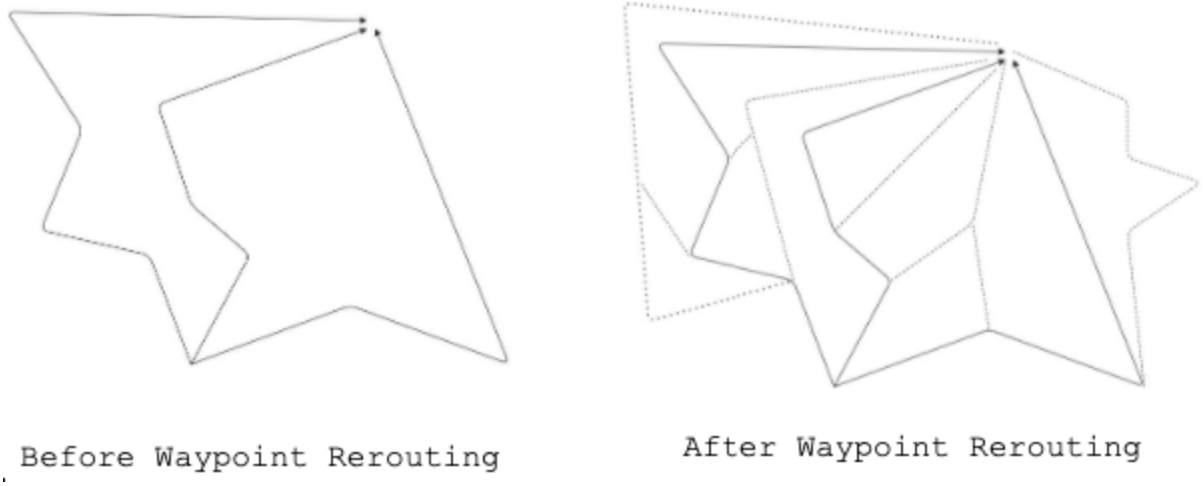
Safety Algorithm

Routing – Waypoint Rerouting

The traditional problem with routing is that it is difficult to provide a complete solution because routing engine services only provide the best paths (time-wise) and this makes it miss the route which might have been the safest. Using the Google API, for example, we get only 3 routes maximum. To get all possible routes between the origin and the destination, we use a recursive algorithm – we reroute on waypoints. This means that on a given route, we try to find new directions on every turn. So, when one route would have just had us take a left on a turn, query the routing engine for new directions from that turn's location again and see if it considers going straight and turning right as possibilities too. If yes, we add those to our list of routes. This leads to an exponential rise in the number of routes.



Before Waypoint Rerouting

After Waypoint Rerouting

Population Density Algorithm

Population Density algorithm uses data from EU's GHSL calculations which estimates population density based on satellite imaging and India's census surveys. This data gives us a high resolution which yields data for tiles as small as 0.07 km^2.

Along each route, we evaluate every point on the road and assign it a population density danger rating. It is notoriously difficult to understand how danger rises with increase in population density, so we instead started mapping how change of danger increases with increase in population density. A bell curve perfectly describes how danger changes with increase in population density – the increase is not high at start, it keeps increasing till a high point and then, after the population density becomes extremely high, it becomes so saturated that increase in number of people has no effect on danger.

Let danger be represented by 'D' and population density be represented by 'P'

$$\frac{dD}{dP} = \frac{K}{a \times \sqrt{2\pi}} e^{-(P-b)^2/2a^2}$$

where $K = 4500000000,$

$a = 18800,$

$b = 30000$

$$\Rightarrow \int dD = \int \frac{K}{a \times \sqrt{2\pi}} e^{-(P-b)^2/2a^2} \cdot dP$$

$$\therefore D = 2.25 \cdot erf(\,(P - 3000)/(2000\sqrt{2})\,)$$

After normalisation to 0 *to* 1 *range,*

$$D = (2.25 \cdot erf(\,(P - 3000)/(2000\sqrt{2})\,) + 1.949)/3.1795, \ \forall\, P \in R$$

We input the population density at a point in this formula to get the associated danger with it scaled between 0 and 1. After obtaining ratings for individual points, we combine the ratings of all points along the route to achieve the final rating. Refer to the section on *Rating Combination and Addition* for reading more about the process.

Covid Points Algorithm

This algorithm uses district-wise data for COVID cases made public by the government to keep people away from infested areas. We use a simple equation for calculating the district's threat.

$$District's\ covid\ threat\ (DCT) = total\ district\ cases\ /\ district\ population$$

Each point along the route is assigned a rating depending on the district they are in. But, because of the increasing number of COVID cases, it will become difficult for this algorithm to remain functional on an absolute scale. To make sure it keeps functioning, we normalise the DCT by the highest DCT of any point along the route. This makes Covid Points a comparative algorithm rather than an absolute one.

$$Point's\ Rating = DCT/DCT_{max}$$

After obtaining ratings for individual points, we combine the ratings of all points along the route to achieve the final rating. Refer to the section on *Rating Combination and Addition* for reading more about the process.

Point Avoid Algorithm

Thousands of coordinates across India describing the location of hospitals, public transport spots, non-sanitary areas, etc. are taken and the algorithm drives the user away from these hotspots. We assign all hotspots a weightage according to how dangerous they are (the more the danger, the higher the weightage). To evaluate one point along the route, we first sum up the danger effect on that point from every hotspot in the vicinity and then, we fit the answer to a logistic model to scale answers from 0 to 1. Mathematically,

$$Danger\ Effect\ (DE)\ =\ Weightage\ (W)\ /\ distance\ between\ point\ and\ hotspot\ (r)^2$$

$$Danger\ Effect\ Sum\ (DES)\ =\ \sum_{i=1}^{n} DE_i$$

$$Finally,\ Danger\ Rating\ =\ 1/(1\ +\ e^{-0.02(DES-150)})$$

After obtaining ratings for individual points, we combine the ratings of all points along the route to achieve the final rating. Refer to the section on *Rating Combination and Addition* for reading more about the process.

To increase time efficiency of the algorithm, we geohash all locations; this means that we divide India into small area-wise sectors and assign codes to locations based on what sector, sub-sector, sub-sub-sector (and so on) they are in. Only those locations which are in the vicinity of a point are accessed for calculations. Also, to increase iteration speeds through these geohash codes, we arrange them into a dictionary and train the computer to read these hashes like a human would read a dictionary.

Containment Zones Algorithm

Containment zones fetches data from government sites to analyse which areas have been declared as a containment zone. The algorithm identifies if a path is inside a containment zone. If yes, it eliminates that route from consideration.

Using ray casting, we find if a point lies in a polygon that demarcates the containment zone. To do this, we keep track of how many intersections are there between a ray casted from our point and the lines of each polygon. Let this number be I. Here is the logic:

$$if \sum_{n=1}^{N} I_n \ mod \ 2 \ = 1, \ point \ is \ in \ a \ containment \ zone.$$

$Where \ N \ is \ the \ total \ number \ of \ containment \ zones.$

Traffic Algorithm

We continuously update (every 10 minutes) a database that contains data of all the roads of India. We query the data for roads that lie on our route and assign the points on that road the rating for that road. Simply,

$$Point \ Rating \ = \ Road \ Rating \ = \ Jamming \ Factor \ / \ 10$$

Jamming factor (JF) is a factor between 0 and 10 which tells how jammed a road is. After obtaining ratings for individual points, we combine the ratings of all points along the route to achieve the final rating. Refer to the section on *Rating Combination and Addition* for reading more about the process.

Temperature & Humidity Algorithm

The change in temperature and humidity affect the rate of spread of coronavirus or the *R-value* (how many people can be infected through a single infected person). Generally, the principle followed is that as temperature increases and as humidity decreases the rate of transmission of virus also slows down.

Relationship between R-value (reproductive number) and temperature:

$$Rt \ = \ e^{(\gamma - \delta Tc)}$$

T(c)= temperature in Celsius
γ= 0.752038698 (constant) {For value of T (c) between <=92 and >= -20)
δ= 0.01732867958 (constant)

Relationship between R-value (reproductive number) and absolute humidity:

$$Rh \ = \ e^{(\alpha - \beta H)}$$

H=Absolute Humidity
$\alpha \ = \ 0.89323534577778 \ (constant)$ { For the value of H ranging between < 13 and >=5}
$\beta \ = \ 0.051344235555556 (constant)$

$\alpha = -3.6043653412\ (constant)$ {For Value of H ranging between >=13 and <=15.5}
$\beta = +0.2772588724\ (constant)$

These relations are based on the loess models presented in a COVID-transmission pattern study [1] which were modified to be adjusted to Indian spread patterns.

Using the open weather map API we got our input of temperature and Humidity for specific coordinates (this process for API call was made faster through a python API wrapper). From the API we received relative humidity values which were then converted into absolute humidity values through the following equation.

Water Vapour Saturation Pressure

$$Pws = A * (10^{(m*T(Celcius)/(T(celcius)+Tn(constant)})$$

A=6.116441 (constant)
m=7.591386 (constant)
Tn=240.7263(constant)

Water Vapour Pressure

$$Pw = Pws * (Relative\ humidity/100)$$

$$Absolute\ Humidity = 2.16679 * (Pw * 100)/(T)$$

Then, we developed a scale from 0 - 100 where 0 and 100 had specific extreme ranges. The extreme values for temperature were taken as +92 $^{\circ}C$ and -20 $^{\circ}C$and for humidity were <5 and >15.5. In the end, the R final values were converted into scale value through the following equation:

Final Risk Rating (Temperature) = $floor(-16.766047547135 + 38.910505836576 * Rt)$
Final Risk Rating (Humidity) = $floor(-50.0001000002 + 50.0003000006 * Rh)$

In the end, these values were together taken in the ratio (Risk(T): Risk(H) = 2.5:1) as temperature was found as a more dominating factor for change in the Rvalue. With this, we got our output as the cumulative Final Risk Rating on a scale from 0 -100 depending upon temperature and humidity conditions of a coordinate.

Rating Combination and Addition

[1] (2020, March 12). The Role of Environmental Factors on Transmission ... - SSRN. Retrieved September, 2020, from https://www.ssrn.com/abstract=3552677

Four of the algorithms find point rating but need to integrate all point ratings together to get a route rating. Here's how we do that. We plot a frequency distribution using rating brackets of 0.005 on the x-axis. Using a regression model, we make a probability density function from the data we have for all points and find an expected value from it:

$$Expected\ Value\ (E)\ =\ \int_{-\infty}^{\infty} Rating\ \times\ Frequency\ \cdot\ d\ (Rating)$$

Finally, we factor in the distance, DIST, to give higher danger ratings to routes. It has a 30% importance in the combination process.

$$Final\ Danger\ Rating\ (FDR)\ =\ 7E/10\ +\ 3/10\ \cdot\ (\ 1\ -\ e^{q/288}\ )$$
$$Where\ q\ is\ the\ number\ of\ ratings\ in\ the\ route$$

Finally, to get the safety rating, we subtract the Danger Rating from 1:

$$Final\ Safety\ Rating\ =\ 1\ -\ FDR$$

Integrated Algorithm

To understand how much weightage we want to give to each algorithm, we use artificial intelligence. We created a lot of test data where we, backend developers, picked between two routes with arbitrary values for all of the 6 algorithms. As we kept picking between the two routes, the AI system understood how much importance was given to each algorithm and finally outputted a percentage importance for each algorithm.