DISCOVERING AN IDEAL PLAN FOR GETTING THE MOST OUT OF A TRIP TO DISNEYLAND

Wen Lin Tan, Dongjin Li, Qiao Liu and Wilbert Lam
Department of Mathematics, University of Washington - Seattle

INTRODUCTION

Disneyland: 'The Happiest Place on Earth'. Thanks in part to some great nostalgia (and nifty marketing), Disneyland has indeed attracted millions of children around the world in search for some of the everlasting joys of Mickey Mouse, Disney Princesses, and Johnny Depp on a pirate ship. All the kids want to do is run around and do everything they can, but for the parents, they need a bit more planning. The lines are long, the rides take time, and there are too many rides in Disneyland to go to in one sitting. But thankfully for the parents, this paper intends to alleviate that problem and provide a cure to their migraines. Finding a good route that lets everyone experience the most possible rides, while minimizing the wait times and avoiding riding the same ride continuously, keeps everyone occupied and excited. This will be the goal of this paper.

Now the tools we will use in this analysis will primarily be integer programming, with some graph theory to help us model the rides and the paths in between. We can assume each ride is a vertex in a graph, with the edges being the path it takes to get in between any two rides. With this in mind, our initial problem can be investigated by discovering a circular path in which no vertex appears more than once in the said path.

BACKGROUND

An early idea for this project was set to optimizing the winnings from a carnival based off different games and stands. It was difficult to find a carnival with the information we needed, so we eventually came to the conclusion that Disneyland (while not exactly a carnival) could be a good and accessible location to use. From there we decided that because Disneyland focused more on enjoyability rather than carnival winnings, it would be reasonable to maximum the number of rides we could go on instead, in a certain amount of time without repeating rides. Thanks to the power of Google Maps, we were able to find distances and times between rides.

Of course, the idea of traveling between nodes of 'rides' brings the classic question of the traveling salesperson problem into play. In the traveling salesperson problem, the goal is to hit all the different cities without repeat and then finish at the starting city. While the traveling

salesperson problem intends on hitting all stops (nodes) in the cheapest/fastest way possible[4], our problem focuses on any arbitrary number of rides in the fastest time possible. Nevertheless, very similar basic principles apply to both problems.

The origins of the traveling salesperson problem (abbreviated as TSP from this point on) are unclear, but a similar situation was analyzed by two mathematicians, Sir William Hamilton and Thomas Kirkman[3]. The two men were heavily involved in graph theory, and they came up with a game called the Icosian Game in which one tried to find a Hamiltonian cycle with all the available points. However, the practical uses were not fully considered until around 1930 when Merrill Flood attempted to mathematically formulate the TSP to aid in school bus routing. Finding the optimal solution to the TSP has proved extremely fickle and numerous algorithms have been tried ranging integer linear programming and plane cutting to dynamic programming. Despite this, the TSP remains an NP-hard problem (non-deterministic polynomial time hard) and any algorithms worst case scenario still remains around 1.5 times worse than the optimal solution. For almost 40 years, the worst case scenario for the TSP was indeed set at 1.5 times the optimal solution by Professor Nicos Christofides of Imperial College London and only recently was this one-upped by a team of scientists at the Stanford and McGill universities by a mind-blowing 'four hundredths of a trillionth of a trillionth of a trillionth of a trillionth of a percent'[2]. Of course, this ridiculously tiny margin of improvement really only further acknowledges the difficulties of solving the TSP and why it has continued to confound and fascinate people over the last two centuries.

Fortunately, the TSP has many variations, some which are much more manageable to investigate and solve. This leads us to our Disneyland trip planning problem.

GENERAL IDEA: This is a Traveling Salesperson Problem Scenario in Disney Land. We want to maximize the number of rides we are taking starting with a fix location. Here, we declare the fix location as the entrance. It is also known as vertex 1. We also have set that the ending point needs to be the starting point. This is a reasonable assumption because assuming that we are going to Disney Land with a few groups of people, we set the entrance to be the meeting point at a certain time.

TERMINOLOGY:

Throughout this paper, we will be using these terminologies

1. Vertex is a destination. Vertex is represented by v. It will be in binary. $v_i = 1$ if in path, else $v_i = 0$.
2. Edge describes the connection from a location to another different location. Edge is represented by x. $x_{i,j}$ = 0 when i = j, because this describes an edges coming from vertex j to j, which does not happen in out condition set, as we do not set to be able to visit a location more than once. It will be in binary. $x_{i,j}$ = 1 if edge of from vertex i to vertex j is taken.

$x_{i,j}$ =0 if edge of from vertex i to vertex j is not taken.

3. Degree of a vertex is the number of edges connecting a vertex. Degree is represented by d.
4. N is the number of vertices. We will set N to be a number in the solution section but, we calling it N here for generalization.
5. We collected the time taken to walk from one ride to another for each ride we choose using Google map, which is essentially the time variable $t_{i,j}$. This information is attached in the Appendix Section under Walking Time (p19 – p 20). Since we choose 13 rides in total, then the graph we construct has 13 vertices and (13*13 - 13) = 156 edge variables. It has 156 edge variables as $x_{i,j}$ = 0 when i = j. It is trivial to include them in the model.

Location	Vertex	Wait Time	Ride Time	Combined Time
Main Street (Treated as entrance)	v1	0	0	0
Indiana Jones	v2	43	10	53
Pirates of the Caribbean	v3	18	15	33
Pirates Lair	v4	25	45	70
Haunted Mansion	v5	15	9	24
Buzz Lightyear	v6	18	4.5	22.5
Finding Nemo Submarine	v7	40	13	53
DisneyLand Monorail (near Nemo)	v8	10	15	25
Autopia	v9	20	4.5	24.5
Alice in Wonderland	v10	34	4	38
Mad Tea Party	v11	10	1.5	11.5
Mickey's House	v12	23	5	28
The DisneyLand Story	v13	30	1	31

Table 1.0: contains information about the vertex notation and the total ride[5] and wait time[6]'

.

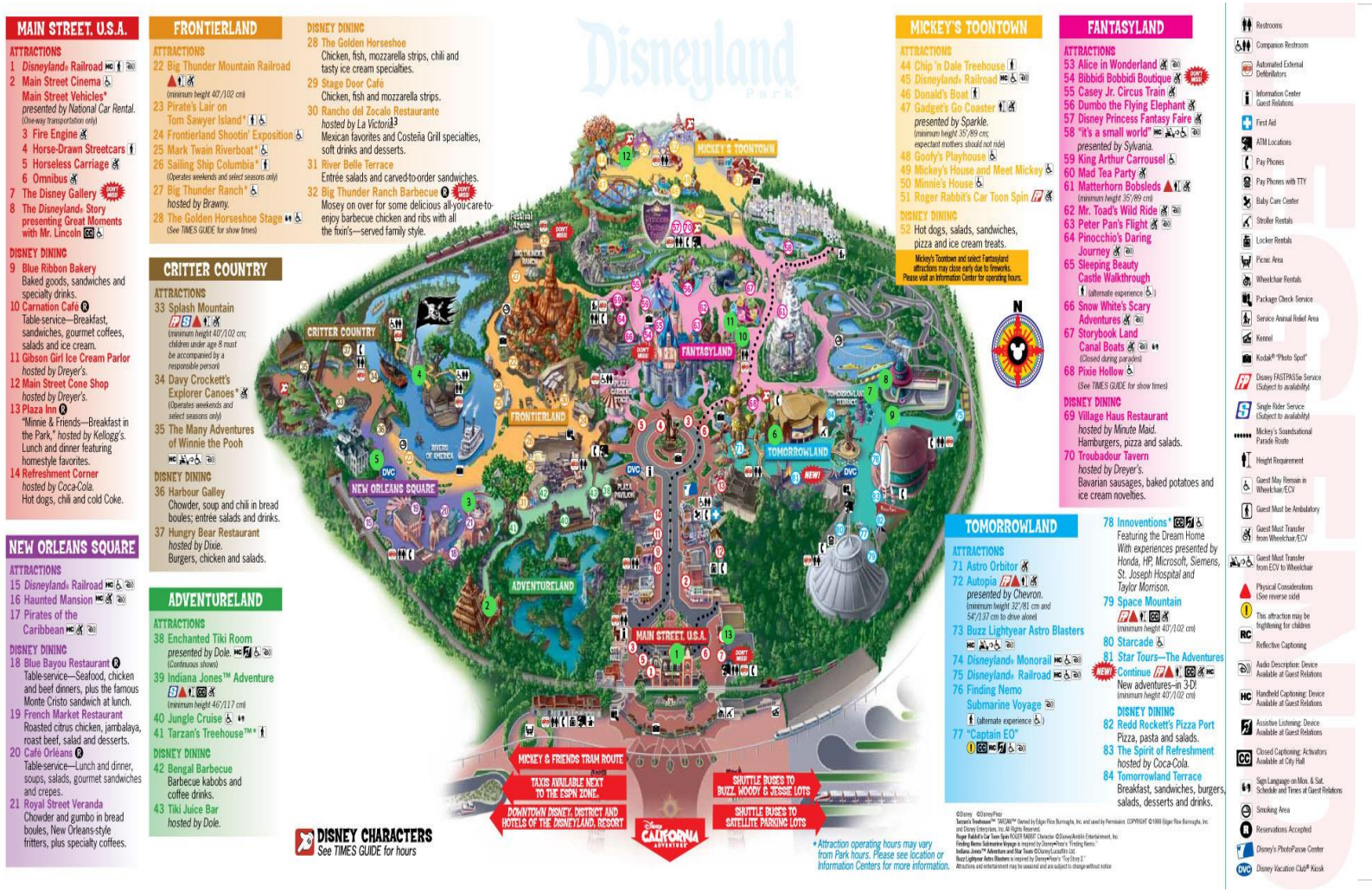

Figure 1.0: The location of the vertices is marked and label with green dots.

OBJECTIVES:

1. As described in the general idea, we have set the entrance (vertex 1) as our starting and ending point. On a practical level, people usually return to the starting point, like entrance, parking location, airport, etc. Therefore, out vertex 1 needs to have two degrees.
2. We limit ourselves that we need to pass through as much vertex as possible without returning to the previous vertex. For example, if we have gone from Pirates of the Caribbean to Indiana Jones, we cannot go back to Pirates of the Caribbean.
3. We set our path to be a circle. For clearer illustration, $1 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$. This means for every vertex, we have been through needs to have degree of 2. The path is not broken.
4. We want to maximize the number of edges in Disney Land.

$$\sum_{i=1}^{N} \sum_{j=1}^{N} x_{i,j}\,, where\ i,j = 1,2 \dots N \qquad \text{Eq. 1.0}$$

CONSTRAINTS:

All the constraints are implemented in Java to produce a text file. [Refer to Appendix: Code in java to generate a text file for LP Solve p21-p 25]
In the constraints, $i \neq j$ is described in Terminology (2).

- Constraint 1 (Time Constraint in Disney Land)
  $$\sum_{i=1}^{N} \sum_{j=1}^{N} t_{i,j}\, x_{i,j} = Total\ Time\ in\ DisneyLand,\ i,j = 1,2 \dots N \qquad \text{Eq. 1.1}$$
  $t_{i,j}$ is the total time taken to walk from vertex i to vertex j, ride time and waiting time in vertex j
- Constraint 2 (Constrain Degree Number and Vertex with Degree Number)
  $$d_1 = 2 \qquad \text{Eq. 1.2}$$
  This special case is due to our Objective (2) to set vertex 1, the entrance as the starting and ending point. This implies there are 2 different edges connecting to vertex 1.
  $$v_i \leq \ d_i\,, i,j = 2 \dots N \qquad \text{Eq. 1.3}$$
  This is to set the lower bound of degree d_i. If $v_i = 0,\ d_i = 0$. This means if a vertex is not in the path, the degree of that vertex is zero. If $v_i = 1$, it sets the lower bound of d_i

- Constraint 3 (No Going Back Constraint)
  $$x_{i,j} + x_{ji} \leq 1,\ i,j = 1,2 \dots N, i \neq j \qquad \text{Eq. 1.4}$$
  This fixes Objective (2). This constraint describes that we cannot go back to the previous vertex. Therefore, when $x_{2,3} = 1, x_{3,2} = 0$ or $x_{2,3} = 0, x_{3,2} = 1$. It also means $x_{2,3} = 0, x_{3,2} = 0$ when vertex 2 and vertex 3 are not taken.

- Constraint 4 (Degree Number Constrains the Occurrence of Specific Edges)
  $$\sum_{j=2}^{N} x_{1,j} + \sum_{j=2}^{N} x_{j,1} = d_1,\ j = 2,3 \dots \text{N} \qquad \text{Eq. 1.5}$$

This is a special case where we fix the starting point, vertex 1. d_1 = 2 is set to ensure that the path will leave from vertex 1 and come back to vertex 1

$\sum_{j=1}^{N} x_{i,j} + \sum_{j=1}^{N} x_{j,i} \leq d_j,\ j = 1, 2, 3 \ldots \text{N}\ , i = 1,2 \ldots \text{N}\ , i \neq j$ Eq. 1.6

This is to fix the occurrence of edges with the degree number. This is to make sure the vertex where the edges in coming out will become a coming in to another edge. If $d_j = 0$, it implies that there is no connected with v_j.

- Constraint 5 (Fix An Edge With One Vertex Coming In And Another Vertex Coming Out)

  $\sum_{j=2}^{N} x_{i,j} <= 1,\ i = 2,3 \ldots N, j = 1,2,3 \ldots \text{N}, i \neq j$ Eq. 1.7 a

  $\sum_{i=2}^{N} x_{i,j} <= 1,\ i = 2,3 \ldots N, j = 1,2,3 \ldots \text{N}, i \neq j$ Eq. 1.7 b

  This is to limit only edges with v_i will go only to one v_j, vice versa.

  If $\sum_{j=1}^{N} x_{4,j} = 0\ , j = 2, 3 \ldots \text{N}$, it implies it has no connection for vertex 4.

- Constraint 6 (Constrain edges with vertex)

  $\sum_{j=1}^{N} (x_{i,j} + x_{j,i}) = v_i\ \ i = 1,2 \ldots N, j = 1, 2 \ldots \text{N}, i \neq j$ Eq. 1.8

  This constrains vertex with edges. If v_3 = 1, it implies $\sum_{j=1}^{N} (x_{3,j} + x_{j,3}) = 1$. It means that there are edges connected with vertex 3. Constraint 5 handles cases where a v = 0

- Constraint 7 (Ensure connection in path)

  $x_{i,j} \leq \sum_{k=1}^{N} x_{j,k} \quad i, j, k = 1,2 \ldots N, i \neq j \neq k$ Eq. 1.9

  This is to ensure that vertex that the edges ends will start for another edge.

  If $x_{2,3} = 1,\ \sum_{k=1}^{N} x_{3,k} = 1$. If $x_{2,3} = 0,\ \sum_{k=1}^{N} x_{3,k} = 0$.

- Constraint 8 (Constraint the max degree number)

  $d_i \leq 2, i = 1, 2,3 \ldots N$ Eq. 2.0

- Constraint 9 (Degree number can only take integer)

  $d_i \geq 0\ , i = 1, 2,3 \ldots N$ Eq. 2.1

  However, d_1 = 2 because we have fixed it to be the starting and ending point.

- Constraint 10 (Edges and vertex in binary)

  $x_{i,j}, v_i \in \{0,1\}$ Eq. 2.2

  A binary edge set if the edge is a path or not. $x_{1,2} = 1$ implies that we come from vertex 1 to vertex 2. $x_{1,2} = 0$ implies we do not walk from vertex 1 to vertex 2.

  A binary vertex keeps track the existence of a vertex in a path. $v_2 = 1$ implies that we had arrived to vertex 2. $v_2 = 0$ implies we do not visit vertex 2.

IMPLEMENTATION AND SOLUTIONS (VARIATION IN TERMS OF TIME WE CAN STAY IN DISNEYLAND):

1. We coded in Java to obtain a text file format for LP Solve. All the 10 constraints stated are written with this code. [Refer to Appendix: Code in java to generate a text file for LP Solve, p21 – p 25]. We obtained the information for ride time and wait time. We sum those times and made a text file [Refer to Appendix: Text File contains Time Constraint Information, p25 - p26], so that we can obtain the time data through when coding. We coded the above constraints for making to a text file format where LP Solve can read.
2. The generated text file is in Appendix. [Refer to Appendix: Text File for LP Solve, p26 – p 28]
3. We use LP solve to solve the problem. We tried multiple variations to our solution. It works well and we settle to use N = 13. Details about more variations are discussed in the variation section.

This is the result with total time constraint of **500**. (Case I)

Value of objective function:			13		
d1	2	v1	1	x1_13	1
d2	2	v2	1	x2_4	1
d3	2	v3	1	x3_2	1
d4	2	v4	1	x4_1	1
d5	2	v5	1	x5_12	1
d6	2	v6	1	x6_11	1
d7	2	v7	1	x7_9	1
d8	2	v8	1	x8_10	1
d9	2	v9	1	x9_6	1
d10	2	v10	1	x10_5	1
d11	2	v11	1	x11_8	1
d12	2	v12	1	x12_3	1

Table 1.1: It is the result for case I

d13	2	v13	1	x13_7	1

The values of all other variables which are not listed in the table are 0.

Then we try to solve the problem with different time constraints and want to see how the results get changed.

We tried another with **180** minutes (Case II). The result is as following:

Value of objective function:			7		
d1	2	v1	1	x1_6	1
d3	2	v3	1	x3_12	1
d5	2	v5	1	x5_3	1
d6	2	v6	1	x6_5	1
d8	2	v8	1	x8_11	1
d11	2	v11	1	x11_1	1
d12	2	v12	1	x12_8	1

Table 1.2: It is the result for case II

The values of all other variables which are not listed in the table are 0.

We tried another one which is case III (with **380** minutes). The following is the result we get:

Value of objective function:			12		
d1	2	v1	1	x1_11	1
d2	2	v2	1	x2_13	1
d3	2	v3	1	x3_5	1
d5	2	v5	1	x5_2	1
d6	2	v6	1	x6_3	1
d7	2	v7	1	x7_12	1
d8	2	v8	1	x8_9	1
d9	2	v9	1	x9_7	1
d10	2	v10	1	x10_6	1
d11	2	v11	1	x11_8	1
d12	2	v12	1	x12_10	1
d13	2	v13	1	x13_1	1

Table 1.3: It is the result for case III

The values of all other variables which are not listed in the table are 0.

COMMENTARY ON SOLUTION

From the above tables, we can get the path for each case.

- Total time: 500 minutes (8 hours and 20 minutes) (Case I where time is infinity based on the 13 locations we are only considering)
  1 → 13 → 7 → 9 → 6 → 11 → 8 → 10 → 5 → 12 → 3 → 2 → 4 → 1

  In this case, we have 8 hours and 20 minutes to enjoy in the Disneyland. The path has 13 vertices, which means we can play all the 13 rides in 500 minutes. Based on this graph, the total time needed to 462.5 minutes. This implies that our constraint 1 in this case is not binding. The vertex and degree for each connection is 1 and 2 respectively. This means we have fulfilled our Objectives (1) and (3). The time taken to run this in LP Solve is 0.143s

This is the path marked on the map:

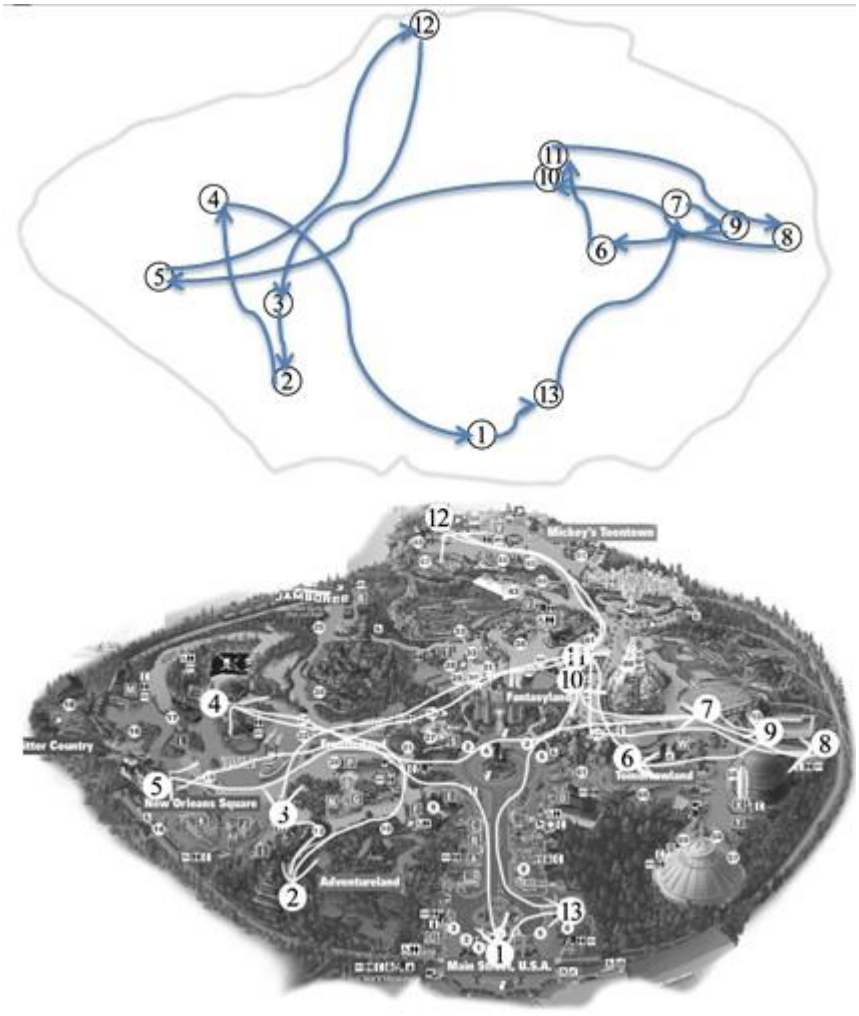

Figure 1.1: Path of 1 → 13 → 7 → 9 → 6 → 11 → 8 → 10 → 5 → 12 → 3 → 2 → 4 → 1

- Total time: 180 minutes (3 hours) (Case II)
  1 → 6 → 5 → 3 → 12 → 8 → 11 → 1

  In this case, we only have 3 hours to play in the Disneyland. The value of objective function is 7, that is, the maximum number of rides we can take in 3 hours is 7. Based on this path, the time needed for seven locations is 176.0 minutes. This implies that our constraint 1 in this case is not binding. The vertex and degree for each connection is 1 and 2 respectively. This means we have fulfilled our Objectives (1) and (3). The time taken to run this in LP Solve is 1.524s

  This is the path marked on the map:

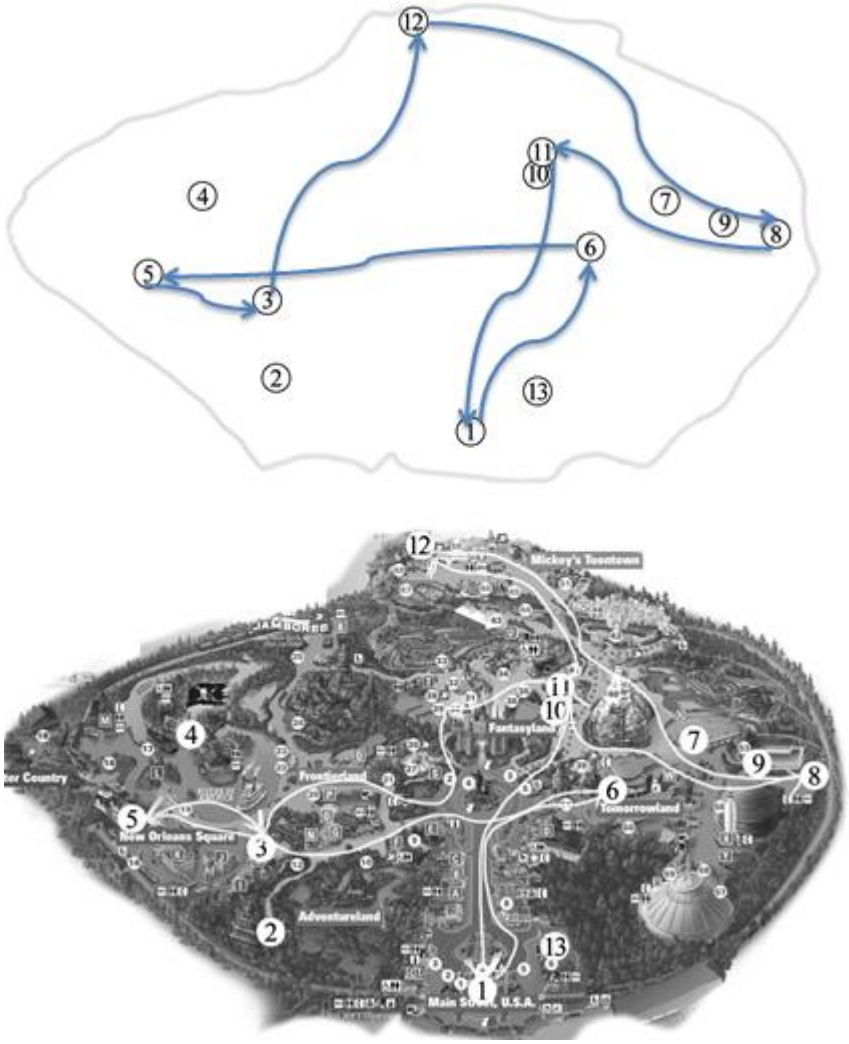

Figure 1.2: Path of 1 → 6 → 5 → 3 → 12 → 8 → 11 → 1

Total time: 380 minutes (6 hours and 20 minutes) (Case III)

1 → 11 → 8 → 9 → 7 →12 → 10 → 6 → 3 →5 → 2 → 13 →1

In this case, there is 6 hours and 20 minutes for us to stay in the Disneyland. The value of objective function tells us that we can play at most 12 rides in 380 minutes. Based on this path, the time needed for seven locations is 379.5 minutes. This implies that our constraint 1 in this case is not binding. The vertex and degree for each connection is 1 and 2 respectively. This means we have fulfilled our Objectives (1) and (3). The time taken to run this in LP Solve is 1.876s

This is the path marked on the map:

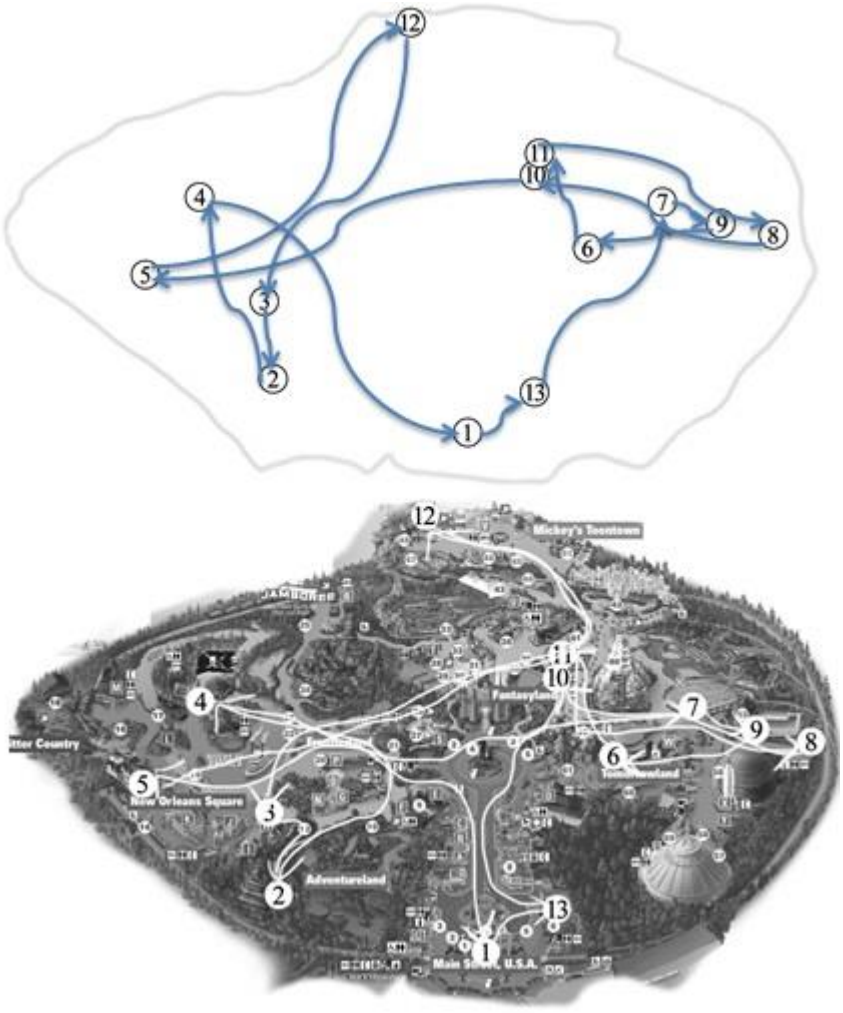

Figure 1.3: Path of 1 → 11 → 8 → 9 → 7 →12 → 10 → 6 → 3 →5 → 2 → 13 →1

The time constraint for each cases are not binding because of the constraint of fixing the edges (xi_j) to be binary and the ti_j contains decimal places. The choice of time constraint is arbitrary, where we choose an 'infinity' time for our Case I, a constraining time for Case II and somehow reasonable time constraint in Case III. The model is sensible because the longer the time we have in Disneyland, we can go more location. In addition, each connected vertex has degree of 2 which fulfills are Objectives (1) and (3).

VARIATIONS BY CONSIDERING THE QUALITY OF THE RIDES/ DESTINATION AND TIME CONSTRAINT:

Variation I (Or the first trial with different N < 13)
Part I
This is the variation that we first for our model. This part is crucial as guidance for the validity of the model

We tried where N = 5 with 180 minutes

The result is as following:

Value of objective function:			5		
d1	2	v1	1	x1_5	1
d2	2	v2	1	x5_4	1
d3	2	v3	1	x4_3	1
d4	2	v4	1	x3_2	1
d5	2	v5	1	x2_1	1

Table 1.4: It is the result for Variation 1, where N = 5 with 180 minutes.

The values of all other variables which are not listed in the table are 0. In 180 minutes, we managed to visit all 5 vertices. Based on this path, the time needed for 5 locations is 134.5 minutes. This implies that our constraint 1 in this case is not binding. The vertex and degree for each connection is 1 and 2 respectively. This means we have fulfilled our Objectives (1) and (3). The time taken to run this in LP Solve is 0.030s

The path is 1→5→4→3→2→1

We tried where N = 8 with 180 minutes

The result is as following:

Value of objective function:			8		
d1	2	v1	1	x1_3	1
d2	2	v2	1	x3_2	1
d3	2	v3	1	x2_6	1
d4	2	v4	1	x6_4	1
d5	2	v5	1	x4_7	1
d6	2	v6	1	x7_5	1
d7	2	v7	1	x5_8	1
d8	2	v8	1	x8_1	1

Table 1.5: It is the result for Variation 1, where N = 8 with 180 minutes.

The values of all other variables which are not listed in the table are 0. In 180 minutes, we managed to visit all 8 vertices. Based on this path, the time needed for 8 locations is 157.0 minutes. This implies that our constraint 1 in this case is not binding. The vertex and degree for each connection is 1 and 2 respectively. This means we have fulfilled our Objectives (1) and (3). The time taken to run this in LP Solve is 0.023s

The path is 1→3→2→6→4→7→5→8→1

We tried where N = 8 with 100 minutes

The result is as following:

Value of objective function:			5		
d1	2	v1	1	x1_6	1
d5	2	v5	1	x6_7	1
d6	2	v6	1	x7_5	1
d7	2	v7	1	x5_8	1
d8	2	v8	1	x8_1	1

Table 1.6: It is the result for Variation 1, where N = 8 with 100 minutes.

The values of all other variables which are not listed in the table are 0. In 180 minutes, we managed to visit all 8 vertices. Based on this path, the time needed for 5 locations is 94.5 minutes. This implies that our constraint 1 in this case is not binding. The vertex and degree for each connection is 1 and 2 respectively. This means we have fulfilled our Objectives (1) and (3). The time taken to run this in LP Solve is 0.048s

The path is 1→6→7→5→8→1

Variation II

Part I

In this variation, we will introduce the ratings of each ride to represent the level of enjoyment. The data[1] are as follow:

	Location	Ratings(stars)
1	Main Street	2
2	Indiana Jones	5
3	Pirates of the Caribbean	5
4	Pirates Lair	3
5	Haunted Mansion	4
6	Buzz Lightyear	4

Table 1.7: Rating of respective vertices

7	Finding Nemo Submarine	4
8	DisneyLand Monorail (near Nemo)	3
9	Autopia	2.5
10	Alice in Wonderland	3.5
11	Mad Tea Party	2
12	Mickey's House	3
13	The DisneyLand Story	2

In order to maximize the level enjoyment, which is represented by the sum of ratings from each ride we take, we will change the objective function to:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} rating_j x_{i,j}\,, where\ i,j = 1,2 \ldots N \qquad \text{Eq. 2.3}$$

Therefore, code in java to generate a text file for LP Solve [Refer to Appendix: code in java to generate a text file for LP Solve, p21 – p 25] is added to line 17

```
double[] ratings = {0, 2, 5, 5, 3, 4, 4, 4, 3, 2.5, 3.5, 2, 3, 2};
// Maximize the total ratings
    output.print("max:");
     for (int i = 1; i <= N; i++) {
      double value = ratings[i];
          for (int j = 1; j <= N; j++) {
                if (i != j) {
                   String variable = "x" + j + "_" + i;
                     output.print(" + " + value + variable);
                }
             }
          }
```

The objective function that we are maximizing is
max: + 2.0 x2_1 + 2.0 x3_1 + 2.0 x4_1 +….+ 2.0 x12_13.
When N = 13 with total time 100 minutes, the path we are going to take is:

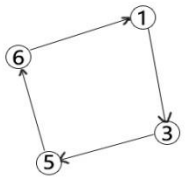

Figure 1.4 This is the connection that gives us a rating of 15 stars

Part II

Changing total time constraints and get the sequence of rides that we can take.

#	time limit(minute)	ride sequence	Rating
1	30	N/A	0
2	60	1→6→8→1	11
3	90	1→6→8→9→1	13.5
4	120	1→5→6→8→9→1	17.5
5	150	1→11→3→5→6→8→1	22
6	180	1→11→8→6→5→9→12→1	22.5
7	210	1→6→5→3→11→8→9→12→1	27.5

Table 1.8: It is the result for variation case

This model is sensible as the rating increases with the number vertices are going to visit.

We have done two major variations in this model

1. We changed the time we can stay in DisneyLand with different N and (p 13-p15)
    a. This may seem a not a big variation, but we varied at least 10 values of time constraints. This took most of our time because we checked and modified the code
    b. We also tried different N with different time constraint for checking the sensibility of our model because we wished for this model to work for other problems that is similar to this.

2. We included the ratings for each vertex (p15- p16)

Beyond current variation:

1. Since we have fixed vertex 1 as starting point, we can fix other locations as starting point as well. This means this model can be applicable for other purposes than this
2. We have ideas to fix the rides that we must go. Theoretically, we would fix the degree number of a vertex. However, this model can fail if the time is too small, like 30 minutes, because based on out model, we need at least 3 places to go with 4 different edges and there are no combinations that gives 30 minutes path

CHALLENGES AND COMMENTS:

- At first, we wanted to do a directed path where we are not going back to the starting point. However, we had a hard to constrain where the degree for vertex starting point and vertex ending point to have degree one each. We can fix the starting point degree but the ending point is random where we could find a way to fix the degree of ending point.
- The path is disconnected. For example, $1 \rightarrow 3 \rightarrow 4$ then $5 \rightarrow 6$. However, constraint 7 has ensured the path connection.
- The decision to fix a starting point is uncertain. At first, we coded using random starting points, but later we found that the path will never connect to vertex 1. This does not make sense because vertex 1 is the main street (starting point). Therefore, we fix the starting by fixing the degree number of 2 for vertex 1.
- We encountered a lot of parse error in LP solve because we were not familiar to use it and it is easy to miss something in the text file. It can be as easy as a semi-colon. This took some of our time to figure out.
- Our first decent algorithm needs 36 minutes to run when N = 13, N is the number of vertices. However, we changed the code to be confining constraints
- Our model is not perfect as we spent like least than a month to develop it. No model is perfect in reality. However, it has potential to go beyond. Another fact that we are not comfortable is we took a leap of faith that LP Solve will give the best solution. We believe that this model can be improved other than using linear programming. In the LP Solve, it uses simplex method to optimize the variables that we desire.

CONCLUSION:

Finding the best route to visit Disneyland is by no ways easy. But, thanks to the power of integer programming and graphs, as well as some snippets of code, we can indeed find that there are paths we can take that can maximize the rides we can visit without repeat. Similarly, we can also find a path that can maximize the cumulative fun rating in a certain time limit. If a family chooses to stay for around an eight hour trip, they can in fact visit all the rides we have included

in our investigation and finish at the same ride they started. In essence, we have actually solved an example of the Traveling Salesperson problem. This model can also be extended to problems similar to this. Throughout the process of this investigation, we have discovered when visiting Disneyland, it is not necessary to be burdened by long wait lines and traveling times. So the next time you, your family, or your friends drop by Disneyland, remember there is in fact a way to visit all your favourite rides without having to spend an eternity waiting around.

REFERENCES/BIBLIOGRAPHY:

1.Touringplans.com, Disneyland Attractions - Our Ratings. <https://touringplans.com/disneyland/attractions/our-ratings>

2. Klarreich, Erica. "Computer Scientists Find New Shortcuts for Infamous Traveling Salesman Problem." Wired.com. Conde Nast Digital, 30 Jan. 2013. Web. 02 Feb. 2016. <http://www.wired.com/2013/01/traveling-salesman-problem/>.

3. "TSP History Home." TSP History Home. UWaterloo, Jan. 2007. Web. 02 Feb. 2016. <http://www.math.uwaterloo.ca/tsp/history/>.

4. Johnson, David S., and Lyle A. McGeoch. "Traveling Salesman Problem: A Case Study in Local Optimization."SpringerReference (November 20, 1995). University of British Columbia. Web. 2 Feb. 2016. <https://www.cs.ubc.ca/~hutter/previous-earg/EmpAlgReadingGroup/TSP-JohMcg97.pdf>.

5. "Disneyland Wait Times." Disneyland Wait Times. Disney. Web. 22 Jan. 2016. <https://touringplans.com/disneyland/wait-times>.

6. "Duration of Disneyland Attractions." Duration of Disneyland Attractions. Disney. Web. 27 Jan. 2016. <https://touringplans.com/disneyland/attractions/duration>.

APPENDIXES:

Walking Time Table: This is the table which the time taken to move from vertex i to vertex j. This is taken from Google Map

	Time (Min)		Time (Min)		Time (Min)		Time (Min)		Time (Min)		Time (Min)		Time (Min)
x1,1	0	x2,1	5	x3,1	5	x4,1	6	x5,1	7	x5,1	7	x6,1	4
x1,2	5	x2,2	0	x3,2	1	x4,2	1	x5,2	2	x5,2	2	x6,2	4
x1,3	5	x2,3	1	x3,3	0	x4,3	1	x5,3	1	x5,3	1	x6,3	4
x1,4	6	x2,4	1	x3,4	1	x4,4	0	x5,4	1	x5,4	1	x6,4	4
x1,5	7	x2,5	2	x3,5	2	x4,5	1	x5,5	0	x5,5	0	x6,5	5
x1,6	4	x2,6	4	x3,6	4	x4,6	4	x5,6	5	x5,6	5	x6,6	0
x1,7	6	x2,7	5	x3,7	5	x4,7	6	x5,7	7	x5,7	7	x6,7	3
x1,8	6	x2,8	5	x3,8	5	x4,8	6	x5,8	7	x5,8	7	x6,8	2
x1,9	6	x2,9	6	x3,9	5	x4,9	6	x5,9	8	x5,9	8	x6,9	3
x1,10	5	x2,10	4	x3,10	4	x4,10	6	x5,10	6	x5,10	6	x6,10	3
x1,11	6	x2,11	10	x3,11	10	x4,11	11	x5,11	12	x5,11	12	x6,11	9
x1,12	9	x2,12	8	x3,12	8	x4,12	9	x5,12	10	x5,12	10	x6,12	6
x1,13	1	x2,13	4	x3,13	5	x4,13	6	x5,13	6	x5,13	6	x6,13	3

	Time (Min)		Time (Min)		Time (Min)		Time (Min)		Time (Min)		Time (Min)
x7,1	6	x8,1	6	x9,1	6	x10,1	5	x9,1	6	x11,1	6
x7,2	5	x8,2	5	x9,2	6	x10,2	4	x9,2	6	x11,2	10
x7,3	5	x8,3	5	x9,3	5	x10,3	4	x9,3	5	x11,3	10
x7,4	6	x8,4	6	x9,4	6	x10,4	6	x9,4	6	x11,4	11
x7,5	5	x8,5	7	x9,5	7	x10,5	5	x9,5	7	x11,5	12
x7,6	3	x8,6	2	x9,6	3	x10,6	3	x9,6	3	x11,6	9
x7,7	0	x8,7	1	x9,7	1	x10,7	3	x9,7	1	x11,7	11
x7,8	1	x8,8	0	x9,8	1	x10,8	2	x9,8	1	x11,8	2
x7,9	1	x8,9	1	x9,9	0	x10,9	4	x9,9	0	x11,9	11
x7,10	3	x8,10	2	x9,10	3	x10,10	0	x9,10	3	x11,10	10
x7,11	11	x8,11	2	x9,11	11	x10,11	10	x9,11	11	x11,11	0
x7,12	6	x8,12	6	x9,12	6	x10,12	4	x9,12	6	x11,12	14
x7,13	4	x8,13	5	x9,13	4	x10,13	4	x9,13	4	x11,13	6

	Time (Min)		Time (Min)								
x12,1	9	x13,1	1								
x12,2	8	x13,2	4								
x12,3	8	x13,3	4								
x12,4	9	x13,4	5								
x12,5	10	x13,5	6								
x12,6	7	x13,6	3								
x12,7	8	x13,7	5								
x12,8	6	x13,8	5								
x12,9	7	x13,9	5								
x12,10	4	x13,10	5								
x12,11	14	x13,11	6								
x12,12	0	x13,12	8								
x12,13	8	x13,13	0								

Code in java to generate a text file for LP Solve

```
// Group 4
// DisneyLand Traveling Sales Person Problem
// The path way is a circle
// The main entrance (vertex 1) is the starting and ending point
// This algorithm is flexible to solve fixed starting point situation

import java.io.*;     // for File, FileNotFoundException
import java.util.*;   // for Scanner, List, Set, Collections

public class TravelingSalesPersonDisneyLand {
   public final static int N = 13;
   public final static int TotalTime = 180;

   public static void main(String[] args) throws FileNotFoundException {
      PrintStream output = new PrintStream(new
File("DisneyLandMaxEdgefs.txt"));
      output.println("/* Objective function */");

      //Obtain the ride time + wait time info from a text file
      Scanner input = new Scanner(new File("IPdata_13_total.txt"));
      double [][] time = new double [N + 1][N + 1];
      for (int i = 1; i <= N; i++) {
         for (int j = 1; j <= N; j++) {
            time[i][j] = input.nextDouble();
         }
      }

```

```
27        output.println("/* Maximize the number of edges */");
28        // Maximize the number of edges
29        output.print("max:");
30        for (int i = 1; i <= N; i++) {
31           for (int j = 1; j <= N; j++) {
32              if (i != j) {
33                 String variable = "x" + i + "_" + j;
34                 output.print(" + " + variable);
35              }
36           }
37        }
38        output.println(";");
39        output.println();
40
41
42        //  Constraint 1 Wait Time + Ride Time
43        output.println("/* Constraint 1 Wait Time + Ride Time into
consideration */");
44        output.println();
45        output.print("    ");
46        for (int i = 1; i <= N; i++) {
47           for (int j = 1; j <= N; j++) {
48              if (i != j) {
49                 double getTime = time[i][j];
50                 String variable = "x" + i + "_" + j;
51                 output.print(" + " + getTime + " " + variable);
52              }
53           }
54        }
55        output.println(" <= " + TotalTime + ";");
56        output.println();
57
58        // Constraint 2 (Constrain Degree Number and Vertex with Degree
Number)
59        // Limit vertex i cannot be larger than degree i
60        // Special constraint on vertex 1 (entrance) d1 = 2;
61        // Other degrees can be equal or less than 2 but cannot be <0
62        // Ensure circular path
63        output.println("/* Constrain Degree Number and Vertex with Degree
Number */");
64        output.println();
65        output.println("     d1 = 2;");
66        for (int i = 2; i <= N; i++) {
67           output.print("     ");
68           output.println("+ d" + i + ">= " + "v" + i + ";");
69        }
70        output.println();
71
72        // Constraints  3 No Going back
73        // + x1_2 + x2_1 <= 1;
74        output.println("/* Constraints  3 No Going back  */");
75        output.println();
76        for (int i = 1; i <= N; i++) {
77           for (int j = 1; j <= N; j++) {
78              String variable1 = "x" + i + "_" + j;
79              String variable2 = "x" + j + "_" + i;
80
```

```
 81              if (i != j) {
 82                 output.print("     ");
 83                 output.print("+ " + variable1 + " + " + variable2);
 84                 output.println(" <= 1;");
 85              }
 86           }
 87        }
 88        output.println();
 89
 90        // Constraint 4 (Degree Number Constrains the Occurrence of
Specific Edges)
 91        // If j Sum of edge ij + j Sum edge ji = 0 it means degree i = 0
 92        // If i Sum of edge ij + i Sum edge ji = 0 it means degree j = 0
 93        // + x1_2+ x1_3+ x1_4+ x1_5+ x1_6+ x1_7+ x2_1+ x3_1+ x4_1+ x5_1+
x6_1+ x7_1 = d1;
 94        // + x2_1+ x2_3+ x2_4+ x2_5+ x2_6+ x2_7+ x1_2+ x3_2+ x4_2+ x5_2+
x6_2+ x7_2 <= d2;
 95        output.println("/* Constraint 4 (Degree Number Constrains the
Occurrence of Specific Edges)  */");
 96        output.println();
 97         for (int i = 1; i <= N; i++) {
 98           String variable1 = "";
 99           String variable2 = "";
100           for (int j = 1; j <= N; j++) {
101              if (i != j) {
102                 variable1 = variable1 + "+ x" + i + "_" + j;
103                 variable2 = variable2 + "+ x" + j + "_" + i;
104
105              }
106           }
107           output.print("     ");
108           output.print(variable1  + variable2);
109           if (i == 1) {
110                    output.println(" = d" + i+ ";");
111           } else {
112              output.println(" <= d" + i+ ";");
113           }
114        }
115        output.println();
116
117         //Constraint 5 (Fix An Edge With One Vertex Coming In And Another
Vertex Coming Out)
118        output.println("/* Constraint 5 (Fix An Edge With One Vertex Coming
In And Another Vertex Coming Out) */");
119        output.println();
120        for (int i = 1; i <= N; i++ ) {
121           String variable1 = "";
122           String variable2 = "";
123           for (int j = 1; j <= N; j++) {
124              if( i != j) {
125                 variable1 = variable1 + "+ x" + i + "_" + j;
126                 variable2 = variable2 + "+ x" + j + "_" + i;
127              }
128           }
129           output.print("     ");
130           output.println(variable1 + " <= " + 1 + ";");
131           output.print("     ");
```

```
132          output.println(variable2 + " <= " + 1 + ";");
133       }
134       output.println();
135
136       // Constraint 6 (Constrain edges with vertex)
137       output.println("/* Constraint 6 (Constrain edges with vertex) */");
138       output.println();
139       output.println("     ");
140       for (int j = 1; j <= N; j++) {
141          String variable = "+ v" + j + "=" ;
142
143          for (int i = 1; i <= N; i++) {
144             if( i != j) {
145                variable = variable +" + x" + j + "_" + i;
146             }
147          }
148          output.print("     ");
149          output.println(variable + " ;");
150       }
151       output.println();
152
153       // Constraint 7 (Ensure connection in path)
154       // To ensure all the edges are connected
155       // Example output
156       //+ x13_12 <= + x12_2+ x12_3+ x12_4+ x12_5+ x12_6+ x12_7+ x12_8+
x12_9+ x12_10+ x12_11;
157       output.println("/* Constraint 7 (Ensure connection in path) */");
158       output.println();
159       for (int i = 1; i <= N; i++) {
160
161          for (int j = 1; j <= N; j++ ){
162             if (i != j) {
163                String variable = "";
164                String variable1 = "";
165                variable = "+ x" + i + "_" + j;
166                for (int k = 1; k <= N; k++) {
167                   if (i != j && k != j && i != k) {
168                      variable1 = variable1 + "+ x" + j+ "_" + k;
169                   }
170                }
171                output.println ("     "+ variable + " <= " + variable1 +
";");
172             }
173          }
174       }
175       output.println();
176
177       // Constraint  8 (Constraint the max degree number)
178       // Special case is on vertex 1 when we fix the starting point.
179       // d1 = 2, di <= 2
180       //  cannot be larger than 2
181       output.println("/* Constraint  8 (Constraint the max degree number)
*/");
182       output.println();
183       for (int i = 2; i<= N; i++) {
184          output.println("     d"+ i+" <= 2;");
185       }
```

```
186         output.println();
187
188         // Constraint 9 (Degree number can only take integer)
189         output.println("/* Constraint 9 (Degree number can only take
integer)  */");
190         output.println();
191         output.print("int ");
192         for (int i = 2; i <= N; i++) {
193            output.print("d" + i);
194            if (i == N) {
195               output.print(";");
196            } else {
197               output.print(",");
198            }
199         }
200
201         // Constraint 10 (Edges and vertex in binary)
202         output.println("/* Constraint 10 (Edges and vertex in binary) */");
203         output.println();
204         output.print("bin ");
205         for (int i = 1; i <= N; i++) {
206            for (int j = 1; j <= N; j++) {
207               String variable = "x" + i + "_" + j;
208               if (i == N && j == N){
209                  output.print(variable + ",");
210               } else {
211                  output.print(variable + ",");
212               }
213            }
214         }
215         for (int i = 1; i <= N; i++) {
216            if ( i != N) {
217               output.print("v" + i + ", ");
218            } else {
219               output.print("v" + i  + "; ");
220            }
221         }
222      }
223   }
```

Text File contains Time Constraint Information

0.0	59.0	32.0	26.5	36.0	26.0
58.0	31.0	5.0	58.0	35.0	26.5
38.0	30.5	53.0	30.0	5.0	58.0
76.0	43.0	34.0	30.5	54.0	30.0
31.0	17.5	71.0	42.0	33.0	29.5
26.5	37.0	26.0	21.5	71.0	42.0

21.5	32.0	29.0	59.0	32.0	31.5
36.0	32.5	25.5	38.0	35.0	42.0
36.0	44.0	53.0	76.0	6.0	25.5
6.0	23.5	26.0	31.0	63.0	28.0
54.0	38.0	25.5	25.5	43.0	39.0
34.0	37.0	41.0	54.0	81.0	1.0
70.0	4.0	22.5	26.0	36.0	57.0
25.0	57.0	34.0	24.5	31.5	37.0
26.5	37.0	35.0	41.0	64.0	75.0
59.0	74.0	6.0	22.5	27.0	30.0
31.0	29.0	58.0	34.0	35.5	25.5
30.5	22.5	38.0	35.0	48.0	58.0
44.0	56.0	76.0	5.0	11.5	30.0
22.5	27.0	31.0	57.0	42.0	29.5
37.0	27.5	24.5	37.0	37.0	43.0
37.0	41.0	54.0	76.0	9.0	17.5
7.0	20.5	25.0	29.0	61.0	36.0
55.0	34.0	25.5	25.5	41.0	31.0
34.0	34.0	40.0	56.0	79.0	
71.0	6.0	13.5	27.0	34.0	
24.0	58.0	34.0	28.5	29.5	
27.5	38.0	36.0	38.0	61.0	
60.0	76.0	6.0	21.5	31.0	

Text File for LP Solve

/* Objective function */
/* Maximize the number of edges */
max: + x1_2 + … + x13_12;

/* Constraint 1 Wait Time + Ride Time into consideration */

+ 58.0 x1_2 + 38.0 x1_3 + 76.0 x1_4 + 31.0 x1_5 + 26.5 x1_6 + 59.0 x1_7 + 31.0 x1_8 + 30.5 x1_9 + 43.0 x1_10 + 17.5 x1_11 + 37.0 x1_12 + 32.0 x1_13 + 5.0 x2_1 + 34.0 x2_3 + 71.0 x2_4 + 26.0 x2_5 + 26.5 x2_6 + 58.0 x2_7 + 30.0 x2_8 + 30.5 x2_9 + 42.0 x2_10 + 21.5 x2_11 + 36.0 x2_12 + 35.0 x2_13 + 5.0 x3_1 + 54.0 x3_2 + 71.0 x3_4 + 26.0 x3_5 + 26.5 x3_6 + 58.0 x3_7 + 30.0 x3_8 + 29.5 x3_9 + 42.0 x3_10 + 21.5 x3_11 + 36.0 x3_12 + 36.0 x3_13 + 6.0 x4_1 + 54.0 x4_2 + 34.0 x4_3 + 25.0 x4_5 + 26.5 x4_6 + 59.0 x4_7 + 31.0 x4_8 + 30.5 x4_9 + 44.0 x4_10 + 22.5 x4_11 + 37.0 x4_12 + 37.0 x4_13 + 7.0 x5_1 + 55.0 x5_2 + 34.0 x5_3 + 71.0 x5_4 + 27.5 x5_6 + 60.0 x5_7 + 32.0 x5_8 + 32.5 x5_9 + 44.0 x5_10 + 23.5 x5_11 + 38.0 x5_12 + 37.0 x5_13 + 4.0 x6_1 + 57.0 x6_2 + 37.0 x6_3 + 74.0 x6_4 + 29.0 x6_5 + 56.0 x6_7 + 27.0 x6_8 + 27.5 x6_9 + 41.0 x6_10 + 20.5 x6_11 + 34.0 x6_12 + 34.0 x6_13 + 6.0 x7_1 + 58.0 x7_2 + 38.0 x7_3 + 76.0 x7_4 + 29.0 x7_5 + 25.5 x7_6 + 26.0 x7_8 + 25.5 x7_9 + 41.0 x7_10 + 22.5 x7_11 + 34.0 x7_12 + 35.0 x7_13 + 6.0 x8_1 + 58.0 x8_2 + 38.0 x8_3 + 76.0 x8_4 + 31.0 x8_5 + 24.5 x8_6 + 54.0 x8_7 + 25.5 x8_9 + 40.0 x8_10 + 13.5 x8_11 + 34.0 x8_12 + 36.0 x8_13 + 6.0 x9_1 + 59.0 x9_2 + 38.0 x9_3 + 76.0 x9_4 + 31.0 x9_5 + 25.5 x9_6 + 54.0 x9_7 + 26.0 x9_8 + 41.0 x9_10 + 22.5 x9_11 + 34.0 x9_12 + 35.0 x9_13 + 5.0 x10_1 + 57.0 x10_2 + 37.0 x10_3 + 76.0 x10_4 + 29.0 x10_5 + 25.5 x10_6 + 56.0 x10_7 + 27.0 x10_8 + 28.5 x10_9 + 21.5 x10_11 + 32.0 x10_12 + 35.0 x10_13 + 6.0 x11_1 + 63.0 x11_2 + 43.0 x11_3 + 81.0 x11_4 + 36.0 x11_5 + 31.5 x11_6 + 64.0 x11_7 + 27.0 x11_8 + 35.5 x11_9 + 48.0 x11_10 + 42.0 x11_12 + 37.0 x11_13 + 9.0 x12_1 + 61.0 x12_2 + 41.0 x12_3 + 79.0 x12_4 + 34.0 x12_5 + 29.5 x12_6 + 61.0 x12_7 + 31.0 x12_8 + 31.5 x12_9 + 42.0 x12_10 + 25.5 x12_11 + 39.0 x12_13 + 1.0 x13_1 + 57.0 x13_2 + 37.0 x13_3 + 75.0 x13_4 + 30.0 x13_5 + 25.5 x13_6 + 58.0 x13_7 + 30.0 x13_8 + 29.5 x13_9 + 43.0 x13_10 + 17.5 x13_11 + 36.0 x13_12 <= 180;

/* Constrain Degree Number and Vertex with Degree Number */

d1 = 2;
+ d2>= v2;

.

.

.

+ d13>= v13;

/* Constraints 3 No Going back */

+ x1_2 + x2_1 <= 1;

.

.

.

+ x13_12 + x12_13 <= 1;

/* Constraint 4 (Degree Number Constrains the Occurrence of Specific Edges) */

+ x1_2+ x1_3+ x1_4+ x1_5+ x1_6+ x1_7+ x1_8+ x1_9+ x1_10+ x1_11+ x1_12+ x1_13+ x2_1+ x3_1+ x4_1+ x5_1+ x6_1+ x7_1+ x8_1+ x9_1+ x10_1+ x11_1+ x12_1+ x13_1 = d1;

.

.

.

+ x13_1+ x13_2+ x13_3+ x13_4+ x13_5+ x13_6+ x13_7+ x13_8+ x13_9+ x13_10+ x13_11+ x13_12+ x1_13+ x2_13+ x3_13+ x4_13+ x5_13+ x6_13+ x7_13+ x8_13+ x9_13+ x10_13+ x11_13+ x12_13 <= d13;

/* Constraint 5 (Fix An Edge With One Vertex Coming In And Another Vertex Coming Out) */

```
+ x1_2+ x1_3+ x1_4+ x1_5+ x1_6+ x1_7+ x1_8+ x1_9+ x1_10+ x1_11+ x1_12+ x1_13 <= 1;
+ x2_1+ x3_1+ x4_1+ x5_1+ x6_1+ x7_1+ x8_1+ x9_1+ x10_1+ x11_1+ x12_1+ x13_1 <= 1;
            .

            .

            .

+ x1_13+ x2_13+ x3_13+ x4_13+ x5_13+ x6_13+ x7_13+ x8_13+ x9_13+ x10_13+ x11_13+ x12_13 <= 1;

/* Constraint 6 (Constrain edges with vertex) */


+ v1= + x1_2 + x1_3 + x1_4 + x1_5 + x1_6 + x1_7 + x1_8 + x1_9 + x1_10 + x1_11 + x1_12 + x1_13 ;
            .

            .

            .

+ v13= + x13_1 + x13_2 + x13_3 + x13_4 + x13_5 + x13_6 + x13_7 + x13_8 + x13_9 + x13_10 + x13_11 +
x13_12 ;

/* Constraint 7 (Ensure connection in path) */

+ x1_2 <= + x2_3+ x2_4+ x2_5+ x2_6+ x2_7+ x2_8+ x2_9+ x2_10+ x2_11+ x2_12+ x2_13;
            .

            .

            .

+ x13_12 <= + x12_1+ x12_2+ x12_3+ x12_4+ x12_5+ x12_6+ x12_7+ x12_8+ x12_9+ x12_10+ x12_11;

/* Constraint  8 (Constraint the max degree number) */

    d2 <= 2;
            .

            .

            .

    d13 <= 2;

/* Constraint 9 (Degree number can only take integer)  */

int d2,d3,d4,d5,d6,d7,d8,d9,d10,d11,d12,d13;

/* Constraint 10 (Edges and vertex in binary) */
bin x1_2,x1_3,..., x13_12 ,v1 ... v13;
```