**20**

15

**Pizza Delivery using PDDL**

Introduction to Artificial Intelligence Coursework 2015

Group Members:

- Tahmidul Islam (K1462015)

- Hassan Minhas (K146907)

- 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.*

08

**Fall**

**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.

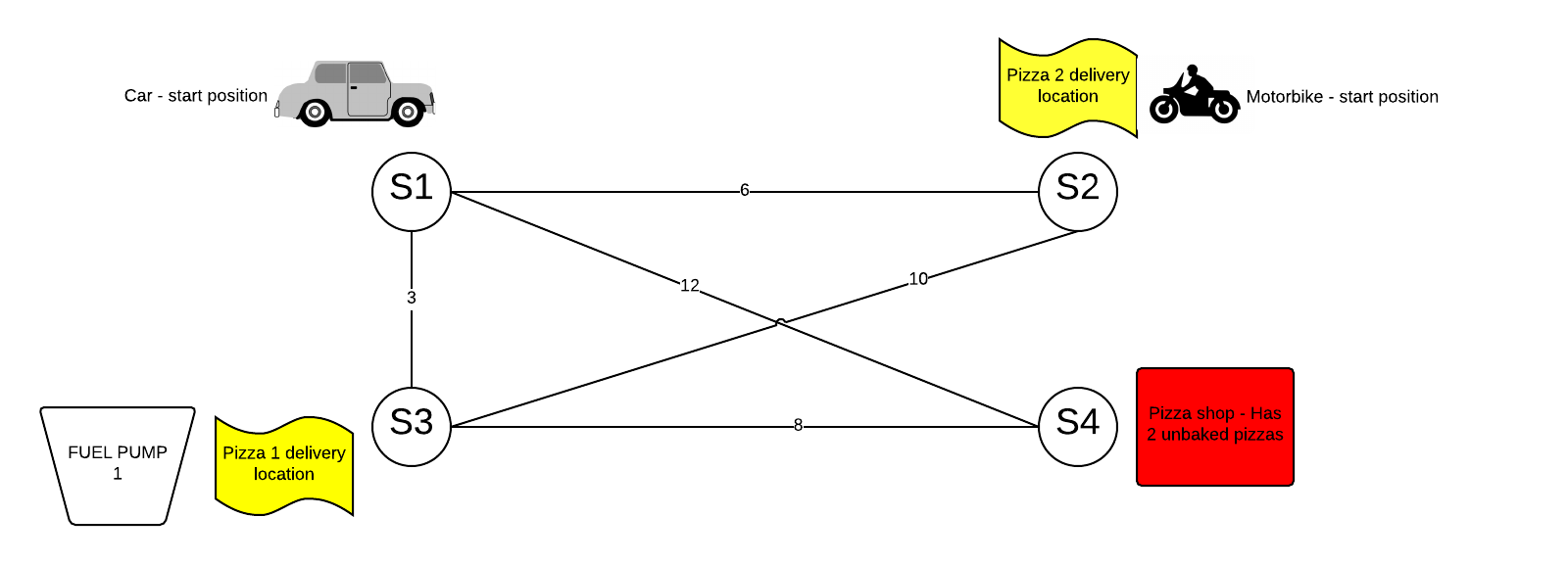
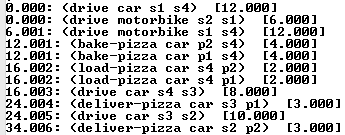
Furthermore since no two orders will be the same, we have taken into account that our domain needs to be domain-independent and therefore able to work with multiple types of problems involving different vehicles, number of streets, number of pizza orders etc. We have therefore run multiple tests using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

Example:



Domain File

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

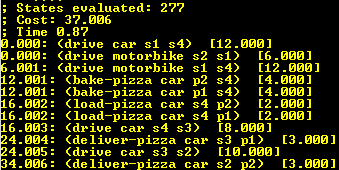
enter image description hereAs we came to realise there was a substantial number of factors to consider and after much consideration it was decided that our main method of upscaling would be increasing number of streets, connections and pizza. Furthermore, we were aware that increasing the number of 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 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.

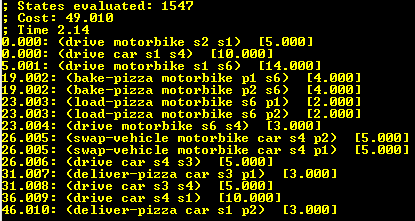
For the number of pizzas needing delivering we decided to increment it by 2, every time we increased the number of streets. When deciding on the number of cars to use, we felt 2 would be enough as this would allow the planner to make use of the swap-vehicle action we created. We kept the number of cars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

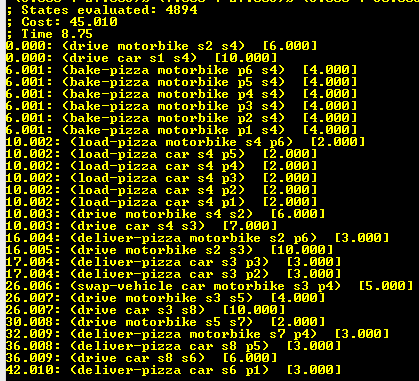
|  |  |  |
| --- | --- | --- |
| Number of streets | Number of connections | Number of pizzas |
| 4 | 5 | 2 |
| 6 | 10 | 4 |
| 8 | 18 | 6 |
| 10 | 27 | 8 |

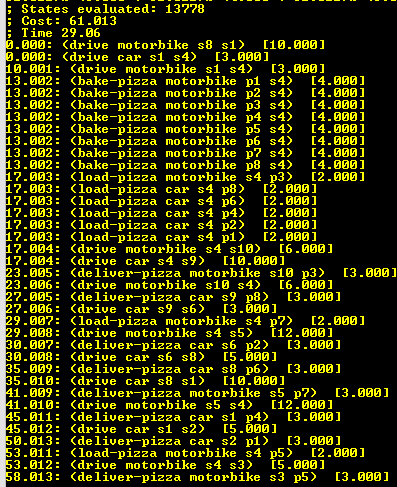


After creating the problem files and running the planner it was clear to see a clear trend between the number of streets (as well as connections, pizza etc.) and also the time taken for the planner to generate a solution. When running the first problem of using 4 streets, the planner was relatively quick, due to the fact that only 277 states needed to be evaluated. But after it got to 8 streets, the time the planner took to product a solution increased almost ten fold. It was clear to see the reason being because at 8 streets the planner had to evaluate 4894 states.

The graph showing the







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.

**Results**

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.

Appendix

(Pizza Domain, problem file)

**Report Plan**

Intro –

Our domain

* Briefly explain idea behind project
  + ~~Pizza delivery is a way of life~~
  + ~~In London, delivery has become a crucial aspect of restaurant/takeaways.~~
  + ~~Competitive~~
  + ~~Delivery can take up a lot of resources~~
  + ~~Improve delivery service~~
  + ~~This planner could be used to produce the most efficient solution for delivering pizzas (or any other type of food) to customers~~
* ~~Similar to Maps application, but takes into account factors specific to the domain. ( vehicles, fuel, etc)~~
* Talk about problem relaxations and limitations of domain
  + What we don’t take into account and why
    - ~~Pizza baking – In our domain, we assume that the pizzas have already been made. We do not take into account the cooking time of the pizza. The domain is strictly limited to delivery (more on this in the evaluation, see below)~~
    - ~~The planner is limited in that it can only take one set of deliveries at a time. In reality, the business could have a constant stream of deliveries. A plan could be produced for the current set of deliveries. However, there could also be more orders, thus making the current plan obsolete. The situation of the business could change within minutes, especially at peak-time.~~
* Explain choice of planner
  + ~~We tested the domain and planner using Optic~~
  + Say what type of search the planner uses

Domain -

* Briefly describe types, functions, durative-actions
  + Why we used durative actions
    - The key concept of the domain is to find approximate timings for delivering  sets of pizzas to specific areas
    - We made the main subtasks of pizza deliveries as durative actions so that the planner takes these subtasks into account when calculating time

Results -

* Discuss results of several problem files of varying sizes (table, chart, etc)
* Illustrate the time of the planner against the size of the problem
* use graphs when possible
* Compare times of the same problem between Optic and JavaFF?

Evaluation

* Issues during the project
  + To add action for baking the actual pizza
  + One issue that we found with this is that it would be too big to implement the baking side of the domain.
  + For this project, we wanted to focus on the logistics side of pizza delivery.
* Points of interest of the domain
* How to expand the domain in the future
  + We mentioned earlier about an issue we had with the domain, and deciding if we should add pizza baking into the domain.
  + This would be worth exploring in the future.
  + We could make it so that ‘orders’ are delivered to the customer instead of just pizzas. These order object could consist of several other objects such as pizzas, sides, etc. These could each have durative actions.

Appendix