This report was written by:

- Mateusz Przewlocki, 1609577
- Valentine Vialle, 1575227

Introduction

The domain we have chosen to model involves the transportation of a substance, such as water, that is likely to evaporate under desert conditions. Transporting this substance would generate a profit, which is to be maximised by the planner in this domain.

In this report, we have used the OPTIC planner.

Domain Analysis

Our domain has five types. The **truck** and **location** types are fundamental objects in the domain, while the **city**, **well** and **station** types are locations in the domain. The truck type represents a truck that can transport water, while the location types represent possible locations that can be reached in the domain.

There are four predicates. The **at** predicate describes whether a truck is at a given location or not. The **free** predicate indicates whether a truck is free to perform an action, to prevent it from trying to perform two actions at once. The **has-water** and **has-no-water** predicates are used in determining whether a truck can access water from a well, and the end goal of the planner, which is to have all the wells be empty. As the metric used in the problems is maximising the cash, which is a function, this makes the problem more interesting by forcing the planner to revisit cities to sell water to them and therefore maximise the cash.

There are many functions in the domain. **Fuel-cost**, **water-cost**, **fuel-per-distance**, **cash**, **current-time** and **water-decay-rate** are all constant numbers. Fuel-cost and water-cost are self-explanatory. Fuel-per-distance is a function to describe how much fuel a truck uses to move a certain distance. Cash is a function representing all the cash that a company has, which changes whenever a truck buys fuel (the company reimburses the truck drivers for the cost of the fuel) and when water is sold to a city. The current-time and water-decay-rate functions would have been used to make the water evaporate according to the time since the program started, although we could not think of a way to implement this in PDDL.

Four actions are present, and they are all durative actions. There is a **go-to** action, which transports a truck from one location to another. The **refuel** action refuels a truck according to the time spent at the station, if the truck is at a station action. The **take-all-water** action takes all water from a well, and the **sell-all-water** action sells all the water that a truck contains to a city.

We tried to implement actions which only take some of the water from a well, however this was not possible to implement in the OPTIC planner. We also tried to implement actions to make the truck only take some water from the wells, but again this was not possible and would make it difficult to determine whether the well has been emptied once the truck has done so.

Appendix 1 – PDDL Domain

```
(define (domain water-transport)
      (:requirements :typing :durative-actions :numeric-fluents :fluents
:equality :duration-inequalities :conditional-effects)
      (:types
            truck location - object
            city well station - location
```

```
	)
	(:predicates
		(at ?x - truck ?y - location)
		(has-water ?x - location)
		(has-no-water ?x - location)
		(free ?x - truck)
	)
	(:functions
		(fuel-cost)
		(water-cost)
		(fuel-per-distance)
		(cash)
		(current-time)
		(water-decay-rate)
		(speed ?x - truck)
		(distance ?x ?y - location)
		(fuel ?x - truck)
		(fuel-capacity ?x - truck)
		(water ?x - truck)
		(water-capacity ?x - truck)
		(water-level ?x - location)
	)
	(:durative-action go-to
		:parameters (?x - truck ?y - location ?z - location)
		:duration (= ?duration (/ (distance ?y ?z) (speed ?x)))
		:condition (and
			(at start (at ?x ?y))
			(at start (free ?x))
			(over all (>= (fuel ?x) 0)))
		:effect (and
			(at start (not (free ?x)))
			(at end (at ?x ?z))
			(at end (not (at ?x ?y)))
			(at end (free ?x))
			(at end (decrease (fuel ?x) (* fuel-per-distance (distance ?y
?z)))))
	)
	(:durative-action refuel
		:parameters (?x - truck ?y - station)
		:duration (>= ?duration 0)
		:condition (and
			(at start (free ?x))
			(over all (at ?x ?y))
			(over all (> (cash ?x) 0))
			(over all (< (fuel ?x) (fuel-capacity ?x))))
		:effect(and
			(at start (not (free ?x)))
			(at end (free ?x))
			(increase (fuel ?x) (* #t 1))
			(decrease (cash) (* #t fuel-cost)))
	)
	(:durative-action take-all-water
		:parameters (?x - truck ?y - well)
		:duration (= ?duration (water-level ?y))
		:condition (and
			(at start (at ?x ?y))
			(at start (free ?x))
			(at start (has-water ?y))
			(over all (< (water ?x) (water-capacity ?x))))
		:effect(and
			(at start (not (free ?x)))
			(at end (free ?x))
			(at end (increase (water ?x) (water-level ?y)))
			(at end (assign (water-level ?y) 0))
			(at end (has-no-water ?y))
			(at end (not (has-water ?y))))
	)
	(:durative-action sell-all-water
		:parameters (?x - truck ?y - city)
		:duration (= ?duration (water ?x))
		:condition (and
			(at start (at ?x ?y))
			(at start (free ?x)))
		:effect(and
			(at start (not (free ?x)))
			(at end (free ?x))
			(at end (increase (cash) (* (water-cost) (water ?x))))
			(at end (assign (water ?x) 0))
			(at end (has-water ?y)))
	)
)
```