Case Study: AI Task Planning Setup for an Industrial Scenario with Mobile Manipulators

Stefan-Octavian Bezrucav, Malte Kaiser, Burkhard Corves
Institute of Mechanism Theory, Machine Dynamics and Robotics, RWTH Aachen University
{bezrucav, kaiser, corves}@igmr.rwth-aachen.de

Abstract

AI task planning approaches are increasingly used in projects with flexible processes, where it must be autonomously deliberated about the selection and scheduling of the actions for the involved actors. In order to implement AI task planning in such cases, the considered scenario must be modelled as a planning problem and a specific planning system for solving it needs to be chosen. However, multiple models for such a planning problem are possible and there exist numerous planning systems. The main challenge at this point is to choose those pairs (planning problem - planning system) that deliver the best results for that specific scenario.

In this paper, an industrial scenario derived from the use-cases of an EU-Project Sharework is targeted. In this scenario mobile manipulators, humans and Autonomous Grounded Vehicles (AGVs), share the working area in order to manipulate items and to execute tasks. After the presentation of a first model for the planning problem for this scenario and its limitation, three further alternatives are introduced. We evaluate the effects on the relevant metrics for such scenarios of these three different modelling variants for the required task planning problems, in combination with four state-of-the-art AI planning systems. The results suggest, that the most suitable combination is the one of a distance-based variant and the TFD planning system.

Introduction

In the last years, in the context of Industry 4.0, the transition to more flexible workspaces shared by humans and robots was started (Kadir, Broberg, and Da Souza Conceição 2018). In order to coordinate the actions of the involved actors in such applications and to react fast to new orders or changes in the environment powerful task planning approaches are required. They should be able to generate fast and autonomously plans through which the system can be brought in a state in which all goals are achieved. Given the required high flexibility, the focus is set on plan synthesis approaches and not only on scheduling approaches, that are ordering and allocating a set of predefined actions. Such plan synthesis strategies are AI planning techniques, as classical planning or temporal planning, that both determine the required actions to reach the set goals and schedule them (Ghallab, Dana, and Traverso 2016).

In order to use these planning strategies the characteristics of the considered scenario must firstly be abstracted and represented as a planning problem. This modelling step is a quite challenging one because no explicit rules for this process are given in the literature. Thus, the modelling process is usually an iterative one, where the experience of the developer plays a critical role.

On the other hand, several off-the-shelf planning systems for solving the generated planning instances are available in the community. Most of them follow the same strategies, but they still differ in the way in which they combine different basic approaches, as heuristics functions, for their search algorithms. These differences can have an impact on the quality of the generated results for different models of the same planning problem.

In this paper we consider an exemplary industrial scenario, that is derived from three of the four use-cases of the EU-funded project Sharework project[1]. This is a generic scenario in the sense that it combines standard environment elements, such as benches, tools and items and typical actors, such as mobile manipulators executing common tasks, such as manipulating objects. Here, humans and AGVs cooperate in order to process a set of items and execute a set of generic tasks. If required, this scenario can be further specialized to a high variety of concrete scenarios, for example, a train door assembly scenario.

As humans are directly involved in these processes, it should be ensured that their acceptance for a high-level coordination system dictating them the actions to be carried out is given. In order to sustain their acceptance, long idle times, for example for the planning process, are not allowed. Thus, we impose a hard deadline for solving the planning problems of 60 seconds.

The main contribution of this paper is the extensive experiment done for different task planning models for the industrial scenario and different planning systems, given the hard planning time constraints. The interpretation of the obtained results may give the first hints about a suitable AI task planning setup for this kind of scenarios.

In the following sections the industrial scenario and a comprehensive model for its corresponding task planning instances are introduced. The model is encoded in a language that can be understood by the considered planning systems. In our case, this is the Planning Domain Defini-

[1]https://sharework-project.eu/

tion Language (PDDL) (McDermott et al. 1998). We have selected four planning systems to solve the initial planning instances for the industrial scenario. These planning systems are also used for the further analysis. They all have delivered good results in the Temporal Track of the 2018 International Planning Competition (Coles et al. 2018) and were available online at the time point when this work was done. They are POPF, OPTIC, TFD and TFLAP. Given the results obtained with these planning systems, we present in a short analysis the limitations of the initial model for the planning problems of the industrial scenario and introduce three alternatives. In the last sections we present the setup of the main experiment with the three modelling alternatives and four planning systems and the obtained results. We conclude this paper with a comprehensive interpretation of these results.

Background

AI or Automatic task planning enables computer systems to deliberate about their actions rather than only act reactively by using if-else-trees or finite state machines. In classical task planning the planning domain is modelled as a state transition system $\Sigma = (S, A, \gamma, cost)$, where:

- S is a finite set of states described by grounded atoms ϕ that the system can be in.
- A is a finite set of actions, with preconditions and effects, that can be performed in the system.
- $\gamma : S \times A \rightarrow S$ is a function that maps a state and an action to a state.
- $cost : S \times A \rightarrow [0, \infty)$ is a function assigning a cost to a combination of a state and an action.

The planning problem P is defined as the triple $P = (\Sigma, s_0, g)$ with Σ being a state transition system, s_0 the initial state of the system and g a set of atoms that are supposed to be true in a goal state. (Ghallab, Dana, and Traverso 2016)

A plan $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ is a sequence of actions that transforms the initial state to a goal state and its cost is the sum of the costs of the actions in that plan. The objective of classical planning is to find a valid plan π with minimal cost for the given problem P.

An extension of classical task planning is the temporal task planning. It is capable of considering the temporal dimension of actions while reasoning about a plan. This is implemented through extending the action representation from classical task planning by assigning a duration to the action and defining points in time during this duration at which the conditions must hold and at which the effects are applied. (Ghallab, Dana, and Traverso 2016)

This yields the representation of a simple durative action da as the tuple:

$$da = (c_{\vdash}, c_{\leftrightarrow}, c_{\dashv}, a_{\vdash}, a_{\dashv}, d_{\vdash}, d_{\dashv}, \Delta) \qquad (1)$$

where

- $c_{\vdash}$, $c_{\leftrightarrow}$, $c_{\dashv}$ are the conditions that must hold at the beginning, during the entire execution and at the end of the durative action.
- $a_{\vdash}$, $a_{\dashv}$ are the add effects that are applied at the beginning and at the end of the durative action.
- $d_{\vdash}$, $d_{\dashv}$ are the delete effects that are applied at the beginning and at the end of the durative action.
- Δ is the duration of the durative action.

In this representation the effects are separated into two different sets. The add and the delete effects. Add effects set atoms to true and delete effects set atoms to false. (Coles et al. 2010)

In order to solve temporal task planning problems several planers that use different heuristics in different setups exist. POPF is one of them. It was developed by Coles et al. and it combines grounded forward search with linear programming in order to solve planning problems with continuous linear numeric change. Moreover, is able to handle temporal planning problems with required concurrency. This is implemented through the planner scheduler interaction. Its most important characteristics is the approach of achieving a partial ordering of the actions by delaying the commitment to action choices during forward search. The information from the partial ordering is also used to modify the FF heuristic (Hoffmann and Nebel 2001). This modified heuristic is then used to guide the search. (Coles et al. 2010)

OPTIC stands for Optimizing Preferences and TIme-dependent Costs and is based on POPF, but unlike POPF it uses a mixed integer program. The mixed integer program is used to optimize preferences during the construction of the plan. Similar to POPF, OPTIC also uses a temporal relaxed planning graph heuristic for search guidance. Additionally, an admissible heuristic is used for search pruning. (Benton, Coles, and Coles 2012).

Further on, TFLAP is a temporal forward partial-order planner that works similar to OPTIC but it follows the partial order planning approach more closely. It does not commit to an action ordering if that is not required. This way TFLAP is able to insert actions at any point of the plan in contrast to OPTIC where actions are only inserted in the frontier state. Thus TFLAP achieves a higher flexibility and possibly higher plan quality. However, this results in a higher computational effort. TFLAP uses an A* search in plan-space guided by the FF heuristic and a landmark heuristic. (Sapena, Marzal, and Onaindia 2018)

Last but not least, TFD stands for Temporal Fast Downward. It is a temporal planning system that can handle temporal problems with numeric fluents. TFD uses the context-enhanced additive heuristic by (Helmert and Geffner 2008) to guide its A* search in the space of time-stamped states. The context-enhanced additive heuristic is an inadmissible heuristic defined for sequential planning tasks. In order to use this heuristic for temporal planning the durative actions have to be modified. The durative actions are replaced by non-temporal instant actions. These instant actions are either compressed actions, start actions, wait actions or actions derived from the application of logical axioms. TFD is able to find plans for some problems which require concurrency in the plan but not for all kinds of concurrencies. (Eyerich, Mattmüller, and Gabriele Röger 2009)

Related Work

Automated task planning was and is used in many applications. For each of them, the considered planning problem must be modelled, usually in Planning Domain Definition Language (PDDL), and solved with a corresponding planning system. Unfortunately, only very few works in the literature present the PDDL modelling process of the planning problems for industrial scenarios. Some works describe this process for other domains of interest or for other automated planning concepts, as timeline-based planning, such as in (Borgo et al. 2019).

The medication and the physical activity units required by a patient depending on the level of pain he or she is experiencing was modelled as a planning problem in (Alaboud and Coles 2019). They have used the more powerful version of the PDDL language, PDDL+, to model both discrete events and continuous processes. Alaboud and Coles also compared the results obtained for their planning problem with two PDDL+ solvers: OPTIC (Benton, Coles, and Coles 2012) and ENHSP (Scala et al. 2016).

A PDDL+ model for the path planning problem of UAVs was presented in (Kiam et al. 2018). Kiam et al. have also used the ENHSP planer (Scala et al. 2016) to generate reliable plans. Cashmore et al. have formulated their planning problem for the mission control of AUVs as a temporal problem in PDDL2.1 (Cashmore et al. 2014) and have used only one planner to solve it: POPF (Coles et al. 2010).

In (Crosby et al. 2017) an industrial kitting scenario, where more robots are involved, is considered. For the high-level control of the actors two modules, a *Mission Planner* and a *Task Planner* are used. In the task planning process the robots' skills and the state of the environment are mapped to PDDL structures and together with the goals are sent to the automated planner Fast Downward (Helmert 2006). Crosby et al. do not require specialized models of the planning problems solved in the Task Planner module. Furthermore, they also do not compare the results obtained with different automated planning systems, as their models do not bring the Fast Downward planner to its limits. (Crosby et al. 2017)

In (Huckaby, Vassos, and Christensen 2013) and (Bezrucav and Corves 2020) comprehensive models of planning problems for manufacturing or industrial scenarios are presented. In the first work the modelling of the planning problems in PDDL is done based on the representation of the considered system and of the processes in the System Model Language (SysML). In the second work the PDDL modelling is done by an expert, considering the requirements of the different use-cases an European project. Furthermore, in both works only the OPTIC (Benton, Coles, and Coles 2012) planner was used to solve the planning instances.

In this work, we go beyond the related work and conduct an experiment with different modelling approaches and different planning systems. Based on the results, we offer a set of suggestions about the most suitable modelling process for an industrial scenario with multiple mobile manipulators. In the next section this scenario is described in more detail.

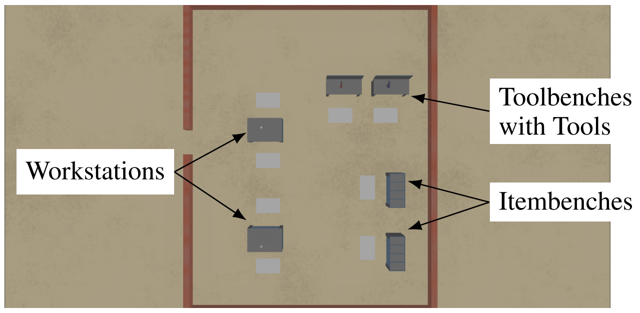

Figure 1: Overview of the Gazebo simulation of the industrial scenario

Scenario and Planning Problem

In this section the scenario and the corresponding planning problems are presented.

Scenario

In this work, an industrial scenario that consists of an environment, tools, items and two types of actors in the form of mobile manipulators is considered. The characteristics of this scenario were derived from three of the four use-cases of the EU-Project Sharework. It combines those features that are common in the three use-cases and, therefore, typical for such an industrial scenario.

Figure 1 shows an overview of the environment. It models a shop floor containing two toolbenches, two itembenches and two workstations with two workplaces each. The locations from where the benches and workstations can be accessed are marked on the floor with grey rectangles. Movement is possible in the entire area and each location can be reached from every other location. The toolbenches serve to store the tools at the beginning of each simulated working cycle. The tools need to be attached by the agents in order to perform tasks or process items.

The items are stored at the itembenches and must be transported to the workstations for processing. The workstations are just plain tables where items can be unloaded and processed or tasks can be executed.

The involved actors are of type *human* and *AGV* and they are both considered mobile manipulators, as they can navigate in the environment and execute specific trajectories with one or two of their arms. The agents can execute the following actions:

- *move* between two poses.
- *load/unload* items from/to any bench.
- *attach/detach* tools from the toolbenches.
- *process_item* with a tool at a specific workstation.
- *execute_task* with a tool at a specific workstation.

All actions beside the first one must be executed at the locations of the benches or workstations. Further on, only through the *move* action the actors can navigate between these locations. Thus, the *move* action can be a bottleneck in the planning process.

Listing 1: Excerpt from the PDDL domain file with the move action

```
1   (:durative-action move_agent
2   :parameters (?agent - agent ?from ?to ↩
        - agentpose)
3   :duration (= ?duration 2)
4   :condition
5   (and (at start (at ?agent ?from))
6   (at start (not_acting ?agent))
7   (at start (free ?to)))
8   :effect
9   (and (at start (not (at ?agent ?from))↩
        )
10  (at start (not (free ?to)))
11  (at start (free ?from))
12  (at start (not (not_acting ?agent)))
13  (at end (at ?agent ?to))
14  (at end (not_acting ?agent))
```

Furthermore, these actions are modelled such that the following requirements are met:

- Only one actor can be at a specific location at the same time.
- One actor can not execute two actions at the same time
- Different actors can be engaged in different actions at the same time.
- The tasks must be executed at the workstations with specific tools.
- The items must be moved from the itembenches to the workstations and be processed with specific tools.

The planning problem related to this scenario is presented in the next section.

Planning Problem

For the above industrial scenario a first task planning problem is described in PDDL. The original model contains the definition of the required PDDL types (e.g. locations, agents, etc.), PDDL predicates (e.g. $free\ ?location\ -\ location$, $at\ ?object1 - object\ ?object2\ -\ object$, etc.) and of the actions presented above. Some special characteristics of the model are the different positions types *agentpose*, *toolpose* and *itempose*, each being defined for specific objects. In this way, a more flexible allocation of the objects to the poses is achieved. For example, an *agent* that has an *toolpose* can carry a *tool* with it, while another *agent* that has only one *itempose* can carry only an *item*.

In Listing 1 the basic PDDL description of the *move* action is presented. The action is modelled with a fixed duration of 2 time units (line 2). It can be executed when the *from* and *to* locations correspond to the correct benches or workstations, the *agent* is at the beginning at the *from* location, is *not acting* and the *to* location is free (lines 5-9). If the execution of the action can be started, the *from* position is marked as free (line 13), as the agent is not anymore there

Listing 2: Excerpt from the PDDL domain file with the execute task action for the human actor

```
1   (:durative-action execute_task_human
2   :parameters (?agent - human ?agentpose↩
        - agentpose ?task - task ?tool - ↩
        tool ?toolpose - toolpose)
3   :duration (= ?duration 2)
4   :condition
5   (and (at start (at ?agent ?agentpose))
6   (at start (not_acting ?agent))
7   (at start (at ?tool ?toolpose))
8   (at start (at ?toolpose ?agent))
9   (at start (tool_for_task ?tool ?task))
10  (at start (pose_for_task_execution ? ↩
        agentpose ?task))
11  (at start (task_pending ?task)))
12  :effect
13  (and (at start (not(task_pending ?task↩
        )))
14  (at start (not (not_acting ?agent)))
15  (at end (task_executed ?task))
16  (at end (not_acting ?agent))
```

(line 11). Concurently, the predicate marking the *to* location as free is deleted (line 12). Further on, the special construct with the *not_acting* predicate in lines 14 and 16 ensures that the *actor* can execute only one action at a time.

Different to the *move* action, the *execute_task_human* action is carried out at the specific location *agentpose*, as presented in Listing 2, line 5. Furthermore, the action can start when the *agent* has the corresponding *tool*, the *tool* is the one required for the given *task* and the *task* was not yet executed (lines 6-11). When finished, the *task* is marked as executed (line 15). The same construct with the *not_acting* predicate impose the execution only of this action at a time by the *human* (lines 14 and 16). All other actions beside the *move* one are modelled similarly to the *execute_task_human* action.

In each planning problem at least two agents (one human and one AGV or two AGVs) are considered and at least one task must be executed or one item must be processed.

Limitations

One important issue of the current modelling approach is that the plans found by all considered domain independent automatic task planning algorithms for quite simple planning instances, where only two to three tasks must be executed or items must be processed by the two actors involved, are suboptimal. Most of them contain a lot of unnecessary *move* actions. Listing 3 shows excerpts taken from such a plan. This plan was computed by the automatic task planning algorithm POPF.

It can be seen that the agent *agv1* performs a move action from the location *tb1_pose* to the location *wb21_pose* at time 6. The next action performed by *agv1* is another move action to location *ib1_pose* at time 16. *agv1* does not perform any action at location *wb21_pose*. Therefore it is not necessary for *agv1* to move to the location *wb21_pose*. Ad-

Listing 3: Excerpts from a plan obtained with the initial model

```
6.01:(move_base agv2 tb2_pose ↩
    wb22_pose)
6.01:(move_base agv1 tb1_pose ↩
    wb21_pose)
16.02:(generic_action_2 agv2 tool2 ↩
    task8 wb22_pose)
16.02:(move_base agv1 wb21_pose ↩
    ib1_pose)
26.03:(load_item_on_agv agv1 item4 ↩
    ib1_pose nr0 nr1 nr2) ...
```

ditionally, the other agent *agv2* does not occupy the target location *ib1_pose* of the second move action of *agv1* during the relevant time. These types of issues appear more often in longer plans. This means that the computed plans lead to unnecessarily long execution times.

New Modelling Variants

In order to tackle the issues of generating suboptimal plans with unnecessary move actions from the modelling side, three different approaches are implemented:

1. Consider the distance moved by the agents and use it as metric for the planning problem (referred to as the *basic* approach).
2. Include a special precondition for move actions that is only true if the move action is reasonable (referred to as the *reasonable* approach).
3. Completely eliminate the explicit move action by merging it with the other actions (referred to as the *combined* approach).

Basic Approach The first approach is implemented using *numeric fluents* in the PDDL model. A variable called *moved_distance* is increased after each move action by the geometric distance between the initial and the target location. A *distance* is a function of two locations, the initial and the goal pose. This function is evaluated in the effects of the move action according to the parameters of the move action. The result from the *distance* function is then used to increase the value of the *moved_distance* (see Listing 4) which is also set as the optimization function of the planning problem.

Listing 4: Excerpt from the PDDL domain file with integrated *moved_distance* variable

```
(:durative-action move_agent
:parameters (?agent - agent ?from ?to ↩
    - agentpose)
[...]
:effect (and
[...]
(increase (moved_distance) (distance ?↩
    from ?to)) ) )
```

Reasonable Approach The second approach for preventing unnecessary move actions is based on the idea that there are situations where it is *reasonable* for an agent to move and other situations where it is not. Through the use of a special precondition the movement is made only possible in situations where it is *reasonable*. The challenge with this approach is to find as general specifications of such situations as possible. The reason is that by preventing actions in certain situations the space of possible solution plans is reduced. If the solution space is reduced too much, the plans lose flexibility and it might even lead to the problem becoming unsolvable. For example, consider a situation from the industrial scenario where an AGV is supposed to transport three items from a location A to another location B. The AGV is capable of transporting two items at once, thus it is *reasonable* to load two items at location A before moving the first time to location B. This would reduce the total number of move actions and thereby the makespan. However, if it were defined that movement is only *reasonable* when the AGV is completely loaded or unloaded, there would not exist a solution to the example problem. The AGV would not be allowed to move from location A to B for the second time, since only one item remains at location A. In the end, this only leads to the conclusion that movement is *reasonable* whenever the AGV is loaded as much as is *reasonable* in that current state. However, determining which amount of items to load is *reasonable* in a certain state is a planning problem itself and can not generally be answered.

Therefore, the second approach is reduced to one restriction. It is defined to be *reasonable* for an agent to move only if the last action performed by this agent was not a move action. This restriction is fairly general, yet still there exist scenarios when it does not apply. Consider a scenario with only two locations and two agents, for example. One of the agents has to perform tasks at both locations while the other agent already finished its tasks. However, the latter agent still has to move in order to unblock its location for the other agent. This example is rather artificial and it does not apply to most industrial scenarios. Therefore, the proposed restriction might still create good results for planning problems in industrial scenarios. The implementation is straightforward. A new predicate with an agent as variable is introduced in the PDDL domain (see Listing 5). The predicate controls whether it is reasonable to move for the agent in question. This *reasonable* predicate is added to the preconditions and the delete effects of the move action (see Listing 5) and to the add effects of all other actions.

Thereby, whenever any action except a move action is performed by an agent the *reasonable* predicate becomes true for this agent. Whether more than one of those actions are performed has no additional effect on the *reasonable* predicate. Performing a move action is only possible for an agent when its *reasonable* predicate is true and has the effect that the predicate becomes false. Thus, it is not possible for a plan to contain two or more consecutive move actions for one agent.

Listing 5: Excerpt from the PDDL domain file with *reasonable_to_move* predicate

```
(:predicates
(reasonable_to_move ?agent - agent))
[...]
(:durative-action move_agent
[...]
:condition  (and
[...]
(reasonable_to_move ?agent) )
:effect (and
[...]
(not (reasonable_to_move ?agent)) ) )
```

Combined Approach The last approach also aims at preventing consecutive move actions but with a different technique. The explicit move action is removed from the PDDL model and integrated into the *other* actions. An *other* action means here every action that is not a move action. For example, a *load* action becomes a *move and load* action. In order to combine two actions, their preconditions and effects have to be merged. During this merge the preconditions of the second action that are satisfied by the effects of the first action have to be omitted in the combined action. For example, a load action has the precondition that the agent performing the load operation is at a location adjacent to the loaded item. If this load action is combined with a move action, the mentioned precondition can be removed since the move action satisfies it through its add effects. The conditions and effects of the combination of a move and an *other* action therefore are as follows:

- combined condition: $cond = c_{\text{move}} \cup (c_{\text{other}} \setminus a_{\text{move}})$
- combined add effects: $add = (a_{\text{move}} \setminus d_{\text{other}}) \cup a_{\text{other}}$
- combined delete effects: $del = (d_{\text{move}} \setminus a_{\text{other}}) \cup d_{\text{move}}$

For simplicity the move and the *other* action are regarded as instant actions considering only their total conditions c_x, add effects a_x and delete effects d_x with $x = \text{move}$ for the move action and $x = \text{other}$ for the *other* action. However, the concept is applicable to durative actions with minor adjustments as well.

Furthermore, when combining the move action with every *other* action, it is important to consider states where only an *other* action has to be performed without a move action because the agent already is at the right location. In these states it is not possible to use the combined action with a dummy movement where the initial and the goal location are equal. The reason is, that such a move action would require the agent performing it to be at the same location that is also required to be free. This constitutes a logical contradiction. Therefore, all the *other* actions also have to be included in the domain model in their single form in addition to the combined variants.

Planner	OPTIC	POPF	TFD	TFLAP
Failed plans[%]	9.78	7.53	1.58	12.56

Modelling variant	*basic*	*reasonable*	*combined*
Failed plans[%]	4.66	11.25	7.68

Optimization variable	*makespan*	*distance*
Failed plans[%]	7.01	8.73

Table 1: Percentage of failed plans for the different values of the independent variables.

Experiment and Results

In order to find out the most suitable modelling method and automated planner an extensive experiment was conducted for the above presented scenario. Firstly the experiment was designed, afterwards results were generated and interpreted.

Design of Experiments

Three independent inputs variables for the experiment and their values are set as follows:

- The approach with which the PDDL domain and problem are modelled (Modelling variant): *basic*, *combined*, *reasonable*
- The metric which is supposed to be optimized (Optimization variant): *makespan*, *distance*
- The planning system used (Planning system): OPTIC, POPF, TFD, TFLAP

Beside those variables the number and types of agents (e.g. one AGV and one human, two AGVs, etc.), as well as the number of items and tasks (e.g. 1 task, 1 item, 3 tasks and 3 items, etc.) were varied, in order to give more generality to the obtained results. The dependent variables measured in the experiment are properties of the computed plans and they are:

- Validity: Does the planner solve the planning problem in 60 seconds
- Makespan: If generated, what is the duration of the plan
- Moved distance: If generated, what is the total moved distance according to the actions from the plan

In total 3 (Modelling variants) * 2 (Optimization variants) * 4 (Planning systems) * 252 (variations of the agents, items and tools) = 6048 tests were run using a computer with an Intel® Core™ i7-7700 processor and 31.3 GB of memory.

Presentation of the Results

In the following, the effects on the percentage of failed plans, on the average makespan and on the average moved distance are presented. Each value from the following tables and figures is a mean value for a specific instantiation of a independent input, given the test results for all other variations of the other independent inputs and the agents, items and tasks variations. For example, the values of the dependent variables obtained for the planner OPTIC are mean values over all tests executed for all possible combinations of the three modelling variants, the two optimization variants and

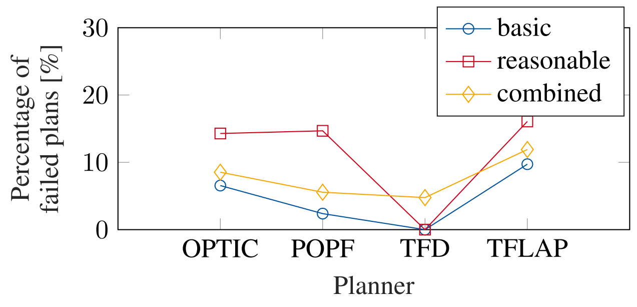

Figure 2: Interaction of the planning system and the modelling variant regarding the percentage of failed plans

Planner	OPTIC	POPF	TFD	TFLAP
Average makespan[-]	28.49	30.02	25.47	27.73

Modelling variant	*basic*	*reasonable*	*combined*
Average makespan[-]	31.13	32.52	31.54

Optimization variable	*makespan*	*distance*
Average makespan[-]	30.15	29.84

Table 2: Average makespan for the different values of the independent variables.

the 252 variations related to the different number of tasks, items and agents.

In Table 1 the percentage of planning problems for which no valid plan was generated depending on the different values of the independent variables is depicted. It can be observed that the planner TFD, the *basic* modelling variant and the planning instances for which it was optimized with respect to the *makespan* return the best results in each of their categories.

The interactions between the four planning systems and the two modelling variants with respect to the percentage of failed plans is depicted in Figure 2. The TFD planner returns the best results for all modelling variants.

In the following, the unsuccessful plans can be eliminated. In order to generate consistent results, all of the following results are an average of the dependent variables over all planning problems for which all four planning systems have generated a valid plan.

When analysing the next experiment output, the average makespan, it can be observed from the values of Table 2 that the planner TFD returns once again the best results in its category and the *basic* modelling variant is the most suitable one. Further on, the average makespan is almost the same for both optimization methods.

The interaction of the planning system and the modelling variant regarding the average makespan is depicted in Figure 3. The main acknowledgement is related to the planner TFD for which the best makespan values are obtained independent of the modelling variant used.

The last output of the experiment which is evaluated is the average moved distance. The obtained values are depicted in Table 3. In this case the planner TFLAP and the *reasonable* modelling variant return the best results. Further on, for both

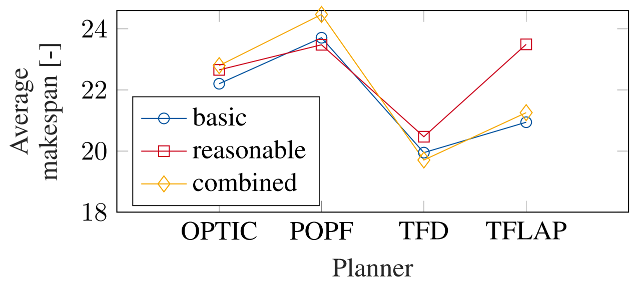

Figure 3: Interaction of the planning system and the modelling variant regarding the average makespan

Planner	OPTIC	POPF	TFD	TFLAP
Average moved distance[-]	35.11	37.89	28.34	26.86

Modelling variant	*basic*	*reasonable*	*combined*
Average moved distance[-]	37.33	32.25	35.14

Optimization variable	*makespan*	*distance*
Average moved distance[-]	34.34	34.37

Table 3: Average moved distance for the different values of the independent variables.

optimization variants almost the same values are obtained.

The results of the interactions between the planning system and the modelling variants are depicted in Figure 4. It is clear that the most appropriate planner is TFLAP, followed closely by TFD. Moreover, the *reasonable* approach is the one that delivered the results of the highest quality for all planning systems.

Interpretation of the Results

In this section the effects of the independent variables (modelling variant, optimization variant, planning system) on the dependent variables (validity, makespan and moved distance) are discussed.

At first, the effect of the modelling variant on the percentage of failed plans is investigated. A first important conclusion that can be made, is that the approaches for reducing unnecessary move actions generally lead to a higher percent-

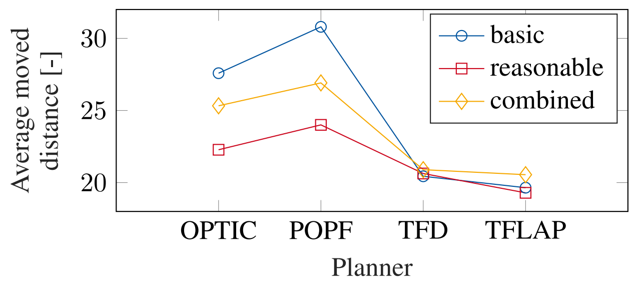

Figure 4: Interaction of the planning system and the modelling variant regarding the average moved distance

age of failed plans (see Figure 2). The reason is probably that it is harder for the planning systems to compute a solution plan when using these modelling approaches, especially under the hard time constraints of only 60 seconds. However, the effect is more significant for the planning systems OPTIC and POPF while it is smaller for the planning systems TFLAP. TFD even manages to find a valid plan for all planning problems independent from the modelling variant, except for the unsolvable ones. A possible reason for this may be, that TFD as well as TFLAP use other heuristic functions to guide their search. This might make them less susceptible for dead ends while planning. These dead ends may be a reason why the *reasonable* approach leads to the highest percentage of failed plans. If in the *reasonable* approach a move action is planned that leads to an agent being at a location where the agent can not perform any *other* action, this agent is blocked for the entire rest of the plan. If all agents become blocked the search results in a dead end. In comparison, these kinds of dead ends are not possible when using the *basic* approach, because the agent could simply move again. When using the *combined* approach these dead ends are also not possible, since an agent can always move, as long as there are actions it can perform at its destination. When the planning system encounters a dead end while searching for a plan, it needs to backtrack and restart the search from an earlier state resulting in a longer search time. Thus, the dead ends in the *reasonable* approach might be responsible for the higher percentage of failed plans when using this modelling variant.

Further on, it can be observed that the approaches to reduce unnecessary move actions through the modelling of the PDDL planning problem actually results in a lower average moved distance (see Figure 4). Thus, those approaches not only prevent sequential unnecessary move actions but also reduce the total averaged moved distance. However, as discussed in this section this improvement comes at the price of higher planning complexity and therefore more failed plans.

The optimization for the *distance* variable was introduced in order to reduce unnecessary move actions by reducing movement in general. However, as can be seen in Table 3 it does not lead to a decrease in the average moved distance compared to optimizing for the *makespan*. On the other hand, the optimization of the *distance* leads to a higher percentage of failed plans (see Table 1). Thus, using a numeric fluent to track the *distance* and setting this value as the metric that is supposed to be minimized in the planning process fails to have the desired effect of reducing the moved distance. A possible reason may be that the planning systems might require more time to optimize their solution plans as desired. Or the complexity of the planning problems is already too high for the planning systems and they do not manage to perform an effective optimization.

As already mentioned in the previous sections, the planning system used to solve the PDDL planning problem has various effects on the planning process. Moreover, the effects of the other independent variables strongly interact with the choice of the planning system. All in all, TFD shows the best performance with the lowest percentage of failed plans (see Table 1), the lowest average makespan (see Table 2) and only outperformed by TFLAP regarding the average moved distance (see Table 3). The latter being the only drawback of TFD, since it is not capable of minimizing the moved distance. The reason for this result might be that TFD is only able to handle temporal planning problems with a limited requirement of concurrency. However, since these features are not necessary in the current planning problems in the industrial scenario TFD is able to outperform the more complex planning systems, which offer more features at the price of a more complex planning process.

Conclusion and Future Work

In this paper a generic industrial scenario derived from real use-cases with an initial model for its corresponding task planning problems are introduced. Given this scenario configuration and the constraints, for example that at a specific time point each actor can execute only one action and can be only alone at a location, suboptimal plans are generated by all selected planning systems. In order to tackle this issue different modelling variants are introduced. In a comprehensive experiment they are tested in combination with different planning systems. The obtained results are analysed given a set of relevant metrics. The interpretation of these results concludes this paper.

The generated results may give a first direction to follow when setting up an AI task planning setup for these kind of industrial applications, with *move* and specialized actions and with multiple mobile manipulators agents that are active at the same time. The suggested choice is the *basic* planning model in combination with the TFD planning system. Further on, the set of experiments already contains many variations of the planning problem for the considered scenario, given by the different number and types of items, tasks and agents, as well as by the various initial states and goals. Thus, these results can also be used as a preliminary proof of concept for the deployment of the suggested modelling variant and planning system in real use-cases.

As a next step, the results of the experiment can be further analysed, especially in order to determine further synergies between the modelling approaches and the planning systems. Further on, the authors think to slightly modify some of the planning systems in order to add spatial information as guidance in their search strategies.

Acknowledgements

This article was created as part of the Sharework project, that has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 820807.

References

Alaboud, F. K.; and Coles, A. 2019. Personalized Medication and Activity Planning in PDDL+. In Benton, J.; Lipovetzky, N.; Onaindia, E.; Smith, D. E.; and Srivastava, S., eds., *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling*, 492–500. AAAI Press.

Benton, J.; Coles, A.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceeding of the Twenty-Second International Conference on Automated Planning and Scheduling*. AAAI Press. ISBN ISBN 978-1-57735-562-5.

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., eds., *Workshop on Planning and Robotics (PlanRob) at International Conference on Automated Planning*.

Borgo, S.; Cesta, A.; Orlandini, A.; and Umbrico, A. 2019. Knowledge-based adaptive agents for manufacturing domains. *Engineering with Computers* 35(3): 755–779. ISSN 0177-0667. doi:10.1007/s00366-018-0630-6. PII: 630.

Cashmore, M.; Fox, M.; Larkworthy, T.; Long, D.; and Magazzeni, D. 2014. AUV mission control via temporal planning. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 6535–6541. IEEE. ISBN 978-1-4799-3685-4. doi:10.1109/ICRA.2014.6907823.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In Brafman, R., ed., *Proceedings of the 20th International Conference on Automated Planning and Scheduling*, 42–49. AAAI Press. ISBN 978-1-57735-449-9.

Coles, A.; Coles, A.; Martinez, M.; and Sidiropoulos, P. 2018. International Planning Competition 2018. URL https://ipc2018-temporal.bitbucket.io/.

Crosby, M.; Petrick, R. P. A.; Rovida, F.; and Krueger, V. 2017. Integrating Mission and Task Planning in an Industrial Robotics Framework. In Barbulescu, L., ed., *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling*, 471–479. AAAI Press. ISBN 978-1577357896.

Eyerich, P.; Mattmüller, R.; and Gabriele Röger. 2009. Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning. In Gerevini, A.; Howe, H.; Cesta, A.; and Refanidis, R., eds., *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling*, 114–121. ICAPS and International Conference on Automated Planning and Scheduling, AAAI Press. ISBN 978-1-57735-406-2. URL https://aaai.org/ocs/index.php/ICAPS/ICAPS09/paper/view/754/1101.

Ghallab, M.; Dana, N.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press. ISBN 9781107037274.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26: 191–246. doi:10.1613/jair.1705.

Helmert, M.; and Geffner, H. 2008. Unifying the Causal Graph and Additive Heuristics. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling*, 140–147. AAAI Press. ISBN 978-1-57735-386-7.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System. In *Journal of Artificial Intelligence*, 253–302. AI Access Foundation and Morgan Kaufmann Publishers.

Huckaby, J.; Vassos, S.; and Christensen, H. I. 2013. Planning with a task modeling framework in manufacturing robotics. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5787–5794. IEEE. ISBN 978-1-4673-6358-7. doi:10.1109/IROS.2013.6697194.

Kadir, B. A.; Broberg, O.; and Da Souza Conceição, C. 2018. Designing human-robot collaborations in Industry 4.0. In *Proceedings of the DESIGN 2018 15th International Design Conference*, Design Conference Proceedings, 601–610. Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Croatia and The Design Society, Glasgow, UK. doi:10.21278/idc.2018.0319.

Kiam, J. J.; Scala, E.; Ramirez, M.; and Schulte, A. 2018. Using a Hybrid AI-Planner to Plan Feasible Flight Paths for HAPS-Like UAVs. In Finzi, A.; Karpas, E.; Nejat, G.; Orlandini, A.; and Srivastava, S., eds., *Workshop on Planning and Robotics (PlanRob) at International Conference on Automated Planning*, 55–64.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language. URL https://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf.

Sapena, O.; Marzal, E.; and Onaindia, E. 2018. TFLAP: a temporal forward partial-order planner. URL https://ipc2018-temporal.bitbucket.io/planner-abstracts/team2.pdf.

Scala, E.; Haslum, P.; Thiebaux, S.; and Ramirez, M. 2016. Interval-Based Relaxation for General Numeric Planning. In Kaminka, G. A., ed., *Proceedings of the Twentysecond European Conference on Artificial Intelligence*, number 285 in Frontiers in artificial intelligence and applications, 655–663. IOS Press. ISBN 978-1-61499-672-9.