Essay 2: Airport Optimization

Paul Beggs

November 10, 2025

1 Introduction

In this essay, we will discuss how modern airport problems such as delayed flights, parking congestion, and trash pickup can be mitigated with the help of logic programming, supervised and unsupervised learning, and reinforcement learning, respectively.

2 Logic Programming

Logic programming can be used for transparent communication regarding cancelled, delayed, and on-time flights. Our system can operate with two components: facts (made up of raw data) and rules (in the form of a horn clauses). In the following example, we will examine how a delayed flight can be translated to airport staff. With these facts made up of metrics from the plane and from weather stations, we get the following:

- `position UA2322, (32°30'16.4"N 93°24'23.7"W)` (this is the position of flight UA2322 at those coordinates),
- `speed UA2322 480`,
- `weather STL turbulent`, and
- `runway_status STL 09L` (09L is the runway designation for 90° left).

Then, for horn clauses, we would have many predicates in the body:

```
is_approaching ?plane ?airport <-
 position ?plane ?p_coords;
 airport_location ?airport ?a_coords;
 calculate_distance ?p_coords ?a_coords ?dist;
 less_than ?dist 10;
 speed ?plane ?s;
```

This literally translates to "A `?plane` is approaching an `?airport` if we find the `?plane`'s `?p_coords`, we find the `?airports`'s `?p_coords`, we can `calculate_distance` between them to get `?dist`, that `?dist` is `less_than 10` miles, and we find the `?plane`'s speed `?s`." Now, we

add one more rule: `is_in_pattern ?plane <- is_approaching ?plane ?airport; weather ?airport turbulent`. If this pops up, then we learn that the airplane is delayed and is currently flying around waiting for the weather to subside.

Now, we can examine a propagation example: we feed the facts into the horn clause to get an output. The system checks `is_approaching ?plane ?airport`. It then binds `?plane` to `UA2322`, `?airport` to `STL`, and `?p_coords` to `(32°30'16.4"N 93°24'23.7"W)`. Then, we use `calculate_distance` and bind `?dist` to `1.7` miles. Then we run `less_than 1.7 10`, which evaluates to true, and tells us that the plane is close to the airport. Following that, it binds `?s` to `480` miles per hour, so that we know the speed. Thus, all conditions are met, and a new fact is created: `is_approaching UA2322 STL`. We use this with another fact `weather STL turbulent`, and find that `is_in_pattern UA2322` is the reason for the delay.

3 Supervised Learning

We could employ supervised learning to help identify cars and alleviate parking congestion by utilizing cameras that are capable of scanning single lots in a parking garage for cars. We would have to have a camera for every lot, but let's assume that the airport has this capacity. Using individual cameras has the benefit of allowing airport staff to know which exact car is in the lot for security purposes, and allows them to know if a lot is empty or occupied. For this capacity to be achieved, we will employ a k-nearest-neighbors algorithm that uses images and labels for cars. In this procedure, we must label each image of a car with its name (or access that image/label data through a database). With this data, we can store all training labels in a (very) large array. Then, we classify new information with a distance function to compare the images (e.g., 0 implies the two images are the same, and anything greater than 0 implies they are different). Then, we pick each of the closest pictures and put them in a heap to find the lowest distances.

Once we have our images, the labels, and their relative distances, we can move onto classification for new images. For example, we could introduce a picture of a Hyundai Palisade and evaluate it among a list of other cars using a histogram. For example, the Hyundai Palisade looks similar to a Kia Telluride, so our histogram would tally each car image/label and see which one gets the most votes. Depending on this outcome, we would then be able to say that the car is, in fact, a Hyundai Palisade.

Finally, we must also talk about identifying if a spot is empty. This can be done by using many pictures of vacant lots. In that, to tell if it is empty, we must compare the pixels of a lot that is labeled empty, and the new lot that looks empty. Since these images (should) be almost identical, the algorithm can identify the lot as empty or at least occupied more often than not. Note that, in practice, we should have pictures ranging from all different times of day for the empty lot so that our pixel values properly correspond to the ranging light values (this is also true for the cars). Maybe we could have subsets for different times of day, and then when it comes to grading, we have a function to get the current time so that we can compare images of the same time.

4 Unsupervised Learning

We can use unsupervised learning in the form of a self organizing map (SOM) to help airport staff direct traffic. We would train the SOM using multiple pictures of the lot at varying fullness. This has the added benefit of creating regions on the map that correspond to different full levels (e.g., "25% full", "50% full"). The nodes in the map would then be used for identifying states of the lot like "empty," "X amount full," and "packed." Now, when a new picture is fed into the system, the SOM identifies the best matching node, and classifies the current state of the lot. Then, this information can be fed to an airport staff worker at a tollbooth to direct traffic to the distinct zones in the lot.

5 Reinforcement Learning

We have only talked about reinforcement learning in the context of Q-tables and robotics. So, for the airport, we can try to properly implement the failed robotic janitor from Essay 1 with this method. That is, we will build a robot that uses Q-learning for picking up trash and avoiding non-trash objects. To start, the robot will use a 2-dimensional map that consists of states and actions. The actions, for example, could be "move forward," "turn left," "turn right," and "move backward," and the states could be the sensor's readings (e.g., "trash directly ahead," "obstacle on left").

During training, the robot would spend its time in a simulation (the "exploration phase") to populate values on the Q-table. That is, it would try an action, consequentially receive a reward (e.g., +10 for collecting trash, +5 for avoiding objects, -7 for bumping luggage), and update the Q-value for that state-action pair. During this training, our learning rate will fluctuate: At first, it's close to 1, so we disregard the previous Q-value, and update the state with new information, and we continue to lower it as we explore more of the space. This comes directly from the equation,

$$Q(s,a) = (1-\alpha)\big(Q(s,a)\big) + \alpha\big(\gamma \cdot Q(s', besta) + r(s)\big),$$

where $\alpha \in [0,1]$ is our dynamic learning rate, $\gamma \in [0,1]$ is our fixed discount, and $r(s)$ is the reward function. Thus, by training this way, we deploy a robot that has already learned from its mistakes, and is capable of effectively picking up trash while avoiding non-trash in the airport.

6 Conclusion

We turned to logic programming to help with airplane statuses by using facts, horn clauses and Boolean logic. This allows us to use metric to gauge availability. Then, we explored an implementation of recognizing cars in a parking lot for uses such as security and spot by spot availability by using supervised learning. We also explored unsupervised learning for lot recognition to get specific percentages of how full a lot is. Thus, we can see that supervised and unsupervised learning's applications are similar in using images for classification, but

different in how they are gauging availability. Finally, we used reinforcement learning to help with trash pickup in the airport by using a simulated environment to train the robot, and then when we deploy the robot, we switch it to it's exploit mode to do the actual cleaning.