This report was written by:

- Mateusz Przewlocki, 1609577
- Valentine Vialle, 1575227

Planning Analysis

We created three problems for our domain, ranging from simple to complex. Unfortunately, the planner did not work as expected for refuelling the trucks, so we had to give the trucks reasonably large quantities of fuel to make the planner work properly.

Results:

Problem	**Objects**	**States evaluated**	**Cost**
Problem 1	9	354	1330.024
Problem 2	15	504	1800.036
Problem 3	27	899	3925.129

Figure 1 – Table showing results from 3 problems made for the domain

From these results, we can clearly see that the planner takes more time to solve more complex problems, as expected. This is because if there are more places for the planner to evaluate, then there are more states for it to evaluate. The state space complexity increases as more objects are added to the problem. We predict that the number of states will increase exponentially as more objects are added, like in the graph below.

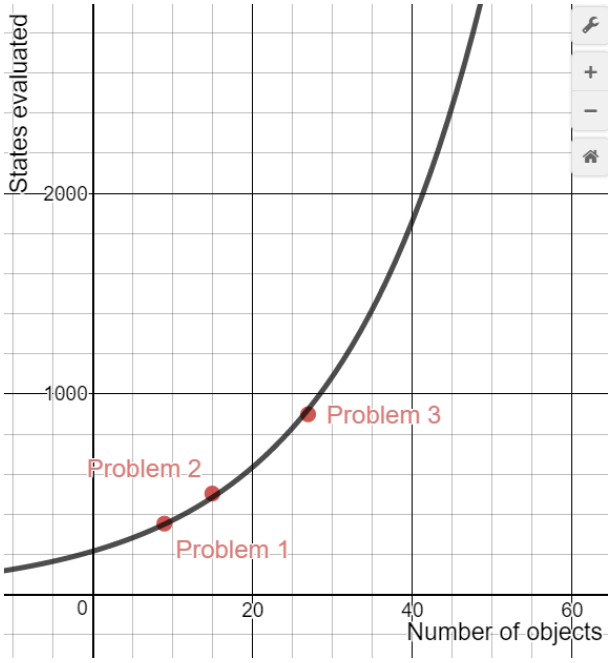

Figure 2 – Graph showing rate of change of state space (created using Desmos)

The difference in the costs can be attributed to the varying number of wells with water in each of the problems. It can also be attributed to the fuel difference in problem 3, as it needed more fuel to complete the problem without having to refuel. (Refuelling made the planner "stuck" on trying to figure out a solution for too long.)

However, we found that the planner often gave suboptimal solutions. Sometimes the planner's final action was simply to take the water from a well and not sell it to a city, resulting in less cash gained, so the metric was not, in practice, maximised. This is because the goal in our problems was simply to

have all wells be emptied and all cities have water. Once a city has had water sold to it from one well, it would not need water from another.

Flaws and Problems of Domain and Planning

We had to simplify the domain and problems considerably, and remove factors which could be considered important if a situation like the one we modelled were to become real.

For example, we left out wages for the truck drivers, as it would be difficult for the planner to determine when the wages should be paid. If they were to be paid at the end of the job, then it would also not be necessary to include this factor in the planning as all the drivers' wages would just to be subtracted from the amount of cash the company has at the end of the job.

We also did not consider the fact that cities consume the water brought to them, so in a real-life situation, it'd be endless and we'd have to find new wells constantly. The planner, at least to our knowledge of PDDL, would not be able to locate new wells all the time, nor would it be able to keep calculating solutions for the problem indefinitely. Therefore, we decided on a simple goal, providing at least some water to the cities.

Certain actions, like refuelling or drawing some water from the well (as opposed to all of it) made the planner show overly complex behaviour, even for very simple problems, and made the planner get "stuck" in incomplete states. Therefore, when writing the problem files, we have had to make the trucks have amounts of fuel dependent on how big the state space was and how high the distances were. As for the water drawing actions, they had the same problem as the refuelling, so we had to make the amounts of water in each well be no greater than the capacities of the trucks.

```
Resorting to best-first search
Running WA* with W = 5.000, not restarting with goal states
b (29.000 | 40.000)b (28.000 | 150.002)b (27.000 | 160.003)b (26.000 | 160.003)
b (25.000 | 170.004)b (24.000 | 260.004)b (23.000 | 270.004)b (22.000 | 380.007
)b (21.000 | 390.007)b (20.000 | 420.008)b (19.000 | 430.008)b (18.000 | 520.00
9)b (17.000 | 530.009)b (16.000 | 640.012)b (15.000 | 650.012)b (14.000 | 740.0
13)b (13.000 | 750.013)b (12.000 | 790.015)b (11.000 | 890.015)b (11.000 | 800.
015)b (10.000 | 900.017)b (9.000 | 910.017)b (8.000 | 1000.018)b (7.000 | 1010.
018)b (6.000 | 1180.021)b (5.000 | 1220.021)b (5.000 | 1110.021)
```

Figure 3 – The state the planner was stuck in in the situation described above

```
.022)b (2.000 | 1230.023)b (1.000 | 1330.023)(G);;;; Solution Found
; States evaluated: 354
; Cost: 1330.024
; Time 0.98
0.000: (go-to truck1 city1 city2)  [40.000]
40.001: (go-to truck1 city2 well6)  [10.000]
50.002: (take-all-water truck1 well6)  [100.000]
150.003: (go-to truck1 well6 city1)  [10.000]
160.004: (sell-all-water truck1 city1)  [100.000]
260.005: (go-to truck1 city1 well3)  [10.000]
270.006: (take-all-water truck1 well3)  [100.000]
370.007: (go-to truck1 well3 city1)  [10.000]
380.008: (go-to truck1 city1 city2)  [40.000]
420.009: (sell-all-water truck1 city2)  [100.000]
520.010: (go-to truck1 city2 well5)  [10.000]
530.011: (take-all-water truck1 well5)  [100.000]
630.012: (go-to truck1 well5 city1)  [10.000]
640.013: (sell-all-water truck1 city1)  [100.000]
740.014: (go-to truck1 city1 city2)  [40.000]
780.015: (go-to truck1 city2 well4)  [10.000]
790.016: (take-all-water truck1 well4)  [100.000]
890.017: (go-to truck1 well4 city1)  [10.000]
900.018: (sell-all-water truck1 city1)  [100.000]
1000.019: (go-to truck1 city1 well2)  [10.000]
1010.020: (take-all-water truck1 well2)  [100.000]
1110.021: (go-to truck1 well2 city1)  [10.000]
1120.022: (sell-all-water truck1 city1)  [100.000]
1220.023: (go-to truck1 city1 well1)  [10.000]
1230.024: (take-all-water truck1 well1)  [100.000]
```

Figure 4 – The planner being successful when the amount of fuel is increased

Conclusion

Our planning helped us to understand how to apply PDDL to real-life problems. We discovered the strong points of such type of planning. Planning a problem like this could be useful if a company were to maximise its profits while only having limited resources (limited fuel, limited money to buy the fuel

with etc.). Therefore, AI planning is not just something which is full of toy examples and useless in the real world. Writing PDDL is also easy, and there are many AI planners at our disposal. Many factors can be considered in AI planning, like temporal planning, resources such as fuel and cash, minimising and maximising certain conditions etc. We also discovered the weak points of AI planning. For example, we had to simplify the problems and domain considerably for the OPTIC planner to be able to solve the problems without running into problems.

Overall, however, we have seen that AI planning is very useful in creating solutions for problems. Where it would take us a very long time to figure out solutions for problems with medium-sized state spaces, AI planners can do this much faster, and even for larger problems.

Appendix 1 – PDDL Problem Definitions

Problem 1 (Simplest)

```
(define (problem water-transport-1)
      (:domain water-transport)
      (:objects
            truck1 - truck
            city1 city2 - city
            station1 - station
            well1 well2 well3 well4 well5 well6 - well
      )
      (:init
            (= (fuel-cost) 5)
            (= (fuel-per-distance) 0.5)
            (= (water-cost) 2)
            (= (current-time) 0)
            (= (cash) 100)
            (= (water-decay-rate) 0.01)

            (at truck1 city1)
            (free truck1)
            (= (water-capacity truck1) 100)
            (= (water truck1) 0)
            (= (fuel truck1) 200)
            (= (fuel-capacity truck1) 200)
            (= (speed truck1) 1)

            ; distances
            (= (distance city1 city2) 40)

            (= (distance city1 well1) 10)
            (= (distance city1 well2) 10)
            (= (distance city1 well3) 10)

            (= (distance city2 well4) 10)
            (= (distance city2 well5) 10)
            (= (distance city2 well6) 10)

            (= (distance well3 station1) 15)
            (= (distance well6 station1) 15)

            ; inverse distances
            (= (distance city2 city1) 40)

            (= (distance well1 city1) 10)
            (= (distance well2 city1) 10)
```

```
            (= (distance well3 city1) 10)

            (= (distance well4 city1) 10)
            (= (distance well5 city1) 10)
            (= (distance well6 city1) 10)

            (= (distance station1 well3) 15)
            (= (distance station1 well6) 15)

            (= (water-level well1) 100)
            (= (water-level well2) 100)
            (= (water-level well3) 100)
            (= (water-level well4) 100)
            (= (water-level well5) 100)
            (= (water-level well6) 100)

            (has-water well1)
            (has-water well2)
            (has-water well3)
            (has-water well4)
            (has-water well5)
            (has-water well6)

      )
      (:goal
            (and
                  (has-no-water well1)
                  (has-no-water well2)
                  (has-no-water well3)
                  (has-no-water well4)
                  (has-no-water well5)
                  (has-no-water well6)

                  (has-water city1)
                  (has-water city2)
            )
      )

      (:metric maximize (cash))
)
```

Problem 2

```
(define (problem water-transport-2)
      (:domain water-transport)
      (:objects
            truck1 - truck
            city1 city2 city3 city4 - city
            station1 station2 - station
            well1 well2 well3 well4 well5 well6 well7 well8 - well
      )
      (:init
            (= (fuel-cost) 5)
            (= (fuel-per-distance) 0.5)
            (= (water-cost) 2)
            (= (current-time) 0)
            (= (cash) 100)
            (= (water-decay-rate) 0.01)
```

```
(at truck1 city1)
(free truck1)
(= (water-capacity truck1) 100)
(= (water truck1) 0)
(= (fuel truck1) 200)
(= (fuel-capacity truck1) 200)
(= (speed truck1) 1)

; distances
(= (distance city1 city2) 20)
(= (distance city1 city3) 20)
(= (distance city1 city4) 20)

(= (distance city1 station1) 5)
(= (distance city1 well1) 10)
(= (distance well1 well2) 10)

(= (distance city2 well3) 15)
(= (distance city2 well4) 15)

(= (distance city3 station2) 5)
(= (distance city3 well5) 10)

(= (distance city4 well6) 5)
(= (distance city4 well7) 10)
(= (distance city4 well8) 15)

; inverse distances
(= (distance city2 city1) 20)
(= (distance city3 city1) 20)
(= (distance city4 city1) 20)

(= (distance station1 city1) 5)
(= (distance well1 city1) 10)
(= (distance well2 well1) 10)

(= (distance well3 city2) 15)
(= (distance well4 city2) 15)

(= (distance station2 city3) 5)
(= (distance well5 city3) 10)

(= (distance well6 city4) 5)
(= (distance well7 city4) 10)
(= (distance well8 city4) 15)

(= (water-level well1) 100)
(= (water-level well2) 100)
(= (water-level well3) 100)
(= (water-level well4) 100)
(= (water-level well5) 100)
(= (water-level well6) 100)
(= (water-level well7) 100)
(= (water-level well8) 100)
```

```
            (has-water well1)
            (has-water well2)
            (has-water well3)
            (has-water well4)
            (has-water well5)
            (has-water well6)
            (has-water well7)
            (has-water well8)

      )
      (:goal
            (and
                  (has-no-water well1)
                  (has-no-water well2)
                  (has-no-water well3)
                  (has-no-water well4)
                  (has-no-water well5)
                  (has-no-water well6)
                  (has-no-water well7)
                  (has-no-water well8)

                  (has-water city1)
                  (has-water city2)
                  (has-water city3)
                  (has-water city4)
            )
      )

      (:metric maximize (cash))
)
```

Problem 3 (Most complex)

```
(define (problem water-transport-3)
      (:domain water-transport)
      (:objects
            truck1 - truck
            city1 city2 city3 city4 city5 city6 - city
            station1 station2 station3 station4 - station
            well1 well2 well3 well4 well5 well6 well7 well8 well9 well10 well11
well12 well13 well14 well15 well16 - well
      )
      (:init
            (= (fuel-cost) 5)
            (= (fuel-per-distance) 0.5)
            (= (water-cost) 2)
            (= (current-time) 0)
            (= (cash) 100)
            (= (water-decay-rate) 0.01)

            (at truck1 city1)
            (free truck1)
            (= (water-capacity truck1) 100)
            (= (water truck1) 0)
            (= (fuel truck1) 500)
```

```
(= (fuel-capacity truck1) 500)
(= (speed truck1) 1)

; distances
(= (distance city1 city2) 30)
(= (distance city2 city3) 30)
(= (distance city3 city4) 30)
(= (distance city4 city5) 30)

(= (distance city1 well1) 5)
(= (distance well1 station1) 5)
(= (distance station1 well2) 5)
(= (distance station1 well3) 5)

(= (distance city2 well4) 5)
(= (distance city2 well5) 5)
(= (distance well5 station2) 10)
(= (distance station2 well6) 10)
(= (distance station2 well7) 10)

(= (distance city3 well8) 5)
(= (distance city3 well9) 5)
(= (distance well9 station3) 10)
(= (distance station3 city6) 20)
(= (distance city6 well10) 5)

(= (distance city4 well11) 5)
(= (distance city4 well12) 5)

(= (distance city5 station4) 10)
(= (distance station4 well13) 5)
(= (distance station4 well14) 5)
(= (distance station4 well15) 5)
(= (distance city5 well16) 5)

; inverse distances
(= (distance city2 city1) 30)
(= (distance city3 city2) 30)
(= (distance city4 city3) 30)
(= (distance city5 city4) 30)

(= (distance well1 city1) 5)
(= (distance station1 well1) 5)
(= (distance well2 station1) 5)
(= (distance well3 station1) 5)

(= (distance well4 city2) 5)
(= (distance well5 city2) 5)
(= (distance station2 well5) 10)
(= (distance well6 station2) 10)
(= (distance well7 station2) 10)

(= (distance well8 city3) 5)
(= (distance well9 city3) 5)
(= (distance station3 well9) 10)
```

```
      (= (distance city6 station3) 20)
      (= (distance well10 city6) 5)

      (= (distance well11 city4) 5)
      (= (distance well12 city4) 5)

      (= (distance station4 city5) 10)
      (= (distance well13 station4) 5)
      (= (distance well14 station4) 5)
      (= (distance well15 station4) 5)
      (= (distance well16 city5) 5)

      (= (water-level well1) 100)
      (= (water-level well2) 100)
      (= (water-level well3) 100)
      (= (water-level well4) 100)
      (= (water-level well5) 100)
      (= (water-level well6) 100)
      (= (water-level well7) 100)
      (= (water-level well8) 100)
      (= (water-level well9) 100)
      (= (water-level well10) 100)
      (= (water-level well11) 100)
      (= (water-level well12) 100)
      (= (water-level well13) 100)
      (= (water-level well14) 100)
      (= (water-level well15) 100)
      (= (water-level well16) 100)

      (has-water well1)
      (has-water well2)
      (has-water well3)
      (has-water well4)
      (has-water well5)
      (has-water well6)
      (has-water well7)
      (has-water well8)
      (has-water well9)
      (has-water well10)
      (has-water well11)
      (has-water well12)
      (has-water well13)
      (has-water well14)
      (has-water well15)
      (has-water well16)

)
(:goal
      (and
            (has-no-water well1)
            (has-no-water well2)
            (has-no-water well3)
            (has-no-water well4)
            (has-no-water well5)
            (has-no-water well6)
```

```
                (has-no-water well7)
                (has-no-water well8)
                (has-no-water well9)
                (has-no-water well10)
                (has-no-water well11)
                (has-no-water well12)
                (has-no-water well13)
                (has-no-water well14)
                (has-no-water well15)
                (has-no-water well16)

                (has-water city1)
                (has-water city2)
                (has-water city3)
                (has-water city4)
                (has-water city5)
                (has-water city6)
          )
     )

     (:metric maximize (cash))
)
```