $\mathbf{X_k} \in \mathbb{R}^{\mathrm{N} \times (\mathrm{M}+1)}$ a feasibility mask and $X_{i,j} = 1$ if robot i is available and task j is ready, else $X_{i,j} = 0$. The first term encourages accurate prediction of feasible rewards, while the second discourages high values for infeasible ones. λ balances the two terms (in practice, $\lambda = 0.1$), intuitively: accurate feasible predictions are more important than suppressing infeasible ones, as the bipartite matching will select high reward tasks. We use the ADAM optimizer [41] to train the network.

6.1.3. Hyperparameter Search

To optimize the performance of our Sadcher IL model, we conduct a hyperparameter search using the OPTUNA framework[1]. The search space targets key architectural and training parameters and is limited to discrete, computationally feasible values, guided by prior empirical validation. OPTUNA automates the search by constructing a sampler that suggests promising hyperparameter combinations based on previous exploration of the search space. Since training the IL model takes only around 1 hour, hundreds of different combinations were tried during the hyperparameter search. Table 6.1 summarizes the explored hyperparameters and the best values identified through the optimization process and Fig. 6.2 shows the average loss for the runs during hyperparameter search.

Parameter	Type	Tried Values	Best Value
Embedding Dimension d_k	Categorical	{32, 64, 128, 256, 512}	256
Fead-Forward Dimension d_{ff}	Categorical	{64, 128, 256, 512, 1024}	512
Transformer Heads Z_{T}	Categorical	{2, 4, 8, 16}	4
Transformer Layers L_{T}	Integer	{1, 2, 3, 4, 6, 8}	2
GAT Heads Z_{GAT}	Categorical	{2, 4, 8, 16}	8
GAT Layers L_{GAT}	Integer	{1, 2, 3, 4, 6, 8}	1
Loss Weight Factor λ	Categorical	{0.1, 0.2, 0.3}	0.1
Learning Rate α	Continuous	{1e-4, 5e-4, 1.e-3, 5e-3}	1e-3
Dropout (Transformer) δ	Continuous	{0.0, 0.1, 0.2, 0.3, 0.4}	0.0

Table 6.1: Summary of the hyperparameter search for the Imitation Learning Model

The final model, configured with the best-found hyperparameters, has approximately 3.3 million trainable parameters. Of these, about 0.1% belong to the embedding layers, 4% to the GATN encoders, 64% to the Transformer encoders, and 16% each to the reward MLP and idle MLP components.

[1] https://optuna.org/

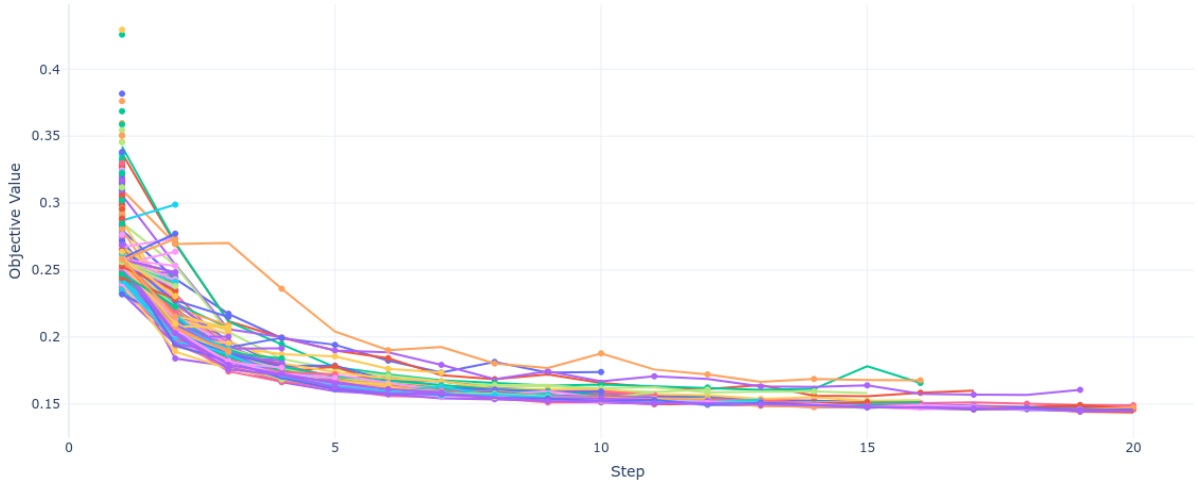

Fig. 6.2. Objective value (average loss per batch) over steps (full training epochs) for different OPTUNA training runs during hyperparameter optimization episodes for the Imitation Learning Sadcher model.

6.2. Results

We report makespan, travel distance, and computation time metrics for all six evaluated algorithms. Results are averaged over hundreds of unseen, randomized problem instances, varying task locations, skill requirements and durations, robot capabilities, depot locations, and precedence constraints. All experiments are conducted on a consumer-grade machine with an AMD Ryzen 7 4800H CPU (8 cores, 2.9 GHz) and NVIDIA GeForce GTX 1650 GPU (4 GB GDDR6 VRAM).

6.2.1. Compared Algorithms

Baselines

We compare our method to: (1) An optimal MILP formulation based on [24], adding precedence constraints and omitting stochastic travel times. The decentralized RL framework HeteroMRTA [35], adapted for precedence constraints by masking out tasks that have incomplete predecessors during action selection. We compare against (2), the single solution variant HeteroMRTA, where agents choose the highest-probability task at decision steps, and (3), the sampling variant S-HeteroMRTA (Boltzmann weighted-random action selection) which returns the best makespan solution across 10 runs per instance. We also implement and compare (4), a greedy heuristic that assigns robots to tasks based on reducing the remaining skill requirements the most and breaking ties based on travel time (shortest first).

Sadcher Variants

(5) The Sadcher framework predicts robot-task rewards deterministically as described in Section 5. We also benchmark (6), a S-Sadcher variant, which samples reward matrices from a normal distribution centered around the deterministic output then used by the bipartite matching. This introduces stochastic variations in resulting schedules. As for S-HeteroMRTA we run this process 10 times per instance and select the best-performing rollout.

6.2.2. Training-Domain Evaluation

We evaluate the algorithms on 500 randomized problem instances of the size of the training domain (8 tasks, 3 robots, 3 precedence constraints). Results are shown in Fig. 6.3 and 6.4.

Makespan

The MILP formulation provides optimal makespans, establishing a baseline for comparing the average relative gaps of other methods. Sample-Sadcher (gap: 3.8%) and Sadcher (gap: 6.8%) are the best-performing non-optimal algorithms. HeteroMRTA without sampling performs worst (gap: 21.5%), but sampling reduces the optimality gap to 10.8%, leveraging its RL policy, which follows a sampling strategy during training. In the pairwise comparison in Fig. 6.4, Sadcher achieves a lower makespan

for 403 of 500 instances (80.6%, binomial test: p $\approx 2 \times 10^{-45}$). Sample-Sadcher outperforms Sample-HeteroMRTA on 389 of 500 instances (77.8%, binomial test: p $\approx 3 \times 10^{-37}$). The greedy algorithm reaches an average gap of 20.4%. All pairwise differences in average makespan are statistically significant ($p < 0.05$) according to the Wilcoxon test (see Table 6.2, except between Sample-HeteroMRTA and Greedy, where no significant difference is observed ($p = 0.2$).

Computation Time

For dynamic scenarios with real-time requirements, the time per assignment decision (t_{dec}) is crucial. MILP cannot compute instantaneous assignments, but only globally optimal schedules. Sample-HeteroMRTA and Sample-Sadcher rely on rolling out the full scenario to select the best assignments. Therefore, these three algorithms, do not yield a time per decision, but only for full solution construction (t_{full}). Due to its simplicity, the greedy algorithm computes the fastest (t_{dec}: 0.080 ms; t_{full}: 1.7 ms). HeteroMRTA is significantly faster (t_{dec}: 9.1 ms; t_{full}: 0.20 s) than Sadcher which needs to solve the relatively expensive bipartite matching for each decision (t_{dec}: 22 ms; t_{full}: 0.57 s). Sample-HeteroMRTA computes full solutions in 0.96 s, Sadcher in 5.7 s and the MILP on average in 76 s. In the worst case, MILP can take up to 12 minutes, rendering it infeasible for real-time applications, even on small problem instances.

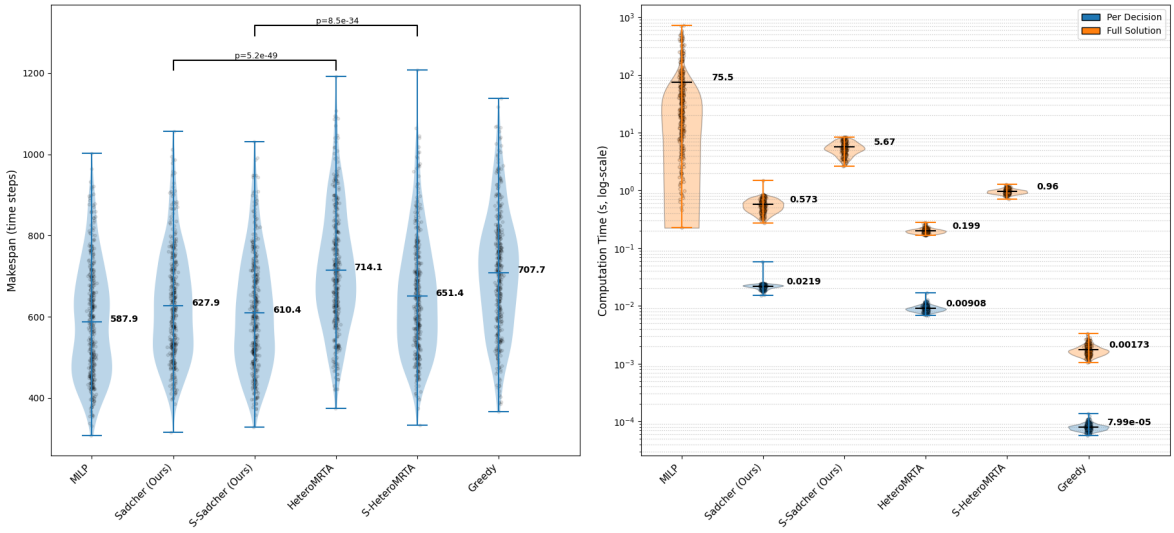

Fig. 6.3. Comparison on 500 unseen, randomized problem instances (8 tasks, 3 robots, 3 precedence constraints) for makespan (left), and computation time (right). Lower means better performance. For algorithms requiring full solution construction, only total computation time is reported; for methods returning instantaneous assignments, both time per decision and total computation time are shown.

	MILP	Sadcher	S-Sadcher	HeteroMRTA	S-HeteroMRTA	Greedy
MILP	1.0e+00	1.2e-58	8.6e-48	5.5e-69	2.4e-58	3.4e-70
Sadcher		1.0e+00	3.4e-20	5.2e-49	5.0e-11	4.0e-48
S-Sadcher			1.0e+00	3.7e-62	8.5e-34	2.0e-63
HeteroMRTA				1.0e+00	7.2e-52	2.0e-01
S-HeteroMRTA					1.0e+00	1.9e-31
Greedy						1.0e+00

Table 6.2: Upper-triangle matrix of pairwise Wilcoxon p-values for makespan comparison reported in Fig. 6.3. Sample-Sadcher is abbreviated as S-Sadcher and Sample-HeteroMRTA as S-HeteroMRTA

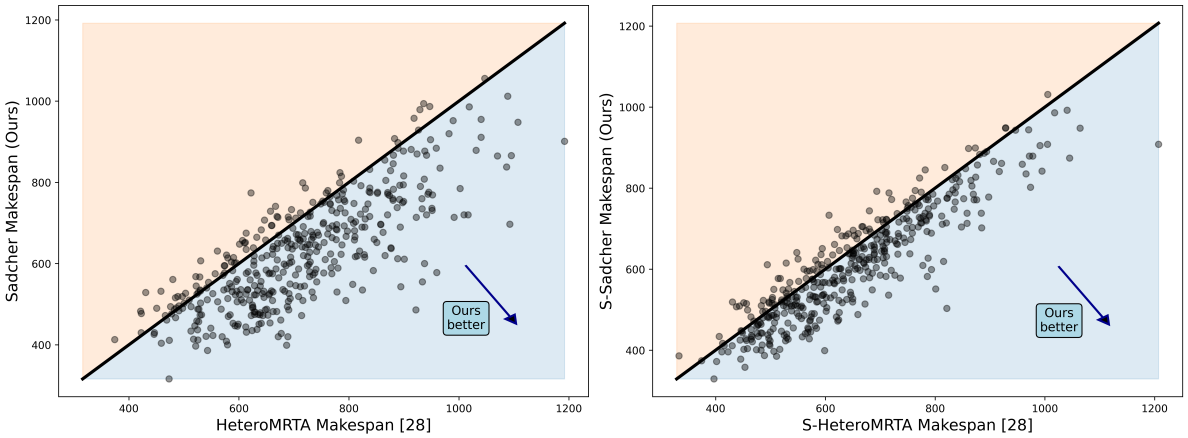

Fig. 6.4. Pair-wise makespan comparison of HeteroMRTA vs. Sadcher (left) and Sample-HeteroMRTA vs. Sample-Sadcher (right). Each point represents one solved problem instance. Points below the diagonal parity line indicate instances where (Sample-)Sadcher achieved a lower makespan, and points above the parity line indicate instances where (Sample-)HeteroMRTA achieved a lower makespan.

6.2.3. Out-of-Domain Generalization

In the following, we analyze the generalization for out-of-domain problem instances, with the number of robots scaled to $N \in [3, 20]$ and tasks to $M \in [6, 250]$. The Sadcher model was trained only on optimal schedules for problems with $N = 3$ robots and $M = 8$ tasks, whereas HeteroMRTA was trained on a different distribution during RL: $N \in [9, 25]$ and $M \in [15, 50]$. We impose a 1-hour time limit for solution generation, under which the MILP fails to solve even moderately sized instances.

Fig. 6.5 shows the performance and runtime scalability for 3-robot scenarios for all compared algorithms. The optimal MILP performs best for problem sizes it can solve within the time limit, outperforming Sadcher by 6% to 7%. Across all task numbers M, Sample-Sadcher and Sadcher are the best-performing non-optimal methods. For $M \leq 60$, Sample-HeteroMRTA outperforms the Greedy algorithm, for $M \geq 60$, greedy surpasses Sample-HeteroMRTA and converges to a relative performance gap w.r.t. Sadcher of around 4%. The non-sampling HeteroMRTA performs worst across all M. Both sampling methods improve performance more on smaller problems. This is because the smaller solution space increases the chance that repeated rollouts will yield substantially different - and better - schedules. As the task count grows, this benefit diminishes, since the likelihood that small variations lead to meaningfully improved schedules lowers. Due to the its exponential complexity, the exact MILP solver fails to find any solutions for problems with 10 or more tasks within the time limit and requires approximately 500 s to solve 9 tasks-problems. In contrast, Greedy is near-instantaneous across all scales taking less than 3 ms to generate new task assignments even for the 250-task problems. HeteroMRTA has a per-decision computation time of less than 20 ms, showing minimal sensitivity to scaling. Full solution generation for Sample-HeteroMRTA requires from 1–40 s. Sadcher is slower (20–80 ms per decision) and computation time scales worse than HeteroMRTA, with Sample-Sadcher runtimes ranging from 4 to 500 s. The increasing runtime gap is due to Sadcher's bipartite matching step, whose cost grows with problem size, while HeteroMRTA's sampling cost remains nearly constant.

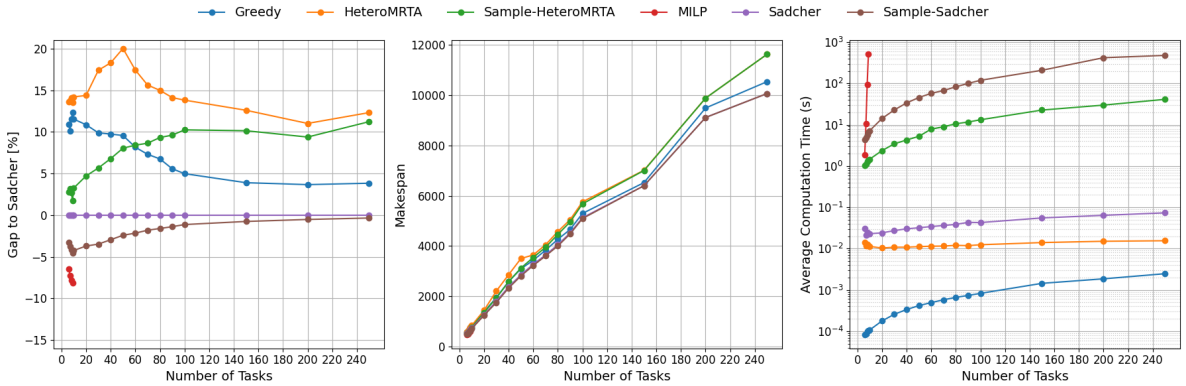

Fig. 6.5. Comparison for problem instances with $N = 3$ robots, task counts $M \in [6, 250]$, $S = 3$ skills, and $M/5$ precedence constraints. Left: Relative makespan gap to the Sadcher algorithm (negative values indicate better performance), Center: absolute makespan, Right: Computation time (for algorithms requiring full solution construction (Sample-HeteroMRTA, Sample-Sadcher, MILP), total computation time is reported, for methods returning instantaneous assignments (HeteroMRTA, Sadcher, Greedy), time per decision is reported). Each point represents the mean across 100 runs per problem size.

In Fig. 6.6 we report the scalability results for 5-robot problems. The computation times are similar to the 3-robot case (Fig. 6.5): The MILP solver fails to find solutions for problems with 10 or more tasks within the time limit, Greedy remains fastest, HeteroMRTA has a per-decision computation time of less than 20 ms. The full solution generation for Sample-HeteroMRTA can take up to 40 s for $M = 250$. Sadcher is slower (20–100 ms per decision) and Sample-Sadcher runtimes range from 5 to 600 s. However, relative performance, shifts significantly compared to the 3-robot analysis: MILP outperforms Sadcher by 12% to 14%. Across all task numbers, Sample-Sadcher remains the best-performing learning-based method, but Sample-HeteroMRTA outperforms Sadcher for instances with $M \leq 9$ and Greedy for $M \geq 60$. For problems with $M \geq 10$, Sadcher is superior and Greedy surpasses Sample-HeteroMRTA for $M \geq 60$ and reaches its best relative performance at $M = 150$ (gap around 2%). The non-sampling HeteroMRTA is better than Greedy for $7 \leq M \leq 10$. Both sampling methods improve performance more on smaller problems, as explained above in the 3-robot analysis (smaller solution space).

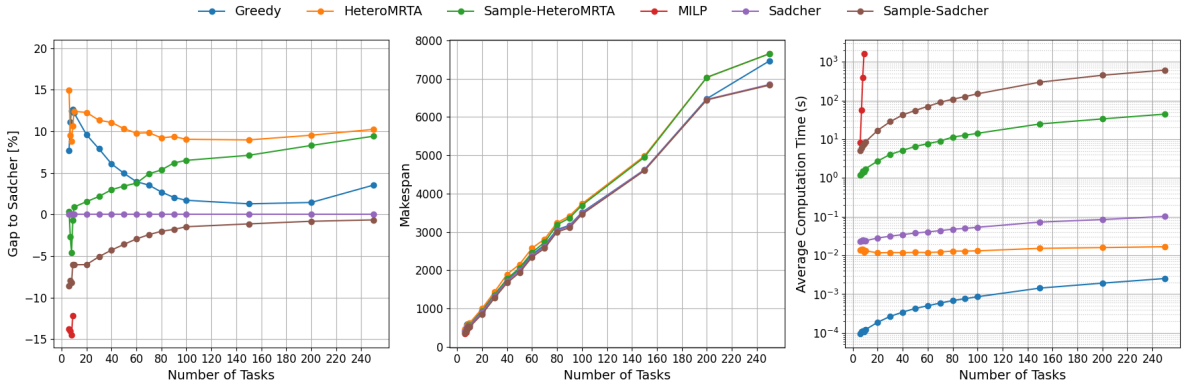

Fig. 6.6. Comparison for problem instances with $N = 5$ robots, task counts $M \in [6, 250]$, $S = 3$ skills, and $M/5$ precedence constraints. Left: Relative makespan gap to the Sadcher algorithm (negative values indicate better performance), Center: absolute makespan, Right: Computation time (for algorithms requiring full solution construction (Sample-HeteroMRTA, Sample-Sadcher, MILP), total computation time is reported, for methods returning instantaneous assignments (HeteroMRTA, Sadcher, Greedy), time per decision is reported). Each point represents the mean across 100 runs per problem size.

In Fig. 6.7 we extend the scalability analysis to 7-robot problems. The patterns for computation times remain consistent with the 3- and 5-robot cases (Fig. 6.5, 6.6), but the MILP solver now takes up to 2000 s to solve $M = 9$ instances, and Sampling-Sadcher up to 950 s for $M = 250$. Relative performance rankings shift for $N = 7$: MILP has a performance gap of 11% to 16%, Sample-Sadcher remains the best learning-based algorithm, but only the best non-optimal method for $M \leq 150$. Sample-HeteroMRTA outperforms Sadcher for $M \leq 40$. Notably, the performance of all learning-based methods

degrades with high task numbers: Greedy begins to outperform Sample-HeteroMRTA at $M \geq 60$, Sadcher at $M \geq 100$, and Sample-Sadcher at $M \geq 200$.

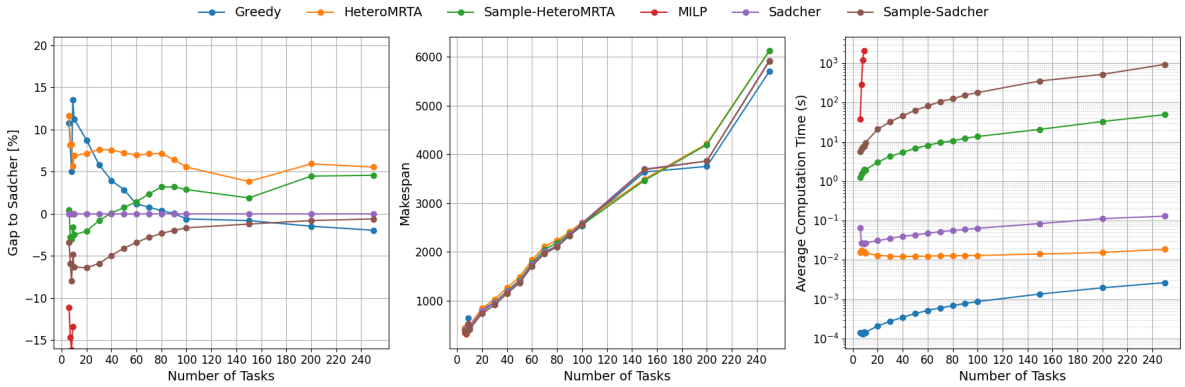

Fig. 6.7. Comparison for problem instances with $N = 7$ robots, task counts $M \in [6, 250]$, $S = 3$ skills, and $M/5$ precedence constraints. Left: Relative makespan gap to the Sadcher algorithm (negative values indicate better performance), Center: absolute makespan, Right: Computation time (for algorithms requiring full solution construction (Sample-HeteroMRTA, Sample-Sadcher, MILP), total computation time is reported, for methods returning instantaneous assignments (HeteroMRTA, Sadcher, Greedy), time per decision is reported). Each point represents the mean across 100 runs per problem size.

Scaling to 20-robot problems leads to notable shifts in generalization results, as shown in Fig. 6.8. MILP results are omitted, as no solutions are found within the 1-hour time limit. Computation time patterns are consistent with the 3-, 5-, and 7-robot cases (Figs. 6.5, 6.6, 6.7), but Sadcher exhibits the steepest increase as task count M grows: Greedy remains fastest (<3 ms per instance), HeteroMRTA stays under 20 ms per decision, and Sample-HeteroMRTA takes up to 55 s for $M = 250$. Sadcher reaches 40–300 ms per decision, with Sample-Sadcher requiring 12–2000 s for full 10 rollouts. Relative performance changes significantly: Sample-Sadcher is the best method for $M \leq 70$, but Greedy becomes superior beyond that. Sample-HeteroMRTA consistently beats Sadcher, yet only outperforms Greedy for $M \leq 50$. Notably, this is the only problem size where HeteroMRTA surpasses Sadcher (for $M \geq 150$), and the performance gap for $M \leq 100$ between the two narrows compared to smaller robot teams.

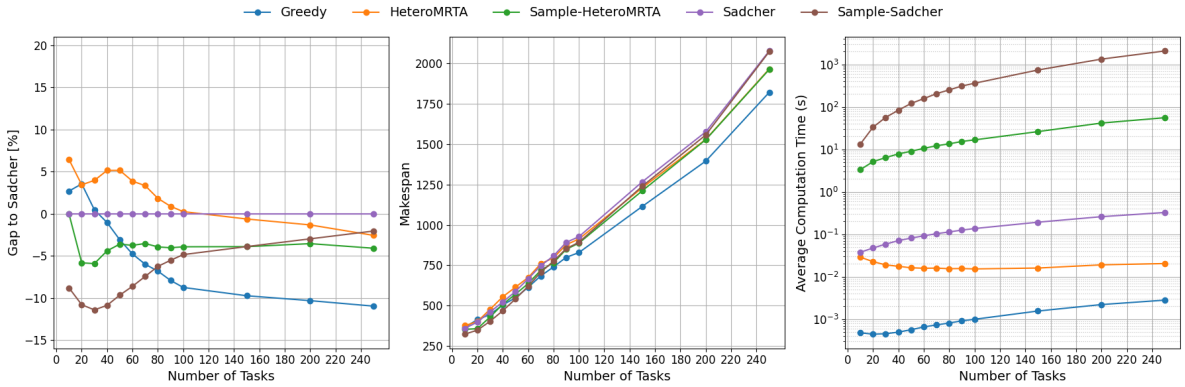

Fig. 6.8. Comparison for problem instances with $N = 20$ robots, task counts $M \in [6, 250]$, $S = 3$ skills, and $M/5$ precedence constraints. Left: Relative makespan gap to the Sadcher algorithm (negative values indicate better performance), Center: absolute makespan, Right: Computation time (for algorithms requiring full solution construction (Sample-HeteroMRTA, Sample-Sadcher), total computation time is reported, for methods returning instantaneous assignments (HeteroMRTA, Sadcher, Greedy), time per decision is reported). Each point represents the mean across 100 runs per problem size.

6.2.4. Focus on Precedence Constraints

The model demonstrates an emergent understanding of task dependencies by prioritizing the assignment of predecessor tasks – tasks that enable other dependent tasks once completed. This strategy improves overall scheduling performance by unlocking successor tasks earlier, thereby increasing the number of available tasks in future decision steps and enabling better global optimization of the task allocation sequence. This behavior emerges during the training phase.

To quantify this, we define the predecessor assignment ratio r_{prec}S, which measures how strongly the model favors assigning predecessor tasks compared to a uniform (random) selection strategy:

$$r_{\text{prec}} = \frac{1}{N} \sum_{i=1}^{N} \frac{a_{\text{prec}}^{(i)}}{a_{\text{total}}^{(i)} \cdot r_{\text{ready}}^{(i)}} \tag{6.4}$$

with:

- N: Total number of decision steps.
- $a_{\text{prec}}^{(i)}$: Number of predecessor tasks assigned at decision step i.
- $a_{\text{total}}^{(i)}$: Total number of tasks assigned at step i.
- $r_{\text{ready}}^{(i)}$: Fraction of ready tasks at step i that are also predecessors:

$$r_{\text{ready}}^{(i)} = \frac{a_{\text{ready, prec}}^{(i)}}{a_{\text{ready, total}}^{(i)}} \tag{6.5}$$

where:

- $a_{\text{ready, prec}}^{(i)}$: Number of ready tasks that are predecessors at step i.
- $a_{\text{ready, total}}^{(i)}$: Total number of ready tasks at step i.

Our model shows an average predecessor assignment ratio$r_{\text{prec}} \approx 1.71$. A value > 1 indicates that the policy assigns predecessor tasks earlier than would be expected from a policy that is not informed about task dependencies.

6.3. Discussion

The results presented in this chapter highlight the performance of our IL methods, Sadcher and Sample-Sadcher, in comparison to several baselines across varying problem sizes. For problem instance within the training domain (8 tasks, 3 robots), both methods significantly outperform the baselines. Sample-Sadcher achieves an average optimality gap of 3.8%, with Sadcher following at 6.8%, compared to 10.8% for the best baseline, Sample-HeteroMRTA. These results show that the reward prediction framework combined with bipartite matching yields high-quality schedules.

One contributor to this performance is the model's emergent understanding of precedence constraints. The IL-trained policy learns to assign predecessor tasks early, increasing the availability of successor tasks and enabling more efficient future assignments. This behavior is quantified by a predecessor assignment ratio of approximately 1.71, indicating that the model selects enabling tasks significantly more often than methods that rely on masking to avoid assigning unready tasks.

In generalization experiments on out-of-distribution problems, both Sadcher methods maintain strong performance as the number of tasks increases. For team sizes of 3 and 5 robots, Sample-Sadcher remains the best-performing non-optimal method across all task counts. An intuitive explanation is that increasing the number of tasks while keeping the number of robots fixed is similar to solving multiple smaller subproblems sequentially, where local scheduling rules learned during training remain effective. However, performance declines with larger robot teams. For 7 robots, Sample-Sadcher performs best only up to 150 tasks, and for 20 robots, only up to 70 tasks. In these cases, the increased coordination complexity introduces new dynamics - larger teams require different local scheduling strategies and

decision logic that diverge from the training distribution. As a result, the greedy baseline, although simple, begins to outperform all learning-based methods at large-scale problems.

Real-time applicability becomes a challenge for Sample-Sadcher on large problem instances, as it requires generating multiple full rollout solutions. As problem size increases, computation times increase to several minutes. This is mainly due to the repeated bipartite matching at each decision step, whose cost grows with the number of robots and tasks. In comparison, HeteroMRTA scales more efficiently in terms of runtime.

These findings motivate the use of RL for fine-tuning our model to increase performance on big problem instances (see Chapter 7). RL does not rely on access to ground-truth schedules. As optimal solutions for medium- and large-scale problems are computationally infeasible to generate, IL cannot be extended to these domains. RL provides a practical alternative that can train directly on medium and large problem instances.

7

Reinforcement Learning

To further increase performance – specifically scalability, which becomes a bottleneck for larger problem sizes (see Section 6.2.3) – we experiment with using Reinforcement Learning (RL) to fine-tune the imitation-learned model. This also allows for the comparison between IL and RL. To do so, we frame the problem in two different ways: once to apply discrete RL and once to apply continuous RL.

7.1. General

Markov Decision Process
RL is a paradigm of machine learning concerned with agents making sequential decisions to maximize a cumulative reward signal. The agent learns by interacting with its environment, receiving feedback in the form of rewards for the actions it takes. This learning process is typically formalized as a Markov Decision Process (MDP). An MDP provides a mathematical framework for modeling sequential decision-making. An MDP is usually defined by a tuple (S, A, P, R, γ), where:

- S is a finite set of states, representing all possible configurations of the environment relevant to the decision-making agent.
- A is a finite set of actions available to the agent.
- $P(s'|s, a)$ is the state transition probability function, denoting the probability of transitioning from state $s \in S$ to state $s' \in S$ after taking action $a \in A$.
- $R(s, a, s')$ is the reward function, which specifies the immediate reward received by the agent after transitioning from state s to state s' as a result of action a. This can be simplified to $R(s, a)$ or $R(s')$.
- $\gamma \in [0, 1]$ is the discount factor, which balances the importance of immediate rewards versus future rewards. A value closer to 0 prioritizes immediate rewards, while a value closer to 1 gives more weight to long-term rewards.

The agent's decision-making strategy is called a policy, denoted $\pi(a|s)$, which is a mapping from states to probabilities of selecting each possible action. The primary goal in RL is to find an optimal policy π^* that maximizes the expected cumulative discounted reward, often referred to as the expected return, starting from an initial state.

Proximal Policy Optimization
Proximal Policy Optimization (PPO) [42] is a highly effective and widely used policy gradient algorithm in Reinforcement Learning, known for its stability and sample efficiency. PPO aims to learn a policy $\pi_\theta(a|s)$, parameterized by θ, by optimizing a surrogate objective function that encourages conservative policy updates. This helps to avoid performance collapses that can occur with overly aggressive updates often seen in standard policy gradient methods.

The core idea of PPO is to constrain the policy update at each iteration, ensuring that the new policy does not deviate too drastically from the old policy. This is achieved through a clipped surrogate objective function, defined as:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{7.1}$$

In this objective, $\hat{\mathbb{E}}_t$ denotes the expectation over a batch of timesteps. The term $r_t(\theta)$ is the probability ratio between the current policy π_θ and the policy used to collect the data (the old policy) $\pi_{\theta_{\text{old}}}$:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{7.2}$$

where s_t and a_t are the state and action at timestep t. The term $\hat{A}_t$ is an estimator of the advantage function at timestep t, which typically quantifies how much better action a_t is compared to the average action in state s_t. The $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ function constrains $r_t(\theta)$ to the interval $[1-\epsilon, 1+\epsilon]$, where ϵ is a small hyperparameter (e.g., 0.1 or 0.2). This clipping discourages the policy from moving too far from the old policy by penalizing large changes in the probability ratio, thereby ensuring more stable learning. PPO algorithms typically alternate between sampling trajectories from the environment using the current policy and performing multiple epochs of stochastic gradient ascent on the objective $L^{CLIP}(\theta)$ using the collected data.

7.1.1. Reinforcement Learning on our use case

State

As for the Imitation Learning approach, we model the state/input to our RL policy as robot and task graphs: The N heterogeneous robots with $S_i \subseteq \mathcal{S}$ distinct skills, are modeled as an undirected graph $\mathcal{G}^r = (\mathcal{R}, \mathbf{C})$, where each vertex in $\mathcal{R} = \{\mathbf{r}_i\}^N$ is a robot. Robot states $\mathbf{r}_i = [\mathbf{p_i^r}, t_i^r, a_i^r, \mathbf{c_i^r}]$ include position $\mathbf{p_i^r} \in \mathbb{R}^2$, remaining duration at the current task t_i^r, the robot's availability $a_i^r \in \{0,1\}$, and the binary capability vector over the global skill set $\mathbf{c_i^r} \in \{0,1\}^{|\mathcal{S}|}$. $\mathbf{C} \in \{0,1\}^{N\times N}$ represents the network connection among the robots. For simplicity, we assume a fully connected graph, so $C_{i,j} = 1\ \forall i,j$. However, our model is designed to accept any connected graph structure as input. The M tasks and their respective precedence constraints are represented as a directed acyclic graph $\mathcal{G}^t = (\mathcal{T}, \mathbf{P})$. Each task is a vertex in $\mathcal{T} = \{\mathbf{t}_j\}^M$, and is described by $\mathbf{t}_j = [\mathbf{p_j^t}, t_j^t, \mathbf{r_j^t}, s_j^t]$, with position $\mathbf{p_i^r} \in \mathbb{R}^2$, expected duration t_j^t, required skills $\mathbf{r_j^t} \in \{0,1\}^{|\mathcal{S}|}$ and status $\mathbf{s_j^t} \in \{0,1\}^3$. The status indicates whether tasks are ready, assigned, or incomplete, e.g., $\mathbf{s}_j^t = [1,0,1]$ represents a task that is ready to be scheduled, currently not assigned, and incomplete. Precedence constraints are encoded in the edges $\mathbf{P}^{M\times M}$, where $P_{i,j} = 1$ means the i-th task is a predecessor of the j-th task.

Action

The form of the action depends on which of the two RL formulations is used: For the discrete variant (Section 7.3) the policy outputs, for every currently available robot, a soft-maxed categorical distribution over the M real tasks that are sampled from to generate into robot-task assignments. For the continuous variant (Section 7.2), the policy outputs an $(N \times (M+1))$ sampled reward matrix – including an explicit idle task, that the bipartite matcher turns into assignments.

Transition Function

The simulation is fully deterministic: once the current state s and the chosen action a (the set of robot–task assignments) are known, the simulator advances the clock to the next decision point and updates every component of the state in a determinstc way – robots traverse straight-line paths at a constant speed, task timers count down at a fixed rate, and precedence flags are flipped as soon as all prerequisites finish. Hence the state-transition kernel reduces to a Dirac delta,

$$P(s' \mid s, a) = \begin{cases} 1, & \text{if } s' = \text{sim}(s,a), \\ 0, & \text{otherwise.} \end{cases} \tag{7.3}$$

where $\text{sim}(s,a)$ is the deterministic simulator that computes the successor state.

Reward
The reward function is sparse, provided only at the termination of an episode (i.e., when all tasks are completed and robots have reached their final destination or the worst-case makespan is exceeded). The worst-case makespan T_{wc} is equivalent to the time a single robot would need to perform all tasks back-to-back while always traveling the maximum possible distance between subsequent tasks:

$$T_{\text{wc}} = \sum_{j=1}^{M} T_{e,j} \; + \; \sum_{j=1}^{M} \max_{i \in \{1,\dots,M\}} T_{t,i,j} \tag{7.4}$$

where $T_{e,j}$ is the execution time of task j and $T_{t,i,j}$ is the travel time for robot i to reach task j. When the simulation time exceeds T_{wc} (because a deadlock occured), we set $T_{\text{ms, policy}} = T_{\text{wc}}$. Since sampled problem instances can have different complexity and expected makespans (e.g., due to varying task durations and requirements), using the negative makespan directly as the reward signal is unstable. Therefore, we normalize the reward w.r.t the makespan the greedy algorithm finds. This gives a good estimate on the difficulty of the sampled instance and is fast to evaluate. The total episodic reward R_{ep} is designed to guide the policy towards minimizing the makespan relative to the baseline greedy scheduler:

$$R_{ep} = -\frac{T_{\text{ms, policy}} - T_{\text{ms, greedy}}}{T_{\text{ms, greedy}}} \tag{7.5}$$

where $T_{\text{ms, policy}}$ is the makespan achieved by the learned policy in the episode, and $T_{\text{ms, greedy}}$ is the makespan achieved by the greedy scheduling heuristic on the same problem instance. A positive reward indicates an improvement over the greedy scheduler.

Discount Factor
Throughout all experiments, we use the standard discount factor $\gamma = 0.99$.

7.1.2. Training

For both the discrete formulation (Section 7.3) and the continuous formulation (Section 7.2) we employ the PPO implementation provided by the `skrl` library [43].[1] Across all experiments — independent of the formulation-specific tuned hyper-parameters — we explore all combinations of the following design choices:

1. **Initialization.** Either initialize the policy network with weights obtained from the IL stage, or train the policy entirely from scratch (random weight initialization).
2. **Frozen encoders versus training the full network.** When the policy is initialized from IL, we optionally freeze all encoder layers and optimize only the final MLP that maps the latent representation to the reward. This keeps roughly 85% of the total parameters fixed.
3. **Advantage estimation with or without a learned critic.** Because our reward depends simultaneously on the performance of the greedy baseline and the current policy, fitting a value function $V(s)$ proved difficult, as two similar instances can yield highly different returns for the greedy policy. We therefore compared the standard PPO variant, which learns $V(s)$, with an alternative that replaces $V(s)$ by a constant zero baseline (explained below).

Generalized Advantage Estimation
PPO estimates the advantage $\hat{A}_t$ using Generalized Advantage Estimation (GAE) [44] to reduce variance:

$$\hat{A}_t^{\text{GAE}}(\gamma, \lambda) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+l}, \tag{7.6}$$

with the temporal-difference residual

$$\delta_t \; = \; r_t + \gamma V(s_{t+1}) - V(s_t). \tag{7.7}$$

[1] https://skrl.readthedocs.io/en/latest/

Zero-critic variant
Setting $V(s_t) \equiv 0$ for all t turns the TD residual into $\delta_t = r_t$, yielding a discounted-return estimator

$$\hat{A}_t^{\mathrm{GAE}}(\gamma, \lambda) \;=\; \sum_{l=0}^{\infty} (\gamma\lambda)^l \, r_{t+l}. \tag{7.8}$$

Thus, when the critic is disabled PPO effectively optimizes online Monte-Carlo returns while still benefiting from the PPO clipping objective and KL divergence threshold.

7.2. Continuous Reinforcement Learning

To stay close to the IL formulation—allowing the reuse of the pre-trained network parameters and preserving feasibility guarantees—we first explore a continuous RL approach.

7.2.1. Formulation

Extending the general formulation (Section 7.1.1), the continuous RL approach keeps the scheduling pipeline of the IL baseline. The policy therefore does not directly sample actions from categorical distributions like the discrete approach (Section 7.3). Instead, the policy samples a reward matrix $\mathbf{R} \in \mathbb{R}^{N\times(M+1)}$, visualized in Fig. 7.1 and task assignments are generated via bipartite matching based on this reward matrix. Because the bipartite matcher enforces feasibility, assignments are only made when the coalition of assigned robots fully satisfies a task's skill requirements. To allow for strategic waiting, the action space includes an explicit idle/waiting task $M{+}1$. This enables robots to delay their assignments when no suitable coalition can yet be formed or when postponing execution is advantageous.

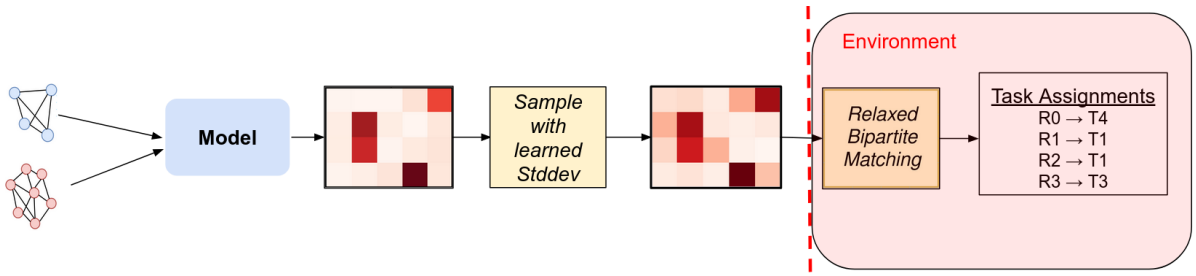

Fig. 7.1. High-level overview of the continuous Reinforcement Learning pipeline. The bipartite matching is part of the environment and not the agent's policy, which outputs the sampled reward assignment matrix as its action.

Stochastic reward matrix sampling
The policy outputs a mean matrix $\boldsymbol{\mu}_{\mathsf{k}} \in \mathbb{R}^{N\times(M+1)}$ together with a single log-standard-deviation parameter $\ell_{\mathsf{k}} \in \mathbb{R}^{+}$, which is shared by every entry of the score matrix: $\ell_{i,j} = \ell \;\; \forall i, j$. Hence the common standard deviation is $\sigma_{\mathsf{k}} = \exp(\ell_{\mathsf{k}})$. Drawing element-wise noise $\varepsilon_{\mathsf{k}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, the reward matrix passed to the bipartite matcher is sampled with the re-parameterization trick

$$\mathbf{R}_{\mathsf{k}} \;=\; \boldsymbol{\mu}_{\mathsf{k}} + \; \sigma_{\mathsf{k}}\varepsilon_{\mathsf{k}} \tag{7.9}$$

Extraction of task assignments
We employ a relaxed bipartite matching formulation (no strict one-to-one matching), which finds the optimal assignment matrix $\mathbf{A}^*{}_{\mathsf{k}} \in \mathbb{R}^{\mathrm{N}\times(\mathrm{M}+1)}$ that maximizes the selected edge reward encoded in $\mathbf{R}_{\mathsf{k}}$, exactly like in the IL formulation, described in Section 5.2. Technically, this matching step is handled by the environment. The PPO agent itself only outputs a stochastic reward matrix $\mathbf{R}_k$ given the current state s_k. In this sense, the policy does not make actual task-selection decisions directly but instead learns to produce continuous-valued preferences over all possible robot–task pairs:

$$\pi_\theta(s_k) \;\longrightarrow\; \mathbf{R}_k \in \mathbb{R}^{N\times(M+1)} \tag{7.10}$$

These scores are then interpreted and resolved into assignments by the environment.

Training on continuous action space
Because the re-parameterization trick $\mathbf{r} = \boldsymbol{\mu} + \sigma\varepsilon$ is differentiable, gradients flow through both the mean matrix and the shared log-std ℓ. With the PPO formulation (Section 7.1) , the policy therefore learns to:

(i) raise $\boldsymbol{\mu}_{i,j}$ for robot–task pairs that the matcher selects in high-reward episodes,

(ii) lower it for pairs that lead to dead-ends or long makespans,

(iii) and anneal σ over time, shrinking exploration as soon as the clipped objective signals convergence, i.e. the best-performing episodes are sampled close to the mean $\boldsymbol{\mu}$.

7.2.2. Results

We evaluate various PPO hyperparameter configurations in combination with the training strategies described in Section 7.1.2 to fine-tune the continuous RL policy. As shown in Figure 7.2, the initial average reward (i.e., performance gap relative to the greedy baseline) starts around 4%. While several runs initially show improvements during training, none surpass the performance of the pure IL approach, which achieves an average performance gap of 8% (indicated by the blue dashed line). The IL policy is trained on smaller instances (8 tasks, 3 robots), whereas the fine-tuning is performed on medium-sized instances (20 tasks, 5 robots) to assess the feasibility of RL to improve scalability. In some training runs, the policy collapses - typically caused by overly aggressive or unstable hyperparameter settings, like high clip ratio r_t or high learning rates α. We also experiment with curriculum learning, gradually increasing the problem size during fine-tuning up to 100 tasks and 20 robots. However, this approach also resulted in policy collapse or failed to yield improvements over the pure IL baseline.

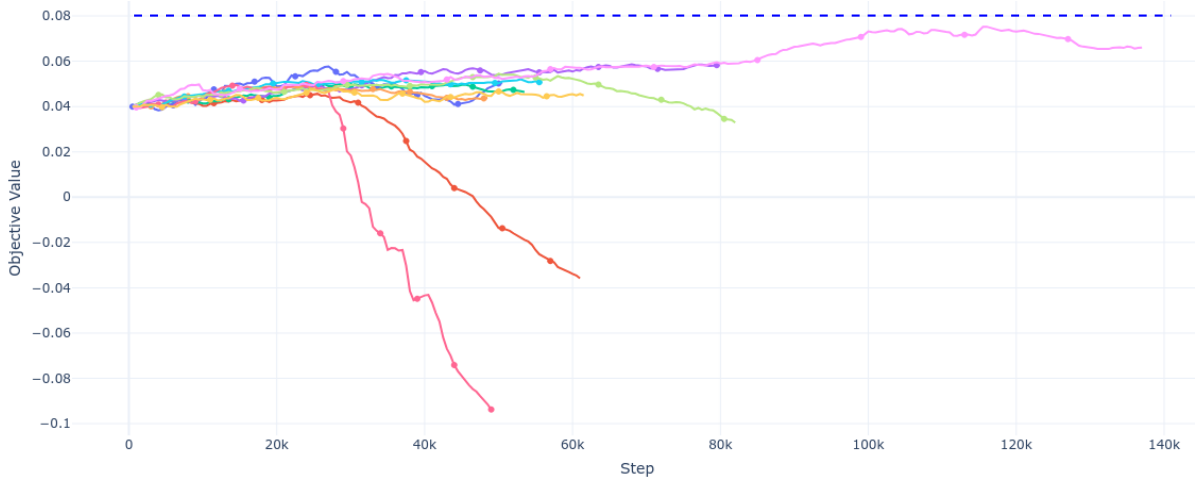

Fig. 7.2. Continuous RL fine-tuning training runs with different hyperparameters. Objective value (mean reward as performance gap w.r.t. greedy, see Section 7.1.1) over finished training step (episodes). The model is fine-tuned on problem instances of 20 tasks, 5 robots, 3 skills, and 5 precedence constraints. On this problem size, the pure IL approach achieves a value of 0.08 (blue horizontal line). Some runs approach this value, but none outperform it.

7.2.3. Discussion

The continuous RL formulation does not yield improvements over the pure IL baseline. Several interrelated factors might contribute to this result:

High-dimensional action space
In the continuous formulation, the policy outputs an entire reward matrix of size $\mathbb{R}^{N\times(M+1)}$ at each timestep. For instance, with 20 tasks and 5 robots, this amounts to 105 continuous values per decision step. Such a high-dimensional action space makes optimization difficult: gradient signals must propagate through many parameters, increasing the risk of vanishing gradients and hindering the policy to update in a way that improves scheduling decisions.

Obscured reward–action mapping due to black-box matching
Another critical challenge lies in the separation between the policy's output (assignment reward matrix) and the actual action taken (task assignments). The assignment logic is handled as part of the environment via bipartite matching. While this design offers feasibility guarantees, it introduces a black-box step that obscures the reward–action mapping. The policy does not directly choose actions; it only influences them via the output assignment matrix, making it harder to learn which parts of the matrix led to high or low-performing episodes.

Improving strong IL baseline proves hard
The IL baseline policy imitates optimal schedules and already achieves strong performance. Fine-tuning this policy through PPO may not yield further gains, particularly because the solution landscape discovered by IL differs from what RL gradients typically optimize. In other words, while IL approximates optimal actions through direct supervision, RL relies on sparse, delayed feedback and trial-and-error exploration. Starting from an IL policy may bias the RL policy into a local optimum that is hard to escape through PPO updates alone.

Environment complexity and sparse rewards
The RL agent operates in a highly complex environment with sparse, episodic rewards — only provided after task completion. This lack of intermediate signals severely restricts the policy's ability to learn effective long-horizon strategies. The challenge is further intensified by the large action space and the black-box reward–action mapping introduced by the bipartite matching step.

7.3. Discrete Reinforcement Learning

Due to the limitations observed with continuous RL — particularly the high-dimensional action space and the obscured reward-action mapping — we also design and evaluate a discrete RL formulation. This reduces the action space dimensionality and gives the policy direct control over task selection, but does not guarantee feasibility.

7.3.1. Formulation

In addition to the general formulation (Section 7.1.1), the discrete RL variant models the action selection process as follows (see Fig. 7.3): At each decision point k, every available robot $i \in \{1, \dots, N_{\text{free}}\}$ selects one of the M real tasks. In contrast to the IL and continuous RL settings, we omit the explicit `IDLE` action (task $M+1$) in the discrete formulation. This is because the discrete policy samples actions independently for each robot from a categorical distribution, allowing robots to select tasks even if they cannot complete them alone. In the continuous and IL variants, the explicit idle action is necessary: the bipartite matching step enforces that only valid coalitions capable of completing the task are scheduled. When such coalitions cannot yet be formed, or when it is strategically advantageous to delay assignment, robots can instead choose to wait and move toward the most likely future task.

The policy network (same as for the other approaches, omitting the `IDLE` MLP) $f_\theta : \mathcal{S} \to \mathbb{R}^{N \times (M)}$ first maps the graph state s_k to a matrix $\mathbf{R}_k$:

$$\mathbf{R}_k = f_\theta(s_k) \;=\; \begin{bmatrix} R_{1,1} & \dots & R_{1,M} \\ \vdots & & \vdots \\ R_{N,1} & \dots & R_{N,M} \end{bmatrix} \tag{7.11}$$

Applying a row-wise (per-robot) soft-max to the matrix $\mathbf{R}_k$ yields the categorical distributions over tasks for each robot. The resulting policy π_θ defines the probability that robot i selects task j at timestep k:

$$\pi_\theta(a_i = j \mid s_k) \;=\; \frac{\exp R_{k,i,j}}{\sum_{l=1}^{M} \exp R_{k,i,l}}, \qquad j \in \{1, \dots, M\} \tag{7.12}$$

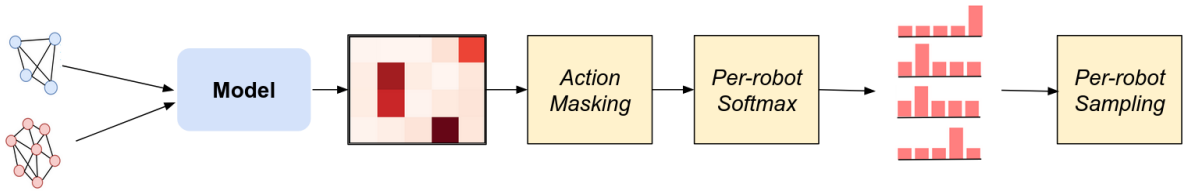

Fig. 7.3. High-level overview of the discrete Reinforcement Learning pipeline. In contrast to the IL and continuous RL approaches, bipartite matching is omitted—so feasibility is not enforced by design. Instead, action masking is used to guide the categorical distributions toward feasible task assignments.

Action masking
Not every entry corresponds to a legal action or makes sense in the broader task assignment context. To guide the policy towards better/feasible assignments, we apply a binary mask $\mu_{k,i,j} \in \{0,1\}$ over the reward matrix. An entry $\mu_{k,i,j} = 1$ is valid if and only if all of the following conditions are met:

(i) task j is ready (predecessors completed) and incomplete,

(ii) robot i has at least one skill that matches task j's requirements.

Entries not meeting these conditions are masked out by setting their corresponding logit to a large negative value:

$$R_{k,i,j} \;=\; \begin{cases} R_{k,i,j}, & \mu_{k,i,j} = 1, \\ -\infty, & \mu_{k,i,j} = 0. \end{cases}$$

Applying the row-wise (per-robot) soft-max yields a categorical distribution with no probability mass for the masked-out entries.

Subtractive assignment
To inform the policy which robots are already assigned to a task, we subtract the capabilities of robots that are already assigned to task j from its skill-requirement vector $\mathbf{r}_j^t$:

$$\tilde{\mathbf{r}}_{j,k} \;=\; \mathbf{r}_j^t \;\ominus \bigvee_{i \in \mathcal{R}_j^{\text{assigned}}} \mathbf{c}_i,$$

where $\mathcal{R}_j^{\text{active}}$ is the set of robots currently committed to j and $\ominus$ denotes element-wise clipped subtraction on $\{0,1\}$. c_i are the capability vectors of the assigned robots. Only if $\tilde{\mathbf{r}}_{j,k} \neq \mathbf{0}$ can another robot contribute, which tightens the mask μ and hence the action space.

Training on discrete action space
In the discrete setting, the policy outputs logits that define a categorical distribution over tasks for each robot. During training, PPO samples task selections from these distributions and uses the resulting rewards to adjust the logits. Gradients flow through the softmax via the log-likelihood of the sampled actions, allowing the policy to:

(i) increase logits for task choices that consistently lead to high rewards,

(ii) decrease logits for choices that result in poor outcomes,

(iii) and reduce entropy over time as the clipped objective favours more confident, stable selections.

7.3.2. Results

Since the task assignment logic in the discrete RL approach differs from both the IL and continuous RL variants, we first train the policy on small problem instances. This enables the network to learn how to decode latent embeddings in a way that is not optimized for input into a bipartite matcher, but rather for sampling directly from a categorical distribution. As shown in Figure 7.4, the discrete RL approach initially produces a high number of infeasible episodes, in contrast to the continuous RL setup (Section 7.2) where feasibility is guaranteed through bipartite matching. This causes a negative performance gap relative to the greedy baseline during the early training phase. After approximately 500,000 training episodes, the discrete policy reaches parity with the greedy baseline and reduces

the infeasibility rate to around 2%. The performance gap eventually stabilizes at roughly 4%, while the pure IL baseline—trained on these small instance sizes—achieves a significantly better 12% gap w.r.t. greedy. This highlights that IL remains superior for problem sizes seen during training. However, the primary motivation for exploring RL lies in its potential to improve scalability to problem sizes, that we cannot generate optimal solutions for. Therefore, we continue training the best discrete RL model with varying hyperparameters on medium-sized instances with 20 tasks and 5 robots. The results, shown in Figure 7.5, indicate that some configurations achieve slight performance improvements, but the best-observed performance gap is 5.2%—still worse than the IL baseline, which achieves an 8% gap on the same problem size. For reference, the best continuous RL run on this instance size reached approximately 7.5% (see Figure 7.2).

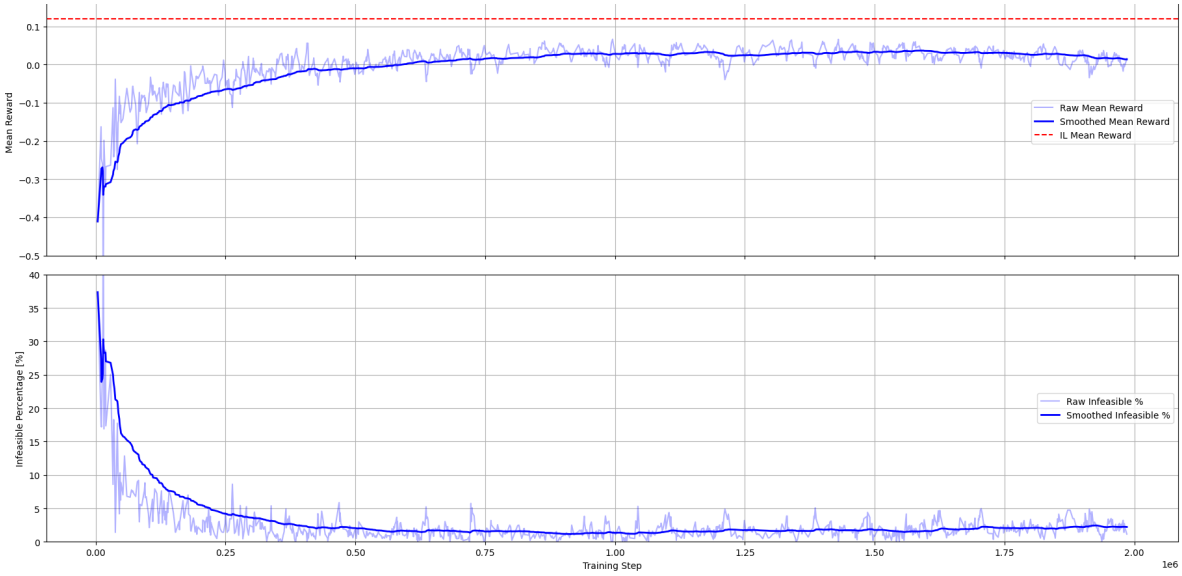

Fig. 7.4. Discrete RL training run on instances of 10 tasks and 3 robots. Top: Mean reward as performance gap w.r.t. greedy, see Section 7.1.1, over finished training steps (episodes). The red horizontal line shows the performance of the pure IL baseline on this problem size (12%). Bottom: Ratio of infeasible episodes over finished training episodes.

Fig. 7.5. Discrete RL fine-tuning training runs with different hyperparameters. Objective value (mean reward as performance gap w.r.t. greedy, see Section 7.1.1) over training steps (finished episodes). The previously found model (trained with discrete RL on 10 tasks, 3 robots) is fine-tuned on problem instances of 20 tasks, 5 robots, 3 skills, and 5 precedence constraints. For reference: On this problem size, the pure IL approach achieves a value of 0.08

7.3.3. Discussion

The discrete RL formulation offers several practical advantages but fails to reach or improve the performance of the IL baseline.

Computation Time
Training and inference in the discrete RL setup are faster than in the continuous formulation, primarily due to the removal of the bipartite matching step. Sampling directly from categorical distributions is faster than solving the bipartite optimization problem.

Improved Policy–Action Alignment
The discrete formulation provides a more direct connection between the policy output and the final task assignments. This avoids the black-box behavior introduced by the bipartite matcher in the continuous setup and the lower-dimensional action space aligns better with typical PPO use cases.

Lack of Feasibility Guarantees
A major downside is the possibility of deadlocks occurring. Unlike the continuous approach, the discrete method does not guarantee that assigned coalitions meet all skill requirements. Although action masking mitigates infeasibility, it cannot fully prevent it, especially in early training phases.

Performance
Despite better alignment with RL design principles, the discrete RL formulation does not match or surpass the continuous RL approach or the IL baseline in terms of performance or scalability.

Model Architecture Limitations
The current architecture is not optimized for discrete RL. Specifically, inference is triggered globally for all robots whenever any robot becomes available, even though only a subset needs new assignments. This results in redundant computation. A more suitable design would allow per-robot inference, using inputs relative to the querying robot to generate a single categorical distribution over tasks. This could enable better performance and more stable training

8

Simulation Environment

In order to compare the baselines and the proposed methods, we develop a light-weight simulation environment to represent task and robot states and roll out instantaneous task assignment decisions into full schedules.

8.1. Core Data Structures

This section outlines the core data structures used in the simulation framework, defining the states of tasks and robots, and schedules at both full-horizon and instantaneous levels. Task precedence constraints are represented using an adjacency matrix, where each directed edge (i, j) encodes a dependency between two tasks: t_i has to be completed before t_j can be started.

Tasks

Attribute	Description
`task_id`	Unique integer
`location`	Position in $\mathbb{R}^2$
`remaining_duration`	Timesteps remaining to complete the task
`requirements`	Boolean vector indicating required skills
`ready`	Boolean, are all predecessors completed?
`assigned`	Boolean, currently assigned?
`incomplete`	Boolean, True until `remaining duration` = 0

Robots

Attribute	Description
`robot_id`	Unique integer
`location`	Current position in $\mathbb{R}^2$
`remaining_workload`	Timesteps left to complete assigned task
`capabilities`	Boolean vector indicating available skills
`current_task`	Task currently assigned to the robot
`available`	Boolean, `True` if `current_task` is `IDLE` or `None`
`speed`	Movement speed per timestep

Full-Horizon Schedule

Attribute	Description
`makespan`	Total time until all robots reach the end location
`robot_schedules`	Map: `robot_id` $\mapsto$ list of `(task_id, start_time, end_time)` triplets

Instantaneous Assignment

Attribute	Description
`robot_assignments`	Map: `robot_id` $\mapsto$ next `task_id`

8.2. Simulation Logic

Algorithm 1 High-Level Simulation Logic

Require: Problem instance $\mathcal{P}$, Scheduler type s
$sim \leftarrow$ InitializeSimulation$(\mathcal{P})$
$\mathcal{S} \leftarrow$ create_scheduler(s, sim)
while not `all_tasks_done`(sim) **do**
  $\mathcal{A} \leftarrow \mathcal{S}$.`calculate_robot_assignments`$(sim)$
  sim.`assign_tasks_to_robots`$(\mathcal{A})$
  sim.`step_until_next_decision()`
end while

Algorithm 2 sim.`step_until_next_decision()`

$a_{\text{prev}} \leftarrow -1$
while true **do**
  for all robots r **do**
    if r.`current_task` is not IDLE **then**
      r.`move_to_task`$(r$.`current_task`$)$
    else if sim.`robot_can_still_contribute_to_tasks`(r) **then**
      r.`move_to_task`$($`find_task_to_premove_to`$(r))$
    else
      r.`move_to_task`$($`EXIT`$)$
    end if
  end for
  sim.`update_task_status`$()$
  sim.`update_task_duration`$()$
  sim.`update_robot_status`$()$
  sim.`timestep` $\leftarrow sim$.`timestep` $+ 1$
  $a \leftarrow$ number of available robots
  if sim.`sim_done` **or** sim.`robot_finished_task()` **or** sim.`new_task_announced()` **then**
    break
  end if
  if sim.`timestep` $\geq sim$.`worst_case_makespan` **then**
    sim.`finish_simulation`$()$
    sim.`makespan` $\leftarrow sim$.`worst_case_makespan`
    break
  end if
  $a_{\text{prev}} \leftarrow a$
end while

In Algorithm 1, the simulation is initialized with a problem instance and a selected scheduler type (e.g., greedy, SADCHER, ...). The main loop runs until all tasks are completed. At each iteration, the sched-

uler generates an instantaneous assignment of robots to tasks based on the current simulation state. After the assignment is sent to the robots, the control is passed to the inner simulation loop (Algorithm 2), which advances the system until a new scheduling decision is required. This design ensures that the scheduler is invoked only when necessary—specifically, when the set of available robots changes due to task completion or failure.

Algorithm 2 describes the logic that advances the simulation until a new scheduling decision is required or a deadlock has been detected. In the latter case, the simulation is terminated; in the former case, control is handed over to the high-level scheduler (Algorithm 1):

1. **Move robots (lines 3–10).**
   For every robot r_i exactly one of the following targets is selected and the robot moves toward it:
   (a) *Assigned task.* Robots already allocated to a task t_j continue toward its location; if they are already there, they remain stationary.
   (b) *Pre-move.* A robot that is currently idle (t_{M+1}) heads toward the "best" normal task
   $$t^i_{\text{highest}} = \arg\max_{1 \le j \le M} R_{i,j},$$
   where $R_{i,j}$ is the individual reward estimate. This speculative motion can shorten the eventual start-up delay once suitable coalition partners become free.
   (c) *Exit.* If r_i can no longer contribute to any tasks (i.e., none of its capabilities are needed for any unfinished/unassigned tasks), it moves to the exit/end location.
2. **Task status update (lines 12–13).**
   Each task's state is refreshed: *ready*, *assigned*, or *incomplete*. If all the required skills are present in the assigned coalition and all coalition robots are at the task location, the remaining duration is decremented.
3. **Robot status update (line 14).**
   Robots who have just completed tasks become *available* and *unassigned*. Robots that have been scheduled become *unavailable* and *assigned*.
4. **Termination check.**
   After the global clock advances by one step, the loop exits when
   (i) a robot has finished a task or a new task is announced, which signals that computing a new assignment is necessary (lines 17-18), or
   (ii) the execution time exceeds a worst-case makespan, defined as the sum of all travel times and processing times. If a simulation instance runs for more than that, a deadlock has occured.

8.3. Example Visualization

For the solution visualized in Fig. 8.1, robot $R0$ possesses all skills, and after completing task $T2$, it focuses on completing the tasks that require multiple skills in the ”lower” area where $y \leq 20$. On the other hand, robots $R1$ and $R2$ only possess skills 0 ($[1, 0, 0]$) and skills 1 and 2 ($[0, 1, 1]$) respectively. After completing some tasks, that they can handle individually without a coalition, they form a dynamic coalition on tasks $T3$, $T6$, and $T12$ to shorten the overall makespan. Without dynamic coalitions, $R0$ would have required a lot of time to execute all tasks that require multiple skills and $R1$ and $R2$ are not capable of executing by themselves. The gap in the schedule of $R1$ between $T1$ and $T3$ is a waiting period, in which the robot has strategically pre-moved to later form the coalition on $T3$. This example shows how the Sadcher algorithm coordinates heterogeneous teams with coalition formation.

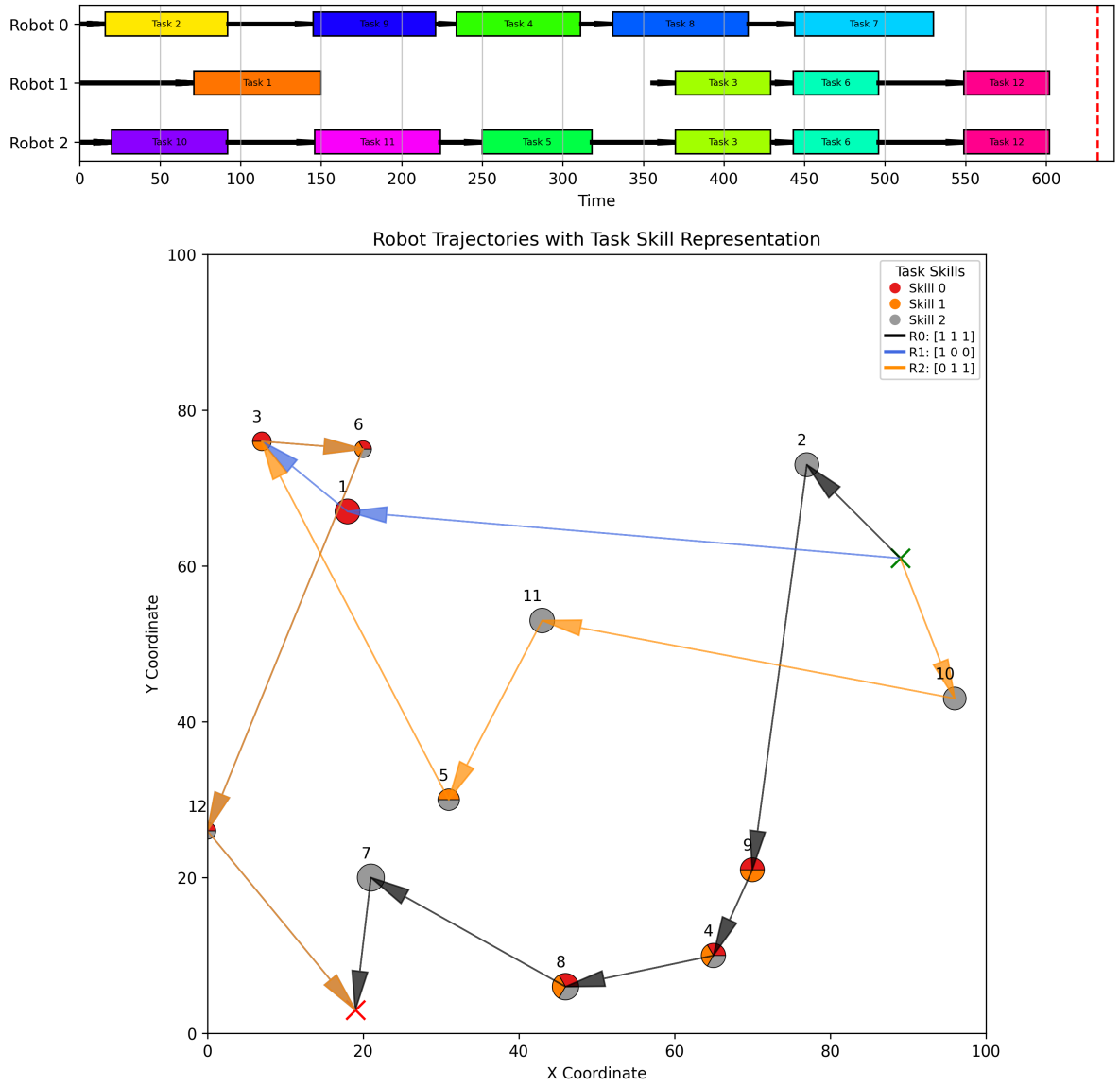

Fig. 8.1. Solution example on randomized problem instance with 12 tasks, 3 robots, and 3 distinct skills in the simulator. Colored circles represent tasks, where the circle size corresponds to execution time and the color indicates required skills. All robots are initialized at the start location and have to finish at the end location. Top: Schedule of the 3 robots over time. Black arrows represent travel times, and the colored boxes represent task execution. Bottom: Corresponding trajectories of the 3 robots in grey/blue/orange. When two robots move together, the arrow has a mixed color. For example, between tasks $T3$ and $T6$, robots $R1$ and $R2$ form a coalition.

9 Conclusion

9.1. Summary and Contributions

In this work, we proposed Sadcher - an IL framework to address real-time task assignment for heterogeneous multi-robot teams, incorporating dynamic coalition formation and precedence constraints. The combination of reward prediction and relaxed bipartite matching yields strong performance with feasibility guarantees. (Sample-)Sadcher outperforms RL-based and heuristic baselines in makespan across small to medium-sized robot teams and a wide range of task counts. One contributor to this performance is the model's emergent understanding of precedence constraints. It learns to assign predecessor tasks early, increasing the availability of successor tasks and enabling more efficient future assignments. Sadcher scales better with task number than with robot team size. An intuitive explanation is that increasing the number of tasks while keeping the number of robots fixed is similar to solving multiple smaller subproblems sequentially, where local scheduling rules learned during training remain effective. In cases with more robots, the increased coordination complexity introduces new dynamics - larger teams require different local scheduling strategies and decision logic that diverge from the training distribution. Sadcher can generate assignments in real-time across all tested problem sizes, but the sampling variant S-Sadcher is only real-time for smaller problems, as the bipartite matching step introduces computational overhead, particularly for larger problems. Additionally, Sadcher relies on a large dataset of expert demonstrations for training, which is computationally expensive to generate.

These findings motivated the use of RL to fine-tune our model for improved performance on large-scale problem instances. Unlike IL, RL does not require ground-truth schedules, which are infeasible to compute for medium and large problems. Despite extensive experimentation with both continuous and discrete PPO-based formulations, RL did not yield improvements over the IL baseline. In the continuous setup, RL preserves feasibility guarantees through bipartite matching, but learning is hindered by a high-dimensional action space and the indirect reward-action relationship, as the bipartite matching operates as a black-box component of the environment. The policy's output only influences assignments indirectly via a reward matrix, making effective gradient updates difficult. In the discrete formulation, the policy directly controls assignments and is faster to train and infer, but it lacks feasibility guarantees and occasionally produces invalid schedules, particularly in early training. Even after fine-tuning, performance remains below that of the IL baseline. While RL holds potential - especially due to its independence from ground-truth labels - realizing this potential might require rethinking the model architecture and reward structure.

We also detailed how to generate a dataset of small-scale optimally-solved problem instances for this complex use case and released our dataset of 250,000 of these solutions. The dataset can serve as expert demonstrations for IL or as an optimal baseline to benchmark against.

9.2. Future Research

Future work could extend our framework with more real-life constraints to incorporate battery budgets, deadlines, or time windows. Another improvement could be to replace the current binary capability modeling and instead incorporate task execution efficiency, where robots vary in how quickly they can perform certain tasks. This would allow the framework to better reflect heterogeneous skill levels and enable more nuanced coalition formation.

Our approach, like most related work, requires structured input in the form of predefined robot and task graphs. Bridging the gap between perception and planning systems through integrated task generation, decomposition, and scheduling would improve the autonomy of robot teams in real-world environments by decreasing reliance on manually specifying tasks.

The field would profit from a standardized benchmark dataset. This dataset should cover a wide range of problem sizes, incorporating both heterogeneous and homogeneous robots, coalition formation scenarios, and various temporal constraints. As a first step toward this goal, we released a dataset of 250,000 optimal schedules focused on small-scale, complex MRTA settings. Future research could build on this foundation by including simpler (e.g., homogeneous robots) scenarios in the same dataset format, which would allow complex approaches to be benchmarked against simpler, existing techniques on the use cases the simpler methods can handle. The dataset should also include larger problem scenarios (e.g., more tasks/robots) and solutions optimized for different objective functions, as well as standardized train-test splits.

References

[1] Chayan Sarkar, Himadri Sekhar Paul, and Arindam Pal. “A Scalable Multi-Robot Task Allocation Algorithm”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 2577-087X. May 2018, pp. 5022–5027. DOI: 10.1109/ICRA.2018.8460886. URL: https://ieeexplore.ieee.org/document/8460886 (visited on 10/20/2024).

[2] Yara Rizk, Mariette Awad, and Edward W. Tunstel. “Cooperative Heterogeneous Multi-Robot Systems: A Survey”. en. In: *ACM Computing Surveys* 52.2 (Mar. 2020), pp. 1–31. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3303848. URL: https://dl.acm.org/doi/10.1145/3303848 (visited on 10/17/2024).

[3] Haris Aziz et al. *Task Allocation using a Team of Robots*. en. arXiv:2207.09650 [cs]. July 2022. URL: http://arxiv.org/abs/2207.09650 (visited on 10/14/2024).

[4] A. Farinelli, L. Iocchi, and D. Nardi. “Multirobot systems: a classification focused on coordination”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 34.5 (Oct. 2004). Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), pp. 2015–2028. ISSN: 1941-0492. DOI: 10.1109/TSMCB.2004.832155. URL: https://ieeexplore.ieee.org/document/1335496 (visited on 10/25/2024).

[5] Matteo Bettini, Ajay Shankar, and Amanda Prorok. *Heterogeneous Multi-Robot Reinforcement Learning*. en. arXiv:2301.07137 [cs]. Jan. 2023. URL: http://arxiv.org/abs/2301.07137 (visited on 10/15/2024).

[6] Chuang Huang, Hao Zhang, and Zhuping Wang. “Task Allocation of Multi-robot Coalition Formation”. en. In: *Proceedings of 2021 5th Chinese Conference on Swarm Intelligence and Cooperative Control*. Ed. by Zhang Ren, Mengyi Wang, and Yongzhao Hua. Singapore: Springer Nature, 2023, pp. 221–230. ISBN: 978-981-19399-8-3. DOI: 10.1007/978-981-19-3998-3_22.

[7] Weiheng Dai, Aditya Bidwai, and Guillaume Sartoretti. “Dynamic Coalition Formation and Routing for Multirobot Task Allocation via Reinforcement Learning”. en. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. Yokohama, Japan: IEEE, May 2024, pp. 16567–16573. ISBN: 9798350384574. DOI: 10.1109/ICRA57147.2024.10611244. URL: https://ieeexplore.ieee.org/document/10611244/ (visited on 10/17/2024).

[8] G. Ayorkor Korsah, Anthony Stentz, and M. Bernardine Dias. “A comprehensive taxonomy for multi-robot task allocation”. en. In: *The International Journal of Robotics Research* 32.12 (Oct. 2013), pp. 1495–1512. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364913496484. URL: https://journals.sagepub.com/doi/10.1177/0278364913496484 (visited on 10/16/2024).

[9] Liwang Zhang et al. “Task Allocation in Heterogeneous Multi-Robot Systems Based on Preference-Driven Hedonic Game”. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. May 2024, pp. 8967–8972. DOI: 10.1109/ICRA57147.2024.10611476. URL: https://ieeexplore.ieee.org/document/10611476 (visited on 11/29/2024).

[10] Walker Gosrich et al. *Multi-Robot Coordination and Cooperation with Task Precedence Relationships*. en. arXiv:2209.14417 [cs]. May 2023. DOI: 10.48550/arXiv.2209.14417. URL: http://arxiv.org/abs/2209.14417 (visited on 12/10/2024).

[11] Matthew C. Gombolay, Ronald J. Wilcox, and Julie A. Shah. “Fast Scheduling of Robot Teams Performing Tasks With Temporospatial Constraints”. In: *IEEE Transactions on Robotics* 34.1 (Feb. 2018). Conference Name: IEEE Transactions on Robotics, pp. 220–239. ISSN: 1941-0468. DOI: 10.1109/TRO.2018.2795034. URL: https://ieeexplore.ieee.org/document/8279546 (visited on 11/21/2024).

[12] Ishaq Ansari et al. “CoLoSSI: Multi-Robot Task Allocation in Spatially-Distributed and Communication Restricted Environments”. In: *IEEE Access* 12 (2024). Conference Name: IEEE Access, pp. 132838–132855. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3434645. URL: https://ieeexplore.ieee.org/document/10613777/?arnumber=10613777 (visited on 11/01/2024).

[13] Hamza Chakraa et al. “Optimization techniques for Multi-Robot Task Allocation problems: Review on the state-of-the-art”. en. In: *Robotics and Autonomous Systems* 168 (Oct. 2023), p. 104492.

ISSN: 09218890. DOI: 10.1016/j.robot.2023.104492. URL: https://linkinghub.elsevier.com/retrieve/pii/S0921889023001318 (visited on 10/15/2024).

[14] Esther Bischoff et al. *Multi-Robot Task Allocation and Scheduling Considering Cooperative Tasks and Precedence Constraints*. en. arXiv:2005.03902 [eess]. May 2020. URL: http://arxiv.org/abs/2005.03902 (visited on 10/24/2024).

[15] Ragesh K. Ramachandran, James A. Preiss, and Gaurav S. Sukhatme. "Resilience by Reconfiguration: Exploiting Heterogeneity in Robot Teams". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. ISSN: 2153-0866. Nov. 2019, pp. 6518–6525. DOI: 10.1109/IROS40897.2019.8968611. URL: https://ieeexplore.ieee.org/document/8968611 (visited on 12/10/2024).

[16] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. "Multi-robot Task Allocation: A Review of the State-of-the-Art". en. In: *Cooperative Robots and Sensor Networks 2015*. Ed. by Anis Koubâa and J.Ramiro Martínez-de Dios. Cham: Springer International Publishing, 2015, pp. 31–51. ISBN: 978-3-319-18299-5. DOI: 10.1007/978-3-319-18299-5_2. URL: https://doi.org/10.1007/978-3-319-18299-5%5C_2 (visited on 10/15/2024).

[17] Félix Quinton, Christophe Grand, and Charles Lesire. "Market Approaches to the Multi-Robot Task Allocation Problem: a Survey". en. In: *Journal of Intelligent & Robotic Systems* 107.2 (Feb. 2023), p. 29. ISSN: 1573-0409. DOI: 10.1007/s10846-022-01803-0. URL: https://doi.org/10.1007/s10846-022-01803-0 (visited on 11/11/2024).

[18] William Babincsak, Ashay Aswale, and Carlo Pinciroli. "Ant Colony Optimization for Heterogeneous Coalition Formation and Scheduling with Multi-Skilled Robots". In: *2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. Dec. 2023, pp. 121–127. DOI: 10.1109/MRS60187.2023.10416771. URL: https://ieeexplore.ieee.org/document/10416771 (visited on 10/17/2024).

[19] Bo Fu et al. *Robust Task Scheduling for Heterogeneous Robot Teams under Capability Uncertainty*. June 2021. URL: https://arxiv.org/pdf/2106.12111 (visited on 11/12/2024).

[20] Pranab Muhuri and Amit Rauniyar. "Immigrants Based Adaptive Genetic Algorithms for Task Allocation in Multi-Robot Systems". In: *International Journal of Computational Intelligence and Applications* 16 (Dec. 2017), p. 1750025. DOI: 10.1142/S1469026817500250.

[21] Maria Gini. "Multi-Robot Allocation of Tasks with Temporal and Ordering Constraints". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 31.1 (Feb. 2017). ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v31i1.11145. URL: https://ojs.aaai.org/index.php/AAAI/article/view/11145 (visited on 11/28/2024).

[22] Williard Joshua Jose and Hao Zhang. "Learning for Dynamic Subteaming and Voluntary Waiting in Heterogeneous Multi-Robot Collaborative Scheduling". en. In: *IEEE International Conference on Robotics and Automation (ICRA)* (2024).

[23] Brian P. Gerkey and Maja J. Matarić. "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems". en. In: *The International Journal of Robotics Research* 23.9 (Sept. 2004), pp. 939–954. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364904045564. URL: https://journals.sagepub.com/doi/10.1177/0278364904045564 (visited on 10/15/2024).

[24] Ashay Aswale and Carlo Pinciroli. *Heterogeneous Coalition Formation and Scheduling with Multi-Skilled Robots*. en. arXiv:2306.11936 [cs]. June 2023. URL: http://arxiv.org/abs/2306.11936 (visited on 08/21/2024).

[25] Ishaq Ansari et al. "Cooperative and load-balancing auctions for heterogeneous multi-robot teams dealing with spatial and non-atomic tasks". In: *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. ISSN: 2475-8426. Nov. 2020, pp. 213–220. DOI: 10.1109/SSRR50563.2020.9292605. URL: https://ieeexplore.ieee.org/abstract/document/9292605 (visited on 11/01/2024).

[26] Muhammad Irfan and Adil Farooq. "Auction-based task allocation scheme for dynamic coalition formations in limited robotic swarms with heterogeneous capabilities". In: *2016 International Conference on Intelligent Systems Engineering (ICISE)*. Jan. 2016, pp. 210–215. DOI: 10.1109/INSE.2016.7475122. URL: https://ieeexplore.ieee.org/document/7475122 (visited on 10/17/2024).

[27] Hamza Chakraa et al. "A Centralized Task Allocation Algorithm for a Multi-Robot Inspection Mission With Sensing Specifications". In: *IEEE Access* 11 (2023). Conference Name: IEEE Access, pp. 99935–99949. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3315130. URL: https://ieeexplore.ieee.org/abstract/document/10250421 (visited on 10/15/2024).

[28] Muhammad Usman Arif. “Robot coalition formation against time-extended multi-robot tasks”. In: *International Journal of Intelligent Unmanned Systems* 10.4 (June 2021). Publisher: Emerald Publishing Limited, pp. 468–481. ISSN: 2049-6427. DOI: 10.1108/IJIUS-12-2020-0070. URL: https://doi.org/10.1108/IJIUS-12-2020-0070 (visited on 10/17/2024).

[29] Xiao-Fang Liu et al. “Strength Learning Particle Swarm Optimization for Multiobjective Multirobot Task Scheduling”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 53.7 (July 2023). Conference Name: IEEE Transactions on Systems, Man, and Cybernetics: Systems, pp. 4052–4063. ISSN: 2168-2232. DOI: 10.1109/TSMC.2023.3239953. URL: https://ieeexplore.ieee.org/document/10041025/?arnumber=10041025 (visited on 10/24/2024).

[30] Amanda Prorok et al. *The Holy Grail of Multi-Robot Planning: Learning to Generate Online-Scalable Solutions from Offline-Optimal Experts*. arXiv:2107.12254. July 2021. URL: http://arxiv.org/abs/2107.12254 (visited on 10/15/2024).

[31] Zheyuan Wang, Chen Liu, and Matthew Gombolay. “Heterogeneous graph attention networks for scalable multi-robot scheduling with temporospatial constraints”. en. In: *Autonomous Robots* 46.1 (Jan. 2022), pp. 249–268. ISSN: 0929-5593, 1573-7527. DOI: 10.1007/s10514-021-09997-2. URL: https://link.springer.com/10.1007/s10514-021-09997-2 (visited on 08/13/2024).

[32] Steve Paul, Payam Ghassemi, and Souma Chowdhury. *Learning Scalable Policies over Graphs for Multi-Robot Task Allocation using Capsule Attention Networks*. en. arXiv:2205.03321 [cs]. May 2022. URL: http://arxiv.org/abs/2205.03321 (visited on 10/14/2024).

[33] Batuhan Altundas et al. “Learning Coordination Policies over Heterogeneous Graphs for Human-Robot Teams via Recurrent Neural Schedule Propagation”. en. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. arXiv:2301.13279 [cs]. Oct. 2022, pp. 11679–11686. DOI: 10.1109/IROS47612.2022.9981748. URL: http://arxiv.org/abs/2301.13279 (visited on 10/14/2024).

[34] Fuqin Deng et al. “A Learning Approach to Multi-robot Task Allocation with Priority Constraints and Uncertainty”. In: *2022 IEEE International Conference on Industrial Technology (ICIT)*. Aug. 2022, pp. 1–8. DOI: 10.1109/ICIT48603.2022.10002805. URL: https://ieeexplore.ieee.org/abstract/document/10002805 (visited on 10/14/2024).

[35] Weiheng Dai et al. “Heterogeneous Multi-robot Task Allocation and Scheduling via Reinforcement Learning”. en. In: *IEEE Robotics and Automation Letters* 10.3 (Mar. 2025), pp. 2654–2661. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2025.3534682. URL: https://ieeexplore.ieee.org/document/10854527/ (visited on 04/20/2025).

[36] Peng Gao et al. “Collaborative Scheduling with Adaptation to Failure for Heterogeneous Robot Teams”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. May 2023, pp. 1414–1420. DOI: 10.1109/ICRA48891.2023.10161502. URL: https://ieeexplore.ieee.org/document/10161502/?arnumber=10161502 (visited on 10/14/2024).

[37] Paola Ardón et al. *Affordances in Robotic Tasks – A Survey*. en. arXiv:2004.07400 [cs]. Apr. 2020. URL: http://arxiv.org/abs/2004.07400 (visited on 11/12/2024).

[38] Petar Veličković et al. *Graph Attention Networks*. en. arXiv:1710.10903 [stat]. Feb. 2018. DOI: 10.48550/arXiv.1710.10903. URL: http://arxiv.org/abs/1710.10903 (visited on 04/29/2025).

[39] Ashish Vaswani et al. *Attention Is All You Need*. arXiv:1706.03762 [cs]. June 2017. DOI: 10.48550/arXiv.1706.03762. URL: http://arxiv.org/abs/1706.03762 (visited on 04/29/2025).

[40] Bing Xu et al. *Empirical Evaluation of Rectified Activations in Convolutional Network*. en. arXiv:1505.00853 [cs]. Nov. 2015. DOI: 10.48550/arXiv.1505.00853. URL: http://arxiv.org/abs/1505.00853 (visited on 04/29/2025).

[41] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980 [cs]. Jan. 2017. DOI: 10.48550/arXiv.1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 05/01/2025).

[42] John Schulman et al. *Proximal Policy Optimization Algorithms*. arXiv:1707.06347 [cs]. Aug. 2017. DOI: 10.48550/arXiv.1707.06347. URL: http://arxiv.org/abs/1707.06347 (visited on 05/15/2025).

[43] Antonio Serrano-Muñoz et al. “skrl: Modular and Flexible Library for Reinforcement Learning”. In: *Journal of Machine Learning Research* 24.254 (2023), pp. 1–9. URL: http://jmlr.org/papers/v24/23-0112.html.

[44] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. en. arXiv:1506.02438 [cs]. Oct. 2018. DOI: 10.48550/arXiv.1506.02438. URL: http://arxiv.org/abs/1506.02438 (visited on 05/19/2025).