Optimization-based or AI Task Planning for Scenarios with Cooperating Mobile Manipulators?

Stefan-Octavian Bezrucav, Yifan Liu and Burkhard Corves
Institute of Mechanism Theory, Machine Dynamics and Robotics, RWTH Aachen University, Eilfschornsteinstrasse 18, 52062 Aachen, Germany

Keywords: Task Planning, Mobile Manipulators, MILP, CP, Temporal AI Planning.

Abstract: Task planning has become one of the most important components of the control system for teams of cooperating robotic systems in complex scenarios. It plays such a critical role, as it must determine, order and assign a high variety of different tasks to the involved actors, such that at the end, the goals are reached while some metric, as the total execution time, is minimized. In this paper the analysis of three task planning approaches, mixed-integer linear programming (MILP), constraint programming (CP) and automated temporal planning (TP), with respect to three criteria is targeted. These criteria are the CPU time for plan generation, the plan makespan and the flexibility of the modelling approach. For the analysis, an intricate scenario in a kitchen environment with a team of multiple mobile manipulators is developed. The models and results are then compared to derive the advantages and disadvantages of each strategy.

1 INTRODUCTION

With the increasing complexity of automation scenarios it is necessary that more actors cooperate in order to reach more elaborate goals. These complex goals imply that the actors must not execute anymore only a small, fixed set of repetitive actions. They need to perform intricate tasks that must be selected and scheduled based on their capabilities and considering interactions, the actual status of the system and the goals. In these cases, the process of determining the sequence of tasks needed to be executed and their assignment to the actors is not anymore trivial. This process is called *task planning* and it becomes even more challenging when some optimization goals, like a minimal total execution time, are thrived.

This paper is part of a bigger work in which six different planning strategies for automation scenarios with cooperating mobile manipulators are studied. They are Mixed-Integer Linear Programming, Constraint Programming, Temporal Planning, Auction Algorithms, Genetic Algorithms and Hierarchical Task Network. In the followings, the focus is set only on the first three approaches. The usability of these approaches is analysed by evaluating them with respect to three factors: the model flexibility (the effort required to adapt the model for a different planning problem of the same scenario), the plan makespan (total execution time) and the CPU time to generate that plan. For this purpose, a special scenario with mobile manipulators in a kitchen environment is designed to present the different possibilities to model and solve task planning problems.

2 BACKGROUND

In this paper, three different planning strategies are studied. Mixed-Integer Linear Programming (MILP) and Constraint Programming (CP) are two mathematical optimization approaches. MILP tackles problems which contains a series of variables (continuous or integer), parameters, linear constraints and a linear objective function to be optimized. The standard technique to solve MILP problems is represented by branch-and-bound algorithms (Land, A. H. and Doig, A. G., 1960). The idea of this method is to iteratively partition the feasible solution region into more subdivisions (nodes) and ignore the integer-requirements to acquire a linear program (LP) in each subdivision. The LP at each node in the search tree is then solved with the Simplex method, which returns a set of boundaries that are used to guide the solving process (Murty, 1985). Task planning problems can be then formulated as MILP problems and solved with the above algorithms. For example, in (Booth et al., 2016) and (Yi et al., 2020), MILP is applied for multi-robot task planning problems in a retirement home and in a kitchen scenario.

Bezrucav, S., Liu, Y. and Corves, B.
Optimization-based or AI Task Planning for Scenarios with Cooperating Mobile Manipulators?.
DOI: 10.5220/0010509001150122
In *Proceedings of the 18th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2021)*, pages 115-122
ISBN: 978-989-758-522-7

CP is more general than MILP, as it can solve problems with various types of variables and constraints in a wide range of arithmetic expressions (Gervet, 2006). The solving process, called Constraint Propagation, is also a tree search. Different to MILP, instead of solving a LP-program, at each node the values set of a variable is reduced by eliminating those values that violate the problem constraints (Rossi et al., 2006). Like MILP, CP is also used in many industrial applications, such as assignment problems (Simonis, 2001) and scheduling problems (Behrens, J. K. et al., 2019) and (Mokhtarzadeh, M. et al., 2020). In (Booth et al., 2016) and (Ku, W.-Y., Beck, J. C., 2016), CP is applied to solve the same task planning and job shop scheduling problems, as MILP does, in order to make a comparison between them.

The last planning strategy, Temporal Planning (TP), is an AI planning technique. The planning problem is usually described through a domain in which the types of the elements from the environment, the predicates and the possible actions are defined. The predicates are boolean functions which can take one or more previously defined types, as variables. Each state of the environment is described through a set of grounded predicates, which represent the boolean functions that are true for specific input values. For example, the predicate *agv_at_pose ?agv ?pose* can be grounded to *agv_at_pose agv1 pose2*, which implies that *agv1* is at pose *pose2*. At the end, the actions are used to modify the values of the predicates and thus transform one state in another one. Given an initial state described by a set of grounded predicates, the TP algorithms search and order with specific heuristics the grounded actions that must be executed such that the system is transformed from the given state to a state in which all set goals are achieved. The planning problem is encoded in the *Plannin Domain Definition Language* (PDDL) (Fox and Long, 2003) that can be understood by different automated planners. One widely used deterministic heuristic temporal planner is POPF (Coles et al., 2010). Such approaches were used for example in (Bezrucav and Corves, 2020).

3 MODEL AND CONCEPT

In this paper, a scenario in the kitchen of a restaurant is defined, in which a team of mobile manipulators executes a series of specific tasks in this shared space. The scenario consists of two parts, the physical world and the planning problem. The physical world can be further divided into the environment and the mobile manipulators, both of which are shown in Figure 1. The environment contains two tool-banks, two refrigerators, two cookers, two worktables and two normal tables. The refrigerators K_1, K_2 store all the necessary food required to cook meals and the tool-benches S_1, S_2 offer the necessary tools for the cooking process. The worktables T_1, T_2 and cookers H_1, H_2 provide the places to process the food. On the right side, there are two normal tables T_3, T_4 where robots can put the finished food on. There is only one fixed parking pose near to all locations (white squares), where the mobile manipulators can park.

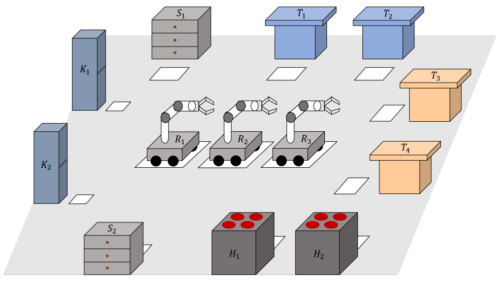


Figure 1: The kitchen scenario with mobile manipulators.

The planning problem includes the initial state of the system, the goals, the available actions and further rules that must be obeyed. In the initial state, all pasta are kept in portions in refrigerator K_1 and all salads in refrigerator K_2. The tools to process the salad and pasta are stored in tool-banks S_1 and S_2. At the beginning, the robots are parked on the initial positions shown in Figure 1. There are two kinds of goals in this framework: processing the salad and cooking the pasta. The actions that the robots can execute are *move*, *load*, *unload*, *attach*, *detach*, *generic_action_1* and *generic_action_2*. The execution duration of all actions except for the *move* action are fixed. The execution duration of the latter represents the travel time between the different locations. *generic_action_1* is the action to process the salad and *generic_action_2* implies cooking the pasta. The generic actions must be executed with the corresponding tools and food portions and at the selected positions. The salad must be processed on worktable T_1 or T_2 and pasta should be cooked on the cooker H_1 or H_2. After the pasta or salad is well processed, the salad must be served on the table T_3 and the pasta on T_4. Moreover, the robots should also bring the tools back to the corresponding tool-banks at the end.

A task precedence graph for one goal to process the salad and two goals to cook the pasta is shown in Figure 2. The actions required in order to reach these two types of goals and their precedence are detailed with the help of this precedence graph. In order to process the salad (T1 - T7), any of the robots must travel to the refrigerator K_1, *load* the salad, then move

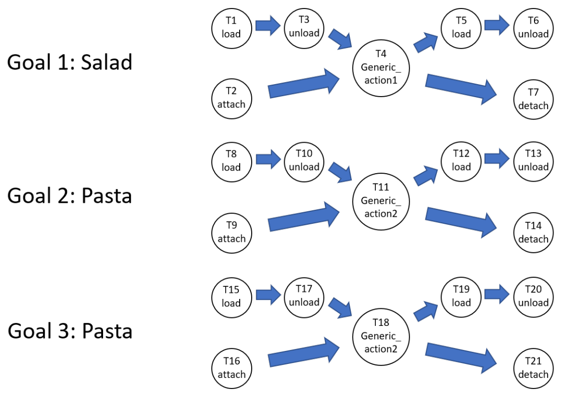


Figure 2: The task precedence graph for the planning problem.

to the worktable T_1 or T_2 and *unload* the salad. Besides, the same robot, or another one, should go to the tool-bank S_1 and *attach* the tool. The *load* action (T1) must be executed before the *unload* action (T3), but there is no precedence constraint between the *attach* action (T2) and the *load*, *unload* actions (T1, T3). After T1, T2 and T3 are finished, *generic_action_1* (T4) can be carried out in order to make the salad. Then, the finished salad must be *loaded* from worktable (T5) and be taken to table T_3 as well as *unload* it (T6). At last, the tool should be *detached* at tool-bank S_1 (T7). Analogously, the only precedence constraint between tasks T5, T6 and T7 is that T5 should be finished before T6 starts. Additionally, all the tasks can be done cooperatively, by different mobile manipulators, as long as the constraints are all fulfilled. The second and third goals, to cook the pasta, can be reached with a similar pattern (T8 - T14 and T15- T21 in Figure 2).

In order to make the task planning problems for this scenario closer to real-world problems, some general rules are also made and formulated as constraints:

1. Each robot can take up to two portions of food at a time.
2. Each robot can have up to one tool at a time.
3. Each location in the environment can be visited by only one robot at a time.
4. Precedence rule: The fixed task precedence rules in Figure 2 must be obeyed.
5. Same-robot rule: Some tasks must always be finished by the same robot.
6. Same-place rule: Some tasks must always be finished at the same places.

While the first three rules are straight-forward, the fourth rule is important especially when tasks that are related in one precedence order are allocated to different robots, e.g. T3 is assigned to robot R1 and T4 to R2. This is a known as a cross-schedule dependency.

Table 1: Task planning problems with G goals and R robots.

	R	G	goal types
Pb. 1	3	3	(salad, pasta, pasta)
Pb. 2	2	3	(salad, pasta, pasta)

In this case, the two tasks are separated in two schedules, therefore, the schedules of these robots should be carefully managed, so that the tasks are executed in the correct order. Apart from the precedence constraints, those tasks from Figure 2 are also not completely independent one from another. Some tasks must be finished by the same robot, such as T1 and T3, because the robot who loads the food to its platform should also unload it. The working places of some tasks should be the same like T3 and T4, because *generic_action_1* should be executed where the food is unloaded. Hence, the last two rules are also included.

Due to the high-complexity of the model that considers that many interactions and constraints for the actors and tasks, only planning problems with a low number of robots and goals are solvable. In this context, only two task planning problems in the kitchen scenario for R robots and G goals are considered. Their parametrization is presented in Table 1.

4 MODELLING AND RESULTS

This section includes the description of the models and the plans generated with the MILP, CP and Temporal planning approaches.

4.1 MILP and CP

The same model can be used for both the MILP and CP strategies. It is defined as:

Parameters:

N : set of tasks, $N = \{1, 2, ..., J\}$

M : set of robots, $M = \{1, 2, ..., R\}$

O : sets of possible positions for each task
$O = \{1 : [..], .., J : [..]\}$

p_{ij} : coord. of position j for task i, $\forall i \in N, j \in O[i]$

a_i : coord. of the initial position of robot i, $\forall i \in M$

t_i : time consumption of task i, $\forall i \in N$

H : a large integer, such as 100000

P : set of task pairs who should obey the precedence rule

S : set of task pairs who should obey the same-robot rule

W : set of task pairs who should obey the same-place rule

T : set of "attach" and "detach" task pairs

F : set of "load" and "unload" task pairs

Variables:

$$x_{ij} = \begin{cases} 1 & \text{if task } i \text{ is assigned to robot } j \\ 0 & \text{otherwise}, \forall i \in N, j \in M \end{cases}$$

$$z_{in} = \begin{cases} 1 & \text{if task } i \text{ is executed at position } n \\ 0 & \text{otherwise}, \forall i \in N, n \in O[i] \end{cases}$$

$$y_{ikj} = \begin{cases} 1 & \text{if task } k \text{ is after task } i \text{ in robot } j\text{'s plan} \\ 0 & \text{otherwise}, \forall i,k \in N, j \in M \end{cases}$$

$$q_{inkm} = \begin{cases} 1 & \text{if task } k \text{ is after task } i \text{ and both tasks are executed at the same position} \\ 0 & \text{otherwise}, \forall i,k \in N, i \neq k, n \in O[i], m \in O[k], p_{in} = p_{km} \end{cases}$$

s_i : start time of task i, $\forall i \in N$

t_{max} : total execution time of the plan

Constraints:

1. $\sum_{j \in M} x_{ij} = 1 \qquad \forall i \in N$
2. $\sum_{n \in O[i]} z_{in} = 1 \qquad \forall i \in N$
3. $s_i + t_i + |p_{in} - p_{km}| \leq s_k + H * ((1 - y_{ikj}) + (1 - x_{ij}) + (1 - x_{kj}) + (1 - z_{in}) + (1 - z_{km}))$
   $s_k + t_k + |p_{in} - p_{km}| \leq s_i + H * (y_{ikj} + (1 - x_{ij}) + (1 - x_{kj}) + (1 - z_{in}) + (1 - z_{km}))$
   $\forall i,k \in N, i \neq k, n \in O[i], m \in O[k], j \in M$
4. $s_i + t_i \leq s_k + H * ((1 - q_{inkm}) + (1 - z_{in}) + (1 - z_{km}))$
   $s_k + t_k \leq s_i + H * (q_{inkm} + (1 - z_{in}) + (1 - z_{km}))$
   $\forall i,k \in N, i \neq k, n \in O[i], m \in O[k], p_{in} = p_{km},$
5. $y_{ikj} = 0 \qquad \forall i,k \in N, i = k, j \in M$
6. $y_{ikj} + H * (x_{il} - 1) \leq 0, y_{ikj} + H * (x_{kl} - 1) \leq 0,$
   $\forall j,l \in M, j \neq l, \forall i,k \in N$
7. $s_i \geq |a_j - p_{in}| - H * ((1 - x_{ij}) + \sum_{k \in N} y_{kij} + (1 - z_{in})), \forall i \in N, j \in M, n \in O[i]$
8. $s_i + t_i + |p_{in} - p_{km}| \leq s_k + H * ((1 - z_{in}) + (1 - z_{km})) \forall (i,k) \in P, n \in O[i], m \in O[k]$
9. $x_{ij} = x_{kj} \qquad \forall (i,k) \in S, j \in M$
10. $z_{in} = z_{kn} \qquad \forall (i,k) \in W, n \in O[i]$ or $O[k]$
11. $y_{a_1 a_2 j} + y_{a_1 b_2 j} + y_{b_1 a_2 j} + y_{b_1 b_2 j} < 4$
    $\forall (a,b) \in T, \forall j \in M$
12. $y_{a_1 a_2 j} + y_{a_1 b_2 j} + y_{a_1 c_2 j} + y_{b_1 a_2 j} + y_{b_1 b_2 j} + y_{b_1 c_2 j} + y_{c_1 a_2 j} + y_{c_1 b_2 j} + y_{c_1 c_2 j} < 9$
    $\forall a,b,c \in F, a \neq b, b \neq c, a \neq c, \forall j \in M$
13. $t_{max} \geq s_i + t_i \qquad \forall i \in N$

Objective function: min t_{max}

The model consists of four parts. In the parameter part, two sets of integer numbers N, M are used to represent all tasks and robots, respectively. In both problems, J, the total number of tasks, is equal to 21 and R, the total number of robots, is 2 or 3. Furthermore, some tasks have multiple possible locations where they can be executed, hence, a set O is added to provide the list of all possible locations where each task can be executed. For each pair of a task i and a position j where it can be executed, the coordinate of that position is defined as p_{ij}. Further on, the initial location of robot i is fixed with the position parameter a_i. By setting the positions of each task, the initial and end conditions, such as the initial and end positions of the objects from the environment (e.g. the tools and the foods) are defined. Further on, the sets of task pairs P, S, W, T, F are defined. A part of the task pairs corresponding to the precedence graph are presented in Table 2.

Table 2: Task pairs in parameters P, S, W, T, F.

Parameter	Task pairs
P	(1,3), (2,4), (3,4), (4,5), (4,7), (5,6)...
S	(1,3), (2,4), (4,7), (5,6)...
W	(3,4), (4,5)...
T	(2,7); (9,14);...
F	(1,3), (5,6); (8,10), (12,13)...

Six variables are defined in this model. x_{ij} and z_{in} are binary decision variables that present the decision of task and position assignment, respectively. x_{ij} is assigned a value of 1 if task i is allocated to robot j and 0 otherwise. z_{in} has a value of 1, if the position n is chosen as the location of task i.

Further on, y_{ikj} and q_{inkm} are both binary decision variables for task sequence. y_{ikj} has the value 1 if two tasks i and k are assigned to the same robot j and task i precedes task k. On the other hand, q_{inkm} has a value of 1, if both tasks i and k are executed at the same place, thus $p_{in} = p_{km}$, and task i precedes task k. Moreover, s_i presents the start time of task i and t_{max} is the total duration of the schedule.

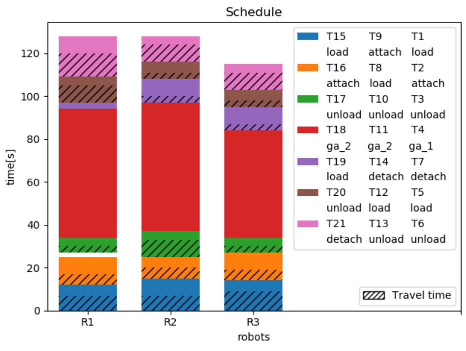
The constraints are defined by the equalities and inequalities 1-13. The constraints 1 and 2 ensure that each task is assigned to a robot and each task has exactly one location where it can be executed. After that, the constraint 3 contains two sets of equations, which are disjunctive sequence constraints to prevent the time intervals of the tasks in each robot's schedule from overlapping. Both equations take real effect when the term multiplied with H is equal to 0. For the first equation, this happens when tasks i and k are executed at position n and m respectively ($1-z_{in} = 1-z_{km} = 0$), tasks i and k are both assigned to robot j ($1-x_{ij} = 1-x_{kj} = 0$) and task i precedes k ($1-y_{ikj} = 0$). The entire inequality then turns to $s_i + t_i + |p_{in} - p_{km}| \leq s_k$. With the travel time between two tasks calculated as $|p_{in} - p_{km}|$ (each robot's speed is fixed to 1[-]/s), the inequality forces the start time of task k to be greater than the finish time of task i plus the travel time between the poses for tasks i and k. Otherwise if task i should be after k, thus $y_{ikj} = 0$, the second equation will take similar effect.

However, if two tasks are planned at the identical location, their time intervals should also not overlap even when they might not be assigned to the same robot (general rule 3). Therefore, constraint 4 is set up. In case that tasks i and k are executed at locations n and m and their positions are same ($p_{in} = p_{km}$), if task i precedes k ($q_{inkm} = 1$) the first equation becomes $s_i + t_i \leq s_k$ and constrains the start time of task k and vice versa.

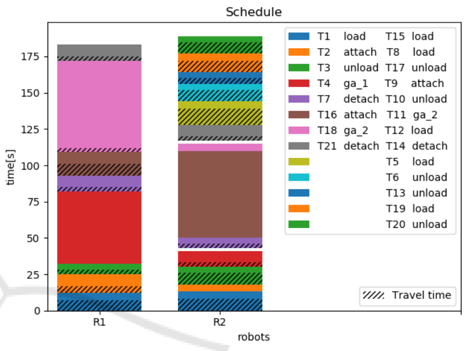
So far, constraint 3 over y_{ikj} is defined for the case that $i \neq k$ and there is no constraint over y_{ikj} when $i = k$, so constraint 5 is added to constrain y_{ikj} for this case. Besides, the two inequalities in 3 take real effects only when task i, k are both assigned to robot j ($x_{ij} = x_{kj} = 1$), but there is also no restriction on the value of y_{ikj}, when task i or k is not assigned to robot j. Hence, constraint 6 is defined. In 6, if task i is assigned to another robot l instead of j ($x_{il} = 1$) the inequality turns to $y_{ikj} \leq 0$ and y_{ikj} can only be 0 in such case. When task k is allocated to another robot, the second equation takes similar effect.

Inequality 7 is developed to constrain the start time of the first task in each robot's schedule. If $x_{kj} = 1$ and $\sum_{k \in N} y_{kij} = 0$, no task precedes task i in robot j's schedule, thus, the start time s_i should be larger than the travel time between initial position j and the location of task i ($|a_j - p_{in}|$).

Constraints 8-12 are developed to make the model generate plans that obey the general rules from Section 3. Constraint 8 is valid for all task pairs in P to ensure the fixed precedence orders in Figure 2. Constraint 9 and 10 make sure that each task pair in S is allocated to the same robot and each task pair in W is executed at the same place. These constraints correspond to the general rules 4, 5 and 6. Constraints 11 and 12 aim at limiting the number of tools and food on each robot respectively (general rule 1 and 2). The last constraint ensures that the variable t_{max} is equal to or greater than the finish time of any task. At last, the objective is to minimize t_{max}.



(a) CP result Problem 1.



(b) CP result Problem 2.

Figure 3: The generated plans for both problems using CP.

In order to compare both strategies, both MILP- and CP-model are implemented in Python and the problem is solved with the MILP-solver and CP-solver in the same optimization software Google OR-Tools (Da Col and Teppan, 2019). The generated plans using CP are shown in Figure 3. The makespan (total execution time) of the solution plans and the computational efforts are presented in Table 3.

Table 3: MILP and CP solution data for Pb. 1 and Pb. 2.

Method	Properties	Pb. 1	Pb. 2
MILP	schedule length [s]	128	189
	CPU time [s]	49429	3540
CP	schedule length [s]	128	189
	CPU time [s]	5.6	4.5

The total length of the plans generated by MILP and CP are identical, although the task sequence and allocation are different. The reason behind this is that both methods stop when they find one of the possible multiple optimal solutions. An optimal solution still can include idle time slots that appear due to the general constraints of the model (e.g. cross-schedule constraints). The high quality of the plans is also shown in Figure 3. The pauses between tasks are reduced as

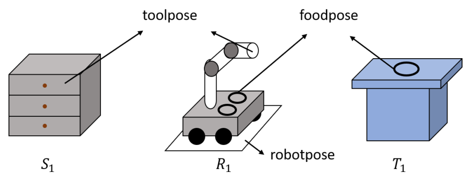


Figure 4: The examples of objects that belong to types ”robotpose, foodpose, toolpose”.

much as possible and the tasks are allocated to different robots appropriately, so that the total length of the schedules is minimized.

However, there is an important difference between the computational time using both strategies. The MILP-solver needs about 13h and 1h to solve the problems, while the CP-solver only 5.6s and 4.5s. This striking contrast is due to different optimization principles and the property of the problem that most variables are binary variables. During the search process, at each node, CP uses propagation to remove unnecessary values from the domain of variables. This strategy is especially efficient when dealing with binary variables, as their domain is only $\{0,1\}$ and the propagation engine only needs to check quite a small number of values in each iteration. As in our model nearly 99% of variables are binary variables, CP manages to solve the problem very fast. Nevertheless, this advantage is not exploited by the MILP solver, as MILP solves an integer linear program to get the optimum at each node. Therefore, the optimization strategy of MILP ignores the advantage of binary variables' small domain and treats them as normal continuous variables.

4.2 Temporal Planning

In this section, the temporal planning (TP) model written in PDDL and the plans generated for it with the POPF planner are presented and discussed.

This model consists of two parts: the domain and the problem model. The domain model can be further split into types, predicates, functions and actions.

As shown in Listing 1 four basic types of objects and three different types of positions (Figure 4) are defined (lines 2-3). Robotposes are places where robots can stand on (white squares). Tools and food can be put on toolposes and foodposes, respectively, and these types are defined to set limitations on the number of tools and food items that can be held at a time. In the predicates-part (lines 4-11), the first predicates describe the state of different objects and the following ones are used to determine whether the correct position, tool and food are chosen for each generic action. The function *move_cost* is used to calculate the travel time.

Listing 1: Excerpt from the PDDL domain file.

```
1  (:types
2  robot food tool position - object
3  robotpose foodpose toolpose - ↩
       position)
4  (:predicates
5  (at ?o1-object ?o2-object)
6  (free ?pose-position)
7  (not_acting ?robot-robot)
8  (uncooked ?f-food)(cooked ?f-food)
9  (food_for_ga_1 ?f-food)
10 (tool_for_food ?t-tool ?f-food)
11 (pos_for_ga_1 ?p-robotpose) ...
12 (:functions
13 (move_cost ?from ?to - robotpose))
14
15 (:durative-action move
16 :parameters
17 (?r - robot ?from ?to - robotpose)
18 :duration
19 (= ?duration(move_cost ?from ?to))
20 :condition
21 (at start((at ?r ?from)(free ?to)
22           (not_acting ?r))
23 :effect
24 (at start((not (at ?r ?from))
25           (not (not_acting ?r))
26           (not (free ?to))))
27 (at end((at ?r ?to)(not_acting ?r)
28         (free ?from))))
29 ...
```

Further on, in the domain file seven actions are defined: *move, load, unload, attach, detach, generic_action_1, generic_action_2*, corresponding to the actions that robots can execute. The setting for the *move* action is also shown in Listing 1, in lines 16-28. Its duration is calculated with the function *move_cost*. According to the defined conditions, it can only be executed when robot r is at start position $from$, in an idle state, and the goal place *to* is free. The effect in ”at start” means that all predicates $at, not\_acting, free$ are removed immediately after the action execution is started. At the end of this action, the state is modified such that robot r is at the target position *to*, in an idle state, and the start place $from$ is free. The other actions are defined similarly, but due to space issues are not presented here.

In the problem-model the objects, the initial state and the goal state of the planning instance are defined as depicted in Listing 2. Using the types from the domain, $food$, *tool* and *robot* objects are declared (lines 2-4). Further on, for each robot and for each location a fixed number of robotposes, foodposes and toolposes are set (lines 5-7). The initial state is set up with grounded predicates. In lines 9-10, it is defined that all robots are not executing any task and that all the food is uncooked. In lines 11-13, the initial positions for the robots, the tools and the food are fixed. After that, in lines 14-15, the foodposes and toolposes are connected to the corresponding robots or robotposes. All these positions should be free at first. The locations, food items and tools required for both *generic_actions* are specified in lines 16-17 and in line 18 all *move_costs* are set. In the goal state the cooked meals must be placed on tables T_3 and the tools must be returned to the tool-banks S_1 (lines 21-23). At last, the objective is set to minimize the total time (line 24).

Listing 2: The objects, the initial state, the goal state and the metric in the problem file.

```
 1  (:objects
 2  food1 food2 food3 - food
 3  tool1 tool2 tool3 - tool
 4  robot1 robot2 robot3 - robot
 5  robot1_fp1 ... - foodpose
 6  robot1_tp1 ... - toolpose
 7  cooker1 ... - robotpose...)
 8  (:init
 9  (not_acting robot1) ...
10  (uncooked food1) ...
11  (at robot1 initial_pose1) ...
12  (at food1 refrigerator1_fp1) ...
13  (at tool1 toolbank1_tp1) ...
14  (at robot1_fp1 robot1)
15  (free robot1_fp1) ...
16  (pos_for_ga_1 wt1)
17  (food_for_ga_1 food1) ...
18  (tool_for_food tool1 food1) ...
19  (=(move_cost cooker1 wt1) 8) ...
20  (:goal
21  ((at food1 table3_fp1) ...
22  (at tool1 toolbank1_tp1) ...
23  (cooked food1) ...))
24  (:metric minimize (total-time))
```

The main feature of this modelling approach is its high flexibility. The model only provides the initial and end state without offering a clear guidance of the intermediate steps of achieving the goal. This increases the difficulty of solving the problem, but makes the approach much more flexible. The generated plan for the first problem is shown in Figure 5. Its total lengths of 255s can be explained by the large proportion of time spent on travelling. The second drawback is that the tasks are not allocated properly. Only two tasks are assigned to robot R_2, who could take more tasks from R_1 to reduce the total working time. There are two possible reasons for the bad plan quality. Firstly, the heuristic in the planner POPF could be the cause. Similar issues are also reported in the paper introducing POPF (Coles et al., 2010). Secondly, as mentioned above, offering no clear guidance of the intermediate steps to achieve the goal makes the

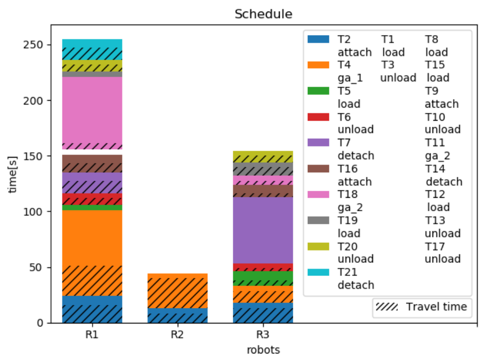


Figure 5: Problem 1, makespan: 255s.

number of possible states in the search space large and that increases the difficulty to solve the problems.

4.3 Comparison and Discussions

In this section, the three strategies are compared from three perspectives: model flexibility, CPU time and makespan.

The model flexibility referred here relates to the effort required to adapt the model for a different planning problem of the same scenario. This is a crucial factor, because there may be various types of planning problems in a real-world scenario. In order to evaluate the flexibility, a new planning problem based on the old one from Figure 2 is defined: the salad no longer needs to be processed and can be put directly on table T_3 or T_4. Starting with the CP and MILP model, the parameters and the sets P, S, W, T, F must be changed by hand, as the number of tasks is reduced and some task pairs from the problem's sets are removed. Furthermore, the indices of all parameters and the precedence graph must also be modified. On the other hand, for the TP model only the initial state in the problem-part needs to be modified: *cooked food1* is updated to *uncooked food1*. This modification is quite simple comparing to that of CP and MILP model. The reason behind this is that the mathematical model is based on the task precedence graph, which substantially provides a clear direction of how to solve the problem. Therefore, when the requirements of the problem change, the direction and all inputs related to it have to be modified. In contrast, the TP model offers little limitation and guidance on intermediate steps in the process of moving from the initial state to the end state. Hence, the change of intermediate steps can be ignored and only the initial- and goal state need to be adapted. In conclusion, the MILP and CP models are evaluated with low flexibility and TP model has very high flexibility.

Table 4: CPU time of runs using each strategy.

	CP	MILP	TP
CPU Pb. 1 [s]	5.6	49429	47
CPU Pb. 2 [s]	4.5	3540	3.2

Table 5: Makespan of plans generated with each strategy.

	CP	MILP	TP
Makespan Pb. 1 [s]	128	128	255
Makespan Pb. 2 [s]	189	189	306

The second factor is the CPU time, which is depicted for all problems and all planning approaches in Table 4. All computations were done on a computer with Intel® Core™ i5-6200U CPU at 2.30 GHz and 8 GB of RAM. For both problems, CP shows great and stable performance regarding the CPU time, whereas MILP is almost impractical. The computational time with TP is even lower than that with CP for problem 2, but is also relatively high for problem 1. This means that when the number of objects in the model increases, the CPU time with TP also grows rapidly. Hence, CP is the best choice among these three strategies with respect to the CPU time. Considering the makespans of the generated plans, as depicted in Table 5, CP and MILP are both highly efficient, while TP shows bad performance.

Moreover, for each strategy multiple ways to develop a model are possible and different models may lead to different results. This is the case for the TP approach, where the quality of the plan highly depends on the matching between the used heuristic and the model of the planning problem. However, if different CP or MILP based models are applied for the same problem, the CPU time may be different, but the generated plan is always the global optimal solution regarding makespan.

5 CONCLUSIONS

In this work the modelling and solving of a complex task planning problem with three planning strategies is presented. CP shows the greatest performance in terms of CPU time and plan quality (makespan of plans), while temporal planning is the best when considering the flexibility of the models. In the next steps, we aim to solve the model with further MILP, CP and TP planners. A further direction for our future work is to combine the CP and TP approaches to complement each other and thus being able to improve the overall performance on all three factors.

REFERENCES

Behrens, J. K., Lange, R., and Mansouri, M. (2019). A constraint programming approach to simultaneous task allocation and motion scheduling for industrial dual-arm manipulation tasks. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE.

Bezrucav, S.-O. and Corves, B. (2020). Improved ai planning for cooperating teams of humans and robots. In Cashmore, M., Orlandini, A., and Finzi, A., editors, *Workshop on Planning and Robotics (PlanRob) at International Conference on Automated Planning*.

Booth, K. E. C., Tran, T. T., Nejat, G., and Beck, J. C. (Jan. 2016). Mixed-integer and constraint programming techniques for mobile robot task planning. *IEEE Robotics and Automation Letters*, 1(1):500–507.

Coles, A., Coles, A., Fox, M., and Long, D. (2010). Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*, pages 42–49. AAAI Press.

Da Col, G. and Teppan, E. (2019). Google vs ibm: A constraint solving challenge on the job-shop scheduling problem. *Electronic Proceedings in Theoretical Computer Science*, 306:259–265.

Fox, M. and Long, D. (2003). pddl2.1 : An extension to pddl for expressing temporal planning domains. *Journal Of Artificial Intelligence Research*, 20:61–124.

Gervet, C. (2006). Constraints over structured domains. In Francesca Rossi Peter van Beek Toby Walsh, editor, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, pages 605–638. Elsevier.

Ku, W.-Y., Beck, J. C. (2016). Mixed integer programming models for job shop scheduling: A computational analysis. *Computers & Operations Research*, 73:165–173.

Land, A. H. and Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, 28:497–520.

Mokhtarzadeh, M., Tavakkoli-Moghaddam, R., Vahedi-Nouri, B., and Farsi, A. (2020). Scheduling of human-robot collaboration in assembly of printed circuit boards: a constraint programming approach. *International Journal of Computer Integrated Manufacturing*, 33(5):460–473.

Murty, K. G. (1985). *Linear and Combinatorial Programming*. R.E. Krieger.

Rossi, F., Peter van Beek, and Walsh, T., editors (2006). *Handbook of Constraint Programming: Introduction*. Elsevier Science.

Simonis, H. (2001). Building industrial applications with constraint programming. In Comon, H., Treinen, R., and Marché, C., editors, *Constraints in computational logics: theory and applications*, pages 271–309. Springer-Verlag Berlin Heidelberg, France.

Yi, J.-s., Ahn, M. S., Chae, H., Nam, H., Noh, D., Hong, D., and Moon, H. (2020). Task planning with mixed-integer programming for multiple cooking task using dual-arm robot. In *17th International Conference on Ubiquitous Robots (UR)*, pages 29–35. IEEE.