Integrated Task and Path Planning for Collaborative Multi-Robot Systems

Aman Aryan*
Qualcomm, India
aman.aryan0@gmail.com

Manan Modi*
Jupiter Money, India
modimanann@gmail.com

Indranil Saha
IIT Kanpur, India
isaha@cse.iitk.ac.in

Rupak Majumdar
MPI-SWS, Germany
rupak@mpi-sws.org

Swarup Mohalik
Ericson Research, India
swarup.kumar.mohalik@ericsson.com

Abstract

We propose a generic multi-robot planning mechanism that combines an optimal task planner and an optimal path planner to provide a scalable solution for complex multi-robot planning problems. The integrated planner, through the interaction of the task planner and the path planner, produces optimal collision-free trajectories for the robots. We illustrate our general algorithm on an object pick-and-drop planning problem where a group of robots is entrusted with moving objects from one location to another in the workspace. We solve the task planning problem by reducing it into an SMT solving problem and employing the highly advanced *SMT solver Z3* to solve it. To generate collision-free movement of the robots, we extend the state-of-the-art algorithm *Conflict Based Search with Precedence Constraints* with several domain-specific constraints. We evaluate our integrated task and path planner extensively on various instances of the object pick-and-drop planning problem and compare its performance with a state-of-the-art multi-robot classical planner. Experimental results demonstrate that our planning mechanism can deal with complex planning problems and outperforms a state-of-the-art classical planner both in terms of computation time and the quality of the generated plan.

CCS Concepts

• **Computer systems organization → Embedded systems**; *Redundancy*; Robotics.

Keywords

Multi-robot systems, task planning, motion planning, SMT Solver

ACM Reference Format:
Aman Aryan*, Manan Modi*, Indranil Saha, Rupak Majumdar, and Swarup Mohalik. 2025. Integrated Task and Path Planning for Collaborative Multi-Robot Systems. In *ACM/IEEE 16th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2025) (ICCPS '25), May 6–9, 2025, Irvine, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3716550.3722028

* This research was carried out when the author was with IIT Kanpur, India.

ICCPS '25, Irvine, CA, USA

ACM ISBN 979-8-4007-1498-6/2025/05
https://doi.org/10.1145/3716550.3722028

1 Introduction

A major component of the software controlling a robotic system is a *planner* that guides the robots to safely move through their workspace and perform the designated tasks in a proper way. A planner for an application involving mobile robots need to have two components: a *task planner* that decides which tasks should be performed by which robots and in what order, and a *path planner* that provides the collision-free trajectories to be followed by the robots to reach the designated locations to perform the tasks. The task planning and the path planning problems cannot be addressed entirely independently as the assignment of a task to a robot is directly related to the amount of effort the robot needs to invest in reaching the task locations.

Consider a multi-robot application where a group of mobile robots is entrusted with the responsibility of delivering objects from one location to another in a workspace. The task assignment to the robots depends on the time required to traverse the distance between the initial locations of the robots and various task locations and the distance between the task locations when a robot has to perform multiple tasks. The traverse time between different locations depends on the collision-free optimal trajectories of the robots, which can only be obtained from a multi-robot path planner.

Two different approaches are, in general, employed to solve a multi-robot planning problem offline for a static environment. In the first approach, the multi-robot task assignment and the path planning problems are formulated and solved as a monolithic problem (e.g., [5, 14, 25]). In the second approach, the task assignment problem is solved based on a heuristic to measure the trajectory lengths approximately (e.g., [2, 8, 28]). As the task assignment is not carried out based on collision-free trajectories, a local collision avoidance strategy (e.g. [13]) is employed during the execution of the plan. The shortcoming of the first approach is that it either fails to provide a multi-robot trajectory with a guarantee on its optimality [5], or the algorithm that can produce an optimal plan takes a prohibitively large amount of time to compute the collision-free trajectories [14, 25]. The second approach can find a plan quickly, but the generated plans are guaranteed to be neither collision-free nor optimal.

To bridge this gap, we design a scalable algorithm to generate optimal collision-free trajectories for multi-robot systems. The proposed algorithm works as follows. It first estimates the lengths of the independent trajectories between all locations of interest through which a robot may need to move. Based on the estimated trajectory lengths, the task planner generates a task assignment

corresponding to optimal trajectories for the robots based on the estimated length of the trajectories between any two locations. The outcome of the task assignment is a sequence of locations to be visited by all the robots. In the second step, we generate collision-free trajectories for the robots to reach their designated locations in sequence by means of an optimal multi-robot path planner. If the cost of the trajectories obtained in this step is more than that of the trajectories obtained during task assignment, we look for another same-cost or a suboptimal task assignment for which the cost of the collision-free trajectories obtained by solving the multi-robot path planning problem may be better than the collision-free trajectories obtained in the previous step. In this way, we alternate between the task planner and the path planner until we find a task assignment with optimal-cost collision-free trajectories.

We illustrate our general algorithm on an offline multi-agent pick-and-drop planning problem where a group of robots move objects from one location to another in a workspace. Our problem statement is similar to [18] except that we have defined a designated base location for robots to return after finishing the tasks. We transform the task-planning problem into an SMT-solving problem that incorporates many application-specific operational constraints and solve it using the Z3 [7] solver. Additionally, we employ the existing optimal multi-robot path planning algorithm MLA*-CBS-PC [34] to accommodate the sequential goal locations for each robot, thereby serving as the optimal path planner.

We extensively evaluate our algorithm on various instances of the object pick-and-drop planning problem for warehouse management and disaster response applications and compare the performance of our planner with the state-of-the-art multi-robot classical planner ENHSP [1]. Experimental results demonstrate that our planning mechanism can deal with complex planning problems and outperforms the state-of-the-art classical planner in terms of computation time and quality of the generated plan.

In summary, we make the following contributions:

- We provide a general multi-robot planning algorithm that induces an interaction between the task planner and the path planner to generate optimal collision-free trajectories for the robots to complete the mission successfully (Section 3).
- We provide an SMT-based task planner for object pick-and-drop applications. Our task planner is capable of incorporating many application-specific operational constraints. Furthermore, we adapt the state-of-the-art graph-based multi-robot path planner MLA*-CBS-PC [34] to deal with a sequence of goal locations for each robot using plans generated from our task planner (Section 4).
- We demonstrate the overall algorithm for the multi-agent pickup and delivery application on warehouse maps as well as randomly generated maps (representing a disaster-stricken area) and compare it to the state-of-the-art classical planner ENHSP (Section 5).

2 Problem

In this section, we define our problem formally and illustrate it with an example.

2.1 Preliminaries

2.1.1 Workspace. The workspace, denoted by $\mathcal{W}$, is represented as a 2-D rectangular grid. We assume that the robots, as well as the task objects, occupy one grid block each at any time instance. Obstacles may occupy some of these grid blocks and thus cannot be used by the robots, tasks, or movements. Formally, the workspace is represented by a tuple $\langle L_X, L_Y, \Omega \rangle$, where L_X and L_Y denote the length and the width of the workspace, and Ω denotes the set of grid blocks that are occupied by obstacles.

2.1.2 Robots. The set of robots is denoted by $\mathcal{R}$. Each robot $r_i \in \mathcal{R}$ is defined as a tuple $\langle s_i, \Gamma_i, \Lambda_i, attributes_i \rangle$. The symbol s_i denotes the start location of robot r_i. The symbols Γ_i and Λ_i denote the set of *motion primitives* and *action primitives* for robot r_i, respectively. To keep the exposition simple, we assume that each robot has five basic motion primitives: move up, move down, move left, move right, and stay. However, our methodology seamlessly applies to any complex set of motion primitives for a robot. The action primitives for a robot are application-specific. For example, for a pick-and-drop application, the robot has action primitives for *picking up* and *dropping off* an object. We assume that all of these primitives take one time step regardless of the robot's direction. Moreover, the motion and action primitives are deterministic, i.e., the application of a primitive to a robot in a state moves the robot to a unique next state. We denote by $attributes_i$ a set of attributes of robot r_i that may be required depending upon the nature of the problem. For example, in a pick-and-drop example, an attribute for a robot could be the number of objects or the total amount of weight the robot can carry at once.

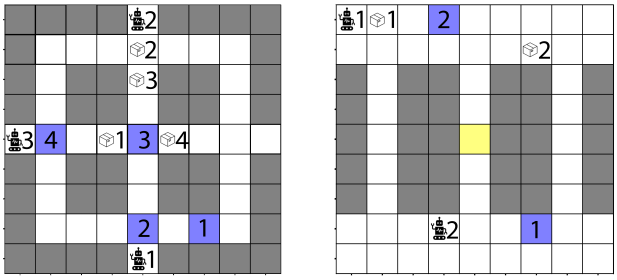
2.1.3 Tasks. The set of tasks associated with a problem is denoted by $\mathcal{T}$. A task $t_i \in \mathcal{T}$ is defined as a tuple $\langle L_i, attributes_i \rangle$. Here, L_i is a sequence of locations that need to be visited by a robot in the same order to complete the task. We denote by $attributes_i$ a set of attributes of the task that may be required for planning depending upon the nature of the problem. For example, a task t_i may be associated with a deadline d_i; in that case, the last location in L_i must be visited before d_i.

2.1.4 Plan and Trajectory. We capture the behaviour of a robot in the workspace as a sequence of states. The state of robot r_i at time step t is denoted by $\sigma_i(t)$. Given a state σ and a motion or action primitive ν, the robot's next state σ' is given by $\texttt{next}(\sigma, \nu)$.

Definition 1 (Plan). *The plan for a robot r_i is the sequence of motion and action primitives executed by the robot.*

Definition 2 (Trajectory). *For robot r_i with plan $\nu_i = (\nu_i(1), \nu_i(2), \ldots, \nu_i(T_i))$, the trajectory is given by $\sigma_i = (\sigma_i(0), \sigma_i(1), \ldots, \sigma_i(T_i))$, where $\sigma_i(0) = s_i$ and for all $i \in \{1, \ldots, T_i\}$, $\sigma_i(j) = \texttt{next}(\sigma_i(j-1), \nu_i(j))$. The symbol T_i denotes the length of the plan ν_i and the trajectory σ_i. The trajectory of the multi-robot system $\mathcal{R} = \{r_1, \ldots, r_n\}$ is denoted by $\Sigma = [\sigma_1, \sigma_2, \ldots, \sigma_n]$, where σ_i denotes the trajectory of robot r_i.*

2.1.5 Optimality Criteria for a Trajectory. The cost of executing a trajectory $\sigma_i = (\sigma_i(0), \sigma_i(1), \ldots, \sigma_i(T_i))$ is equal to its length T_i. Now, the quality of a multi-robot trajectory Σ is captured by one of the following two attributes.



(a) (b)

Figure 1: Examples of workspaces showing warehouse scenarios: (a) without Intermediate Location, (b) with an Intermediate Location.

Definition 3 (Makespan). *The makespan of the trajectories* $\Sigma = [\sigma_1, \sigma_2, \ldots, \sigma_n]$ *is given by* $C = \max_i T_i$.

Definition 4 (Total cost). *The total cost of the trajectories* $\Sigma = [\sigma_1, \sigma_2, \ldots, \sigma_n]$ *is given by* $C = \sum_i T_i$.

Note that the makespan and total cost are equal for a single robot system. We will use the terms *plan* and *trajectory* interchangeably to denote the solution from our algorithm.

2.2 Problem Definition

Here, we provide the formal definition of the problem.

Definition 5 (Problem). *Given a workspace* $\mathcal{W}$*, a set of tasks* $\mathcal{T}$*, and a set of robots* $\mathcal{R}$*, find optimal makespan or optimal total cost collision-free trajectories* Σ *for the robots such that all tasks are completed while also ensuring that the robots return to their initial positions.*

Example. Consider the workspaces shown in Figure 1. They represent typical warehouse scenarios. Boxes in the images denote the pickup locations for these objects. The blue grid locations denote their drop locations. The grey-coloured grid blocks are occupied by obstacles and must be avoided. In the figure, the robots are shown in their initial locations. The robots can carry multiple objects at a time. To pick up an object, a robot needs to be in the grid block where the object is placed. The same is true for dropping an object. The problem is to find the task assignment to the robots to decide which robot should carry which object to its goal location and the collision-free trajectories for the robots to carry out their tasks successfully.

In Figure 1a, there are 4 objects that need to be moved to some specific goal locations. Three robots r_1, r_2, and r_3 have to move the four objects from their current locations to their goal locations.

In Figure 1b, the yellow block denotes the intermediate drop block. A robot can drop an object on the yellow block, and the object can be picked up from there by another robot. Thus, having an intermediate block allows the robots to collaborate on delivering a specific object. In the scenario presented in Figure 1b, let us attempt to find the plan with optimal makespan. The collision-free plan without the intermediate drop would be r_1 completing task t_2 and returning to its base location in 16 steps and r_2 completing t_1 and returning to its base location in 26 steps. So the makespan of this plan becomes 26. The collision-free trajectory for the two robots r_1 and r_2 are shown in Figure 2.

time	r_1	r_2
0	$(Start, (0, 0))$	$(Start, (7, 3))$
1	$(Move, (1, 0))$	$(Move, (7, 2))$
2	$(Move, (1, 1))$	$(Move, (7, 1))$
3	$(Move, (1, 2))$	$(Move, (6, 1))$
4	$(Move, (1, 3))$	$(Move, (5, 1))$
5	$(Move, (1, 4))$	$(Move, (4, 1))$
6	$(Move, (1, 5))$	$(Move, (3, 1))$
7	$(Move, (1, 6))$	$(Move, (2, 1))$
8	$(Pick_2, (1, 6))$	$(Move, (1, 1))$
9	$(Move, (0, 6))$	$(Move, (0, 1))$
10	$(Move, (0, 5))$	$(Pick_1, (0, 1))$
11	$(Move, (0, 4))$	$(Move, (1, 1))$
12	$(Move, (0, 3))$	$(Move, (2, 1))$
13	$(Drop_2, (0, 3))$	$(Move, (3, 1))$
14	$(Move, (0, 2))$	$(Move, (4, 1))$
15	$(Move, (0, 1))$	$(Move, (5, 1))$
16	$(Return, (0, 0))$	$(Move, (6, 1))$
17	$(---, (0, 0))$	$(Move, (7, 1))$
18	$(---, (0, 0))$	$(Move, (7, 2))$
19	$(---, (0, 0))$	$(Move, (7, 3))$
20	$(---, (0, 0))$	$(Move, (7, 4))$
21	$(---, (0, 0))$	$(Move, (7, 5))$
22	$(---, (0, 0))$	$(Move, (7, 6))$
23	$(---, (0, 0))$	$(Drop_1, (7, 6))$
24	$(---, (0, 0))$	$(Move, (7, 5))$
25	$(---, (0, 0))$	$(Move, (7, 4))$
26	$(---, (0, 0))$	$(Return, (7, 3))$

Figure 2: Trajectories of the two robots for the problem shown in Figure 1(b) without the intermediate block.

If we allow the robots to use the intermediate block for object transfer, r_1 can pick up t_1 and drop it to the intermediate block; then it can continue to pick up and drop t_2 and return to its base location in 24 steps. However, this reduces the time taken by r_2 to process t_1. Now, r_2 can pick up t_1 from the intermediate location, drop it to its drop location, and return to its base station. Execution of this plan takes 21 steps to complete, thus making the overall makespan 24. Since we optimize the makespan, the total cost metric may increase. In this scenario, the total cost increases from 42 to 45. The trajectories for both robots are shown in Figure 3.

Thus, intermediate locations help in finding a better plan for our optimization criteria, and our goal would be to design a planner that can efficiently exploit the availability of such opportunities.

3 Integrated Task and Path Planning Algorithm

In this section, we provide an algorithm to solve the problem described in Section 2. One could reduce the problem to an Integer-Linear Programming or an SMT-solving problem and generate a solution for the task assignment as well as the trajectories for the robots. However, this monolithic approach rarely scales with the

time	r_1	r_2
0	$(Start, (0, 0))$	$(Start, (7, 3))$
1	$(Move, (0, 1))$	$(Move, (7, 4))$
2	$(Pick_1, (0, 1))$	$(Move, (6, 4))$
3	$(Move, (0, 2))$	$(Move, (5, 4))$
4	$(Move, (1, 2))$	$(Move, (4, 4))$
5	$(Move, (1, 3))$	$(Move, (4, 4))$
6	$(Move, (1, 4))$	$(Move, (4, 4))$
7	$(Move, (2, 4))$	$(Move, (4, 4))$
8	$(Move, (3, 4))$	$(Move, (4, 4))$
9	$(Move, (4, 4))$	$(Move, (5, 4))$
10	$(InterDrop_1, (4, 4))$	$(Move, (5, 4))$
11	$(Move, (3, 4))$	$(Move, (4, 4))$
12	$(Move, (2, 4))$	$(InterPick_1, (4, 4))$
13	$(Move, (1, 4))$	$(Move, (5, 4))$
14	$(Move, (1, 5))$	$(Move, (6, 4))$
15	$(Move, (1, 6))$	$(Move, (7, 4))$
16	$(Pick_2, (1, 6))$	$(Move, (7, 5))$
17	$(Move, (0, 6))$	$(Move, (7, 6))$
18	$(Move, (0, 5))$	$(Drop_1, (7, 6))$
19	$(Move, (0, 4))$	$(Move, (7, 5))$
20	$(Move, (0, 3))$	$(Move, (7, 4))$
21	$(Drop_2, (0, 3))$	$(Return, (7, 3))$
22	$(Move, (0, 2))$	$(---, (7, 3))$
23	$(Move, (0, 1))$	$(---, (7, 3))$
24	$(Return, (0, 0))$	$(---, (7, 3))$

Figure 3: Trajectories of the two robots for the problem shown in Figure 1(b) with the intermediate block.

number of robots, the number of tasks, and the size of the workspace. Instead, we embrace a decoupled approach where the task and the path planning problems are solved independently. However, through an interaction between the task and the path planner, we ensure that the finally generated plans satisfy the task completion requirement and that the corresponding paths are collision-free and optimal.

Our proposed methodology is shown in Algorithm 1. We advocate the use of an SMT solver to solve complex task assignment problems. The procedure TASK_PLANNER (lines 1-4) takes $\mathcal{W}, \mathcal{R}, \mathcal{T}$, a set $\mathcal{A}$ of forbidden task assignments, a lower bound l_b, and an upper bound u_b as inputs. It produces as output a task assignment $\mathcal{L} = [\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_{|\mathcal{R}|}]$, with minimum total cost or makespan within specified bounds. It returns $\emptyset$ if there does not exist a feasible task assignment within the bounds. Here, $\mathcal{L}_i$ denotes the sequence of locations that robot r_i must visit. In Section 4, we will present the details of the task planner with an example of a multi-robot pick-and-drop application.

The following procedure PATH_PLANNER (lines 5-8) takes the task assignment $\mathcal{L}$ produced by the TASK_PLANNER procedure and generates *optimal* and *collision-free* trajectories. The procedure also returns the trajectory's total cost or makespan depending upon the optimization criterion. In Section 4.3, we will present the details of the path planner.

The main procedure INTEGRATED_PLANNER (lines 9-29) induces an interaction between the task planner and the path planner to find the optimal collision-free trajectories for the robots. There could be several task assignments with the same cost. Thus, once a task assignment $\mathcal{L}$ is produced by the task planner, we need to ensure that the task planner does not generate the same task assignment again. We use the set $\mathcal{A}$ for this purpose. We keep on storing the task assignments with the same cost in $\mathcal{A}$ and provide it as the set of prohibited assignments while invoking the task planner with the same lower bound of the cost. This set is initialized as an empty set (line 10). We initialize *cur_task_cost* (denoting the cost of the current task assignment) as 0 and *opt_plan_cost* (denoting the minimum cost of the collision-free paths for any assignment) as ∞ (line 11) and repeat the procedure below until *cur_task_cost* is less than *opt_plan_cost* (lines 12-27). We invoke the TASK_PLANNER with the *cur_task_cost* as lower bound and *opt_plan_cost* as upper bound to get the best task assignment $\mathcal{L}$ with cost *task_cost* based on some heuristic cost of movements between important locations (line 13). If the task planner cannot produce a plan (returns $\emptyset$), we terminate the loop (lines 14-16). Otherwise, for this task assignment $\mathcal{L}$, we invoke the PATH_PLANNER, which outputs the collision-free trajectory with cost *plan_cost* (line 17). If we find that the new task assignment $\mathcal{L}$ has a higher cost compared to *cur_task_cost*, then we update *cur_task_cost* with *task_cost* and reset the exclusion list $\mathcal{A}$ (lines 18-21). Subsequently, we add this task assignment $\mathcal{L}$ to the set $\mathcal{A}$ (line 22). We update the *opt_plan* and *opt_plan_cost* if the current trajectory has a better cost (lines 23-26).

3.1 Illustrative Example

We illustrate the algorithm on the example introduced in Figure 1a in Section 2 with makespan as optimization criteria. Here, we use the A* search algorithm [11] to find a trajectory for a robot between two locations. In the below task assignments, pickup represents move and pickup. Similarly, the drop represents move and drop. Since there is no intermediate location, all pickups are the boxes' initial locations, and drops are their respective drop locations. The minimum makespan returned by the task planner is 18, and the corresponding task assignment is as follows:

r_1 : `pickup` $-$ 1, `drop` $-$ 1
r_2 : `pickup` $-$ 2, `pickup` $-$ 3, `drop` $-$ 3, `drop` $-$ 2
r_3 : `pickup` $-$ 4, `drop` $-$ 4

In the above task assignment, r_1 starts from grid location (8, 4), visits grid location (4, 3) to pick up object-1 and then visits grid location (7, 6) to drop object-1, and then finally returns to grid location (8, 4). The distances computed by the A* algorithm for these movements are 5, 6, and 3, respectively. Also, r_1 spends two units of time step to pick and drop the object, thus making the total time steps 16. Similarly, the costs for robots r_2 and r_3 are 18 and 12, respectively. Therefore, the effective makespan of the plan is 18. This heuristic cost is generated by calculating the costs individually without considering the robot-robot collisions. Using the task assignment, we compute a collision-free trajectory using the path planner. The cost of collision-free trajectories that the path planner returns is 19, 18, and 17, respectively. So, the overall makespan becomes 19. Since the estimated task assignment cost is 18 and the collision-free cost is 19, there may be some plans with a

Algorithm 1 Integrated Planner using Task and Path Planner

```
1: procedure TASK_PLANNER (W, R, T, A, l_b, u_b)
2:     // find optimal task assignments using a heuristic cost for movements.
3:     return ⟨L, task_cost⟩
4: end procedure
5: procedure PATH_PLANNER (W, R, L)
6:     // find the optimal collision-free trajectories for robots following the given task assignments in L.
7:     return ⟨plan, plan_cost⟩
8: end procedure
9: procedure INTEGRATED_PLANNER (W, R, T)
10:    A ← ∅; opt_plan ← ∅
11:    cur_task_cost ← 0; opt_plan_cost ← ∞
12:    while cur_task_cost < opt_plan_cost do
13:        ⟨L, task_cost⟩ ← TASK_PLANNER (W, R, T, A, cur_task_cost, opt_plan_cost)
14:        if L == ∅ then
15:            break
16:        end if
17:        ⟨plan, plan_cost⟩ ← PATH_PLANNER (W, R, L)
18:        if (cur_task_cost < task_cost) then
19:            cur_task_cost ← task_cost
20:            A ← ∅
21:        end if
22:        A ← A ∪ {L}
23:        if (plan_cost < opt_plan_cost) then
24:            opt_plan ← plan
25:            opt_plan_cost ← plan_cost
26:        end if
27:    end while
28:    return ⟨opt_plan, opt_plan_cost⟩
29: end procedure
```

cost of 18, resulting in a makespan less than 19. So, we continue to find more plans and obtain the next task assignment as follows:

r_1 : pickup − 1, drop − 1
r_2 : pickup − 2, drop − 2
r_3 : pickup − 4, pickup − 3 drop − 3 drop − 4

The makespan of the above task assignment is 18. The path planner returns a plan with a makespan of 19, the same as the previously found plan's makespan. We continue searching for task assignments. The third assignment that we obtain also has a makespan of 18. It is as follows:

r_1 : pickup − 1, drop − 1
r_2 : pickup − 2, pickup − 3, drop − 2, drop − 3
r_3 : pickup − 4, drop − 4

The above task assignment differs slightly from the first assignment, in which r_2 drops object-2 before dropping object-3. The estimated cost returned by the task planner for r_1, r_2, and r_3 is 16, 18, and 12, respectively. Executing the path planner with this task assignment returns a collision-free trajectory with costs of 18, 18, and 12, respectively, thus making the makespan 18. So, this collision-free trajectory becomes the minimum collision-free trajectory, and the minimum cost is updated to 18. As the collision-free cost is not greater than the estimated cost, we terminate the algorithm.

3.2 Theoretical Analysis

We formally prove that Algorithm 1 produces the optimal trajectories satisfying the task requirements.

Theorem 1 (Optimality). *There does not exist a task assignment for which the cost of the collision-free trajectories would be less than the cost of the trajectories returned by Algorithm 1.*

Proof. Let us assume that Algorithm 1 returns collision-free trajectories for the robots with cost C for a task assignment $\mathcal{L}$. The heuristic cost for the assignment is C_h. Now, let us assume that there exists a task assignment $\mathcal{L}'$ for which the cost of the collision-free trajectories is C' where $C' < C$, but this task assignment was not considered by Algorithm 1. The heuristic cost for the assignment $\mathcal{L}'$ is C'_h. As heuristic cost must always be a lower bound for the cost of the collision-free trajectories, $C_h \leq C$ and $C'_h \leq C'$. Then either (I) $C'_h < C_h$ or (II) $C_h \leq C'_h$.

Case I: In this case, $\mathcal{L}'$ must have been considered by the planner before $\mathcal{L}$ as the task planner returns the task assignment with the minimum possible heuristic cost.

Case II: As $C'_h \leq C'$ and $C' < C$, therefore $C'_h < C$. In this case, the planner must have considered $\mathcal{L}'$ after generating collision-free trajectories for $\mathcal{L}$ as $C'_h < C$ and $C_h \leq C'_h$. Our Integrated Planner explores all task assignments with heuristic costs less than C.

Thus, in both cases, our assumption that Algorithm 1 did not consider $\mathcal{L}'$ is wrong. Hence, if the heuristic cost considered in the task planning procedure gives a lower bound on cost and the Path Planner gives the minimum cost collision-free paths corresponding to the task assignment, then the integrated planner will always generate collision-free trajectories for the robots with optimal cost. □

Note: As the number of task assignments is finite for a well-formed MAPD instance, the optimality of Algorithm 1 establishes its *completeness* as well.

4 Application to Multi-Robot Pick-and-Drop Problem

In this section, we illustrate our planning mechanism for the object pick-and-drop application, as shown in Figure 1. As the tasks are pick-and-drop, L_i for each task t_i contains two entries: $L_i(0)$ denotes the pickup location and $L_i(1)$ represents the drop location.

4.1 Task Planning Algorithm

The general SMT-based task planning algorithm is shown in Algorithm 2. The `generate_smt_instance` function generates the SMT constraints for the task planner. We use the notion of *action-step* in our SMT formulation. In each action step, all robots can perform an action related to movement, pickup, or drop. In our constraints, we keep track of the time taken for each action step for each robot. There is no constraint on how long these actions can take here; we do not generate the final paths but rather just the task assignment. The time required for an action that requires a movement from location $\mathbf{x}$ to location $\mathbf{x}'$ is captured by $\texttt{dist}(\mathbf{x}, \mathbf{x}')$, as we assume

Algorithm 2 Task Planner

```
1: procedure TASK_PLANNER (W, R, T, A, l_b, u_b)
2:     S ← generate_smt_instance (W, R, T, A)
3:     if S.check() ≠ SAT then
4:         return ∅
5:     end if
6:     while (l_b ≤ u_b) do
7:         S' ← S
8:         mid ← (l_b + u_b)/2
9:         S ← S ∧ (cost ≥ l_b)
10:        S ← S ∧ (cost ≤ mid)
11:        if S.check() = SAT then
12:            u_b ← S.get(cost) − 1
13:        else
14:            l_b ← mid + 1
15:        end if
16:        S ← S'
17:    end while
18:    L ← S.get(task_assignment)
19:    cost ← S.get(cost)
20:    return ⟨L, cost⟩
21: end procedure
```

that a movement from one grid cell to another takes one unit of time. We compute $\texttt{dist}(\mathbf{x}, \mathbf{x}')$ using the A* search algorithm [11], which is guaranteed to be an under-approximation of the distance between $\mathbf{x}$ and $\mathbf{x}'$ while computing the collision-free trajectories for the robots. For a task assignment problem, the number of action steps is denoted by Z, which is the same for all the robots.

4.2 SMT Encodings Of Constraints

In this section, we describe the constraints in detail to capture two variants of the pick-and-drop problem.

4.2.1 Completing pick-and-drop tasks. Here, we present the SMT constraints to capture the basic object pick-and-drop problem as illustrated in Figure 1. We define LOC as a set of all the task's pickup and drop locations. Thus, $LOC = \bigcup_{t_m \in \mathcal{T}} \{L_m(0), L_m(1)\}$.

The following are the variables used to track the state of the system: $pos_{i,j}$ denotes the location of robot r_i after the j^{th} action-step. This location can be one of the locations from the sets LOC and s_i for all $j \geq 1$. We denote by $pos\_time_{i,j}$ the time step at which robot r_i is at location $pos_{i,j}$ in the j^{th} action-step. The symbol $action_{i,j}$ denotes on which task's object r_i will perform an action in the j^{th} action-step. The value of the variable can be either -1 if no action is performed or the task number. The symbol $loc_{i,j}$ denotes the location of task t_i in the j^{th} action-step. This location can be either $L_i(0)$ or $L_i(1)$, or it can be -1 in case the task object is being carried by some robot. The symbol $being\_carried_{i,j}$ denotes by which robot the object of task t_i is being carried in the j^{th} action-step. It is either the identifier of the robot if the task is in transition or -1 if it is steady.

Initial State. The initial state of the system is captured by the following constraints.

$$\forall r_i \in \mathcal{R},\ pos_{i,0} = s_i \ \wedge\ pos\_time_{i,0} = 0 \ \wedge\ action_{i,0} = -1$$
$$\forall t_i \in \mathcal{T},\ loc_{i,0} = L_i(0) \ \wedge\ being\_carried_{i,0} = -1. \tag{1}$$

Picking an object. A robot can go to $L_m(0)$ only if it picks up the object of task t_m from there. If robot r_i wants to pick up an object from one of the pickup locations in action step j, then the constraints formulation is as mentioned below.

$$pick(r_i, t_m, j) \equiv$$
$$loc_{m,j-1} = L_m(0) \tag{2a}$$
$$\wedge\ pos_{i,j} = L_m(0) \ \wedge being\_carried_{m,j} = i \tag{2b}$$
$$\wedge\ pos\_time_{i,j} = pos\_time_{i,j-1} + \texttt{dist}(pos_{i,j-1}, L_m(0)) + 1 \tag{2c}$$
$$\wedge\ loc_{m,j} = -1 \ \wedge action_{i,j} = m. \tag{2d}$$

Equation 2(a) captures that task t_m is at location $L_m(0)$ in the $j-1$ action-step. Equation 2(b) captures that robot r_i is at location $L_m(0)$ in action-step j and the object for task t_m is being carried by robot r_i in action-step j. Equation 2(c) captures the time taken by robot r_i while moving from its location in the previous action-step $pos_{i,j-1}$ to its location in the current action-step $L_m(0)$ and one unit of time for picking up the object associated with the task t_m by r_i. Equation 2(d) ensures that $loc_{m,j}$ is set to -1 as the object for task t_m is being carried by a robot now and sets $action_{i,j}$ as m to indicate the pickup of the object t_m by robot r_i in action-step j.

Dropping an object. A robot can go to one of the drop locations only if it drops an object there. If r_i wants to drop an object to one of the drop locations in action-step j, then the constraints formulation is as below.

$$drop(r_i, t_m, j) \equiv$$
$$being\_carried_{m,j-1} = i \tag{3a}$$
$$\wedge\ pos_{i,j} = L_m(1) \ \wedge being\_carried_{m,j} = -1 \tag{3b}$$
$$\wedge\ pos\_time_{i,j} = pos\_time_{i,j-1} + \texttt{dist}(pos_{i,j-1}, L_m(1)) + 1 \tag{3c}$$
$$\wedge\ loc_{m,j} = L_m(1) \ \wedge action_{i,j} = m. \tag{3d}$$

Equation 3(a) captures that task t_m must be carried by robot r_i in action-step $j-1$ to be able to drop it in action-step j. Equation 3(b) captures that robot r_i is at location $L_m(1)$ in action-step j and changes $being\_carried_{m,j}$ to -1 as the object will be dropped. Equation 3(c) captures the time taken by robot r_i while moving from its location in the previous action-step $pos_{i,j-1}$ to its location in the current action-step $L_m(1)$ and one unit of time to drop the task t_m by r_i. Equation 3(d) set $loc_{m,j}$ to indicate that the object for task t_m has been dropped at its final location in action-step j and $action_{i,j}$ to m to indicate dropping of the object for task t_m by robot r_i in action-step j.

Doing Nothing. A robot can also do nothing for one action step, which is captured as follows.

$$stay(r_i, j) \equiv pos_{i,j} = pos_{i,j-1} \ \wedge\ action_{i,j} = -1$$
$$\wedge\ pos\_time_{i,j} = pos\_time_{i,j-1}. \tag{4}$$

Returning the Base Station. A robot can also return to the base station from a drop location if it is no longer required to do more tasks.

$$return(r_i, j) \equiv$$
$$pos_{i,j} = s_i \ \wedge \ action_{i,j} = -1 \ \wedge \tag{5a}$$
$$pos\_time_{i,j} = pos\_time_{i,j-1} + \texttt{dist}(pos_{i,j-1}, s_i). \tag{5b}$$

Equation 5(a) captures that the robot r_i is at base station s_i at action-step j. Equation 5(b) captures the time taken by robot r_i while moving from its location in the previous action-step $pos_{i,j-1}$ to its base station in the current action-step.

Robots' All Possible Actions. Combining Equations (2) - (5), for each robot r_i for each possible action-step j, we get the constraint below:

$$\bigwedge_{r_i \in \mathcal{R}} \bigwedge_{j=1}^{Z} \Big(stay(r_i, j) \ \vee \bigvee_{k \in \{s_i\} \cup LOC} \Big((pos_{i,j-1} = k) \ \wedge$$
$$return(r_i, j) \bigvee_{t_m \in \mathcal{T}} \big(pick(r_i, t_m, j) \ \vee drop(r_i, t_m, j) \big) \Big) \Big). \tag{6}$$

Robot and Task-Object Movement Consistency. We add the constraints to enforce that the task objects move only when being carried by one of the robots.

$$\bigwedge_{t_m \in \mathcal{T}} \bigwedge_{j=1}^{Z} \Big(\bigwedge_{r_i \in \mathcal{R}} action_{i,j} \neq m \Big) \implies (loc_{m,j} = loc_{m,j-1}$$
$$\wedge\, being\_carried_{m,j} = being\_carried_{m,j-1}). \tag{7}$$

Equation 7 ensures that if no robot is performing an action on task t_m, then t_m's location and being carried status remain the same. Note that only picking up or dropping is classified as performing an action. A robot carrying a task's object does not mean that he is performing an action on that task.

Task Completion. The following constraint ensures that each task object is at its goal location in the last action step.

$$\bigwedge_{t_m \in \mathcal{T}} (loc_{m,Z} = L_m(1)). \tag{8}$$

The final set of constraints is obtained as the conjunction of constraints capturing the initial states and those in Equations (1), (6), (7) and (8).

4.2.2 Enabling collaboration. In this subsection, we present the additional constraints that enable collaboration among the robots with the help of intermediate locations, as illustrated in Figure 1b.

A robot can visit one of the *intermediate blocks* to either pick up or drop off an object. While picking up from an intermediate block, a validation of the timing consistency between the drop-off and pick-up of an object is required. We introduce new SMT variables named $loc\_time_{i,j}$ to add this ability. $loc\_time_{i,j}$ denotes the time step at which task t_i will be available at $loc_{i,j}$ at the j^{th} action-step. It is -1 if the task object is in transition.

Assume that a robot r_1 dropped the object of task t_l at location i_1 in action step j with $loc\_time_{l,j} = 20$. Now, suppose another robot r_2, which has been idle for all the action steps up to step $j+1$, goes to pick up this object. So, $pos_{2,j+1} = i_1$, but it is possible that $pos\_time_{2,j} + \texttt{dist}(pos_{2,j}, i_1) < 20$. Thus, even though r_2 will go to pick up the object at a later action step, it will reach the location before the task object is available there. Thus, in our constraints, we need to accommodate this possibility into the computation of pos_time as the action will be completed only when the pickup is done.

To accommodate the intermediate locations in $\mathcal{I}$ in our constraints, we update LOC as follows:

$$LOC = \Big(\bigcup_{t_m \in \mathcal{T}} \{L_m(0), L_m(1)\} \Big) \cup \Big(\bigcup_{i_n \in \mathcal{I}} \{i_n\} \Big).$$

Picking an object from an intermediate location. Constraints formulation for r_i picking up one of the task objects from one of the *intermediate blocks* in action step j is as below.

$$pick\_intermediate(r_i, t_m, i_n, j) \equiv$$
$$loc_{m,j-1} = i_n \tag{9a}$$
$$\wedge\ loc\_time_{m,j-1} \leq pos\_time_{i,j-1} + \texttt{dist}(pos_{i,j-1}, i_n) + 1 \tag{9b}$$
$$\wedge\ pos_{i,j} = i_n \ \wedge being\_carried_{m,j} = i \tag{9c}$$
$$\wedge\ pos\_time_{i,j} = pos\_time_{i,j-1} + \texttt{dist}(pos_{i,j-1}, i_n) + 1 \tag{9d}$$
$$\wedge\ loc_{m,j} = -1 \ \wedge loc\_time_{m,j} = -1 \tag{9e}$$
$$\wedge\ action_{i,j} = m \tag{9f}$$

Equation (9) is similar to Equation (2) except the extra constraint in Equation 9(b), which ensures that the task object is at the location before the robot reaches there to pick it up. Moreover, we need to add constraints to update loc_time in Equation (2), (3) and (7).

Waiting at an intermediate location. Constraints formulation for r_i waiting at an intermediate block in action step j is shown below. This is needed to take care of the situation when the robot reaches the intermediate location before the intended object is dropped there.

$$wait\_intermediate(r_i, t_m, i_n, j) \equiv$$
$$loc_{m,j-1} = i_n \tag{10a}$$
$$\wedge\ loc\_time_{m,j-1} > pos\_time_{i,j-1} + \texttt{dist}(pos_{i,j-1}, i_n) + 1 \tag{10b}$$
$$\wedge\ pos_{i,j} = i_n \ \wedge being\_carried_{m,j} = i \tag{10c}$$
$$\wedge\ pos\_time_{i,j} = loc\_time_{m,j-1} + 2 \tag{10d}$$
$$\wedge\ loc_{m,j} = -1 \ \wedge loc\_time_{m,j} = -1 \tag{10e}$$
$$\wedge\ action_{i,j} = m \tag{10f}$$

Equation (10) is similar to Equation (2) except the changes in Equation 10(b) and Equation 10(d). Equation 10(b) ensures that this is the case where the robot has reached the location before the task object. Equation 10(d) sets the $pos\_time_{i,j}$ to the time at which the task object can be picked up by the robot. After a robot drops the task at $loc\_time_{m,j-1}$ time, any other robot will take at least 1 unit of time to reach that location and 1 more unit to pick up the task from the intermediate location.

Dropping an object at an intermediate location. Constraints formulation for r_i dropping one of the task objects it carries to one

of the *intermediate blocks* in action step j is shown below.

$$
\begin{aligned}
&drop\_intermediate(r_i, t_m, i_n, j) \equiv \\
&\quad being\_carried_{m,j-1} = i && \text{(11a)}\\
&\wedge\ pos_{i,j} = n\ \wedge being\_carried_{m,j} = -1 && \text{(11b)}\\
&\wedge\ pos\_time_{i,j} = pos\_time_{i,j-1} + \texttt{dist}(pos_{i,j-1}, i_n) + 1 && \text{(11c)}\\
&\wedge\ loc_{m,j} = n\ \wedge action_{i,j} = m && \text{(11d)}\\
&\wedge\ loc\_time_{m,j} = pos\_time_{i,j} && \text{(11e)}
\end{aligned}
$$

Equation (11) is similar to Equation (3) as dropping at the intermediate location is similar to dropping at the task's goal location.

Robots' all possible actions. Finally, we have to change Equation (6) to

$$
\begin{aligned}
\bigwedge_{r_i \in \mathcal{R}} \bigwedge_{j=1}^{Z} \Big(& stay(r_i, j)\ \vee \bigvee_{k \in \{s_i\} \cup LOC} \Big((pos_{i,j-1} = k) \wedge \\
& \big(return(r_i, j) \vee \\
& \bigvee_{t_m \in \mathcal{T}} \big(pick(r_i, t_m, j)\ \vee drop(r_i, t_m, j) \vee \\
& \bigvee_{i_n \in \mathcal{I}} (pick\_intermediate(r_i, t_m, i_n, j) \vee \\
& \quad wait\_intermediate(r_i, t_m, i_n, j) \vee \\
& \quad drop\_intermediate(r_i, t_m, i_n, j)\ )\ )\big)\Big) \qquad \text{(12)}
\end{aligned}
$$

The final set of constraints is obtained as the conjunction of constraints capturing the initial states and those in Equation (1), (12), (7) and (8).

4.2.3 Other operational constraints. In our task planning framework, we can easily add other operational constraints. The constraints can be mainly of two types based on their association with time. If the constraint is associated with time, e.g., deadline, we need to handle the constraint in Task Planner as well as Path Planner. However, constraints like capacity are not related to time and can only be handled through Task Planner. We have added two constraints to demonstrate both types.

Capacity constraints. We can assign specific weights to task objects and specific weight-carrying capacities to robots. This constraint is independent of time, so it needs to be handled in Task Planner only. Let the variable $capacity_{i,j}$ denote the weight carrying capacity of robot r_i in action-step j and $weight_l$ denote a constant weight of object for task t_l. Now, we add the following constraint to all the sets of constraints involving a pickup:

$$
capacity_{i,j-1} \geq weight_l\ \wedge capacity_{i,j} = capacity_{i,j-1} - weight_l. \qquad \text{(13)}
$$

This checks for weight satisfiability before assigning a task to the robot and updates the weight-carrying capacity of the robot after picking it up. Similarly, for all the set of constraints involving a drop operation (Equation (3), (11)), we add:

$$
capacity_{i,j} = capacity_{i,j-1} + weight_l. \qquad \text{(14)}
$$

This updates the weight-carrying capacity of the robot after dropping.

Deadline constraints. We can also add a specific deadline $deadline_m$ to each task t_m by adding the following constraint for each task in Equation (8). This constraint is related to time, so it needs to be handled in Task Planner as well as Path Planner.

$$
loc\_time_{m,Z} \leq deadline_m. \qquad \text{(15)}
$$

4.2.4 Exclusion. We provide a way to add an already found task assignment $\mathcal{A}$ as an exclusion to the SMT planner so that the task planner finds the best solution excluding the already found assignments. Let $POS_{i,j}$ be the position of robot r_i and action step j in the existing solution.

$$
\bigvee_{r_i \in \mathcal{R}} \Big(\bigvee_{j \in \mathcal{Z}} (pos_{i,j} \neq POS_{i,j}) \Big) \qquad \text{(16)}
$$

To add the existing solution as an exclusion, we add Equation 16 to the set of constraints given to the SMT solver.

4.2.5 Objective function. We present the two cost functions related to the total cost and makespan of the trajectories.

(1) Total Cost: Here, we minimize the total work done by all the robots.

$$
\texttt{minimize}\ \Big(\sum_{r_i \in \mathcal{R}} pos\_time_{i,Z} \Big).
$$

(2) Makespan: Here, we minimize the time required to complete the mission.

$$
\texttt{minimize}\ \Big(\max_{r_i \in \mathcal{R}} pos\_time_{i,Z} \Big).
$$

The value of Z must be set such that it satisfies the condition $Z \geq 1 + \lceil |\mathcal{T}|/|\mathcal{R}| \rceil * 2$ for the problem to be solvable. In this case, the workload is balanced among the robots, which means that each robot is assigned at most $\lceil |\mathcal{T}|/|\mathcal{R}| \rceil$ tasks. Since a robot has to perform two actions per task and one action to return to its base, the value of Z must be at least $1 + \lceil |\mathcal{T}|/|\mathcal{R}| \rceil * 2$. On the other hand, if we search through all possible tasks while ignoring load balancing among robots, the condition becomes $Z \geq 1 + |\mathcal{T}| * 2$, which implies that in an extreme case, even one robot could complete all tasks while the other robots remain idle.

The task planner uses a binary search algorithm to optimize the cost function guided by the SMT constraints. Note that modern SMT solvers like Z3 [7] provide a mechanism to solve a minimization problem directly within the solver. However, our experience shows that attempting to solve an optimization problem directly using an SMT solver often fails to succeed within a reasonable time. In contrast, the binary search-based optimization method can successfully produce the result within a bound.

4.3 Path Planning

For the path planner, we adopt the CBS-PC algorithm [34] for multi-agent pathfinding for precedence-constrained goal sequences. CBS-PC uses Multi-Label A* [9] as its low-level planner. Multi-Label A* can find optimal paths for a sequence of goal locations. As we deal with intermediate drops and pickups, the intermediate pickup must be executed after the intermediate drop for the same task. This is taken care of by the precedence constraints presented in the algorithm. We also introduce the following enhancements to the basic CBS-PC algorithm: (i) makespan optimization criteria along

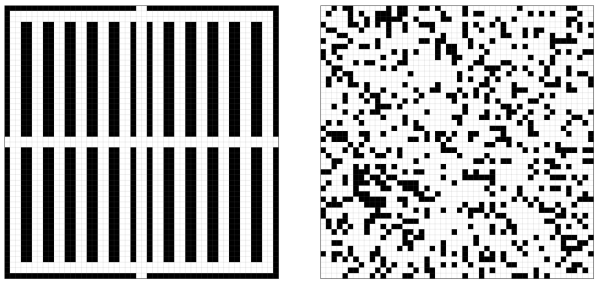


Figure 4: Warehouse (left) and Randomly generated (right) 50×50 map

with the sum of total costs, (ii) inclusion of deadlines support for goals and checkpoints, and (iii) handling empty goals as the task planner may not assign tasks to some robots.

5 Evaluation

We evaluate our planning methodology on various instances of pick-and-drop application scenarios.

5.1 Experimental Setup

For our experiments, we use a desktop computer with an i7-4770 processor with a 3.90 GHz frequency and 12 GB of memory. We use Z3 SMT solver [7] to solve task-planning problems. For MA*-CBS-PC, we adapt the C++ code provided by [34] with appropriate modifications. The source code of our implementation is available at https://github.com/iitkcpslab/Opt-ITPP.

In our experiments, we consider two planners: one optimizes the makespan (opt_makespan), and the other optimizes the total cost (opt_cost). We compare these planners with a state-of-the-art classical planner ENHSP-20 [1, 27]. Since our planners deal with numeric values for capacities and deadlines, we require a classical planner to support numeric values and provide optimal solutions. We explored the possibility of modeling our problem as a constrained TSP problem and utilizing the meta-heuristic algorithm LKH3 [12] to get a near-optimal solution. However, we did not find any extension of LKH3 that can deal with all the constraints we consider in our problem. On the other hand, it was quite straightforward to model our exact problem in SMT as well as in ENHSP-20.

We use a predefined workspace resembling a warehouse and a workspace with a randomly generated obstacle map resembling a disaster-stricken area (shown in Figure 4) to evaluate our algorithm. For any data point, we take the average of the results for 10 scenarios where the initial location of the robots and the task locations are generated randomly. For the evaluation on random workspace, the obstacle maps are also generated randomly.

In all the experiments, we have set the timeout (TO) as 3600s. In presenting the results, for all the cases where the planner fails to solve some problem instance within 3600s, we present the metric value as the average of the values for the instances the planner could solve successfully. We also mention the number of timeouts in the parenthesis to indicate the number of instances facing timeout.

5.2 Results

5.2.1 Comparison for varying workspace size. In this evaluation, we experiment with 2 robots and 2 tasks with $Z = 5$ for varying workspace sizes ranging from 10×10 to 100×100. Table 1 and Table 2 show the effect of varying map sizes on computation time (timeouts), makespan, and total cost for our planners and the ENHSP-20 planner for predefined and random workspace respectively. Our planners are able to solve all the problems in a few seconds. The ENHSP planner was able to solve 15% of the problems for the smallest 10×10 map and was unable to solve any problem with a larger map size. The result presents the makespan and the total cost, which is as per the expectations, showing a linear increase in makespan and total cost respectively with an increase in map size.

5.2.2 Comparison for varying Robots and Tasks without Collaboration. From the previous evaluation, we observe that the classical planner cannot solve problems for map size more than 10×10. So, in this experiment, we use maps of size 9×9. We experiment with 2 and 3 robots and the number of tasks ranged from 2 to 5. Since we aim for a load-balanced solution, we use a minimum satisfiable Z as it forces every robot to perform some work. Table 3 and 4 shows the effect of varying number of robots and tasks on computation time (# timeouts), makespan, and total cost. The classical planner cannot solve any problem for more than 3 tasks. Even for 3 tasks, it can solve only some of the problem instances. On the other hand, our planners perform significantly better compared to the classical planner. From the table, we also observe that the opt-makespan planner is more scalable compared to other planners.

5.2.3 Comparison for varying Robots and Tasks with Collaboration. We perform these experiments with a setup similar to the previous one, but we add some intermediate locations in the maps (randomly for randomly generated maps and predefined for warehouse maps). We execute the planner with both the optimization criteria for multiple values of Z. A value of Z=3 implies no collaboration; with a higher value of Z, the opportunity for intermediate pickup and drop arises. Table 5 and 6 represents the computation times (# timeouts), makespan, and total cost for various numbers of robots and tasks, and Z. For each robot and task, the computation time increases drastically for each increase in Z for our planner. Our planner cannot solve all the problems for 4 robots and 4 tasks with $Z = 9$. However, our planners are able to solve more problems faster than the classical planner. Higher Z values improve makespan for makespan optimization and total cost for total cost optimization.

5.2.4 Scalability Analysis. We also evaluate our algorithm for N robots and N tasks, where N ranges from 2 to 20 for a 100×100 workspace, to determine the scalability of our algorithm. The value of Z is decided to make the problem satisfiable without collaboration. Table 7 presents the computation time (Number of timeouts), makespan and total cost for varying numbers of robots and tasks. Our planner can successfully execute up to 19 robots with 19 tasks without experiencing failures for a timeout of 3600s. We also evaluate the time distribution between the task and the path planner. On average, the task planner consumes more than 98% of the total computation time. As the task planner explores a large search space to find the sequence of actions, the combinatorial explosion of possibilities makes the search exponentially large.

Table 1: Effect of Map Size on Computation Time, Makespan, and Total Cost for Different Planners on Warehouse Workspaces

MapSize	ENHSP			opt_makespan			opt_cost		
	Time (# TO)	Makespan	Total Cost	Time (# TO)	Makespan	Total Cost	Time (# TO)	Makespan	Total Cost
10	$2711.1_{\pm 1435.1}(7)$	$25.33_{\pm 3.06}$	$41.33_{\pm 1.15}$	$1.9_{\pm 0.6}(0)$	$27.6_{\pm 2.8}$	$49.4_{\pm 7.43}$	$4.0_{\pm 0.8}(0)$	$30.2_{\pm 4.37}$	$46.4_{\pm 4.2}$
20	TO(10)	-	-	$2.3_{\pm 0.7}(0)$	$56.2_{\pm 7.39}$	$102.0_{\pm 17.59}$	$4.3_{\pm 1.3}(0)$	$60.2_{\pm 7.15}$	$93.4_{\pm 12.26}$
30	TO(10)	-	-	$2.5_{\pm 0.5}(0)$	$74.2_{\pm 14.22}$	$134.6_{\pm 26.97}$	$5.4_{\pm 1.4}(0)$	$76.9_{\pm 16.31}$	$126.5_{\pm 24.08}$
40	TO(10)	-	-	$2.6_{\pm 0.8}(0)$	$104.2_{\pm 15.04}$	$189.7_{\pm 25.44}$	$6.2_{\pm 1.6}(0)$	$116.3_{\pm 10.63}$	$172.1_{\pm 30.7}$
50	TO(10)	-	-	$2.9_{\pm 1.0}(0)$	$137.8_{\pm 17.92}$	$248.2_{\pm 50.14}$	$4.9_{\pm 1.2}(0)$	$153.6_{\pm 32.36}$	$233.6_{\pm 40.27}$
60	TO(10)	-	-	$3.3_{\pm 0.8}(0)$	$181.0_{\pm 16.12}$	$330.5_{\pm 25.22}$	$6.1_{\pm 1.2}(0)$	$197.5_{\pm 32.78}$	$306.3_{\pm 48.84}$
70	TO(10)	-	-	$3.5_{\pm 1.2}(0)$	$199.8_{\pm 38.66}$	$368.0_{\pm 70.88}$	$5.5_{\pm 1.4}(0)$	$219.6_{\pm 53.15}$	$343.2_{\pm 68.43}$
80	TO(10)	-	-	$3.5_{\pm 1.4}(0)$	$227.6_{\pm 38.26}$	$409.2_{\pm 64.81}$	$7.0_{\pm 1.2}(0)$	$245.0_{\pm 37.4}$	$383.0_{\pm 69.55}$
90	TO(10)	-	-	$3.6_{\pm 1.3}(0)$	$234.2_{\pm 30.62}$	$443.6_{\pm 64.17}$	$6.6_{\pm 1.5}(0)$	$253.8_{\pm 40.1}$	$415.0_{\pm 63.53}$
100	TO(10)	-	-	$3.4_{\pm 1.2}(0)$	$282.2_{\pm 39.13}$	$522.5_{\pm 84.79}$	$7.1_{\pm 1.0}(0)$	$311.0_{\pm 65.67}$	$471.2_{\pm 61.54}$

Table 2: Effect of Map Size on Computation Time, Makespan, and Total Cost for Different Planners on Random Workspaces

MapSize	ENHSP			opt_makespan			opt_cost		
	Time (# TO)	Makespan	Total Cost	Time (# TO)	Makespan	Total Cost	Time (# TO)	Makespan	Total Cost
10	TO(10)	-	-	$1.8_{\pm 0.9}(0)$	$32.8_{\pm 6.12}$	$60.0_{\pm 9.48}$	$4.3_{\pm 1.1}(0)$	$36.0_{\pm 6.25}$	$56.2_{\pm 9.35}$
20	TO(10)	-	-	$2.4_{\pm 1.2}(0)$	$64.0_{\pm 8.42}$	$117.75_{\pm 20.18}$	$3.6_{\pm 2.2}(0)$	$70.25_{\pm 12.21}$	$109.0_{\pm 17.3}$
30	TO(10)	-	-	$2.3_{\pm 0.7}(0)$	$87.4_{\pm 20.18}$	$155.8_{\pm 41.77}$	$4.5_{\pm 1.1}(0)$	$92.4_{\pm 27.08}$	$149.2_{\pm 41.42}$
40	TO(10)	-	-	$2.8_{\pm 1.2}(0)$	$128.22_{\pm 14.91}$	$242.22_{\pm 29.11}$	$5.1_{\pm 1.7}(0)$	$138.78_{\pm 17.66}$	$218.56_{\pm 30.1}$
50	TO(10)	-	-	$2.9_{\pm 1.0}(0)$	$150.22_{\pm 19.01}$	$264.89_{\pm 51.21}$	$5.2_{\pm 1.8}(0)$	$167.11_{\pm 28.64}$	$248.44_{\pm 42.15}$
60	TO(10)	-	-	$2.7_{\pm 1.3}(0)$	$182.0_{\pm 37.08}$	$337.25_{\pm 61.2}$	$5.8_{\pm 2.8}(0)$	$195.0_{\pm 35.97}$	$307.0_{\pm 44.5}$
70	TO(10)	-	-	$3.4_{\pm 1.1}(0)$	$194.67_{\pm 20.12}$	$367.78_{\pm 36.8}$	$5.9_{\pm 2.4}(0)$	$200.0_{\pm 20.42}$	$352.89_{\pm 32.51}$
80	TO(10)	-	-	$3.0_{\pm 1.3}(0)$	$234.5_{\pm 41.07}$	$450.75_{\pm 71.9}$	$5.3_{\pm 3.1}(0)$	$254.25_{\pm 53.26}$	$406.5_{\pm 75.32}$
90	TO(10)	-	-	$3.3_{\pm 0.9}(0)$	$234.67_{\pm 52.13}$	$422.44_{\pm 108.64}$	$5.9_{\pm 2.5}(0)$	$251.11_{\pm 68.47}$	$381.78_{\pm 81.09}$
100	TO(10)	-	-	$3.2_{\pm 1.1}(0)$	$271.6_{\pm 27.19}$	$505.8_{\pm 74.21}$	$5.9_{\pm 1.8}(0)$	$294.8_{\pm 51.97}$	$475.4_{\pm 52.61}$

Table 3: Effect of the Number of Robots and Tasks on Computation Time, Makespan, and Total Cost for Different Planners on Warehouse Workspaces without Collaboration

R	T	ENHSP			opt_makespan			opt_cost		
		Time (# TO)	Makespan	Total Cost	Time (# TO)	Makespan	Total Cost	Time (# TO)	Makespan	Total Cost
2	2	$30.0_{\pm 12.0}(0)$	$21.8_{\pm 3.91}$	$37.7_{\pm 5.44}$	$0.0_{\pm 0.0}(0)$	$21.1_{\pm 3.21}$	$37.2_{\pm 3.94}$	$0.1_{\pm 0.3}(0)$	$21.3_{\pm 3.33}$	$36.7_{\pm 3.53}$
2	3	$3345.8_{\pm 4014.5}(5)$	$25.6_{\pm 4.56}$	$40.6_{\pm 4.22}$	$2.4_{\pm 1.0}(0)$	$25.2_{\pm 2.66}$	$46.9_{\pm 6.62}$	$3.7_{\pm 1.2}(0)$	$28.2_{\pm 5.12}$	$42.8_{\pm 4.34}$
2	4	TO(10)	-	-	$2.6_{\pm 0.8}(0)$	$26.6_{\pm 2.67}$	$51.8_{\pm 4.57}$	$5.0_{\pm 1.5}(0)$	$27.4_{\pm 3.27}$	$48.1_{\pm 2.85}$
3	3	$3362.9_{\pm 749.8}(9)$	$12.0_{\pm 0.0}$	$28.0_{\pm 0.0}$	$0.4_{\pm 0.7}(0)$	$19.2_{\pm 3.65}$	$49.9_{\pm 13.26}$	$0.4_{\pm 0.5}(0)$	$19.3_{\pm 3.71}$	$48.3_{\pm 11.44}$
3	4	TO(10)	-	-	$13.9_{\pm 6.1}(0)$	$22.8_{\pm 2.2}$	$64.7_{\pm 8.56}$	$64.6_{\pm 14.0}(0)$	$25.6_{\pm 3.1}$	$47.6_{\pm 5.32}$
3	5	TO(10)	-	-	$24.9_{\pm 11.8}(0)$	$24.8_{\pm 1.4}$	$71.2_{\pm 7.07}$	$317.2_{\pm 93.2}(0)$	$27.7_{\pm 2.5}$	$61.7_{\pm 4.4}$

Table 4: Effect of the Number of Robots and Tasks on Computation Time, Makespan, and Total Cost for Different Planners on Random Workspaces without Collaboration

R	T	ENHSP			opt_makespan			opt_cost		
		Time (# TO)	Makespan	Totla Cost	Time (# TO)	Makespan	Total Cost	Time (# TO)	Makespan	Total Cost
2	2	$562.8_{\pm 550.0}(0)$	$21.3_{\pm 5.29}$	$36.8_{\pm 10.54}$	$0.1_{\pm 0.3}(0)$	$20.4_{\pm 4.4}$	$36.8_{\pm 10.12}$	$0.1_{\pm 0.3}(0)$	$21.2_{\pm 5.35}$	$36.2_{\pm 9.54}$
2	3	$3276.2_{\pm 1023.9}(9)$	$24.0_{\pm 0.0}$	$30.0_{\pm 0.0}$	$2.1_{\pm 0.9}(0)$	$25.2_{\pm 2.15}$	$45.0_{\pm 7.79}$	$3.4_{\pm 0.7}(0)$	$27.4_{\pm 4.01}$	$41.8_{\pm 6.21}$
2	4	TO(10)	-	-	$3.0_{\pm 0.7}(0)$	$27.8_{\pm 3.33}$	$53.6_{\pm 6.59}$	$5.0_{\pm 1.6}(0)$	$28.0_{\pm 3.65}$	$51.6_{\pm 5.95}$
3	3	TO(10)	-	-	$0.3_{\pm 0.5}(0)$	$20.4_{\pm 4.2}$	$55.6_{\pm 12.64}$	$0.4_{\pm 0.5}(0)$	$20.8_{\pm 4.64}$	$51.0_{\pm 8.55}$
3	4	TO(10)	-	-	$9.7_{\pm 4.0}(0)$	$23.6_{\pm 1.58}$	$63.5_{\pm 6.24}$	$59.8_{\pm 20.9}(0)$	$28.0_{\pm 2.67}$	$49.6_{\pm 3.37}$
3	5	TO(10)	-	-	$20.4_{\pm 3.9}(0)$	$25.2_{\pm 3.16}$	$68.2_{\pm 10.81}$	$210.1_{\pm 78.6}(0)$	$27.8_{\pm 4.66}$	$62.2_{\pm 8.92}$

6 Related Work

Several authors have presented algorithmic solutions for finding optimal task assignments and the corresponding collision-free paths for multi-robot applications. Concurrent goal assignment and planning problem has been addressed by Turpin et al. for obstacle-free

Table 5: Effect of the Number of Robots and Tasks on Computation Time, Makespan, and Total Cost for Different Planners on Warehouse Workspaces with Collaboration

R	T	Z	ENHSP			opt_makespan			opt_cost		
			Time (# TO)	Makespan	Total Cost	Time (# TO)	Makespan	Total Cost	Time (# TO)	Makespan	Total Cost
2	2	5	$13.3_{\pm 9.9}(0)$	$26.2_{\pm 4.42}$	$27.4_{\pm 2.84}$	$0.2_{\pm 0.4}(0)$	$19.6_{\pm 3.56}$	$35.1_{\pm 5.05}$	$0.2_{\pm 0.4}(0)$	$19.6_{\pm 3.56}$	$34.2_{\pm 4.67}$
2	2	7	TO(10)	-	-	$1.7_{\pm 0.7}(0)$	$19.3_{\pm 3.33}$	$34.4_{\pm 5.3}$	$1.9_{\pm 0.6}(0)$	$26.2_{\pm 4.66}$	$27.4_{\pm 2.99}$
2	2	9	TO(10)	-	-	$15.7_{\pm 10.1}(0)$	$19.3_{\pm 3.33}$	$34.8_{\pm 5.07}$	$22.8_{\pm 7.7}(0)$	$26.2_{\pm 4.66}$	$27.4_{\pm 2.99}$
3	3	5	TO(10)	-	-	$0.5_{\pm 0.5}(0)$	$20.8_{\pm 3.25}$	$54.3_{\pm 9.75}$	$0.8_{\pm 0.4}(0)$	$21.4_{\pm 3.35}$	$51.3_{\pm 6.5}$
3	3	7	TO(10)	-	-	$11.7_{\pm 2.9}(0)$	$20.4_{\pm 2.84}$	$51.6_{\pm 7.93}$	$53.1_{\pm 23.4}(0)$	$26.6_{\pm 3.78}$	$40.6_{\pm 4.43}$
3	3	9	TO(10)	-	-	$729.2_{\pm 426.9}(0)$	$20.4_{\pm 2.84}$	$51.6_{\pm 7.93}$	$2458.9_{\pm 1487.2}(6)$	$32.0_{\pm 1.63}$	$32.0_{\pm 1.63}$
4	4	5	TO(10)	-	-	$2.4_{\pm 1.6}(0)$	$20.0_{\pm 2.57}$	$68.1_{\pm 9.88}$	$4.4_{\pm 1.6}(0)$	$20.9_{\pm 2.85}$	$60.1_{\pm 7.76}$
4	4	7	TO(10)	-	-	$505.8_{\pm 1096.5}(1)$	$19.22_{\pm 2.39}$	$66.44_{\pm 11.82}$	$1054.6_{\pm 239.4}(0)$	$24.4_{\pm 2.95}$	$44.8_{\pm 5.43}$
4	4	9	TO(10)	-	-	TO(10)	-	-	TO(10)	-	-

Table 6: Effect of the Number of Robots and Tasks on Computation Time, Makespan, and Total Cost for Different Planners on Random Workspaces with Collaboration

R	T	Z	ENHSP			opt_makespan			opt_cost		
			Time (# TO)	Makespan	Total Cost	Time (# TO)	Makespan	Total Cost	Time (# TO)	Makespan	Total Cost
2	2	5	$117.7_{\pm 113.1}(0)$	$26.4_{\pm 5.79}$	$27.0_{\pm 4.41}$	$0.0_{\pm 0.0}(0)$	$20.6_{\pm 4.3}$	$37.3_{\pm 9.53}$	$0.2_{\pm 0.4}(0)$	$20.8_{\pm 4.67}$	$36.6_{\pm 9.13}$
2	2	7	TO(10)	-	-	$1.7_{\pm 0.5}(0)$	$20.6_{\pm 4.53}$	$36.6_{\pm 9.62}$	$2.0_{\pm 0.8}(0)$	$26.4_{\pm 6.1}$	$27.0_{\pm 4.64}$
2	2	9	TO(10)	-	-	$13.9_{\pm 5.9}(0)$	$20.6_{\pm 4.53}$	$37.3_{\pm 10.04}$	$17.5_{\pm 5.0}(0)$	$26.4_{\pm 6.1}$	$27.0_{\pm 4.64}$
3	3	5	TO(10)	-	-	$0.5_{\pm 0.5}(0)$	$22.4_{\pm 3.56}$	$58.6_{\pm 9.3}$	$0.5_{\pm 0.5}(0)$	$23.2_{\pm 4.4}$	$53.0_{\pm 7.81}$
3	3	7	TO(10)	-	-	$10.4_{\pm 4.2}(0)$	$21.7_{\pm 3.47}$	$57.1_{\pm 14.3}$	$51.3_{\pm 24.1}(0)$	$27.2_{\pm 4.24}$	$39.8_{\pm 5.77}$
3	3	9	TO(10)	-	-	$937.9_{\pm 686.2}(0)$	$21.5_{\pm 3.5}$	$57.7_{\pm 11.89}$	$2799.5_{\pm 1347.3}(7)$	$32.0_{\pm 5.29}$	$32.0_{\pm 5.29}$
4	4	5	TO(10)	-	-	$1.3_{\pm 0.5}(0)$	$21.2_{\pm 2.86}$	$67.5_{\pm 9.63}$	$2.8_{\pm 0.6}(0)$	$21.8_{\pm 3.16}$	$61.8_{\pm 7.51}$
4	4	7	TO(10)	-	-	$78.6_{\pm 96.6}(0)$	$20.7_{\pm 2.67}$	$64.7_{\pm 11.99}$	$1118.3_{\pm 549.4}(0)$	$25.5_{\pm 3.5}$	$45.6_{\pm 5.25}$
4	4	9	TO(10)	-	-	$2498.6_{\pm 1360.5}(5)$	$20.8_{\pm 2.28}$	$62.8_{\pm 9.65}$	TO(10)	-	-

Table 7: Effect of the number of Robots and Tasks on Computation Time and Makespan for Optimizing Makespan

R	T	Time (# TO)	Makespan	Total Cost
2	2	$0.4_{\pm 0.5}(0)$	$225.8_{\pm 35.31}$	$390.4_{\pm 46.4}$
3	3	$0.8_{\pm 0.4}(0)$	$258.2_{\pm 20.69}$	$655.3_{\pm 59.03}$
4	4	$1.6_{\pm 0.5}(0)$	$257.2_{\pm 31.82}$	$817.8_{\pm 131.36}$
5	5	$4.4_{\pm 1.0}(0)$	$250.6_{\pm 23.42}$	$991.7_{\pm 87.86}$
6	6	$8.6_{\pm 2.2}(0)$	$239.0_{\pm 32.64}$	$1138.4_{\pm 173.03}$
7	7	$14.3_{\pm 4.3}(0)$	$267.0_{\pm 39.45}$	$1420.2_{\pm 158.17}$
8	8	$26.8_{\pm 10.2}(0)$	$256.8_{\pm 32.99}$	$1593.5_{\pm 173.57}$
9	9	$44.0_{\pm 16.8}(0)$	$281.2_{\pm 43.01}$	$1888.2_{\pm 231.84}$
10	10	$62.1_{\pm 20.3}(0)$	$264.4_{\pm 32.3}$	$1991.0_{\pm 246.13}$
11	11	$103.6_{\pm 42.7}(0)$	$279.8_{\pm 51.42}$	$2314.5_{\pm 280.51}$
12	12	$131.3_{\pm 23.5}(0)$	$290.2_{\pm 32.92}$	$2609.3_{\pm 229.89}$
13	13	$244.0_{\pm 67.9}(0)$	$279.4_{\pm 32.95}$	$2648.8_{\pm 282.94}$
14	14	$288.9_{\pm 79.4}(0)$	$287.4_{\pm 30.41}$	$3011.3_{\pm 259.7}$
15	15	$459.3_{\pm 101.4}(0)$	$278.0_{\pm 43.73}$	$3112.8_{\pm 374.12}$
16	16	$817.0_{\pm 351.2}(0)$	$267.4_{\pm 47.41}$	$3241.5_{\pm 405.7}$
17	17	$1044.9_{\pm 175.7}(0)$	$270.0_{\pm 30.08}$	$3397.1_{\pm 282.86}$
18	18	$1362.9_{\pm 375.2}(0)$	$269.8_{\pm 32.17}$	$3646.4_{\pm 356.12}$
19	19	$1659.1_{\pm 429.8}(0)$	$276.2_{\pm 28.09}$	$3949.2_{\pm 373.36}$
20	20	$3053.3_{\pm 883.4}(7)$	$309.33_{\pm 33.25}$	$4384.67_{\pm 351.15}$

environments [29] and in the environment cluttered with obstacles [30] without a guarantee of optimality. On the other hand, the optimal goal assignment and the collision-free path-finding problem have been addressed in [3, 14, 19]. Recently, Okumura and Défago [24] have proposed a sub-optimal but fast algorithm for simultaneous target assignment and path planning efficiently for a large-scale multi-robot system. Though the goal assignment is a form of task assignment, it is beyond the scope of these algorithms to deal with complex constraints (e.g., payload capacity, task deadline) for the robots or the possibility of robot-robot collaboration. Though the problem of transferring payloads in packet transfers [21] and deadline-aware planning [22] in a multi-agent environment have been studied, the proposed solutions apply to the very specific problems. Several authors have presented mechanisms to solve the integrated task and path planning problem for multi-robot systems, where the task specifications are given using linear temporal logic [10, 16, 31]. These methods are either not scalable [31] or compromise on finding collision-free paths to achieve scalability [10, 16].

As our task planner is based on SMT solving, we outline the research work that employed various SMT techniques in the context of robot task and path planning. Nedunuri et al. [23] first propose an SMT-based solution to solve the integrated task and motion planning problem for a static environment from a user-given plan outline. Subsequently, Dantam et al. [6] extend this idea to generate motion plans for a robot arm in a dynamic environment by getting feedback from the motion planner and invoking the SMT solver in the incremental mode. However, these papers do not aim to provide optimal solutions. Furthermore, they deal with only one robot manipulator and thus do not deal with the complexities that arise from collision avoidance for multiple mobile robots. Leofante et al. [17]

present an SMT-based integrated planning framework where the high-level task and path planning problem for a fixed horizon is encoded as a monolithic SMT problem which is solved using an SMT solver extended with the capability of finding an optimal solution. However, their method does not consider collisions among robots, and such an integrated planning strategy can tackle only a few robots and small workspaces. Imeson and Smith [15] introduced a combination of a SAT solver and a TSP solver to solve the integrated multi-agent goal assignment and path planning problem. Though their solution provides an optimality guarantee on the total cost of the independent paths for the robots, these paths are not guaranteed to be collision free. Saha et al. [26] introduce a scalable method for solving the collision-free path planning problem for a given goal assignment for a multi-robot system. They exploit the SMT solver's capability to generate an unsatisfiable core to assign priorities to the robots to avoid any potential deadlock situation.

Several researchers have focused on the multi-robot pickup and delivery problem. Michal et al. [32] provides a distributed algorithm to solve a well-formed multi-agent pickup-delivery problem. Ma et al. [18, 20] provide several algorithms addressing the MAPD problem across online and offline contexts. These approaches perform path planning in two stages, resulting in sub-optimal collision-free trajectories. Our approach employs CBS-PC [34], which efficiently computes optimal collision-free trajectory. Though we take the pickup-delivery problem as an application, our SMT-based approach is more general in dealing with many complex constraints in a task planning problem. Some approaches based on Large Neighborhood Search [4, 33] are efficient and scalable. However, these algorithms do not guarantee optimality or completeness; in contrast, our approach is complete and optimal.

7 Conclusion

We have presented a generic integrated task and path planning algorithm for multi-robot systems and demonstrated the applicability of this framework on the pickup delivery problem that is at the core of any automated warehouse management system. Our planning framework provides an opportunity to combine the strength of an optimal task planner and an optimal path planner to design an optimal planner capable of solving complex multi-robot logistics planning problems which is beyond the scope of the state-of-the-art multi-agent classical planners.

Acknowledgments

This research was supported in part by a research grant from Max-Planck Society, Germany to support the Partner Group between MPI-SWS, Germany and IIT Kanpur, India and DFG project 389792660 TRR 248–CPEC.

References

[1] [n. d.]. The ENHSP Planning System. https://sites.google.com/view/enhsp/
[2] Aakash and I. Saha. 2022. It Costs to Get Costs! A Heuristic-Based Scalable Goal Assignment Algorithm for Multi-Robot Systems. In *ICAPS*, Vol. 32. 2–10.
[3] K. Brown, O. Peltzer, M. A. Sehr, M. Schwager, and M. J. Kochenderfer. 2020. Optimal sequential task assignment and path finding for multi-agent robotic assembly planning. In *ICRA*. IEEE, 441–447.
[4] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, and P. J. Stuckey. 2021. Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. *IEEE Robotics and Automation Letters* 6, 3 (2021), 5816–5823.
[5] M. Crosby, M. Rovatsos, and R. P. A. Petrick. 2013. Automated Agent Decomposition for Classical Planning. In *ICAPS*, Vol. 23. 46–54.
[6] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki. 2018. An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research* 37, 10 (2018), 1134–1151.
[7] L. M. de Moura and N. Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*. 337–340.
[8] I. Gavran, R. Majumdar, and I. Saha. 2017. Antlab: A Multi-Robot Task Server. *ACM Trans. Embedded Comput. Syst.* 16, 5 (2017), 190:1–190:19.
[9] F. Grenouilleau, W.-J. van Hoeve, and J. N. Hooker. 2019. A multi-label A* algorithm for multi-agent pathfinding. In *ICAPS*, Vol. 29. 181–185.
[10] D. Gujarathi and I. Saha. 2022. MT*: Multi-Robot Path Planning for Temporal Logic Specifications. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2022, Kyoto, Japan, October 23-27, 2022*. IEEE, 13692–13699.
[11] P. E. Hart, N. J. Nilsson, and B. Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE* 4, 2 (1968), 100–107.
[12] K. Helsgaun. 2017. An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. *Roskilde: Roskilde University* 12 (2017).
[13] D. Hennes, D. Claes, W. Meeussen, and K. Tuyls. 2012. Multi-robot collision avoidance with localization uncertainty. In *AAMAS*. 147–154.
[14] W. Hönig, S. Kiesel, A. Tinka, J. Durham, and N. Ayanian. 2018. Conflict-based search with optimal task assignment. In *AAMAS*. 757–765.
[15] Frank Imeson and Stephen L. Smith. 2019. An SMT-Based Approach to Motion Planning for Multiple Robots With Complex Constraints. *IEEE Trans. Robotics* 35, 3 (2019), 669–684.
[16] Y. Kantaros and M. M. Zavlanos. 2020. STyLuS*: A Temporal Logic Optimal Control Synthesis Algorithm for Large-Scale Multi-Robot Systems. *Int. J. Robotics Res.* 39, 7 (2020).
[17] Francesco Leofante, Erika Ábrahám, Tim Niemueller, Gerhard Lakemeyer, and Armando Tacchella. 2019. Integrated Synthesis and Execution of Optimal Plans for Multi-Robot Systems in Logistics. *Inf. Syst. Frontiers* 21, 1 (2019), 87–107.
[18] M. Liu, H. Ma, J. Li, and S. Koenig. 2019. Task and path planning for multi-agent pickup and delivery. In *AAMAS*. 1152–1160.
[19] H. Ma and S. Koenig. 2016. Optimal Target Assignment and Path Finding for Teams of Agents. In *AAMAS*. 1144–1152.
[20] H. Ma, J. Li, T. K. Satish Kumar, and S. Koenig. 2017. Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks. In *AAMAS*. 837–845.
[21] H. Ma, C. A. Tovey, G. Sharon, T. K. Satish Kumar, and S. Koenig. 2016. Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem. In *AAAI*. 3166–3173.
[22] H. Ma, G. Wagner, A. Felner, J. Li, T. K. Satish Kumar, and S. Koenig. 2018. Multi-Agent Path Finding with Deadlines. In *IJCAI*. 417–423.
[23] S. Nedunuri, S. Prabhu, M. Moll, S. Chaudhuri, and L. E. Kavraki. 2014. SMT-based synthesis of integrated task and motion plans from plan outlines. In *ICRA*. 655–662.
[24] K. Okumura and X. Défago. 2023. Solving simultaneous target assignment and path planning efficiently with time-independent execution. *Artif. Intell.* 321 (2023), 103946.
[25] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia. 2014. Automated composition of motion primitives for multi-robot systems from safe LTL specifications. In *IROS*. 1525–1532.
[26] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia. 2016. Implan: Scalable Incremental Motion Planning for Multi-Robot Systems. In *ICCPS*. 43:1–43:10.
[27] E. Scala, P. Haslum, S. Thiébaux, and M. Ramirez. 2020. Subgoaling techniques for satisficing and optimal numeric planning. *JAIR* 68 (2020), 691–752.
[28] M. Turpin, N. Michael, and V. Kumar. 2013. Trajectory planning and assignment in multirobot systems. In *Algorithmic Foundations of Robotics*. Springer, 175–190.
[29] M. Turpin, N. Michael, and V. Kumar. 2014. Capt: Concurrent assignment and planning of trajectories for multiple robots. *I. J. Robotics Res.* 33, 1 (2014), 98–112.
[30] M. Turpin, K. Mohta, N. Michael, and V. Kumar. 2014. Goal assignment and trajectory planning for large teams of interchangeable robots. *Auton. Robots* 37, 4 (2014), 401–415.
[31] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus. 2013. Optimality and Robustness in Multi-Robot Path Planning with Temporal Logic Constraints. *I. J. Robotics Res.* 32, 8 (2013), 889–911.
[32] M. Čáp, J. Vokřínek, and A. Kleiner. 2015. Complete Decentralized Method for Online Multi-robot Trajectory Planning in Well-formed Infrastructures. In *ICAPS*. 324–332.
[33] Q. Xu, J. Li, S. Koenig, and H. Ma. 2022. Multi-goal multi-agent pickup and delivery. In *RSJ (IROS)*. IEEE, 9964–9971.
[34] H. Zhang, J. Chen, J. Li, B. C Williams, and S. Koenig. 2022. Multi-Agent Path Finding for Precedence-Constrained Goal Sequences. In *ICAAMS*. 1464–1472.