Using Linear Algebra to Calculate Geospatial Distances

Introduction

As big data analytics become increasingly prevalent in modern day businesses, there is an increasing demand for efficient algorithms for geospatial distance calculations. As ride sharing and logistics companies try to optimize their delivery routes, their data engineers scramble to process large amounts of datasets for insight generation. The problem is that most companies and research institutions tend to depend on Google's Distance Matrix API for distance calculation. The API provides information that pertain to distance, routes, and directions for every pair of latitude and longitude provided to it. However, the API is not free and can cost up to 10 USD for every 1000 distances calculated. This could amount to hundreds of thousands of dollars for companies that have large amounts of data and services to process and manage.

In this paper, I shall propose a simple algorithm for cities with grid plans—a type of city plan in which streets run at right angles to each other, forming a grid. The grid plan is a popular kind of city layout that many cities across the world follow. In fact, many Asian civilizations have used the grid system for their city planning. During the Renaissance Period, the Europeans designed many of their colonies in grid-style, giving many of the major cities in America their iconic design.

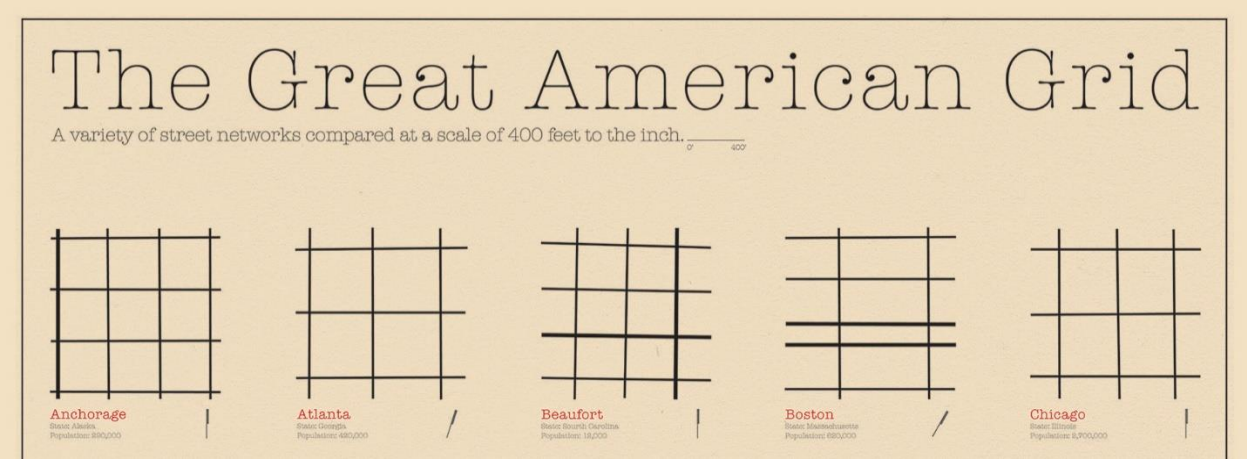
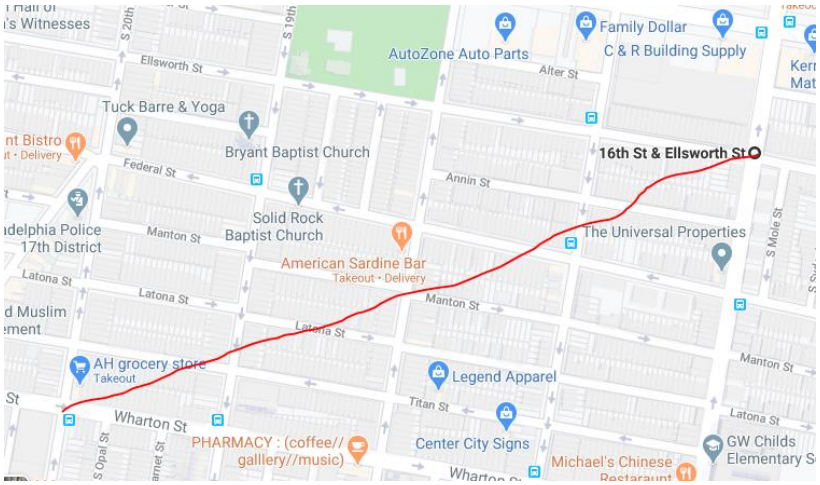


Figure 1:The Great American Grid[1]

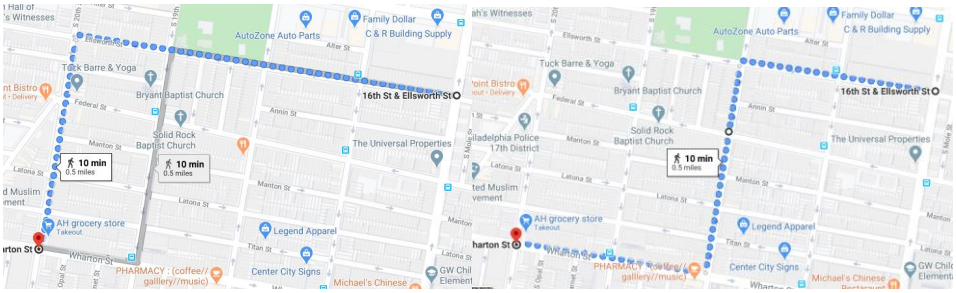
The orthogonal geometry of the city grids makes it ideal to develop a simple algorithm to calculate the distance to get from one place to another with the help of orthogonal vectors. The problem is that most conventional distance calculation algorithms usually give us the Euclidean distance between two points. Euclidean distance is a useful metric for determining the shortest possible path from one point to another.



However, it is not very useful for vehicle or foot travel since we have to abide by the geometry of paths and roads. Hence, it would be a lot more useful to measure the Manhattan distance, also known as the city block distance, between two points. It is defined as the sum of the lengths of the projections of the

[1] *https://upload.wikimedia.org/wikipedia/commons/e/e0/American_Grid_Comparison.jpg*

line segment between the points onto the coordinate axes (defined by the horizontal and vertical paths in a grid system city). If we were to look at the two images below, you would notice that the Manhattan distance appears to be similar regardless of the path taken (as long as the traveler does not take too many unnecessary turns).



This would be a far better estimate of the distance one has to travel to get from one point to another in a grid design city. In fact, this would also be the same distance that a vehicle would have to travel. Hence, it makes a lot of sense to use Manhattan distance for the benefit of practicality in our algorithm.

Another problem is that most conventional algorithms tend to neglect the spherical shape of our planet which gives rise to different distances travelled across the same difference in longitudes at different latitudes.

Distance Between Longitudes:

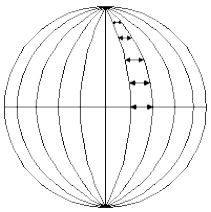


Figure 2: Distance between Longitudes[2]

The solution to this problem is to employ the Haversine formula which accounts for the latitude and longitude during the process of the distance calculation.

[2] https://calgary.rasc.ca/latlong.htm

$$d = 2r \arcsin\left(\sqrt{\operatorname{hav}(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\operatorname{hav}(\lambda_2 - \lambda_1)}\right)$$
$$= 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

d is the distance between the two points along a great circle of the sphere
r is the radius of the sphere.
φ_1, φ_2 are the latitude of point 1 and latitude of point 2 (in radians),
λ_1, λ_2 are the longitude of point 1 and longitude of point 2 (in radians).

However, it is important to note that the haversine formula cannot be guaranteed correct to better than 0.5% because the Earth is not a perfect sphere.

Usage of the algorithm

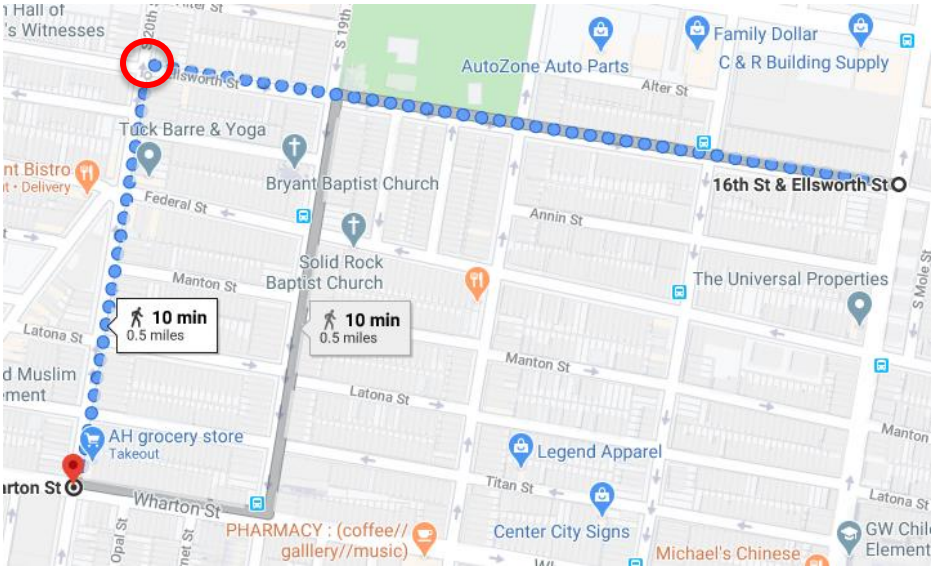
The user would input the addresses of two locations that belong to the same road in the city. The user will then input the addresses of another two locations for the distance calculation. The algorithm will return the Manhattan distance between the two locations. Refer to https://github.com/samueltan97/geo_distance_lin_alg to see the code

Explanation of the algorithm

The initial input will be processed to generate a pair of latitude-longitude coordinates. The differences in their latitudes and longitudes will give us a direction vector that would be parallel to one of the coordinate axes in the city. A 90 degrees rotation matrix will be applied onto the vector to transform it into its orthogonal pair. This new vector would be perpendicular to the original direction vector and parallel to the other coordinate axis in the city. The orthogonal vectors can be stored for future distance calculations within the city.

After finding the two orthogonal vectors that form the basis for the grid of the city, the next step is to find the “mid-point” of the Manhattan distance between the two coordinates. It is not the halfway mark

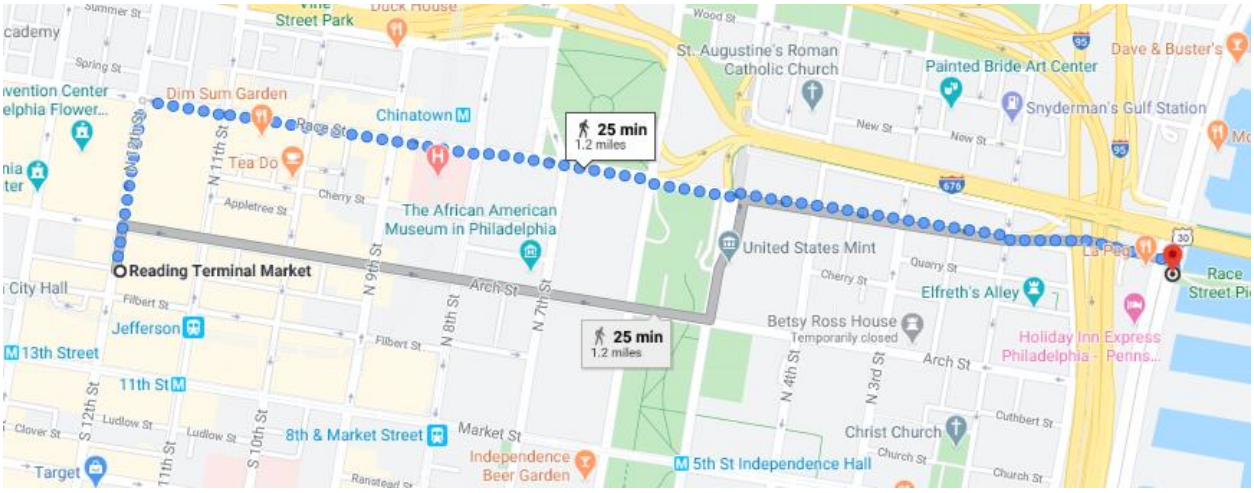
for the distance that a person must travel but the end of the path that one has to travel along one of the orthogonal vectors before taking a turn and travelling in the perpendicular direction to reach the destination. It is circled in red in the image below.

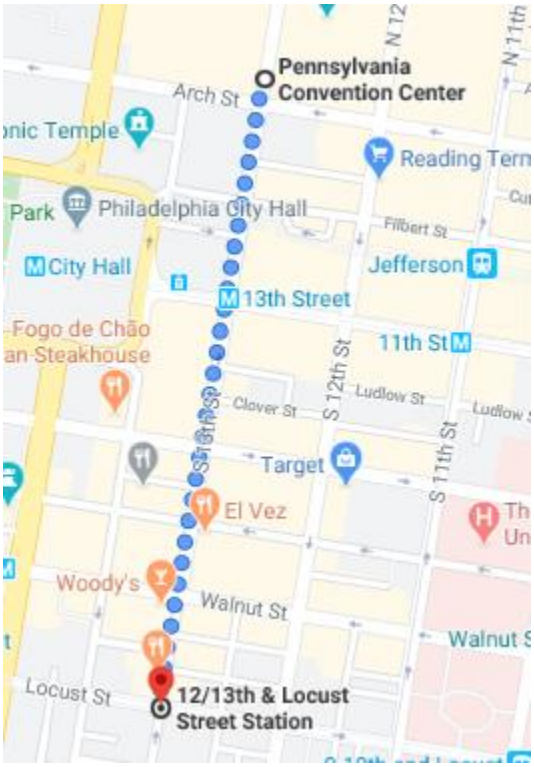


To find the "mid-point", we need to build an augmented matrix. The two orthogonal vectors would form the coefficient portion of the augmented matrix. The differences in latitude and longitude between the start and destination would form the augmented column of the augmented matrix. By solving this augmented matrix, we are able to get the coefficients for the two orthogonal vectors that would form the two orthogonal paths that would give us the Manhattan distance. We can get our "mid-point" by adding one of the weighted orthogonal vector to the coordinates of the start point. By applying the Haversine formula twice—once on the distance between the start point and the mid-point and the second time on the mid-point and the destination—we are able to calculate the Manhattan distance between two points.

Example in Philadelphia

Let us try to find the distance between Reading Terminal Market and Race Street Pier. First, we need to provide the algorithm with two locations (Pennsylvania Convention Center and the 12/13th & Locust Street Station) that are on the same road.





	Latitude	Longitude
Pennsylvania Convention Center	39.9543931	-75.1609668
12/13th & Locust Street Station	39.947944	-75.162363

Reading Terminal Market	39.9533113	-75.15943
Race Street Pier	39.9556634	-75.1504502

To develop the first orthogonal vector, we first try to find the unit vector that lies in the direction from Pennsylvania Convention Center to the 12/13th & Locust Street Station. Following which, we rotate the vector by 90 degrees to get the orthogonal pair before building a coefficient matrix, A, with the two orthogonal vectors

$$\Delta longitude = -75.162363 - (-75.1609668) = -0.0013962$$

$$\Delta latitude = 39.947944 - 39.9543931 = -0.0064491$$

$$\mathbf{a} = \begin{bmatrix} \Delta longitude \\ \Delta latitude \end{bmatrix} = \begin{bmatrix} -0.0013962 \\ -0.0064491 \end{bmatrix}$$

$$|\mathbf{a}| = (\Delta longitude^2 + \Delta latitude^2)^{\frac{1}{2}} = 0.00659850477$$

$$\hat{\mathbf{a}} = \frac{\mathbf{a}}{|\mathbf{a}|} = \frac{1}{0.00659850477} \begin{bmatrix} -0.0013962 \\ -0.0064491 \end{bmatrix} = \begin{bmatrix} -0.21159339090637744 \\ -0.9773577834778525 \end{bmatrix}$$

$$\hat{\mathbf{a}} = \frac{\mathbf{a}}{|\mathbf{a}|} = \frac{1}{0.00659850477} \begin{bmatrix} -0.0013962 \\ -0.0064491 \end{bmatrix} = \begin{bmatrix} -0.21159339090637744 \\ -0.9773577834778525 \end{bmatrix}$$

$$\hat{\mathbf{b}} = \begin{bmatrix} cos\frac{\pi}{2} & -sin\frac{\pi}{2} \\ sin\frac{\pi}{2} & cos\frac{\pi}{2} \end{bmatrix} \hat{\mathbf{a}} = \begin{bmatrix} cos\frac{\pi}{2} & -sin\frac{\pi}{2} \\ sin\frac{\pi}{2} & cos\frac{\pi}{2} \end{bmatrix} \begin{bmatrix} -0.21159339090637744 \\ -0.9773577834778525 \end{bmatrix} = \begin{bmatrix} 0.9773577834778525 \\ -0.21159339090637744 \end{bmatrix}$$

$$A = \begin{bmatrix} \hat{\mathbf{a}} & \hat{\mathbf{b}} \end{bmatrix} = \begin{bmatrix} -0.21159339090637744 & 0.9773577834778525 \\ -0.9773577834778525 & -0.21159339090637744 \end{bmatrix}$$

Next, we need to calculate the actual difference in latitude and longitude between the start and end points. The Haversine formula requires the latitudes and longitudes to be converted into radians.

	Latitude (in radians)	**Longitude (in radians)**
Reading Terminal Market	0.697316829259256	-1.3117795174222018
Race Street Pier	0.6973139773912581	-1.311419482177454

$$\Delta longitude = -1.311419482177454 - (-1.3117795174222018) = 0.00036003524$$

$$\Delta latitude = 0.6973139773912581 - 0.697316829259256 = -0.00000285186$$

$$\mathbf{d} = \begin{bmatrix} \Delta longitude \\ \Delta latitude \end{bmatrix} = \begin{bmatrix} 0.00036003524 \\ -0.00000285186 \end{bmatrix}$$

$$A\mathbf{x} = \mathbf{d}$$

$$\begin{bmatrix} -0.21159339090637744 & 0.9773577834778525 \\ -0.9773577834778525 & -0.21159339090637744 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 0.00036003524 \\ -0.00000285186 \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} -0.00007339378289680806 \\ 0.0003524866852006069 \end{bmatrix}$$

Now that we have the weights for the different orthogonal vectors, we will be able to construct the "mid-point" by adding one of the orthogonal vectors (multiplied by its weight) to the coordinates of the Reading Terminal Market. After constructing the "mid-point", we should be able to find the Manhattan distance between Reading Terminal Market (marked by S) and Race Street Pier (marked by E). The distance calculated by Google is approximately 1.3 miles. My algorithm gives us 1.37 miles which is within 0.5% error.

$$\mathbf{S} = \begin{bmatrix} -1.3117795174222018 \\ 0.697316829259256 \end{bmatrix}$$

$$\mathbf{E} = \begin{bmatrix} -1.311419482177454 \\ 0.6973139773912581 \end{bmatrix}$$

$$\mathbf{M} = \mathbf{S} + -0.00007339378289680806 \begin{bmatrix} -0.21159339090637744 \\ -0.9773577834778525 \end{bmatrix}$$

$$Distance = \mathbf{haversine}(\mathbf{S}, \mathbf{M}) + \mathbf{haversine}(\mathbf{M}, \mathbf{E}) = 1.37 \approx 1.3$$

Experiments

	Calculated (miles)	**Actual (miles)**
New York City: Carnegie Hall to Brick House Statue	2.0	1.9

Los Angeles: Broadway Hollywood Building to Paramount Pictures Studio	1.5	1.6
Xi'an, China: Fengcheng Balukou to Chongqing Bazhiyuan Hot Pot	4.3	3.8
St. Petersburg, Russia: Suzdal'skiy Prospekt to Grazhdanskiy Prospekt	4.7	4.8
Bueno Aires: Universidad De General Sarmiento to Gaspar Campos 4849-4801, B1665LVN José C. Paz, Buenos Aires, Argentina	3.6	3.3

Conclusion

It turns out that the algorithm does not work perfectly on every city that "proclaims" to have a grid plan. One of the possible reasons is that their roads are not perfectly perpendicular. Another possible reason is that not every part of the city follows the grid plan. This is especially true for cities which have done a lot of renovation to their roads. Another likely reason is that Google Maps might propose a horizontal path that is traveled at a different latitude than what the algorithm has artificially constructed. This might account for the slight discrepancy in the distance calculation. Nonetheless, the distance calculation is still fairly accurate for cities with roads which are very perpendicular to each other. Future possible improvements include using stored pairs of orthogonal vectors for different parts of the city to account for the slight angular changes. Another possible improvement is to get the route that travels through every part of the city stored in a database before integrating every single Haversine distance using

the basis vectors to form the coordinates that lie on the route. This would expand the usage of the algorithm beyond grid plan cities