Multi-Modal Route Optimization with Hybrid Travel

Urban transportation is essential for millions of people, but current route recommendation systems often fail to account for hybrid suggestions to lower wait times that significantly impact user experience. This research introduces a new hybrid-multimodal routing algorithm that addresses this issue by combining different modes of transportation within a single trip.

The algorithm's goal is to optimize urban travel by allowing users to avoid unpleasantries like traffic jams, long walks, and awkward street routing by switching between modes of transport. Our study shows that this approach can significantly improve the cost, duration,and overall quality of urban trips. This approach will use ML to help users to optimize their travel by choosing various means of transportation.

Key contributions of this research include:

1. **A machine learning-based route prediction algorithm:** This algorithm utilizes historical trip data and real-time information to predict optimal routes that combine different modes of transportation, considering current conditions. (MAIN GOAL)
2. **A user experience model:** This model leverages machine learning techniques to understand user preferences and predict how different factors, such as time and mode of transportation, impact their overall experience (OPTIONAL)
3. **User preferences:** the time-optimal solution will be improved with user experience, e.g. their preferences about means of transportation (OPTIONAL)

Training data: Google Maps Routing API, factors:

- Average speed of each transportation mode
- Cost (OPTIONAL)

ML Models:

- To be determined

ML Training data:

- Routes acquired from Google Maps API
- Or
- Manually created training / testing data

Details of the project

Means of Transportation Used:

- Walk avg speed: TBD
- Cars avg speed: TBD
- Bus avg speed: TBD
- Cycle avg speed: TBD

For Python code, the **m** parameter will be textual lowercase (“walk” for Walk and so on)

Problem definition

- **problem definition: that a user goes from A to B, using N transportation means**
- **this includes the factor of dividing the A-B path into N subpaths, each can use different transportation means**
- **we would start with the trivial problem of having N=1**

A = Starting Point
B = Ending Point
N = Number of transportation modes

Subpaths: Divide A->B path into N segments each corresponding to a specific transportation mode

Optimization:
- Least Time
- Shortest Distance
- Lowest Cost (optional)
- Least Environmental Impact (optional)

Start with single N=1 transportation mode, compare each transportation mode of N=1 (bike, walk, car, bus), try and combine different subsections to determine the optimal route and modes

Possible Problems/Challenges:
- Number of possible routes increase exponentially with N (number of transportation modes) and number of available options
- Access to data (limited by free tier currently - 1000 google maps API requests per month)
- Real-time factors: Traffic, delays, detours, construction, and unpredictable events impact the optimal pre-planned route

ML Training data

- **using google maps as a source data for training (API+Python)**

1. Create a shared project to enable cooperation
2. Enable APIs: *(simplifying up to discussion)*
    a. Directions API (directions/travel time)
    b. Distance Matrix API (distance/travel time)
    c. Geocoding API (converting addresses to coordinates)
    d. Places API (for finding bus stops and stations)
3. Obtain API key
4. **geopy** library for Python
5. Pandas (data storage and manipulation)

Data Collection Strategy:

- Classify / organize data / naming accordingly
- Generate queries using Google Directions API
- Store data in JSON or PostgreSQL(pgadmin4)

Python function proposal for getDistance

```
import requests
import json
from geopy.distance import geodesic  # For direct distance calculation

def getDistance(A, B, m):
    if m == "direct":
        return geodesic(A, B).meters

    API_KEY =
    base_url = "https://maps.googleapis.com/maps/api/distancematrix/json"

    params = {
        "origins": f"{A[0]},{A[1]}",
        "destinations": f"{B[0]},{B[1]}",
        "mode": m,
        "units": "metric",
        "key": API_KEY,
    }

    response = requests.get(base_url, params=params)

    if response.status_code == 200:
        data = json.loads(response.text)
        if data["status"] == "OK":
            try:
                distance_meters =
data["rows"][0]["elements"][0]["distance"]["value"]
                return distance_meters
            except (KeyError, IndexError):
                pass  # Handle missing data in the API response

    # Return None if API call failed or data was invalid
    return None

# Example Usage:
paradise_coords = (36.0968, -115.1703)
freemont_street_coords = (36.1699, -115.1398)
```

```
driving_distance = getDistance(paradise_coords, freemont_street_coords,
"driving")
walking_distance = getDistance(paradise_coords, freemont_street_coords,
"walking")
bicycling_distance = getDistance(paradise_coords, freemont_street_coords,
"bicycling")
transit_distance = getDistance(paradise_coords, freemont_street_coords,
"transit")

print("Driving distance:", driving_distance, "meters")
print("Walking distance:", walking_distance, "meters")
print("Bicycling distance:", bicycling_distance, "meters")
print("Transit distance:", transit_distance, "meters")
```

Sample output:

```
Driving distance: 10845 meters
Walking distance: 9293 meters
Bicycling distance: 11278 meters
Transit distance: 18632 meters
```

Notes:

- Change to miles or feet?
- Only applies to N=1
- Need to figure out how to implement subpaths

Python function proposal for getTime

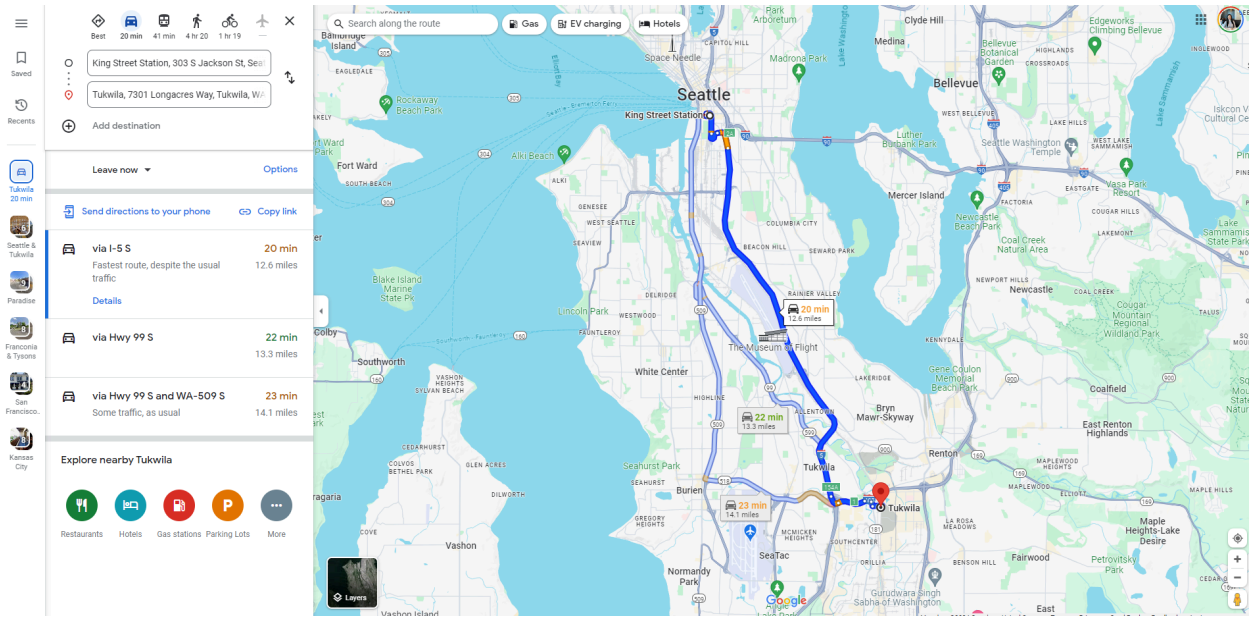
```python
import requests
import json

def getTime(A, B, m):

    API_KEY =
    base_url = "https://maps.googleapis.com/maps/api/distancematrix/json"

    params = {
        "origins": f"{A[0]},{A[1]}",
        "destinations": f"{B[0]},{B[1]}",
        "mode": m,
        "units": "metric",  # Ensures time is returned in seconds
        "key": API_KEY,
    }

    response = requests.get(base_url, params=params)

    if response.status_code == 200:
        data = json.loads(response.text)
        if data["status"] == "OK":
            try:
                duration_seconds =
data["rows"][0]["elements"][0]["duration"]["value"]
                return duration_seconds
            except (KeyError, IndexError):
                pass  # Handle missing data in the API response

    # Return None if API call failed or data was invalid
    return None

# Example Usage:
paradise_coords = (36.0968, -115.1703)
freemont_street_coords = (36.1699, -115.1398)

driving_time = getTime(paradise_coords, freemont_street_coords, "driving")
walking_time = getTime(paradise_coords, freemont_street_coords, "walking")

print("Driving time:", driving_time, "seconds")
```

```
print("Walking time:", walking_time, "seconds")
```

Determine the initial region: the map area on which we'll test the initial design. Must have various and not obvious routes

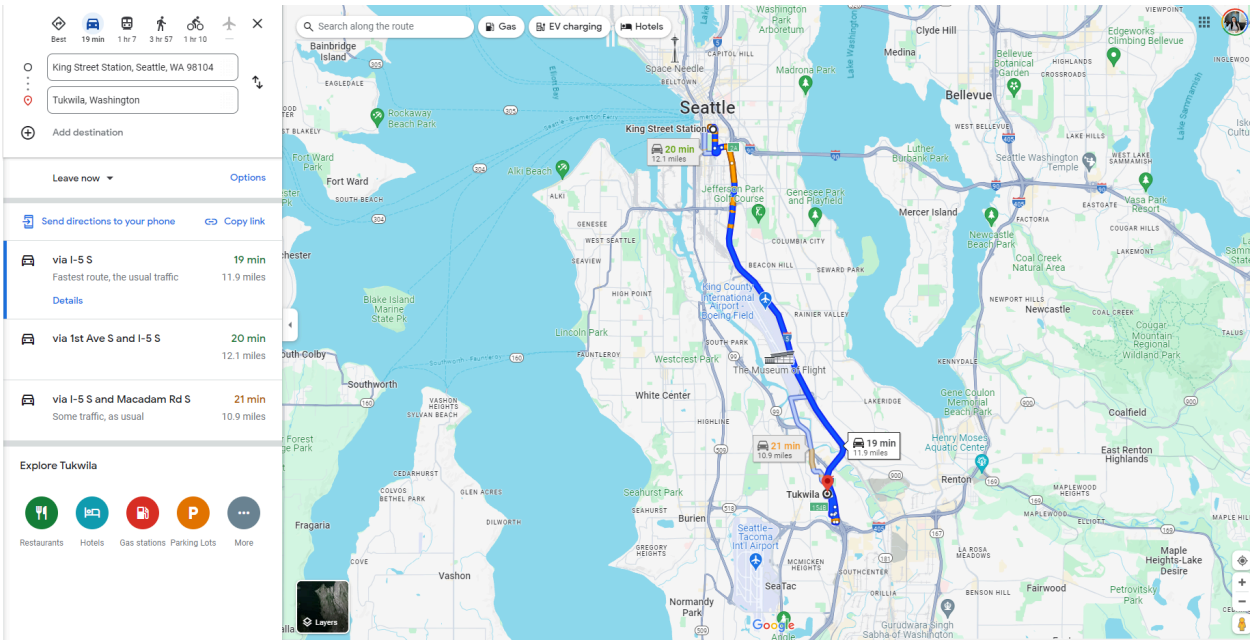
Suggested region: Seattle, Washington
Example: From King Street Station to Tukwila, there are multiple modes of transport:

- Driving
- AMTrack (Train) / Light Rail Station
- Bus
- Walking
- Biking

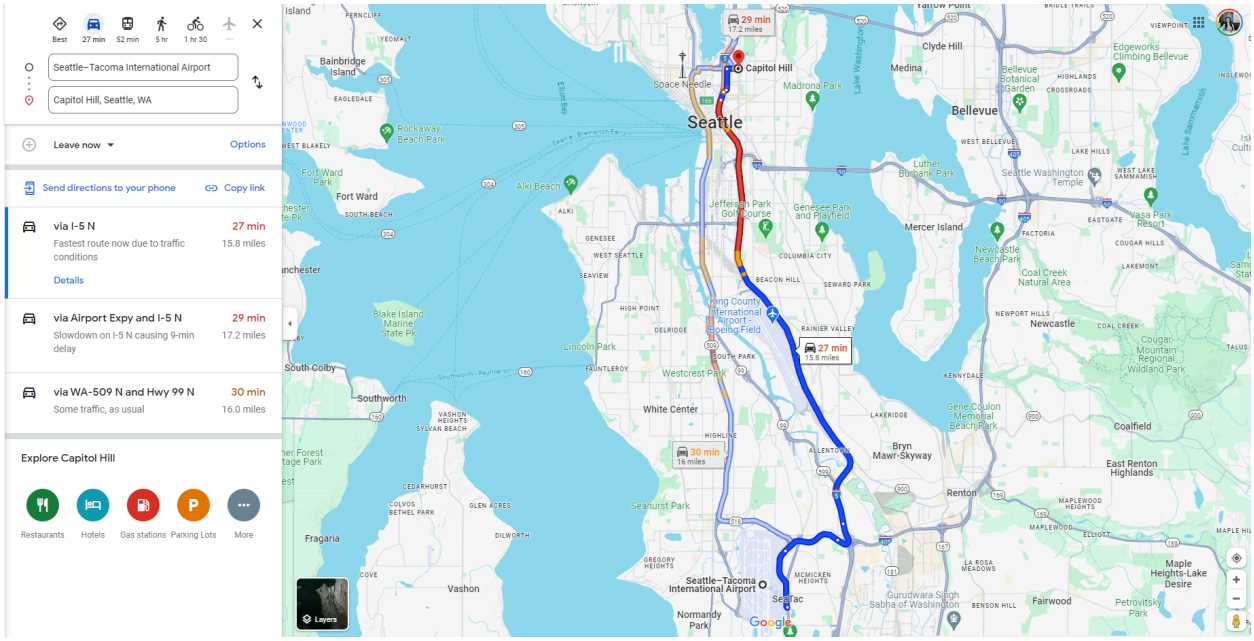


For the initial region, define several sources and destinations

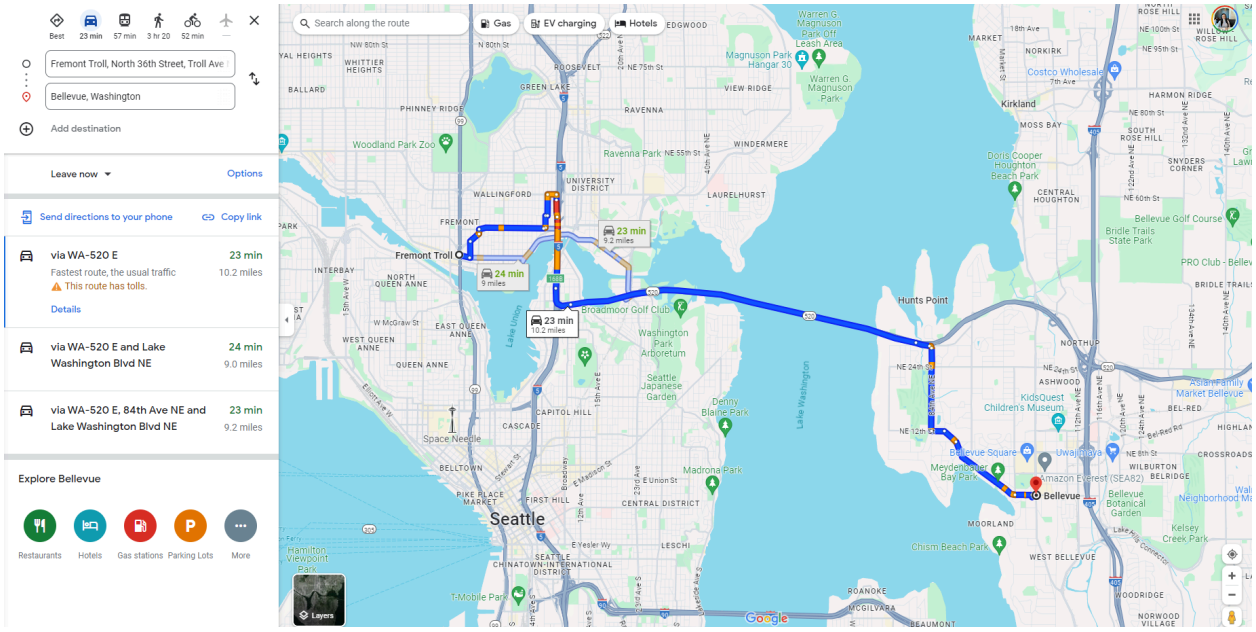
King Street Station to Tukwila



SeaTac (airport) to Capitol Hill (Light Rail Station)



Fremont to Bellevue



For the initial region, define the subpaths - this way also the transportation means

King Street to Tukwila -

- Driving
- AMTrack (Train)
- Walking

SeaTac (airport) to Capitol Hill (Light Rail Station)

- Driving
- Light Rail
- Biking

Fremont to Bellevue

- Driving
- Biking
- Walking
- Bus

Determine if Google API allows to get path for specified time of the day

How to Use It

1. **API Key:** Get a Google Maps Directions API key from the Google Cloud Console.
2. **API Request:** Structure your API request to include:
    - **Origin:** The starting point of your journey.
    - **Destination:** The ending point of your journey.
    - **Departure Time/Arrival Time:** The time you want to start or arrive.
    - **Traffic Model:** Indicate if you want to consider real-time or historical traffic. (Optional)

Example Request (URL Format)

```
https://maps.googleapis.com/maps/api/directions/json?origin=Las+Vegas,+NV&destination=Los+Angeles,+CA&departure_time=1626220800&traffic_model=best_guess&key=AIzaSyBgpzppQdR9sSJDA-MKJS4llKxaAao6h_E
```

- In this example:
    - `departure_time=1626220800` represents a Unix timestamp for a specific time of day.
    - `traffic_model=best_guess` tells the API to use the best available traffic information (either real-time or historical).

Response

The API will return a JSON response with:

- The calculated route(s)
- Travel distance

- Estimated travel time (considering traffic conditions for the specified time)
- Step-by-step directions

Here is my code that implements the departure time, using both my personal addresses

```
import requests
import time

api_key =
origin = "10788+Rivendell+Ave+Las+Vegas+NV+89135"
destination = "11330+Westbrook+Mill+Ln+Fairfax+VA+22030"
departure_time = int(time.time()) + 3600  # Current time + 1 hour (in
seconds)

url =
f"https://maps.googleapis.com/maps/api/directions/json?origin={origin}&des
tination={destination}&departure_time={departure_time}&key={api_key}"

response = requests.get(url)

if response.status_code == 200:
    data = response.json()
    routes = data.get("routes", [])  # Handle cases where "routes" might
be missing
    if routes:
        legs = routes[0].get("legs", [])  # Handle cases where "legs"
might be missing
        if legs:
            distance = legs[0]["distance"]["text"]
            duration = legs[0]["duration"]["text"]  # Get the duration as
well
            print("Distance:", distance, "\nDuration:", duration)
        else:
            print("No legs found in the route.")
    else:
        print("No routes found for the given parameters.")
else:
```

```
    print("Error:", response.status_code,
response.json().get("error_message", "Unknown error"))  # More detailed
error info
```

output :

```
Distance: 2,449 mi
Duration: 1 day 11 hours
```

Criteria Options:

Duration: Time (lowest preferred)

Cost: Using a formula and assumptions applicable to all paths to calculate pricing -

$$Fare = Base\ Fare + (Time\ Rate \times Time) + (Distance\ Rate \times Distance)$$

Base Fare = *$3.00*

Time rate = *$0.40 per minute*

Distance rate = *$0.93* per kilometer or *$1.50* per mile

BASE FARE FOR TRANSIT: $2.50
COST FOR BIKING or WALKING: $0

Physical Effort:
effort_weights = {'walking': 1.0, 'transit': 2.0, 'driving': 3.0, 'bicycling': 3.5}

Environmental Impact:
Biking: duration *= 0.90 10% duration reduction as an incentive
Walking: duration *= 0.90 10% duration reduction as an incentive

Peak hour performance: Evaluates how different routes and modes perform during high-traffic periods.
Biking: duration *= 0.95 5% duration reduction during peak hours
Walking: duration *= 0.95 5% duration reduction during peak hours
Public Transit: duration *= 0.90 10% duration reduction during peak hours
Using Link Light Rail, buses, or the Seattle Streetcar.

Evening Rush Hour:

- Usually 4:00 PM PST to 6:30 PM PST on weekdays
- Particularly congested on major arterials leaving downtown

FORMULA FOR SCORE:

score = total_duration + (total_cost * 60) + (total_effort * 350)

Formulas using variables instead of hardcode:

Cost: $C = BF + (tr * t) + (dr * d)$

Where: BF = Base Fare , tr = Time Rate, t = Time, dr = Distance Rate, d = Distance

Defaults are currently set to:

Base Fare = *$3.00*

Time rate = *$0.40 per minute*

Distance rate = *$0.93* per kilometer or *$1.50* per mile

BASE FARE FOR TRANSIT: $2.50
COST FOR BIKING or WALKING: $0

Physical Effort:
$\{'walking': w, 'transit': T, 'driving': D, 'bicycling': b\}$

Default:
effort_weights = {'walking': 1.0, 'transit': 2.0, 'driving': 3.0, 'bicycling': 3.5}

Environmental Impact and Peak Hour Performance:
$EI = x * dur$
$PH = x * dur$

Where:
x = value to reduce or increase by
dur = duration

FORMULA FOR SCORE: $S = td + (tc * x1) + (te * x2)$
Where:
td = total duration
tc = total cost
te = total effort
x1 = weight for total cost
x2 = weight for total effort
default: score = total_duration + (total_cost * 60) + (total_effort * 350)

FORMULA A(default):

Cost: $C = BF + (tr * t) + (dr * d)$
Base Fare = *$1*
Time rate = *$1 per minute*
Distance rate = *$1* per kilometer or *$0* per mile
BASE FARE FOR TRANSIT: $1
COST FOR BIKING or WALKING: $0

Physical Effort:
$\{'walking': w, 'transit': T, 'driving': D, 'bicycling': b\}$
w = 1
T = 1
D = 1
b = 1

Environmental Impact
$EI = x * dur$
x = 1
for walking and bicycling

Peak Hour Performance:
$PH = x * dur$

Bicycling or Walking:
x = 1

Transit
x = 1

Driving x = 1

SCORE: $S = td + (tc * x1) + (te * x2)$
x1 = 1
x2 = 1

ADDED PEAK HOURS ON TO SEE DIFFERENCE!!!

FORMULA F1:

Cost: $C = BF + (tr * t) + (dr * d)$

Base Fare = *$3.00*

Time rate = *$0.40 per minute*

Distance rate = *$0.93* per kilometer or *$1.50* per mile

Physical Effort:

$\{'walking': w, 'transit': T, 'driving': D, 'bicycling': b\}$

w = 1.0

T = 2.0

D = 3.0

b = 3.5

Environmental Impact

$EI = x * dur$

x = 0.90

for walking and bicycling

Peak Hour Performance(OFF):

$PH = x * dur$

Bicycling or Walking:

x = 0.95

Transit

x = 0.90

Driving x = 1.50

```
                if 'bicycling' in mode_combination or 'walking' in
mode_combination:
                    total_duration *= 0.95
                elif 'transit' in mode_combination:
                    total_duration *= 0.90
                elif 'driving' in mode_combination:
                    total_duration *= 1.50 # 50% increased duration during
rush hour
```

SCORE: $S = td + (tc * x1) + (te * x2)$

x1 = 60

x2 = 350

FORMULA F2:

Cost: $C = BF + (tr * t) + (dr * d)$

Base Fare = *$8.00*

Time rate = *$0.80 per minute*

Distance rate = *$0.80* per kilometer

BASE FARE FOR TRANSIT: $8

COST FOR BIKING or WALKING: $0

Physical Effort:

$\{'walking': w, 'transit': T, 'driving': D, 'bicycling': b\}$

w = 0.0

T = 2.0

D = 3.0

b = 4.0

Environmental Impact

$EI = x * dur$

x = 0.80

for walking and bicycling

Peak Hour Performance(OFF):

$PH = x * dur$

Bicycling or Walking:

x = 0.80

Transit

x = 0.80

Driving

x = 1.6

Driving x = 1

SCORE: $S = td + (tc * x1) + (te * x2)$

x1 = 350

x2 = 60