RouteDOC: Routing with Distance, Origin and Category Constraints (Demonstration Paper)

Thomas Frohwein[*]
Zachary Garwood[*]
Dylan Hampton[*]
Kevin Knack[*]
Nate Schenck[*]
Britney Yu[*]
Joe Zuber[*]
Goce Trajcevski
tfroh,zgarwood,dhampton,kcknack,nschenck,britneyy,zubes,gocet25@iastate.edu
Iowa State University
Ames, Iowa, USA

Xu Teng
ESRI
Redlands, California, USA
xteng@esri.com

Andreas Züfle
Emory University
Atlanta, Georgia, USA
azufle@emory.edu

ABSTRACT

Route planning based on user's preferences and Points of Interests (POIs) is one of the most popular applications of Location-Based Services (LBS). Variants of route planning consider distance constraints (e.g., the maximum length of the route), origin constraints (e.g., a set of possible starting locations of the route), and category constraints (e.g., a multiset of POI categories that the route must visit). However, the problem of deciding whether a route exists that visits all required POI categories under the distance constraint is known to be NP-hard. Assuming $P \neq NP$, this means that there is no efficient (polynomial time) solution to find such paths. Recently, approximate algorithms have been proposed for searching for such a path. This demonstration leverages several of these algorithms to provide a web-based system with a graphical user interface (UI) which allows the users to find a path that: (a) satisfies a distance limit; (b) generates a route to visit a list of POIs, based on the user's preferred categories; (c) provides a set of hotels (as possible starting locations of the path). If the approximate search algorithms are able to find such a path, it will be displayed on a Mapbox-based map interface that shows: (1) all POIs on a path and (2) alternative paths if any were found. The system then allows a user to explore the returned paths, select a path, or refine their constraints. Moreover, the system allows the users to select which approximate algorithm they would prefer to execute.

CCS CONCEPTS

• **Information systems** → *Information systems applications*; • **Mathematics of computing** → **Paths and connectivity problems**.

[*]These authors contributed equally to this research.

KEYWORDS

PoI sequence, Category diversity, Distance constraint, Path origin

ACM Reference Format:
Thomas Frohwein, Zachary Garwood, Dylan Hampton, Kevin Knack, Nate Schenck, Britney Yu, Joe Zuber, Goce Trajcevski, Xu Teng, and Andreas Züfle. 2023. **RouteDOC**: Routing with Distance, Origin and Category Constraints (Demonstration Paper). In *Symposium on Spatial and Temporal Data (SSTD '23), August 23–25, 2023, Calgary, AB, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3609956.3609977

1 INTRODUCTION

Location-Based Services (LBS) have become ubiquitous in everyday life [7, 11], e.g., helping us find the shortest path to the nearest coffee shop using Global Positioning System (GPS) technology or find places inside a mall using wifi technology [2]. But in many cases, a user is not only interested in a single point of interest (POI) but may want to visit a set of POIs. For example, a tourist may want to visit two museums and one restaurant, or a user surprising their spouse with a fancy dinner may need to visit one western supermarket, one eastern supermarket, and one candle store. The motivation for this work is illustrated by the following:

Example 1.1. Sofia booked her flight to Calgary, Canada to attend a conference. She has one extra day after the conference to explore Calgary for her first time. Having only an extra day to spend for sightseeing and to optimize her tourist experience, Sofia would like to find a path such that: (1) the length of the path is limited to no more than 5,000 meters of walking distance; (2) the path includes at least two museums, at least one park, and at least one coffee shop to recharge; and (3) she needs to know which hotel would enable the constraints.

A practical algorithm to solve Sofia's request would be, starting at each conference hotel location, to sequentially find the shortest path to the nearest relevant POI. Such a solution, however, may incur a long travel distance as the nearest museum may be far from any further relevant POIs. More formally, catering to Sofia's quest amounts to answering a query which generates a *Path with Distance, Origins and Category constraints* (**PaDOC**). In addition to tourist based scenarios, the **PaDOC** query is important in ride-hailing and food delivery applications to find (near-) optimal routes for drivers to minimize traveled distance and thus, minimize emissions and impact on traffic. In such scenarios, the current location of candidate drivers corresponds to the origin constraint; a time budget of how much longer the driver wishes to work constitutes a distance constraint on the length of route; and the ride-hailing customers or food-pickup locations (generally denoted as resources) correspond to visited PoIs. There may be additional categorical constraints such as some drivers preferring food-deliveries or some resources requiring a high driver-rating. Another application is the selection of pick-up location for an e-scooter, when an individual is planning to visit certain types of stores, events, friends, etc. - within limited distance.

Recent work formally introduced the **PaDOC** query [12, 13] and experimentally demonstrated that the popular straightforward approaches to explore the (graph of the) road network are extremely inefficient and often cannot guarantee finding a solution within acceptable running time due to the NP-hardness of the problem. Two approximate algorithms were devised that explore the space of possible paths using a branch-and-bound approach. In this demonstration paper, we describe and present a system that leverages these algorithms in the backend and uses a Mapbox-based UI as the frontend to allow users to find solutions to a **PaDOC** query, display the results and all of their required POI categories within the distance limit. In the following, Section 2 surveys the algorithms employed by the backend of our **RouteDOC** system. Then, Section 3 describes the demonstrated system and how users can interact with it. Section 4 describes the scenario demonstrated at SSTD' 23.

2 BACKGROUND

The query supported by **RouteDOC** returns a path that satisfies distance, origin, and category constraints and is formally defined as follows [13].

Definition 2.1 (Path with Distance, Origin, and Category Constraints). Let $\mathcal{G} = (V, E)$ be a road network having POIs located on vertices and C denote all categories among the PoIs in $\mathcal{G}$. Given a positive value $\varepsilon \in \mathbb{R}^+$ and a vector $\boldsymbol{\theta} = \langle \theta_1, ..., \theta_{|C|} \rangle$ where θ_i $(i = 1, ..., |C|) \in \mathbb{N}$ that represents the desired number of PoIs in the i^{th} category, a path π is called a *<u>P</u>ath with <u>D</u>istance <u>O</u>rigin and <u>C</u>ategory Constraints* (**PaDOC**) if:

$$\pi.\text{length} \leq \varepsilon$$
$$\pi[0] \in O$$
$$\pi.\text{P}_i \geq \theta_i \ (\forall\, i = 1, ..., |C|),$$

where π.length is the length of Path π, $\pi[0]$ denotes the first vertex (the origin) of π, $O \subseteq V$ denotes candidate origins, and $\pi.\text{P}_i$ denotes the number of POIs of category i on π.

The problem of finding a **PaDOC** path is a (further) generalization of the *Generalized Traveling Salesman Path* (GTSP) [10] in which the path requires at least one POI from each category – whereas a **PaDOC**path may require more than one POI from a given category. The GTSP, in turn, is a generalization of the Traveling Salesman Problem (TSP) [8], where each category corresponds to exactly one location and deciding if a path of length no more than ϵ exists, is known to be NP-complete [9]. Since finding a **PaDOC** path is a generalization of finding a TSP path, it is at least as hard as finding a TSP path, and thus, is NP-hard.

Since finding an optimal solution is not feasible for large graphs (unless $P = NP$), **RouteDOC** supports four approximate algorithms to find a **PaDOC** path. These algorithms leverage an index structure called the *k-closest-category matrix* [13]. It stores, for each vertex of the road network v and for each POI category i, the set of k vertices having the k closest POIs of category i to v as well as the distance to these vertices. Details on this index structure and its efficientconstruction can be found in [13].

Random Walk with Research (RWWR). This algorithm starts at a vertex v randomly selected from the list of origin vertices O and builds a path by randomly selected adjacent edges while the distance constraint ϵ is not violated. If the resulting path satisfies the category constraints, it is returned. Otherwise, the algorithm restarts. The algorithm is forced to terminate without a result after ten seconds of unsuccessful search. This algorithm does not utilize the k-closest-category matrix.

Greedy Dijkstra Search. Using the k-closest-category matrix a simple Greedy algorithm iteratively selects to extend a partial path by including (the shortest path to) the closest vertex that contains a POI of a category that is yet needed by the current path. This is done for each possible origin location until a **PaDOC** path is found or an unsuccessful search is returned.

Origin-First Branch and Bound. This algorithm initializes the search for a **PaDOC** path with all the origin locations as incomplete paths. For each incomplete path, an upper-bound and a lower-bound distance of the optimal extension of this path are computed. The upper-bound uses the idea that if a Greedy solution of distance d exists, then the optimal solution must have a distance of no more than d. The lower-bound uses the distance maximum of all the minimum distance to categories that are yet required by the incomplete path (which can be derived from the k-closest-category matrix). This lower-bound assumes a best case where all required POI categories are "on the way" to the furthest POI category. We iteratively expand the currently active incomplete paths (stored in a priority queue) having the least upper-bound and prune any path having a lower-bound greater than the distance budget. This process is iterated until a path is expanded (and returned) that contains all required PoI categories or until the queue is empty and NULL is returned indicating that no result path is found.

POI-First Branch and Bound. This algorithm is a variant of the Branch and Bound algorithm described in the previous paragraph. Instead of initializing the search at the possible origin location, this algorithm initializes the search at relevant POI locations and builds paths "from the end". Like the Origin-First variant, this algorithm expands incomplete paths by including POI categories required by the query. Once a path is found that includes all POI categories, this algorithm checks of the first POI on this path can be reached from one of the origin locations within the distance constraint. Details on the algorithms can be found in [13].

3 THE ROUTEDOC FRAMEWORK

This section describes the technical details of our **RouteDOC** framework which are summarized in Figure 1.

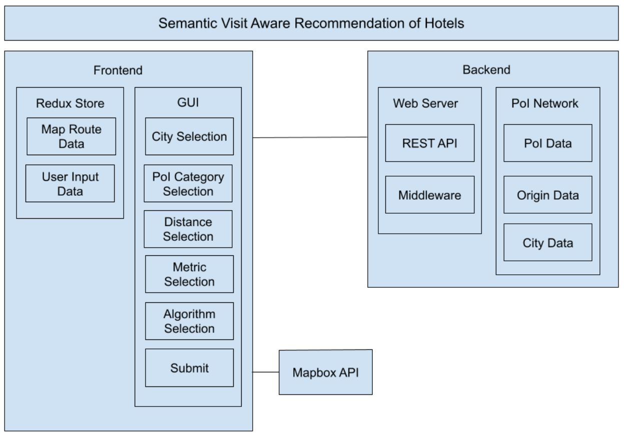


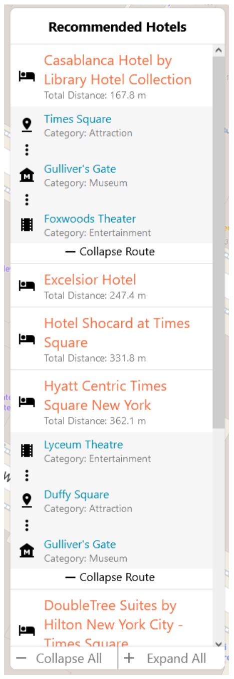
Figure 1: Framework Architecture.

3.1 Frontend

The front is implemented using TypeScript as the main language. React was used as the frontend framework to design reusable components to use throughout the application. We believe that our choice of TypeScript and React helps to make our application fulfill its needs of code readability, maintainability, and extensibility. We used Redux with React for the state management in our application. One state stores all of the user input for the application. The other state stores the map, its functions and its data. Different components of the application can control the map and the map can control them. Having Redux for our state management has helped make it easier to modify, access, and store our application's state which helps our application maintain readability and extensibility. We decided use MapBox as our visualization tool to show routes generated from our routing algorithm on the backend. MapBox has an extremely robust API and is very customizable, so we were able to fit it to our application's needs. MapBox also has many out-of-the-box features that fit the usability and visuals we wanted our application to include.

3.2 Backend

For our implementation of the backend, we decided to use Python as our primary programming language as the **PaDOC** path computation algorithms [12, 13] are implemented in Python to allow seamless integration. In order to develop our REST API, we chose Flask as our web framework [5]. Flask is a lightweight and flexible framework that allows developers to create web applications quickly and easily. Unlike other frameworks, like Django [6], Flask is less opinionated, which means developers have more control over their application's design and architecture. Additionally, Flask is highly customizable and provides the necessary features needed for our project, making it ideal. Finally, to deploy our application, we utilized Apache HTTP Server [3] on our virtual machine. Apache HTTP Server is a widely used web server that provides a robust and scalable platform for hosting web applications. To enable Apache to interact with Flask, we set up WSGI (Web Server Gateway Interface) [4] as our middleware. By setting up WSGI as our middleware,


Recommended Hotels

Casablanca Hotel by Library Hotel Collection
Total Distance: 167.8 m

Times Square
Category: Attraction

Gulliver's Gate
Category: Museum

Foxwoods Theater
Category: Entertainment

— Collapse Route

Excelsior Hotel
Total Distance: 247.4 m

Hotel Shocard at Times Square
Total Distance: 331.8 m

Hyatt Centric Times Square New York
Total Distance: 362.1 m

Lyceum Theatre
Category: Entertainment

Duffy Square
Category: Attraction

Gulliver's Gate
Category: Museum

— Collapse Route

DoubleTree Suites by Hilton New York City - Times Square

— Collapse All + Expand All

Figure 2: Visualization of results: Each PaDOC path is identified by its corresponding origin location (hotels in this scenario). Selecting a PaDOC-path expands the POIs and their categories (here: 1 Museum, 1 Attraction, 1 Entertainment) and shows the corresponding path on the map (cf. Figure 3.

we enabled Apache to communicate with Flask and forward requests to it. Implementation details, code, and documentation can be found at https://github.com/Dylan-Hampton/Semantic-Visit-Aware-Recommendation-of-Hotels. The repository also contains a link (https://sdmay23-34.sd.ece.iastate.edu/) to the Senior Design project where the "Design Documents" tab contains reports detailing the varios design decisions, ideations, testing (unit, interface, integration, regression), etc.

4 ROUTEDOC DEMONSTRATION SCENARIO

Our demonstration will feature two road networks of New York City, USA and Chicago, USA. The datasets used for constructing the POI category enriched network consist of two main resources: (1) Regular road network from OpenStreetMap; and (2) Attractions/PoIs along with related reviews crawled from TripAdvisor. We group POIs into six categories using their textual ratings from TripAdvisor: Attraction, Entertainment, Memorial, Museum, Park, and Shop. This classification is done using Latent Dirichlet Allocation [1] using the approach detailed in [14]. The user interface of our demonstration allows users to specify the query constraints including the number of requires POI categories for each class and the distance constraint. For the origin constraint, we assume all the hotels in the selected region. Once the user specifies the query parameters and presses the "Submit" button, the demonstrator will show the resulting **PaDOC** paths up to a maximum of (user specified parameter of) ten results. The backend algorithms

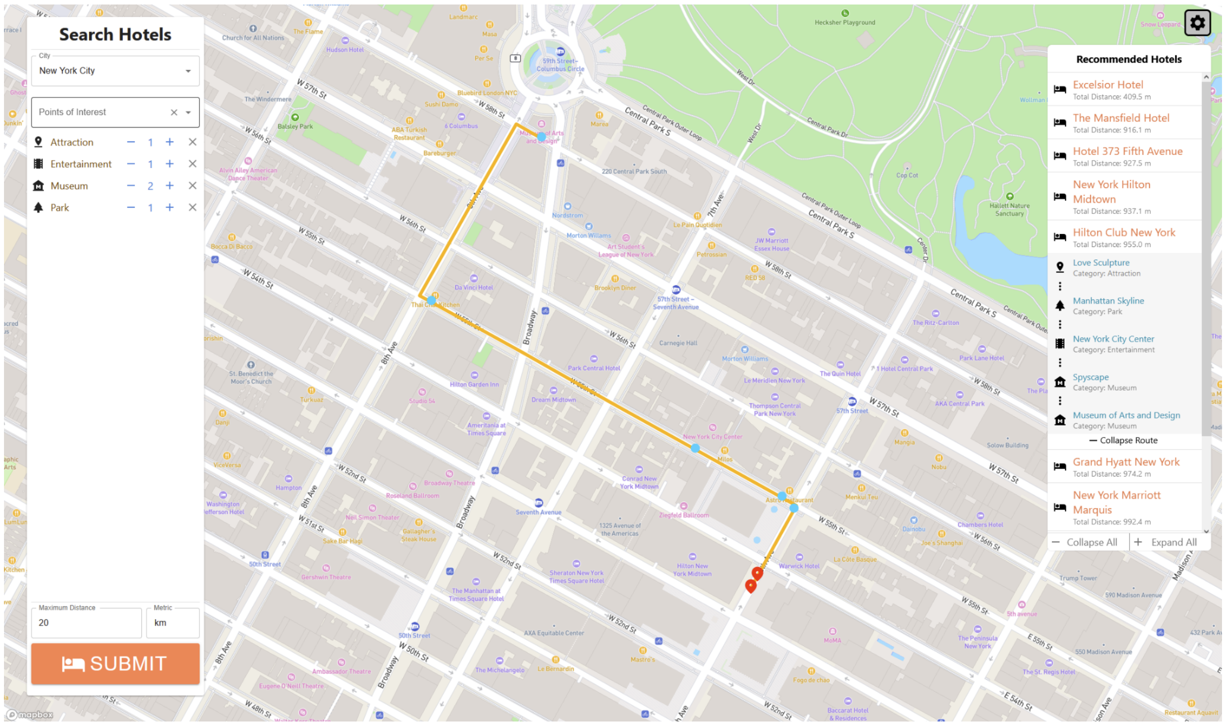


Figure 3: Screenshot of the demonstrator showing 1) the query parameters including required POI categories and distance contraints on the left, 2) query results on the right (detailed in Figure 2), and 3) a map-view of the selected query result.

will terminate early once ten paths are found. For each resulting **PaDOC** path the name of the origin POI (hotels in this scenario) are displayed. The user-interface also allows to select between the four algorithms surveyed in Section 2.

Once the results are displayed, the users may then click on each results to: (1) expand the list of POIs (and their categories) found on the path, and (2) display the path on the map. Users may also interact directly with the map, which highlights the locations of all origin locations in the query result. Clicking on any origin location also expands the list of POIs and shows the **PaDOC** path starting from that origin location.

For demonstration, we will initially showcase the New York, USA, map and demonstrate **RouteDOC** for default settings. Then, conference participants may interact with the demonstration by changing query parameters. For example, a user may stress-test the system by asking for a path that includes nine museums which will, depending on the specified distance threshold, either return no results, or return a very lengthy journey across the city. In addition to showing useful paths for trip planning, this demonstration will also allow conference participants to observe the fast running times of the algorithms proposed in [13] despite the theoretical complexity of the problem. We will explain that this efficiency is to the underlying index structure which pre-computes distances to nearest POIs of each category, and we will clarify to participants that while our algorithms may not be able to find optimal paths in many cases, they will be able to find practically useful paths in most cases. A video demonstration of **RouteDOC** can be found online at https://www.youtube.com/watch?v=YsJRH95kbng.

Acknowledgment: Research partly supported by NSF SWIFT 2030249 and 2302968.

REFERENCES

[1] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.
[2] Muhammad Aamir Cheema. 2018. Indoor location-based services: challenges and opportunities. *SIGSPATIAL Special* 10, 2 (2018), 10–17.
[3] Roy T. Fielding and Gail Kaiser. 1997. The Apache HTTP server project. *IEEE Internet Computing* 1, 4 (1997), 88–90.
[4] James Gardner. 2009. The web server gateway interface (wsgi). *The Definitive Guide to Pylons* (2009), 369–388.
[5] Miguel Grinberg. 2018. *Flask web development: developing web applications with python.* " O'Reilly Media, Inc.".
[6] Adrian Holovaty and Jacob Kaplan-Moss. 2009. *The definitive guide to Django: Web development done right.* Apress.
[7] Haosheng Huang, Georg Gartner, Jukka M Krisp, Martin Raubal, and Nico Van de Weghe. 2018. Location based services: ongoing evolution and research agenda. *Journal of Location Based Services* 12, 2 (2018), 63–93.
[8] Eugene L Lawler. 1985. The traveling salesman problem: a guided tour of combinatorial optimization. *Wiley-Interscience Series in Discrete Mathematics* (1985).
[9] Christos H Papadimitriou. 1977. The Euclidean travelling salesman problem is NP-complete. *Theoretical computer science* 4, 3 (1977), 237–244.
[10] Michael N Rice and Vassilis J Tsotras. 2012. Exact graph search algorithms for generalized traveling salesman path problems. In *SEA*.
[11] Jochen Schiller and Agnès Voisard. 2004. *Location-based services.* Elsevier.
[12] Xu Teng, Goce Trajcevski, and Andreas Züfle. 2021. Semantically Diverse Paths with Range and Origin Constraints. In *ACM SIGSPATIAL*. 375–378.
[13] Xu Teng, Goce Trajcevski, and Andreas Züfle. 2023. Distance, Origin and Category Constrained Paths. *ACM Trans. Spatial Algorithms Syst.* (May 2023). https://doi.org/10.1145/3596601 (Accepted, to appear).
[14] Xu Teng, Jingchao Yang, Joon-Seok Kim, Goce Trajcevski, Andreas Züfle, and Mario A Nascimento. 2019. Fine-Grained Diversification of Proximity Constrained Queries on Road Networks. In *SSTD*.