Optimizing the Theme Park Experience

Maximizing Visitor Satisfaction at Disney's California Adventure

Jordan Huard and Gabriel da Motta

Pomona College, Claremont, CA

ABSTRACT

The goal of this paper is to find optimal paths through the theme park rides in Disney's California Adventure which improve guest satisfaction while minimizing waiting and walking times. To do so, we gathered walking time, waiting time, ride duration and satisfaction data and used it to create a graphical representation of the park. Through careful deliberation on the graph structure, we frame the question as a shortest path problem, employing modified versions of the Bellman-Ford algorithm as the main technique for finding the solution. Using satisfaction data from different groups of people, we find and analyze results from the model, commenting on interesting properties and investigating the relationship between time passed and total satisfaction gained. Finally, we deliberate on the model, discussing main points of the paper while acknowledging limitations and opportunities for future work.

Key words. graph theory, discrete optimization, industrial engineering, user experiences, hospitality and tourism

1. Introduction

In 2018, Disney's California Adventure in Anaheim, California was the state's second busiest theme park with 9.8 million visitors, only surpassed by its neighboring park, Disneyland. This averages out to nearly 27,000 visitors each day, though factors such as time of year, weather, and new attractions certainly make some days busier than others. With only 37 total attractions, this will make for a very crowded park. Going to the park is expensive too. The price of a ticket to the park costs each attendee at least $104 during off-peak season and as much as $149 during peak season. This can cost up to nearly $10 per hour per person just to be inside the park, as long as the visitor stays all day. This is a very large investment for one day of entertainment, especially for families.

There are many different strategies a person or group of people can take when visiting a theme park. Perhaps they choose to rush to the back of the park to beat the more leisurely guests and get on some of those rides first. To help create a possible itinerary, sites like thrill-data.com show the wait times for attractions in 5-minute increments for all dates at most major parks. With the data displayed as a graph, trip-planners can quickly scan the site to find when wait times at a specific attraction are shortest and when these rides should be avoided. This is a great tool but checking multiple dates for all attractions would be very time consuming.

It would be ideal if there were a way to create optimal paths for visitors at a theme park such that they could go on the rides they want while spending the least amount of time waiting in line, essentially doing nothing. With wait times for popular attractions nearing 3 hours, visitors want to make the most out of their time at the park and the money they spent to be there. This is exactly what we set out to do in this paper. If visitors can assign levels of satisfaction to rides throughout the park, an optimal path can be created for them using past wait time data and their desire to experience certain attractions. Personally, we are also interested in finding an optimal path through the park for Jordan's family when they visit for graduation.

In Yue Shen's M.S.E. thesis, *Amusement Park Visitor Routes Design and Optimization*, she explores a very similar problem. Using Disney's Epcot in Orlando, Florida as an example, Shen relies heavily on employee surveys and guest surveys to gather the necessary data and focuses on mixed integer programming as her primary method. Similarly to this paper, her goal is to maximize guest happiness by constructing optimal routes for groups of guests to take throughout the park. Some key factors such as walking times, wait times, and satisfaction levels are not taken from, but also found in Shen's paper. Shen also places many constraints not only on what guests are theoretically able to do, but on what guests actually want to do. She also found very significant results: when guests followed a suggested route, they spent on average 70% less time waiting in line.

2. Methods

In this paper, we focus on providing a solution so guests can be maximally satisfied and not spend too much time waiting in line when it just isn't worth it.

2.1. Data

The data collected for this project refers to 16 rides at Disney's California Adventure. While there are 37 total attractions, many of them are not rides and some of the rides were closed for the period of data collection. Since we want to maximize guest satisfaction while adhering to time constraints, we need to collect data relevant to these two categories. As far as the time-related data goes, we need to find walking times between rides, duration of rides, and wait times for each ride.

The values for ride duration come from the lengths of YouTube videos that show the entirety of the ride. These values were rounded up a bit to account for the time it takes to get

on and off each ride. If the duration of the YouTube video for ride i is Y_i, then the adjusted duration of ride i (in minutes) is set as $D_i = \lceil Y_i \rceil + 2$.

To find the time it takes to walk between rides, we went to Google Maps and entered coordinates to find all walking times between rides that don't have other rides directly in between them. We are then able to find the connection between any two rides in the park by finding the shortest path given these smaller segments of time. Let the walking time (in minutes) between the two attractions i and j using this method be $G_{i,j}$. We also round up on this value to account for crowds and distractions that might slow the visitor down. This value is denoted as $T_{i,j} = \lceil 4G_{i,j}/3 \rceil$.

Wait times vary greatly throughout the day at any theme park. To collect this data, we found wait times recorded for each ride every five minutes on thrill-data.com. We define $W_{i,t}$ as the wait time for ride i at time interval t, where t is measured in minutes. Notice that our original data comes in clumps of 5 minutes, but defining the wait time per minute is beneficial for our model implementation, with the wait times for some specific t being translated to our data at time $\lfloor t/5 \rfloor \cdot 5$. Since this data had to be converted from points on a graph to a spreadsheet format, data entry is very time consuming so wait time data was collected for Nov. 9, Nov. 16, Nov. 23, and Nov. 30. We decided to collect data for Saturdays in November in hopes of getting some data without insanely high variance. Unfortunately, with so few data points for each time interval, the variation is still high. There was quite a lot of temperature variation among the four days we considered, so there was an especially large amount of variation among wait times for Grizzly River Run, a wet raft ride fantastic for hot days yet likely unpleasant during chillier weather. The coefficient of variance is defined by $sd(X)/\overline{X}$. Among the time intervals for this ride, the maximum coefficient variance was 1.245 indicating that the standard deviation was nearly 25% larger than the mean. This deals with our time-related data, but we still need to deal with the satisfaction levels.

Satisfaction Level: a numerical representation of how happy or satisfied someone feels after experiencing a specific attraction

Satisfaction levels are data that are very personal and guest-specific. Therefore, this data will change based on who we are tailoring our algorithm towards. Each visitor in a group rates their expected satisfaction of each ride from 0 to 10 where 0 indicates a ride they would really despise going on and 10 indicates an attraction that is necessary to experience for their visit to be complete. We assume that groups going to the park together would like to stay together, so for use in our algorithm, we take the average of the satisfaction levels given for each ride and then these are used to determine a path for the group. We also decided to have the satisfaction level of ride i to decrease each time the group goes on it. This way the visitors won't spend all their time on just a few rides. Overall satisfaction is likely increased by experiencing more rides rather than going on the same one many times. We define the satisfaction level as $S_{i,n}$ for any ride i where n is the number of times the group has experienced the ride. As explained above, $S_{i,n}$ decreases as n increases. Specifically, we use the linear relation $S_{i,n} - 3 = S_{i,n+1}$. Lastly, the satisfaction levels were collected from the authors and some of their friends and family members.

2.2. Preliminary Graph Model

To get results, we need to convert this data into a graph, and to do so we will use a mostly topographical approach. Consider a satellite top down map of the park, where all rides are visible. We place a vertex on top of each each ride (and one on the entrance) and connect them through edges whenever there is a path in between that does not pass through another ride. This allows us to create a simple graph with 17 vertices and 26 edges. The resulting construction can be seen in **Figure 1**.

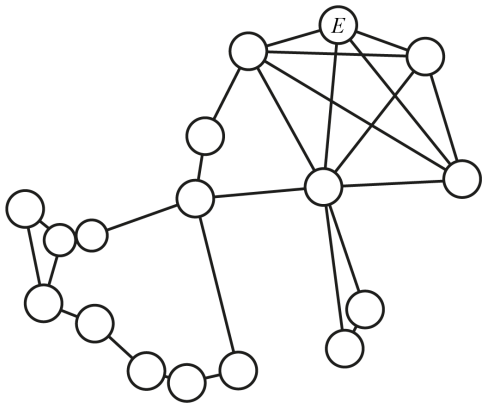

Fig. 1: Simple topographically based graph.

We now determine how the graph holds the meta information our algorithm will require, i.e. ride duration, wait times, walk times and satisfaction levels. For this specific interpretation, we assign time related information to edges. In particular, we assign the walking times to the main edges we see in the graph. For the other information, we consider each vertex as a more complex abstraction. **Figure 2** illustrates this. Here we have inner vertices which feature directed edges. For some particular ride X, the edge going into the sub-vertex has weight of the ride duration D_x added to the line time, given in a function based on the particular time t the ride is being accessed. Meanwhile, the edge coming from the sub-vertex has weight of the ride satisfaction $S_{X,n}$, where n is number of times the ride has been used prior to this usage.

The two most complicated weights come from the line wait time $W_{X,t}$ and the satisfaction $S_{X,n}$ since they can change depending on other factors. Since this graph is being built to fit a computer algorithm, with use a hash map function to fetch the specific wait time or satisfaction for the specific t or n required. Keeping track of time and number of times each ride was used is a responsibility deferred to the algorithm.

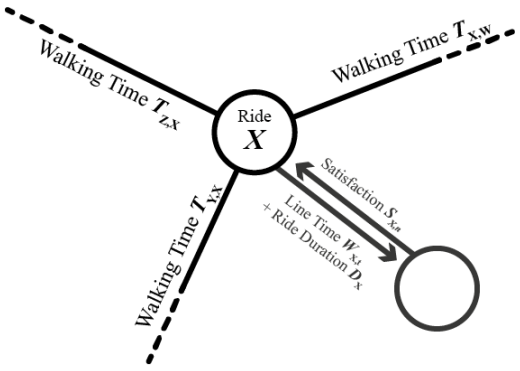

Fig. 2: Meta information in a vertex.

2.3. Optimizing

With our preliminary graph model complete, we move towards solving it for results. The goal is to find the path from the entrance to the last ride we use such that our satisfaction level S_f is as high as possible, all while staying beneath the time limit t_f set by the moment we enter the park t_0 and the moment we exit, all measured in minutes. We also have an additional constraints of having 3 breaks for 1 hour each. We imagine that this time would be used for eating meals, using restrooms, going shopping and any other activity outside of rides. We divide the solving operation in the following steps:

- Finding the shortest walking routes to each available ride from each available ride. This allows us to store the shortest walking times between any given pair of rides, and since walking times are static, i.e. they don't change with time, we do not ever need to recalculate them.
- Consider that the time from t_0 to t_f has $M = (t_f - t_0)$ total minutes. We then restructure our graph, now considering each vertex as M separate vertices, and find the path with highest satisfaction, taking into account the extra constraints of shopping and eating.

2.3.1. Simplified Bellman-Ford Algorithm

To perform the first step, i.e. to find the shortest walking routes, we consider a shortest path problem on a simple weighted graph, were walking times are edge weights. We can solve this problem using the Bellman-Ford algorithm, a $O(|V|\cdot|E|)$ run time process. We are only looking for the time taken and we know we will not find negative-weight cycles, so we can simplify the algorithm. The pseudocode for this simpler Bellman-Ford can be seen in **Algorithm 1**, where *source* is the initial vertex, V is the set of all vertices, E is the set of all edges, and (u, v).*weight* is the weight of edge (u, v). It returns the collection *dist*, where $\forall v \in V$, *dist*$[v]$ is the time taken from vertex *source* to vertex v.

```
function bellman-ford(source, V, E):
    for each vertex v in V do
        dist[v] := +∞
    end
    dist[source] := 0

    repeat |V| − 1 times
        for each edge (u, v) in E do
            if dist[u] + weight[(u, v)] < dist[v] then
                dist[v] := dist[u] + weight[(u, v)]
            end
        end
    end
    return dist
end
```

Algorithm 1: Simplified Bellman-Ford algorithm.

If we run this algorithm once for each vertex in our graph, we find all the shortest paths between any two given vertices. To prevent needlessly repeating this operation, we store these results for the second portion of our algorithm.

2.3.2. Reframing the Model

Our next task is to find the path of highest satisfaction through the rides while spending less time than the time limit we set. This operation is further complicated by the changing wait times at each moment of the day, the decaying satisfaction as you go into the same ride multiple times, and the necessity of shopping and eating throughout the day. What we find is that the current simple graph, featuring only 17 vertices and 26 edges, has a hard time handling this situation, and we must therefore reframe it to fit a shortest path solving algorithm. We tackle this per constraint:

Highest Satisfaction:
This is the core of the problem and the metric we use to frame the graph as a shortest path problem. To do this, we negate the satisfaction and set them as the edge weights for riding any particular rides. Notice that this creates negative cycles, but we handle those in the next item.

Time Limit and Changing Wait Times:
Implementing a secondary and limiting metric in a shortest path algorithm is far more difficult. In our current setup, we have negative cycles that are not compatible with the Bellman-Ford algorithm, so our goal is to plot a new graph where edges flow with time. Here, we change our graph so that each ride is represented by $M = (t_f - t_0)$ vertices instead of just 1. Edges now flow from lower times to higher times, which means they will reach an end at t_f, where the park closes. This also solves the changing wait times, since each vertex represents a point in time and can have its own static wait time. For illustrative purposes, consider a toy example with 3 rides with instant duration and no wait times, where walking from one to the other takes 1 time unit. The reframed graph would look as follows:

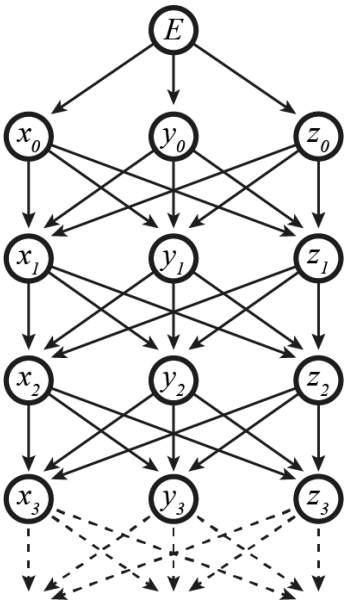

Fig. 3: Toy example with 3 rides.

Of course, if the time to walk, wait in line or the duration was longer than 1 minutes, the edges would shift accordingly. For example, suppose it takes 5 minutes to walk from x to z and that the wait times plus duration at z are continuously 2 minutes. The edge from x_0 to the ride z will now be going to z_7. There is a slight issue where we need the wait times at z_5 to get to z_7, but we deal with this algorithmically during the graphs construction. For our complete model, we originally had 16 vertices and 895 possible minutes (from 8AM to 11:55PM), so now we jump to $16 \cdot 895 + 1 = 14321$ vertices.

Satisfaction Decay:
Properly handling the decaying satisfaction is perhaps the most

difficult task in our model. We originally believed this would make the problem NP-Hard, but we found a way to frame it so that a modified form of the Bellman-Ford algorithm can find the shortest path. The trick is to not only keep track of the predecessor p to each vertex v, but also of a successor s, defined to be the next step in the best path which includes the current vertex v at the current point of execution.

To understand why this helps us, consider a step on edge (u, v) of our shortest path algorithm. If the Bellman-Ford condition passes for vertices u and v, i.e. dist[u] + weight[(u, v)] < dist[v], we initialize another check before updating the distances, predecessors and successors. First we use the predecessors of u and the successors of v to find the path $P_u + P_v$ created if we allow this to be the new best path. We use this path to calculate from scratch its total satisfaction *sat*. Then we find the distance of the last vertex s_{max} that is in the path, which indicates satisfaction sat_0 of the path v is currently part of, i.e. $P_0 + P_v$. If and only if the new path has better satisfaction, i.e. $sat < sat_0$ (recall that satisfactions are negative), we make the change. For illustrative purposes, see **Figure 4**, which shows all the two paths $P_u + P_v$ and $P_0 + P_v$ being considered. Also notice that distance of s_{max} is the distance of $P_0 + P_v$, so more scratch calculation is not needed.

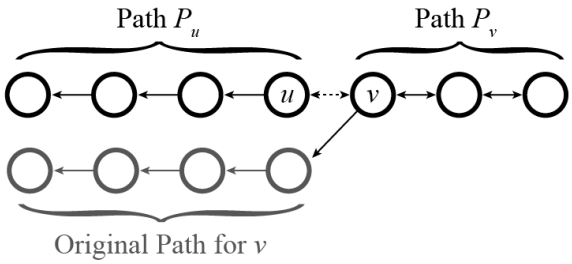

Fig. 4: Paths P_u, P_v and P_0 during a check for u and v.

The correctness of this approach relies on the the successor s of a vertex v always having v as its predecessor, so we must be careful on how to update successors. To ensure this, our update will set the successor of the predecessor of v to nothing, and only then will assign u as the new predecessor of v. We also will set v as the successor of u. The last step to complete this update is to change the distances of each vertex in the successor path of v to their new values, as opposed to just changing the distance of v.

```
for each edge (u,v) in E do
    if dist[u] + weight[(u,v)] < dist[v] then
        P_p := path made by predecessors of u
        P_s := path made by successors of v
        calculate from scratch sat of P_p + P_s
        let s_max be the last vertex of P_s

        if sat < dist[s_max] then
            for each vertex w in P_s do
                P_w := path of successors of v until w
                dist[w] := sat of P_p+ path from v to w
            end
            successor[predecessor[v]] := null
            predecessor[v] := u
            successor[u] := v
        end
    end
end
```

Algorithm 2: Section of modified Bellman-Ford algorithm.

Taking Breaks:
Handling breaks is much easier, but it comes at some computational costs. Our solution is to simply consider breaks as a stellar ride accessible instantly from any other ride. It has a satisfaction of -45 (less than -10) so that it is a must, but the decay on it is much greater, getting its satisfaction to 0 after three breaks in the path. To prevent the algorithm from taking from jumping in time, we need a break ride for each minute. Furthermore, to prevent it from using breaks as a shortcut from one ride to another, we must have a break per ride. This gets us to a total of $895 * 16 = 14320$ break vertices, totalling our model at 28641 vertices.

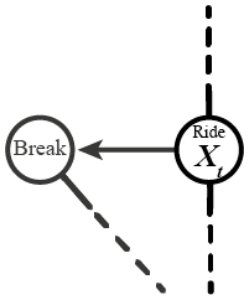

Fig. 5: Depiction of a break vertex related to ride X at time t.

The Bellman-Ford algorithm requires $|V| - 1 = 28640$ iterations to ensure correctness and given the amount of extra computation we do in each iteration, this becomes a long task for even a fast machine to perform. After timing samples of 100 iterations, we found that on average, it takes 2 minutes and 31 seconds for 100 iterations to complete on our experimental setup. This means that to complete the whole 28641 runs we would need roughly 12 hours of runtime.

However, while $|V| - 1$ iterations will ensure correctness, it is only needed for a true worst case scenario, and it is possible to achieve the optimal answer much faster. Since we know a lot about the layout of our graph, we order the edge selection by having the algorithm pick edges from lower timestamps first, progressing temporally through the graph. Through experimentation, we found that for all our test cases, the algorithm performs no changes to its results after iteration 20, and most times it already found the answer by iteration 12. Therefore, we never need to wait more than a minute to find the optimal path for any particular test scenario we use.

3. Results

We will first take a look at the optimal rides-only path for the satisfaction levels of the authors of this paper (see **Table 1**).

It is important to note that these results don't include any time to eat, rest, or shop. If the authors followed this itinerary, they would always either be walking to a ride, waiting in line, or experiencing a ride. From the table, we can see that the authors go on 13 of the 16 rides that data was collected for. There is also quite a bit of diversity in our ride order with the exception of Grizzly River Run and The Little Mermaid early on in the day. This seems to be a very efficient path though as Grizzly River Run gets longer wait times in the afternoon when the temperature increases. It is also efficient because this ride is directly connected to Soarin' and The Little Mermaid, the rides immediately preceding and following it in the itinerary. The Little Mermaid also gets repeated fairly early on likely due to its central location and low wait times in the morning.

Order	Ride Name	Satisfaction	Exit Time
1	Guardians of the Galaxy	6	8:16
2	Monster's Inc	6.5	8:33
3	Soarin'	6.5	9:01
4	Grizzly River	7.5	9:18
5	Grizzly River	4.5	9:33
6	Little Mermaid	6	9:50
7	Incredicoaster	8.5	10:19
8	Little Mermaid	3	10:37
9	Golden Zephyr	2	10:49
10	Pixar Pal	4.5	11:29
11	Midway Mania	7.5	12:35
12	Incredicoaster	5.5	1:15
13	Radiator Springs	9	3:05
14	Monsters Inc	3.5	3:50
15	Mater's Junkyard	3	4:17
16	Incredicoaster	2.5	5:01
17	Midway Mania	4.5	6:21
18	Jessie's Carousel	1	6:35
19	Goofy's Sky School	3	7:11
20	Grizzly River	1.5	7:36
21	Radiator Springs	6	9:20
22	Guardians of the Galaxy	3	10:18
23	Soarin'	3.5	10:52
24	Radiator Springs	3	After Close
Total		111.5	

Table 1: Optimal rides-only path for Gabe and Jordan.

Radiator Springs Racers is located at the far end of the Cars Land section of the park. Due to its distance from central points in the park and its long wait times throughout the day, a lot of time must be spent to ride it. Because of this problem, the algorithm uses a fantastic strategy by getting in line for this ride just barely before the park closes. Per the park's policy, anyone in a line for a ride at closing time will be allowed to go on that ride one last time.

It seems that maybe we should start by going to Radiator Springs Racers since the wait time is only small at the very start of the day. However, when we forced the path to visit this attraction first, we still arrived at a satisfaction level of 111.5, the same one we found without starting at Radiator Springs. Since the satisfaction levels begin to approach zero towards the end of the day, this might mean that we still get to experience the same rides with the same levels but just in a different order. We also experimented with spending only six hours at the park, but when we forced the path to visit Radiator Springs first, we actually got a worse total satisfaction level (77.5) than we didn't force this (81). Though it seemed that we should do this, this is not always the best strategy to take.

Another interesting relationship to consider is how our accumulated satisfaction level changes over the time spent at California Adventure. In **Figure 6**, we can see that the rate of this change is very high for the first few hours of our stay at the park. By the afternoon, this rate slows down considerably. Using R, we have also added a piecewise pair of linear approximations for satisfaction levels that fit the data very well. We can write these approximations as:

$$f(x) = \begin{cases} 0.3093x + 1.6980 & 0 \le x < 160 \\ 0.07707x + 41.91768 & 160 \le x \le 900 \end{cases}$$

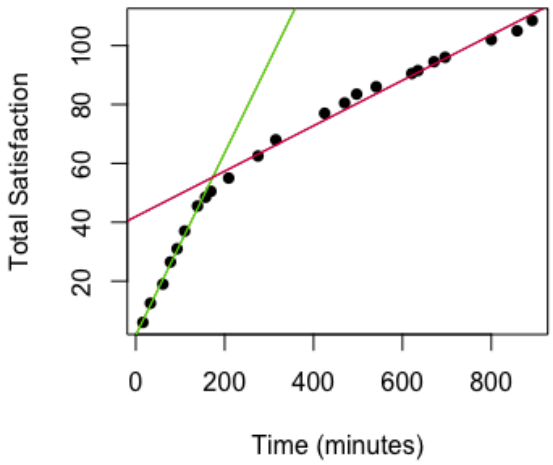

Fig. 6: Approximating satisfaction accumulation for authors (rides only).

where $f(x)$ is the satisfaction level and x is the number of minutes the park has been open. This also assumes that the guests are in the park for the whole day. According to these approximations, the guests gain about 18.5 satisfaction points per hour on average in the early morning. Afterwards however, wait times increase and satisfaction gain potential decreases as riders have already been on quite a few rides. As a result, the guests gain about only 4.6 satisfaction points per hour from 10:40 AM until close. This rate of efficiency is less than 25% of the rate experienced in the morning! The average satisfaction level per ride during the first period is 6.0625 compared to 3.9375 during the second period, so part of this statistic is likely due to the limited options of high-satisfaction rides. However, the significantly longer wait times also play a very big role here.

We are also interested in results for the same group of people but where they also have time allotted in the day to eat meals and take other breaks. These results are shown in **Table 2**.

Contrary to the first set of results, this itinerary does provide a few opportunities to eat, rest, or shop in the form of three one-hour breaks. Due to the addition of the breaks, however, we only get to go on 11 unique rides throughout the day based on our assigned satisfaction levels. We don't get to go on Jessie's Critter Carousel or Goofy's Sky School, but they weren't very satisfying anyway.

Interestingly, the beginning of this path doesn't visit rides that are very close to each other like we do in the first model. After moving to Incredicoaster from Soarin', we visit Grizzly River Run instead of spending more time at Pixar Pier, where attractions are clustered together. The itinerary instructs the guests to stay in areas later on in the path though. For example, steps 8-12 are all around Pixar Pier, steps 14-16 are in Hollywood Land, steps 17-19 are fairly centered, and we finish off the day in Cars Land. While the path might not look as efficient as the path in the first model, it is important to get to Soarin' and Incredicoaster early in the day as the lines get significantly longer by the afternoon.

Just as we did for the first model, we also plot the points and find linear approximations for sections of our data using R. This

Order	Ride Name	Satisfaction	Exit Time
1	Soarin'	6.5	8:17
2	Incredicoaster	8.5	8:45
3	Incredicoaster	5.5	9:03
4	Grizzly River	7.5	9:24
5	Mater's Junkyard	3	9:41
6	Break	–	10:41
7	Little Mermaid	6	10:59
8	Golden Zephyr	2	11:11
9	Break	–	12:11
10	Midway Mania	7.5	1:20
11	Pixar Pal	4.5	2:07
12	Incredicoaster	2.5	2:59
13	Soarin'	3.5	4:02
14	Monsters Inc	6.5	4:44
15	Guardians of the Galaxy	6	5:42
16	Monsters Inc	3.5	6:22
17	Grizzly River	4.5	6:47
18	Little Mermaid	3	7:11
19	Grizzly River	1.5	7:29
20	Radiator Springs	9	9:15
21	Break	–	10:15
22	Radiator Springs	6	After Close
Total		97	

Table 2: Optimal path (including breaks) for Gabe and Jordan.

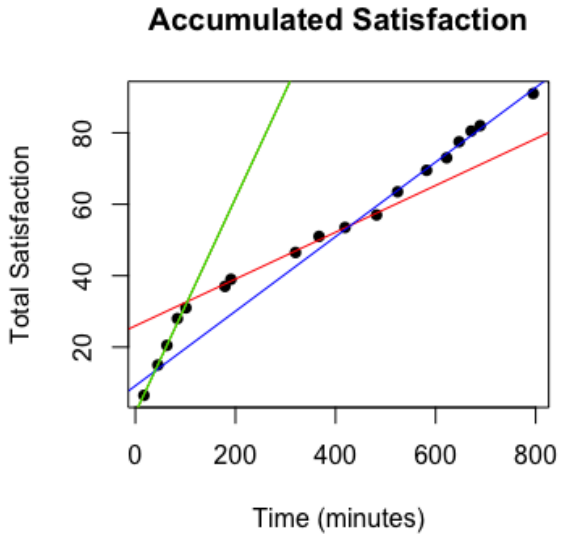

Fig. 7: Approximating satisfaction accumulation for authors (including breaks).

is seen in **Figure 7**. From the scatter plot itself, we can see that the satisfaction level rises quickly in the first couple hours of the day before growing much more slowly throughout the afternoon. Then as evening comes around, it starts to grow a little faster. We can represent this with a 3-part piecewise function:

$$f(x) = \begin{cases} 0.3005x + 1.5713 & 0 \le x \le 101 \\ 0.06548x + 25.96389 & 101 < x \le 482 \\ 0.1043x + 9.1903 & 482 < x \le 900 \end{cases}$$

where $f(x)$ is the satisfaction level and x is the number of minutes the park has been open assuming the guests have been there the whole time. The algorithm forces the program to choose three one-hour breaks, but it seems that there should be more breaks in the afternoon since that's when we make the fewest satisfaction points per hour. We make 18 satisfaction points per hour until around 9:40 AM, followed by a measly 3.9 points per hour until around 4:00 PM where we make about 6.3 points per hour for the rest of the night. The limited efficiency in the afternoon is most likely due to increased wait times since the efficiency picks up later in the evening, but the fact that we wait until the end of the day to go on Radiator Springs may also play a part in this, giving the average satisfaction gain in the evening a hefty boost. The sharp drop in satisfaction gain also immediately follows a mandatory one-hour break which is interesting.

Lastly, we want to implement our algorithm to create a useful potential itinerary for a group that will actually be going to Disney's California Adventure in May: Jordan's family. After doing a bit of research on the rides, both parents contributed their expected satisfaction levels in hopes of finding a path that will maximize their experience. The path (with breaks) is given below in **Table 3**.

Order	Ride Name	Satisfaction	Exit Time
1	Soarin'	8.67	8:17
2	Monsters Inc	8.67	8:37
3	Monsters Inc	5.67	8:51
4	Mater's Junkyard	3.67	9:07
5	Break	–	10:07
6	Little Mermaid	8	10:24
7	Little Mermaid	5	10:38
8	Little Mermaid	2	10:52
9	Incredicoaster	7	11:27
10	Midway Mania	8	12:34
11	Break	–	1:34
12	Incredicoaster	4	2:20
13	Jessie's Carousel	3	2:33
14	Mater's Junkyard	0.67	3:07
15	Radiator Springs	9.5	5:03
16	Soarin'	5.67	6:13
17	Grizzly River	6.67	6:32
18	Grizzly River	3.67	6:47
19	Jumpin Jellyfish	1	7:11
20	Pixar Pal	5.5	7:53
21	Luigi's Roadsters	3	8:30
22	Break	–	9:30
23	Monster's Inc	2.67	9:59
24	Guardians of the Galaxy	3.67	10:54
25	Radiator Springs	6.5	After Close
Total		112.2	

Table 3: Optimal path (including breaks) for Jordan's family.

There are a couple of patterns of immediate interest in this generated path. First, there are three clusters of consecutive re-

peated rides, which is the most we've seen in any path so far. This is likely due to the especially high ratings of two of these rides (Monsters Inc and The Little Mermaid) and their relatively short wait times. Also, the path between rides is very direct up until step 14 when the path goes to Cars Land from Pixar Pier. The guests are told to not skip any rides that are on the way to their next destination until step 10 where Jessie's Critter Carousel is temporarily skipped but revisited at step 13. For at least the first half of the path, walking time is held to a very low minimum which is very beneficial in terms of the rate of satisfaction gain. We'll take a closer look below in **Figure 8** where we will examine patterns on the scatter plot as we've done before.

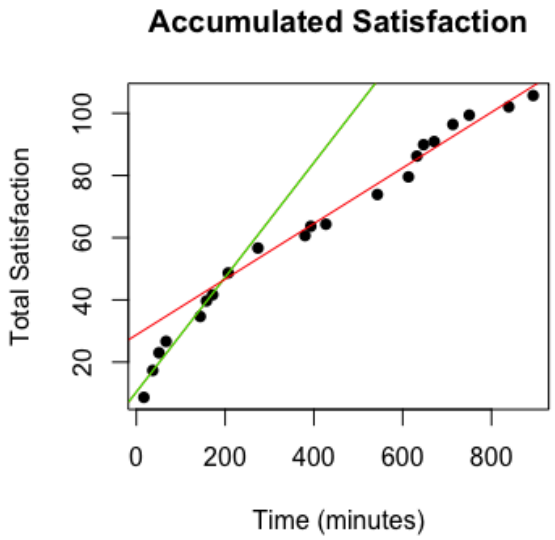

Fig. 8: Approximating satisfaction accumulation for Jordan's family (including breaks).

Our linear approximations don't fit the data quite as well for this model. Looking at the scatter plot, there are a few instances where the rate of satisfaction gain is high and then decreases so a linear model isn't ideal. Nevertheless, this piecewise function will still serve to show the significant differences between rates of satisfaction gain in the first few hours and later on in the day. The function is:

$$f(x) = \begin{cases} 0.1843x + 10.3881 & 0 \le x \le 207 \\ 0.08944x & 207 < x \le 900 \end{cases}$$

where $f(x)$ and x are defined as before. Up until around 11:30 AM, the average satisfaction gain is around 11 points per hour. For the rest of the day, the group will gain a little over 5 points per hour. The difference between the rates is not as significant as in other graphs, but the same pattern applies where the rate of satisfaction gain is quite steep at first.

4. Discussion

Through use of the algorithms outlined in this paper, we were able to generate paths that we strongly believe to be optimal. Even though intuitive strategies didn't always match our models, we found that the algorithms always produced results that were at least as good as our intuitive strategies. In this section, we'll discuss a few main points from the paper and share some ideas about what could be added or done differently in future work on this project.

One key takeaway from the results is the early decline of the satisfaction gain rate. This happened for all three paths we found and is likely so prominent due to the way that we handle the satisfaction decay. This brings up two potential questions. First, assuming this is how satisfaction levels should be handled, then is the majority of the day at California Adventure actually worth it? Of course, tickets are good for a whole day and the total satisfaction increases throughout the day, so it is clearly worth staying. However, it is possible that it would be a better investment to spend an extra $50 for a Park Hopper ticket that will grant access to both California Adventure and Disneyland. If this were to happen, fewer rides would be repeated and guests could obtain a much higher satisfaction level over the course of their stay. Would a one-day Park Hopper be better than spending two days and getting a one-day park ticket for each park?

In general, it is also very difficult to assign numerical values to guest satisfaction. There is the possibility that making the satisfaction decay linear is not appropriate, and there may be great variance in how future guests assign satisfaction levels to rides. While we defined $S_i = 0$ to represent a ride that the guest does not want to ride and $S_i = 10$ as a must-experience attraction, assigning numerical values between 0 and 10 is fairly arbitrary and changes based on the guest giving these ratings. For example, Jordan's dad gave Guardians of the Galaxy a satisfaction level of 5 even though he doesn't actually want to go on the ride. The quality of the generated path is very dependent on the consistency with which future guests give satisfaction levels.

Among the results, we saw that a common strategy in the path was to go on a ride multiple times in a row. This is especially visible in **Table 3** when the path suggests going on The Little Mermaid three times consecutively. The same exact events happen every time someone goes on this ride so it would be rather boring to go on this so many times in a row. To deal with this, it would be ideal if the satisfaction level for a ride decreases if it was recently on the path. The satisfaction level could then increase after the guest goes on a couple of other rides.

In both models involving breaks, breaks tended to happen around the same time. The three breaks were scheduled for morning, noon, and quite late at night. It would be nice if one break happened around dinnertime so the guests don't get hungry, but perhaps the break is delayed so guests can take advantage of shorter wait times while others are eating dinner. We wanted to implement a second type of break that would ensure guests have time off to eat near both lunch and dinner, but this was difficult to include in the algorithm. Ultimately, we chose to give three breaks that can be used for eating, using the bathroom, shopping, or just leisurely exploring the park.

While rides tend to be the most exciting part of California Adventure for most visitors, the park also hosts a plethora of shows, character meet and greets, and other interactive attractions such as animation lessons and backlot tours. These attractions all happen at specific times and we did not include these in our model because the number of attractions would more than double. We also wanted to focus on working around wait times and these attractions don't have wait times. If a guest were to use an itinerary we designed for them, they could check attraction times and replace a ride they don't care too much about with that experience.

Lastly, it would be ideal to collect wait time data for more than four days, but the data collection process is very time consuming. If we had the data for 30 Saturdays rather than 4, we

would have smaller variance and be much more confident about our mean wait times that are a key component in our model.

References

Craven, Scott. "Yes, Disney Theme Parks Are Getting More Crowded. See Which One Got The Most Visitors." *Azcentral*, Arizona Republic, 24 May 2019.

Shen, Yue. "Amusement Park Routes Design and Optimization." The University of Texas, 2012.

"Theme Park Tickets." *Disneyland Resort*, https://disneyland.disney.go.com/-tickets/.