Artificial Intelligence for Robotics II

Assignment Report

Benedetti Lorenzo, Manera Andrea G. P., Pagano Francesco, Sani Ettore

May 2022

Contents

1 Introduction **2**
1.1 Brief description of the model . . . 2
1.2 Choice of the Planning Language . . . 2
1.3 Implementation General scheme . . . 3

2 Code description **4**
2.1 Domain . . . 4
2.1.1 Predicates . . . 6
2.1.2 Functions . . . 7
2.1.3 Actions . . . 7
2.1.4 Events . . . 8
2.1.5 Processes . . . 9
2.2 Problem instances . . . 9

3 ENHSP different choices of search strategies and heuristics **10**

4 Results and conclusions **13**
4.1 Choice of the planner . . . 13
4.2 Output discussion . . . 13

Chapter 1

Introduction

1.1 Brief description of the model

This report describes the procedure used to solve the assignment of the Artificial Intelligence for Robotics II course. Our task was to model a Warehouse scenario in which robots can load and carry crates.

We chose to model all the proposed optional extensions, therefore the final problem has the following structure:

- The **crates** loaded on the conveyor belt can be heavy or light. Sometimes they could also be fragile: they need to be carefully loaded on the conveyor belt. Moreover, some crates belong to the same crate set and therefore they need to be subsequently loaded.
- Two **mobile robots** that can load and unload crates from the shelf and on the loading bay. They move with different velocities depending on the crate type they are moving. These two robots need recharging.
- Two **loaders**:
  - The first loader can load heavy and light crates. We will refer to this loader with the *expensive* adjective.
  - Instead, the second one can only load light crates. We will refer to this loader with the *cheap* adjective.

A thing to point out is that: some problem specifications could have been interpreted in a non-standard way. Therefore we're reporting all the assumptions we have made during our project implementation:

1. The mover recharges every time it is in the loading bay like it is a wireless recharging area. So, for each time unit, the robot stays in the loading bay, and its battery level increases. The 0 battery level is the last acceptable level whereby the robot movement is allowed. This means that it is as if the robot had 21 battery levels.
2. During the loaders' loading process of a crate, the loading bay can be filled up by another crate; this is because two crates can be loaded simultaneously on the conveyor belt since two loaders are available.
3. At the end of the plan, the mobile robots must be in the loading bay to be ready for other tasks.

1.2 Choice of the Planning Language

To describe the model, we chose the *PDDL+* description language. This choice is mainly based on three aspects:

- The model must take into account time.
- The need for a mixed discrete-continuous domain, where some variables may change in time, such as the travelled distance of the mover and its remaining battery.

- The need to have processes and events in the domain: they describe how the model evolves and how our agents interact with the environment.

In particular the model follows the *action-process-event* schema, where:

- *Actions* are planning decisions: the planner uses them to start processes.
- *Processes* describe how the model evolves in time.
- *Events* stop processes when they reach their goal.

1.3 Implementation General scheme

The figure 1.1 depicts the involved processes and objects following the logical sequence of the events.

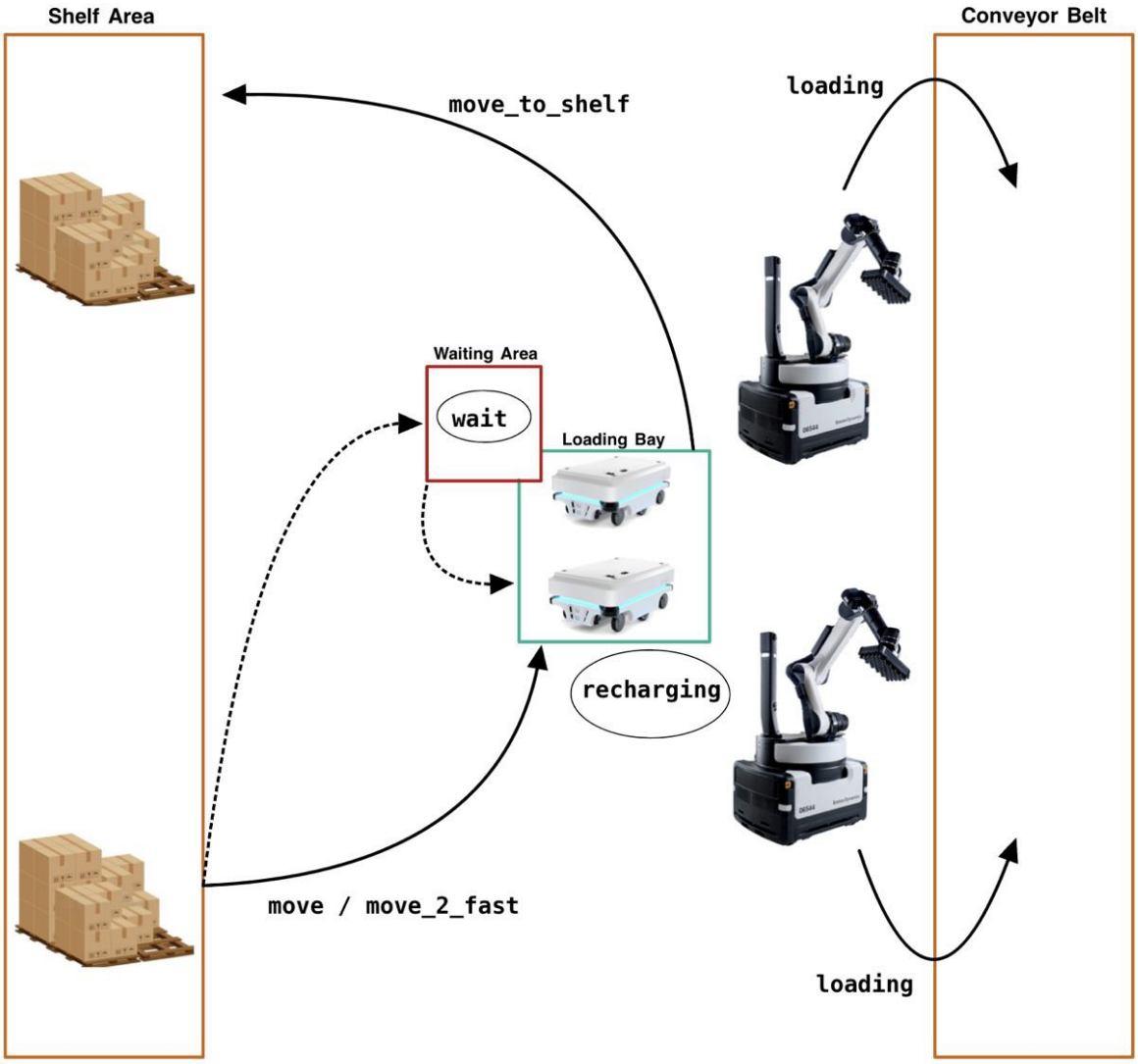

Figure 1.1: General project scheme, highlighting processes

Chapter 2

Code description

2.1 Domain

In this section, we're going to describe all the predicates, actions, functions, processes, and events that we have implemented. We used actions and events to start and stop processes which are associated with:

- **Robot movements**. In this case, some functions are increased such as the robot's travelled distance.
- **Recharging**. Here, the process effect is to increase the function correlated with the movers' battery level.
- **Waiting**. The robot waits until the loading bay is free.
- **Crate Loading**. A variable that represents the time taken by the loading process is increased.

The general idea is that the mobile robots start moving when the action *new_order* is taken and the process *move_to_shelf* allows the robot to reach the crate in the shelf area. In such a way one or more movers are associated with a single crate. Then, crates are carried when one among these three actions is taken depending on the type of crate:

- *carry1_standard*: for using just one mover robot for a light crate.
- *carry2_standard*: for using two mover robots for either a heavy crate or a fragile crate.
- *carry2_fast*: for using two mover robots for a light crate.

Then, the robots can start doing the *move* process, that increments the function associated to the robot travelled distance, following the formula:

$$distance = \frac{150 \cdot t}{weight} \tag{2.1}$$

In case of the *carry2_fast* action, the formula becomes:

$$distance = \frac{100 \cdot t}{weight} \tag{2.2}$$

We obtained these formulas by inverting the one on the assignment specifications document.

These processes can stop thanks either to the *put_down* (for dropping a single crate in the loading bay) or *put_down2* (for dropping two crates) events when the travelled distance reaches the crate distance value. At this point, the two loaders can pick different kinds of crates up.

When the action *pick_up_expensive* is taken, the expensive loader is involved in loading a light or a heavy crate. On the other hand, when *pick_up_cheap* is taken the cheaper loader is involved in loading exclusively light crates.

Through the *loading* process, crates can finally be unloaded on the conveyor belt. When the first crate is put on the conveyor, it is classified as the first to be unloaded by the event *first_of_sequence*. The

next crate to be unloaded instead is classified as the one next to the first of the sequence by the event *next_of_sequence*.

The principal effect of this strategy is that the crates on the conveyor belt are marked as "near in sequence". This has been useful for the optional extension about crates belonging to the same group.

Moreover, some Knowledge Engineering strategies learned in class have been applied to our domain model, such as:

1. preconditions most likely to be unsatisfied have been listed earlier.
2. main effects have been listed earlier.

In the following section, we're reporting a description of every single predicate, action, function, process, and event we used.

2.1.1 Predicates

Predicate	Description
crate_at_shelf ?c - crate	Predicate to indicate if the crate is at the shelf.
crate_at_bay ?c - crate	Predicate to indicate if the crate is at the loading bay.
crate_on_belt ?c - crate	Predicate to indicate if the crate is on the conveyor belt.
is_fragile ?c - crate	Predicate to differentiate between fragile and not fragile crates.
crate_on_mover ?c - crate ?m - mover	Predicate used to know if a crate is being carried by a mover.
crate_picked_from_loader ?c - crate ?l - loader	Predicate to indicate if a crate has been picked by the loader.
crate_placement ?c - crate	Predicate needed because some crates have to be grouped on the conveyor belt.
last_of_sequence ?c - crate	Predicate to define the last crate of a group.
near_in_sequence ?cn ?cl - crate	Predicate to know if two crates are near in a group.
robot_at_bay ?m - mover	Predicate to indicate if the mover is at the loading bay.
robot_reaching_crate ?m - mover ?c - crate	Predicate to know if the mover is moving toward a crate.
robot_at_crate ?m - mover ?c - crate	Predicate to know if the crate has been reached by the mover.
robot_waiting ?m - mover	Predicate that indicate if the mover is waiting for the loading bay to be free.
moving_to_shelf ?m - mover	Predicate that indicate if the mover is moving to the shelf.
moving_to_conveyor ?m - mover	Predicate that indicate if the mover is waiting for the loading bay to be free.
moving_to_shelf ?m - mover	Predicate that indicate if the mover is moving to the shelf.
moving_to_conveyor ?m - mover	Predicate that indicate if the mover is moving to the loading area.
moving_fast_to_conveyor ?m - mover	This predicate has been declared since if two movers carry a light crate together they need less time to cover the same distance.
robot_coupled ?m - mover	Predicate to know if two movers are coupled (in order to move a heavy crate or a light one faster).
is_picking ?l - loader	Predicate to know if the loader is picking a crate.
bay_is_full	Predicate to know if the loading bay is full.
conveyor_is_empty	Predicate to know if the conveyor belt is empty.

Table 2.1: Predicates Description

2.1.2 Functions

Functions	Description
weight ?c - crate	Function to define the weight of a crate.
distance ?c - crate	Function to define the distance of a crate from the loading bay.
travelled ?m - mover	Function to keep track of the distance covered by the mover.
battery ?m - mover	Function for the battery of the mover robot.
battery_max	Threshold for the maximum battery level
battery_min	Threshold for the minimum battery level.
loading_time ?l - loader	Function to indicate the time needed by a loader to load a crate on the belt.
weight_threshold	Threshold to distinguish between heavy and light crates.
loading_threshold	Threshold that represent the time taken by the loader to load a crate.
loading_threshold_fragile	Threshold that represent the time taken by the loader to load a fragile crate.

Table 2.2: Functions Description

2.1.3 Actions

new_order

This action is activated when a new order is placed and it activates the process to move the robot to the shelf. It requires the robot to be located at the loading bay and the crate to be on the shelf; as an effect, it tells whether the robot is moving to the crate.

carry1_standard

This is the action to carry a light crate with a single mover. To take place the robot has to be in the same location as the crate and the weight of the crate must be lower than the weight threshold.

carry2_standard

This is the action to carry a heavy crate or to carry a fragile crate and to do so we need two mover robots. It requires both robots to be at the same location of the crate. As result, the two robots are coupled and they are moving to the loading area.

carry2_fast

This is the action to carry a light crate with two movers. The structure is the same as the action seen before, but in this case, we check if the weight of the robot is under the threshold.

pick_up_expensive

This action refers to the loading part. There are two loaders in the environment, the cheaper and more expensive one. pick_up_expensive refers to the expensive loader, that can load on the conveyor belt any type of crate (heavy or light). It requires the presence of the crate at the loading bay and that the loader is not picking up anything. As consequence, the loader is picking up the crate and the loading bay is not full.

pick_up_cheap

This action is similar to the previous one, but is related to the cheaper loader; this loader can only pick up light crates, which weight is lower than the weight threshold.

2.1.4 Events

reached_crate

This event is activated every time a mover has reached a crate on the shelf. It requires that the distance travelled by the robot is greater or equal to the distance of the crate from the loading bay, so it can be activated only after the process move_to_shelf. As an effect, it tells that the robot has reached the crate and assigns 0 to the distance travelled by the mover.

busy_bay

This event is used to report whether the loading bay is full, so it's not possible to put a crate in this location and the robot goes in "*waiting state*". It requires a crate to be on the mover and that the distance travelled by the robot is greater or equal to the distance of the crate from the loading bay. As said, the effect is to put the robot in the "*waiting state*".

put_down

This is the event to unload a crate from a mover. It requires a crate to be on the mover, that the distance travelled by the robot is greater or equal to the distance of the crate from the loading bay and that the bay is free. As result, the crate will be in the loading bay, the bay will be full and both the robot and the crate will be at the loading bay.

put_down2

This is the event to unload a crate that has been carried by two movers. The preconditions are the same as the previous action, except for the predicate robot_coupled. Also, the effect of this action is the same as before.

stop_waiting

This event is used to exit the "*waiting state*" if the loading bay is no longer full. It requires the robot to be in the "*waiting state*", the crate to be on the mover, and that the loading bay is not full. The effects are: the robot is no more in the "*waiting state*" (it is at the loading bay) and the crate is at the loading bay.

stop_waiting2

This event is the same as the previous one. The only difference is that it is used when two movers are carrying the same crate.

put_on_belt

This event is needed to stop the loading process; at this stage, the crate has been put on the conveyor belt. The most important requirement is that the loader is picking up the crate from the loading bay. The main effects are that the crate is on the conveyor belt and the loader is not picking up any crate.

put_carefully_on_belt

This event is the same as the previous one, except that it is related to the fragile crates. It refers to the fragile crates that have been put on the conveyor belt by the loader.

first_of_sequence

This event was created to group crates. It refers to the crate that is put on the conveyor belt for the first time (until that moment the conveyor belt is empty). It requires two important preconditions: the conveyor belt must be empty and the crate_placement predicate must be true, a condition that is verified as a consequence of the put_on_belt/put_carefully_on_belt event. As a principal effect, the crate put on the conveyor belt becomes the last one, therefore the conveyor belt is no more empty.

next_of_sequence

This event is strictly related to the first_of_sequence event. It refers to the next crate that has been put on the conveyor belt. Like the previous event, this one was created to group crates. It mostly requires the conveyor belt to be empty. The principal effect is that the crates put on the conveyor belt are marked as "near in sequence".

2.1.5 Processes

move_to_shelf

This process is used to move the robot from the loading bay to the shelf. It requires a minimum amount of battery to take place and, as an effect, it increases the distance travelled by the mover and decreases its battery.

move

This is the process to move the robot mover that is carrying a crate to the loading bay and it is activated thanks to the actions carry1_standard and carry2_standard. It requires that the battery of the robot is larger than battery_min; as an effect, it increases the distance travelled by the robot and decreases its battery.

move2_fast

This is the process to move fast the robot that is carrying a crate to the loading bay and it is activated thanks to the action carry2_fast. It requires that the battery of the robot is larger than battery_min and, as an effect, it increases the distance travelled by the robot and decreases its battery.

wait

This is the process to put the robot in the "*waiting state*" if the loading bay is full.

recharging

This process is related to the recharging of the robot. It is needed to recharge the robot every time it is at the loading bay. Of course, the robot must be at the loading bay and its battery must be equal to or less than the battery_max value. The only effect of this process is to increase the battery level.

loading

This is the process that manages the loading of the crate made by the loader. Of course, as a precondition, the loader has to pick up the crate; the only effect regards the increase of the loading time value.

2.2 Problem instances

The assignment specifies four problem instances with increasing planning complexity. All problem instances define the objects and the thresholds required by the domain, in particular:

- Movers, with their initial battery level.
- Crates, with their weight and distance.
- Loaders, as expensive or cheap ones.
- Weight threshold, to distinguish between heavy and light crates
- Loading threshold: the time required to load normal and fragile crates.
- Initial position of crates and movers.

To satisfy the requirement that some crates must be loaded consequently, the goals are specified with and and or conditions.

The predicate *near_in_sequence* defines if two crates are loaded one next to the other, regardless of the order. If more than two crates belong to the same set, all possible combinations of positions are acceptable, so they are put in or condition.

Chapter 3

ENHSP different choices of search strategies and heuristics

After the domain and problem instances implementation we focused on finding the best search strategy and heuristics for our model. Therefore we tried to plan with different ENHSP settings by adding the *-planner* flag in the command line. This flag sets different heuristics/search strategies configurations.

We analyzed the planned output coming from different planner settings and we noticed that a change in the heuristic or the search strategy modifies a lot of the plan quality, indeed we decided to run our problems with all the allowable configurations.

The considered search strategies are the ones that we saw during the lessons:

- Greedy Best First Search (GBF). Uses heuristic function as evaluation function: $f(n) = h(n)$. Always expands the node that is closest to the goal node. It tries to find a solution as quickly as possible.
- A*. Here the evaluation function is: $f(n) = h(n) + g(n)$. Where $g(n)$ is the cost to reach the node n. This method is used for finding optimal plans but it takes more time to find a solution concerning GBF.
- wA*. The evaluation function is the same as A* but the heuristic component is weighted by a parameter that is w.

The allowable planners' configurations are:

- sat-hmrp: Greedy Best First Search plus MRP heuristic. [8]
- sat-hmrph: Same as before but with helpful actions. [8]
- sat-hadd: Greedy Best First Search with numeric hadd. [8]
- sat-aibr: A* plus AIBR heuristic. [8]
- opt-hmax: A* with hmax numeric heuristic. [8]
- opt-hrmax: Same as before, but with redundant constraints. [8]
- opt-blind: this is a baseline blind heuristic that gives 1 to the state where the goal is not satisfied and 0 to the state where the goal is satisfied. [8]

We are reporting the output plan of the problem instances tried with different planners[1] :

Planner	*Plan Length*	*Elapsed Time*	*Metric*	*Planning Time*	*Heuristic Time*	*Search Time*	*Expanded Nodes*	*States Evaluated*
sat-hmrp	55	30	41	300	41	60	214	322
sat-hmrph	55	30	41	325	45	71	214	322
sat-hadd	61	30	42	335	41	61	106	327
sat-aibr	55	30	41	13 630	13 016	13 365	38 874	45 555
opt-hmax	49	24	35	3 730	2 509	3 446	139 661	187 888
opt-hrmax	49	24	35	4 220	2 846	3 953	152 133	209 567
opt-blind	41	18	28	6 664	23	6 424	773 704	895 490

Table 3.1: Different planner configuration planned on the first problem instance and default deltas.

Planner	*Plan Length*	*Elapsed Time*	*Metric*	*Planning Time*	*Heuristic Time*	*Search Time*	*Expanded Nodes*	*States Evaluated*
sat-hmrp	85	53	67	1 595	921	1 273	45 038	52 578
sat-hmrph	83	51	65	1 806	1 112	1 518	46 009	53 596
sat-hadd	89	53	67	752	342	475	10 247	17 104
sat-aibr	80	48	62	1 194	805	861	789	1 180
opt-hmax	80	45	59	17 206	12 448	19 911	635 803	852 888
opt-hrmax	82	47	61	17 830	12 667	17 512	640 976	873 968
opt-blind	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM

Table 3.2: Different planner configuration planned on the second problem instance and default deltas.

Planner	*Plan Length*	*Elapsed Time*	*Metric*	*Planning Time*	*Heuristic Time*	*Search Time*	*Expanded Nodes*	*States Evaluated*
sat-hmrp	137	101	117	1 678	1 027	1 317	33 326	49 971
sat-hmrph	138	102	118	1 650	1 096	1 379	34 141	50 768
sat-hadd	148	108	123	1 496	838	1 210	43 451	64 607
sat-aibr	136	102	117	6 342	5 890	5 997	5 836	9 465
opt-hmax	138	100	116	110 847	80 362	110 520	3 799 571	4 892 422
opt-hrmax	138	100	116	120 181	89 097	119 854	3 547 412	4 670 686
opt-blind	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM

Table 3.3: Different planner configuration planned on the third problem instance and default deltas.

Planner	*Plan Length*	*Elapsed Time*	*Metric*	*Planning Time*	*Heuristic Time*	*Search Time*	*Expanded Nodes*	*States Evaluated*
sat-hmrp	114	64	86	1 000	509	675	9 876	13 342
sat-hmrph	114	62	85	884	382	538	6 211	8 255
sat-hadd	117	61	83	32 034	23 981	32 744	702 464	1 248 662
sat-aibr	120	70	92	194 566	192 293	194 233	137 043	211 637
opt-hmax	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM
opt-hrmax	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM
opt-blind	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM	ROOM

Table 3.4: Different planner configuration planned on the fourth problem instance and default deltas.

[1]ROOM: Run Out Of Memory. The planner can not find a solution to this problem instance with the given amount of memory.

Moreover, we made some considerations about the *world evolution delta*. This is a discretization delta that tells us how often the exogenous processes and events are checked, and their effect applied to the world. ([1])

The default value of the world evolution delta is 1, therefore we tried to make some changes by setting this value to a smaller one such as 0.5 or 0.1. The result is that the smaller is this value, the greater the amount of time taken by the planner to find the solution, but the smaller is this parameter, the more accurate the simulation. We can state that by observing the plan outputs.

We report a table with the generated output quality parameters of the second instance problem planned with different *world evolution delta* and the default planner:

Deltas/	*Plan Length*	*Elapsed Time*	*Metric*	*Planning Time*	*Heuristic Time*	*Search Time*	*Expanded Nodes*	*States Evaluated*
delta = 1	98	62	76	917	396	584	8 845	15 420
delta = 0.5	145	54.5	69	1 088	445	709	9 313	16 259
delta = 0.1	577	54.1	69	1 354	445	709	9 313	16 259
delta = 0.05	1 113	54.05	69	1 447	390	955	9 313	16 259

Table 3.5: Second problem instance plan with different World Evolution Delta, with default planner.

Chapter 4

Results and conclusions

4.1 Choice of the planner

The results that we reported in chapter 3 in the tables: 3.1, 3.2, 3.3, 3.4 show how there not exists a unique planner that gives us an optimal plan for every problem instance. More precisely we noticed that:

- For very simple problems (very few crates to be loaded), such as the first instance, the best planner is *opt-blind*. This configuration expands a great number of nodes because of its "blind" nature, indeed the average planning time is greater than other planners. On the other hand, it reduces the elapsed time to the smallest value among the other ones, meaning that it finds the best plan. We tried to use this planner with more complicated instances, but it always runs out of memory.
- To obtain an optimal plan, with simple or not so many complicated instances such as the first, second, and third problem, the best planner resulted being *opt-hmax*. This planner expands a great number of nodes and the planning time is acceptable (it always took less than two minutes to generate an output). The generated elapsed time is also very low, compared to the others. Also in this case, with very complicated problems it runs out of memory.
- Among the planners that always can generate an output, the best one is *sat-hmrph*. This is a sat-planner, this means that it generates a plan that reaches the goal state in a non-optimal way. However, it is a good compromise between the elapsed time and the number of evaluated states.

  For this reasons, we choose *sat-hmrph* as the preferred planner.

Concerning the *World Evolution Delta*, as we can see from table 3.5 the elapsed time of the plan reduces from $delta = 1$ to $delta = 0.5$ but it doesn't change a lot if we keep reducing the value of delta. So it seems that the best trade-off about the choice of the discretization factor to select is a *world evolution delta* equal to 0.5. As regards the *Planning Delta*, we decided to leave the default value.

4.2 Output discussion

As told in the section section 4.1 if we reduce the *World Evolution Delta* using the default planner we obtain a smaller elapsed time. But if we do the same thing with the *sat-hmrph* planner we sometimes obtain a higher one. However, we obtain a better plan quality (in terms of both elapsed time and expanded nodes) if we use the *sat-hmrph* planner with a *World Evolution Delta* equal to 1. For this reason, we submitted the output files using this latter configuration. Here below we report the table with the generated output of each instance planned with the *World Evolution Delta* equal to 1 and the *sat-hmrph* planner.

Instance/	*Plan Length*	*Elapsed Time*	*Metric*	*Planning Time*	*Heuristic Time*	*Search Time*	*Expanded Nodes*	*States Evaluated*
Instance 1	55	30	41	306	42	75	214	322
Instance 2	83	51	65	1 582	1 032	1 362	46 009	53 596
Instance 3	138	102	118	1 480	974	1 256	34 141	50 768
Instance 4	114	62	85	714	340	455	6 211	8 255

Table 4.1: Final results. *World Evolution Delta* set to 1 and *sat-hmrph* as planner

In the output files we submitted, there are printed both actions and events, to show what happens during the execution.

Some hints to interpret the output files:

- *new_order* starts the mover.
- The variables of *new_order* and *reached_crate* must be the same.
- All actions and events with *2* needs two movers.
- The first crate put on belt activates *first_of_sequence*.
- Crates that need to be loaded consequently must be *near_in_sequence*.
- Between *put_down* and *new_order* the robot is recharging.

Bibliography

[1] Vallati, Mauro. Search and Heuristics for Classical Planning. Sept. 2022.

[2] E. Scala, P. Haslum, S. Thiebaux: Heuristics for Numeric Planning via Subgoaling, IJCAI 2016

[3] E. Scala, P. Haslum, S. Thiebaux, M. Ramirez, Interval-Based Relaxation for General Numeric Planning, ECAI 2016

[4] E. Scala, P. Haslum, D. Magazzeni, S. Thiebaux: Landmarks for Numeric Planning Problems, IJCAI 2017

[5] M. Ramirez, E. Scala, P. Haslum, S. Thiebaux: Numerical Integration and Dynamic Discretization in Heuristic Search Planning over Hybrid Domains in arXiv

[6] D. Li, E. Scala, P. Haslum, S. Bogomolov Effect-Abstraction Based Relaxation for Linear Numeric Planning In IJCAI 2018

[7] E. Scala, P. Haslum, S. Thiebaux, M. Ramirez Subgoaling Techniques for Satisficing and Optimal Numeric Planning in JAIR 2020

[8] Sites.google.com. 2022. ENHSP - How to Use It. [online] Available at: https://sites.google.com/view/enhsp/home/how-to-use-it [Accessed 10 May 2022].