Optimizing Walking Paths Based on Interestingness

Felipe Ducau
Center for Data Science
New York University
Email: fnd212@nyu.edu

Maria Elena Villalobos Ponte
Center for Data Science
New York University
Email: mvp291@nyu.edu

Sebastian Brarda
Center for Data Science
New York University
Email: sb5518@nyu.edu

***Abstract*—When providing walking directions to a destination, web mapping services usually suggest the shortest/fastest route (in terms of distance and/or time). The goal of this work is to automatically suggest alternative more enjoyable routes, which might take marginally more time but go through spots that would be more interesting to the user. In order to do that, we define a scoring function that weights each path attractiveness based on social media data. Afterwards, we propose two graph based optimization algorithms, create and evaluate a prototype system.**

***Keywords*—Maps, Enjoyable paths, Graph search algorithms.**

I. Introduction

Efficiency has become a cult in our society. We are always striving to do things faster, quicker, and with the lower cost. This is even more pronounced within the engineering and scientific community.

In this context, popular GPS and mapping systems such as Google Maps^{TM}, Apple Maps Connect^{TM}and Waze^{TM}, are not the exception. When providing directions to a place, these systems suggest the shortest route in terms of distance and/or time. However, the fastest route is not always the most enjoyable, or the one that maximizes utility for users. Furthermore, it is not always the one that connects us with nature, makes us meet new people, or help us discover new places.

In this project we create a prototype system to automatically suggest alternative routes to the shortest one, which might take marginally more time, but go through more attractive spots for the user. The main application (although not the only one) would be for tourists visiting the city. Users that are exploring the city eventually need to go from point A to point B, but they do not necessarily want this trip to be optimized with respect to time. Instead they probably want to maximize their experience while walking. To meet this goal we define a new notion of *attractiveness* based on social media data. This new metric along with an efficient implementation of graph based routing algorithms allows us to create path suggestions in real time.

To make the information flow into the system automatic and scalable, we resort in social media information. That would allow for easy implementation and replication across different cities. In section II we explain what data we used, how we gathered it and the pre-processing task. In section III discuss the information needed to define how interesting a block of the city is and define the notion of *interestingness* which can be tuned according to the needs of the user. Later in section IV, we propose two graph optimization algorithms to create alternative paths that maximize the interestigness metric. Finally, we create our prototype system for New York City in section V and evaluate the results on section VI.

A. Related Work

The Good City Life organization^{TM}has worked on the problem of associating human senses and perception to urban concepts. They have estimated sensory layers of the city based on human perception and social media data. They have used crowd-sourcing to score images of cities for the notion of beautiful, quiet and happy in [1] and [2]. They have also proposed a way to use these scores to generate emotionally pleasant routes as alternatives to the shortest path [3].

II. Data extraction and Prepossessing

A. Data extraction

In order to tackle this problem, we required both the (1) graph structure for New York City and (2) some way to compute the "attractiveness" or "interestingness" of a place.

For (1) we used data from OpenStreetMap [4] which contains the coordinates for each corner of each block in Manhattan. From this data we constructed a graph consisting 8,828 nodes, with their respective latitude and longitude, and 12,036 edges.

For (2) we used social media data from Flickr^{TM}. Part of these data was obtained through the Flick API. It consisted of the metadata of each Flickr publication and contained the following fields: *n_comments, n_views, n_tags, title, photo_id, user_account_location, neighbourhood, date, user_id, latitude, longitude, description*. We downloaded metadata for 216,535 random photos taken between 2015 and 2016 in Manhattan.

Unfortunately, the number of *"likes"* for each photo was not present in the metadata, so we scraped it with BeautifulSoup [5] package. We also scraped the static URL for each photo in order to build our prototype UI.

B. Data pre-processing

After downloading the whole metadata and joining the scraped data by *photo_id*, we proceeded to preprocessing. We created a graph with the corners of the city as nodes and its connections as edges, each edge representing a block or *path unit*. Fortunately, these edges contained as well several paths inside green areas such as Central Park.

The next step was to map each photo to its respective nearest edge in the graph. We used a shortest distance algorithm from GeoPandas and assigned the right *edge_id* to every photo metadata. After that mapping we were ready to compute aggregated metrics for each edge of the city. We then computed the number of likes, comments, photos and views for each edge of the graph.

Later, we looked at the "accounts origin" entry in the pictures metadata, meaning the location indicated in the Flickr profile of the users in order to differentiate new yorkers (locals) from tourists. We re-computed aggregate metrics both for "tourists" and "locals". Since the number of tourist users was clearly higher than the number of locals, we normalized all the local metrics by the ratio of $\frac{n\_photos\_tourists}{n\_photos\_locals}$. Finally, for each edge we kept the static URL of the photo of that edge with the highest amount of total views. The result was a table with aggregated metrics consisting of 10 columns: *edge_id*, *url_static*, *likes_tourist*, *comments_tourist*, *views_tourist*, *n_photos_tourist*, *likes_local*, *comments_local*, *views_local*, *n_photos_local*.

III. Defining a function for Interestingness

This was one of the most difficult tasks of the project, since it is completely subjective how interesting or attractive a place or path is. Our main assumption is that people take pictures of the places they like the most. In other words, the number of pictures that a certain place has in Flickr would highly correlate with how attractive or interesting the place is. This is the same criteria that the creators of SightsMAP™ [6] used.

We decided to consider 2 main user defined dimensions to determine the level of interestingness of a certain edge: how local vs. touristic a place is and how unique vs. popular. Each of these parameters would be in the $[0, 1]$ interval. For the first parameter, 1 would mean completely touristic, and 0 completely local. For the second parameter, 1 would be completely popular and 0 would be completely unique.

How local vs. touristic was relatively easy to compute, since in our data we could differentiate photos taken by tourists from photos taken by locals. So we basically took a linear combination of the *likes*, *n_photos*, *n_comments* and *n_views* based on the user parameters. For example if the parameter was 0.3 we would compute $likes = 0.3 \cdot likes\_tourists + 0.7 \cdot likes\_locals$

For how unique vs. popular we created an arbitrary score. We supposed that if a certain place was popular, then the number of photos taken of that place would correlate linearly with the level of popularity. A unique place on the contrary, is an interesting place that people might not know a priori. So we thought that if a place had a small amount of photos, but users who watched the photo in Flickr after it was posted liked it, then the place was not very popular but very interesting. At first we tried with $\text{UNIQUENESS} = \frac{n\_comments\ +\ n\_likes}{n\_photos}$. The results made complete sense, after manually reviewing the photos and paths, but the magnitude of the uniqueness metric was too smooth across edges. So we finally decided to penalize even further photos with high popularity and defined $\text{UNIQUENESS} = \frac{n\_comments\ +\ n\_likes}{n\_photos^2}$ which spiked the distribution of the metric accross edges further.

After setting the values for uniqueness and popularity, both were normalized in order to sum up to 1 across the whole graph. Later a linear combination was taken between both based on user parameter. For example if user parameter for popularity was 0.6, then we would define total interestigness by edge as: $\text{INTERESTINGNESS} = 0.6 * popularity + 0.4 * uniqueness$

The final result after user parameter input, is a vector that maps each edge_id to a "interestingness score" which sums to 1 across the whole graph. The interestingness weight of each edge will vary depending on the parameters provided by user.

IV. Graph Algorithms for Path optimization

In this section we study two different alternatives to build paths between a source point in the city A to a target point B based on a given interestingness weight of the edges in a graph. For this purpose we set an additional user defined parameter: the maximum additional percent time that the alternative path can take compared to the shortest path.

There are two main characteristics that we take into consideration for these algorithms: (1) we need them extensively explore several possible paths; and (2) we need them to be fast enough to create alternative paths in real time so that they could be used in a production system.

A. Randomized Dijkstra Algorithm

We first look at the implementation in [7]. In their work they propose to compute the K shortest paths between source and destination, and them choose the one that maximizes their objective function. When we tried this method on the graph of Manhattan we found out that because of the layout of the city, the top 500 paths are very similar, only differing in a couple of edges. This means that if we wanted to have diversity in the potential paths to consider we would have to make K very large. As a consequence, it becomes impractical to be applied in a real-time setting because of computational cost.

Following this line of thinking, but with the idea of introducing variance[1] between the different generated paths we propose the following algorithm that we refer as *Randomized Dijkstra* which works as follows: (1) First we randomly sample a percentage m of the total number of edges $|E|$ in the graph G according to the probability distribution of edge removal $p_r(e)$. (2) Then we remove the sampled edges from the G and (3) we compute the shortest path between source and destination. (4) Finally we add back the removed nodes and (5) repeat this procedure K times.

This algorithm allows us to get diversity of paths while keeping the total running time short. Since in each iteration of the algorithm we are removing some edges of the graph randomly, it is very unlikely that the resulting path will be equal to the shortest path unless the percentage of removed edges was very small. Experimentally we obtained good results by setting the percentage of nodes to be removed $m = 0.15$. For the probability of edge removal we used $p_r(e) = \log(\text{interestingness}(e))$, where the interestingness function is defined in section III and $K = 30$. The total running

[1]This idea is in essence similar to the concept of variance creation introduced by Machine Learning algorithms like Random Forests or Boosted Trees

time is $O(N \cdot |V|^2)$, where $|V|$ is the total number of edges of the graph.

Once we have K candidate paths $P_1, \ldots, P_K$ from our algorithm, we first filter them according to our time constraint. Later we score them by the sum of the interestingness weights of the edges that compose each path and keep the top I alternative paths.

$$\text{Interestingness(P)} = \sum_{i=0}^{N} \text{interestingness}(e_i) \tag{1}$$

where the path is $P = (e_1, \ldots, e_N)$.

B. $A*$ Algorithm

As a second approach to solve our problem we consider the $A*$ search algorithm by defining an heuristic function that reflects our dual optimization problem (maintaining a direct path between source and destination while trying to maximize the interestingness of the traversed edges).

$A*$ is a best-first search algorithm, meaning that it computes its solution by searching among all possible paths to the target for the one that minimizes a cost function. Among these paths it first considers the ones that appear to lead faster to the solution. At each iteration of its main loop, $A*$ needs to determine which of its partial paths to expand into one or more longer paths in a greedy way. It does so based on an estimate of the cost (total weight) to the final node. Specifically, $A*$ selects the path that minimizes $f(n,t) = g(n) + h(n,t)$, where n is the last node on the path, $g(n)$ is the cost of the path from the start node to n, and $h(n,t)$ is a heuristic that estimates the cost of the cheapest path from n to the target node t.

To adapt this framework to the problem under consideration we define the following heuristic function:

$$h(n,t) = d_e(n,t) - \lambda\phi(n) \tag{2}$$

where $d_e(n,t)$ is the euclidean distance between source and destination, $\phi(n)$ is the *node potential*, defined as the sum of the interestingness of all the edges connected to node n, and λ is a value that balances the relative weight of both terms. It is worth noticing that higher values of λ will produce larger (more interesting) paths while lower values will produce more direct paths. Experimentally we found that a reasonable value for λ is $0.5 \cdot 10^8 \cdot x$, where x is the multiplicative factor of the shortest path.

Then we use this algorithm to produce M candidate paths as follows:

```
1)  Initialize $\lambda = 0.5 \cdot 10^8 \cdot x$
2)  Set $c = 1$
3)  while $c < M$
4)      Compute the candidate path $p$ using $A*$
5)      Save the path it meets the length criterion
6)      $\lambda \leftarrow \lambda/100$
```

The running time of the algorithm in this case is highly dependent in the number of paths that we want to generate and the actual graph in which is applied. In an attempt to estimate the running time we generated 500 random source and target nodes in the graph and computed 3 paths using the above procedure in the graph of Manhattan. The mean number of iterations needed in this setup was 3.43. We estimate then the running time of the algorithm under this conditions to be $O(1.145M|V|)^2$, which makes it more efficient than the previous algorithm presented.

V. Prototype System

We built a web based demo to showcase the behavior of the implemented algorithms under different user parameters. This system follows the Model-View-Controller (MVC) design pattern. The model corresponds to the resulting node and edge scores for Manhattan, the view is built using HTML, CSS and JavaScript, and the controller is composed by the algorithm implementations in Python.

In the UI, the user is asked to input the origin and destination locations, after these are selected, the user is prompted with the shortest path. Then, the user should input a score from 0 to 1 of how Local or Touristic, Unique or Popular, the path should be. Also, a constraint on how much additional time the user is willing to walk compared to the shortest path. Finally, the user is allowed to select which algorithm to use. As an output, the user is given three proposed paths with the highest score, examples for both algorithms are shown in figures 3 and 2. Flickr pictures with highest amount of views within the path edges are shown on the right side to give the user an idea of what to expect from the journey. [3]

VI. Evaluation

We evaluate the results of this project in two different tasks. On one side we look at how each of the introduced algorithms in IV perform in terms of achieved "interestingness" scores [4]. On the other side, we evaluate how appropriate our interestingness metric is.

A. Evaluation of the algorithms

For the evaluation of the algorithms presented in this work we analyze two dimensions of its performance: (1) how much the interestingness metric is increased when the paths are generated with each of the algorithms compared to the shortest path[5]; (2) how much longer the paths are compared with the shortest path.

We generated 500 random start and end points in Manhattan with their respective shortest and most interesting path for each algorithm. Both the unique vs. popular and local vs. touristic parameters were set at 0.5 and the extra time with respect to the shortest path to $2x$.

The results for the experiments are presented in figure 1. By all means the $A*$ with the proposed heuristic is better than the randomized Dijkstra algorithm. When using this method, the

[2] $A*$ algorithm running time is $O(|E|)$

[3] For a demo video of the prototype system and please visit https://www.youtube.com/watch?v=GAvCeND9iRI

[4] Please note that even if the interestingness metric we defined was not representative of user interestingness, we can still evaluate how good the algorithms are in maximizing this metric

[5] That is what other mapping systems would have suggested

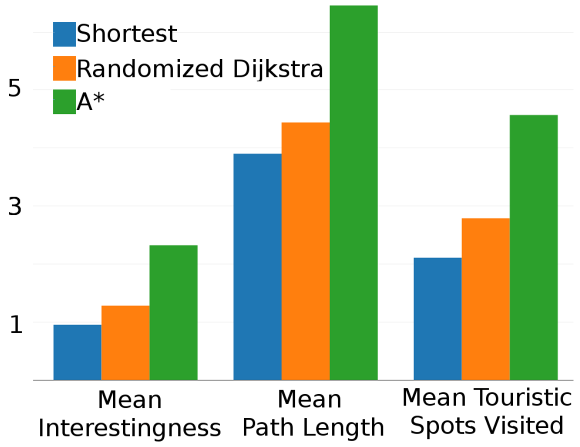

Fig. 1. Evaluation of the path characteristics obtained by each algorithm after 500 random runs.

interestingness level increases 143% on average with respect to the shortest path while the randomized Dijstra approach only shows an increase of 34%. In terms of path length, $A*$ finds paths which are, 65% longer on average than the shortest while randomized Dijkstra paths are only about 14% longer than the shortest. This means that $A*$ takes takes more advantage of the extra time available by the user.

It is worth noticing that the increase in the interestingness of $A*$ paths is not just a consequence of traversing longer paths but also making smarter routing choices

B. Evaluation of the Interestingness Score

As we mentioned in section III, how interesting or attractive a certain place or path is, will vary according to the different preferences of users. Because of that, there is no deterministic or automatic way to evaluate it.

We decided then to evaluate if the paths we generate when computed with maximum touristic and popular parameters were in fact going through the top touristic spots of the city. For this purpose we downloaded Carto^{TM}dataset of top NYC places. This dataset contains the top 309 spots of NYC with geolocation. We further filtered the table to get the touristic spots that resulted to be 210. After that, we created 500 new pairs of random start and end points in Manhattan and computed 500 paths with both the Randomized Dijkstra and the $A*$ algorithm. Finally we count the amount of touristic spots visited by each method. The results in figure 1 show that both algorithms generate paths that go through more touristic spots than the shortest paths, obtaining an increment of 32% for the case of randomized Dijkstra and an increment of 125% for $A*$

VII. Conclusions and Future Work

We have proposed and prototyped a mapping system that generates and suggests alternative, more interesting paths for the user. Given user input parameters (touristic vs. local, popular vs. unique and maximum additional time) our system shows not only the shortest/fastest walking path, but also other routes that take marginally more time but account for a higher total interestingness defined by a scoring function. In order to

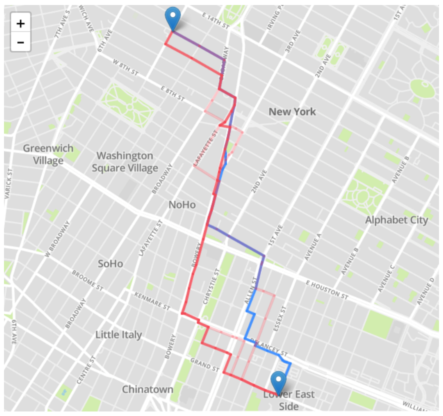

Fig. 2. Suggested paths produced by randomized Dijkstra algorithm.

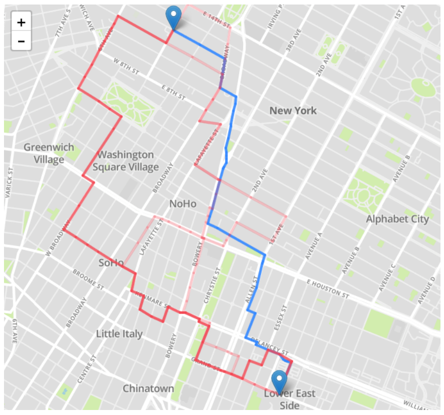

Fig. 3. Suggested paths produced by $A*$ algorithm

define this interestningess weights, we used social media data and computed several aggregated metrics at the edge/square level. The graph search algorithms we used proved to be suitable for real-time applications.

For future work, we would like to improve the interestingness function with users' feedback. For the purpose of this project, we created a deterministic scoring based on our intuition and received good informal feedback from our demo. However, to come up with a statistically significant solution, we would like to proceed with human evaluation in order to better understand user preferences and clusters. At first, this evaluation could be conducted in the form of surveys and beta testing. Later, it could be achieved by reinforcement learning and real-time feedback through a mobile application. Additionally, we think there is still room to improve the algorithms and make them even more efficient.

The prototype we built could be easily scaled to other. With more and better social data, we believe that a full functional online mapping system could be built for commercial application.

Furthermore, the system could be expanded to include other sources of data. This expansion in the data sources could also provide the possibility to expand the "dimensions" that we are mapping into the city. For example, we could add a user parameter that changes interestingness weights based on "quiet vs. busy", or "natural vs. urban".

Acknowledgment

The authors would like to thank Harish Doraiswamy from Center for Data Science, New York University. Harish was our adviser for the project and provided insight and expertise that greatly assisted the research.

Statement of contribution

Members of the team equally contributed to the development of this project.

References

[1] D. Quercia, L. M. Aiello, R. Schifanella, and A. Davies, "The digital life of walkable streets." ACM Press, pp. 875–884. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2736277.2741631

[2] D. Quercia, N. K. O'Hare, and H. Cramer, "Aesthetic capital: what makes london look beautiful, quiet, and happy?" in *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, pp. 945–955. [Online]. Available: http://dl.acm.org/citation.cfm?id=2531613

[3] D. Quercia, R. Schifanella, and L. M. Aiello, "The shortest path to happiness: recommending beautiful, quiet, and happy routes in the city." ACM Press, pp. 116–125. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2631775.2631799

[4] M. M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, Oct. 2008. [Online]. Available: http://dx.doi.org/10.1109/MPRV.2008.80

[5] V. G. Nair, *Getting Started with Beautiful Soup*. Packt Publishing, 2014.

[6] T. Tammet, A. Luberg, and P. Järv, *Sightsmap: Crowd-Sourced Popularity of the World Places*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 314–325. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36309-2_27

[7] D. Quercia, R. Schifanella, and L. M. Aiello, "The shortest path to happiness: Recommending beautiful, quiet, and happy routes in the city," *CoRR*, vol. abs/1407.1031, 2014. [Online]. Available: http://arxiv.org/abs/1407.1031

[8] D. Quercia, L. M. Aiello, R. Schifanella, and A. Davies, "The digital life of walkable streets," *CoRR*, vol. abs/1503.02825, 2015. [Online]. Available: http://arxiv.org/abs/1503.02825

[9] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Inf. Process. Lett.*, vol. 24, no. 6, pp. 377–380, Apr. 1987. [Online]. Available: http://dx.doi.org/10.1016/0020-0190(87)90114-1

[10] D. Quercia, N. K. O'Hare, and H. Cramer, "Aesthetic capital: What makes london look beautiful, quiet, and happy?" in *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work &#38; Social Computing*, ser. CSCW '14. New York, NY, USA: ACM, 2014, pp. 945–955. [Online]. Available: http://doi.acm.org/10.1145/2531602.2531613

[11] B. Ludwig, B. Zenker, and J. Schrader, *Recommendation of Personalized Routes with Public Transport Connections*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 97–107. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10263-9_9

[12] X. Meng, X. Lin, and X. Wang, "Intention oriented itinerary recommendation by bridging physical trajectories and online social networks," in *Proceedings of the ACM SIGKDD International Workshop on Urban Computing*, ser. UrbComp '12. New York, NY, USA: ACM, 2012, pp. 71–78. [Online]. Available: http://doi.acm.org/10.1145/2346496.2346508

[13] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Management Science*, vol. 17, no. 11, pp. 712–716, 1971. [Online]. Available: http://www.jstor.org/stable/2629312

[14] D. Eppstein, "Finding the k shortest paths," *SIAM J. Comput.*, vol. 28, no. 2, pp. 652–673, Feb. 1999. [Online]. Available: http://dx.doi.org/10.1137/S0097539795290477