20 | 15

Introduction to Artificial Intelligence Coursework 2015

Tahmidul Islam

Group Members:

- Tahmidul Islam (K1462015)
- Hassan Minhas (K1461907)
- Muhammad Rohman (K1461540)

We declare that we understand the nature of plagiarism as defined in the Undergraduate Handbook and that the content of this coursework submission is entirely our own work.

Pizza Delivery using PDDL

4CCS1IAI – Introduction to Artificial Intelligence – Coursework 2015

Introduction

Currently the 2nd most popular takeout food is pizza, and over the past 5 years it has surpassed the growth rate of all other food services. An example of a successful pizza parlour would be Pizza Hut, with over 12,583 total restaurants within the US and 90 other countries. A pivotal key to its success lies in their ability to make pizza, at great quality but also being able to deliver the pizza quickly.

It is this that gave us our inspiration to model the delivery aspect of a pizza company. Quality of delivery service is an important factor in retaining customers. Fast, consistent delivery can result in more regular customers. However, slow and inconsistent delivery services can result in customers deeming the business unreliable. Also, since drivers working for a pizza parlour would have orders fairly regularly, it is very important to keep fuel-cost as low as possible in order to maintain a higher profit margin.

Domain and PDDL

The purpose of our domain and problem files is to allow the user to plan a route for delivering a set number of pizzas to locations in the most efficient way possible, whilst taking into account time, fuel and the fact that there may be more than one possible vehicle transporting the pizza, and therefore our plan allows us to make efficient use of this fact.

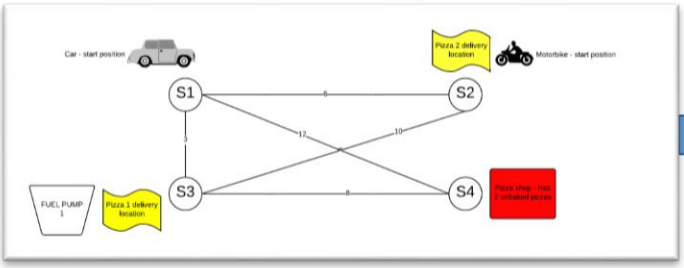
Furthermore since no two orders will be the same, we have taken into account that our domain needs to be domain-independent and therefore able to work with multiple types of problems involving different vehicles, number of streets, number of pizza orders etc. We have therefore run multiple tests using different problem files to check a valid plan is still produced.

During the creation of this domain we came up with a number of assumptions of this system. One assumption is that the time it take it takes between routes does not change (meaning it will not take in to account of any traffic or bad weather conditions).

Also, we always assume that there will be a parking space near the delivery location when in reality this is not always the case.

During the course of the project we felt the best way before modelling the problem in PDDL was to model the problem by hand, generate a solution from this and then use the planner to compare solutions. An example can be seen below, where we model a very small problem. We then modelled this into a problem file and checked the solution. Initially whilst modelling this in PDDL, we noticed the planner wouldn't produce valid plans as it would allow the car to move between streets without the streets being linked and drive without having sufficient fuel etc. Identifying these problems, helped us to fix the domain file accordingly and it was this try and fail method that allowed us to create a working domain file to be used with the problem files. (An image of a problem we modelled has been included in the appendix)

Example:




```
0.000: (drive car s1 s4)  [12.000]
0.000: (drive motorbike s2 s1)  [6.000]
6.001: (drive motorbike s1 s4)  [12.000]
12.001: (bake-pizza car p2 s4)  [4.000]
12.001: (bake-pizza car p1 s4)  [4.000]
16.002: (load-pizza car s4 p2)  [2.000]
16.002: (load-pizza car s4 p1)  [2.000]
16.003: (drive car s4 s3)  [8.000]
24.004: (deliver-pizza car s3 p1)  [3.000]
24.005: (drive car s3 s2)  [10.000]
34.006: (deliver-pizza car s2 p2)  [3.000]
```

Domain File

```
(define (domain PizzaDomain)
(:requirements :strips :typing :fluents :durative-actions)
(:types vehicle pizza street fuel_Pump)
(:predicates
  (at-vehicle ?v - vehicle ?s - street)
  (fuelPump_Location ?p - fuel_Pump ?s - street)
  (pizza-baked ?p - pizza)
  (at-pizza ?p - pizza ?l - street)
  (pizza-loaded ?p - pizza ?v - vehicle)
  (address ?p - pizza ?s - street)
  (pizza-delivered ?p - pizza)
  (street-linked ?x ?y - street))

(:functions (fuel_level ?v - vehicle)
            (fuel-required ?s1 ?s2 - street)
            (fuel_used ?v - vehicle)
            (fuel_wasted ?v - vehicle))
```

```
(:durative-action drive
    :parameters (?v - vehicle ?from ?to - street)
    :duration (= ?duration (fuel-required ?from ?to))
    :condition (and (at start (street-linked ?from ?to))
                    (at start (at-vehicle ?v ?from))
                    (at start (>= (fuel_level ?v)
                    (fuel-required ?from ?to)))
                    )
    :effect (and (at start (not (at-vehicle ?v ?from)))
                 (at end (decrease (fuel_level ?v)
                 (+ (fuel-required ?from ?to)(fuel_wasted ?v))))
                 (at end (increase (fuel_used ?v)
                 (+ (fuel-required ?from ?to)(fuel_wasted ?v))))
                 (at end (at-vehicle ?v ?to))))
```

```
(:durative-action load-pizza
    :parameters (?v - vehicle ?s - street ?p - pizza)
    :duration (= ?duration 2)
    :condition (and (at start (pizza-baked ?p))
                    (at start (at-vehicle ?v ?s))
                    (at start (at-pizza ?p ?s)))
    :effect (and  (at start (not(at-pizza ?p ?s)))
                  (at end (pizza-loaded ?p ?v))))
```

```
(:durative-action deliver-pizza
    :parameters (?v - vehicle ?s - street ?p - pizza)
    :duration (= ?duration 3)
    :condition (and (at start (address ?p ?s))
                    (at start (at-vehicle ?v ?s))
                    (at start (pizza-loaded ?p ?v)))
    :effect (and  (at start (not(pizza-loaded ?p ?v)))
                  (at end (pizza-delivered ?p))))
```

```
(:durative-action swap-vehicle
    :parameters (?v1 ?v2 - vehicle ?s - street ?p - pizza)
    :duration (= ?duration 5)
    :condition (and (at start (at-vehicle ?v1 ?s))
                    (at start (at-vehicle ?v2 ?s))
                    (at start (pizza-loaded ?p ?v1)))
    :effect (and  (at start (not(pizza-loaded ?p ?v1)))
                  (at end (pizza-loaded ?p ?v2))
                  ))
```

```
(:durative-action refuel
    :parameters (?v - vehicle ?f - fuel_Pump ?s - street)
    :duration (= ?duration 10)
    :condition (and (at start (at-vehicle ?v ?s))
                    (at end (fuelPump_location ?f ?s)))
    :effect (and (at end (increase (fuel_level ?v) 10))))
```

```
(:durative-action bake-pizza
    :parameters (?v - vehicle ?p - pizza ?s - street)
    :duration (= ?duration 4)
    :condition (and (at start (at-vehicle ?v ?s))
                    (at start (at-pizza ?p ?s)))
    :effect (at end (pizza-baked ?p))))
```

In this part of the domain file, we defined what predicates we would need, as well as the type of objects we would need (details of objects are displayed in appendix).

The functions fuel_level is used to keep track of a vehicles current fuel level and fuel_required is used to specify how much fuel is required to drive from one street to another.

The fuel_wasted function stores the amount of fuel wasted by a vehicle due to it being in-efficient. An example would be a motorbike vs a car. Since a motorbike is lighter, it ultimately wastes less fuel compared to a car.

Drive Action

This action would allow a vehicle to move from one street to another. One of the parameters of the action is that the vehicle has less than or equal to the fuel required, therefore preventing the planner allowing a vehicle to drive without sufficient fuel. It also has a duration which is proportionate to the fuel required to drive from one street to another, as if two streets require more fuel to travel from and to, it would logically entail that it takes longer to travel from and to. The action also subtracts the fuel required + fuel wasted from the fuel level of the vehicle.

Load Pizza Action

The load pizza action, allows a vehicle to load a pizza and requires that both the vehicle and the pizza are at the same location.

Deliver Pizza

This action allows a vehicle to deliver a pizza to a customer, if and only if the vehicle is in the same street as the delivery address of the pizza.

Swap Vehicle

This was one of our actions that we thought could potentially save a pizza company a substantial amount of money. If two vehicles were out for delivery, but one was running low on fuel, it could transfer the pizza onto another vehicle with sufficient fuel, and ultimately prevent the need to refuel which would cost the pizza company more.

Refuel

This action would allow a vehicle to refuel, if and only if there was a fuel pump at the same location as where the vehicle is. The duration was set as 10 as this is higher than the swap-vehicle action, and therefore the planner would prefer, if possible to swap-vehicle as opposed to refuelling.

Bake-Pizza

This action was included so the planner could take into consideration the time taken for a pizza to be baked, before it could be loaded.

Analytics

The planner we used for our domain was OPTIC. This planner uses A* search and also is a "Temporal planner where plan cost is determined by preference of time". Since time was something that was very important to our plan, we believe that this planner would be appropriate to use.

In order to analyse how the planner would hold up, we tested it with different sets of problems. To do this, we first established a way of deciding how to upscale the problem.

There were many factors to consider when considering the best way to upscale the problem including:

- Number of streets
- Number and position of links between streets
- Number of pizzas needing delivering
- Number and position of fuel pumps
- Fuel wasted by a specific vehicle
- Number of vehicles to be included

As we came to realise there was a substantial number of factors to consider and after much consideration it was decided that our main method of upscaling would be increasing number of streets, connections and pizza. Furthermore, we were aware that increasing the number of links between streets would make the planner find an easier solution, however it was less realistic as not all streets are linked in real life. In order to counteract this we decided to calculate the maximum and minimum number of links we could have between streets and then calculate the average of this.

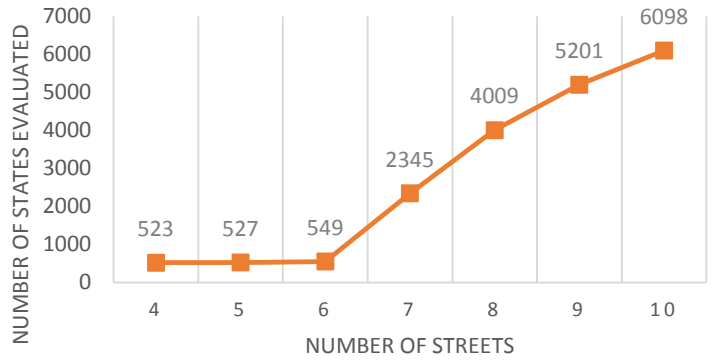
To calculate the maximum number of links we used the following formula, where n stands for the number of streets. For the minimum, it was n – 1.

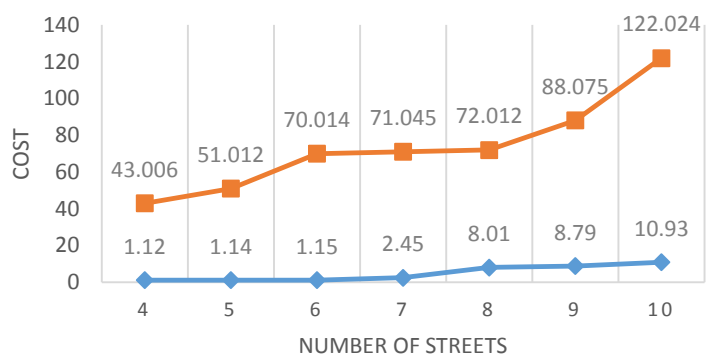
$$\binom{n}{2} = \frac{n!}{(n-2)!2!} = \frac{n(n-1)}{2}$$

For example, if we had 4 streets, the maximum number of links we could have would be 4(3) / 2 = 6, and the minimum would be 3, therefore giving an average of 6 + 3 = 9 / 2 = 4.5 = 5 links.

For the number of pizzas needing delivering we decided to increment it by 1, every time we increased the number of streets. When deciding on the number of cars to use, we felt 2 would be enough as this would allow the planner to make use of the swap-vehicle action we created. We kept the number of cars and fuel pumps constant when carrying out tests as well as the amount of initial fuel of both vehicles.

No. of Streets	No. of Links	No. of Pizzas	No. of States Evaluated	Cost	Time
4	5	2	523	43.006	1.12
5	7	3	527	51.012	1.14
6	10	4	549	70.014	1.15
7	15	5	2345	71.045	2.45
8	18	6	4009	72.012	8.01
9	22	7	5201	88.075	8.79
10	27	8	6098	122.024	10.93





After creating the problem files and running the planner it was clear to see a clear trend between the number of streets (as well as connections, pizza etc.) and also the time taken for the planner to generate a solution.

When running the first problem of using 4 streets, the planner was relatively quick, due to the fact that only 523 states needed to be evaluated. However after increasing the number of streets to 7-8 streets, there was a significant increase in cost, time and more importantly states evaluated. We felt, that despite the large size of the problem, i.e. at 10 streets, there was 27 links, the plan generated by the planner was still a very good, as the cost did not increase significantly.

```
; States evaluated: 549
; Cost: 70.014
; Time 1.15
```

6 Streets, 10 links, 4 Pizzas

```
; States evaluated: 4009
; Cost: 72.012
; Time 8.01
```

8 Streets, 18 links, 6 Pizzas

```
; States evaluated: 6098
; Cost: 122.024
; Time 10.93
```

10 Streets, 27 links, 8 Pizzas

Whilst creating the problem files we noticed that we could incorporate the use of metric, an `(:metric minimize (fuel_used car)))` would like to be decreased. We felt that due to the fact that fuel was an extremely important factor we should specify in the problem file to minimise the fuel_used metric. In addition, we also had to decide which vehicle we would like to minimise the fuel usage of, and decided to go with car as the car would be more expensive to use. After running the planner with and without the metric it was clear to see that the planner would choose the actions that increased fuel usage less frequently.

The graph showing the cost and time increase as the number of streets (and connections, pizzas etc) increased, shows that the planner did scale up pretty well considering the gradient of the cost line wasn't very high and showed a more steady increase.

After confirming that the plans produced were valid plans, we tried to check on how optimum the plan was. From the screenshot of one of the plans produced, we could see that the swap-vehicle action was more frequently used then the refuel action, which indicated that the planner was trying to reduce the fuel_used by the vehicles. Furthermore, we believe the plans produced were optimum, and saw that in many situations when the planner had the choice of using a less expensive actions, it would choose the shorter one which ultimately led to a less expensive plan in terms of fuel costs and also planner costs.

Screenshot plan: 8 streets, 18 links, 6 Pizzas

```
; States evaluated: 4009
; Cost: 72.012
; Time 8.01
0.000: (drive motorbike s2 s4)  [6.000]
0.000: (drive car s1 s4)  [10.000]
6.001: (bake-pizza motorbike p6 s4)  [4.000]
6.001: (bake-pizza motorbike p5 s4)  [4.000]
6.001: (bake-pizza motorbike p4 s4)  [4.000]
6.001: (bake-pizza motorbike p3 s4)  [4.000]
6.001: (bake-pizza motorbike p2 s4)  [4.000]
6.001: (bake-pizza motorbike p1 s4)  [4.000]
10.002: (load-pizza motorbike s4 p6)  [2.000]
10.002: (load-pizza car s4 p5)  [2.000]
10.002: (load-pizza car s4 p4)  [2.000]
10.002: (load-pizza car s4 p3)  [2.000]
10.002: (load-pizza car s4 p2)  [2.000]
10.002: (load-pizza car s4 p1)  [2.000]
10.003: (drive motorbike s4 s2)  [6.000]
10.003: (drive car s4 s5)  [6.000]
16.004: (deliver-pizza motorbike s2 p6)  [3.000]
16.004: (refuel car fp2 s5)  [6.000]
16.004: (refuel car fp2 s5)  [6.000]
16.005: (drive car s5 s7)  [2.000]
16.005: (drive motorbike s2 s3)  [10.000]
18.006: (deliver-pizza car s7 p4)  [3.000]
22.005: (drive car s7 s8)  [8.000]
26.006: (refuel motorbike fp1 s3)  [6.000]
26.006: (refuel motorbike fp1 s3)  [6.000]
30.006: (deliver-pizza car s8 p5)  [3.000]
32.007: (drive motorbike s3 s8)  [10.000]
42.008: (swap-vehicle car motorbike s8 p3)  [5.000]
42.008: (swap-vehicle car motorbike s8 p2)  [5.000]
42.008: (swap-vehicle car motorbike s8 p1)  [5.000]
42.009: (drive motorbike s8 s3)  [10.000]
52.010: (deliver-pizza motorbike s3 p3)  [3.000]
52.010: (deliver-pizza motorbike s3 p2)  [3.000]
52.010: (refuel motorbike fp1 s3)  [6.000]
52.010: (refuel motorbike fp1 s3)  [6.000]
58.011: (drive motorbike s3 s6)  [11.000]
69.012: (deliver-pizza motorbike s6 p1)  [3.000]
```

In conclusion, we are extremely happy with the way this project has gone. Since the topic of this coursework is something we as a group can relate to, it was extremely rewarding to be able to develop something that has real application. This project has also given us a very interesting insight into Ai as a whole, and through the issues that we faced throughout the project we feel we have started to see the bigger picture, in terms of how powerful Ai can be.

Appendix

```
(define (domain PizzaDomain)
(:requirements :strips :typing :fluents :durative-actions)
(:types vehicle pizza street fuel_Pump)
(:predicates
  (at-vehicle ?v - vehicle ?s - street)
  (fuelPump_Location ?p - fuel_Pump ?s - street)
  (pizza-baked ?p - pizza)
  (at-pizza ?p - pizza ?l - street)
  (pizza-loaded ?p - pizza ?v - vehicle)
  (address ?p - pizza ?s - street)
  (pizza-delivered ?p - pizza)
  (street-linked ?x ?y - street))

(:functions (fuel_level ?v - vehicle)
            (fuel-required ?s1 ?s2 - street)
            (fuel_used ?v - vehicle)
            (fuel_wasted ?v - vehicle))
```

Objects

- Vehicle
  - This represents a vehicle which is used to transport a pizza to its delivery location
    - An example would be a motorbike or car (which we used in our problem files)
- Pizza
  - This represented the actual pizza needing delivering
- Street
  - In order to model locations, we decided to use streets and each of these streets would be connected to each other to allow a vehicle to drive from one street to another
- Fuel_Pump
  - This represents a place where a vehicle can stop and re-fuel it's fuel_level

Predicates (Not all have been mentioned)

- At-vehicle
  - This keeps track of where a vehicle is currently at in terms of what street it is on
- FuelPump_Location
  - Keeps track of where a fuel_pump is located (what street it is located at)
- Address
  - This predicate details what the delivery location (street) is for a pizza
- Pizza-delivered
  - Once this predicate is true, the planner will know it has been successfully delivered
- Street-linked
  - Allows user to specify which streets are linked together

Functions

- Fuel_Level
  - This models what the current fuel level of a vehicle is
- Fuel-Required
  - This indicates how much fuel is required to travel from one street to another
- Fuel_Used
  - When a vehicle travels from one street to another, it consumes fuel which this function keeps tabs on
    - This is also used as one of the metric parameters to minimize
- Fuel_Wasted
  - In order for to identify which vehicle is more efficient we decided a function called fuel_wasted would be a good way of indicating fuel efficiency
    - The reasoning being that a car would waste more fuel then a motorbike therefor a higher fuel_wasted would indicate a less efficient vehicle

Domain File

```
(define (domain PizzaDomain)
(:requirements :strips :typing :fluents :durative-actions)
(:types vehicle pizza street fuel_Pump)
(:predicates
  (at-vehicle ?v - vehicle ?s - street)
  (fuelPump_Location ?p - fuel_Pump ?s - street)
  (pizza-baked ?p - pizza)
  (at-pizza ?p - pizza ?l - street)
  (pizza-loaded ?p - pizza ?v - vehicle)
  (address ?p - pizza ?s - street)
  (pizza-delivered ?p - pizza)
  (street-linked ?x ?y - street))

(:functions (fuel_level ?v - vehicle)
        (fuel-required ?s1 ?s2 - street)
        (fuel_used ?v - vehicle)
        (fuel_wasted ?v - vehicle))

(:durative-action drive
   :parameters (?v - vehicle ?from ?to - street)
   :duration (= ?duration (fuel-required ?from ?to))
   :condition (and (at start (street-linked ?from ?to))
                (at start (at-vehicle ?v ?from))
                (at start (>= (fuel_level ?v) (fuel-required ?from ?to)))
                )
   :effect (and (at start (not (at-vehicle ?v ?from)))
           (at end (decrease (fuel_level ?v)(+ (fuel-required ?from ?to)(fuel_wasted ?v))))
           (at end (increase (fuel_used ?v) (+ (fuel-required ?from ?to)(fuel_wasted ?v))))
           (at end (at-vehicle ?v ?to))))

(:durative-action load-pizza
   :parameters (?v - vehicle ?s - street ?p - pizza)
   :duration (= ?duration 2)
   :condition (and (at start (pizza-baked ?p))
                (at start (at-vehicle ?v ?s))
                (at start (at-pizza ?p ?s)))
   :effect (and  (at start (not(at-pizza ?p ?s)))
           (at end (pizza-loaded ?p ?v))))

(:durative-action deliver-pizza
   :parameters (?v - vehicle ?s - street ?p - pizza)
   :duration (= ?duration 3)
   :condition (and (at start (address ?p ?s))
               (at start (at-vehicle ?v ?s))
               (at start (pizza-loaded ?p ?v)))
   :effect (and  (at start (not(pizza-loaded ?p ?v)))
           (at end (pizza-delivered ?p))))

(:durative-action swap-vehicle
   :parameters (?v1 ?v2 - vehicle ?s - street ?p - pizza)
   :duration (= ?duration 5)
   :condition (and (at start (at-vehicle ?v1 ?s))
               (at start (at-vehicle ?v2 ?s))
               (at start (pizza-loaded ?p ?v1)))
   :effect (and  (at start (not(pizza-loaded ?p ?v1)))
           (at end (pizza-loaded ?p ?v2))
           ))
(:durative-action refuel
   :parameters (?v - vehicle ?f - fuel_Pump ?s - street)
   :duration (= ?duration 6)
   :condition (and (at start (at-vehicle ?v ?s))
               (at end (fuelPump_location ?f ?s)))
   :effect (and (at end (increase (fuel_level ?v) 10))))

(:durative-action bake-pizza
   :parameters (?v - vehicle ?p - pizza ?s - street)
   :duration (= ?duration 4)
   :condition (and (at start (at-vehicle ?v ?s))
               (at start (at-pizza ?p ?s)))
   :effect (at end (pizza-baked ?p))))
```

Problem File 1

```
(define (problem simpleProblem)
 (:domain pizzaDomain)
 (:objects
    motorbike car - vehicle
    p1 p2 p3 p4 p5 p6 - pizza
    s1 s2 s3 s4 s5 s6 s7 s8 - street
    fp1 fp2 - fuel_pump)
 (:init
(street-linked s1 s2) (= (fuel-required s1 s2) 5)
(street-linked s1 s3) (= (fuel-required s1 s3) 7)
(street-linked s1 s4) (= (fuel-required s1 s4) 10)
(street-linked s1 s6) (= (fuel-required s1 s6) 12)
(street-linked s1 s7) (= (fuel-required s1 s7) 12)
(street-linked s2 s1) (= (fuel-required s2 s1) 5)
(street-linked s2 s3) (= (fuel-required s2 s3) 10)
(street-linked s2 s4) (= (fuel-required s2 s4) 6)
(street-linked s3 s1) (= (fuel-required s3 s1) 7)
(street-linked s3 s2) (= (fuel-required s3 s2) 10)
(street-linked s3 s4) (= (fuel-required s3 s4) 7)
(street-linked s3 s5) (= (fuel-required s3 s5) 4)
(street-linked s3 s6) (= (fuel-required s3 s6) 11)
(street-linked s3 s8) (= (fuel-required s3 s8) 10)
(street-linked s4 s1) (= (fuel-required s4 s1) 10)
(street-linked s4 s2) (= (fuel-required s4 s2) 6)
(street-linked s4 s3) (= (fuel-required s4 s3) 7)
(street-linked s4 s5) (= (fuel-required s4 s5) 6)
(street-linked s5 s3) (= (fuel-required s5 s3) 4)
(street-linked s5 s4) (= (fuel-required s5 s4) 6)
(street-linked s5 s6) (= (fuel-required s5 s6) 10)
(street-linked s5 s7) (= (fuel-required s5 s7) 2)
(street-linked s6 s1) (= (fuel-required s6 s1) 12)
(street-linked s6 s3) (= (fuel-required s6 s3) 11)
(street-linked s6 s5) (= (fuel-required s6 s5) 10)
(street-linked s6 s8) (= (fuel-required s6 s8) 6)
(street-linked s7 s1) (= (fuel-required s7 s1) 2)
(street-linked s7 s5) (= (fuel-required s7 s5) 2)
(street-linked s7 s8) (= (fuel-required s7 s8) 8)
(street-linked s8 s3) (= (fuel-required s8 s3) 10)
(street-linked s8 s6) (= (fuel-required s8 s6) 6)
(street-linked s8 s7) (= (fuel-required s8 s7) 8)

 (fuelPump_Location fp1 s3)
 (fuelPump_Location fp2 s5)

 (at-vehicle motorbike s2)
 (= (fuel_level motorbike) 30)
 (= (fuel_used motorbike) 0)
 (= (fuel_wasted motorbike) 2)

 (at-vehicle car s1)
 (= (fuel_level car) 30)
 (= (fuel_used car) 0)
 (= (fuel_wasted car) 5)

 (at-pizza p1 s4) (at-pizza p2 s4)
 (at-pizza p3 s4) (at-pizza p4 s4)
 (at-pizza p5 s4) (at-pizza p6 s4)

 (address p1 s6) (address p2 s3) (address p3 s3) (address p4 s7) (address p5 s8) (address p6 s2)
 )
 (:goal (and
        (pizza-delivered p1)   (pizza-delivered p2)
        (pizza-delivered p3) (pizza-delivered p4)
        (pizza-delivered p5)
        (pizza-delivered p6)))

 (:metric minimize (fuel_used car)))
```

This is one of the problem files used during analysis.

It models

- 8 streets
- 18 Links
- 6 Pizzas

An image of the problem we modelled is included below.