A More Personalized Trip-Builder

Reese Weingaertner RW71@rice.edu & Jacob Lapp jl372@rice.edu

COMP 631: Rice University

Final Video Link: https://www.youtube.com/watch?v=rQLkObf8uIU

It's your third day on a business trip in Oklahoma City. You have no idea what to do with your time outside of meetings, but do not want to sift through dozens of untailored recommendations on the internet. If you have ever traveled to a city that you have never visited before, with no person to guide you (paid or otherwise) then the former may sound more familiar than the latter. Googling "things to do in Oklahoma City" will inevitably lead you to the most "popular" spots anyone passing through town is going to. The best spots in any given city path are not easily found using Yelp, Google maps, or any other readily available travel guide. Further, creating an itinerary of different activities using sources like Yelp can be difficult. How can you plan a day where you get to see multiple places, without spending so many hours in transit? What if an app could schedule the perfect day for you, just like if you had a close friend who could lead you places they know you will love? If you have a specific "vibe" or specific request, it is time to have a platform that is more amenable to your request.

Our application is a data-driven, personalized itinerary generation for the perfect vacation. Review-based applications tend to recommend the same places for everyone, and are cumbersome for making cohesive plans. Some day-plans exist through services like "The Nudge,"[1] but these are hand tailored, only available for select cities, and are not real-time generated for events. Other generic services like Google Maps essentially just plot all business

[1] https://www.nudgetext.com/

on a map and do not curate to the user.[2] Lastly, Yelp itself can filter by activity/category but struggle to take additional information and use it for user-curation.[3] It appears that there is an open niche for this kind of application.

Data: The data to power this application comes from data collected through the Yelp platform. The data that has been collected includes information about things to do, such as parks, museums, restaurants, bars and more – and provides such textual information as the categories it is a part of, location, review count, rating, and price. There are two main APIs used for data collection: the official Yelp Fusion Developers API[4], with an unofficial implementation in python[5]; and the reviews-specific web crawler API for Yelp Reviews through Unwrangled[6]. Using Yelp's developer portal, we obtain unique API keys for querying their website and learn the general structure of the returned results. Once those are obtained, we utilize the unofficial Python library YelpAPI, which is referenced to by the official page as useful and reliable, to conduct our queries.

To fetch activities and information about those activities, the 'search_query()' method in the YelpAPI library for the Fusion API is used. This method allows us to search for a term, location, and a limit of how many businesses to return. Regarding the locations queried, we queried 75 of the largest cities in the United States, in order of largest population. The 20 categories used in every query are fixed, and they include strings including 'pizza', 'performing

2	5		Most underrated cuppa in east Austin. I love walking my dog behind their building on the hike and bike trail because it just smells soooo good! Support this Austin staple!	2024-01-03	https://www.yelp	Parker M.	East Austin, Austin, TX

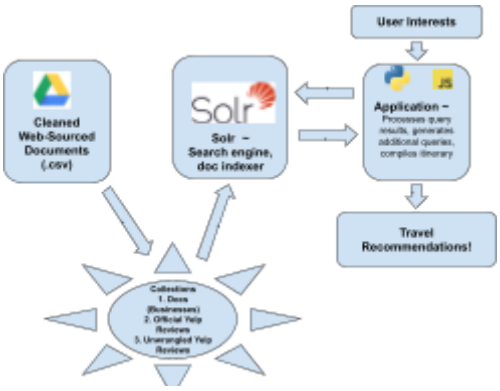
arts', 'parks', etc. The limit parameter is also fixed, returning a maximum of 50 businesses per

[2] https://www.google.com/maps
[3] https://www.yelp.com/
[4] https://github.com/Yelp/yelp-fusion?tab=readme-ov-file#code-samples
[5] https://github.com/lanl/yelpapi
[6] https://docs.unwrangle.com/yelp-reviews-api/

location. For every city in the query, we transform the resulting API pull and the data is formatted as a .csv file with all the documents for that city. That means that each .csv file contains 1000 documents about the activities in that city. Thus, we have an approximate minimum of ~75,000 documents to be used in our search engine, from this data source. Unwrangled Yelp Reviews API is utilized next, performing a web crawl, and was used to obtain tens of thousands of review documents. In conclusion, 75,000 activity documents and ~50,000 review documents were obtained as part of the first phase of this project. Further data collection of reviews, from Yelp or different sources will enhance the quality and accuracy of our end product. Other areas of improvement include expanding the categories of activities and variable lengths of activities per location (so larger locations return more businesses and smaller locations return less).

As part of the second task, we link our Yelp crawlers to Apache Solr, a popular tool to index and query large amounts of data. First, we downloaded Solr version 8.11.3 and used said version to launch our search engine locally. After cleaning the data obtained via the first task, we used the Solr Post command to upload the data for each city's attractions and associated reviews. These documents contain the geographic information, category information, ratings and free form reviews for potential stops on the itinerary. Our Solr database contains three collections: one corresponding to each of our data sources. The main core contains general information for a large array of attractions on Yelp, including name, overall rating, and geographic location. A business ID is also stored. This can be used to access reviews, which have a corresponding business ID, in our other two collections for reviews obtained from Yelp's Fusion API, and the Unwrangled Yelp Review web scraper. This is how we have access to specific user-provided information about the business, such as whether they thought the spot was

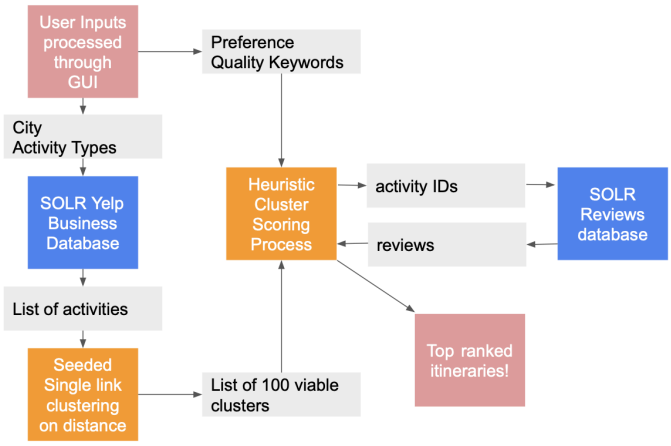
"trendy" or "touristy", which can be searched for or omitted from results based on the user's interests. The architecture of the utility up to this point is provided in the figure below. This highlights the key facets of the application's structure. One of the important parts is the application currently existing in Python script format. This application takes in user interests, queries the Solr databases for potential locations to visit, and uses the results of those queries to find even more places that are close by or rank highly in heuristics. A brief walkthrough of just the Solr implementation can be found at https://youtu.be/mZX3TbB-ulY .



For the final component of the project, we tie together the collected data and Solr's indexing ability by creating an application and a graphical user interface (GUI). The application is composed of two main classes. The first Python class, SolrSearchApp, is for connecting the user's input to Solr and presenting all of the retrieved information in a pleasant manner. Using the 'tkinter' module, the GUI contains drop-down options for the user to select where they are traveling, activities they are interested in, a minimum review score and count, how long they want their itinerary to be, and any other keywords they want (i.e. 'hidden gem', 'local', etc.). Once the user inputs all their parameters and clicks 'Search', the Python script builds a unique Solr query with those parameters. It then sends that çrafted query to the Solr server, returning a JSON with businesses that match the query.

Once a response is generated from Solr with businesses that match the query, a second Python class called "Node" unsurprisingly treats the resulting businesses as a graph. We perform **single link clustering** on the businesses (nodes) to group nearby businesses based on their

distance from each other. The distance based clusters serve as a baseline for grouping businesses into nearby plans before we consider the content of the reviews. The application provides 100 different viable itineraries that consist of businesses that are close to each other, so that users do not have to worry about massive transportation hastles. Once businesses are placed into distance-based clusters, our Solr reviews database then is queried on the preferences inputted by the user, finding which businesses match the user's inputted keywords best. This is how the application returns plans with "just the right" requests. Once the textual analysis of the businesses are complete, the 100 viable potential clusters are scored based on the sentiment of their text reviews. These new clusters are scored **heuristically**, combining distance and similarity. Finally, the heuristic clusters are the final representation of travel plans, since they have now been optimized to be within close proximity, contain relevant activities and features, and match the specific 'vibe' or requests of the user. Above is the final architecture of the application.



Results: Check out our video demonstration can be found at this link. Example itineraries are generated, and can be analyzed there if you wish!

We have found that the itinerary generation application is capable of generating practical itineraries to the users' specifications – and we plan to use it ourselves on a future trip! Clusters of activities can be found in every city that we have looked at, including Houston, New Orleans, Dallas and more.

Currently, heavy emphasis is placed on having the activities close by, but with some tweaking, user preferences could be accounted for even more, and distance less - if the user has a car, for instance. A different implementation may start from textual analysis of activity reviews, and from favorable analysis try to create clusters with the best fit. Behind the scenes, this is an entirely different application, so it was cut from scope – but an app that could employ both methods would be even better!

Another improvement to our results that could be implemented is emphasizing the diversity of activities in the itinerary as part of the score. Some generated itineraries will have you going to three different pizza restaurants, one after another - so it could be useful to generate a bigger itinerary than you need, and then cherry-pick the ones you'd like to visit.

As it stands, we built an app that solves the exact problem we hoped to solve - and can't wait to field test it and make even more improvements to the results.