SESUG Paper 78-2025

Proc Optgraph, the Traveling Salesman Problem and Map Drawing

Caleb Greski and Joe DeMaio, Kennesaw State University.

ABSTRACT

Many people, such as Superbowl winning quarterbacks, celebrate success in life by saying “I’m going to Disneyland!” What do you do once you get there? I doubt you want to spend lots of time waiting in line. Perhaps there is an optimized order in which to visit rides and attractions to minimize line wait time. This is known as the Traveling Salesman Problem (TSP). In this paper we demonstrate how to use the Optgraph Procedure in SAS to compute a route for our travelling salesman and construct a visualization of said tour on a map. Using the annotation data set feature allows us to design graphical displays of TSPs which overlay the optimized path on top of any relevant images, such as a map designed by a graphic designer, or a chess board.

INTRODUCTION

Many people celebrate success in life by saying “I’m going to Disneyland!”

Phil Simms first uttered this magic phrase in 1987 after the N.Y. Giants won the Superbowl. Most recently in 2025, Philadelphia Eagles quarterback Jalen “Hurts uttered the magic words — ‘I’m going to Disney World’ — to the camera after his team defeated two-time defending champion Kansas City Chiefs 40-22 in New Orleans. Hurts completed 17 of 22 passes for 221 yards and two touchdowns in the win.” [1]

This platitude is so entrenched into American culture that it was parodied in the Dada song *I’m going to Dizz Knee Land*. Going to Disneyland (or some other theme park) is great! But there are usually a lot of other people there who also want to have fun. We’re not looking forward to waiting in line with an average Disneyland daily attendance of 27,397 of our newest and closest friends. Or perhaps you don’t mind waiting in line as it gives you an opportunity to talk to your friends, but you are opposed to the total walking distance around the park. Depending upon specific but subjective preferences there can be many ways to optimize your day of fun.

Typically, graphs or networks model the Traveling Salesman Problem (TSP). This comes as no surprise as Euler is considered the father of graph theory [2] whose 1759 paper [3] applies graphs as a model to tackle the problem of the closed knight’s tour. In general, a graph $G = (V, E)$ is an ordered pair of sets where the vertex set V represents a set of vertices or nodes for the model and the edge set E represents a relationship between a pair of vertices. Typically for the Traveling Salesman Problem (TSP), vertices represent geographical locations and edges represent the cost of traveling between those geographic locations. The TSP is an applied version of the search for a cycle in a graph that visits every vertex exactly once. Such a cycle is called a Hamiltonian cycle named for Sir William Rowan Hamilton’s Icosian game invented in 1857. This came much to the chagrin of Thomas Kirkman who had previously studied the problem. This general problem is NP-complete and is the first one in existence of such a cycle in graphs that are not complete. See [4] and [5] for surveys on the general problem. Second comes the applied TSP of minimum weight. This paper focuses on the generation and visualization of these optimized tours using SAS.

CREATING A GRAPH IN SAS

One can create a grade with coding commands from inside of a SAS worksheet or one can upload a file that represents a graph. In general, one defines a graph by indicating the vertex set and edge set. Since we are specifically looking at the TSP, we will also need to include edge weights. Our first example shall be the creation of a closed knight’s tour on a rectangular chessboard. The 1-2 moves of the knight are easily defined without the need for an external file of geographical data. To demonstrate the use of edge weights, we will code a preference for a horizontal 2 move rather than vertical. Complete code is found in Appendix A.

First, we define a board size and construct the legal 1-2 moves of a knight for said board. When defining a knight's move, we add in a weight for each move. For ease of coding, we've hard coded a specific $5x8$ board which can be easily changed.

```
data board;
rows = 5;
columns = 8;
do i = 1 to rows;
do j = 1 to columns;
origin =  (i || j);
/* input legal moves of the knight */
/* up 2 right 1 */
if i>=3 and j<= columns-1 then do
      destination = (i-2 || j+1);
      cost=5;
      output;
      end;
```

USING OPTGRAPH TO FIND A TSP ROUTE

Second, we call Optgraph, create the graph and run the TSP command.

```
proc optgraph
data_links = board;
*create graph from variables;
data_links_var
from = origin
to = destination
weight=cost;
* write cycle to file closed_knights_tour;
tsp out = closed_knights_tour;
run;

proc print data=work.closed_knights_tour;
* print to results file;
run;
```

OPTGRAPH OUTPUT

By default, SAS outputs the optimized tour as a table. For the $5x8$ board, an optimized tour has $5x8 = 40$ squares to visit and 40 moves to make. For brevity's sake here are the first five rows of output. The full output can be found in Appendix B.

Obs	origin		destination		cost
1	1	1	2	3	3
2	2	3	1	5	3
3	1	5	2	7	3
4	3	5	2	7	3
5	3	5	4	7	3

Table 1. The first five moves

Knowing we have an optimized tour is great. But reading the tables does not immediately manifest an understanding of what the tour will be.

PROCESSING THE OUTPUT FOR PLOTTING

In order to take the (x,y) string output of `OPTGRAPH` and use it for plotting, we need to process the origin and destination pairs such that they are numeric values that SAS can interpret as a series in `SGPLOT`. The resulting dataset converts each (origin, destination) edge pair into a single column of (x,y) destination coordinates, and so the first and last observations are the same (x,y) pair.

```
data knights_tour1;
    drop destination;
    set knights_tour (obs=1) knights_tour;
    rownum=_n_;
    if origin = lag(origin) then origin = destination;
    if origin = lag(destination) then origin = destination;
    if destination = lag(origin) then origin = origin;
    if destination = lag(destination) then origin = origin;
    x = substr(origin, notspace(origin,1), 1);
    y = substr(origin, notspace(origin,-99), 1);
run;
```

VISUALIZING THE KNIGHT'S TOUR ON A BOARD

Typically graphs have no orientation in the plane; they represent relationships between pairs of vertices. That obviously changes when considering the TSP where orientation is important, as vertices represent locations like on a map or, in our case, on a chessboard. So, constructing a graph with a visualization for orientation is highly valuable technique. Of course, this is easier said than done. Fortunately, SAS provides the tools to do so.

Overlaying a plot on a map is not trivial as different graphical display methods handle this process in different ways. There are ways within ODS to insert images into graphics generation, such as in `GPLOT`, but these methods require lots of adjustment, and a consistent way of both aligning the back-image while also effectively cropping that image to the space of a plot's borders could not be determined. Instead, an effective solution was developed with annotation data sets. Annotations are a way of overlaying graphical items like text and shapes on top of plots, and they can also be used to place these same things on the back layer of a plot. Annotation data sets are a series of observations which each represent a function, commonly to display some graphic or execute an action and are generated within the `DATA` step. Within the data structure, the `function` variable of the observation specifies which pre-defined action will be

used, and some positioning variable determines where this action will occur. Finally, attribute variables like `color`, `rotate`, or `size` can be used to specify other desired qualities. These annotation data sets are used within graphics procedures such as `GCHART` or `SGPLOT`, and by default display any drawn annotations on top of a plot created in a graphics procedure. However, in `SG` procedures, layering can be specified. This allows annotations to be drawn first, and thus underneath anything drawn by an `SG` procedure. In fact, annotation data set functions are executed one after the other from the first observation in the data set to the last, allowing for graphics to be carefully planned and generated around a plot.

By using the image function in an annotation data set with the `layer = back;` option, we can place an image under any plot, as in `PROC SGPLOT`. This allows us to take an external image, like that of a chessboard, and display it underneath an `SGPLOT` of our TSP cycle. The issue that arises from this is the board spilling out of the dimensions of the plot and into the axes. An easy solution to this is to adjust the dimensions of the image by shrinking, stretching, and cropping the image progressively until it lines up with the graph appropriately and no longer spills out of the borders. However, doing this requires a meaningful amount of iterative adjustment to not only adjust the image to match the graph path, but also then adjust the crop of the image as well.

To reduce this manual adjustment, we can add more elements to the annotation data set with polygons. By creating what is known in the film industry as a “matte”, we can create what are essentially 3 “layers” to the image. The bottom layer is the image we want to use like a map, the middle layer is our matte, and the top layer is the `SGPLOT` graph output from SAS. The matte is a series of 4 white polygons that have edges along the borders of the graph borders from `SGPLOT`. These polygons then extend to the end of the output graph and in doing so, anywhere the bottom layer image might spill over, there is a white filled polygon that covers it. The result is an annotation dataset that when used with a corresponding graph made in `PROC OPTGRAPH`, creates clean visualization in `PROC SGPLOT`.

```
data chess;
    length function $10;
    xsys = '3'; ysys = '3'; when = 'a';

    function = 'image'; style = 'fit'; image =
'/home/uXXXXXXXX/sasuser.v94/ThemePark/8x5Chessboard.png';
    drawspace = 'wallpercent'; layer = 'back';
    height = 100; width = 100;
    x1 = 50; y1 = 50;
    output;

    /* TOP POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = -100; y1 = 101.5; output;
    function = 'polycont';  x1 = 200; y1 = 101.5; output;
    function = 'polycont';  x1 = 200; y1 = 200; output;
    function = 'polycont';  x1 = -100; y1 = 200; output;

    /* BOTTOM POLYGON */
    function = 'polygon';
```

```
    display = 'fill'; fillcolor = 'white';  x1 = -100; y1 = -1.5; output;
    function = 'polycont';  x1 = 200; y1 = -1.5; output;
    function = 'polycont';  x1 = 200; y1 = -100; output;
    function = 'polycont';  x1 = -100; y1 = -100; output;

    /* LEFT POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = -1; y1 = -100; output;
    function = 'polycont';  x1 = -1; y1 = 200; output;
    function = 'polycont';  x1 = -200; y1 = 200; output;
    function = 'polycont';  x1 = -200; y1 = -100; output;

    /* RIGHT POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = 101; y1 = -100; output;
    function = 'polycont';  x1 = 101; y1 = 200; output;
    function = 'polycont';  x1 = 200; y1 = 200; output;
    function = 'polycont';  x1 = 200; y1 = -100; output;

run;
```

Now that the annotation dataset has been created for displaying and matting the underlying chessboard image, we need to ensure that the cycle created in `PROC OPTGRAPH` and the chessboard in the annotation dataset are aligned. If these are not aligned correctly, we no longer represent our cycle accurately in physical space. This is fairly simple in the case of a chessboard, as using the `xaxis` and `yaxis grid` option allows us to display a grid in our plot, this can then be used to adjust the dimensions of the plot and the chessboard image such that our grid intersects the center of each square on the board. An important decision made in the annotation dataset creation earlier was to use specify `drawspace="wallpercent"` as this allows us to specify `offsetmin` and `offsetmax` in `SGPLOT`. This is relevant because due to the nature of the discrete data in this example, our graph would be placed in a way where the datapoints in the corners of the plot, like `(1,1)` and `(8,8)`, would end up on the axes of our plot and crop part of the chessboard. By using `drawspace`, `offsetmin` and `offsetmax` we can pad the axes so this is not an issue.

```
ods html close;
ods graphics / imagename='Knights_Tour';
ods graphics on / width=4in height=6in; *this is a conversion of 0.75in per
square +0.25in width to account for extra whitespace;
ods listing gpath='/home/uXXXXXXXX/sasuser.v94/ThemePark/' image_dpi=300;
proc sgplot data = knights_tour nowall sganno=chess;
    series x=x y=y / markers lineattrs=(thickness=4 color=red);
    xaxis label = "Longitude" grid values=(1 2 3 4 5) offsetmin=0.1025
offsetmax=0.1025;
    yaxis label = 'Latitude' grid values=(1 2 3 4 5 6 7 8) offsetmin=0.06
offsetmax=0.06;
run;
```

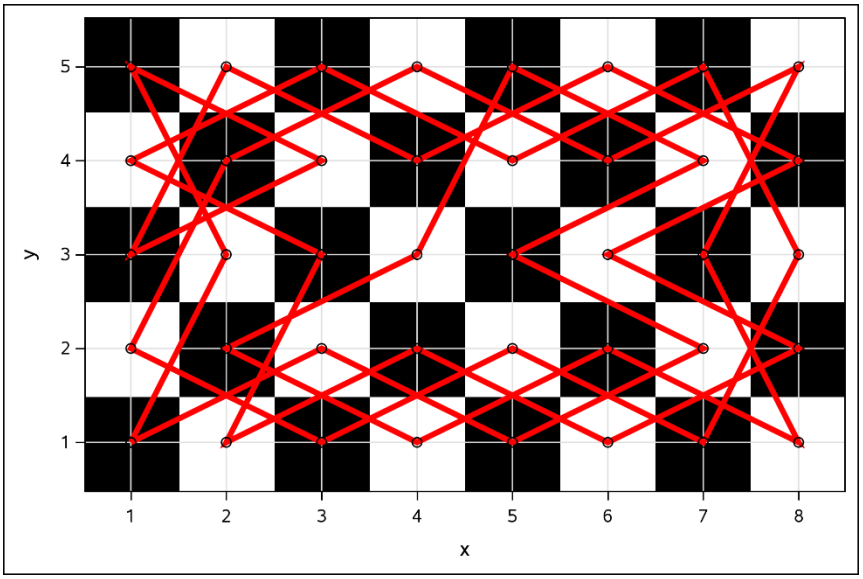

Figure 1. Example Knights Tour Visualization

CREATING A GRAPH FROM EXTERNAL DATA

The TSP is a problem which originated as a mapping problem, as such it is relevant to want to display a TSP on a map to represent the path through physical space. Our second example is the TSP on mapping data of the Disneyland theme park in Anaheim, California. This data was collected from queue-times.com by using Python to web-scrape one year's worth of ride wait time information from 2023 to 2024. The data was cleaned and then aggregated such that each ride has an average wait time for the entire year. Latitude and longitude were collected for each ride, and these were used to calculate an average walking time between each ride using an average walking speed of 2.8 miles per hour. Finally, an edge list was constructed so each ride had an edge to every other ride, and the weight for that edge was the wait time at the destination ride plus the walking time to the destination. In this case, the desired optimization for the time in the park is to minimize both waiting time and walking time. This was exported into a .csv file and imported into SAS, where `PROC OPTGRAPH` was used to construct the tour through the park.

```
proc optgraph data_links = TSP.LUT;
    data_links_var
    from = start
    to = end
    weight = weight;
    tsp out = TSP.Tour;
    run;
```

OPTGRAPH OUTPUT

By default, SAS outputs the optimized tour as a table. As there are 29 rides in the park, there are 29 selections that must be made to construct the TSP. As before, for brevity's sake here are the first five rows of output. The full output can be found in Appendix D.

Obs	origin	destination	cost
1	ALIW	BTMR	34.337
2	ASTR	BTMR	22.897
3	ASTR	BLAB	20.875
4	BLAB	TOUR	22.601
5	JNGL	TOUR	22.958

Table 2. The first five rides visited

With this optimized tour, we can use the same methods as before to construct a plot to display this graph on top of a map.

PROCESSING THE OUTPUT FOR PLOTTING

In order to take the (origin,destination) string output of `OPTGRAPH` and use it for plotting, we need to process the origin and destination pairs such that they are numeric values that SAS can interpret as a series in `SGPLOT`. The resulting dataset converts each (origin, destination) edge pair into a single column of destination labels, and so the first and last observations are the same destination.

```
data TSP.Tour1;
    drop end;
    set TSP.Tour (obs=1) TSP.Tour;
    rownum=_n_;
    if start = lag(start) then start = end;
    if start = lag(end) then start = end;
    if end = lag(start) then start = start;
    if end = lag(end) then start = start;
run;
```

After adjusting the dataset to be compatible with `SGPLOT`, we need to associate the latitude longitude pairs for each ride. This can be done using `PROC SQL` with a .csv file containing a list of each ride and their latitude longitude pair.

```
proc sql;
   create table TSPTourLL as
   select unique Tour1.*, latlong.lat as lat1, latlong.long as long1
      from TSP.Tour1 left join TSP.latlong
      on Tour1.start = latlong.VAR1;
quit;
```

VISUALIZING THE TSP ROUTE ON A MAP

An important item when visualizing graphs in the real world is how said graphs physically interact with our world. For example, there is little value in seeing a path of directions from point A to point B if there is no orientation for what north, south, east, and west are; and especially if the path is plain with no street

labeling or general topography. So, constructing a graph with a visualization for orientation such as on a map is highly valuable technique.

Plotting on a map in SAS is relatively straightforward, as `PROC GMAP` allows geospatial map data sets to be used in plotting, and the default `SASHELP` and `MAPSAS` data sets contain a plethora of mapping coordinates for plotting the borders of different states and countries across the world. These datasets are very useful, however in the case of smaller scale path mapping, or where specific visuals are desired in the graphics, these standard techniques are not applicable. In this application with Disneyland attractions, using `PROC GMAP` with the California data set would produce a map that is too large to understand the TSP cycle, and even if it could be altered to make the TSP visible, important visual landmarks or desired graphical design would be absent. The solution to this is to display the TSP cycle on top of an image of a map, and, as we have already shown, SAS has the tools to do this.

Using the annotation data set, we can display a Disneyland map underneath our graph plot, and using the matting techniques discussed previously, we can minimize the amount of adjustments necessary to create a plot.

```
data map;
    length function $10;
    xsys = '3'; ysys = '3'; when = 'a';
    function = 'image'; style = 'fit'; image =
'/home/uXXXXXXXX/sasuser.v94/ThemePark/DisneyBlankMap.png';
    drawspace = 'wallpercent'; layer = 'back';
    height = 130; width = 187;
    x1 = 58; y1 = 41;
    output;

    /* TOP POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = -100; y1 = 101.5; output;
    function = 'polycont';  x1 = 200; y1 = 101.5; output;
    function = 'polycont';  x1 = 200; y1 = 200; output;
    function = 'polycont';  x1 = -100; y1 = 200; output;

    /* BOTTOM POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = -100; y1 = -1.5; output;
    function = 'polycont';  x1 = 200; y1 = -1.5; output;
    function = 'polycont';  x1 = 200; y1 = -100; output;
    function = 'polycont';  x1 = -100; y1 = -100; output;

    /* LEFT POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = -1; y1 = -100; output;
    function = 'polycont';  x1 = -1; y1 = 200; output;
    function = 'polycont';  x1 = -200; y1 = 200; output;
```

```
    function = 'polycont';  x1 = -200; y1 = -100; output;

    /* RIGHT POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = 101; y1 = -100; output;
    function = 'polycont';  x1 = 101; y1 = 200; output;
    function = 'polycont';  x1 = 200; y1 = 200; output;
    function = 'polycont';  x1 = 200; y1 = -100; output;

run;
```

With the annotation dataset created for matting the underlying map image, aligning the path created by `PROC OPTGRAPH` and the map in `PROC SGPLOT` is important as if the path does not match the map accurately, it no longer represents the desired path through physical space. We can align our image by generating 2 images of identical dimensions and layout, but one of them has anchor points for the path to align with. These anchor points should align with points along the generated path and need to exactly represent the location on the map. In our case, latitude and longitude were used to represent the locations on our map, and so we create two anchor points using rides along our path with the exact latitude and longitude used in the dataset passed to proc opt graph. These two anchor points are also placed in such a way where they do not form a flat line along the x or y axis of the graph. This allows us to calibrate the dimensions of the underlying map image in the annotation dataset such that it is scaled and positioned correctly. Ideally, the more points that are exactly represented on the map image explicitly, the more accurate of an alignment can be made, but fundamentally only two points with differing latitude and longitude are necessary to get an approximation for image adjustment. The iterative process for aligning the image and then path are displayed in Figure 2. Iterative Map Process.

```
ods html close;
ods graphics / imagename='Map_Lines';
ods listing gpath='/home/uXXXXXXXX/sasuser.v94/ThemePark/' image_dpi=300;
proc sgplot data = TSPTemp1 nowall sganno=map; * noborder;
    series x=long1 y=lat1 / markers lineattrs=(thickness=2 color=blue)
datalabel=start;
    xaxis label = "Longitude" grid;
    yaxis label = 'Latitude' grid;
run;
ods _all_ close;
```

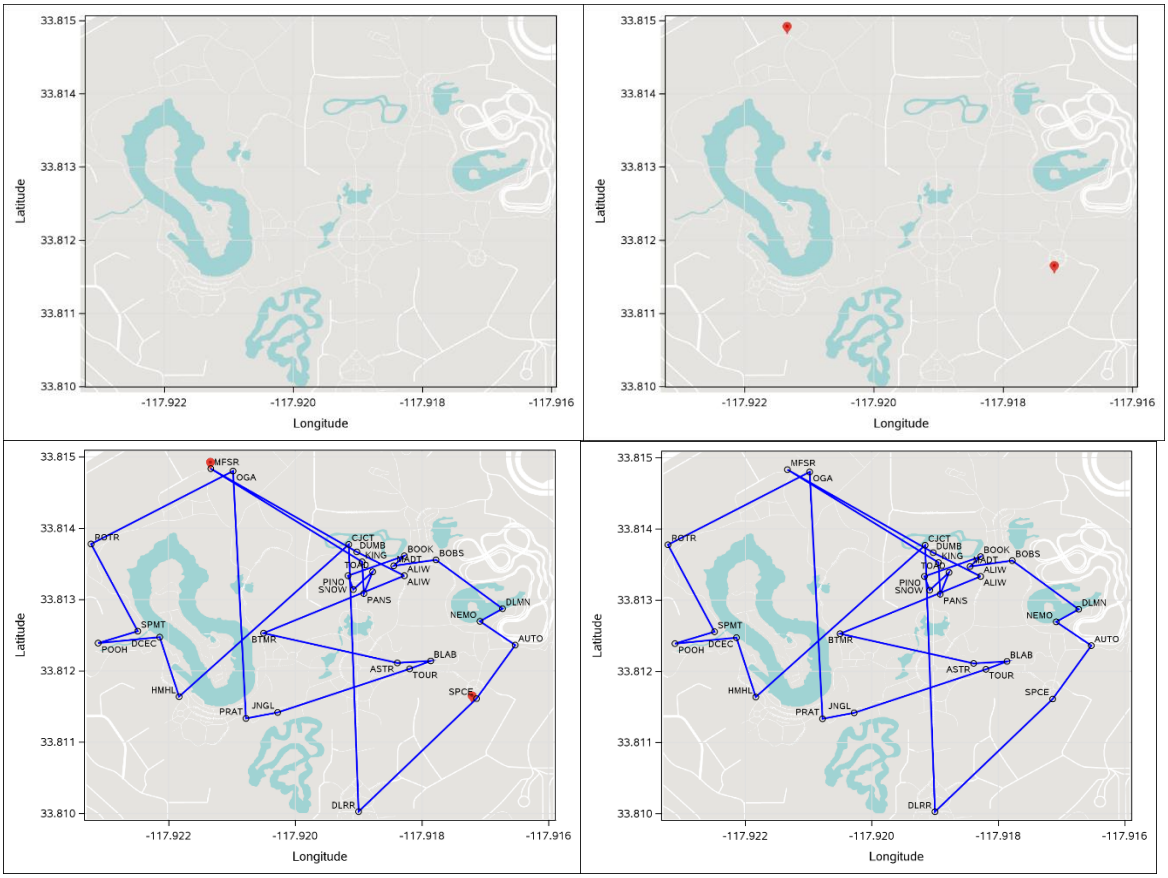

Figure 2. Iterative Map Process

The advantage of this method is not only the ease of developing multi-layered plots and aligning distinct elements effectively, but also in that any image can be used as the under-lay image. This allows us to take an official Disneyland map and, through some adjustment to the image, align our TSP with an artist rendering of the park in Figure 3. Example Artistic Map. The alignment of the graph with the map of the park is less precise than the previous one using images from Google Maps, as the markers used by Disney for their rides are not always consistent with the exact latitude and longitude of the park ride. Additionally, the image itself has some perspective change to it which cannot be fully corrected without detailed image correction.

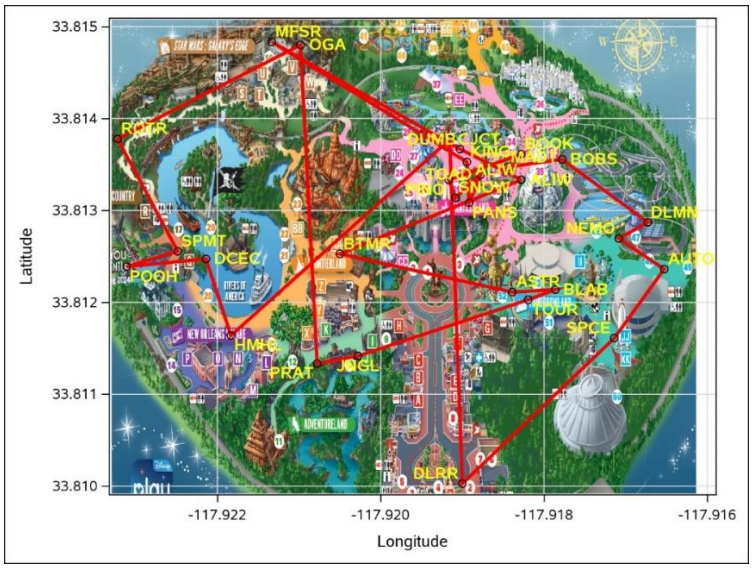

Figure 3. Example Artistic Map

CONCLUSION

SAS is known to be a visualization powerhouse, but, incredibly, it can be pushed even further with some interesting techniques to create truly unique graphics. By using SAS with TSP data, one can visualize these data points in a geometric figure, such as a chess board or from geographic coordinates, on any image. This can help users to understand an optimized tour for the TSP, as generated by `OPTGRAPH`. In this paper we have provided code to generate a TSP and visualize it using the annotation dataset as a guide for other users to generate customized graphics of their own.

REFERENCES

[1] Bevil, Dewayne. “Super Bowl Star Jalen Hurts Going to Disney World.” *Yahoo! Sports*, Orlando Sentinel, sports.yahoo.com/super-bowl-star-jalen-hurts-165700010.html. Accessed 18 Aug. 2025.

[2] Euler, Leonhard. 1736. "Solutio problematis ad geometriam situs pertinentis". *Comment. Acad. Sci. U. Petrop* 8, 128–40.

[3] Euler, Leonhard. 1759. “Solution d’une question curieuse que ne paroit soumise à aucune analyse”. Mémoires de l’académie des sciences de Berlin 15 (1759) 1766 p. 310-337

[4] Bellmore, M., and G. L. Nemhauser. “The Traveling Salesman Problem: A Survey.” *Operations Research*, vol. 16, no. 3, 1968, pp. 538–58. *JSTOR*, http://www.jstor.org/stable/168581.

[5] Alanzi, E., Menai, M.E.B. Solving the traveling salesman problem with machine learning: a review of recent advances and challenges. Artif Intell Rev 58, 267, 2025. https://doi.org/10.1007/s10462-025-11267-x

[6] Kaplan, Emily. “‘I’m Going to Disneyland!’ How Simple Phrase Became Super Bowl Lore.” *SI*, Sports Illustrated, 29 Jan. 2025, www.si.com/nfl/2015/11/10/super-bowl-disney.

[7] “Disneyland Statistics - Daily & Yearly Attendance, Size, Operating Info.” *Mickey Visit - Disney News & Planning Tips*, 30 Dec. 2024, mickeyvisit.com/disneyland-statistics/#:~:text=In%202023%2C%20Disneyland%27s%20total%20amount%20of%20visitors%20between,Park%20%2847%2C260%29%20and%20Disney%20California%20Adventure%20Park%20%2827%2C397%29.

[8] *Dada (USA) – Dizz Knee Land Lyrics | Genius Lyrics*, genius.com/Dada-usa-dizz-knee-land-lyrics. Accessed 14 Aug. 2025.

ACKNOWLEDGMENTS

The authors would like to thank SESUG for inviting us to share our interest and codework in the Traveling Salesman Problem. We also collectively acknowledge SAS and The School of Data Science and Analytics as a unit within The College of Computing and Software Engineering at Kennesaw State University for travel support to present this work. We also appreciate the individuals within all three organizations who have either willingly or unwillingly listened to us discuss this work.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Caleb Greski
PhD Candidate in Data Science
Kennesaw State University
cgreski@students.kennesaw.edu

Joe DeMaio
Professor of Mathematics and Data Science
Kennesaw State University
jdemaio@kennesaw.edu
https://facultyweb.kennesaw.edu/jdemaio/publications.php

APPENDIX A WEIGHTED KNIGHT'S TOUR CODE

```
data board;
rows = 5;
columns = 8;
do i = 1 to rows;
do j = 1 to columns;
origin =  (i || j);
/* input legal moves of the knight */
/* up 2 right 1 */
if i>=3 and j<= columns-1 then do
      destination = (i-2 || j+1);
      cost=5;
      output;
      end;
/* up 1 right 2 */
if i>=2 and j<= columns-2 then do
      destination  = (i-1 || j+2);
      cost=3;
      output;
      end;
/* down 1 right 2 */
if i<=rows-1 and j<= columns-2 then do
      destination  = (i+1 || j+2);
      cost=3;
      output;
      end;
/* down 2 right 1 */
if i<=rows-2 and j<= columns-1 then do
      destination  = (i+2 || j+1);
      cost=5;
      output;
      end;
end;
end;
run;

proc optgraph
data_links = board;
*create graph from variables;
data_links_var
from = origin
to = destination
weight=cost;
* write cycle to file closed_knights_tour;
tsp out = closed_knights_tour;
```

```
run;

proc print data=work.closed_knights_tour;
* print to results file;
run;
```

APPENDIX B FULL OUTPUT WEIGHTED KNIGHT'S TOUR

Obs	origin		destination		cost
1	1	1	2	3	3
2	2	3	1	5	3
3	1	5	2	7	3
4	3	5	2	7	3
5	3	5	4	7	3
6	5	5	4	7	3
7	3	4	5	5	5
8	2	2	3	4	3
9	2	2	1	4	3
10	1	4	2	6	3
11	2	6	1	8	3
12	3	7	1	8	5
13	3	7	5	8	5
14	4	6	5	8	3
15	5	4	4	6	3
16	4	2	5	4	3
17	2	1	4	2	5
18	2	1	1	3	3
19	1	3	2	5	3
20	2	5	1	7	3
21	1	7	3	8	5
22	5	7	3	8	5
23	4	5	5	7	3
24	5	3	4	5	3
25	4	1	5	3	3
26	4	1	3	3	3
27	1	2	3	3	5

Obs	origin		destination		cost
28	1	2	2	4	3
29	2	4	1	6	3
30	1	6	2	8	3
31	3	6	2	8	3
32	3	6	4	8	3
33	5	6	4	8	3
34	4	4	5	6	3
35	5	2	4	4	3
36	3	1	5	2	5
37	3	1	4	3	3
38	5	1	4	3	3
39	5	1	3	2	5
40	1	1	3	2	5

APPENDIX C FULL OUTPUT WEIGHTED DISNEYLAND TOUR

```
data knights_tour;
    drop destination;
    set knights_tour (obs=1) knights_tour;
    rownum=_n_;
    if origin = lag(origin) then origin = destination;
    if origin = lag(destination) then origin = destination;
    if destination = lag(origin) then origin = origin;
    if destination = lag(destination) then origin = origin;
    x = substr(origin, notspace(origin,1), 1);
    y = substr(origin, notspace(origin,-99), 1);
run;

data chess;
    length function $10;
    xsys = '3'; ysys = '3'; when = 'a';
    function = 'image'; style = 'fit'; image =
'/home/uXXXXXXXX/sasuser.v94/ThemePark/8x5Chessboard.png';
    drawspace = 'wallpercent'; layer = 'back';
    height = 100; width = 100;
    x1 = 50; y1 = 50;
    output;

    /* TOP POLYGON */
```

```
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = -100; y1 = 101.5; output;
    function = 'polycont';  x1 = 200; y1 = 101.5; output;
    function = 'polycont';  x1 = 200; y1 = 200; output;
    function = 'polycont';  x1 = -100; y1 = 200; output;

    /* BOTTOM POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = -100; y1 = -1.5; output;
    function = 'polycont';  x1 = 200; y1 = -1.5; output;
    function = 'polycont';  x1 = 200; y1 = -100; output;
    function = 'polycont';  x1 = -100; y1 = -100; output;

    /* LEFT POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = -1; y1 = -100; output;
    function = 'polycont';  x1 = -1; y1 = 200; output;
    function = 'polycont';  x1 = -200; y1 = 200; output;
    function = 'polycont';  x1 = -200; y1 = -100; output;

    /* RIGHT POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = 101; y1 = -100; output;
    function = 'polycont';  x1 = 101; y1 = 200; output;
    function = 'polycont';  x1 = 200; y1 = 200; output;
    function = 'polycont';  x1 = 200; y1 = -100; output;

run;


ods html close;
ods graphics / imagename='Knights_Tour';
ods graphics on / width=4in height=6in;
*this is a conversion of 0.75in per square +0.25in width to account for extra
whitespace;
ods listing gpath='/home/uXXXXXXXX/sasuser.v94/ThemePark/' image_dpi=300;

proc sgplot data = knights_tour nowall sganno=chess;
    series x=x y=y / markers lineattrs=(thickness=4 color=red);
    xaxis label = "Longitude" grid values=(1 2 3 4 5) offsetmin=0.1025
offsetmax=0.1025;
    yaxis label = 'Latitude' grid values=(1 2 3 4 5 6 7 8) offsetmin=0.06
offsetmax=0.06;
run;
```

APPENDIX D FULL OUTPUT WEIGHTED DISNEYLAND TOUR

Obs	origin	destination	weight
1	ALIW	BTMR	34.337152393
2	ASTR	BTMR	22.897301627
3	ASTR	BLAB	20.874882345
4	BLAB	TOUR	22.601005901
5	JNGL	TOUR	22.958074292
6	JNGL	PRAT	20.864312383
7	OGA	PRAT	11.897357604
8	OGA	ROTR	9.9186838552
9	SPMT	ROTR	15.891146673
10	SPMT	POOH	14.68050202
11	DCEC	POOH	14.243664591
12	DCEC	HMHL	14.32590178
13	CJCT	HMHL	20.077476029
14	CJCT	DLRR	19.928925994
15	DLRR	SPCE	24.603345107
16	AUTO	SPCE	21.879439048
17	AUTO	NEMO	21.383305118
18	DLMN	NEMO	8.7066370622
19	DLMN	BOBS	9.8391069315
20	MADT	BOBS	16.39486404
21	MADT	BOOK	15.854168445
22	PINO	BOOK	17.221970564
23	PINO	SNOW	16.356873649
24	TOAD	SNOW	20.166263884
25	TOAD	PANS	20.121462556
26	KING	PANS	8.5457052128
27	KING	MFSR	11.408468192
28	DUMB	MFSR	24.02116884
29	ALIW	DUMB	32.397899514

APPENDIX E DISNEYLAND WEIGHTED TSP AND VISUALIZATION

```
proc import
datafile = "/home/uXXXXXXXX/sasuser.v94/ThemePark/DisneyLUT.csv"
out = TSP.LUT
dbms = csv;
run;

proc optgraph data_links = TSP.LUT;
data_links_var
from = start
to = end
weight = weight;
tsp out = TSP.Tour;
run;

data TSP.Tour1;
    drop end;
    set TSP.Tour (obs=1) TSP.Tour;
    rownum=_n_;
    if start = lag(start) then start = end;
    if start = lag(end) then start = end;
    if end = lag(start) then start = start;
    if end = lag(end) then start = start;
run;

proc sql;
   create table TSPTourLL as
   select unique Tour1.*, latlong.lat as lat1, latlong.long as long1
      from TSP.Tour1 left join TSP.latlong
      on Tour1.start = latlong.VAR1;
quit;

data map;
    length function $10;
    xsys = '3'; ysys = '3'; when = 'a';

    function = 'image'; style = 'fit'; image =
'/home/uXXXXXXXX/sasuser.v94/ThemePark/DisneyBlankMap.png';
    drawspace = 'wallpercent'; layer = 'back';
    height = 130; width = 187;
    x1 = 58; y1 = 41;
    output;

    /* TOP POLYGON */
```

```
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = -100; y1 = 101.5; output;
    function = 'polycont';  x1 = 200; y1 = 101.5; output;
    function = 'polycont';  x1 = 200; y1 = 200; output;
    function = 'polycont';  x1 = -100; y1 = 200; output;

    /* BOTTOM POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = -100; y1 = -1.5; output;
    function = 'polycont';  x1 = 200; y1 = -1.5; output;
    function = 'polycont';  x1 = 200; y1 = -100; output;
    function = 'polycont';  x1 = -100; y1 = -100; output;

    /* LEFT POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = -1; y1 = -100; output;
    function = 'polycont';  x1 = -1; y1 = 200; output;
    function = 'polycont';  x1 = -200; y1 = 200; output;
    function = 'polycont';  x1 = -200; y1 = -100; output;

    /* RIGHT POLYGON */
    function = 'polygon';
    display = 'fill'; fillcolor = 'white';  x1 = 101; y1 = -100; output;
    function = 'polycont';  x1 = 101; y1 = 200; output;
    function = 'polycont';  x1 = 200; y1 = 200; output;
    function = 'polycont';  x1 = 200; y1 = -100; output;

run;

ods html close;
ods graphics / imagename='Map_Lines';
ods listing gpath='/home/uXXXXXXXX/sasuser.v94/ThemePark/' image_dpi=300;
proc sgplot data = TSPTourLL nowall sganno=map; * noborder;
    series x=long1 y=lat1 / markers lineattrs=(thickness=2 color=blue)
datalabel=start;
    xaxis label = "Longitude" grid;
    yaxis label = 'Latitude' grid;
run;
ods _all_ close;
```