Minerva Schools At KGI

CS166 - Final Project - Emergency Response

Quang Tran

CS166 Fall 2019

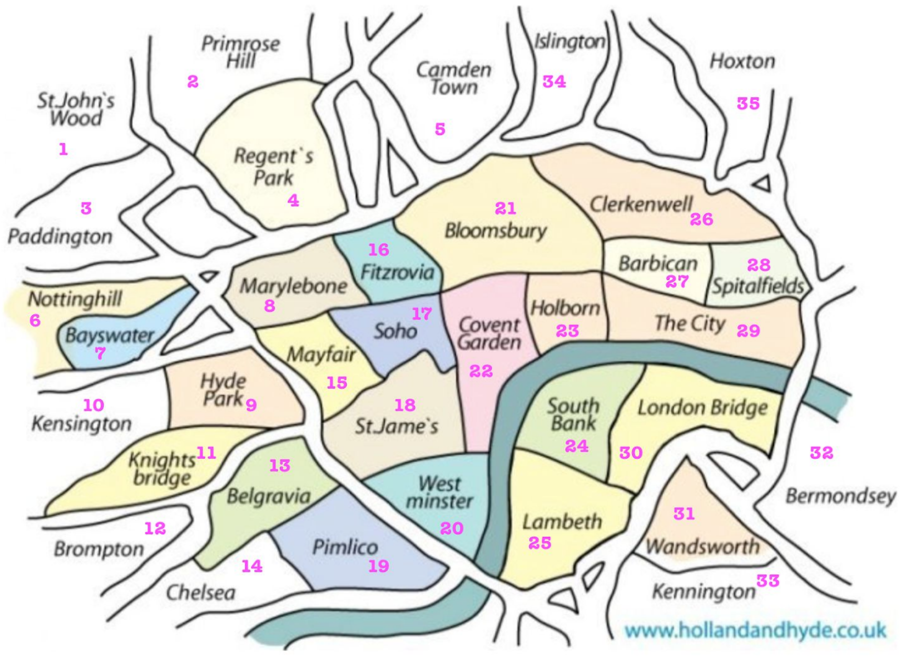
Problem Formulation

When emergency calls arise, ambulances are sent to the emergency locations. Given the relevant information of the neighborhoods in the city, the limited number of ambulances, how to allocate ambulances to different despots so that emergency calls can be attended to as quickly as possible?

The relevant information about each neighborhood includes:

1. The rate at which emergency occurs, λ, measured in **calls per minute**.
2. The distance between the neighborhood and the bordering ones, d, measured **miles**

In this Report, we examine a specific, hypothetical city, inspired by the city map of London below:



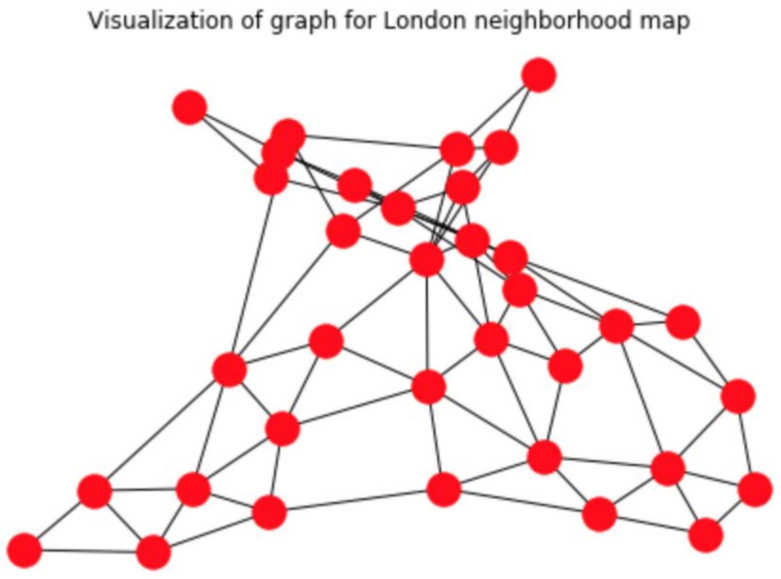
Therefore, the city can be represented by a network with each neighborhood being a node named an integer from 1 to 35.

1. The distance matrix is estimated visually from the map above and is given in the `graph_data.cvs` file.
2. The rates of emergency calls is also hypothetical and are assumed to be given in the table below:

Neighborhoods	**Rates of call (calls/minute)**
1,2,3,4,5	1/30
6,7,9,10,11	1/26
8,15,16,17	1/20
12, 13, 14, 19	1/25
18,20	1/16
21,22,23	1/22
26,27,28,29	1/21
24,25,30	1/24
31,32,33	1/28
34,35	1/31

In practice, the rates of calls can be extracted from past data of the city.

The graph created using `networkx` representing the city map is printed below.



We take this graph and its information as given: the rate of emergency calls of each node and the distances among the nodes. Throughout this project, we will *not* attempt to experiment with different configuration of distances and rates of emergency, as the goal is not to design an allocation strategy that fits *all* graph structures. Rather, the goal is to find one strategy that works for this specific setting of the city. This is more likely to be practical: we design and propose a working strategy for a specific city.

The General Model and The Update Rules

Each node has two attributes:

1. λ : the rate of emergency, measured in calls/min.
2. n : the number of ambulances. This number is positive only if the node is designated as a despot to store ambulances, and n is the number of ambulances. When a call arises, and an ambulance at this node is dispatched, we have to update $n \leftarrow n - 1$ to reflect the change in the number of ambulances. $n = 0$, therefore, means that either the node is not a designated despot or all the ambulances are on duty.

We will discuss the two strategies to allocate ambulances in a later section. This section focuses on the call request generation rules and the dispatch rules.

Call Request Generation

Emergency requests are generated following these rules:

1. Requests for one node are generated independently from those in any other node.
2. In each node, one request arises independently from another. This is reflected in the choice of the exponential distribution given in the next point.
3. For each node, we generate the minutes in the day a call request occurs. For example, if a call arises at 9AM, then $9 * 60 = 540$ is generated. However, we don't generate the time stamps directly. Rather, we start from 0 (12AM), and consecutively generate the waiting time between two consecutive calls. For example, if the first call in the day occurs at 0:30AM, then 30 is first generated. Suppose the next call occurs 12 minutes later, then 12 is generated. We generate these numbers until the day has passed (no more than 60*24=1440 minutes). The waiting time between two consecutive calls is assumed to come from an exponential distribution:

$$\textit{waiting time} \sim f(x; \lambda)$$

Where

$$f(x; \lambda) = \lambda e^{-\lambda x}$$

Exponential distribution is chosen because:

1. The support is positive. We don't want negative waiting time, which would not make sense.
1. The exponential distribution models the waiting time between two events in a Poisson process, which is exactly what we are modeling, if we assume that two events are independent.

2. The mean of this distribution is λ, matching what we set for λ of each node -- the average rate of emergency.

Dispatch Rules

1. When an emergency case happens, a request r is sent. Let :
   a. $t_0(r)$: the time r is sent. A call center is assumed to immediately receive the call (i.e., at $t_0(r)$)
   b. $t_{max}(r)$: the maximum time the patient associated with request r can wait until an ambulance arrives. If no ambulance has arrived by $t_0(r) + t_{max}(r)$, the patient dies. $t_{max}(r)$ is assumed to come from a normal distribution:

$$t_{max}(r) \sim Normal(\mu_0, \sigma_0)$$

Where μ_0 and σ_0 are parameters of the model. In the simulation, I chose $\mu_0 = 15\ min$ and $\sigma = 1\ min$

   c. $l(r)$: the location (the node) at which the emergency happens;
   d. $d(r)$: the difficulty level of the case. The higher d is, the longer it takes the ambulance to attend to (after arriving at the location). $d \in [1, 2, 3]$ and is drawn with relative probabilities [0.3, 0.6, 0.1]. The average time it takes an ambulance to attend to the case is $10d$ (minutes).
2. When a call request r occurs, an ambulance may be dispatched following these rules:
   a. If there are no free ambulances in the entire city, put r on hold.
   b. If there is at least one ambulance free, dispatch the one at the despot closest to $l(r)$. Distance D measured by the shortest single-source weighted path from the despot to $l(r)$. In Python implementation, we use Dijkstra algorithm to do this.
   c. The time it takes an ambulance at despot y to get to node $x = l(r)$ (where the call occurs) is computed by:

$$t = 60 * D/v(ambulance) + Normal(\mu = 0.2 * 60 * D/v(ambulance),\ \sigma = 1)\ \text{(minutes)}$$

Here, D is the distance between two nodes, $v(ambulance) = 30$ (mph) is the average speed of an ambulance. We add a random term drawn from a normal distribution to capture other random factors: traffic, street layouts, weather, etc. and the time to get to the exact location of the emergency in that node. We have to multiply by 60 to convert from *hours* to *minutes.* The standard deviation is small (1 minute) to make sure the chance that t is negative is effectively negligible. Again, we will take the average speed of ambulance as something given and fixed and will not parameterize it. The coefficient $\alpha = 0.2$ for μ and the σ for the Normal distribution can be used to be the parameters of this model, but we won't do that either.

   d. After the ambulance has arrived at the scene, it takes :

$$t \sim Normal(\mu = 10d(r),\ \sigma = 1)\ \text{minutes}$$

to attend to the case.

   e. After servicing the case, the ambulance gets back to its despot in:

$$t = 60 * D/v(ambulance)$$

We don't add a random term this time as we would cheat by assuming the random term is also incorporated in the random term for the attending time.

f. When two or more requests have to be processed (this can happen when a request previously was put on hold due to a shortage of ambulances and now it has to process another request):
    i. Requests sent earlier are prioritized and processed before requests sent later (i.e., according to $t_0(r)$ but with a catch:
    ii. If the patient has died (this happens when the time has passed the maximum endurance time $t_0(r) + t_{max}(r)$ of the patient), then the request is cancelled. We move on to other requests.

The Two Allocation Strategies

1. The totally naive strategy

As the name suggests, we make each node a despot, and evenly distribute the ambulances across despots.

2. The k-medoids clustering strategy (5 sub-strategies)

In this strategy, we divide the nodes into K clusters. In each cluster, we choose a node to be the despot of that cluster. Therefore, we designate K nodes in total as despots to store ambulances. Then, the number of ambulances allocated to a despot is proportional to the rate of emergency in the cluster. The rate of emergency of a cluster is simply the sum of the rates of emergency calls of the nodes in that cluster.

But how to divide the nodes into K clusters? Here, we use the K-medoids clustering method.

The K-medoids algorithm is similar to the K-means algorithm, which aims to minimize the cost of clustering, which is the total distances of the nodes to their respective centers of the clusters. Unlike K-means algorithm, the center of a cluster must be one of the nodes (in K-means, the center does not have to be one of the data points, as it's the mean of all the data points.) However, it's worth noting that K-medoids algorithm may reach a suboptimal solution.

The distance used in this algorithm when applied to our context is, as before, the length of the shortest single-source weighted path.

The algorithm is as follows:

1. Randomly select K nodes as designated medoids.
2. Compute the cost c of such assignment.

```
3. While True:
    a. For each medoid $m$ of the current set of medoids and for each non-medoid $n$ :
        i. Compute the new cost assuming we have swapped the medoid - non-medoid
           statuses of $m$ and $n$
        ii. If the new cost is not lower than $c$, we undo the swap.
        iii. If the new cost is lower than $c$, we record $(m, n)$ along with the reduction in
             cost.
    b. Select the pair $(m, n)$ with the largest positive reduction in cost and do the swap for
       $(m, n)$. If there is no pair that results in a reduction in cost, then terminate the algorithm.
```

Because I plan to assess the strategy on 5 different values of K, this strategy actually includes 5 strategies to evaluate.

There are two issues with this strategy:

1. The running time is practically large (although it's polynomial). It took about half a minute for one run.
2. The solution it gives may be different for different runs, because this is a greedy and random approach.

The above two issues pose a great problem for the simulation: we can't just run the algorithm once to get the medoids, designate despots according the medoids, and then feed it into the simulation. This is because each time we run the algorithm, it may give us a different set of medoids. We may need to run the algorithm for each iteration of the simulation. But we usually needs hundreds of thousands of iterations of simulation, which easily translates to hours of computation.

Solution: for each value of K, I run the K-medoids algorithm on the graph 100 times and observe the results to get a sense of the consistency of the solutions. If all 100 times return the same set of medoids, then I can safely assume that the algorithm is consistent and therefore, need only run the algorithm once to create the ambulance allocation without having to rerun the algorithm each iteration of the simulation.

Below is the result of K-medoids algorithm for different K (the first 3 columns):

K	**Set of medoids chosen**	**Frequency**	**Times used in 1,000 iterations of simulation**
1	{17}	100	1,000
2	{8,28}	100	1,000
3	{3,8,29}	100	1,000
4	{1, 8, 20, 29}	45	450

	{3, 8, 23, 25}	21	210
	{0, 4, 8, 29}	11	110
	{3, 8, 21, 28}	23	230
5	{1, 8, 19, 20, 28}	42	420
	{1, 8, 20, 23, 25}	9	90
	{3, 8, 18, 25, 29}	5	50
	{1, 8, 18, 20, 29}	5	50
	{0, 4, 8, 19, 28}	12	120
	{1, 8, 20, 23, 27}	5	50
	{3, 6, 8, 21, 28}	3	30
	{3, 8, 21, 25, 30}	9	90
	{0, 8, 15, 23, 25}	6	60
	{1, 8, 17, 29, 33}	2	20
	{0, 4, 8, 18, 29}	1	10
	{3, 8, 19, 25, 29}	1	10

We see that for K=1,2,3, the solutions are consistent, but not so for K=4,5. As I will run the simulation 1,000 for each value of K, I will use the set of medoids with relative frequency (the last column). For example, in 1,000 iterations for strategy K=1, I will use the medoid set {17} for all 1,000 iterations. For strategy K=4, I will use {1, 8, 20, 29} for 450 iterations, then switch to {3, 8, 23, 25} for 210 iterations, and so on. This way, I don't have to rerun the expensive K-medoids algorithm

Parameters

From the above sections, a list of all the parameters in this model is

- M : the number of ambulances. In this project, it's 100.
- k : the number of despots. In the naive strategy, $k = M$. In the k-medoids clustering strategy, k is the k in $k-$medoids.
- μ_0, σ_0 : parameters of the normal from which the maximum waiting time of each patient comes from.

The following could have been parameterized as well, but in the scope and the purpose of this project, they are not:

- α : coefficient used in the mean of the normal distribution that generates the random term in the time it takes an ambulance to travel from the despot to the location of emergency. In this project, $\alpha = 0.2$
- σ_1 : standard deviation of the normal distribution that generates the random term in the time it takes an ambulance to travel from the despot to the location of emergency. In this project: $\sigma_1 = 1$
- σ_2 : standard deviation of the normal distribution that models the time it takes to attend a case after the ambulance has arrived. In this project: $\sigma_2 = 1$
- $v_{ambulance}$: the average speed of ambulance. In this project, $v_{ambulance} = 30$ (mph)

Assumptions

Below is a list of the assumptions this model makes:

1. Any two requests are independent from each other, and requests from two different neighborhoods arise independently. Normally, this is an appropriate assumption, but it would not if, for example the requests arise from an epidemic, which certainly creates some dependency among those infected.
2. Requests do not depend on the time in day. This is somewhat unrealistic, because probably there are more emergency cases at night than at day. We assume this for the sake of simplicity.
3. A request being sent is independent of the allocation of the ambulance, and which ambulances have been sent to answer other requests. This is an appropriate assumption, because people, when calling for emergency assistance, do not generally have the knowledge of the distribution of the city ambulances.
4. We also assumed a dispatch strategy (described above). This can be another dimension (along with the strategy on initial allocation of ambulances) to experiment with.
5. We have also assumed various probability distributions on some variables. The choice of the distributions have been justified.
6. There is one call center that can: 1) receive all requests, 2) process all requests, and 3) control and direct how the ambulances in the entire city are dispatched. This is an appropriate assumption, as it is how it is operated in real-life.
7. The average speed of an ambulance is 30mph in all neighborhoods. This is somewhat unrealistic because, probably due to traffic or road settings, some neighborhoods are inherently amenable for faster travelling than others. But to make the model simple, we still assume this.
8. We have assumed that the difficulty level of case is independent of the maximum waiting time. This may or may not be true.

Results

The distributions of death rates

Strategy	Mean	95% confidence interval
Naive	0.236	[0.183; 0.269]
k=1	0.016	[0.005 ; 0.028]
k=2	0.148	[0.103 ; 0.171]
k=3	0.105	[0.068 ; 0.128]
k=4	0.129	[0.069 ; 0.191]
k=5	0.145	[0.088 ; 0.186]

Table 1. Summary statistics for death rates of different strategies.

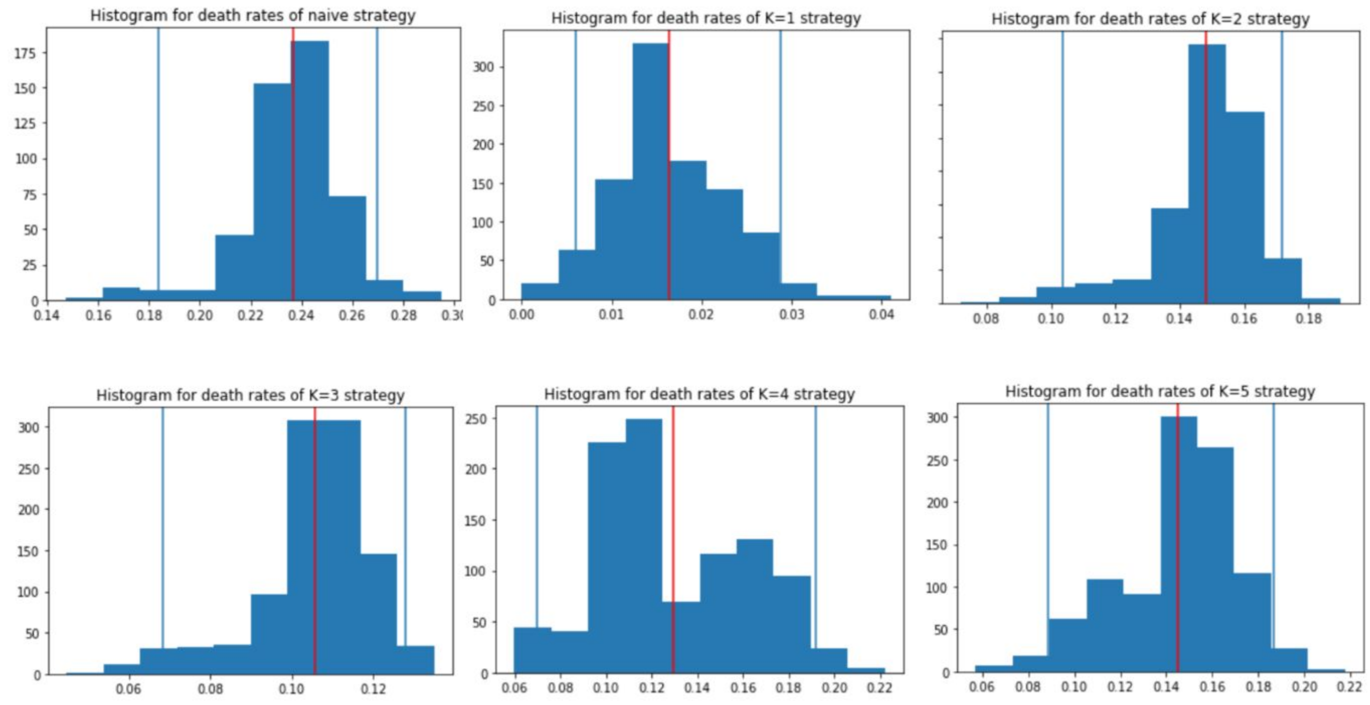


Figure 1. Histograms for death rates of six strategies. The red vertical lines indicate the mean, and the pairs of blue vertical lines draw the 95% confidence intervals.

The distributions of average waiting times

Strategy	Mean	95% confidence interval
Naive	9.036	[8.521 ; 10.277]
k=1	7.485	[6.606 ; 7.760]
k=2	8.185	[7.120 ; 8.570]
k=3	7.896	[6.747 ; 8.252]
k=4	8.033	[6.987 ; 8.501]
k=5	8.219	[7.061 ; 9.250]

Table 2. Summary statistics for average waiting times of different strategies.

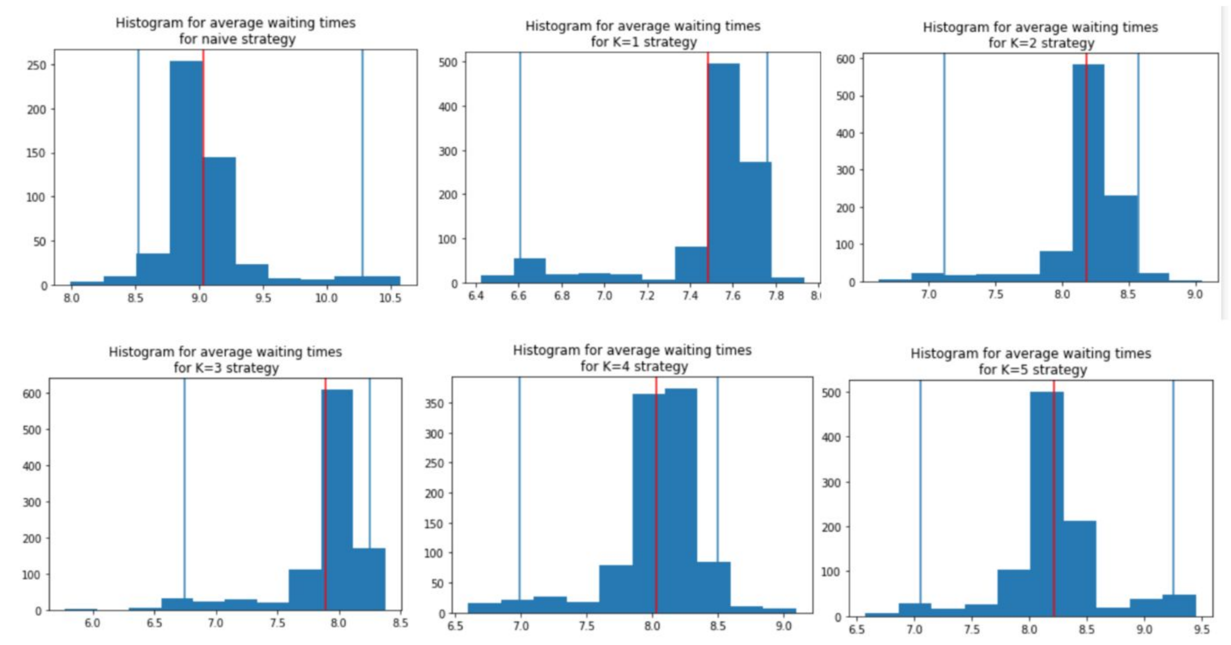


Figure 2. Histograms for average waiting times of six strategies. The red vertical lines indicate the mean, and the pairs of blue vertical lines draw the 95% confidence intervals.

Aggregate result for K-medoids strategies

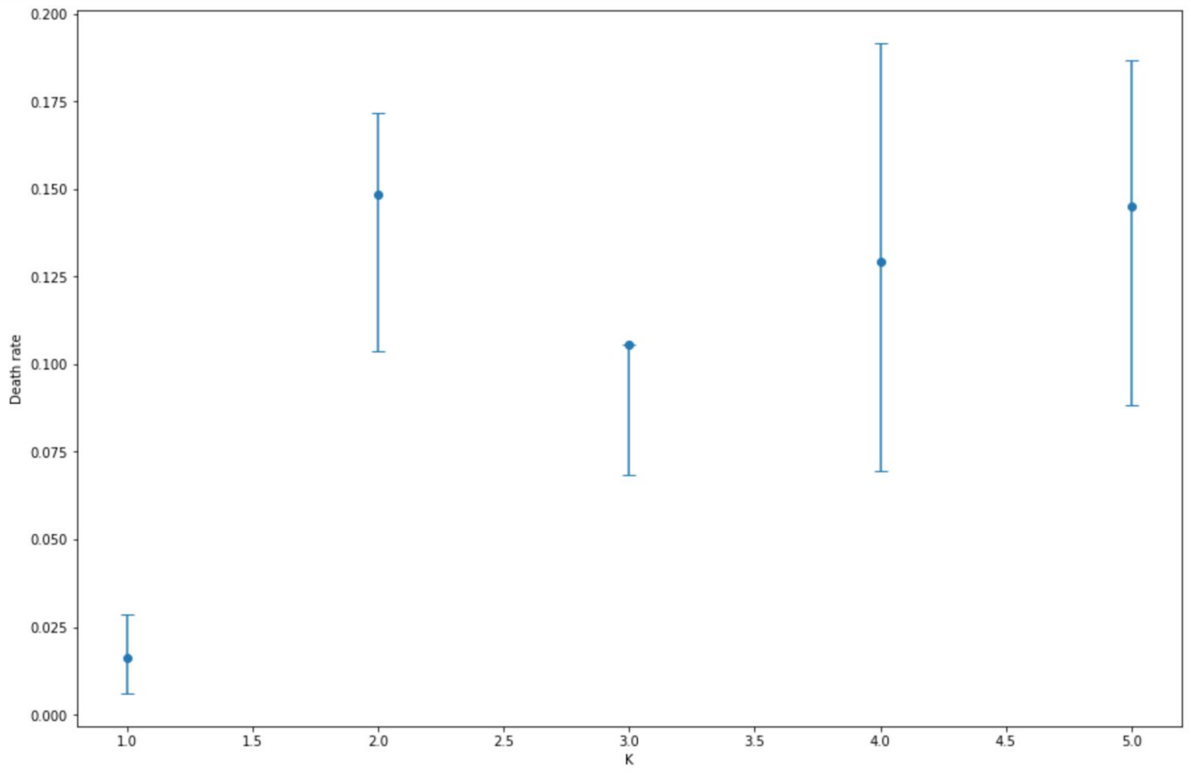


Figure 3. Error bar plot for death rates of 5 K-medoids strategies

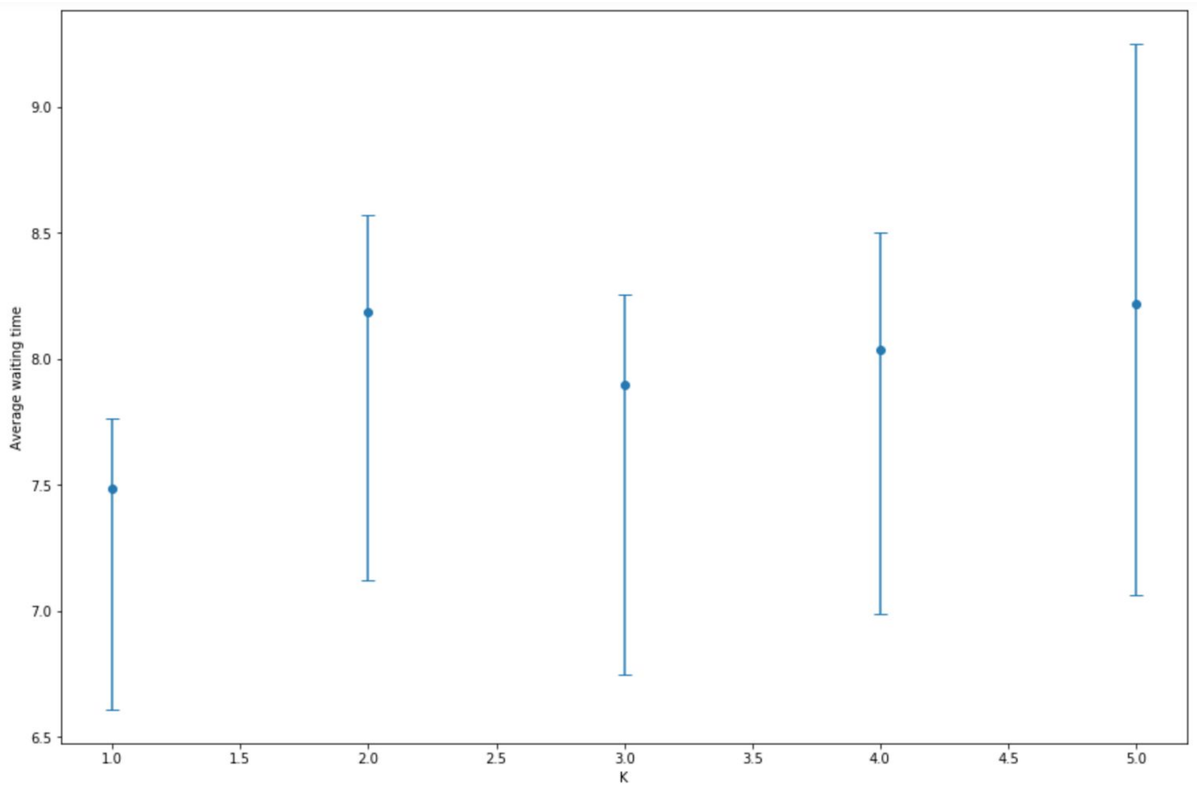


Figure 4. Error bar plot for average waiting times of 5 K-medoids strategies

Correlation between death rates and average waiting times

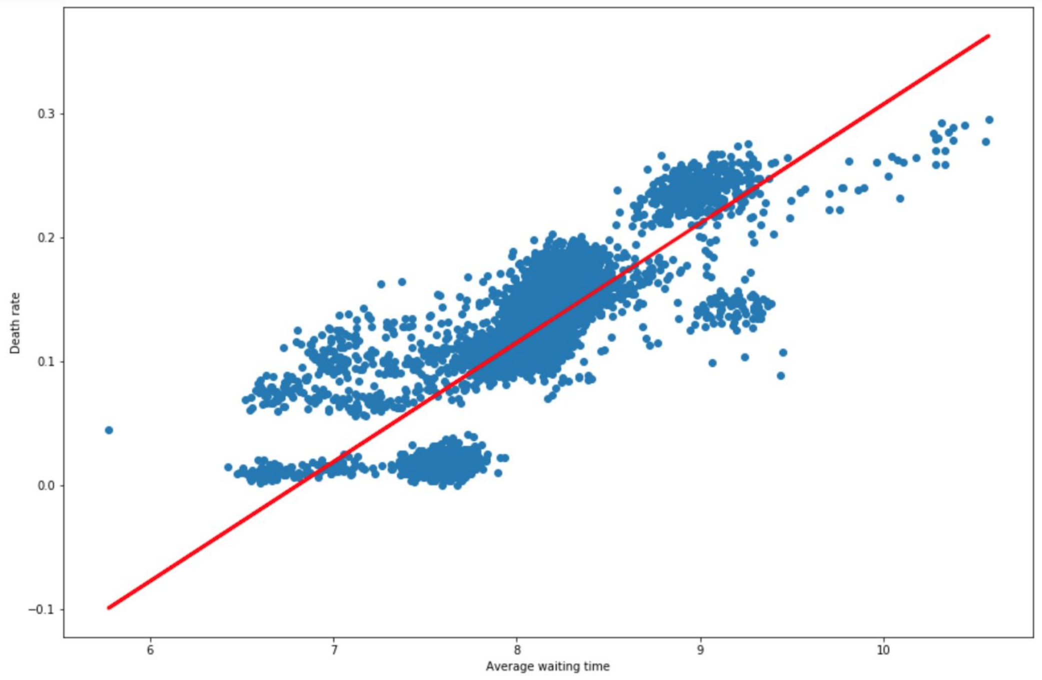


Figure 5. Scatter plot for average waiting time and death rate. The red line is the best fit line. The slope of the line is positive, 0.096

Estimation of how many more simulations needed to reduce the widths of confidence intervals

As a general rule in Monte Carlo simulations, the certainty of the results scale with $\sqrt{n}$, with n being the number of simulations. That said, if we run 100 times more (i.e., 100,000 iterations), then the widths of the confidence intervals are expected to decrease by $\sqrt{100} = 10$ times, but that's no easy task as it's a lot of iterations.

Interpret the results

1. We saw that the naive strategy results in the highest death rates (with mean around 23.7%), and the largest average waiting times (with mean 9.03). This means that on average, this is the worst strategy, with patients having to wait long and death rates are high. This makes sense, considering that the strategy does not make use of any relevant information given, such as the different rates of emergency in different neighborhoods.
2. Among the K-medoid strategies, K=1 results in the best results, with the mean of death rate being extremely low (around 1.6%) and the mean of average waiting times is also the lowest (around 7.5).
3. We also saw that with K=1, the certainty around our estimates are the largest (the confidence interval for the death rate, for example, is significantly narrower than any other K values.) The uncertainty surrounding the average waiting times for K=1 is also the smallest. This unanimously suggests that K=1 is the best option.
4. The result that K=1 is the best option is counter-intuitive and poses a potential problem in our analysis: with K=1, all ambulances are placed into node 17 because it is the despot, and note that the process of determining the despot(s) does not depend on the local rates of emergency but on the weights of edges between the nodes. In other words, the strategy with K=1 does not make use of any information on the different rate of emergency, while we used the same argument to argue why the naive strategy did not work so well. There may be two possible explanations:
    a. K=1 may still not be an optimal value for K because we have not explored values K>5. In fact, figure 3 and 4 showed that there is a fluctuation in the means of death rates and average waiting time as K goes from 2 to 3 to 4 to 5, which suggests a complex synergy existing between K and different components of the model and probably the means of death rates will hit another low point for some large value of K. I did not try strategies with higher K due to limited computational resources and the long time it would take to do so.

   b. The solutions for K=4 and K=5 using K-medoids greedy algorithm are highly inconsistent and we had to use multiple solutions in our simulation. This means we were highly likely to use a lot of sup-optimal solutions for K=4 and K=5. If we had a better algorithm to cluster, the results may have been better.

5. There is a positive correlation between the time a patient has to wait for an ambulance to come and the death rates. Because this speaks to reality (long waits mean the long time the emergency is left unattended and hence more deaths), this adds an extra layer to verify that our model is a good one that is able to model real-life situations.

Course of action

Of the six different strategies we tried, with 100 ambulances, the strategy that places all the ambulance in neighborhood number 17 results in the lowest death rate and smallest average waiting time for patients. We are 95% confident that the death rate is somewhere between 0.5% and 2.8%. If this is something tolerable in your policy, then we can implement this strategy for the city. If you are looking to even further lower the death rate, then there are still other strategies to try, such as strategies with a higher number of despots (increasing K). Our simulation results also suggest that, looking to decrease the waiting time may also be the same as decreasing the death rates.