Your Perfect Day at California Adventure Park
A Disney Park Trip Optimization Problem

Mia Goins, Rhea Kalra, Nikhila Nanduri, Lauren Stampfli

December 2nd, 2024

1 Abstract

This study attempts to create an optimized schedule for a day trip to Disney's California Adventure. To do so, we aim to turn general preferences for a trip to the Disney Adventure Park into an integer programming problem similar to the traveling salesmen problem with considerations towards walking speed, wait times, lightning lane availability, budget, and user preferences. Since the data for the problem is relatively small in scale, we decided to utilize Gurobi's Optimization Software to solve the program. Ultimately, our program produced day schedules that offer a variety of popular activities in the park, though they do tend to include many cross-over paths.

2 Introduction

When planning any trip, some of the main stressors are budgeting, time management, and event planning. With a place like Adventure Park, with over 25,000 visitors per day, it becomes important to optimize your ability to spend as much time on rides and in attractions and as little time in lines as possible. Disney parks should be full of magic and fun for both children and adults alike, but it becomes harder to appreciate this when most of the day is spent glued to a phone checking for the shortest wait times or nearest rides. Of course, Disney offers packages to supplement these issues; however, for those unwilling to spend additional money on alternative scheduling tools, the only potential guidance for planning are online lists of rides and attractions, nothing personally tailored. That is how, "Your Perfect Adventure at California Adventure" was born. This program considers activity preferences, budget, wait times, and walking distances to provide an optimized schedule through the park.

Figure 1: Map of Disney California Adventure Park

3 Formulating the Problem

3.1 Overall Integer Program

We have the following general formulation of the integer program based on the subsequent constraints and descriptions,

$$\min \sum_{i \in V} \sum_{j \neq i \in V} cost(i,j) x_{i,j}$$

subject to

$$\sum_{i \in T} x_{i,j} = y_j \qquad \text{for all } j \neq i \in T$$

$$\sum_{j \in T} x_{i,j} = y_i \qquad \text{for all } i \neq j \in T$$

$$\sum_{j \in T} x_{start\_node,j} = 1$$

$$\sum_{i \in T} x_{i,end\_node} = 1$$

$$\sum_{i \in R} y_i \leq 20$$

$$\sum_{i \in F} y_i \leq 2$$

$$\sum_{i \in A} y_i \leq 36$$

$$\sum_{i \in R} y_i \geq 4$$

$$\sum_{i \in A} y_i \geq 3$$

$$u_i - u_j + 1 \leq (|T| - 1)(1 - x_{i,j}) \qquad \text{for } i, j \in T, i \neq j$$

$$x_{i,j} \leq y_j \qquad \text{for all } i, j \neq i \in V$$

$$u_i \in \mathbb{Z}$$

$$x_{i,j}, y_i \in \{0,1\}$$

3.2 Splitting the Integer Programs

The integer program was initially developed to model a traveling salesmen problem. However, certain customizations are needed to better fit it to navigate the Disney parks. In order to incorporate a lunch location into the problem, we decided to split the program into two sub-problems, a morning and afternoon integer program. This was done in an attempt to accurately model valid eating times and overcome the scheduling issue of inaccurate eating periods. At the end of the first time interval, we greedily choose the next node in the schedule to be a food location, taking into account any user preference for food and budget. This location will then be considered the new starting location for the afternoon integer program. The second integer program includes the additional constraint that the schedule must end at the entrance to the park. Any other food locations desired by the user will be incorporated into the overall integer program.

3.3 Map

We let R denote the set of rides in the schedule, F denote the set of food locations, and A denote the set of attractions where $R \cap F \cap A = \varnothing$ and $T = R \cup F \cup A \subset V$ where V is the set of all vertices in the graph mentioned above.

3.4 Variables

The decision variable indicates the edge between activity i and activity j in the graph created by making edges between every possible activity in the park. This was done to allow us to utilize the traveling salesmen formulation while also creating links between each location to then extrapolate into a schedule.

$$x_{i,j} = \begin{cases} 1 & \text{if the edge } i,j \text{ is included in the schedule} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

We further introduced two "dummy" variables into the problem, accounting for the "dummy" nodes in our graph. Specifically, we will let start_node represent the first node in the graph - the start of the schedule - and end_node represent the last node in the graph - the end of the schedule - and so created the two decision variables,

$$x_{start\_node,j} = \begin{cases} 1 & \text{if the edge } start\_node, j \text{ is included in the schedule} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$x_{i,end\_node} = \begin{cases} 1 & \text{if the edge } i, end\_node \text{ is included in the schedule} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

We also introduced the variables u_i for each $i \in [T]$ where

$$u_i = \text{the number of edges in the schedule until the } i\text{th node is included} \tag{4}$$

This allows us to check for subtours using the Miller-Tucker-Zenlin Formulation of the TSP problem (Miller et al.). Since our integer program is attempting to create continuous tours, this is a necessary constraint.

Finally, we included the variable y_i,

$$y_i = \begin{cases} 1 & \text{if there exists an edge into node } i \text{ in the schedule} \\ 0 & \text{if there does not exist an edge into node } i \text{ in the schedule} \end{cases} \tag{5}$$

This variable was included to allow us to limit the inclusion of activities to a subset of the park activities rather than all possible activities.

3.5 Objective Function

For this problem, we sought to minimize the cost of the schedule according to a personalized cost function. Thus, we created the following general objective function

$$\min \sum_{i \in V} \sum_{j \neq i \in V} cost(i,j) x_{i,j} \tag{6}$$

3.6 Constraints

For this problem, we require two types of constraints: assignment constraints and subroutine constraints. The following assignment constraints ensure that every node in the schedule has an in edge and an out edge. Since $y_i \in \{0,1\}$, this constraint also ensures uniqueness of the edges, and so the issue of repeats or multiple routes through one location does not occur when solving the problem.

$$\sum_{i \in T} x_{i,j} = y_j \qquad \text{for all } j \neq i \in T \tag{7}$$

$$\sum_{j \in T} x_{i,j} = y_i \qquad \text{for all } i \neq j \in T \tag{8}$$

Since we constrain the inclusion of the edges by the variable y_i rather than 1, we needed to add an additional constraint to ensure that y_i changes when an edge is included in the graph. So we added the following constraint.

$$x_{i,j} \leq y_j \qquad \text{for all } i, j \neq i \in V \tag{9}$$

We also needed to require that the first and last node in the schedule only had one edge out and one edge in. This was done using the following constraints:

$$\sum_{j \in T} x_{start\_node,j} = 1 \tag{10}$$

$$\sum_{i \in T} x_{i,end\_node} = 1 \tag{11}$$

The following subroutine constraint from the Miller-Tucker-Zenlin Formulation of the TSP problem ensures that no distinct sub routes will occur in the schedule (Miller et al.).

$$u_i - u_j + 1 \leq (|T| - 1)(1 - x_{i,j}) \qquad \text{for } i, j \in [T], i \neq j \tag{12}$$

We also required that the variables are only assigned the integer values $\{0,1\}$, as is standard,

$$x_{i,j} \in \{0,1\} \tag{13}$$

Beyond the constraints for ensuring a complete and feasible schedule, we added additional specifications according to the problem's use cases. Specifically, in order to ensure that the user visited a reasonable number of rides, attractions, and food locations, we included the following assumptions in our model,

1. You can ride at most 20 rides per half day
2. You stop at at most 2 food stations per half day
3. You can complete at most 36 attractions per half day
4. You must ride at least 4 rides per half day
5. You must visit at least 3 attractions per half day

These assumptions were reflected with the following constraints,

$$\sum_{i \in R} y_i \leq 20 \tag{14}$$

$$\sum_{i \in F} y_i \leq 2 \tag{15}$$

$$\sum_{i \in A} y_i \leq 36 \tag{16}$$

$$\sum_{i \in R} y_i \geq 4 \tag{17}$$

$$\sum_{i \in A} y_i \geq 3 \tag{18}$$

3.7 Cost Function Definitions

For the cost function, we chose to penalize spending more time on any given activity. We also chose to prioritize 'happiness', which is a value given to the program by the user when they first enter their itinerary activities. By penalizing time, we attempt to increase the potential chances for further engagement at the park. We further attempt to differentiate the nodes by including the happiness constraint, adding priority to activities that will be most enjoyable for the user. It is also assumed that the user would generally prefer to save money on the trip. Thus, we included a cost constraint for the additional price of completing any of the activities in the park. The happiness multiplier for the happiness value was included in order to make the value competitive with the time and price constraints.

$$cost(i,j) = time\_to(i,j) + \mu * time\_at(j) + \phi * price(j) - \alpha * happiness(j)$$

where

$time\_to(i,j) =$ the time taken to walk from activity i to activity j

$time\_at(j) =$ time spent at activity j

$happiness(j) =$ happiness value assigned to completing activity j

$price(j) =$ the price of completing activity j

$\mu = 0.5$ if Lightning Lanes used, else 1

$\phi = 1$ if Food Location and cost $\leq$ budget,

infinity if Food Location and cost > budget, else 0

$\alpha =$ happiness multiplier

4 Sourcing Data

4.1 User Preferences

We sourced user preferences in two different ways. For initial testing of the model, we randomly chose the user's favorite activities throughout the park. The second sourcing of user preferences was done through finding existing itineraries online. All activities where assigned happiness ratings between 1 and 300 where the highest ratings where given to the user's most preferred activities. All users were additionally given a base budget of $2000 for the trip.

4.2 Food Prices

We decided to estimate food prices using averages derived from their online price ratings. To do this, we looked up the different food locations and categorized them based on their evaluation of $, $$, $$$, and $$$$. After categorizing them, we created a simple mapping where $ meant the lowest cost and $$$$ meant the highest cost per person.

4.3 Walking Distance and Times

We also needed to estimate the walking distance in minutes between all rides, attractions, and restaurants within the park. Since manually computing the distance between all locations in the park was computationally improbable, we utilized prompt engineering to estimate the distances. Chat GPT references Disney Adventure Land maps and credible sources to estimate walking time in minutes with the assumption that all park goers walk at the rate of 3 miles per hour.

The prompt(s) used were: Given the following rides, attractions , and restaurants in Disneyland please estimate walking distance between all possible combinations. Use approximations based on google images of the Disneyland parks to give me a solid estimate of the walking distance in minutes. While using generative AI to obtain data has its risks, all calculations were looked over manually and verified through use of the Disneyland app and prior research.

4.4 Wait Times

To determine the appropriate wait times for rides and attractions, we created a program that would allow us to extract this information from an active queue times website (Parks). We sent a GET request to fetch the HTML content of the page and then parsed the HTML content using the Beautiful Soup library. Finally, we extracted the titles and created a dictionary to map specific rides to their wait times (code in Appendix 11.3.1).

Additionally, for food locations, we decided to include the time spent eating or in line at that location as its Wait Time. In order to determine the time spent at any given food location, we created a simple mapping from average money spent at the location to approximate time likely spent at that location. As an example, if a single individual were to, on average, spend

$50 at a food spot, this food location is likely a sit down restaurant and thus the time spent at that location is likely around 1.5 hours.

5 Solving with Gurobi

5.1 Overview

We used the Gurobi Optimization Software to solve the program. Since Gurobi requires a license, we obtained free academic licenses using our Andrew emails. We then coded the problem using Python and the Gurobi software (code in Appendix 11.1). We chose to use Gurobi because it utilizes methods for solving integer programs that generally provide the most accurate solutions. When solving with Gurobi, we augmented the model and problem in some significant ways. Since we split the problem into two smaller integer programs, we decided to remove any already visited activities from the activity graph to avoid repeated visits. We also changed the minimization function to promote visiting more nodes. To do so, we subtracted a multiplier times the sum of all y_i in the model. Additionally, we greedily chose the lunch location in between the two programs based on the cost of traveling to that location from the last activity for the morning schedule.

5.2 Results

5.2.1 Single Rider with Significant Budget

This version allows for the use of Single Rider lines - lines that are often shorter and faster.

Model Preferences

In this model, we had the user greatly prefer to visit the rides - ’Toy Story Midway Mania!’, ’Pixar Pal-A-Round’, "Jessie’s Critter Carousel", ’Inside Out Emotional Whirlwind’, "The Little Mermaid: Ariel’s Undersea Adventure", "Goofy’s Sky School", "Jumpin’ Jellyfish", "Golden Zephyr", "Grizzly River Run", "Soarin’ Around the World", "Web Slingers: A Spider-Man Adventure" - food locations - ’Fiddler, Fifer & Practical Cafe’, "Award Wieners", "Schmoozies!", "Fairfax Market", "Pym Test Kitchen" - and attractions - ’Red Car Trolley’, ’Animation Academy’, "Mickey’s PhilharMagic".

Schedule

Morning Schedule:
Park Entrance
Turtle Talk with Crush
Radiator Springs Racers Single Rider
Incredicoaster Single Rider
Silly Symphony Swings Single Rider
Carthay Circle Restaurant
Web Slingers: A Spider-Man Adventure Single Rider
The Bakery Tour
Red Car Trolley

Lunch: Fiddler, Fifer & Practical Cafe

Afternoon Schedule:
The Little Mermaid: Ariel’s Undersea Adventure
Mickey’s PhilharMagic
Grizzly River Run
Lamplight Lounge
Jessie’s Critter Carousel
Jumpin’ Jellyfish
Animation Academy
Port of San Fransokyo Cervecera
Redwood Creek Challenge Trail
Park Entrance

Figure 2: Single Rider Lightning Lane Schedule

5.2.2 Family Trip with Lightning Lane Use

We additionally ran the model while removing the user's ability to access single rider lanes. This model also uses a portion of its budget on Lightning Lanes. For this run of the model, we did not limit the user's preferences to the most significant activities, in doing so, we allowed a slightly greater exploration of the park.

Schedule

Morning Schedule:
Park Entrance
Turtle Talk with Crush
Silly Symphony Swings
Games of Pixar Pier
Carthay Circle Restaurant
The Bakery Tour
Jumpin' Jellyfish
Jessie's Critter Carousel
Grizzly River Run
Red Car Trolley

Lunch: Fiddler, Fifer & Practical Cafe

Afternoon Schedule: The Little Mermaid: Ariel's Undersea Adventure
Mickey's PhilharMagic
Golden Zephyr
Monsters, Inc. Mike & Sulley to the Rescue!
Lamplight Lounge
Incredicoaster
Port of San Fransokyo Cervecerea
Redwood Creek Challenge Trail
Animation Academy
Park Entrance

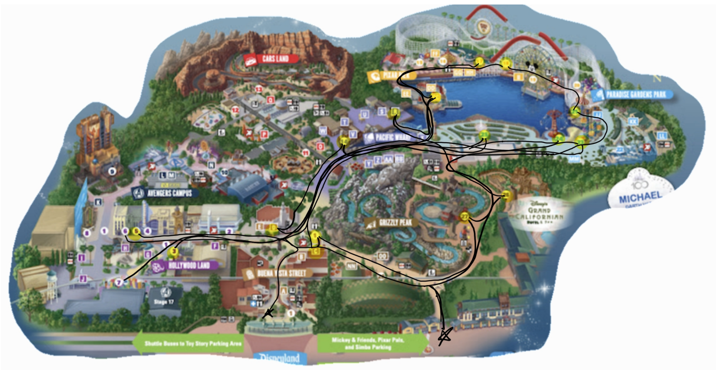

Figure 3: Family Trip with Lightning Lane Use Schedule

5.2.3 Family Trip No Lightning Lane

We additionally generated a schedule for a Family Trip without Lightning Lane use. In this particular park, lightning lanes are limited, thus, these passes are less likely to be purchased. Furthermore, we wanted to account for situations where budgets might not allow for said purchases.

Schedule

Morning Schedule:
Park Entrance
Red Car Trolley
Jessie's Critter Carousel
Jumpin' Jellyfish
Carthay Circle Restaurant
Animation Academy
Silly Symphony Swings
Grizzly River Run
Turtle Talk with Crush

Lunch: Fiddler, Fifer & Practical Cafe

Afternoon Schedule:
Fiddler, Fifer & Practical Cafe
The Little Mermaid: Ariel's Undersea Adventure
Mickey's PhilharMagic
Golden Zephyr
Inside Out Emotional Whirlwind
Incredicoaster
Lamplight Lounge
The Bakery Tour
Port of San Fransokyo Cervecerea
Redwood Creek Challenge Trail
Park Entrance

Figure 4: Family Trip without Lightning Lane Use Schedule

5.3 Analysis

There are differences between all schedules. The singles schedule prefers rides with single rider lanes. This is likely due to their shortened wait time. Comparatively, the family schedules do not visit the Avenger's campus or Cars Land but do prefer slightly different routes likely due to the changed wait times according to lightning lane use. Overall, all of our schedules have a lot of crossover in the center of the park. The likely reason behind this is the influence of the happiness rankings and the minimal influence of walking times and wait times at many of the locations around the park. Thus, some of the far away locations are more preferred before others in similar areas due to a combination of the user's greater preference for those activities and the time spent walking to and completing the activities.

6 Comparison to Online Schedules

In order to further test our model, we compared our schedules with those recommended from reasonable online sources. To do so, we sourced two different schedules from two different sites and ran our model preferring the recommended activities in the sourced itineraries. These itineraries also recommend using the Single Rider lines and Lightning Lanes though they were not necessary.

The first schedule from a Disney Tourist Blog (Bricker, T.) and the second schedule from Wandering Disney Blog (andrewlong7) had a similar structure with both starting in Cars Land, going to Pixar Pier for lunch and then visiting Hollywood Lane. They also both recommended the Cars Land Sh-Boom lighting moment and ending the day with World of Color. For the schedule for the Wandering Disney Blog, after the sixth ride, rides were only recommended if they were under 30 minutes.

Morning Schedule:
Radiator Springs Racers or Web Slingers: A Spider-Man Adventure
Toy Story Midway Mania
Incredicoaster
Pixar Pal-A-Round
Soarin' Around the Word
Grizzly River Run (skip is wait > 30min)
The Little Mermaid Ãriel's Undersea Adventure
Monsters, Inc. Ride & Go Seek
Animation Academy

Lunch: SanFransokyo Square or sit-down meal at Lamplight Lounge

Afternoon Schedule:
Mater's Junkyard Jamboree
Silly Symphony Swings
Golden Zephyr
Snack at Adorable Snowman Frozen Treats
Cars Land Sh-Boom lighting moment
Dinner at Carthay Circle Restaurant
Radiator Springs Racers or Web Slingers: A Spider-Man Adventure (which ever one you didn't do first)
World of Color

Morning Schedule:
Park Entrance
Mickey's PhilharMagic (additional)
Golden Zephyr
The Bakery Tour (additional)
Jumpin' Jellyfish (additional)
Animation Academy
Silly Symphony Swings
Port of San Fransokyo Cervecercva (beer spot)
Pixar Pal-A-Round

Lunch: Carthay Circle Restaurant

Afternoon Schedule:
Aunt Cass Cafe
Redwood Creek Challenge Trail (additional)
Games of Pixar Pier (additional)
Turtle Talk with Crush (additional)
Soarin' Around the World
Lucky Fortune Cookery (additional, subs for the frozen treats)
Incredicoaster
Monsters, Inc. Mike & Sulley to the Rescue!
The Little Mermaid: Ariel's Undersea Adventure
Park Entrance

Figure 5: (Left) Disney Tourist Blog Schedule (Right) Our Schedule

As can be seen in Figure 5, compared to the blog schedule, our schedule does not include Toy Story Midway Mania, Radiator Springs Racers, and Web Slingers but it does add multiple other attractions and food spots. This is likely due to our algorithm guaranteeing 4 rides per half day & 3 attractions per half day which created a more balanced trip.

Morning Schedule:
Radiator Springs Racers
Guardians of the Galaxy – Mission: BREAKOUT! (only if wait < 30min)
Soarin' Around the Word
Grizzly River Run
Incredicoaster
Toy Story Mania (ony is wait < 40min)
Jessie's Critter Carousel
Little Mermaid – Ariel's Undersea Adventure
Goofy's Sky School

Lunch: Lamplight Lounge or in SanFransokyo Square

Afternoon Schedule:
Show at the Hyperion Theater, The Spider-Man Stunt Show, or Carthay Circle
Animation Academy or Mickey's Philharmagic
Monsters Inc. Mike & Sulley to the Rescue (if wait is <15 min)
Snack in the Prickly Pear Soda
Golden Zephyr
Mater's Junkyard Jamboree
Silly Symphony Swings
Luigi's Rollickin' Roadsters
Cars Land Sh-Boom lighting moment
Dinner at Carthay Circle Restaurant
WEB Slingers
World of Color (only if went to WEB Slingers)

Morning Schedule:
Park Entrance
Turtle Talk with Crush (additional)
Silly Symphony Swings
Grizzly River Run
Lamplight Lounge (additional)
The Bakery Tour (additional)
The Little Mermaid: Ariel's Undersea Adventure
Red Car Trolley (additional)
Jessie's Critter Carousel

Lunch: Carthay Circle Restaurant

Afternoon Schedule:
Monsters, Inc. Mike & Sulley to the Rescue!
Incredicoaster
Mickey's PhilharMagic
Golden Zephyr
Pixar Pal-A-Round
Aunt Cass Cafe (additional)
Redwood Creek Challenge Trail (additional)
Animation Academy (additional)
Park Entrance

Figure 6: (Left) Wandering Disney Schedule (Right) Our Schedule

As can be seen in Figure 6, compared to the blog schedule, our schedule does not include Toy Story Midway Mania, Radiator Springs Racers, Guardians, Soarin', Mater's Junkyard, Goofy's and Web Slingers but does include multiple additional attractions and food spots since the model forces a certain amount of each activity.
Our schedule focuses less on rides and more on holistic enjoyment of activities in the park, meaning that most of the day will not be spent in lines but will be spent partaking in other activities. Our schedule does, however, put more emphasis on the user preference towards activities and far less emphasis on the walking distance between activities. As a result, our model tends to output schedules that send the user around the park more compared to the blog's schedule which has some attempts to keep activity completion within a small number of visits to any given area. Regardless, both the blog schedule and our schedule have a significant number of crossovers in the paths through the park.

7 Conclusion

Overall, our model offers a large range of options, though remains comparable to the online schedules. Both the online schedule and our schedule have a tendency to direct the user on crossing paths around the park, likely due to the parks relatively small size and competing cost constraints preferring farther locations that either cost less or are rated more highly by the user over closer locations. Since our schedule also prioritizes popular activities in the park and generates what is considered a full-day itinerary with the ability to customize, the model sufficiently answers the problem we were trying to solve.

8 Future Steps

8.1 Improved Data Gathering

Currently, much of the data used in the models are estimates of the true data. The model could be expanded to different time periods as well as locations, allowing the inclusion of more varied wait times. Additionally, the current model estimates walk times and food costs using averages and best guesses through ChatGPT. A more informed collection of data could result in an improved model. Another significant improvement to our current form of data gathering is using existing ratings for each ride to determine a baseline for the user happiness level. This would both limit the happiness ranking to a standardized scale, 1-10 or 1-5, and give a more informed path through the park to popular areas.

8.2 Solving Model with Heuristics

Rather than using an optimizer such as Gurobi, the model could be solved using common popular heuristics. The Nearest-Neighbor Method, a greedy heuristic for the traveling salesman problem, could be useful since it emphasizes choosing the lowest cost action at every node, which would in turn make for a faster trip through Adventureland. Another important expansion would be to incorporate a heuristic like the Lin-Kernighan algorithm with the existing model. The Lin-Kernighan algorithm takes a model and further improves it by exchanging edges that would result in a reduced tour length (Singh, R.). This has great potential for our current implementation since many of the paths through the parks include crossed edges.

8.3 Expanding Model to Different Parks

Currently, the model only looks at one of Disney's smaller and relatively less visited parks. As a result, the wait times for rides are significantly smaller and the use of Lightning Lanes is limited when compared to some of Disney's larger parks. An expansion of this problem could be to source data from Disneyland or Disney World. That being said, the current problem already has over 300,000 variables when the entire park is considered, so adding more variables has the potential to significantly increase the running time of the Gurobi implementation.

9 References

andrewlong7. (2024, August 7). Disney California Adventure 1-Day itinerary. Wandering In Disney. https://wanderingindisney.com/2024/08/06/disney-california-adventure-1-day-itinerary/

Bricker, T. (2024, February 18). 1-day Disney California Adventure Itinerary. Disney Tourist Blog. https://www.disneytouristblog.com/1-day-disney-california-adventure-plan/

Miller, C. E, R. A Zemlin, and A. W Tucker. "Integer Programming Formulation of Traveling Salesman Problems." Journal of the ACM 7.4 (1960): 326–329. Web.

Gurobi Optimization, LLC. 2024. "Gurobi Optimizer Reference Manual."https://www.gurobi.com.

Midgley, E. (2023, December 13). 2024 Disney California Adventure Map (printable PDF) - plus, map of each land. Mickey Visit - Ultimate Disney Planning Guide. https://mickeyvisit.com/disney-california-adventure-map/

Parks. Queue Times. (n.d.). https://queue-times.com/en-US/parks/17/queue_times

Singh, R. (2024, September 28). Best algorithms for the traveling salesman problem. NextBillion.ai. https://nextbillion.ai/post/algorithms-for-the-traveling-salesman-problem

Winter, S. (2024, October 28). How far will you walk at Disneyland?. Go Informed. https://www.goinformed.net/disneyland-walking-distances/

10 Appendix

10.1 Code for Gurobi Solver

```
from gurobipy import *
import california_adventure_info
# contains all the california adventure specific information that is
    not user's preferences


''' FUNCTION create_tsp_model

creates modified tsp model

ARGs
    cost - Cost matrix
```

```
    num_nodes - Includes Start (0) and End (\in [n])
    ride_indices - list of indicies for rides
    attraction_indices - list of indicies for attractions
    food_indices - list of indicies for food
    alpha - determines how much to value visiting additional nodes
    name - name for model

RETURNs
    model
    x - binary decision variable if edge (i,j) selected
    y - selection variable if node i selected
'''
def create_tsp_model(cost, num_nodes, start_node, end_node,
                     ride_indices, attraction_indices, food_indices,
                     alpha=0.5, name="TSP"):

    model = Model(name)
    x = model.addVars(num_nodes, num_nodes, vtype=GRB.BINARY, name="x")
    y = model.addVars(num_nodes, vtype=GRB.BINARY, name="y")  # selection variable

    # objective function : minimize cost & visit nodes
    model.setObjective(
    quicksum(cost[i][j] * x[i, j] for i in range(num_nodes) for j in range(num_nodes) if i != j)
    - alpha * quicksum(y[j] for j in range(num_nodes)),  # encourages visiting more nodes
    GRB.MINIMIZE)

    # link x and y variables: if node j is visited, y[j] must be 1
    for i in range(num_nodes):
        for j in range(num_nodes):
            if i != j:
                model.addConstr(x[i, j] <= y[j], name=f"link_{i}_{j}")

    # start node constraints
    model.addConstr(quicksum(x[start_node, j] for j in range(num_nodes) if j != start_node) == 1,
                    name="start_node_outgoing")
    model.addConstr(quicksum(x[i, start_node] for i in range(num_nodes) if i != start_node) == 0,
                    name="start_node_no_incoming")

    # end node constraints
    model.addConstr(quicksum(x[i, end_node] for i in range(num_nodes) if i != end_node) == 1,
                    name="end_node_incoming")
```

```python
    model.addConstr(quicksum(x[end_node, j] for j in range(num_nodes) if j != end_node) == 0,
                    name="end_node_no_outgoing")

    # incoming and outgoing constraints for y
    for j in range(num_nodes):
        if j != start_node and j != end_node:
            model.addConstr(
                quicksum(x[i, j] for i in range(num_nodes) if i != j) == y[j],
                name=f"incoming_{j}")

    for i in range(num_nodes):
        if i != start_node and i != end_node:
            model.addConstr(
                quicksum(x[i, j] for j in range(num_nodes) if i != j) == y[i],
                name=f"outgoing_{i}")

    # subtour elimination constraints (MTZ) for intermediate nodes
    u = model.addVars(num_nodes, lb=1, ub=num_nodes, vtype=GRB.CONTINUOUS, name="u")
    for i in range(num_nodes):
        for j in range(num_nodes):
            if i != j:
                model.addConstr(
                    u[i] - u[j] + num_nodes * x[i, j] <= num_nodes - 1,
                    name=f"subtour_{i}_{j}")


    # maximum limits (assumptions)
    model.addConstr(quicksum(y[j] for j in ride_indices) <= 20, name="max_rides")
    model.addConstr(quicksum(y[j] for j in attraction_indices) <= 36, name="max_attractions")
    model.addConstr(quicksum(y[j] for j in food_indices) <= 2, name="max_food_stops") # max 2 because lunch is forced


    # minimum limits
    model.addConstr(quicksum(y[j] for j in ride_indices) >= 4, name="min_rides")
    model.addConstr(quicksum(y[j] for j in attraction_indices) >= 3, name="min_attractions")
    return model, x, y


'''FUNCTION write_schedule
```

```

Helper function that writes the created schedule to given file

ARGs
    model - model
    x - binary decision variables for the model
    nodes - list names of nodes that correspond to the model's node index
    file - file to write the schedule to
    schedule_name - name of the schedule
    start_node - node where the model started (used for traversal purposes)

RETURNs
    None
'''
def write_schedule(model, x, nodes, file, schedule_name, start_node):
    file.write(f"{schedule_name}:\n")

    if model.SolCount > 0:
        visited = set()
        current_node = start_node
        tour = []

        while len(visited) < len(nodes):
            visited.add(current_node)
            tour.append(current_node)

            # find the next node in the tour
            next_node = None
            for j in range(len(nodes)):
                if x[current_node, j].X > 0.5 and j not in visited:
                    next_node = j
                    break

            if next_node is None:
                break  # end of tour
            current_node = next_node

        for node in tour:
            file.write(f"{nodes[node]} is included in the schedule.\n")
    else:
        file.write("No valid solution for this schedule.\n")



'''FUNCTION main

```

```
function that
    - sets user preferences
    - gets all ridess, foods, attractions & establishes indicies
    - creates happiness dictionary
    - creates cost matrix
    - creates and runs morning model (using create_tsp_model)
    - creates and runs afternoon model (using create_tsp_model)
    - writes schedules to files

ARGs
    None
RETURNs
    None
'''
def main():
    # user preferences and budget
    user_budget = 2000
    user_min_happiness = 50 #currently filters nothing out (everyone has a base score of 50)
    fast_pass_cost = 350
    has_fast_pass = False
    is_single = False
    remaining_budget = user_budget - (fast_pass_cost if has_fast_pass else 0)


    users_favorites_rides = ['Toy Story Midway Mania!', 'Pixar Pal-A-Round',
        "Jessie's Critter Carousel", 'Inside Out Emotional Whirlwind',
        "The Little Mermaid: Ariel's Undersea Adventure", "Goofy's Sky School",
        "Jumpin' Jellyfish",  "Golden Zephyr",
        "Grizzly River Run", "Soarin' Around the World",
        "Web Slingers: A Spider-Man Adventure"
    ]
    users_favorites_foods = [
        'Fiddler, Fifer & Practical Ca[U+FFFD], "Award Wieners",
        "Schmoozies!", "Fairfax Market",
        "Pym Test Kitchen"
    ]
    users_favorites_attractions = [
        'Red Car Trolley', 'Animation Academy', "Mickey's PhilharMagic"
    ]


    ALL_RIDES, ALL_ATTRACTIONS, ALL_FOODS = california_adventure_info.get_all_names(is_single)

```

```
    # filter food options by budget
    valid_foods = [food for food in ALL_FOODS if california_adventure_info.get_price(food) <= remaining_budget]

    all_nodes = ALL_RIDES + ALL_ATTRACTIONS + valid_foods


    happiness_dict = california_adventure_info.create_happiness_dictionary(
        users_favorites_rides, users_favorites_foods, users_favorites_attractions,
        all_nodes)

    # only include nodes above some baseline threshold (allows for anti-reqs & stringent schedules)
    min_happiness_threshold = user_min_happiness
    all_nodes = [node for node in all_nodes if happiness_dict.get(node, 0) >= min_happiness_threshold]
    happy_foods = [node for node in valid_foods if happiness_dict.get(node, 0) >= min_happiness_threshold]
    entrance_node = "Park Entrance"
    all_nodes = [entrance_node] + all_nodes

    num_nodes = len(all_nodes)

    # define weights per specification in documentation
    happiness_weight = -1.0
    travel_time_weight = 1.0
    wait_time_weight = 0.5
    price_weight = 0.1

    # create cost matrix
    cost = [[0] * num_nodes for _ in range(num_nodes)]
    for i in range(num_nodes):
        for j in range(num_nodes):
            if i != j:
                travel_time = california_adventure_info.get_travel_time(all_nodes[i], all_nodes[j])
                wait_time = california_adventure_info.get_wait_time(all_nodes[j])
                price = california_adventure_info.get_price(all_nodes[j])
                happiness = california_adventure_info.get_happiness(all_nodes[j], happiness_dict)
                stay_time = california_adventure_info.get_stay_time(all_nodes[j])

                # determine mu based on fast pass availability
```

```
                mu = 0.5 if (has_fast_pass and california_adventure_info.has_fast_pass(all_nodes[j])) else 1.0

                # determine phi based on budget constraints
                if all_nodes[j] in happy_foods:
                    phi = 1.0
                elif all_nodes[j] in ALL_FOODS:
                    phi = float('inf')  # exclude food items exceeding budget
                else:
                    phi = 0.0

                cost[i][j] = (
                    travel_time_weight * travel_time
                    + mu * (wait_time_weight + stay_time) * wait_time
                    + phi * price_weight * price
                    + happiness_weight * happiness)

    # create ride, attraction, and food indices
    ride_indices = [i for i, node in enumerate(all_nodes) if node in ALL_RIDES]
    attraction_indices = [i for i, node in enumerate(all_nodes) if node in ALL_ATTRACTIONS]
    food_indices = [i for i, node in enumerate(all_nodes) if node in happy_foods]

    # morning TSP
    start_node = 0  # Park Entrance node index

    if happy_foods != []:
        best_food = happy_foods[0]
    else: best_food = start_node
    best_food_score = -float('inf')

    for food in happy_foods:
        current_food_score = happiness_dict[food]
        if current_food_score > best_food_score:
            best_food_score = current_food_score
            best_food = food

    selected_food = best_food
    food_node = all_nodes.index(selected_food) if selected_food else start_node

    morning_model, morning_x, morning_y = create_tsp_model(
        cost, num_nodes, start_node, food_node,
        ride_indices, attraction_indices, food_indices, name="Morning_TSP")

```

```
    morning_model.optimize()

    # check feasibility for morning schedule
    if morning_model.status == GRB.INFEASIBLE: # chat wrote this
        print("morning model is infeasible (nuts)")
        morning_model.computeIIS()
        morning_model.write("morning_model.ilp")
        return

    remaining_budget -= california_adventure_info.get_price(selected_food)
    valid_foods = [food for food in ALL_FOODS if california_adventure_info.get_price(food) <= remaining_budget]
    happy_foods = [node for node in valid_foods if happiness_dict.get(node, 0) >= min_happiness_threshold]
    food_indices = [i for i, node in enumerate(all_nodes) if node in happy_foods]

    # afternoon TSP
    afternoon_model, afternoon_x, afternoon_y = create_tsp_model(
        cost, num_nodes, food_node, start_node,
        ride_indices, attraction_indices, food_indices, name="Afternoon_TSP")

    morning_visited_nodes = [i for i in range(num_nodes) if morning_y[i].X > 0.5]
    afternoon_model.addConstr(
        quicksum(afternoon_x[food_node, j] for j in range(num_nodes) if j != food_node) == 1,
        name="start_node_afternoon")

    # ensure no node visited in the morning is revisited in the afternoon
    for node in morning_visited_nodes:
        if node == 0 or node == food_node: # allow revists for park entrance & lunch location (ensures feasibility)
            continue
        afternoon_model.addConstr(afternoon_y[node] == 0, name=f"no_revisit_{node}")

    afternoon_model.optimize()


    # check feasibility for afternoon schedule
    if afternoon_model.status == GRB.INFEASIBLE: # chat wrote this
        print("afternoon model is infeasible (nuts)")
        afternoon_model.computeIIS()
        afternoon_model.write("afternoon_model.ilp")
        return
```

```
    with open("morning.txt", "w") as morning_file:
        write_schedule(morning_model, morning_x, all_nodes, morning_file, "Morning Schedule", 0)

    with open("afternoon.txt", "w") as afternoon_file:
        write_schedule(afternoon_model, afternoon_x, all_nodes, afternoon_file, "Afternoon Schedule", food_node)


if __name__ == "__main__":
    main()
```

10.2 Code for Data Retrieval

10.2.1 Code for Pulling Wait Times

```
import requests
from bs4 import BeautifulSoup
import unicodedata
import pickle

def clean_ride_name(ride_name):
    # remove notes such as "(anonymous user said it was closed, 18 minutes ago)"
    if '(' in ride_name:
        ride_name = ride_name.split('(')[0].strip()
    return ride_name

def clean_wait_time(wait_time):
    # convert "x mins" to integer x
    if wait_time.endswith("mins"):
        return int(wait_time.replace(" mins", ""))
    elif wait_time == "0 mins":
        return 0
    elif wait_time == "-1 mins":
        return -1
    else:
        return -1  # default to -1 for unknown or invalid formats (possible additional filtering here)

def get_rides_and_waits():
    url = 'https://queue-times.com/en-US/parks/17/queue_times'
    all_rides = {}
    response = requests.get(url)

    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'html.parser')

```

```
        for ride in soup.select('.panel-block'):
            ride_name = ride.find('span', class_='has-text-weight-normal')
            ride_name = ride_name.get_text(strip=True) if ride_name else "Unknown Ride"
            ride_name = unicodedata.normalize("NFKD", ride_name).encode("ascii", "ignore").decode("utf-8")
            ride_name = clean_ride_name(ride_name)

            wait_time = ride.find('span', class_='has-text-weight-bold')
            wait_time = wait_time.get_text(strip=True) if wait_time else "Unknown Time"

            if wait_time == "Open":
                wait_time = "0 mins"
            elif wait_time == "Closed":
                wait_time = "-1 mins"

            if "reservation" not in ride_name.lower():
                all_rides[ride_name] = clean_wait_time(wait_time)
    else:
        print(f"Failed to retrieve the page. Status code: {response.status_code}")

    return all_rides


# additional filtering for -1 flags
def extract_missing_rides(all_rides):
    url = 'https://queue-times.com/en-US/parks/17/stats/2024'
    response = requests.get(url)

    missing_rides = {ride_name for ride_name, wait_time in all_rides.items() if wait_time == -1}

    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'html.parser')

        ride_table = soup.find_all("table", class_="table is-fullwidth")[0]
        for row in ride_table.find("tbody").find_all("tr"):
            ride_name = row.find("a").text.strip()
            wait_time = row.find("span").text.strip()

            ride_name = clean_ride_name(ride_name)
            wait_time = clean_wait_time(wait_time + " mins")

            if ride_name in missing_rides:
```

```
                all_rides[ride_name] = wait_time
                #print(f"{ride_name}, {wait_time}")

    return all_rides

def main():
    all_rides = get_rides_and_waits()
    all_rides = extract_missing_rides(all_rides)
    # for ride_name, wait_time in all_rides.items():
    #     print(f"{ride_name}, {wait_time}")
    with open("all_rides.pkl", "wb") as file:
        pickle.dump(all_rides, file)

if __name__ == '__main__':
    main()
```