Je suis pleinement conscient(e) que le plagiat de documents ou d'une partie de document constitue une fraude caractrisée.
Nom, date et signature :

INFORMATIQUE MASTER 1 - TER

Jonathan GRANIER
·
Supervised by : Damien PELLIER ·
Abdeldjalil RAMOUL

June 2017

Abstract The planning is a branch of AI. Among the several techniques of resolution, one of them is the HTN. This technique needs more human implication but is more efficient than other conventional techniques (PDDL). Also, i created a set of benchmarks for two HTN algorithms : an older one, SHOP, and a newer one, GTOHP. Finally , we observed that GTOHP is more efficient than SHOP and PDDL and is more flexible.

Keywords Planning · PDDL · HTN · Benchmark

1 Introduction

For my TER, i worked on several problems' planning. Among the many techniques of resolution, there is the Hierarchical Task Network (HTN) which is usually more efficient than conventional techniques (PDDL). Among the existing algorithms of HTN, i compared two : SHOP and GTOHP. So the question is to know which one is the fastest and if they are more efficient than the traditional approaches. To answer that question , i made some benchmarks for SHOP and GTOHP, from the problems' planning. Firstly we will talk about the traditional approach of the planning and the Hierarchical Task Network. Then we will see the problems' planning and how i created the benchmarks. Finally , we will see the the results of the comparison of the two planners SHOP and GTOHP and the traditional approach.

GRANIER Jonathan
5 Chemin de barletire
Tel.: 0675119970
E-mail: jonathan.granier@etu.univ-grenoble-alpes.fr

2 Planning Domain Definition Language (PDDL)

The planning is a branch of artificial intelligence that focus on the realization of strategies or action sequences (typically for execution by autonomous robot, etc ...).
So, we have a set of atomic actions and starting from an initial situation, we need to arrange these actions to achieve a goal in such a way that the plan is as small as possible.

2.1 Syntax

The Planning Domain Definition Language is an attempt to standardize Artificial Intelligence planning languages. In the usual form, we have two types of files :

- Domain files : In these files , we define the types of the different objects, the predicates on those types and the actions which acts on the predicates. (Example page 12)
- Problem files : Each of these files define a problem linked with a domain. A problem is defined by three things : a set of typed objects, a set of predicates which determines the initial state and a set of predicates which determines the goal.

2.2 Algorithm

The algorithm is an A* (A star) . The initial state is the root of the tree. Each action is a node. The main difficulty is to find a good heuristic . Generally , the heuristics are adapted to certain problems but not all. For my TER i used FastDownWard, an implementation of PDDL's algorithm in C.

3 Hierarchical Task Network (HTN)

3.1 Principle and general syntax

The major problem of PDDL's algorithm is the resolution time. Indeed the algorithms' heuristics are generic and not efficient for complex problems.

This why a new method to solve planning problems has been created : The Hierarchical Task Network (HTN). Unlike conventional approaches of the planning, HTN works by recursive decomposition of a complex task in sub-tasks until each sub-task can be achieved by an action execution.

This decomposition is represented by the addition of methods which are like actions. Each method has parameters and a pre-condition which must be fulfilled to execute the method. But a method in itself does not have any side effects, instead, there is a task list which contains some action or methods.

In a domain , like in classic PDDL's syntax , all variable are typed but abstract.

In a problem , in addition to what is defined in a PDDL's problem, we define a task list which is a sequence of several method call but, like the goal, this time with as parameters real instantiated objects declared in the problem.

To make a recursive or conditional method, we must override that method (same name and parameters). The conditions are made in the pre-condition and predicates.

Thus, in my TER i used two HTN's algorithm : GTOHP and SHOP, which have been implemented in Java, with their own syntax.

3.2 SHOP

SHOP is an old algorithm implemented by Mr. PELLIER. The syntax is very different from the PDDL's syntax. It is not very verbose and doesn't have types or predicate declarations. The solver finds the predicates on its own. Furthermore , in the problem , SHOP doesn't have goal, only a world's initialisation and a task list. This feature does not ensure the same final state of the world as the PDDL solvers. (Example page 13)

Algorithm

The algorithm is similar to the PDDL's algorithm without heuristic. For each method in the problem, it will look for all the objects that match with the predicates of the precondition. When it finds a set that works, it goes into the method and tries to execute the tasks keeping the object already selected. if it doesn't find a set of suitable objects, it will rewind the tree to change the set of objects or go to an overridden version of the method. The algorithm completes when it doesn't have any more methods , only actions. An important note, SHOP doesn't have global informations about the problem.

3.3 GTOHP

GTOHP is a new algorithm implemented by Mr.RAMOUL and the syntax is based on the PDDL's syntax. In contrary to the SHOP it has a verbose syntax. But the major feature is the identification of tasks (with numbers). In a method, it allows the addition of constraints between two tasks and ensure a specific state of the world after the execution of a method. Also, in a problem , it has a goal. This feature allows two things :

- The final state of the world is assured, unlike SHOP.
- To create a recursive main method in the domain which will allows to have only one tasks in a problem. The break point is made by the achievement of the goal.

Algorithm

The algorithm works in 2 steps : Pre-processing and resolution.

1. The pre-processing step consists of an instantiation of all methods with real objects. An instantiated method is a method in which all abstract variables have been replaced by a real object coming from the problem. Then it removes all inaccessible actions and methods, making simplifications with the invariant predicates.
2. The resolution step is similar to the PDDL algorithm. It's a tree with nodes which represent a step of the world. However the solver starts with the defined tasks order in the problem and it only needs to choose the good set of real objects to execute a method or an action. It follows the decomposition and it winds up the nodes when it can't go further. Also , the solver will test the overridden methods in the declaration order in the domain and execute the first method which pre-condition is validated in its steps.

4 Contributions

In the context of my TER , my job was to make benchmarks for the algorithm GTOHP and SHOP . Furthermore i made a number of tools to help me in my work.

4.1 Domains

In this section , i will describe my work. There are 11 domains , but the protocol to create a HTN's domain is always the same :

1. Understand the domain with the objects, predicates, actions and plan generated with PDDL
2. In GTOHP's syntax, Find a good decomposition and don't forget the peculiar cases.
3. if it's possible : make a unique task instead of many task.
4. Optimize the resolution time, in particular the pre-processing. Indeed each variable in a method will increase the generated methods and so the pre-processing time.
5. Test and Debug.
6. Translate, and adapt in SHOP's syntax.
7. Test and Debug.

4.1.1 Barman

This domain defines the realisation of a set of cocktails by a barman. Each cocktail is composed of two ingredients. To do the cocktail, the barman has at

his disposal a set of shots to serve the cocktails, one dispenser per ingredients, a shaker and his two hands. (Schema page 14)

Here the decomposition is quite simple. We serve the first ingredients in a shot and then pour it into the shaker. Do the same with the second ingredient. Shake and serve the cocktail.

One important thing is that there is a concept of dirtiness of shots. We must always pour an ingredient or a cocktail in a clean shot and a clean shaker. Unless it contained the same ingredient before.

However, the plans are not optimal. The tasks order defined by a problem will influence the size of the plan. Indeed , when making a cocktail in the shaker, we have two doses of this cocktail. But if and only if two equivalent cocktails follow each other in the tasks then we can use this double dose. Likewise with clean or dirty shot. If the tasks are defined in the correct order, we can avoid the cleaning of certain shots. But this method amounts to solve the problem "manually". Including this possibility in the domain explodes the problem's complexity.(Curves page 22)

4.1.2 Blocksworld

This domain is a classic domain. We have several blocks stacked on each other and we need to build block towers in a specific order. (Schema page 15)

It's a recursive domain because to pick a block, it needs to be clear , which means it must not have a block above it. To clear a block , we clear the block above and we unstack this block.

Here the plan is optimal because we always need to build one tower and we can't move a block pile. So the tower is built block by block.

However, we have a difference between GTOHP and SHOP. Indeed, with GTOHP we don't need to indicate the base block of the stack. We allow it, when it stacks a block A on another block B, to put the block B on the table first. With SHOP it's impossible to do that.(Curves page 5)

4.1.3 Depots

This domain is similar to the Blocksworld. We need to stack crates on pallets and we have trucks and hoists to carry this crates. They are all in distributor or depot (Only trucks can move between distributor and depot). So we have the same problem with blockworld to clean the crates and the surfaces where they have to go. While clearing a crate, all the crates above that one are put inside the truck. (Schema page 16)

Like Barman the plans are not guaranteed to be optimal. Indeed the tasks order will influence the plan's size.

For example if we have 2 crate , A and B ,A is on B and B is on pallet01 . The goal is : "A is on pallet02 and B is on A". If i say put A on pallet01 and put B on A. The truck will do two round trips even if the optimal solution is with only one trip. (Curves page 23)

4.1.4 Elevator

This domain defines one elevator and a set of people who called the elevator and want to go to a particular floor. Remark : From the 5Th floor going to floor 6 has the same cost as going to floor 12. (Schema page 17)

For this domain i used a single recursive task. As long as there is a passenger who is not served , take it and place it on the right floor. Moreover , on each floor where the elevator stops, it takes all the people who want to climb into the elevator and same thing for those who want to get off (It's a "check floor").

But this solution works with GTOHP but not with SHOP. Indeed, there is no goal in a SHOP's problem and it's the major shortcoming of SHOP. With a recursive task and without goal , the solver will go directly to the smaller decomposition and return a void plan because the domain does not have a predicate to prevent this. So i was obliged to make one task by passenger.

Finally , the plan is optimal with GTOHP but not with SHOP. The "check floor" didn't work, always do nothing.(Curves page23)

4.1.5 Gripper

This domain is very simple. We have one robot with two grippers and a set of balls in a room A. The goal is to transport all the balls in the room B. So it is just one recursive task who carries the balls two by two between the room A and the room B. (Schema page 18) This time , the predicates allow the recursion with SHOP. So the plan is optimal with the two solvers. (Curves page 24)

4.1.6 Hiking

This domain is a little complicated. We have a set of couples of persons and they want to go hiking. All of these people start at the first place and they want to go to the last place. They must therefore walk from place to place. But to walk from one place to another, the next place must have a tent up. So we have a set of tents and a set of cars to carry the tents. Only a car can carry a tent.(Schema page 19)

So it's again one recursive task.

Algorithm :

This solution works with GTOHP and SHOP but i can't guarantee the optimality of the plans' sizes.(Curves page 24)

4.1.7 Movie

This domain is just a sequential problem. We have a person who wants to watch a movie. To do this , he must rewind the movie , reset the counter and take some food (Chips, dip, pop, cheese and crackers).

The difference between the problems is the number of objects. So when i decomposed this domain , i had to be careful of the number of objects on each

Algorithm 1 Hiking Algorithm

```
while There are still places to go do
    Down a tent
    if There is only one car at the current place then
        Go to take a car at the previous place
    end if
    Carry the tent to the next place with two cars
    Up the tent
    Come back with one car
    for all Couple do
        Walk to the next place
    end for
end while
```

method to significantly reduce the time of pre-processing. The plans always have the same size and therefore they are optimal. (Curves page 25)

4.1.8 Mystery

This domain is a particular domain. Indeed all names have been substituted with improbable names. Once translated , we are get a similar domain to Depots where we need to carry some crates from a point A to a point B. This time the points are not all connected to each other. Instead they form a related graph. And we can pass only a limited number of times on one point.(Schema page20)

For the decomposition , the only difficulty is to do the travel between two points. And to do so we need to add two actions in the original domain : Visit and Unvisit. These methods allow to mark and remove the mark on a point. The number of passages on a point is noted by a level. And if a marking system is not used, because of the level on a point, the solver does not notice that he went through the same state. (Curves page 25)

So we do this :

Algorithm 2 Path algorithm

```
Move truck A to B
Visit B
Recursive call
Unvisit B
```

About the plans' size, the solutions are not optimal because we have the same problem as Depots.

4.1.9 Parking

This domain involves parking cars on a street where cars can be double-parked but not triple-parked. The goal is to find a plan to move from one configuration of parked cars to another configuration, by driving cars from one curb

location to another. Also there is one free curb space which guarantees solvability.(Schema page 21)

The decomposition of this problem is not simple. Indeed we have lot of particulars cases which explodes the number of methods. But here is the general idea :

Algorithm 3 Parking algorithm

for all Curb configuration (*curb* car_i car_j) **do**
  Clear car_i (He should not be at the back of a *curb*)
  Clear *curb*
  Put car_i in *curb*
  Clear car_j
  Put car_j in *curb*
end for

The size of the plans depend on the order of the tasks in the problem definition and so the optimality is not certain.(Curves page 26)

4.1.10 Satellite

This domain defines a set of satellites with some measuring tool. We have a set of space objects (Planet , solar, etc) . The goal is to make measurements.

The decomposition is simple :

Algorithm 4 Satellite algorithm

for all Measure to be made **do**
  Turn on the corresponding instrument on a satellite
  Orient the satellite for configuration
  Do the configuration
  Orient the satellite for measure
  Do the measure
end for

It is important to note that an instrument can make several types of measurement. The size of the plans depends on the first instrument turned on. However GTOHP and SHOP don't revert back enough in the search tree . Thus the plans are not optimal. (Curves page 26)

4.1.11 Zenotravel

Domain similar to the domain Elevator but with a plane and fuel management : when a plane no longer has gasoline we need to refuel the plane. It's the same decomposition , with the same problem between GTOHP and SHOP and logically the plans of GTOHP are smaller than SHOP's plans. But GTOHP have a bug and it does not always take the plane with the most gasoline. So

we need to make more refuel with GTOHP , which increases the size of the plans.(Curves page 27)

4.2 Tools developed

To help easing my work, i made two things.
A set of bash and R scripts for launching and getting the data : Plan and duration of execution. So the process is automatic between the execution of PDDL, GTOHP and SHOP with all domain and the generation of graph.

A converter between PDDL GTOHP and SHOP's problems. Indeed, a domain has a lot of problems and the difference between a PDDL's problem and a GTOHP's problem is just the task list. And between GTOHP and SHOP's problem , there is only a syntax difference. Though this converter (in C) is not a parser. Because of the lack of time, i had to "hard code" these tools but in theory a real parser and generator of problems is a better solution. Furthermore, we can add the SHOP's domain generator to the GTOHP's domain. Indeed like the problems, between the GTOHP's domain and SHOP's domain , we have the same amount of information and just a different syntax

4.3 Results and Analyse

From my experience, i have launched all the domains on the same machine with a maximum resolution time of 5 minutes. With all the data, i made 2 things :

- I have produced two graphs per domain. One graph on the resolution time of each solver and another one about the size of the plans. Important note : Every problem of a domain have a number (Depots have 20 problems numbered 1 from to 20) and each problem, the most of the time, is harder than the previous ones.
- Also, i calculated a score for every problem with agile method For a given problem let T* be the minimum time required by any planner to solve the problem. A planner that solves the problem in time T will get a score of $1/(1 + \log10(T/T^*))$ for the problem. Those that do not solve the problem get a score of 0. Runtime below 1 sec get the same score.

The results are mixed. First PDDL is good for two problems : Blocksworld because of the chosen heuristic which is very efficient and Movie because it's a small sequential problem.

Then , there are two type of problem : Those where GTOHP is clearly better, Barman and Parking. And the others which have an equivalent score between GTOHP and SHOP.

However, the domains between GTOHP and SHOP are not strictly equivalent. So, i tried making a decomposition more generalist in GTOHP which didn't work with SHOP.

Table 1: Score on the time

Domains	PDDL	GTOHP	SHOP	Max score
Barman	0	20	15.524	20
Blocksworld	22.114	33.874	35	35
Depots	3.264	17.113	22	22
Elevator	38.526	150	104.281	150
Gripper	5.301	20	14.191	20
Hiking	0.331	19.579	17.414	20
Movie	30	30	30	30
Mystery	14.180	24.909	9.77	30
Parking	0	18.922	5.721	20
Satellite	4.636	18.413	18.598	20
Zenotravel	7.807	17.612	19	19

Table 2: Score on the size of plans

Domains	PDDL	GTOHP	SHOP	Max score
Barman	0	20	18.752	20
Blocksworld	28	35	35	35
Depots	7	19.679	21.713	22
Elevator	55	149.361	140.447	150
Gripper	8	20	20	20
Hiking	1	19.666	19.888	20
Movie	30	30	30	30
Mystery	17	14.161	6.745	30
Parking	0	18.883	3	20
Satellite	6	18.741	17.724	20
Zenotravel	10	17.817	18.112	19

Finally, GTOHP is more efficient than PDDL and SHOP and it is more flexible than SHOP.

5 Conclusions

First , the protocol used for make my work wasn't the best one. Indeed, there are too much differences between GTOHP and SHOP's problem. I should have applied corrections on the GTOHP's domain after the translation into SHOP, or started first to right the domain in SHOP and not in GTOHP. Then the results obtained would have been more correct and representative of the differences between the algorithms.

The main problem with HTN is the decomposition. It is a human intervention that brings its share of error and imperfection which leads to bugs and uncertain quality on decomposition.

Thus, the next step would be to automatize the decomposition. Indeed, in my work, i spotted some patterns like repeating the same action for make an another action (clear a block by example). This is a harder problem which requires many thoughts and discussion.

References

1. MCDERMOTT, Drew, GHALLAB, Malik, HOWE, Adele, et al. PDDL-the planning domain definition language. 1998.
2. Fox, Maria, and Derek Long. "PDDL2. 1: An extension to PDDL for expressing temporal planning domains." J. Artif. Intell. Res.(JAIR) 20 (2003): 61-124.
3. Pellier, D. "PDDL4J planning library." Url: https://github. com/pellierd/pddl4j (2016).
4. Helmert, Malte. "The Fast Downward Planning System." J. Artif. Intell. Res.(JAIR) 26 (2006): 191-246.
5. Erol, Kutluhan, James Hendler, and Dana S. Nau. "HTN planning: Complexity and expressivity." AAAI. Vol. 94. 1994.
6. Ramoul, Abdeldjalil, et al. "HTN Planning Approach Using Fully Instantiated Problems." Tools with Artificial Intelligence (ICTAI), 2016 IEEE 28th International Conference on. IEEE, 2016.
7. Pellier, Damien. Modle dialectique pour la synthse de plans. Diss. Universit Joseph-Fourier-Grenoble I, 2005.
8. International Planning Competition 2014 - `https://helios.hud.ac.uk/scommv/IPC-14/domains_sequential.html`

Annexe 1 : Domain Example (Depots) in PDDL, GTOHP and SHOP

-

```
(:types place locatable - object
    depot distributor - place
        truck hoist surface - locatable
        pallet crate - surface)
```

Fig. 1: Type in PDDL/GTOHP

```
(:predicates (at ?x - locatable ?y - place)
             (on ?x - crate ?y - surface)
             (in ?x - crate ?y - truck)
             (lifting ?x - hoist ?y - crate)
             (available ?x - hoist)
             (clear ?x - surface))
```

Fig. 2: Predicates in PDDL/GTOHP

```
(:action Drive
:parameters (?x - truck ?y - place ?z - place)
:precondition (and (at ?x ?y))
:effect (and (not (at ?x ?y)) (at ?x ?z)))

(:action Lift
:parameters (?x - hoist ?y - crate ?z - surface ?p - place)
:precondition (and (at ?x ?p) (available ?x) (at ?y ?p) (on ?y ?z) (clear ?y))
:effect (and (not (at ?y ?p)) (lifting ?x ?y) (not (clear ?y)) (not (available ?x))
             (clear ?z) (not (on ?y ?z))))

(:action Drop
:parameters (?x - hoist ?y - crate ?z - surface ?p - place)
:precondition (and (at ?x ?p) (at ?z ?p) (clear ?z) (lifting ?x ?y))
:effect (and (available ?x) (not (lifting ?x ?y)) (at ?y ?p) (not (clear ?z)) (clear ?y)
        (on ?y ?z)))

(:action Load
:parameters (?x - hoist ?y - crate ?z - truck ?p - place)
:precondition (and (at ?x ?p) (at ?z ?p) (lifting ?x ?y))
:effect (and (not (lifting ?x ?y)) (in ?y ?z) (available ?x)))

(:action Unload
:parameters (?x - hoist ?y - crate ?z - truck ?p - place)
:precondition (and (at ?x ?p) (at ?z ?p) (available ?x) (in ?y ?z))
:effect (and (not (in ?y ?z)) (not (available ?x)) (lifting ?x ?y)))
```

Fig. 3: All Actions in PDDL/GTOHP

```
(:operator (!Drive ?x ?y ?z)
    (
        ; Predicates
        (truck ?x)
        (place ?y)
        (place ?z)
        (at ?x ?y)
    )
    (
        ;Negative effect
        (at ?x ?y)
    )
    (
        ;Positive effect
        (at ?x ?z)
    )
)
```

Fig. 4: Action Drive in SHOP

```
;-------------------------------------------------------------
;                       Clear a surface
;-------------------------------------------------------------

;; Clear the surface s1 in the place p1 et put what's on it on a truck

;;s1 is clear
(:method do_clear
        :parameters (?s1 - surface ?p1 - place)
        :expansion  (
                    ;Task list
                    (tag t1 (nop))      ; Empty action
                )
        :constraints(
                    and
                    ; Constraints before the first task
                    (before ( and
                            ( clear ?s1)
                            ( at ?s1 ?p1)
                            )
                    t1
                    )
                )
    )

;;s1 is not clear
(:method do_clear
        :parameters (?s1 - surface ?p1 - place)
        :expansion  (
                    ;Task list
                    (tag t1 (do_clear ?c ?p1))      ;Same method
                    (tag t2 (do_get_truck ?t ?p1))  ;Another method
                    (tag t3 (Lift ?h1 ?c ?s1 ?p1))  ;Action
                    (tag t4 (Load ?h1 ?c ?t ?p1))   ;Action
                )
        :constraints(
                    and
                    ; Constraints before the first task
                    (before ( and
                            ( not ( clear ?s1))
                            ( on ?c ?s1 )
                            ( at ?s1 ?p1)
                            ( at ?h1 ?p1)
                            )
                    t1
                    )
                )
    )
```

Fig. 5: Method do_clear in GTOHP

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;             Clear a surface                 ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;; Clear the surface s1 in the place p1 et put what's on it on a truck
(:method (do_clear ?s ?p)

;;s1 is clear
Case1   (
            ;Precondition
            ( surface ?s)
            ( place ?p)
            ( clear ?s)
            ( at ?s ?p)
        )
        (
            ;empty task list
        )
;;s1 is not clear
Case2   (
            ;Precondition
            ( surface ?s)
            ( place ?p)
            ( truck ?t)
            ( hoist ?h)
            ( crate ?c)
            ( on ?c ?s )
            ( at ?s ?p)
            ( at ?h ?p)
        )
        (
            ;Task list
            (do_clear ?c ?p)      ; same method
            (do_get_truck ?t ?p) ; Another method
            (!Lift ?h ?c ?s ?p)  ; Action
            (!Load ?h ?c ?t ?p)  ; Action
        )
)
```

Fig. 6: Method do_clear in SHOP

```
(define (problem depotprob1818 )
(:domain Depot )
    (:requirements :strips :typing :negative-preconditions :htn :equality)

(:objects
    depot0 - Depot
    distributor0 distributor1 - Distributor
    truck0 truck1 - Truck
    pallet0 pallet1 pallet2 - Pallet
    crate0 crate1 - Crate
    hoist0 hoist1 hoist2 - Hoist )
(:init
    (at pallet0 depot0 )
    (clear crate1 )
    (at pallet1 distributor0 )
    (clear crate0 )
    (at pallet2 distributor1 )
    (clear pallet2 )
    (at truck0 distributor1 )
    (at truck1 depot0 )
    (at hoist0 depot0 )
    (available hoist0 )
    (at hoist1 distributor0 )
    (available hoist1 )
    (at hoist2 distributor1 )
    (available hoist2 )
    (at crate0 distributor0 )
    (on crate0 pallet1 )
    (at crate1 depot0 )
    (on crate1 pallet0 )
 )
(:goal
    :tasks  (
                ;Task List
                (tag t1 (do_put_on crate1 pallet1))
                (tag t2 (do_put_on crate0 pallet2))


        )
    :constraints(and
            (after (and
                                ; Goal
                                (on crate0 pallet2 )
                                (on crate1 pallet1 )
    )
                t1)
        )
))
```

Fig. 7: GTOHP problem

```
(defproblem problem byPaser
(

    ; Init

    (depot depot0)
    (distributor distributor0)
    (distributor distributor1)
    (truck truck0)
    (truck truck1)
    (pallet pallet0)
    (pallet pallet1)
    (pallet pallet2)
    (crate crate0)
    (crate crate1)
    (hoist hoist0)
    (hoist hoist1)
    (hoist hoist2)
    (surface crate0)
    (surface crate1)
    (surface pallet0)
    (surface pallet1)
    (surface pallet2)
    (place depot0)
    (place distributor0)
    (place distributor1)

    (at pallet0 depot0  )
    (clear crate1  )
    (at pallet1 distributor0  )
    (clear crate0  )
    (at pallet2 distributor1  )
    (clear pallet2  )
    (at truck0 distributor1  )
    (at truck1 depot0  )
    (at hoist0 depot0  )
    (available hoist0  )
    (at hoist1 distributor0  )
    (available hoist1  )
    (at hoist2 distributor1  )
    (available hoist2  )
    (at crate0 distributor0  )
    (on crate0 pallet1  )
    (at crate1 depot0  )
    (on crate1 pallet0  )

)
    ;;;
    ;;;   goals
    ;;;


(
    (do_put_on crate1 pallet1 )
    (do_put_on crate0 pallet2 )

)
)
```

Fig. 8: SHOP problem

Annexe 2 : Schema

Barman

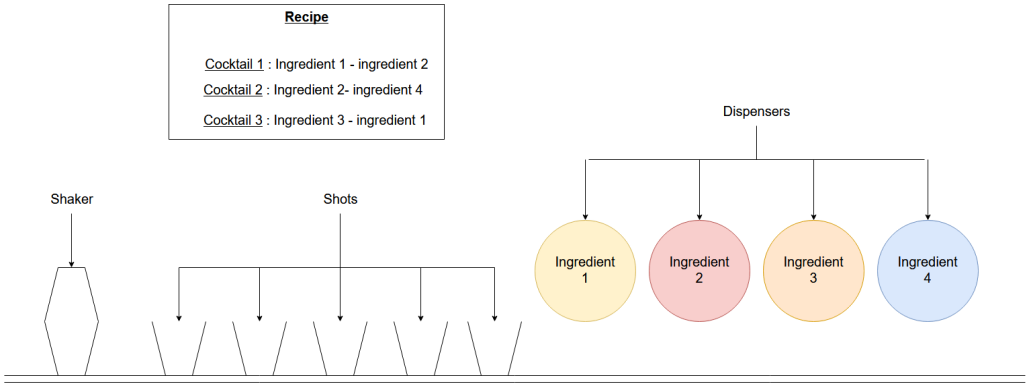


Fig. 9: Barman initial state

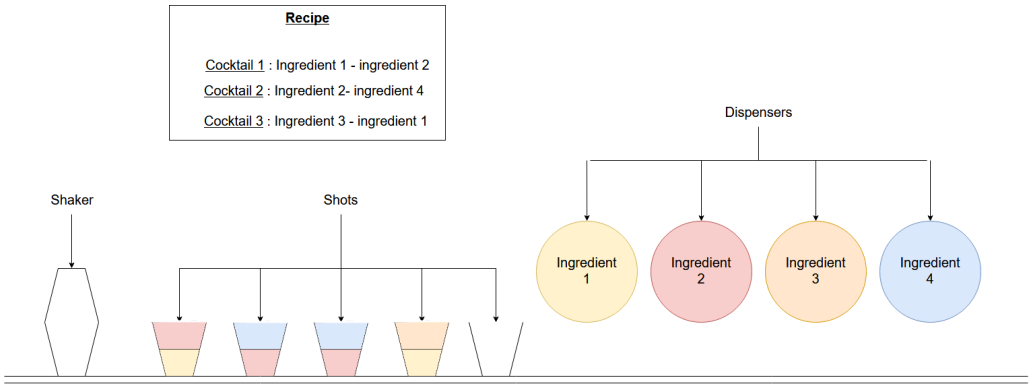


Fig. 10: Barman Goal state

Blocksworld

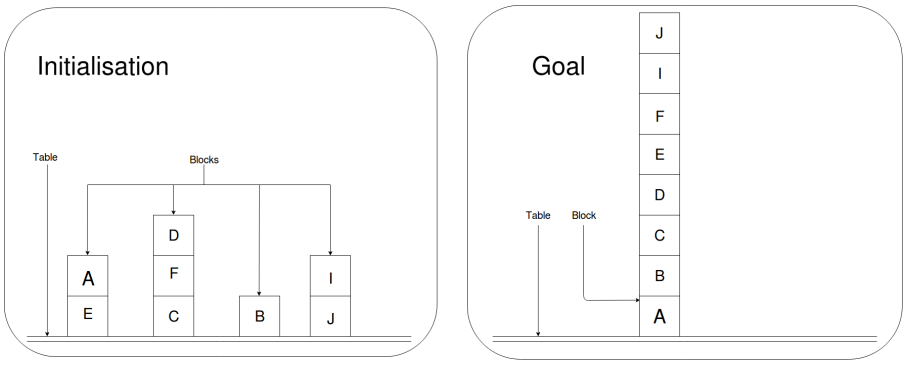
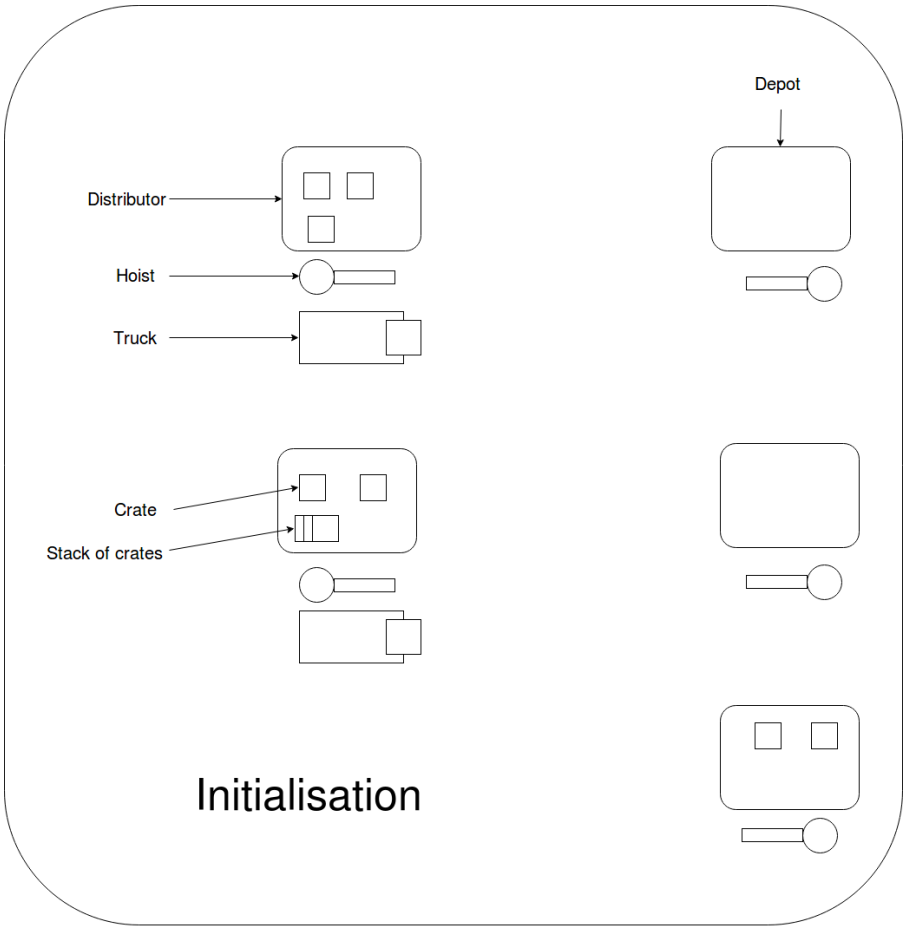
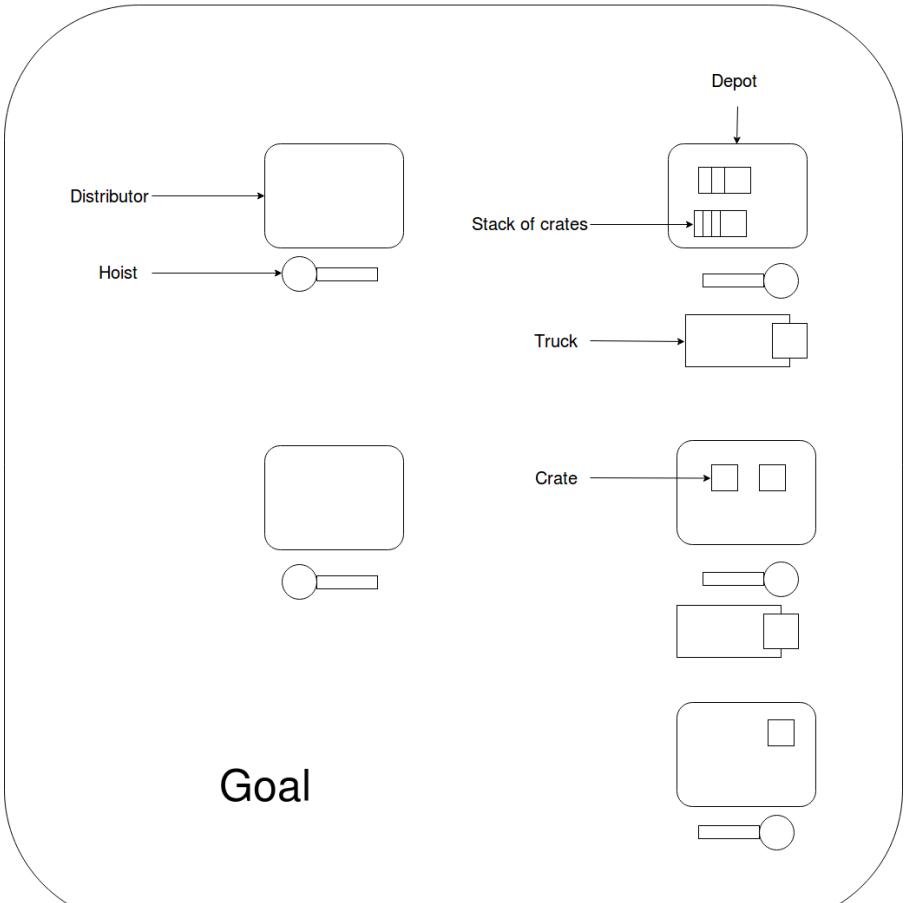


Fig. 11: Schema of Blocksworld

Depots


Depot
Distributor
Hoist
Truck
Crate
Stack of crates
Initialisation


Depot
Distributor
Hoist
Stack of crates
Truck
Crate
Goal

Elevator

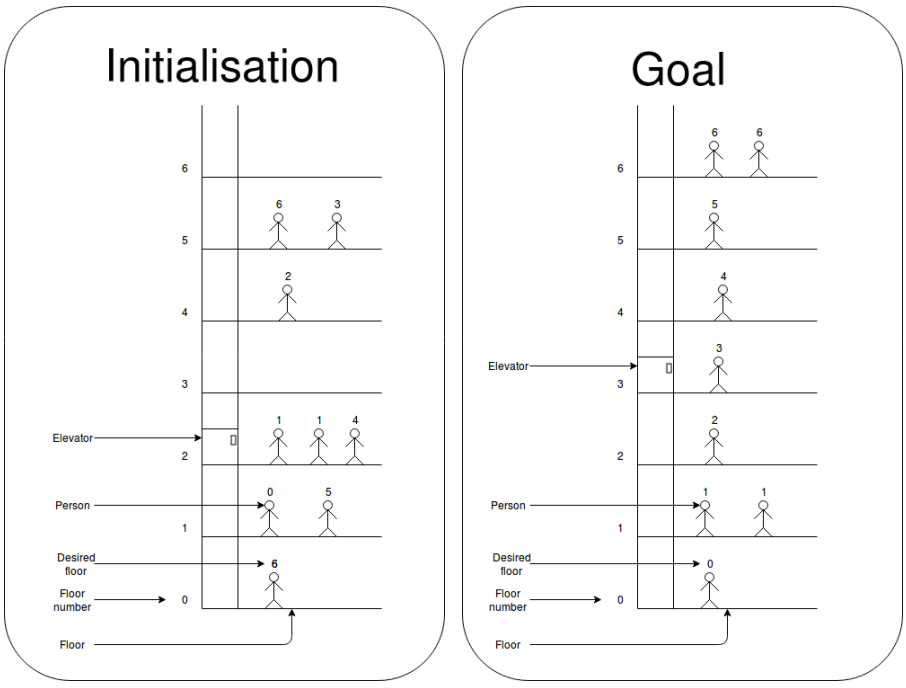


Fig. 13: Schema of Elevator

Gripper

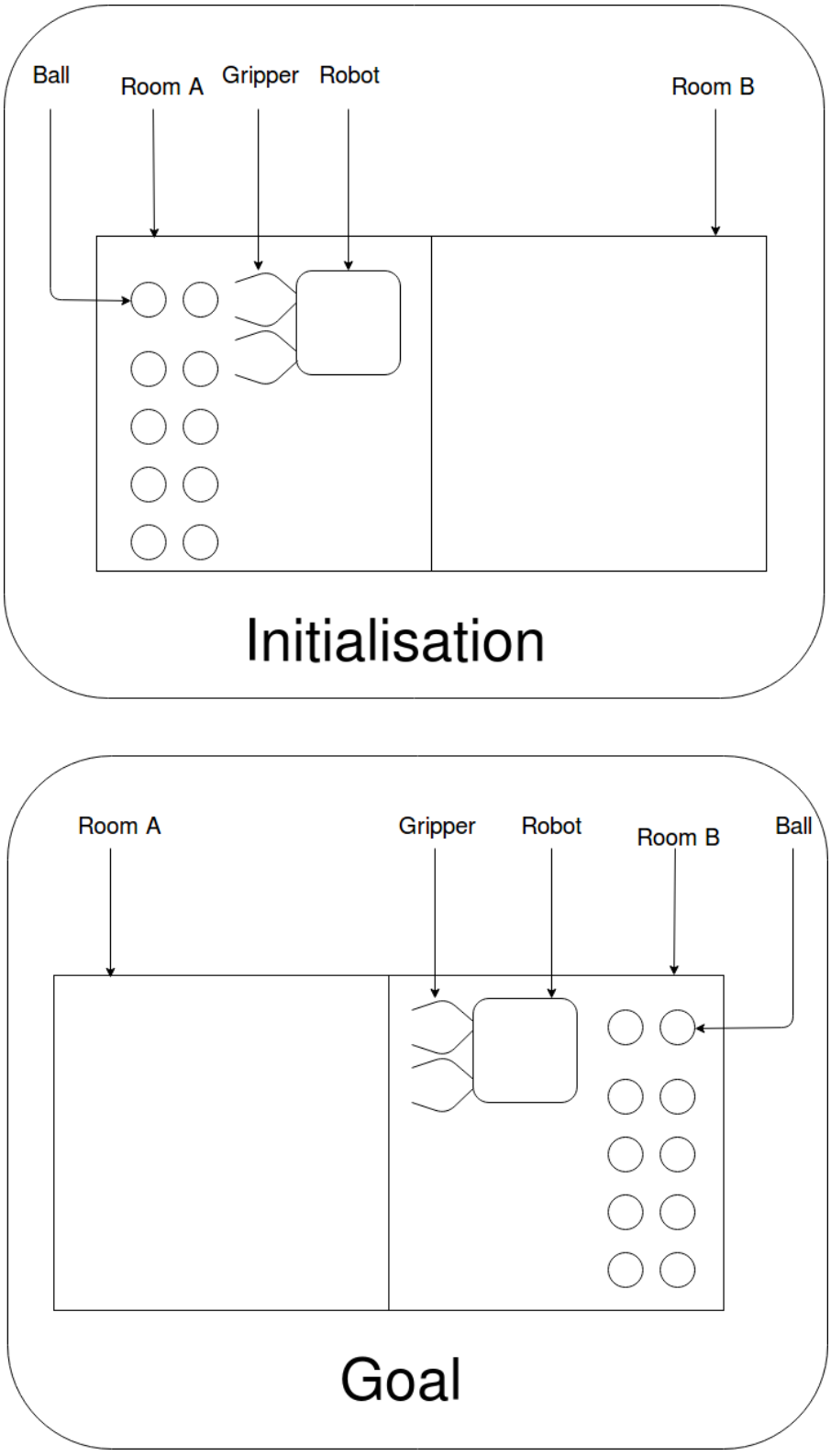


Fig. 14: Schema of Gripper

Hiking

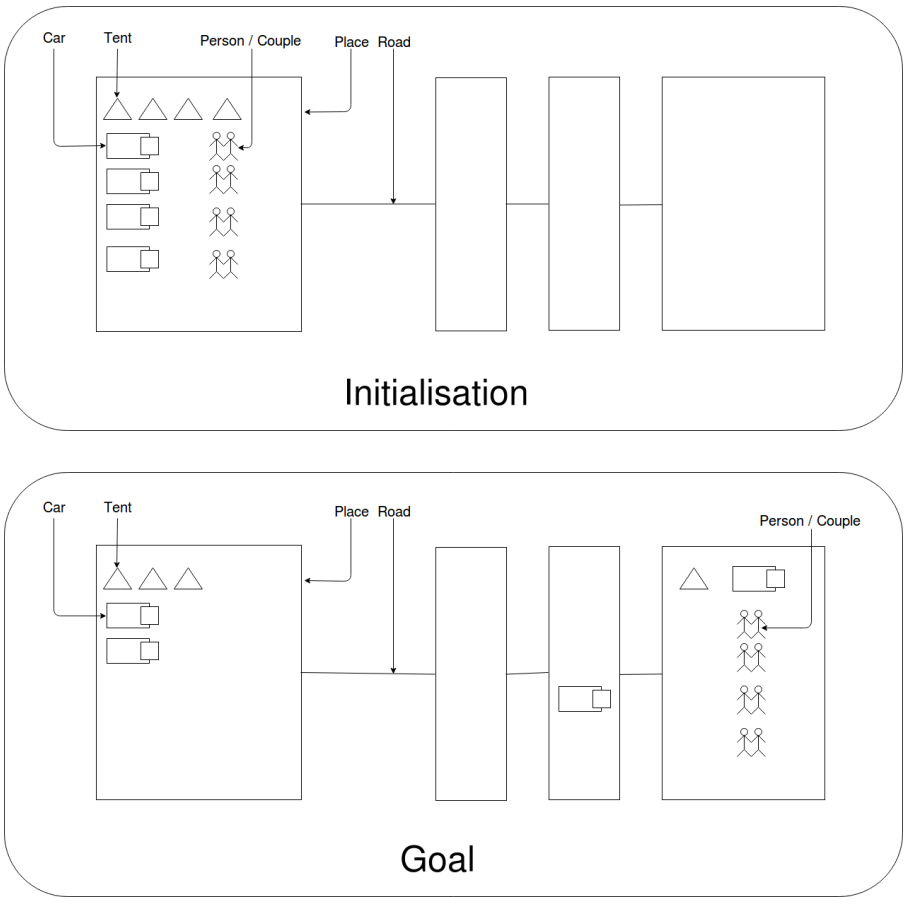


Fig. 15: Schema of Hiking

Mystery

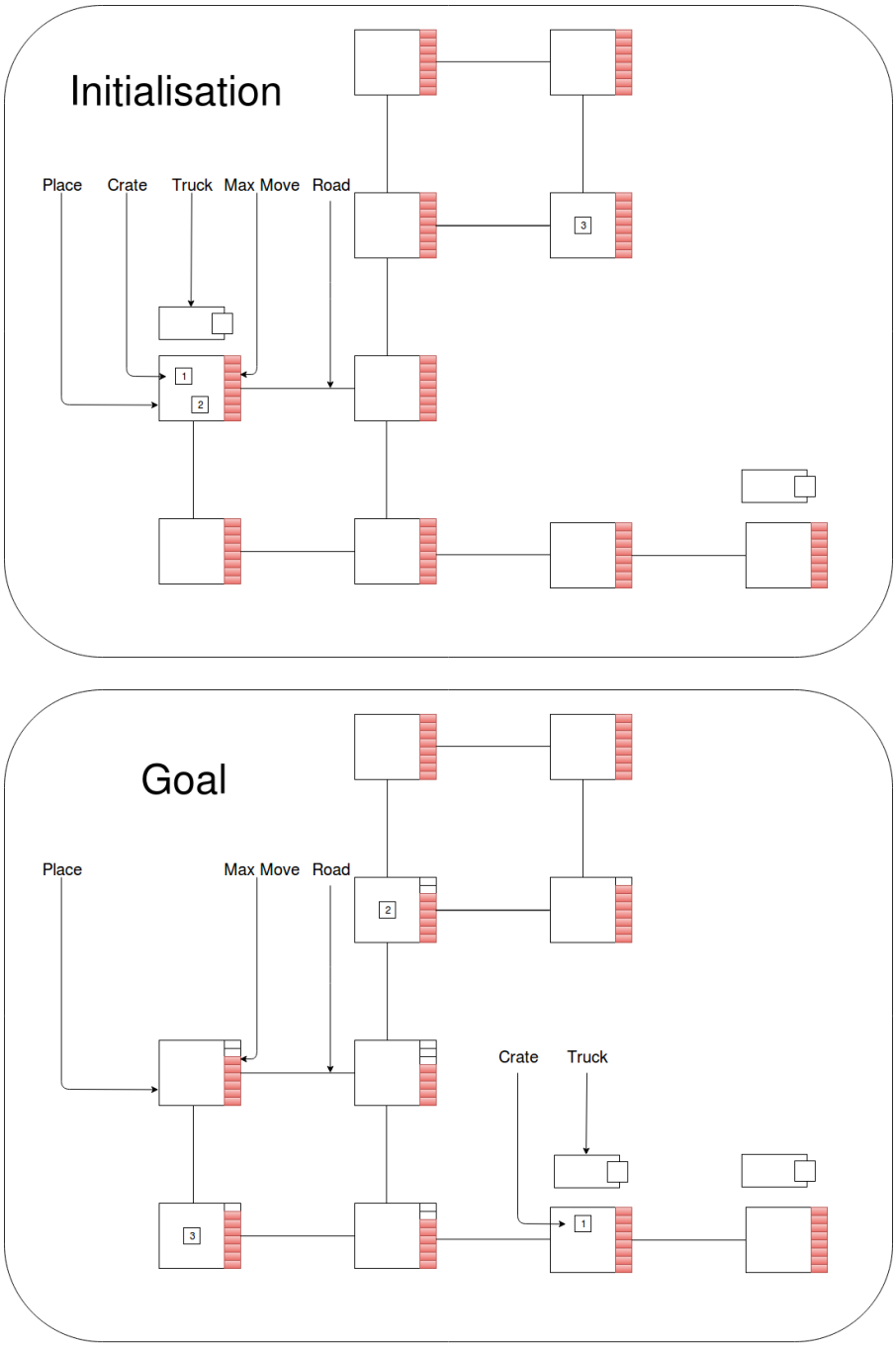


Fig. 16: Schema of Mystery

Parking

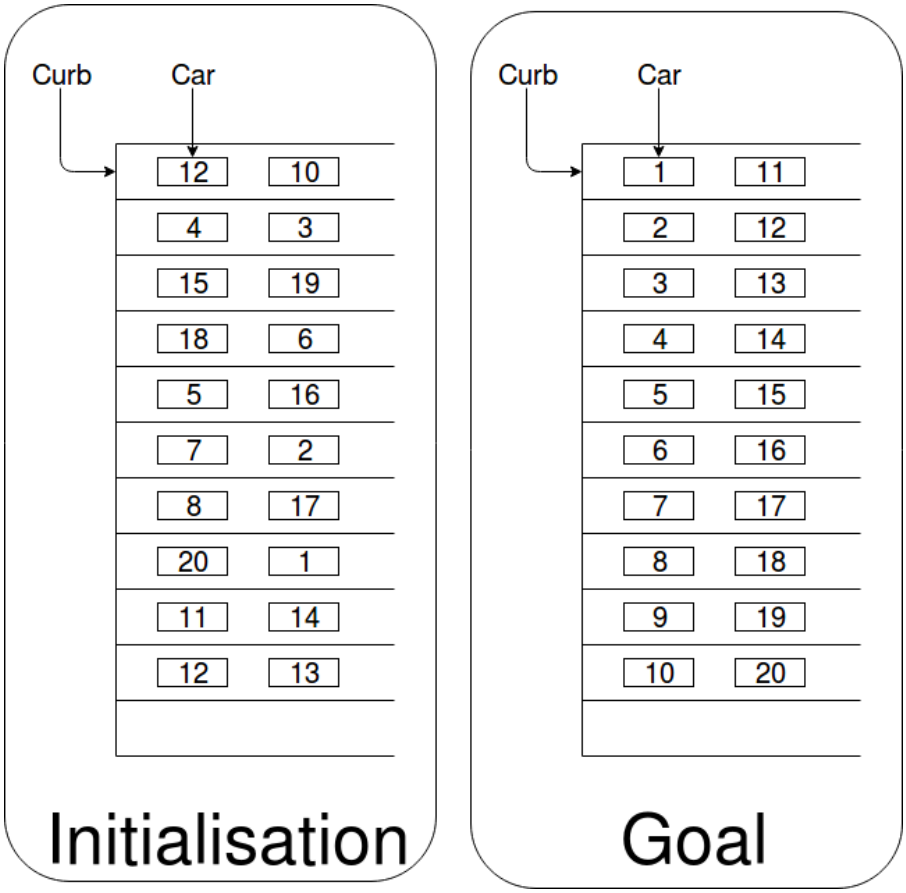
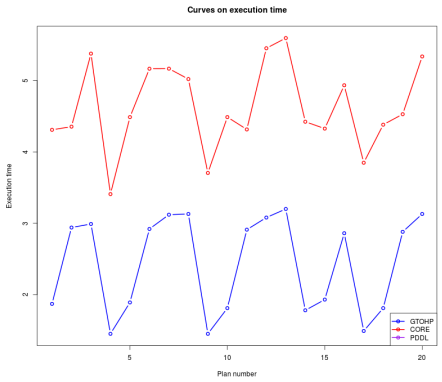


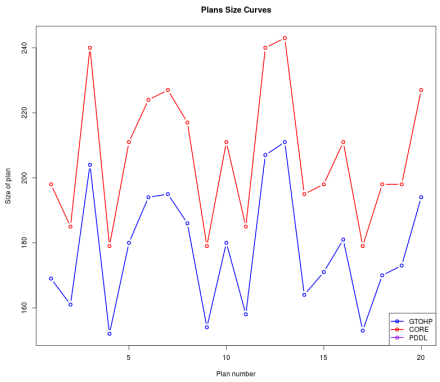
Fig. 17: Schema of Parking

Annexe 3 : Graph

Barman

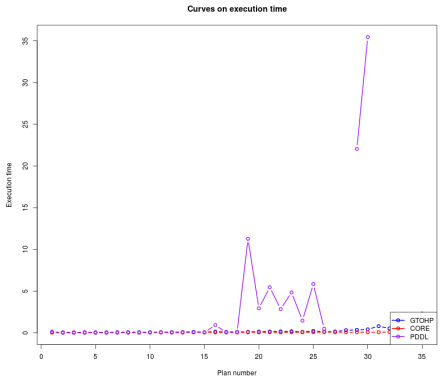


(a) Execution time

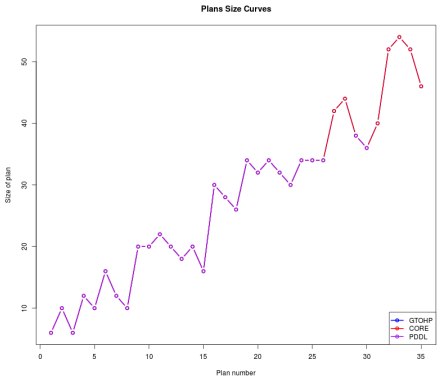


(b) Size of plan

Blocksworld

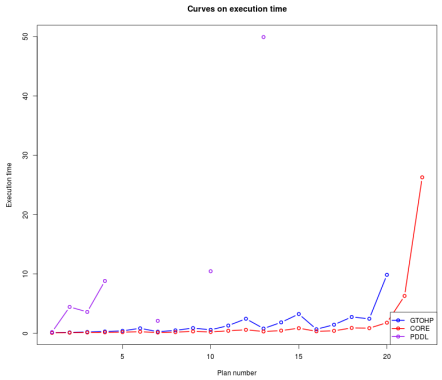


(c) Execution time

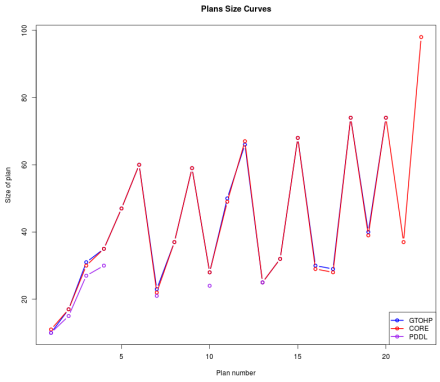


(d) Size of plan

Depots

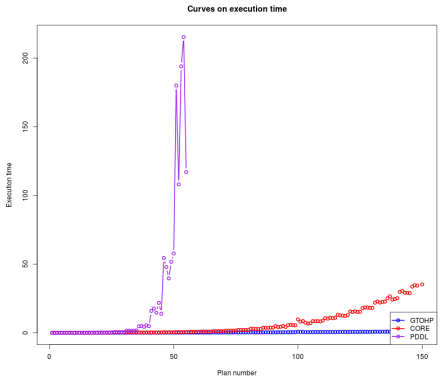


(e) Execution time

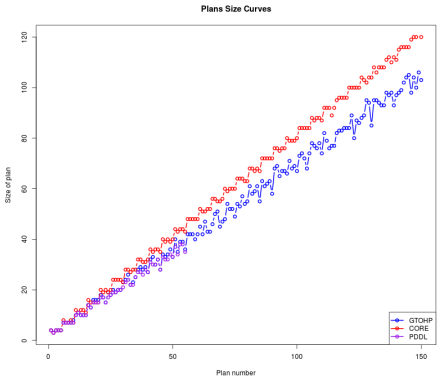


(f) Size of plan

Elevator

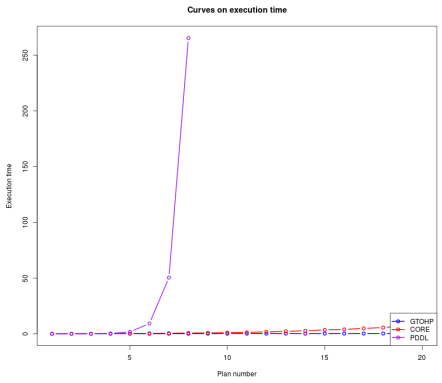


(g) Execution time

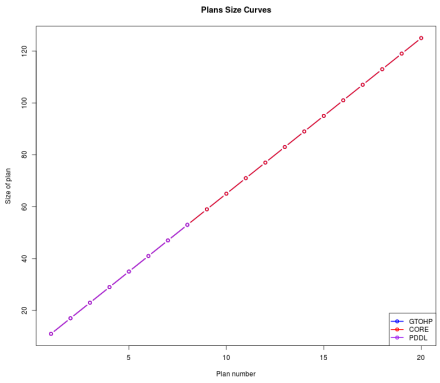


(h) Size of plan

Gripper

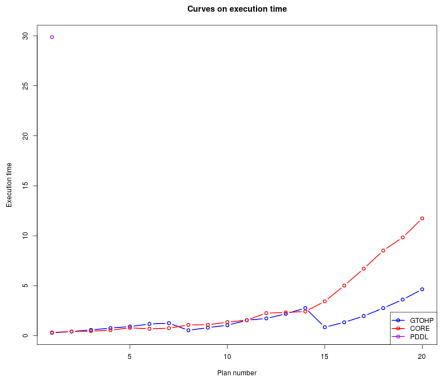


(i) Execution time

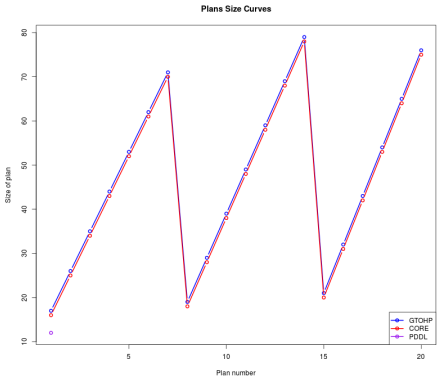


(j) Size of plan

Hiking

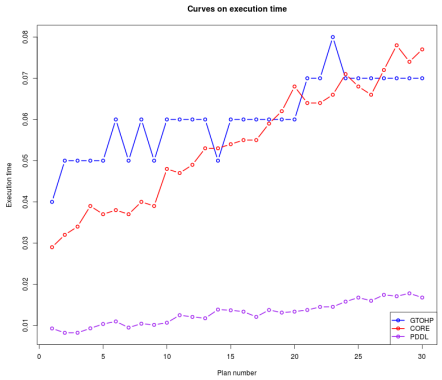


(k) Execution time

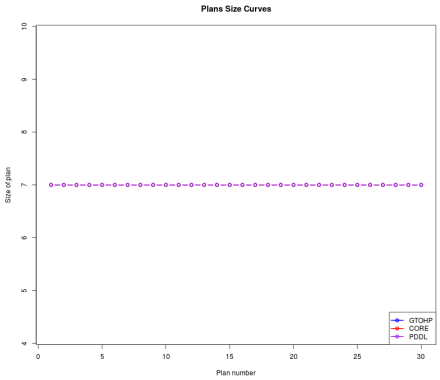


(l) Size of plan

Movie

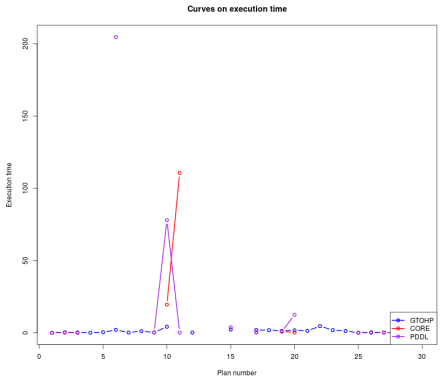


(m) Execution time

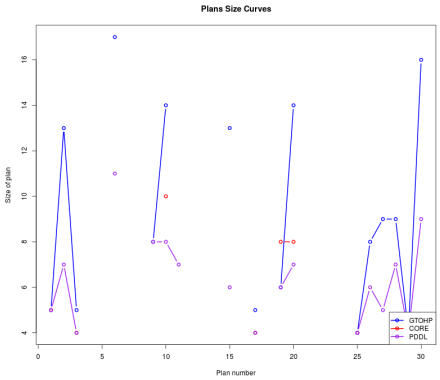


(n) Size of plan

Mystery

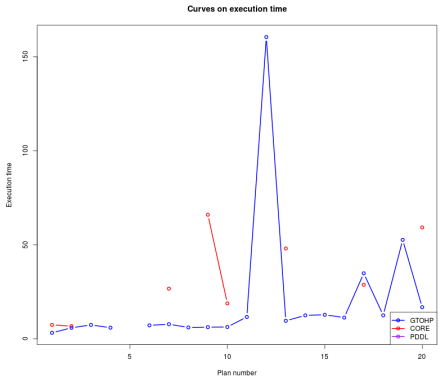


(o) Execution time

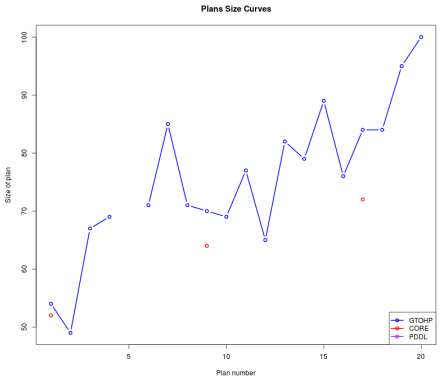


(p) Size of plan

Parking

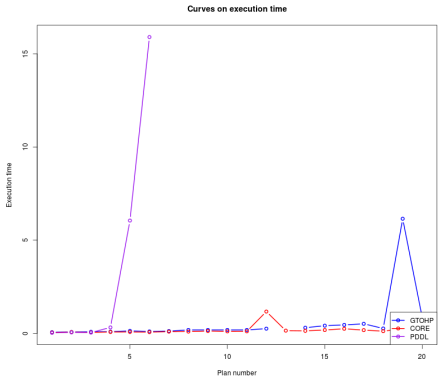


(q) Execution time

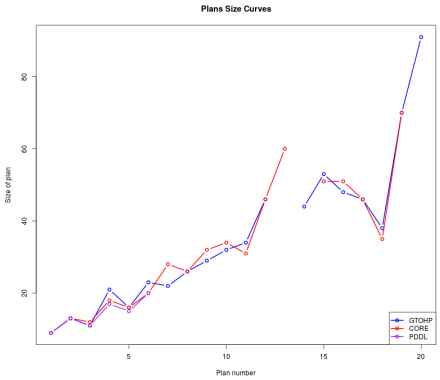


(r) Size of plan

Satellite

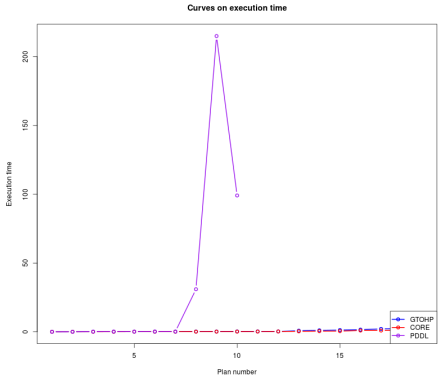


(s) Execution time

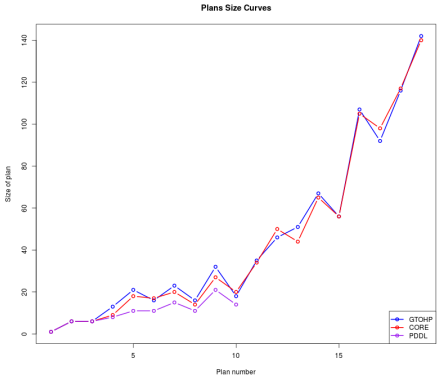


(t) Size of plan

Zenotravel



(u) Execution time



(v) Size of plan