Dynamic Allocation of Emergency Resources

CS221 Milestone Report

Tyler Romero (tromero1@stanford.edu)
Zachary Barnes (zbarnes@stanford.edu)
Frank Cipollone (fcipollo@stanford.edu)

***Abstract*—By dynamically allocating resources such as personnel and vehicles in emergency situations, we can provide a solution that minimizes damage and loss of life while also conserving taxpayer dollars.**

Keywords—public safety; emergency events; reinforcement; MDP; resource allocation; time series

I. Motivation

Each year, emergency responders assist in millions of critical events across the country. The size of an emergency response can scale from a single paramedic to multi-state crews fighting large fires. As emergency events, the time it takes first responders to arrive on scene is critical, with minutes often making the difference between life and death. Because of these factors, standard staffing and resource use is very high to make sure enough responders are available at any given time. These factors make emergency response an important potential application for optimizations. Thus, a model that makes dynamic decisions on allocation of resources would be extremely useful to government and department management in day to day operation.

II. Approach

A. Goals and Scope

Our goal is to use historic per day emergency incidents in a specific geographic region as an input for our model. We will frame this application as a reinforcement learning problem where training examples will be days of historic data drawn from the pool of emergency events for the region as well as potential relevant weather, geographic, structural, and demographic features that may be added if our simple data source is found lacking. This will allow our model to make an optimal allocation of resources over our region of interest at any given time.

The particular region of interest that will be explored is the city of San Diego. Specifically, we will be allocating fire trucks for the San Diego Fire Department. Our model is time sensitive and will be evaluated at time steps covering a 24 hour time period. Our model will be fed a transcript of incidents over a day as its input (fed at the corresponding time step), and will output the locations that the fire trucks have moved to at each time step. We will measure model performance by simulating our model over a test set of randomly selected days, and returning the average response time per incident. Models can then be compared on the basis of their average response time. Response time will be calculated as the minimum distance between an available fire truck and an incident multiplied by a basic speed of 40 mph.

A concrete example input for a timestep would be the following:

[FIRE, Vehicle Fire Freeway,
2016-08-18T23:51:56, (32.839, -117.144)]

And the output for the timestep would be a dictionary with a fire truck number as a key and a location destination (for the following timestep) as an output. E.g. for one timestep with 2 trucks:

{truck1: (32.839, -117.144), truck2: (32.822, -116.909)}

This input, output behavior mirrors that of a real life decision-maker where incidents are reported and directives are given. Our different model

implementations will be compared via response time (the difference in time from when an incident is announced and when a vehicle arrives on the scene). This is an important real world metric, and given our formulation, can be calculated based on input and outputs allowing us to compare models.

B. Data Sources (concrete data example)

Our source of historic emergency incidents is the San Diego Open Data Portal which provides every fire incident responded to in the last year. This dataset is comprised of the type, location, date and time, and category of severity for approximately 150,000 incidents [2]. Historic fire data will possibly be supplemented with weather [3], geographic [4], and demographic data [5] corresponding to the region of interest if needed.

C. Baseline and Oracle

A perfect approach, or oracle, would result from having all of the important knowledge of all future emergency incidents. Thus, this oracle would route trucks so that for any incident a truck is already present and available. A baseline approach would result from knowing nothing about future incidents nor making active routing decisions. This approach would for any given incident simply route the closest truck to the incident and nothing more.

After implementing the two approaches, we found that (depending on initial assumptions), the baseline had a quite high total error, whereas the oracle was very close to zero. This difference is exactly what we expected. The oracle knew where the incident would be taking place and was able to route fire trucks there before the incident actually happened. Therefore, there was always a truck ready to respond exactly where it was needed and when it was needed. On the other hand, the baseline approach had the trucks blindly wander around the city until there was an incident. Even the closest truck was usually far from the incident. Our solution will lie somewhere between these extremes.

D. Our Model

Our approach to a solution is to utilize reinforcement learning to learn in the uncertainty of not knowing when and where future incidents will occur.

The first important design decision we made was the best way to represent each state at which a decision was needed. We need to account for a variable number of firetrucks, as well as possible different time step granularity and different truck speed limits. These are parameters we felt should be set by the user and not hardcoded into the model. We decided to force the state to hold the following information:

1. The position of each truck and it's current assignment
2. Every active incident on the grid of San Diego. An incident will be removed from the state after a truck occupies the same grid space for a certain number of timesteps.

Then, we could simply define the choice of action as we represented it in section *goals and scope*: a new assignment for each truck. Importantly, actions are *not* a specific truck movement. At each timestep, the trucks will be automatically moved (as fast as they can go, as defined by their speed limit) towards their current goal. The only action is an update on this goal.

We face an important challenge in modelling this problem with a Q-learning algorithm. We note that it is not feasible to iterate through every successor state of our model. To show why this is, we note that the list of actions for a state with n trucks and a dxd grid will include every permutation of assignments of each truck to each location on the grid. In other words, the list of successor states is:

$$O((dxd)^n)$$

This obviously becomes an infeasible number to iterate through very quickly. In order to solve this problem, we are forced to abandon listing every state and use a different kind of successor state selection

State Selection

We want to choose between all of these states, but we can immediately limit our search space, at least intuitively. First, we will almost never want to assign a truck to an incident that a different truck could get to faster. Second, we don't want any two assignments to be too close to each other; it would generally be inefficient to have two trucks in the same place. Third, if there are free trucks and active incidents, these trucks should be assigned to these incidents.

Our proposed state selection is a randomized state selection, guided by heuristics. It takes into account the above three ideas in the following ways:

1. Rather than choosing the best *n* assignments for each of *n* trucks, we will be much better off iterating through selections of *n* points, and for each point, simply assigning the closest truck to that point.
2. We use a probability distribution that makes it unlikely that many points will be placed in the same vicinity
3. Before calculating any random point, we include all active incidents in our list of *n* points (if there are more incidents than trucks, we assign each truck to the closest incident).

Our successor state space size can now be a variable! We can simply decide how many randomized states we want to look through, allowing us to easily work with the tradeoff of performance versus running time.

A final important note about choosing the next possible states is that we *always* include two possible states in addition to the random possibilities.

1. Include the state where each truck is assigned to the space it currently occupies. We want to consider the option that our state score will not be improved by moving
2. Include the state where each truck is assigned to the same space it was assigned to in the previous time step. This is ensure that we do at least as well as the last step, and effectively allows the work done in the last time step to persist, if it is beneficial.

Having defined our states, the second major design decision we had to make was how to pick the optimal feature selector.

Feature Selector

Fortunately, it is fairly clear which features of the state are important in the calculation of how good a state is. At a low level, we want trucks to be far from other trucks and we want trucks to be close to incidents. Secondarily, we thought an important predictor might be time of day, so we include that as a feature. Concretely, we include the following features:

1. Average distances between trucks
2. Distance to any active incidents
3. Hour of the day

We do not include any information about the current state in these features. This is important because we will have, as we said before, an infeasible number of states. If we included this information, we would expect to learn nothing because there would be so many features it would be unlikely we would see the same one twice (even if we could store all this information).

E. Challenges

One of our primary challenges remains identifying the best feature extractor for our reinforcement learning model. Since the state space of our MDP is extremely large, it is necessary for our feature extractor to allow our reinforcement learning algorithm to generalize well.

F. Results

We ran two experiments to determine if our model was learning anything compared to a baseline approach. In running these experiments, we kept all explicit routing decisions the same, only comparing the decisions made by Qlearning to the random decisions made by the baseline. We used

the first five days of data from 2009 for this experiment, as well as a 10 by 10 grid representing San Diego, and 3 firetrucks.

In our first experiment, we tested if Qlearning could discover how to get to incidents without explicitly being told. We first ran a random walk, which made completely random routing decisions. Then we used Qlearning (without any heuristic helping to generate good actions) to make all of the routing decisions. Each average response time was highly varied, but the following are the averages over three runs:

	Pure Random Walk	Qlearning - No Heuristic
Average Response Time	210 time steps	115 time steps

This shows that simply by using Qlearning and a feature extractor with basic features, our model learns to outperform random walk when it comes to finding incidents.

In our next experiment, we tested the more realistic situation in which, when a call is made, the nearest truck is routed directly to the location. Here, the only decisions are where to send trucks while they are waiting for a call to be made. Greedy Truck Allocation has the trucks move randomly when there are no ongoing incidents. Qlearning with Heuristic takes advantage of features that give the trucks an idea of their proximities to one another and to ongoing incidents. The response times here were less varied:

	Greedy Truck Allocation	Qlearning - With Heuristic
Average Response Time	1.53 time steps	1.34 time steps

These results show that Qlearning can outperform Greedy Truck Allocation when it comes to average response time. This indicates that our Qlearning model has a better sense of balance between the locations of trucks prior to incidents being reported.

G. Next Steps

Looking ahead, we plan to make our model more detailed. For example, adding Ambulances that can respond to non-fire emergencies. We will also scale up our tests to a 100 by 100 grid representing San Diego with 10 firetrucks and several years worth of data.

In addition, our tests will become more structured. We will train our Qlearning algorithm on one set of data, then set the probability of random exploration to zero and test it on another set of data. We will then compare it to a similarly set up naive algorithm.

Finally, we will improve upon our feature extractor by adding additional features to help our Qlearning algorithm generalize better.

H. Similar Applications

A similar application of machine learning to help emergency responders was done by Bayes Impact. They analyzed Seattle police report data in order to determine ways in which Seattle could better deploy officers with the goal of minimizing serious and violent crime [6].

I. Source Code

Source code for our project can be found at https://github.com/tyler-romero/fire-prediction.

References

[1] Haynes, Hylton JG. “Fire loss in the United States during 2015.” National Fire Protection Association. Fire Analysis and Research Division, 2016.

[2] *San Diego Open Data Portal*. The City of San Diego, n.d. Web. 21 Oct. 2016.

[3] *Weather Underground - Weather Forecast & Reports*. N.p., n.d. Web. 21 Oct. 2016.

[4] *SanGIS*. San Diego Geographic Information Source, n.d. Web. 21 Oct. 2016.

[5] "San Diego Demographic Data." *Census.gov*. N.p., n.d. Web. 21 Oct. 2016.

[6] Wong, Jeff. "Walking the Beat: Mining Seattle’s Police Report Data." *Bayes Impact*. N.p., n.d. Web. 21 Oct. 2016.