Automated Planning: Theory and Practice Report

Noemi Canovi (230487)
University of Trento
Assignment for Automated Planning: Theory and Practice

***Abstract*—In this report is explained the student project addressing the assignment of Automated Planning: Theory and Practice, course of the Master in Artificial Intelligent Systems at the University of Trento, a.y. 2022-2023, run by prof. Roveri. Here, a first brief introduction to the task is given, followed by a thorough explanation. Then, design choices and additional assumptions are listed. Finally, some trials and their results are shown.**

The implementation of the assignment can be found at https://github.com/NoeCanovi/AutomatedPlanningAssignment.

I. Introduction

Automated planning is a class of artificial intelligence algorithms which task is to schedule an activity by creating an action sequence able to reach a goal starting from an initial setting. It is applied in a broad spectrum of applications, such as space missions, logistics and video games.

Here, we see a typical emergency logistic example, in which a set of robotic agents' task is to aid a group of people that have been injured and are stuck in a particular location.

The assignment is divided into five sub-problems, explained properly in the following sections, each highlighting different aspects of automated planning. A brief summary can be seen in Table 1.

Sub-problem	Keywords	Language
1		PDDL
2	carrier, place	PDDL
3	HTN, tasks	HDDL
4	durative actions	PDDL
5	durative actions, plansys2	PDDL, Python, C++

Table 1. Outlook of the sub-problems

II. Understanding of the Problem

The scenario focuses on an emergency services logistic problem, in which the planning system has to schedule the activities of a group of robotic agents. These agents' duty is to carry useful supplies, e.g. medicine, to injured people that can not move from their current location.

At first, supplies, boxes and agents are placed in a location that we will refer as *depot*. Supplies have to be moved inside boxes, which are empty at first but can be filled with content by a robotic agent.

While no people are at the *depot*, they can be find in other locations alone or with other people, stuck and unable to move. In contrast, the robotic agents can move between any arbitrary locations.

In practice, what robotic agents have to do is: fill boxes with required supplies, load them, move between locations, unload boxes and give the content to injured people.

Considering the main scenario has been presented, now an overview of the five sub-problems is given.

In the first sub-problem only a single robotic agent is present, able to carry a box at a time. It is the simplest sub-problem but it lays the foundations of the rest of the assignment.

The second sub-problem differs slightly from the first one: here the robotic agent is able to carry up to four boxes at the same time thanks to a carrier.

These first two parts are both modeled in *Planning Domain Definition Language (PDDL)* [1], while the plan is extracted exploiting *planutils* [2] library, which contains many state-of-the-art planners.

The third sub-problem's conditions are equal to the second's but it is modeled using *Hierarchical Task Network (HTN)* and *Hierarchical Domain Definition Language* [3].

The fourth sub-problem consists of enhancing the second sub-problem introducing durative actions, i.e., now every action takes a proper time. In fact, up to this point only atomic actions were considered, with no duration in time.

At last, the fifth sub-problem focuses on modeling the fourth's sub-problem's scenario exploiting *PlanSys2*, *ROS2 Planning System* [4].

It is important to notice that the third sub-problem is unique compared to the others. Indeed, when using *PDDL* language, we consider a bottom-up approach, in which the basic actions are first defined and then are combined together to create a plan. However, when using *HDDL*, a top-down approach is instead used: the starting point is the main task, i.e. aid a person, that is then split in sub-tasks that will be divided until a plan of primitive sub-tasks is formed. This is useful to introduce domain knowledge but it also complicates the design.

III. Design Choices

Although the scenario appears simple, a fair amount of assumptions have to be establish to make the modeling simple and efficient. While some of them are already set by the

assignment's instructions, and can be found in the above section, others are left to the student's will.

First of all, it has been decided to define a discrete number of boxes, which is four for all problems, and to limit their capacity to one, i.e. a single box can contain a supply at a time. Instead, the *depot* contains an infinite amount of supplies, which are divided in: medicine, tools and food. When a robotic agent fills a box with a certain content, it takes only one unit of the supply. Giving a supply to a person implicate emptying the box and making one person receive the content, in other words, an unit of supply can not be shared among people.

In addition, even if a box is at their current location, people can not open a box and receive the content by themselves, but a robotic agent is needed. This is particularly useful to solve the ambiguous situation in which a box is left at a location where more than one person want the content.

Moreover, the robotic agent is supported with a single robotic arm and, if present, with a carrier. No additional action is required to attach the carrier to the agent.

Lastly, the following situations are not considered: all boxes are filled with not required supplies, a robotic agent is loaded with boxes with unnecessary content, a robotic agent is able to take a supply back after giving it to a person or is able to throw away a content from a box.

A. Structure of the Planning System

In general, a scenario is modeled by a planning system through two different modules: a domain file, which describes the world in which the scenario takes place, and a problem file, which instances the initial and goal conditions.

In particular, the domain file has three major components: requirements, predicates, actions. Requirements can be added to enhance the domain. For instance, *typing* allows to attach a type to objects to better regulate predicates and actions, i.e. the action *move* let the robotic agent only to move between locations. However, it is important to notice not all planners supports every requirement, so one should pay attention when adding them. Predicates can be seen as attributes applicable to objects, for instance the predicate *"ratl ?r - robot ?l - location"* being *True* means the robot *?r* is currently at location *?l*. Actions transform the state of the world and usually need some preconditions to hold before being executed. Then, they are added in the action sequence to form a plan.

An additional element are tasks, which are present only in the third sub-problem being the only one modeled in *HDDL*.

Type	Description
location	where person, robot, box and supply can be
person	person in need of help, is at a specific location
robot	robotic agent, carries boxes to people
box	containts emergency supplies
food medicine tool - supply	supplies to be brought to people
carrier*	contains more boxes, it is attached to the robot
place*	place on the carrier, a carrier contains more places

Table 2. Types of object.
*Carrier and place are not present in the first sub-problem

Predicate	Description
(patl ?p - person ?l - location)	person ?p is at location ?l
(ratl ?r - robot ?l - location)	robot ?r is at location ?l
(batl ?b - box ?l - location)	box ?b is at location ?l
(satl ?s - supply ?l - location)	supply ?s is at location ?l
(contain ?b - box ?s - supply)	box ?b contains supply ?s
(empty ?b - box)	box ?b is empty
(have ?p - person ?s - supply)	person ?p has supply ?s
(loaded ?r - robot ?b - box)	robot ?r has box ?b
(unloaded ?r - robot)	robot ?r is empty
(hascarrier ?r - robot ?c - carrier)	robot ?r has carrier ?c
(hasplace ?c - carrier ?pl - place)	carrier ?c has space ?pl
(hasbox ?pl - place ?b - box)	place ?pl has a box ?b
(free ?pl - place)	place ?pl is free
(canact ?r - robot)	robot ?r is free and can do actions

Table 3. Types of predicates

Tasks can be seen as blocks of actions, and can be decomposed and combined together to create a plan.

The problem file is way simpler than its counterpart: here, all object present in the scenario are listed, along with the predicates that hold at the beginning and the predicates that will hold in the goal states or the tasks to be performed.

In the repository, for each sub-problem a corresponding folder contains both domain and problem files with any additional file required to run them. Instructions to execute the sub-problems are present in the *README* file.

Even if the assumptions behind each sub-problem are unique, the majority of the design is common in all files. Given this, the remaining of the section is going to explain in depth how the different components have been implemented, starting from the domain ones.

B. Requirements

All sub-problems exploit *strips* and *typing*, which are the basic requirements of a planning domain: the first allows to specify add and remove effects and the second concedes attaching types to objects.

In Table 2, one can see the types used in the assignment. Note that *carrier* and *place* are introduced from the second sub-problem onward to allow the robot to carry multiple boxes at the same time.

In addition, in the third sub-problem, *hierarchie* is present to define a Hierarchical Task Network (HTN) domain. Likewise, in the fourth and fifth sub-problems *durative-actions* is called to describe the duration of the actions.

C. Predicates

The totality of the predicates with a brief description are shown in Table 3. The first seven predicates are common to all sub-problems and describe mainly three aspects: if an object is at a certain location, if a certain box contains a supply, and if an injured person received a specific supply.

The predicates *"loaded ?r - robot ?b - box"* and *"unloaded ?r - robot"* are present only in the first sub-problem and tell if the robotic agent is carrying a box.

These are replaced in the remaining of the sub-problems with *"hascarrier ?r - robot ?c - carrier"*, *"hasplace ?c - carrier ?pl - place"*, *"hasbox ?pl - place ?b - box"*, *"free ?pl - place"*. These four predicates are introduced to consider the carrier which lets the robotic agent carry more boxes at the same time. The idea is that the carrier can be attached to a robotic agent and it has few spaces that can be occupied by a box. This makes fairly easy to track the boxes and the occupancy of the robotic agent. A downfall is that places on the carrier are not considered equivalent, so the search space broadens and, depending on the planner, this can lead to a slower search.

As these predicates have been substituted, now the robotic agents can only carry boxes through a carrier. If one wants to set a scenario in which both robotic agents with carriers and robotic agents without are present, one can simply do this reintroducing the predicates and the corresponding actions that have been modified.

Regarding *"canact ?r - robot"*, it was added out of necessity: without this predicate the robotic agents can start multiple actions at the same time, such as filling two boxes in parallel. However, as we consider robotic agents with only one robotic arm this should not be possible. In addition, we neither permit the robot to act when moving between different locations. So, *"canact ?r - robot"* ensures the robot perform an action at a time.

D. Actions

A robotic agents is given five basic actions: *fill, load, unload, move* and *give*. When performing *fill*, the robotic agent fills a box with a certain content, such as medicine, if both the box and the content are at its current location. *Load* and *unload* are two specular actions: *load* is used to load a box onto a robotic agent, if both are at the same location, while *unload* is used to unload a box that is on a robotic agent to the current location. Then, *give* empties a box giving its content to an injured person, if the robotic agent, the box and the person are in the same location. Lastly, *move* let the robotic agent to move between arbitrary locations.

Interesting to notice, *load* and *unload* make the box "disappear" or "appear" at the current location, meaning the action *give* can only be used after unloading the box from the robotic agent. This helps when defining the action *move*: a basic action can be modeled, regardless of the fact the robotic agent is carrying or not something.

In contrast, when filling a box, the supply does not disappear from the location as we made the assumption that supplies are infinite in number. This is also possible because no scenario in which a supply is at a location different from the *depot* has been considered. Indeed, the supply is either at the *depot*, inside a box or in possess of a person.

Differently from predicates, all these actions are modeled distinctively in every sub-problem.

Action	Description	Duration
fill	fills a box with a content	2
load	loads box onto robotic agent	1
unload	unloads box to location	1
move	moves between arbitrary locations	2
give	gives supply to person	4

Table 4. Types of actions and their duration

In the first sub-problem, we consider the case of a single robotic agent able to carry a box at a time. Given this, the domain file is simply a translation from the above basic actions to *PDDL* language.

In the second and third sub-problem the robotic agent is given the possibility to carry multiple boxes through a carrier so the action *load* and *unload* are modified accordingly. In particular, when loading the robotic agent needs to have a carrier, which needs to have at least a free space. Instead, *unload* frees a specific space from the carrier.

Regarding the fourth and fifth sub-problems, durative actions are introduced so actions from the second sub-problem are modified accordingly using *durative-actions* requirement. The duration of the actions are more or less set arbitrarily and can be seen in Table 4. The considerations done are that *fill* should take more time compared to *load* and *unload*, as it requires more precision, and *give* should have the largest duration as the robotic agent can create issues if it moves without caution near injured people.

Note that in the fifth sub-problem action nodes have to be created too, in which the duration of an action is set considering certain progress variables.

E. Tasks

The third sub-problem is modeled through *HTN*, in which a main task, in our case aid injured people, is segmented in sub-tasks. Therefore, actions are taken from the second sub-problem, and tasks and methods, which specify how tasks are performed, are added.

Here, four tasks have been defined: *aid_person, give_supply, fill_and_load* and *prepare_to_fill*.

The main task to be completed is *aid_person*, i.e. carry the requested supply to an injured person. This is separated into two task: *fill_and_load*, which aim is to have a box filled with the requested supply loaded onto the robotic agent, and *give_supply*, in which the box is unloaded and the content is given to the person.

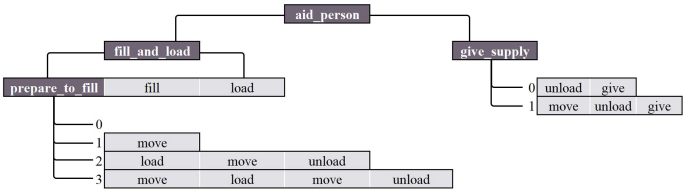

Fig. 1: Tasks

Object	Types
l1, l2, l3	location
p1, p2, p3	person
r1	robot
c1*	carrier
pl1*, pl2*, pl3*, pl4*	places
b1, b2, b3, b4	boxes
f	food
m	medicine
t	tool

Table 5. Objects
*Carrier and places are not present in the first sub-problem

The task *fill_and_load* is divided as well into three components: the task *prepare_to_fill* and two primitive tasks *fill* and *load*, which correspond to the actions with the same name.

Regarding *prepare_to_fill*, it is the task with the higher amount of methods. The simplest case is when the robotic agent and the box are already at the *depot*, so no action takes place. In the second case, the box is still at the *depot*, but the robotic agent is at a different location so it has to move back. The third case happens when both robotic agent and box are at the same location, but different from the *depot*: in this case the robotic agent has to load the box, move back to the *depot* and unload the box. Lastly, when the box and the robotic agent are each at a different locations, both not *depot*, the robotic agent has to move to the current box location, load it, move to the *depot* and unload the box.

In the picture Fig. 1 a schema of the decomposition of the tasks is presented, lighter boxes represents primitive tasks.

F. Problem

Regarding the first, second and fourth sub-problems, the problem files are modeled in *PDDL* and the main differences

Initial State Predicates	
robotic agent, boxes and supplies at depot	places and box are free
ratl r1 depot	free pl1
batl b1 depot	free pl2
batl b2 depot	free pl3
batl b3 depot	free pl4
batl b4 depot	empty b1
satl f depot	empty b2
satl m depot	empty b3
satl t depot	empty b4
robot has carrier, carrier has places	people's location
hascarrier r1 c1	patl p1 l2
hasplace c1 pl1	patl p2 l2
hasplace c1 pl2	patl p3 l3
hasplace c1 pl3	robot can act
hasplace c1 pl4	canact r1

Table 6. Example of initial predicates

Goal Predicates
have p1 f
have p2 f
have p2 t
have p3 m

Table 7. Example of goal predicates

rely to the presence of the carrier, which adds and removes some predicates and objects. Although objects and predicates are the same, in the fifth sub-problem there is no real problem file, and the different components are defined through the terminal. Instead, in the third sub-problem the goal conditions are substituted by tasks addressing *HTN*'s structure and the file is in *HDDL*.

Overall, the objects taken in consideration can be seen in Table 5. Following the assignment's instructions, a robotic agent should carry at maximum four boxes, so four places and four boxes are defined. In general more boxes could be present in the scenario, but this amount is already fine to find an optimal solution. *Food, medicine* and *tool* are infinite in number so one instance for each of them is enough even if more people want the same supply. Note that *depot* is not listed in the objects here as it is declared as constant in the domain file. This does not count in the fifth sub-problem and so *depot* has to be explicitly defined in the terminal along other objects.

An example of predicates defining the initial conditions can be seen in Table 6. These can be applied to all sub-problems but the first, where no carrier is present, and the predicate *canact r1* is only required in the fourth and fifth sub-problems. As requested, these predicates set robotic agent, boxes and supplies at the *depot* and people at different locations. The carrier and the corresponding four places are declared to be attached to the robotic agent. Moreover, all places and boxes are free.

Then, an example of goal predicates is shown in Table 7. The objective is to deliver necessary supplies to injured people, which are in different locations. People may need one or more supply, and more people can request the same type of supply.

Note that this format does not appear in the third sub-problem as the goal is defined as a set of tasks to be completed. In our case, this implies multiple *aid_person* tasks.

IV. Results

Although the scenario appears simple, it is interesting to evaluate how different planners behave, if they achieve finding an optimal solution or if any issue arises.

The results are shown in form of tables in the below subsections and in form of screenshots in the appendix.

A. Sub-problem 1

Let us consider the objects and predicates above mentioned, which are reported in Table 5, 6 and 7. In brief, a single robotic agent has to aid four people by carrying to them the requested supplies. The robotic agent, the boxes and the supplies are all at the *depot*, while injured people are in other locations.

Sub-problem 1			
Planner	Number of Steps	Time (s)	States Evaluated
FF	29	**0.00**	64
LAMA	29	0.04	4883
LAMA-FIRST	29	0.01	99
optimal solution	29		

Table 8. Results of sub-problem 1

In the first sub-problem, the robotic agent lacks the carrier so it can carry only one box at a time. This type of scenario does not offer lots of possibilities to the search algorithm but three different planners were still compared, using *Planutils* to install and run them. The planners are: *Fast Forward (FF)* [5], *LAMA* [6] and *LAMA-FIRST*.

FF is guided by a heuristic that estimates goal distances by ignoring delete lists, and relies on forward search. *LAMA* relies on forward search as well, but uses a pseudo-heuristic that is derived from landmarks, which are propositional formulas that must be true in every solution.

In Table 8 are reported the results of the different planners. As one can expect, given the size of the problem, all planners find an optimal solution, which is composed by 29 steps. Interesting to notice, *FF* and *LAMA-FIRST* evaluates a similar amount of states and the search time is almost null. In contrast, *LAMA* takes more time and evaluates five times more states.

In this simple scenario, *FF* and *LAMA-FIRST* behave better than *LAMA*. Indeed, the optimal solution here can be found really fast.

B. Sub-problem 2

The first and second sub-problems differ only for the presence of the carrier, so the same planners and same initial and goal conditions have been used here.

The presence of the carrier introduces difficulties but also the chance of finding action sequences with fair less steps. Indeed, we can also notice in Table 9 how *LAMA* is now the only planner which has found an optimal solution, while others only found sub-optimal ones.

The reason can simply be found in the fact that *LAMA* continues to search for better solutions until it has exhausted the search space or is interrupted. Naturally, the search is relatively slower compared to *FF* and *LAMA-FIRST* which tend to find a solution faster, even if not optimal.

Sub-problem 2			
Planner	Number of Steps	Time (s)	States Evaluated
FF	21	**0.00**	117
LAMA	**18**	159.39	17527636
LAMA-FIRST	23	0.01	99
optimal solution	18		

Table 9. Results of sub-problem 2

C. Sub-problem 3

Planning and Acting in a Network Decomposition Architecture (PANDA) [7] is a planning system that allows solving different kinds of planning problems, in particular it can address *HTN* problems.

Sub-problem 3							
Problem	P	S	L	O	Number of Steps	Time (s)	Optimal Solution
1	1	1	1	1	9	2.02	9
2	2	1	1	order	19	3.25	17
				not order	19	**2.79**	
3	2	2	2	order	19	3.88	18
				not order	**18**	**3.04**	
4	3	3	2	order	**29**	**4.96**	26
				not order	32	24.55	

Table 10. Results of sub-problem 3. P, S, L columns refers to number of people, number of supplies and number of locations in the problem

As this is the only planner used in this section, it is evaluated using the following goal conditions:

1) one person requires one supply
2) two people who are in the same location require the same type of supply
3) two people in two different locations require a supply each
4) three people in two locations require a supply each

For all but the first, both ordered and not ordered goal tasks were considered. In case of ordered task, one task has to be completed before executing the next. For instance, we could have a person that has higher priority in respect to others, i.e. it needs medicine urgently. Instead, not ordered goal are useful when we have to complete all tasks, i.e., aid all people, no matter the order.

As one can see in Table 10, when the task is totally ordered no optimal solution can be found as sub-tasks can not be interleaved. In other words, the robotic agent have to satisfy one person at a time, returning to the *depot* after completing each task *aid_person*. Even when the tasks are not totally ordered, the planner does not always find optimal solutions, partially due to constraints on the design. Indeed, as preconditions of *aid_person* there is a need of an empty box and a free place on the carrier, which are transmitted to the sub-tasks. So, when two method *aid_person* with the same box and place are selected, sub-tasks of the second goal task could not interleave properly the first, getting stuck until the objects are no longer used. However, when given more freedom to the planner, not bounding the objects from the beginning, the not totally ordered search tends to be really slow and consume lot of space, making it unfeasible to use even for small problems.

D. Sub-problem 4

Given the presence of durative actions, temporal planners have been introduced and their performance evaluated. Similarly to what happened in the first and second-sub-problems, *planutils* was exploited to install and run two temporal planners: *Temporal Fast Downward (TFD)* [9] and *Optimising Preferences and Time-Dependent Costs (OPTIC)* [10].

TFD is based on *Fast Downward* [8], which performs forward heuristic search, and uses an adaptation of the context-enhance additive heuristic. *OPTIC* planner focuses in tempo-

Sub-problem 4								
Problem	P	S	L	Planner	Number of Steps	Maskespan	Time (s)	Optimal Solution (Makespan)
1	1	1	1	TFD	5	10	**0**	10
				OPTIC	5	10	0.32	
2	2	2	2	TFD	11	22	**0**	20
				OPTIC	11	22	0.16	
				OPTIC*	**10**	**20**	0.12	
3	3	4	2	TFD	23	46	**0**	36
				OPTIC	23	46	6.45	

Table 11. Results of sub-problem 4. P, S, L columns refers to number of people, number of supplies and number of locations in the problem. OPTIC* has available only two boxes

ral problem where plan cost is determined by preferences. However, as we do not set any measure, it will be trying to minimize the makespan, i.e. the total duration of the plan.

The following problems have been used to evaluate the performance of the two planners:

1) one person require one supply
2) two people in two locations require one supply each
3) three people in two locations require 4 supplies

As one can see in Table 11, both planners finds similar solutions. However, while *TFD* find a solution really fast, *OPTIC* planner tries to minimize the makespan iteratively, creating sub-optimal plans that are faster than the precedents but not always it is able to find an optimal solution in a adequate time. Indeed, the table collects the best results of the planner before getting terminated, if it got killed.

In addition, for the second problem, in the table are also reported the result of *OPTIC* when it has available only the necessary amount of boxes and places in the carrier, which is two for both. As one can see, it reaches an optimal solution as the search space is smaller.

E. Sub-problem 5

At last, the scenario was implemented through *ROS2 Planning System, PlanSys2*. The scenario and the conditions are no different from the last sub-problem: a robotic agents has to aid one or more people using durative actions.

As *PlanSys2* uses by default *Partial Order Planning Forwards (POPF)* planner [11], which we did not run in the last section, the results are reported in Table 12.

As one can see, the results are similar to the other temporal planners.

V. Conclusion

Here, an application of Automated Planning has been presented, regarding an emergency scenario in which a robot's task is to aid people in need. The different parts of the assignment are useful to, starting by a first basic application, understanding how Automated Planning can be useful and how it can be implemented. In addition, even if the problems are rather simple, it highlights the shortcoming of Automated Planning, such as the trade-off planners make when searching a solution between optimality and time.

Sub-problem 5							
Problem	P	S	L	Planner	Number of Steps	Maskespan	Optimal Solution (Makespan)
1	1	1	1	PlanSys2 POPF	5	10	10
2	2	2	2	PlanSys2 POPF	11	22	20
3	3	4	2	PlanSys2 POPF	23	46	36

Table 12. Results of sub-problem 5. P, S, L columns refers to number of people, number of supplies and number of locations in the problem.

References

[1] M. Ghallab, C. A. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, Chung, K. Golden, S. Penberthy, D. Smith, Y. Sun, D. Weld, "PDDL - The Planning Domain Definition Language", 1998

[2] C. Muise, F. Pommerening, J. Seipp and M. Katz, "Planutils: Bringing Planning to the Masses", In 32nd International Conference on Automated Planning and Scheduling, System Demonstrations and Exhibits, 2022.

[3] D. Höller, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, R. Alford, "HDDL – A Language to Describe Hierarchical Planning Problems" International Workshop on HTN Planning (ICAPS), 2019.

[4] F. Martin, C. Gines, V. Matellan, L.Rodriguez, "PlanSys2: A Planning System Framework for ROS2", 2021.

[5] J. Hoffmann, "FF: The Fast-Forward Planning System", Ai Magazine 22(3):57-62, September 2001.

[6] S. Richter, M. Westphal, "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks", Journal Of Artificial Intelligence Research, Volume 39, pages 127-177, 2010.

[7] D. Höller, G. Behnke, P. Bercher S. Biundo "The PANDA Framework for Hierarchical Planning", Künstl Intell 35, 391–396, 2021.

[8] M. Helmert, "The Fast Downward Planning System", Journal Of Artificial Intelligence Research, Volume 26, pages 191-246, 2006.

[9] P. Eyerich, R. Mattmüller and G. Röger, "Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning", In Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009), 2009.

[10] J. Benton, A. Coles, and A. Coles, "Temporal Planning with Preferences and Time-Dependent Continuous Costs", ICAPS, vol. 22, no. 1, pp. 2-10, May 2012.

[11] A. J. Coles, A. I. Coles, M. Fox, D. Long, "Forward-Chaining Partial-Order Planning" [Coles,A. J. Coles, A. I. Fox, M. Long, D.], Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010.

APPENDIX

```
fill r1 depot b1 f (1)
load r1 depot b1 (1)
move r1 depot l2 (1)
unload r1 b1 l2 (1)
give r1 l2 b1 f p1 (1)
load r1 l2 b1 (1)
move r1 l2 depot (1)
unload r1 b1 depot (1)
fill r1 depot b1 m (1)
load r1 depot b1 (1)
move r1 depot l3 (1)
unload r1 b1 l3 (1)
give r1 l3 b1 m p3 (1)
load r1 l3 b1 (1)
move r1 l3 depot (1)
unload r1 b1 depot (1)
fill r1 depot b1 t (1)
load r1 depot b1 (1)
move r1 depot l2 (1)
unload r1 b1 l2 (1)
give r1 l2 b1 t p2 (1)
load r1 l2 b1 (1)
move r1 l2 depot (1)
unload r1 b1 depot (1)
fill r1 depot b1 f (1)
load r1 depot b1 (1)
move r1 depot l2 (1)
unload r1 b1 l2 (1)
give r1 l2 b1 f p2 (1)
```

Fig. 2: Sub-problem 1 plan by LAMA

```
fill r1 depot b1 f (1)
fill r1 depot b2 m (1)
fill r1 depot b3 t (1)
load r1 b1 depot c1 pl4 (1)
load r1 b2 depot c1 pl3 (1)
load r1 b3 depot c1 pl2 (1)
fill r1 depot b4 f (1)
load r1 b4 depot c1 pl1 (1)
move r1 depot l3 (1)
unload r1 b2 l3 c1 pl3 (1)
give r1 l3 b2 m p3 (1)
move r1 l3 l2 (1)
unload r1 b3 l2 c1 pl2 (1)
give r1 l2 b3 t p2 (1)
unload r1 b4 l2 c1 pl1 (1)
give r1 l2 b4 f p1 (1)
unload r1 b1 l2 c1 pl4 (1)
give r1 l2 b1 f p2 (1)
```

Fig. 3: Sub-problem 2 plan by LAMA

```
0: SHOP_methodaid_person_0_0_precondition(c1,depot,f,r1,b1,pl1,depot,depot,p1,l2)
1: SHOP_methodfill_and_load_0_1_precondition(f,depot,pl1,c1,b1,depot,depot,r1)
2: SHOP_methodprepare_to_fill_0_2_precondition(pl1,f,b1,depot,c1,r1)
3: fill(r1,depot,b1,f)
4: load(r1,b1,depot,c1,pl1)
5: SHOP_methodgive_supply_to_person_1_7_precondition(depot,p1,f,r1,pl1,l2,b1,c1)
6: move(r1,depot,l2)
7: unload(r1,b1,l2,c1,pl1)
8: give(r1,l2,b1,f,p1)
9: SHOP_methodaid_person_0_0_precondition(c1,depot,t,r1,b3,pl3,l2,depot,p2,l2)
10: SHOP_methodfill_and_load_0_1_precondition(t,l2,pl3,c1,b3,depot,depot,r1)
11: SHOP_methodprepare_to_fill_1_3_precondition(r1,pl3,c1,b3,t,depot,l2)
12: move(r1,l2,depot)
13: fill(r1,depot,b3,t)
14: load(r1,b3,depot,c1,pl3)
15: SHOP_methodgive_supply_to_person_1_7_precondition(depot,p2,t,r1,pl3,l2,b3,c1)
16: move(r1,depot,l2)
17: unload(r1,b3,l2,c1,pl3)
18: give(r1,l2,b3,t,p2)
19: SHOP_methodaid_person_0_0_precondition(c1,depot,m,r1,b2,pl1,l2,depot,p3,l3)
20: SHOP_methodfill_and_load_0_1_precondition(m,l2,pl1,c1,b2,depot,depot,r1)
21: SHOP_methodprepare_to_fill_1_3_precondition(r1,pl1,c1,b2,m,depot,l2)
22: move(r1,l2,depot)
23: fill(r1,depot,b2,m)
24: load(r1,b2,depot,c1,pl1)
25: SHOP_methodgive_supply_to_person_1_7_precondition(depot,p3,m,r1,pl1,l3,b2,c1)
26: move(r1,depot,l3)
27: unload(r1,b2,l3,c1,pl1)
28: give(r1,l3,b2,m,p3)
```

Fig. 4: Sub-problem 3 plan by PANDA (case 4 ordered)

```
0.00100000: (fill r1 depot b2 f) [2.00000000]
2.01100000: (load r1 b2 depot c1 pl2) [1.00000000]
3.02100000: (move r1 depot l2) [2.00000000]
5.03100000: (unload r1 b2 l2 c1 pl2) [1.00000000]
6.04100000: (give r1 l2 b2 f p1) [4.00000000]
10.05100000: (move r1 l2 depot) [2.00000000]
12.06100000: (fill r1 depot b3 f) [2.00000000]
14.07100000: (load r1 b3 depot c1 pl2) [1.00000000]
15.08100000: (move r1 depot l2) [2.00000000]
17.09100000: (unload r1 b3 l2 c1 pl2) [1.00000000]
18.10100000: (give r1 l2 b3 f p2) [4.00000000]
22.11100000: (move r1 l2 depot) [2.00000000]
24.12100000: (fill r1 depot b1 t) [2.00000000]
26.13100000: (load r1 b1 depot c1 pl1) [1.00000000]
27.14100000: (move r1 depot l2) [2.00000000]
29.15100000: (unload r1 b1 l2 c1 pl1) [1.00000000]
30.16100000: (give r1 l2 b1 t p2) [4.00000000]
34.17100000: (move r1 l2 depot) [2.00000000]
36.18100000: (fill r1 depot b4 m) [2.00000000]
38.19100000: (load r1 b4 depot c1 pl1) [1.00000000]
39.20100000: (move r1 depot l3) [2.00000000]
41.21100000: (unload r1 b4 l3 c1 pl1) [1.00000000]
42.22100000: (give r1 l3 b4 m p3) [4.00000000]
```

Fig. 5: Sub-problem 4 plan by TFD (case 3)

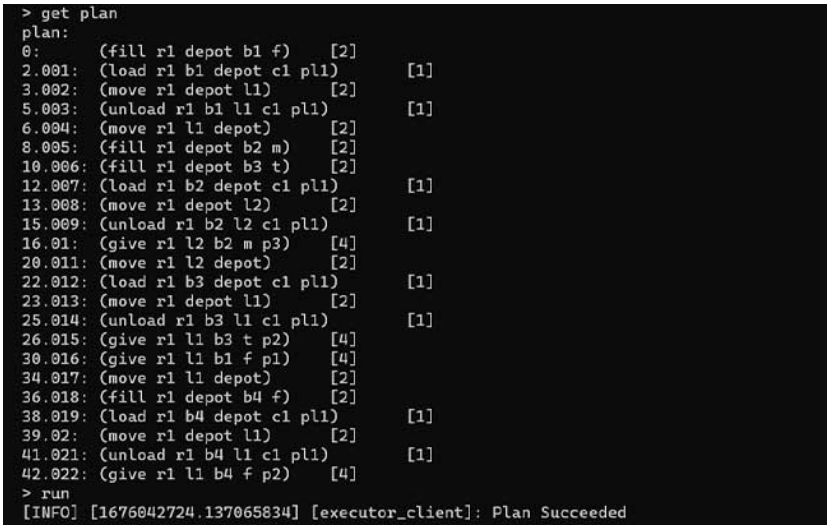

Fig. 6: Sub-problem 5 plan by POPF (case 3)

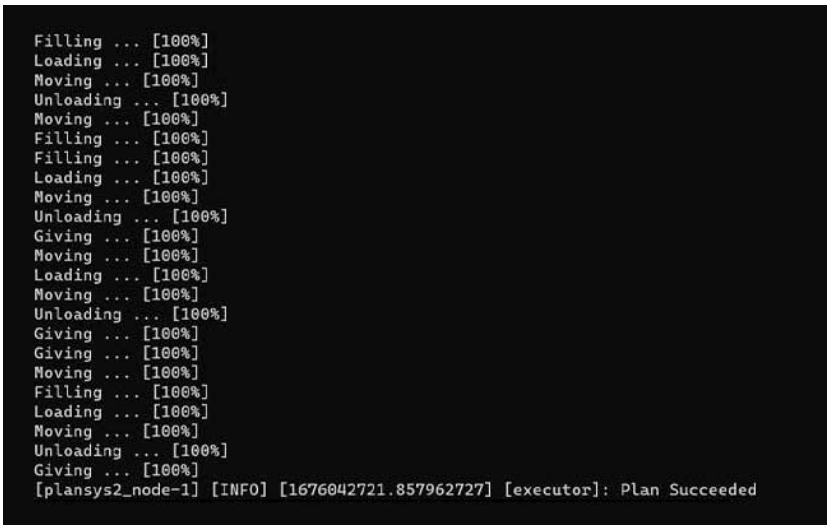

Fig. 7: Sub-problem 5 run (case 3)