# Introduction

## Semester 1: Enhancements to an Existing Map Feature

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motivation</td>
<td>3</td>
</tr>
<tr>
<td>Map View Customization</td>
<td>3</td>
</tr>
<tr>
<td>Adding a Pin to the Map</td>
<td>4</td>
</tr>
<tr>
<td>Pin Customization</td>
<td>4</td>
</tr>
<tr>
<td>Routing and Animation</td>
<td>4</td>
</tr>
<tr>
<td>Connecting Pins with Routes</td>
<td>4</td>
</tr>
<tr>
<td>Animating Routes with Advanced Camera Controls and Visualization Techniques</td>
<td>4</td>
</tr>
<tr>
<td>Route Animation and Camera Control</td>
<td>5</td>
</tr>
<tr>
<td>Camera Movement Along the Route</td>
<td>5</td>
</tr>
<tr>
<td>Computing Camera Position</td>
<td>5</td>
</tr>
<tr>
<td>Dynamic Adjustment in Street View Animation</td>
<td>6</td>
</tr>
<tr>
<td>Smoothing Camera Movements</td>
<td>7</td>
</tr>
<tr>
<td>User Customization in Animation</td>
<td>8</td>
</tr>
</tbody>
</table>

## Semester 2: Trip Itinerary Recommendation System

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choosing a “Root” Location</td>
<td>8</td>
</tr>
<tr>
<td>User Preferences</td>
<td>8</td>
</tr>
<tr>
<td>Retrieving Points of Interest</td>
<td>8</td>
</tr>
<tr>
<td>Calculating Limits for POI Retrieval</td>
<td>9</td>
</tr>
<tr>
<td>Organizing Points of Interest for Enhanced Visualization</td>
<td>9</td>
</tr>
<tr>
<td>A Deeper Dive into Geographic Clustering Using the DBSCAN Algorithm</td>
<td>11</td>
</tr>
<tr>
<td>Selecting Transportation Preferences</td>
<td>12</td>
</tr>
<tr>
<td>Itinerary Construction Using Ant Colony Optimization</td>
<td>12</td>
</tr>
<tr>
<td>Why Ant Colony Optimization?</td>
<td>12</td>
</tr>
<tr>
<td>Implementation Details</td>
<td>13</td>
</tr>
<tr>
<td>Daily Itinerary Construction with ACO</td>
<td>13</td>
</tr>
<tr>
<td>Subpath Construction</td>
<td>13</td>
</tr>
<tr>
<td>Heuristic Calculation</td>
<td>14</td>
</tr>
<tr>
<td>Pheromone Update Rules</td>
<td>14</td>
</tr>
<tr>
<td>Utility Function for Path Evaluation</td>
<td>15</td>
</tr>
<tr>
<td>Best Path Selection</td>
<td>15</td>
</tr>
<tr>
<td>Decision Factors for Transportation Mode</td>
<td>16</td>
</tr>
<tr>
<td>Thresholds Used for Transportation Decision</td>
<td>16</td>
</tr>
<tr>
<td>Integration in Itinerary Planning</td>
<td>17</td>
</tr>
<tr>
<td>Comparative Analysis of Ant Counts</td>
<td>17</td>
</tr>
<tr>
<td>The Impact of Ant Numbers in Ant Colony Optimization</td>
<td>17</td>
</tr>
</tbody>
</table>
Methodology and Findings 17
Decision and Rationale 18
Displaying the Itinerary 18

**Possible Further Enhancements** 20
- Dining Recommendations 20
- Accommodation and Flight Suggestions 20
- Detailed Route Information 20
- Visualization of Public Transport Routes 20
- Custom Itinerary Creation 21
- Restaurant Preferences 21
- Detailed Transportation Preferences 21

**Appendix** 22
Integrating Advanced Mapping and Itinerary Planning Features in Dash

By Zaul Tavangar

Introduction

This paper presents the enhancements and new developments I implemented in Dash, an open-component hypermedia system operated by Brown University, over the course of two consecutive semesters. In the first semester, I focused on enhancing the existing map feature or component, notably by integrating routing and animation capabilities that were previously lacking, thereby improving user interaction and functionality. The subsequent semester was dedicated to creating a trip itinerary recommendation system. This system leverages Ant Colony Optimization (ACO) to suggest and optimize daily itineraries based on points of interest around a specified "root" location, offering route options for driving, walking, or public transit. The paper is organized into two main sections, each detailing the developments from a respective semester.

Semester 1: Enhancements to an Existing Map Feature

Motivation

The initial version of Dash’s map feature primarily supported basic functionalities like location search and placing markers on the map using the Bing Maps API. The primary objective for the first semester project was to enhance the interactivity and capabilities of the mapping feature. To achieve this, I transitioned to the Mapbox API, which offers finer support for custom layering and styling, a comprehensively documented and user-friendly API, and a visually appealing selection of map styles.

Map View Customization

To enhance personalization, the map interface offers extensive customization options. Users can select from nine distinct map styles provided by Mapbox, catering to a variety of aesthetic preferences. These styles include Standard, Streets, Outdoors, Light, Dark, Satellite, Satellite Streets, Navigation Day, and Navigation Night. Additionally, users have the flexibility to adjust the map's bearing, pitch, and zoom levels to suit their specific viewing needs. One can also choose to display terrain for additional geographical detail. This range of customization options ensures that users can create a map view that is both visually appealing and optimally useful for their needs. For a screenshot of the customization menu, please see Figure 7 in the Appendix.
Adding a Pin to the Map

The process of adding a pin begins with a user-initiated location search. Utilizing the Mapbox Geocoding API, the system dynamically displays search results as the user types, with built-in throttling to manage the frequency of API requests. Search results are displayed as simple location names; however, the Mapbox API returns GeoJson features that include detailed geographic and descriptive data for each location. Selecting a search result automatically retrieves its coordinates, places a pin at that location, and centers the map on this point. Alternatively, users can double-click anywhere on the map to trigger the Mapbox Reverse Geocoding API and place a pin at that clicked location based on the coordinates.

Pin Customization

Continuing with the theme of enhancing user engagement, I introduced options for customizing the appearance of individual markers. Users can change both the color and icon of the marker, allowing for a more personalized mapping experience. For example, a user marking hiking trails might choose an icon that visually represents hiking, adding contextual relevance to the map markers. To view this pin customization menu, please refer to Figure 8 in the Appendix.

Routing and Animation

Connecting Pins with Routes

A significant enhancement to the previous map feature was the ability to create and display routes between markers. Users can initiate route planning by selecting the “Get Directions” option from a pin’s menu, which prompts an overlay where they can enter a destination. The system then uses the Mapbox Geocoding API to display potential destinations. Users can select either a new location or an existing pin as their endpoint. Upon confirming the destination, the system offers route options for driving, cycling, and walking — unfortunately the Mapbox API does not support public transit options. Users can view and cycle through the available routes, comparing path, duration, and distance before selecting their preferred route. Once a route is chosen, it is added to the map and highlighted, with further options for customizing the route's color. To see how this route creation process works, please refer to Figure 9 in the Appendix.

Animating Routes with Advanced Camera Controls and Visualization Techniques

This section delves into the methods and technologies used to create dynamic and engaging animations of geographical routes, drawing inspiration from visual representations such as those seen in Tour de France stage previews. These animations employ a mix of 3D terrain data, sophisticated camera movements, and Mapbox technologies. This implementation both draws
heavily from and significantly expands upon concepts discussed in an article from the Mapbox site regarding similar animation techniques.

Route Animation and Camera Control

The animation process begins with a GeoJSON LineString that outlines the route. The main objective is to animate this route incrementally to simulate progression over time. This is accomplished by manipulating the line-gradient paint property, influenced by an “animation phase” calculated from the elapsed time since the start of the animation relative to its total duration, producing a value between zero and one:

```
// Calculation of animationPhase
animationPhase = (currentTime - startTime) / totalDuration;
```

Camera Movement Along the Route

To synchronize the camera with the ongoing animation, functions from turf.js are used to pinpoint the leading edge of the route for each frame. The distance along the route that corresponds to the current animation phase is calculated as follows:

```
const alongPath = turf.along(path, pathDistance * animationPhase).geometry.coordinates;
```

The camera's movement is dictated by its position (altitude and location) and orientation (pitch and bearing). While pitch and altitude remain fixed in third-person views, they adjust dynamically in the first-person or “street view” mode to enhance realism, particularly in mountainous terrains. More details on this adjustment are provided subsequently.

Computing Camera Position

The camera's position is calculated using trigonometric functions that account for its altitude, bearing, and pitch relative to the focal point:

```
const computeCameraPosition = (pitch, bearing, targetPosition, altitude, smooth = false) => {
  let bearingInRadians = (bearing * Math.PI) / 180;
  let pitchInRadians = ((90 - pitch) * Math.PI) / 180;
  let lngDiff = (altitude / Math.tan(pitchInRadians)) *
    Math.sin(-bearingInRadians) / 70000; // ~70km/degree Longitude
  let latDiff = (altitude / Math.tan(pitchInRadians)) *
```

---

This formula calculates the camera's longitudinal and latitudinal displacement based on its altitude and the angles of bearing and pitch, always providing a new viewpoint that ensures the route's leading edge remains in the frame.

Dynamic Adjustment in Street View Animation

In first-person, or street view animation, pitch and altitude are dynamically adjusted to provide a realistic perspective, enhancing the viewer's immersive experience. One particular use case of this type of animation would be to visualize a hiking path along a mountainous trail. Unlike third-person views where pitch and altitude remain static, in street view mode, both these parameters vary in response to the continuously changing terrain elevation and the relative positioning of consecutive points along the route.

The Mapbox API plays an important role in facilitating these adjustments by providing an API endpoint that retrieves elevation data for a specific set of coordinates. This allows the camera’s altitude to be set slightly above the actual terrain elevation. Specifically, once the elevation data is retrieved, a predefined adjustment factor is added to the elevation value to position the camera above obstacles and provide a clear view of the trail ahead. This adjusted altitude ensures that the viewer’s line of sight is directed along the trail, rather than directly at the ground or sky, especially in steep or uneven terrain.

The calculation of pitch in street view is key for maintaining a relatively natural viewpoint. In the ideal scenario where the pitch could exceed 90 degrees, or fall below zero degrees, one could simply tilt the camera upwards on an ascent or downwards on a descent, directly mimicking our natural human head movement. However, due to Mapbox’s limitation where the pitch must lie between zero and 90 (inclusive), the camera cannot be tilted upward beyond the horizontal plane, which poses a challenge in steep and/or mountainous terrains.

To address this limitation, the pitch is calculated based on the difference in elevation between consecutive points along the route, combined with the camera's altitude. The pitch angle is adjusted to ensure the camera looks downward or upward to the next point along the trail without exceeding the 90-degree limit. This is done by calculating the arctangent of the vertical difference divided by the horizontal distance (essentially the slope) between successive points. The formula used is as follows:

```javascript
Math.cos(-bearingInRadians) / 110000; // 110km/degree Latitude
let correctedLng = targetPosition.lng + lngDiff;
let correctedLat = targetPosition.lat - latDiff;
return { lng: correctedLng, lat: correctedLat };
let verticalDifference = nextPointElevation - currentPointElevation;
let horizontalDistance = calculateDistance(currentPoint, nextPoint);
let desiredPitch = Math.atan(verticalDifference / horizontalDistance) * (180 / Math.PI);

This calculated pitch attempts to closely align the viewer’s perspective with the natural line of sight one would have if physically present on the trail, despite the constraints imposed by Mapbox.

Figure 1: A screen-capture taken during a street view animation that follows a hiking path from Courmayeur, Italy to Aosta, Italy. One can notice the altitude of the camera is slightly higher than the actual path, for reasons explained above.

Smoothing Camera Movements
To prevent abrupt changes in camera position and ensure smoother transitions, a linear interpolation method (lerp) is applied. This method smooths out the movement between previous and current positions, crucial for maintaining the visual quality of the animation, especially when the path includes sharp turns or rapid directional changes.

function lerp(start, end, amt) { return (1 - amt) * start + amt * end; }
User Customization in Animation

Another aspect of this animation system is allowing user control over the animation speed and the color of the route. The animation speed affects the calculation of phase increments and thus the pace at which the route is revealed and the camera moves:

```java
animationSpeed = userSelectedSpeed;
updateAnimationPhase(currentPhase + phaseIncrementBasedOnSpeed);
```

By enabling users to adjust these settings, the animation can be further tailored to individual preferences, enhancing the user experience.

This approach to animating routes not only improves how landscapes are visualized but also introduces an interactive element, enabling users to dynamically explore and experience different paths and terrains.

Semester 2: Trip Itinerary Recommendation System

Choosing a “Root” Location

The development of the itinerary begins by selecting an initial “root” location, which is crucial for subsequently retrieving nearby points of interest (POIs). For this best experience, this root location is typically a city or any place rich in POIs. To choose this root location, users once again, as done with the map feature, engage with a search interface that utilizes the Mapbox Geocoding API.

User Preferences

After establishing a root location, users specify their preferences, which guide the itinerary creation. Firstly, users select the date range for their trip using a date range picker, which records the start and end dates. Following this, users rate their interest in six POI categories on a scale from 1 to 10: “Landmarks & Outdoors”, “Religious & Spiritual”, “Arts & Entertainment”, “Education & Public Services”, “Retail”, and “Sports & Recreation”. These preferences are crucial as they influence the selection and quantity of POIs subsequently retrieved from the FourSquare API based on the length of the trip and the user's interests.

Retrieving Points of Interest

The system retrieves POIs through the FourSquare API, using the longitude and latitude of the root location. Each category's request is customized by a specific parameter named categoryIds, which filters the POIs according to the user's preferences.
Calculating Limits for POI Retrieval

Using the duration of the trip, calculated from the user’s imputed starting and ending dates, the system adjusts the number of POIs to retrieve per category based on the user's ratings for each category. An “overshoot factor” of 2.5 is used to slightly increase the number of potential POIs, allowing for a broader selection. The formula for calculating the limit of POIs per category is:

\[
\text{const adjustedLimit} = \text{Math.min(} \text{MAX\_LIMIT, Math.max(} 1, \text{Math.ceil(}((\text{preferenceValue as number}) / 10) * \text{DAYS } \times \text{overshootFactor}))\text{)};
\]

Here, preferenceValue is the user's rating for each category. The formula calculates an initial number of POIs by proportionally allocating more POIs to higher-rated categories and extending this number based on the trip's duration and the overshoot factor. This number is then bounded by a MAX\_LIMIT, 30, to ensure manageability and performance.

This system offers a tailored approach to itinerary planning, leveraging user input to dynamically generate a list of POIs that aligns with personal preferences and the trip’s duration.

Organizing Points of Interest for Enhanced Visualization

To effectively display the retrieved POIs in a manner that is both concise and visually appealing, the design was inspired by Microsoft Silverlight’s Pivot Viewer\(^2\). This approach groups POIs using a chosen parameter (a pivot), sorting them into categorized "buckets." Such an approach becomes especially important when managing large datasets, or in the context of this project, large numbers of POIs. The display, which can be seen in Figure 2 below, resembles a bar graph where the x-axis labels denote categories determined by the pivot, with POIs organized beneath each label in rows.

\(^2\) Microsoft Research, “PivotView Overview,” YouTube video, August 15, 2016, https://www.youtube.com/watch?v=8Aey3xm0CBg.
Figure 2: The viewer for the retrieved POIs. When the pivot changes, the colors of circles representing the POIs on the map also change, and are grouped by color based on the selected pivot parameter. In this example, the preference values for the “Retail” and “Sports & Recreation” categories were set to three, and one, respectively, and ten for the rest.

In my adaptation of the Pivot Viewer, I employed three parameters as pivots: Category, Rating, and Location (proximity). The default pivot, Category, organizes each bucket by POI type such as "Arts & Entertainment" or "Religious & Spiritual", as can be seen in the figure above. When the pivot is set to “Rating”, each bucket corresponds to a range of ratings, such as 1-2 or 6-7. The “Location (proximity)” pivot groups POIs based on their geographical proximity, utilizing the turf.clusterDbScan function to cluster nearby points together, which is calculated using the Haversine formula — the most accurate way of computing distances between two points on the surface of a sphere — integrated within turf's distance function.

Having such a view would be useless without a way to view detailed information for each POI. A click on any of the POI “boxes” triggers an overlay containing specific information regarding that chosen POI, as can been seen in Figure 3 below.
Figure 3: An example of the detailed POI overlay that appears when a POI “box” is clicked.

A Deeper Dive into Geographic Clustering Using the DBSCAN Algorithm

The DBSCAN clustering method is particularly suited for geographic data as it groups points that are closely packed together while marking as outliers those that lie alone in low-density regions. Here’s how the clustering is implemented:

1. Preparation and Distance Calculation:
   - Each POI is converted into a point feature with latitude and longitude coordinates.
   - Distances between every pair of points are calculated to determine the maximum, minimum and average distances, which are essential for setting the DBSCAN parameters.

   ```javascript
   for (let i = 0; i < points.length; i++) {
     for (let j = i + 1; j < points.length; j++) {
       const distance = turf.distance(points[i], points[j]);
       MAX_DIST = Math.max(MAX_DIST, distance);
       MIN_DIST = Math.min(MIN_DIST, distance);
       totalDistance += distance;
       count++;
     }
   }
   let AVG_DIST = totalDistance / count;
   ```

2. Executing DBSCAN:
   - The turf.clustersDbscan function is then called with a dynamically calculated epsilon (the maximum distance between two points for one to be considered as in
the neighborhood of the other) that is a function of the minimum and average distances. This approach ensures adaptability to the geographic distribution of the POIs.

\[
\text{const } \epsilon = (\text{MIN\_DIST} + \text{AVG\_DIST}) / 20; \\
\text{const clustered } = \text{turf.clustersDbscan(} \text{featureCollection}, \epsilon, \{\text{minPoints: 1}\});
\]

3. Post-Clustering Organization
   - Once points are clustered, the results are organized into a dictionary where each cluster key maps to a list of associated POIs. Single-POI clusters are then further grouped together to ensure no POI is left isolated, enhancing the visualization and user understanding of the data.

This structured method of displaying POIs not only aids in the visual interpretation of data but also enhances the usability of the itinerary planning tool by enabling users to easily navigate through a categorized and logically grouped set of POIs.

Additionally, when the pivot (grouping parameter) changes, the groupings are also displayed on a map alongside the main display, with POIs in the same group represented by circles of the same color. This visual representation within the map helps users better understand the groupings and facilitates navigating through the data by making the geographical distributions and relationships clear and straightforward.

Selecting Transportation Preferences
A crucial yet straightforward step in the itinerary creation process involves selecting transportation preferences. Users are prompted to make two key decisions: firstly, whether they plan on renting a car during their trip, and secondly, whether they generally prefer walking or using public transit for moving between locations. These preferences are vital as they significantly influence the transportation options considered during the itinerary planning, and ultimately affect the selection of transit routes between each POI.

Itinerary Construction Using Ant Colony Optimization
The itinerary is constructed using the Ant Colony Optimization (ACO) algorithm, a probabilistic technique for solving computational problems that simulates the behavior of ants searching for food.

Why Ant Colony Optimization?
ACO was chosen for its efficiency in finding good paths through graphs, making it particularly suitable for itinerary planning where multiple stops (nodes) are connected via possible routes.
(edges). It excels over other optimization algorithms like genetic algorithms, another option which was considered, which involve mutations and crossovers, because:

- ACO is more dynamic: It continuously updates its pheromone levels based on the quality of each solution, allowing it to adapt and improve through iterations.
- Lower complexity for small to medium datasets: While genetic algorithms are powerful for large datasets with genetic diversity considerations, ACO provides a more intuitive and less computationally expensive approach for the typically smaller datasets involved in travel itinerary planning.
- Better at local optimization: ACO can more effectively fine-tune solutions to find the best local routes between points, which is essential for creating practical travel itineraries.

Implementation Details

In the implementation, ants represent travelers navigating between POIs based on a set of rules derived from the real-world behavior of ants, including the selection of paths based on the intensity of pheromones, which represent the desirability of particular routes:

1. Initialization: Each ant is initialized at a starting POI and given a set of possible destinations (POIs).
2. Route Building: Each ant builds its route by moving to other POIs, primarily guided by the distance between the POIs and the pheromone levels on each path, although other factors also play a role. As ants travel from one POI to another, routes for driving, transit, and walking are retrieved from Bing Maps' navigation endpoint and stored within the path. The Bing Maps Navigation API was chosen instead of Mapbox’s Directions API, as the latter does not provide public transport routes, which are essential in itinerary planning. The choice of transportation method depends on user preferences and other factors discussed in the subsequent sections.
3. Pheromone Update: Upon completion of a route, the paths taken by the ant are reinforced with pheromones, increasing their attractiveness to future ants. Conversely, paths not taken decay in desirability.

Daily Itinerary Construction with ACO

Unlike traditional ACO implementations that aim to construct a single optimal path, such as to solve the infamous Traveling Salesman Problem, this itinerary planner instead builds a subpath for each day. This approach is more aligned with real-world travel, where activities are generally confined within the constraints of daily schedules. Here's how it differs and some mathematical specifics:

Subpath Construction

Each day is treated as a separate problem where ants simulate a day's travel, including meals and other breaks, and plan routes between POIs accordingly. This daily segmentation not only allows
us to construct separate itineraries day-by-day, but also helps manage complexity by focusing on shorter, more manageable segments. Importantly, the system discourages visiting two consecutive POIs of the same category to enhance the travel experience, reflecting typical tourist behavior where variety is preferred.

Heuristic Calculation

The heuristic value in ACO is critical as it influences the ants' path selection. More specifically, it helps the ants decide which POI to visit next. For my itinerary planner, the heuristic is calculated as follows:

```javascript
// Pseudocode for heuristic calculation
function calculateHeuristic(distance, rating, stats) {
  const ratingInfluence = 0.25;
  const statsInfluence = 0.25;
  const heuristicBase = 1 / (distance + 1); // Avoid division by zero
  return (heuristicBase + ratingInfluence * (rating / 10) + statsInfluence * normalizeStats(stats)) / 3;
}
```

This formula adjusts the heuristic based on the distance to the next POI, the POI's user rating, and statistical data about the POI's popularity (a normalized score based on the total number of photos, ratings, and reviews for a POI within the FourSquare database), providing a balanced approach that considers both quantitative and qualitative factors.

Pheromone Update Rules

After each day's route is constructed, the pheromone levels on the traveled paths are updated to reflect their desirability:

```javascript
// Pseudocode for pheromone update
function updatePheromones(pheromones, path, decayRate, reinforcement) {
  // Apply decay to all pheromone trails
  for each (route in pheromones) {
    pheromones[route] *= (1 - decayRate);
  }
  // Reinforce pheromones on the chosen path
  for each (segment in path) {
    pheromones[segment.start][segment.end] += reinforcement;
  }
}
```
This dual mechanism of decay and reinforcement ensures that the pheromone map evolves over time, reflecting the changing evaluations of route desirability based on ongoing feedback from the ants.

Utility Function for Path Evaluation

The utility of a path is calculated based on the total distance, with shorter paths preferred to minimize travel time and maximize efficiency:

```javascript
// Function to calculate the utility of a path
function calculatePathUtility(path) {
  let totalDistance = 0;
  for each (segment in path) {
    totalDistance += distance(segment.start, segment.end);
  }
  return -totalDistance; // Negative as shorter distances are preferred
}
```

Best Path Selection

At the end of each day, the best path is selected based on its utility:

```javascript
function chooseBestPathOfDay(paths) {
  let bestPath = null;
  let highestUtility = -Infinity;
  for each (path in paths) {
    let utility = calculatePathUtility(path);
    if (utility > highestUtility) {
      highestUtility = utility;
      bestPath = path;
    }
  }
  return bestPath;
}
```

This method ensures that the chosen daily path optimizes the travel experience based on the predefined utility function, balancing the need to explore new paths with the efficiency of proven routes. To prevent visiting the same POI on different days, after the best path has been chosen for a particular day, it is added to a set of visited POIs, and excluded from the rest of the creation process.
Decision Factors for Transportation Mode

1. **User Preferences**: The user’s general transportation preferences (whether they prefer walking vs. public transit, or whether than plan on renting a car) are initially collected, as previously explained. These preferences are crucial as they directly influence the prioritization of transportation modes.

2. **Distance Considerations**: The distance between consecutive POIs also plays a key role. Distances are calculated using turf.distance function, as done when clustering the POIs by geographic proximity.

Thresholds Used for Transportation Decision

- **Walking Threshold**: The maximum distance a user is willing to walk. I chose to set this to 20 minutes of walking time. For users who prefer walking, this threshold is extended to 45 minutes to encourage walking as a more frequent option.
- **Driving Threshold**: Used when a user plans on renting a car. This threshold is set to 75 minutes, a reasonable limit beyond which driving is typically more time-efficient than using public transportation.
- **Public Transit Availability**: Public transit is considered when it provides a timely alternative to walking and driving, especially in urban areas where transit networks are well-developed.

```javascript
function chooseTransportMode(drivingDuration, walkingDuration, transitDuration){
    const driveThreshold = 75 * 60; // 75 min
    const walkingTimeLimit = this.preferWalking ? 45 * 60 : 20 * 60; // if prefer walking 45 min, otherwise 20 min
    if (!this.driving){
        if (walkingDuration < walkingTimeLimit) {
            return 'Walking';
        } else if (transitAvailable) {
            return 'Transit';
        } else {
            return 'Walking';
        }
    } else {
        if (drivingDuration >= driveThreshold){
            return 'Driving';
        } else if (walkingDuration < walkingTimeLimit) {
            return 'Walking';
        } else if (transitAvailable){
            return 'Transit';
        } else {
            return 'Walking';
        }
    }
}
```
Note that even if a user plans to rent a car, walking or public transit might still be chosen if the destination is nearby and within the defined thresholds.

Integration in Itinerary Planning

During itinerary planning, this transportation selection function is called at each step to decide how the traveler will move from one POI to the next. This decision affects the route calculations, estimated travel times, and ultimately, the overall itinerary layout. Transit routes and, more specifically, the durations passed into the function above, are sourced from Bing Maps’ Navigation API, as previously mentioned.

By integrating these preferences and thresholds, the itinerary planning tool ensures that each travel segment is optimized for convenience, efficiency, and personal preference, resulting in a more enjoyable and customized travel experience.

Comparative Analysis of Ant Counts

The Impact of Ant Numbers in Ant Colony Optimization

In Ant Colony Optimization (ACO), the number of ants plays a crucial role in the efficiency and effectiveness of the solution. More ants can enhance the exploration of the search space, potentially leading to better solutions by covering more paths and reinforcing the most successful ones. However, this benefit comes with an increase in computational cost, as more paths need to be calculated and more pheromone information needs to be updated and stored.

Conversely, fewer ants may lead to faster computation times but can risk poorer solutions due to less thorough exploration. This balance between ant quantity, computation time, and solution quality is critical, especially in complex scenarios like itinerary planning where multiple POIs are involved.

Methodology and Findings

To identify the optimal number of ants for itinerary planning, simulations involving 60 POIs were conducted with varying ant counts: 5, 10, 15, 20, and 25. The evaluation criteria focused on computation time and the subjective quality of the resulting itineraries. Analysis of the itineraries revealed that the quality generally improves with an increasing number of ants, at least up to 25 (the maximum value tested).

However, it is crucial to strike a balance between the quality of the itinerary and the processing time. A plotted graph of processing time against the number of ants, shown in Figure 4 below, demonstrates an anticipated increase in time with more ants. Notably, at around 15 ants, there is
a noticeable change in the rate of increase in computation time, where it begins to rise more steeply.

**Itinerary Creation Processing Time vs. Number of Ants**

![Graph showing the relationship between processing time and number of ants.](image)

**Figure 4: Itinerary Creation Processing Time (in seconds), vs. Number of Ants**

**Decision and Rationale**

Using the graph and itinerary quality data, 15 ants was thus identified as the optimal count. This number offers a reasonable compromise, providing well-constructed travel plans without excessive processing time. The choice of 15 ants represents the best balance, maximizing itinerary quality while maintaining manageable computation times (a little over three minutes for 60 POIs).

**Displaying the Itinerary**

Once the itinerary creation process concludes, the complete itinerary is accessible and interactive, organized by day. Each day is presented within an accordion menu that, when expanded, does two things: 1) it shows the routes between POIs for that day on the map and 2) allows detailed exploration and navigation of the day’s activities. Below, Figure 5 illustrates the standard itinerary view, with circles representing the daily POIs color-coded by day. Figure 6 shows the expanded view for a specific day, highlighting the travel routes between the POIs on the map.
Figure 5: The standard itinerary view, with POIs color-coded by day on the map.

Figure 6: The itinerary view when the itinerary for a specific day is expanded.
To finalize the process, the completed itinerary can be saved to the Dash Dashboard, where it can be accessed later for further exploration or use.

**Possible Further Enhancements**

**Dining Recommendations**

One enhancement, currently in progress, involves integrating restaurant and bar suggestions. Currently, while lunch and dinner breaks are scheduled, specific dining recommendations are not provided. An ideal enhancement would allow users to click on a meal break to receive, say, a list of five nearby dining options, positioned conveniently between the preceding and subsequent POIs.

**Accommodation and Flight Suggestions**

Another enhancement would involve incorporating accommodation and flight suggestions. For instance, if the itinerary is planned for Paris with a starting location in Providence, it would be beneficial to suggest flights from Providence to Paris and recommend hotels within Paris. This feature was considered but not initially implemented due to concerns that users might prefer more established platforms for financial transactions involved in booking hotels and flights. However, offering this service could still enhance the overall utility of the itinerary by providing a more comprehensive travel planning experience, even if users choose to search for offers and book through other services.

**Detailed Route Information**

Currently, selecting a public transport route in the itinerary displays essential information (can be seen at the bottom right of Figure 6 on the previous page). An enhancement could involve providing a pop-up with detailed, step-by-step directions for each route segment when clicked. This would require storing additional detailed navigation data retrieved from the Bing Maps API during the Ant Colony Optimization (ACO) algorithm process.

**Visualization of Public Transport Routes**

Enhancing public transport options by displaying specific subway or bus lines on the map upon selection could further improve user interaction. If Bing Maps API does not support this feature, integrating an additional API might be necessary.
Custom Itinerary Creation

The ability for users to define specific starting and ending points, as well as waypoints, would greatly enhance the customization of itineraries. For example, if a user plans separate trips to Paris and Rome, and has built an itinerary for each, they could benefit from a feature that links these two itineraries into a cohesive travel plan. By specifying Paris Charles de Gaulle Airport as the starting point and Rome Fiumicino Airport as the endpoint, the system could suggest optimal flight connections between these cities and help effectively “link” the itineraries.

To implement this feature, flight data (when applicable) would need to be retrieved from an external API that provides comprehensive flight schedules and availability, as well as, ideally, the coordinates describing the flight route, ensuring that users can both plan and visualize the complete journey, including any intermediate stops or waypoints they wish to explore.

Restaurant Preferences

During the itinerary planning phase, users could specify their preferred cuisines and any dietary restrictions. This personalized approach ensures that restaurant suggestions for meal breaks align more closely with individual preferences and dietary needs.

Detailed Transportation Preferences

While current transportation choices are based on whether the user has a car, generally prefers walking vs public transport, and set thresholds for walking and driving, a more tailored approach could allow users to define these thresholds themselves, providing greater flexibility and personalization in the planning process.
Appendix

Figure 7: Map display customization options. The map style is selected from the dropdown and pitch, bearing, and zoom can be set through the inputs.
Figure 8: Pin customization options.

Figure 9: An example of route creation. The figure displays the driving route from Providence, RI, to New London, CT. Other transit routes can be examined by clicking on the bicycle and walking icons.
Figure 10: An example of a route added to the map. Clicking on the “+” icon in the previous figure, Figure 9, yields the route above.
Figure 11: The animation menu. One can control the speed, the line color, and the type of animation (first person/street view vs. third person).
Figure 12: A screen-capture taken near the end of a third-person animation that follows a driving route from Providence, RI, to New London, CT.