AI Planning Assignment 2 Report

Marco Dinacci *
Huddersfield University

April 21, 2020

* ✉ u1972408@hud.ac.uk

Contents

ListofFigures

1 Introduction

1.1 Tools and Development

All PDDL files have been written from scratch in the Vim editor using a PDDL syntax highlighter Persson (2016).

During development plans were compiled using the *planning.domains* Muise (2015) solver which I triggered by reverse engineering the solver request format and writing a simple Python script to make a HTTP POST request. I could then trigger the python script and see the results using a simple keyboard shortcut in Vim. The main reason for this set-up is that I initially run into some issues compiling other planning engines on my laptop, which for work-related purposes must use an unstable, development version of MacOS SDK.

The README files contains instructions on how to reproduce the environment and re-run the experiments.

```
function findPlan() {
    var domText = window.ace.edit($('#domainSelection').find(':selected').val()).getSession().getValue();
    var probText = window.ace.edit($('#problemSelection').find(':selected').val()).getSession().getValue();

    window.solverURL = $('#plannerURL').val();
    if (window.solverURL.slice(-1) === "/")
        window.solverURL = window.solverURL.slice(0, window.solverURL.length-1);

    window.toastr.info('Running remote planner...');

    $('#chooseFilesModal').modal('toggle');

    $.ajax( {url: window.solverURL + "/solve",
             type: "POST",
             contentType: 'application/json',
             data: JSON.stringify({"domain": domText, "problem": probText})})
        .done(function (res) {

                if (res['status'] === 'ok') {
                    window.toastr.success('Plan found!');
                } else {
                    window.toastr.error('Planning failed.');
                }

                showPlan(res);

            }).fail(function (res) {
                window.toastr.error('Error: Malformed URL?');
            });
}
```

Listing 1: Javascript code on planning.domains responsible for triggering the solver

The remote solver simply takes two parameters, the content of the domain file and the content of the problem file, wrap them in a JSON message and send it to the endpoint *http://solver.planning.domains/solve*. I then wrote a simple python script which would send the request and print it to standard output, e.g.

```
python run.py dungeon-domain.pddl problem1.pddl | python -m json.tool -
```

```python
import json
import requests

SOLVER_URL = "http://solver.planning.domains/solve"

def read_file(from_path):
    return "".join(open(from_path).readlines())

if __name__ == "__main__":
    import sys

    domain_file_path = sys.argv[1]
    problem_file_path = sys.argv[2]

    payload = {}
    payload["domain"] = read_file(domain_file_path)
    payload["problem"] = read_file(problem_file_path)

    r = requests.post(SOLVER_URL, data=json.dumps(payload), headers={"
    Content-Type":"application/json"})

    print(json.dumps(r.json(), indent=4, sort_keys=True))
```

Listing 2: Request script

1.2 Planners

For the reasons I explained in the first section I used the *planning.domains* planner during development. I couldn't find any documentation on what planning engine is actually using internally though, but I compared the output of running the three problems using the *planning.domains* engine against both *siw+* and *dfs+* and all the outputs were equivalent so it's possible it makes use of one of these planners.

After trying to compile several engines locally, I settled on using the docker image provided by **Lapkt** Ramirez, Lipovetzky, and Muise (2015). The docker image they provide made it very easy to run a multitude of planners, including:

- siw
- ff
- siw_plus
- siw_then_bsfs

- bfs_f
- dfs_plus

I wrote a simple bash script which runs all planners sequentially and collected their output for later analysis.

```bash
#!/bin/bash

planners=(dfs_plus bfs_f siw siw_plus siw-then-bfsf ff)

source_dir=/dungeon/src

for planner in "${planners[@]}"
do
    echo -e "Using planner $planner"

    mkdir -p output/$planner

    for i in 1 2 3
    do
        echo -e "./$planner --domain $source_dir/dungeon-domain.pddl --problem $source_dir/problem$i.pddl"
        ./$planner --domain $source_dir/dungeon-domain.pddl --problem $source_dir/problem$i.pddl --output output/$planner/$i.ipc
        mv execution.details output/$planner/execution_details_$i
    done
done

echo "All done."
```

Listing 3: Script to run all planners

1.2.1 Performance on given problem set

The analysis between the planners output was done manually using the *diff* Unix command and with the aid of a multi-file and recursive vimdiff plugin Ershov (2016), simply triggered from a shell:

```
diff -r -u directory_1 directory_2 | vim -R -
```

All planners managed to find a solution to all problems with the exception of *siw* which couldn't find a plan neither for problem 2 nor for problem 3.

I compared multiple outputs and observed subtle differences between planners. In terms of time, all planners were blazing fast on the given set of problems and the plans discovered had all the same (optimal) length (6 for problem 1, 12 for problem 2, 13 for problem 3). The only difference observed in a few occasions was the order of the actions taken by the planners.

Because I couldn't find out the solver used by planning.domains, I focused the comparison mostly on two width-based planners, bfsf and dfs+. Unsurprisingly, bfs(f) uses breadth-first-search while dfs+ uses a depth-first search.

Both planners, but in particular dfs+, have a tendency to avoid monsters. In the first problem, both planners hit only one monster. In the second problem, bfsf hits all monsters but dfs preferred instead to go through the traps and encountered only one monster out of three. We can see from the diff how it picks up the sword immediately after the initial room but then instead of facing the monster in r3 it prefers to go to r4 and destroy it right away (because a trap can't be activated with a sword). In the last problem, out of seven monsters, both bfsf and dfs+ faced only one monster.

It seems all planners have figured out that facing monsters require going through a room with a sword first, which might make the plan longer, hence they tried to avoid it.

```
diff -r -u bfs_f/1.ipc dfs_plus/1.ipc
--- bfs_f/1.ipc 2020-04-19 22:10:42.000000000
+++ dfs_plus/1.ipc  2020-04-19 22:10:42.000000000
@@ -1,6 +1,6 @@
 (MOVE PLAYER R0 R1)
 (PICKUP-SWORD R1 PLAYER S0)
-(MOVE PLAYER R1 R3)
-(FACE-MONSTER PLAYER S0 R3 M1)
-(MOVE PLAYER R3 R4)
+(MOVE PLAYER R1 R2)
+(FACE-MONSTER PLAYER S0 R2 M0)
+(MOVE PLAYER R2 R4)
 (MOVE PLAYER R4 R7)
diff -r -u bfs_f/2.ipc dfs_plus/2.ipc
--- bfs_f/2.ipc 2020-04-19 22:10:42.000000000
+++ dfs_plus/2.ipc  2020-04-19 22:10:42.000000000
@@ -1,12 +1,12 @@
 (MOVE PLAYER R0 R1)
 (PICKUP-SWORD R1 PLAYER S0)
-(MOVE PLAYER R1 R3)
-(FACE-MONSTER PLAYER S0 R3 M0)
-(MOVE PLAYER R3 R5)
-(FACE-MONSTER PLAYER S0 R5 M1)
-(MOVE PLAYER R5 R8)
-(FACE-MONSTER PLAYER S0 R8 M2)
-(MOVE PLAYER R8 R9)
+(MOVE PLAYER R1 R4)
 (DESTROY-SWORD PLAYER S0)
-(DISARM-TRAP PLAYER R9 T0)
-(MOVE PLAYER R9 R11)
+(DISARM-TRAP PLAYER R4 T0)
+(MOVE PLAYER R4 R5)
+(FACE-MONSTER PLAYER S0 R5 M1)
+(MOVE PLAYER R5 R7)
+(DISARM-TRAP PLAYER R7 T0)
+(MOVE PLAYER R7 R10)
+(DISARM-TRAP PLAYER R10 T0)
+(MOVE PLAYER R10 R11)
diff -r -u bfs_f/3.ipc dfs_plus/3.ipc
--- bfs_f/3.ipc 2020-04-19 22:10:42.000000000
+++ dfs_plus/3.ipc  2020-04-19 22:10:42.000000000
@@ -5,9 +5,9 @@
```

```
 (DISARM-TRAP PLAYER R7 T1)
 (MOVE PLAYER R7 R11)
 (PICKUP-SWORD R11 PLAYER S0)
+(DESTROY-SWORD PLAYER S0)
 (MOVE PLAYER R11 R14)
 (FACE-MONSTER PLAYER S0 R14 M5)
 (MOVE PLAYER R14 R17)
-(DESTROY-SWORD PLAYER S0)
 (DISARM-TRAP PLAYER R17 T4)
 (MOVE PLAYER R17 R19)
```

Listing 4: "Difference in problem3 output plan between bfsf and dfs+"

1.2.2 IPCs results

In the diff output shown above I compared the two best performant planners (dfs+ in terms of time and bfs_f in terms of plan length) according to Lipovetzky and Geffner (2014) on a range of IPC problems. The results from the paper are reported in figure 1.

Table 2. Time(T), and plan length (Q) of first solution as IPC scores. Bold shows best performer.

		GBFS	SIW	SIW$^+$	FF	DFS$^+$	BFS(f)	PROBE	LAMA'11
Barman	T	0	0	3.59	0	4.52	15.51	**16.56**	10.18
	Q	0	0	15.3	0	17.95	**19.19**	17.39	15.19
Elevators	T	0.01	1.23	11.33	11.78	15.62	2.41	6.16	**16.33**
	Q	1.14	15.24	10.83	**19.44**	10.93	12	14.62	18.24
Floortile	T	0.06	0	0	0.59	0	**3.26**	2.2	2.8
	Q	2.96	0	0	1.83	0	**5.75**	4.61	4.67
NoMystery	T	1.59	0	0	5	0.08	**12.04**	2.52	9.48
	Q	5.74	0	0	5	3.77	**14.57**	5.72	9.72
Openstacks	T	0.64	1.73	15.72	11.62	**19.88**	1.99	1.64	13.94
	Q	13.4	5	**19.83**	18.81	**19.83**	15.38	13.79	19.24
Parcprinter	T	16.06	**20**	**20**	**20**	**20**	7.85	11.74	19
	Q	18.9	19.01	19.01	**19.79**	19.01	16.01	12.88	18.31
Parking	T	0.07	**19.19**	18.82	2.57	18.5	2.35	2.15	5.06
	Q	2.53	**19.53**	19.36	10.94	19.42	10.72	6.27	10.68
Pegsol	T	19.01	0	1	**20**	7.3	19.5	16.86	19.23
	Q	17.89	0	1	17.93	17.1	**18.99**	18	17.92
Scanalyzer	T	5.15	**15.24**	14.32	13.27	14.97	9.91	13.75	10.71
	Q	14.81	13.48	13.4	15.82	14.88	14.04	**16.56**	14.86
Sokoban	T	2.2	0	0	9.47	0.3	6.68	**11.22**	10.08
	Q	9.66	0	0	**13.83**	0.93	10.58	10.29	13.23
Tidybot	T	1.66	3.04	6.88	**14.41**	8.44	2.32	8.02	2.81
	Q	13.04	4.86	14.77	14.43	15.81	15.31	**16.46**	12.54
Transport	T	0.09	3.11	13.67	0.18	**15.17**	4.91	5.84	9.23
	Q	2.66	11.79	12.27	6.04	13.38	14	11.14	**15.82**
VisitAll	T	0.14	17.37	12.81	1.88	12.96	**20**	5.68	6.46
	Q	0.25	19.5	19.5	1.58	19.5	**20**	14.39	14.69
Woodworking	T	1.01	7.92	6.05	**19**	7.18	2.16	2.74	3.44
	Q	2	17.47	17.88	16.97	17.81	19.86	**20**	15.51
All	T	47.69	88.84	124.2	129.78	**144.92**	110.89	107.07	139.96
	Q	104.98	125.88	163.15	162.41	190.91	**206.49**	182.12	200.62

Figure 1: IPC scores

2 Operators

The actions in the dungeon domain file have been lifted almost verbatim from the assignment specifications. E.g.

ASSIGNMENT: The hero can **move** to an adjacent room...
ACTION: *move*

ASSIGNMENT: **Pickup** the sword if present...
ACTION: *pickup-sword*

ASSIGNMENT: **Destroy** the sword...
ACTION: *destroy-sword*

and so on.

2.1 Actions reference

2.1.1 move

```
(:action move
    :parameters (?h - hero ?from - room ?to - room)
    :precondition (and
        (not (room-destroyed ?to))
        (hero-in-room ?h ?from)
        (hero-alive ?h)
        (room-corridor ?from ?to)
        (room-cleared ?h ?from)
    )
    :effect (and
        (hero-alive ?h)
        (hero-in-room ?h ?to)
        (not (hero-in-room ?h ?from))
        (room-destroyed ?from)
    )
)
```

Listing 5: Move action

This action is used to move the hero from one room to another, which can happen only if the hero is in the room, alive, the room has not been visited before, it's been cleared of monsters or traps and there is a corridor connecting the two rooms.

2.1.2 pickup-sword

```
(:action pickup-sword
    :parameters (?r - room ?h - hero ?s - sword)
    :precondition (and
        (hero-alive ?h)
        (hero-in-room ?h ?r)
        (hero-empty-handed ?h)
        (sword-in-room ?s ?r)
        (not ( sword-holding ?h ?s))
    )
    :effect (and
        (hero-alive ?h)
        (hero-in-room ?h ?r)
```

```
        (not (hero-empty-handed ?h))
        (not ( sword-in-room ?s ?r ))
        (not ( sword-destroyed ?s ))
        (sword-holding ?h ?s)
        (room-cleared ?h ?r)
    )
)
```

Listing 6: pickup-sword action

The hero can use this action to pickup a sword, which is necessary to scary monsters away and not getting killed.

2.1.3 destroy-sword

```
(:action destroy-sword
    :parameters (?h - hero ?s - sword)
    :precondition (and
        (hero-alive ?h)
        (sword-holding ?h ?s)
    )
    :effect (and
        (hero-alive ?h)
        (sword-destroyed ?s)
        (hero-empty-handed ?h )
    )
)
```

Listing 7: destroy-sword action

This action destroy the sword, which is necessary since it's only possible to disarm a trap if empty-handed. However, the hero should not take this action in a room with a monster or the hero will be killed.

2.1.4 disarm-trap

```
(:action disarm-trap
    :parameters (?h - hero ?r - room ?t - trap)
    :precondition (and
        (hero-alive ?h)
        (hero-in-room ?h ?r)
        (hero-empty-handed ?h)
        (trap-in-room ?t ?r)
        (trap-armed ?t ?r)
        (not (trap-disarmed ?t ?r))
    )
    :effect (and
        (hero-alive ?h)
        (hero-in-room ?h ?r)
        (hero-empty-handed ?h)
        (trap-in-room ?t ?r)
        (not (trap-armed ?t ?r))
        (trap-disarmed ?t ?r)
        (room-cleared ?h ?r)
    )
```

```
)
```

Listing 8: disarm-trap action

disarm-trap is used by the hero to disarm a trap, which can only be done if the hero doesn't hold a sword (among other pre-conditions).

2.1.5 face-monster

```
(:action face-monster
    :parameters (?h - hero ?s - sword ?r - room ?m - monster)
    :precondition (and
        (hero-alive ?h)
        (hero-in-room ?h ?r)
        (sword-holding ?h ?s)
        (monster-in-room ?m ?r)
        (not (monster-afraid ?h ?m))
    )
    :effect (and
        (hero-alive ?h)
        (hero-in-room ?h ?r)
        (sword-holding ?h ?s)
        (monster-in-room ?m ?r)
        (monster-afraid ?h ?m)
        (room-cleared ?h ?r)
    )
)
```

Listing 9: face-monster action

The last action a hero can execute is to face a monster, for which the hero is required to hold a sword (for defense purposes only).

3 Conclusions

In this report I've described how I've developed the dungeon domain, encoded the three problems described in the assignment specification and the tool I used. I also provided a reference and an explanation of the operators in the domain file. In section 2 I've shown the differences between the two planners output I compared on the given problem set and referenced a more comprehensive evaluation across multiple IPC problems. This evaluation shows how DFS+ was the best performer in terms of *Time* to generate the plans while BFS(f) was the best performer from a *Plan Length* perspective.

References

Ershov, Y. (2016, 01). *vimdiffext.* `https://www.vim.org/scripts/script.php?script_id=5309`. ((Accessed on 04/20/2020))

Lipovetzky, N., & Geffner, H. (2014). Width-based algorithms for classical planning: New results. In *Ecai.*

Muise, C. (2015). *planning.domains.* `http://planning.domains/`. ((Accessed on 04/20/2020))

Persson, P. (2016). *Pddl syntax highlighting for vim.* `https://github.com/PontusPersson/pddl.vim`. ((Accessed on 04/20/2020))

Ramirez, M., Lipovetzky, N., & Muise, C. (2015). *Lightweight Automated Planning ToolKiT.* `http://lapkt.org/`. (Accessed: 04/20/2020)