TDDC17 Artificial Intelligence :

Lab 4 : Introduction to planning

Arnaud Pecoraro

October 2015

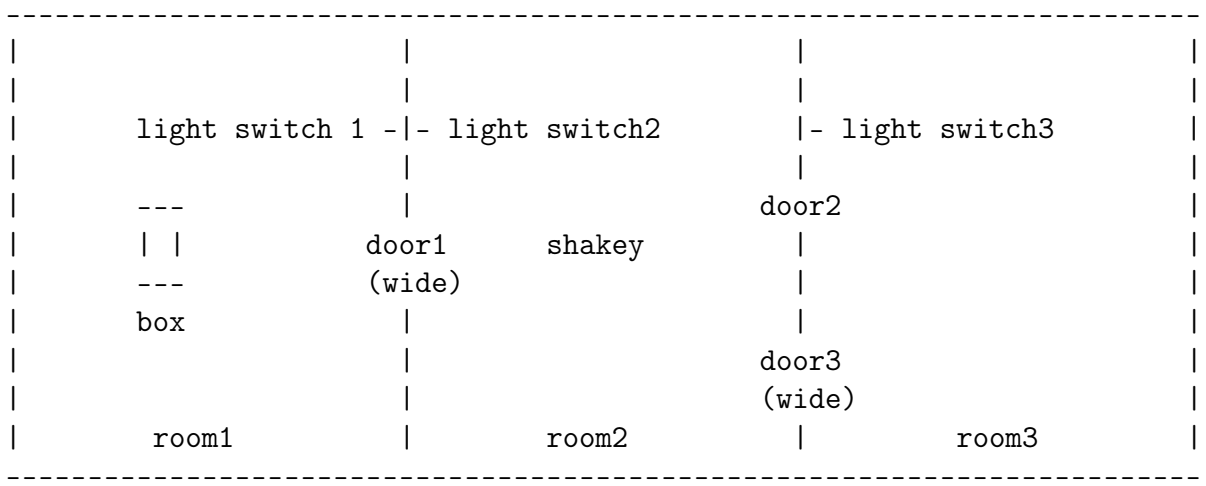
Task 1 : Writing Domain and Problem

The aim of this fourth lab is to learn how to model a problem that can be solved with automatic solvers called planners, using a formal language : PDDL.

Different domains alternatives are suggested. I have chosen to implement the AI classic Shakey's World.

Shakey is a robot with two grippers that is moving in a set of rooms, connected by doors. Each room contains a lightswitch which can be on or off but may also contain big boxes or smaller objects.

The lab page defines the following constraints :
— Shakey can carry small objects, but only two at a time because he has only two grippers.
— For Shakey to be able to find a small object in a room, the light must be on in that room.
— Shakey can not carry boxes, but can push them around. Boxes can only be pushed through wide doors, not narrow ones.
— To switch the light in a room on or off Shakey must climb onto a box that is positioned below the switch (otherwise he can not reach the switch). This may in turn require moving the box into position, possibly from another room.



SRI Shakey's World

The following actions have been implemented in the domain definition file : move, push, climb, descend, turnon and pickup and drop.

A few assumptions have been made :
— Doors are either opened wide or narrow, it is not possible to change the opening.
— Shakey pushing a box from a room to another imply Shakey also being in the destination room.
— When Shakey is on a box, he can only turn a light on, he has to reach the ground before being able to move.

Please refer to the code and comments for more details regarding the implementation.

Testing the domain with a simple problem

The example below is very simple problem just to test the domain. Shakey is in room-1 and must bring the ball from room-1 to room-3 and then go back to room-1. A climbable box is in room-1.

The goal is defined as follow :

```
(:goal (and(in ball room-3) (in shakey room-1 )))
```

Using *vhpop* planner on this simple problem we get the following solution :

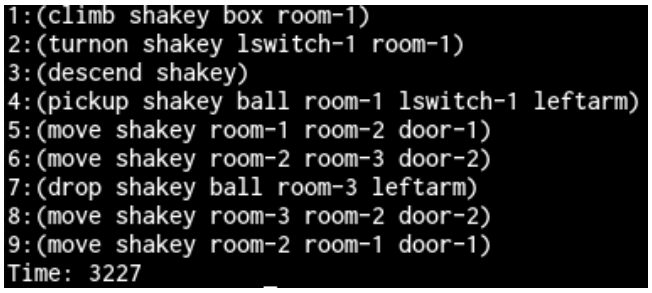


FIGURE 1 – Actions sequence using vhpop

Task 2 : Problems and performance comparison

In this part we are going to test and compare the different planners on a range of problems with increasing difficulty.

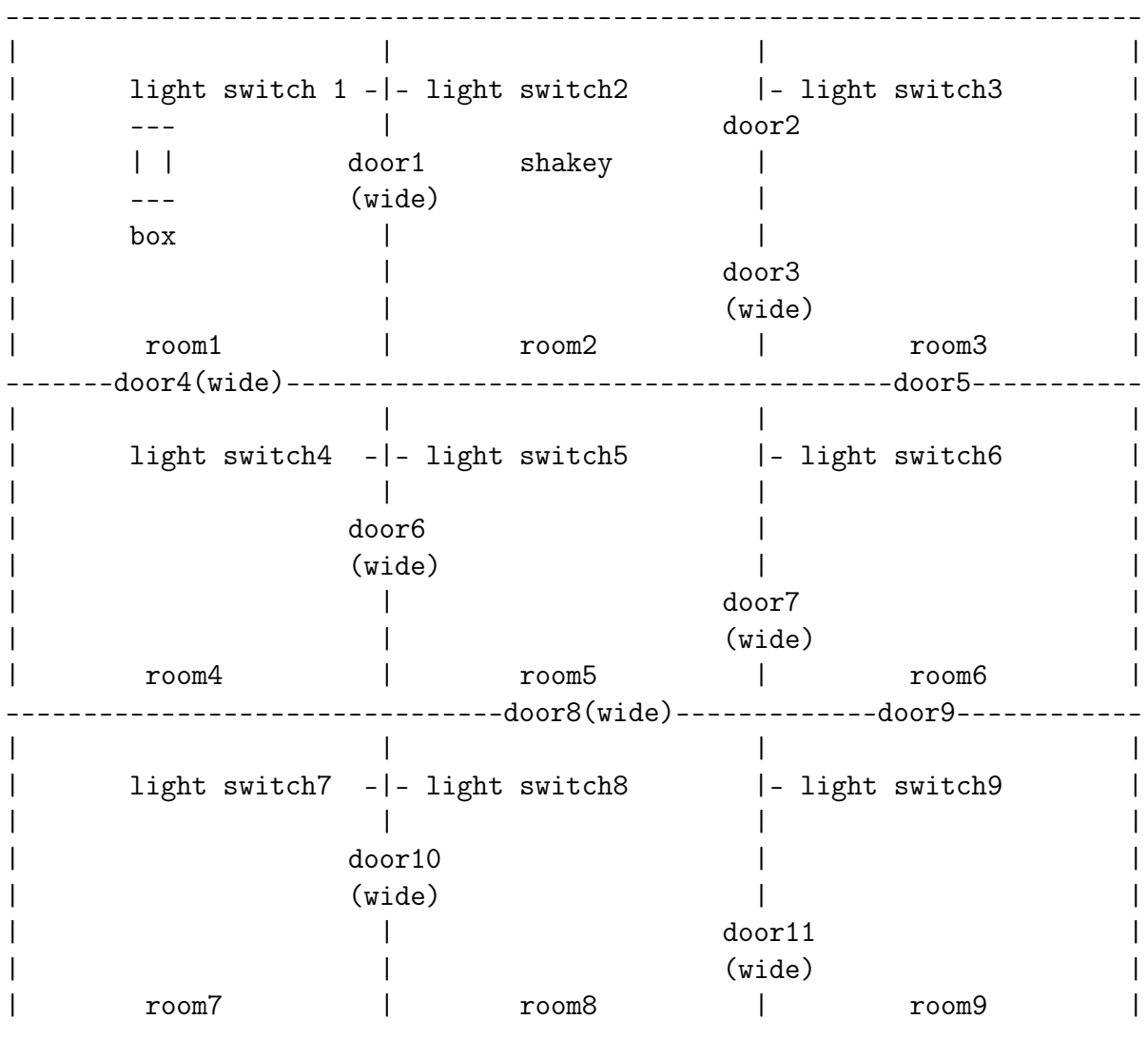
Identifying parameters

In order to create this range of problem, we first have to identify the parameters that can be modified to scale up the difficulty.

— World dimensions : the default domain (see page 1) is composed of only three rooms. If we add more rooms or change the disposition to a grid of (n,m) rooms, the complexity increases rapidly.
— The number of objects : The difficulty increases if we add more balls to carry to different places. If we limit the number of climbable boxes to 1, forcing Shakey to move to the box, then push it to the switch location, and so on...
— The initial state : If the initial state contains already achieved goal conditions, the complexity is reduced.

We are going to increase the number of balls and the world dimension.

The world dimension will be enlarged into a 3 by 3 room grid in the last problems.



SRI Shakey's World 9 rooms

Problems

Here are just the relevant inforations to understand the problems. For the complete description, please refer to the Shakey_probX.pddl files.

— *Shakey_prob1.pddl* : 3 rooms, 3 balls

```
(:init (in box room-2) (in ball-1 room-1) (in ball-2 room-2)
  (in ball-3 room-2) (in shakey room-1))

(:goal (and(in ball-1 room-3)
  (in ball-2 room-1) (in ball-3 room-3)(in shakey room-3)))
```

— *Shakey_prob2.pddl* : 3 rooms, 6 balls

```
(:init (in box room-2) (in ball-1 room-1)
  (in ball-2 room-2) (in ball-3 room-2) (in ball-4 room-1)
  (in ball-5 room-2) (in ball-6 room-3) (in shakey room-1))

(:goal (and(in ball-1 room-3)
  (in ball-2 room-1) (in ball-3 room-3)(in ball-4 room-3)
  (in ball-5 room-1) (in ball-6 room-1)(in shakey room-3)))
```

— *Shakey_prob3.pddl* : 3 rooms, 12 balls

```
(:init (in box room-2) (in ball-1 room-1)
  (in ball-2 room-2) (in ball-3 room-2) (in ball-4 room-1)
  (in ball-5 room-2) (in ball-6 room-3) (in ball-7 room-1)
  (in ball-8 room-2) (in ball-9 room-2) (in ball-10 room-1)
  (in ball-11 room-2) (in ball-12 room-3) (in shakey room-1))

(:goal (and(in ball-1 room-3)
  (in ball-2 room-1) (in ball-3 room-3)(in ball-4 room-3)
  (in ball-5 room-1) (in ball-6 room-1)(in ball-7 room-3)
  (in ball-8 room-1) (in ball-9 room-3)(in ball-10 room-2)
  (in ball-11 room-3) (in ball-12 room-1)(in shakey room-3)))
```

— *Shakey_prob4.pddl* : 9 rooms, 1 ball

```
(:init (in box room-1) (in ball-1 room-3) (in shakey room-2))

(:goal (in ball-1 room-9))
```

— *Shakey_prob5.pddl* : 9 rooms, 3 balls

```
(:init (in box room-1) (in ball-1 room-3) (in ball-2 room-6)
  (in ball-3 room-9) (in shakey room-2))

(:goal (and(in ball-1 room-9) (in ball-3 room-1) (in ball-2 room-7)))
```

— *Shakey_prob6.pddl* : 9 rooms, 6 balls

```
(:init (in box room-1) (in ball-1 room-3) (in ball-2 room-6)
  (in ball-3 room-9) (in ball-4 room-3) (in ball-5 room-3)
  (in ball-6 room-9) (in shakey room-2) (movable shakey))

(:goal
  (and(in ball-1 room-9) (in ball-3 room-1) (in ball-2 room-7)
  (in ball-4 room-9) (in ball-5 room-7) (in ball-6 room-2)))
```

Results

Test protocol : all tests have been done in the same conditions, on the same computer (Liu server, accessed via thinlinc). A time limit of 3600s has been choosen. If no solution has been found, the test failed.

The results on this problem set are presented in *Figure 2*.

Problem	LAMA	VHPOP	IPP	FF
0	0	2.584	0	0
1	0.07	FAIL	0.01	0
2	5.73	FAIL	5.94	0
3	FAIL	FAIL	FAIL	0.01
4	0.02	FAIL	0.24	0
5	28.35	FAIL	14.64	0
6	FAIL	FAIL	FAIL	0.01

(a)

Problem	LAMA	VHPOP	IPP	FF
0	9	9	9	9
1	16	FAIL	16	18
2	29	FAIL	29	37
3	FAIL	FAIL	FAIL	57
4	10	FAIL	10	10
5	32	FAIL	32	42
6	FAIL	FAIL	FAIL	59

(b)

FIGURE 2 – Running time in seconds (a) and path cost (b)

Firstly, we noticed that *VHPOP* failed in every problem but problem 0. In fact the process is automatically killed before reaching the choosen time limit.

As far as speed is concerned, *FF* is a clear winner. Every planning problem is almost solved instantly. However, except in problem 0 and 4, the solution is never optimal.

On the other hand *LAMA* and *IPP* always find the least-cost solution but the running time much longer as the problem become more difficult.

Note : in Figure 2, when LAMA "fails", it means that the algorithms did not finish the exhausted search in time and therefore may not have found the optimal solution yet. It did not mean that no solution where found.

Adding more problems to the sets

In order to better understand the influence of each parameters, I have added three more problems.

— *Shakey_prob2.5.pddl* : 3 rooms, 8 balls

```
(:init (in box room-2) (in ball-1 room-1) (in ball-2 room-2)
  (in ball-3 room-2) (in ball-4 room-1) (in ball-8 room-2)
  (in ball-10 room-1)(in ball-11 room-2) (in ball-12 room-3)
  (in shakey room-1))

  (:goal (and(in ball-1 room-3) (in ball-2 room-1) (in ball-3 room-3)
    (in ball-4 room-3) (in ball-8 room-1)(in ball-10 room-2)
    (in ball-11 room-3) (in ball-12 room-1)(in shakey room-3)))
```

— *Shakey_prob4.5.pddl* : 9 rooms, 2 balls

```
(:init  (in box room-1)(in ball-1 room-3)(in ball-2 room-6)
  (in ball-3 room-9) (in shakey room-2)

(:goal (and(in ball-1 room-9) (in ball-2 room-7)))
```

— *Shakey_prob5.5.pddl* : 9 rooms, 4 balls

```
(:init  (in box room-1) (in ball-1 room-3) (in ball-3 room-9)
  (in ball-5 room-3) (in ball-6 room-9) (in shakey room-2)

  (:goal (and(in ball-1 room-9) (in ball-3 room-1)
    (in ball-5 room-7) (in ball-6 room-2)))
```

Results are presented in the *Figure 3* below. Given the previous tests, *VHPOP* has not been used. The first solution returned by *LAMA* and its running time have been added to the comparison.

Problem	LAMA	LAMA 1Sol	IPP	FF
0	0	0	0	0
1	0.07	0	0.01	0
2	5.73	0.05	5.94	0
2.5	2721.26	0.18	585.02	0
3	FAIL	0.28	FAIL	0.01
4	0.02	0	0.24	0
4.5	1.63	0.01	0.54	0
5	28.35	0	14.64	0
5.5	268.74	0.19	6.21	0.01
6	FAIL	0.24	FAIL	0.01

(a)

Problem	LAMA	LAMA 1Sol	IPP	FF
0	9	9	9	9
1	16	20	16	18
2	29	33	29	37
2.5	33	39	33	43
3	FAIL	53	FAIL	57
4	10	10	10	10
4.5	21	29	21	23
5	32	44	32	42
5.5	29	38	29	38
6	FAIL	63	FAIL	59

(b)

FIGURE 3 – Running time in seconds (a) and path cost (b)

We notice with problems 2.5, 5.5, and 6 that adding more balls increases the difficulty drastically. A clear example is the difference between problem 5 and 5.5.

An interesting fact, is that except in the simpliests problems, the first solution returned by *LAMA* is never optimal. It is not as fast as *FF* and in this range of problem its optimality is comparable, sometimes better, sometimes worst see *Figure 3b* and *Figure 4*.

In this test set, the best solution for optimality would be *IPP*, it always find the optimal solution and sometimes way faster than *LAMA*'s exhaustive search, see problem 5.5 for example.

On the other hand, if speed is what matter the most, *FF* seems to be the best tradeoff.

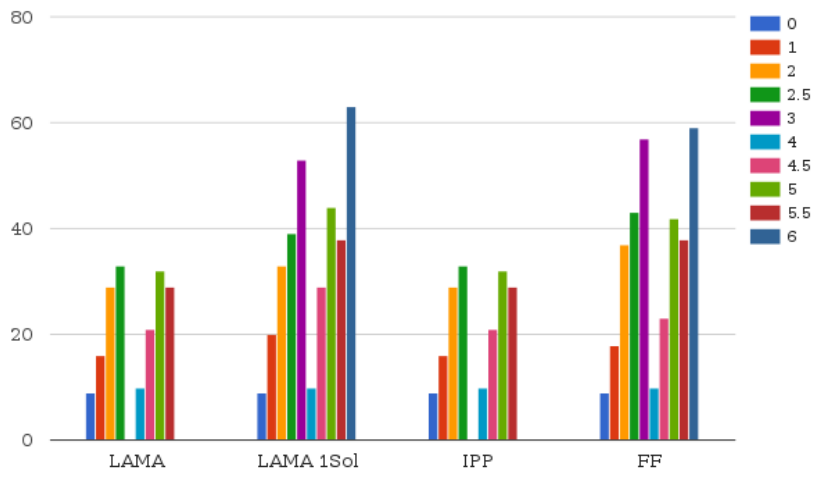


FIGURE 4 – Path cost comparison graph

As a conclusion, we notice that as planning problem grows larger, they cannot be solved optimaly in reasonable time. Here this is the case with problem 6.