A TOUR AROUND US NATIONAL PARKS: A MODIFIED TSP

JUNYOUNG J. CHOI, VICTOR G. NEGRON, AND BRIAN J. WHEATMAN

Abstract. This project focuses on a certain variation of the Traveling Salesman Problem (TSP), which is known as the Prize Collecting Traveling Salesman Problem. For this project, we will be focusing on US National Parks instead of cities. First, we develop an algebraic model that generalizes the problem as a whole. Then we gather the data required to experiment with the model. From here we expect to get results, which will be then be analyzed. Finally, we determine if the model works and what changes can be made to improve the model.

Contents

1. Introduction

The classic Traveling Salesman Problem (TSP), an important problem in operations research, can be stated as the following:

Date: May 11, 2015.

> "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?" [3]

We will explore a more advanced version of the TSP known in general as the Prize Collecting Traveling Salesman Problem. This is the problem when given a set of cities in which each city gives a positive utility to visit, and a maximum distance that can be traveled, to find the path that gains the most utility. Consider the set of 59 US National Parks, located across 27 states in the country and in various U.S. territories. National parks are operated by the National Park Service, and have been an important part of history since Yellowstone was declared the first national park in 1872. National parks are revered for their beauty, diverse ecosystems, and geological features, which make them a good choice as an underlying data set for this project.

We relax the constraint that each site must be visited. Instead, we assign each site an arbitrary utility, based on the number of visitors to the park each year, gained by visiting a particular site. In addition, there is a maximal distance (say "fuel") that we can travel to visit each site, which means that traveling along a road can be effectively thought of as incurring a cost. Other parameters of the TSP are kept, including the requirement that we return to our city of origin (we shall assume Boston, MA in this case). We also introduce a sub-tour elimination constraint to prevent cycles. This constraint is created to force the model to only consider one cycle in the solution, rather than having multiple cycles on different parks within the same solution. That way we can only find the optimal solution of the problem that starts from Boston, which is the starting point of the problem.

2. Algebraic Model

We give the general formulation (for some set of parks) below. We apply the model below to the set of 59 US National Parks and Boston. There are a total of n national parks or in this case, $n = 60$, including Boston. Assume we start at Boston, or point 1. We have the following tables and the LP model. The first table lists the decision variables and other notations that will be used in the model, each of which is defined in the table. With this information, we can formulate the model. The second table gives a description for the objective function and for each constraint of the model. We found that there are six constraints that are required in the model.

TABLE 1. List of all notations (decision variables, parameters, and sets/subsets)

Notation	Decision Variable, Parameter, Set, or Subset	Definition
v_j	Decision Variable	The park we choose to travel to or not.
x_{ij}	Decision Variable	The road chosen to get to park j from park i.
c_j	Parameter	The utility coefficient of traveling to park j, which is determined based on the popularity of the park, or the total number of recreation visitors in a year.
y_{ij}	Parameter	The distance between park i and park j, which corresponds to x_{ij}. This value was computed using GPS coordinates for each park.
d	Parameter	The maximum distance you can travel in your entire trip.
V	Set	It is the set of parks $\{1, ..., n\}$.
V_1	Subset	A subset of V for the parks we actually go to $(v_j = 1)$.
S	Subset	A subset of V_1 that is used to eliminate any sub-tours.

$$max \sum_{j=1}^{n} c_j v_j$$

$$s.t. \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} y_{ij} \leq d$$

$$v_j = \sum_{i=1}^{n} x_{ij}$$

$$\sum_{i \in S} \sum_{j \in V_1 \setminus S} x_{ij} \geq 1, \forall_S \subset V_1$$

$$x_{ii} = 0, \forall_i \in V_1$$

$$v_j \in \{0, 1\}, \forall_j \in V$$

$$x_{ij} \in \{0, 1\}, \forall_i, \forall_j \in V$$

TABLE 2. Description of Objective Function and Constraints

Objective Function	$max \sum_{j=1}^{n} c_j v_j$	Our objective function is to maximize utility by selecting the parks we travel to. This function will be the sum of all parks times the utility coefficient of the park.
Constraint 1	$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} y_{ij} \leq d$	The first constraint indicates that the sum of the distances of the roads we choose ($x_{ij} = 1$) must be less than or equal to the maximum possible distance d.
Constraint 2	$v_j = \sum_{i=1}^{n} x_{ij}$	The second constraint tells us that the park chosen is the sum of all the roads chosen to get to the park.
Constraint 3	$\sum_{i \in S} \sum_{j \in V_1 \setminus S} x_{ij} \geq 1, \forall_S \subset V_1$	The third constraint eliminates any sub-tour that may be created for all subsets of S in V_1, thus avoiding any alternate cycle that does not start from the initial point.
Constraint 4	$x_{ii} = 0, \forall_i \in V_1$	The fourth constraint is the no self path constraint, meaning that after visiting the park you cannot stay in the park or leave the park only to come back again. This applies for all elements in V_1.
Constraint 5	$v_j \in \{0, 1\}, \forall_j \in V$	It defines v_j as a binary variable described below for all elements in V. $v_j = \begin{cases} 1 & \text{if park j is chosen} \\ 0 & \text{if park j is not chosen} \end{cases}$
Constraint 6	$x_{ij} \in \{0, 1\}, \forall_i, \forall_j \in V$	It defines x_{ij} as a binary variable described below for all elements in V. $x_{ij} = \begin{cases} 1 & \text{if road } ij \text{ is chosen} \\ 0 & \text{if road } ij \text{ is not chosen} \end{cases}$

3. DATA

The data on all 59 US National Parks, including their GPS coordinates, land area, number of visitors annually, and dates established is given in Appendix C. This information was obtained from [2]. Data that was not used but was collected for completeness includes dates established and land area. The GPS coordinates were used to compute

distances, which are essential to the model. The number of annual visitors, say ζ_j, was used to establish the utility values for each national park. In particular, the utility value $c_j = \zeta_j^{1/5}$ was used to achieve appropriate scale, but this of course was an arbitrary choice based on a desire to have utilities values in a small range. The true distances are computed in the Julia program and are not presented in the raw data in Appendix C. The utilities were computed in a separate raw data file that was read into the Julia file.

4. Results

Using the Julia/JuMP code presented in Appendix B, given a set of input maximal distances the utility values were computed, which are shown in the below table. The plots for the variational analysis are presented in Appendix A.

Table 3. The maximum utility achieved for each of the following max distances

max distance (km)	value
18000	693.07
16000	653.50
15000	624.47
14000	589.02
13000	543.58
12000	494.60
10000	386.79
9000	325.12
8000	260.99
7000	152.19
6000	131.95
5000	108.62
4000	102.23
3000	59.44
2500	37.04
2100	34.64
TSP Distance (km)	TSP Value
35805.21	832.34

FIGURE 1. The optimal solution for maximum distance of $d = 18,000$ km. The map shows the path from each park to the next following the great circle paths.

FIGURE 2. The optimal solution for the Traveling Salesman Problem that goes through all the National Parks in the United States, including Alaska, Hawaii, and other US territories.

5. Discussion and Additional Analyses

We found that the relationship between the maximum distance and the value of the path does not abide by any simple rules. While it is monotonically increasing, as it has to because at any point it will either take the old path or one that does better, it is neither concave or convex. This makes sense because both the distance needed to reach an additional park and the value of that park are both somewhat arbitrary.

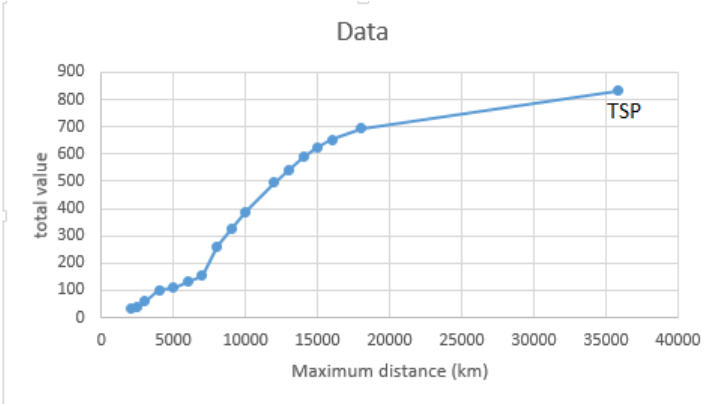

FIGURE 3. This graph shows the optimal objective value for each maximum distance d in kilometers.

6. Conclusion

Our project involved analyzing a variation of the classic TSP. We have considered the set of 59 US National Parks as the set of "parks" (or destinations) and assigned a reward value/utility based on a function of the number of annual visitors to each park. There was a maximal distance that one could travel, which in practice could be modeled by a limitation of fuel and/or travel budget, and for each value of maximal distance a feasible IP solution was generated using Julia/JuMP. The result and the model are constructive in that they generalize the TSP; at a certain maximal critical point, the solution to our model converges to the TSP given a large enough maximal distance input, but will diverge again once it is no longer constrained to only go that distance and might find a different path first that still visits all of the cities. However, it is very often the case that one cannot necessarily visit all points in a particular set of destinations (which is the TSP), so our model provides a solution to that problem through variational analysis of the maximal distance.

7. Further Considerations and Improvements

Most further considerations include changing our set of locations, or the start location to different ones, perhaps famous landmarks around the world such as National Monuments of each country, or changing how the utility of each location is calculated. However, this would merely change the data set and not really lead to a new investigation. We could have also added a constraint stating that the sum of the costs

of visiting each national park to be less than or equal to the maximum budget given for a trip.

A more interesting investigation could see how certain desires would affect the overall outcome. This could be things like a desire to visit as many states as possible, which could be implemented by also having a utility of visiting each state and not just each park, or perhaps certain parks are wanted to be seen above the rest and thus are required to be visited. Another case would be instead of a single person trying to visit as many parks as possible, it would be to have a group of people trying to visit as many parks as possible and thus while they all start and end at the same location, they are trying to visit as many different parks as possible as a group.

Another thing we could have taken into consideration is the season we choose to visit a certain location, Spring, Summer, Fall, or Winter. For example, it would be less costly to visit the parks located in the north during the Summer or Fall to avoid snow hazards. Another example is to vary the utility coefficient of a given park based on the season we decide to visit. Of course we then need to add a time constraint and the time it takes to get from one park to another, which would prove to be too difficult.

Acknowledgments

The authors acknowledge Prof. Jim Orlin, TA's Chiwei Yan and Charles Thraves of 15.053, and fellow classmates who reviewed this paper for their feedback on this project.

References

[1] Movable Type Scripts. *Calculate distance, bearing and more between Latitude/Longitude points*, `http://www.movable-type.co.uk/scripts/latlong.html`, 2015.
[2] National Parks Service. `http://www.nps.gov/index.htm`, 2015.
[3] Wikipedia. *Traveling Salesman Problem*, 2015.

Appendix A. Maps from Results

Asides from the maps that we showed in the Results section, we also got 14 additional maps presented in the following pages, each with different values of d. These values of d are included in the graph we showed in the Discussion and Additional Analysis section.

FIGURE 4. The optimal solution for maximum distance of $d = 2,100$ km.

FIGURE 5. The optimal solution for maximum distance of $d = 2,500$ km.

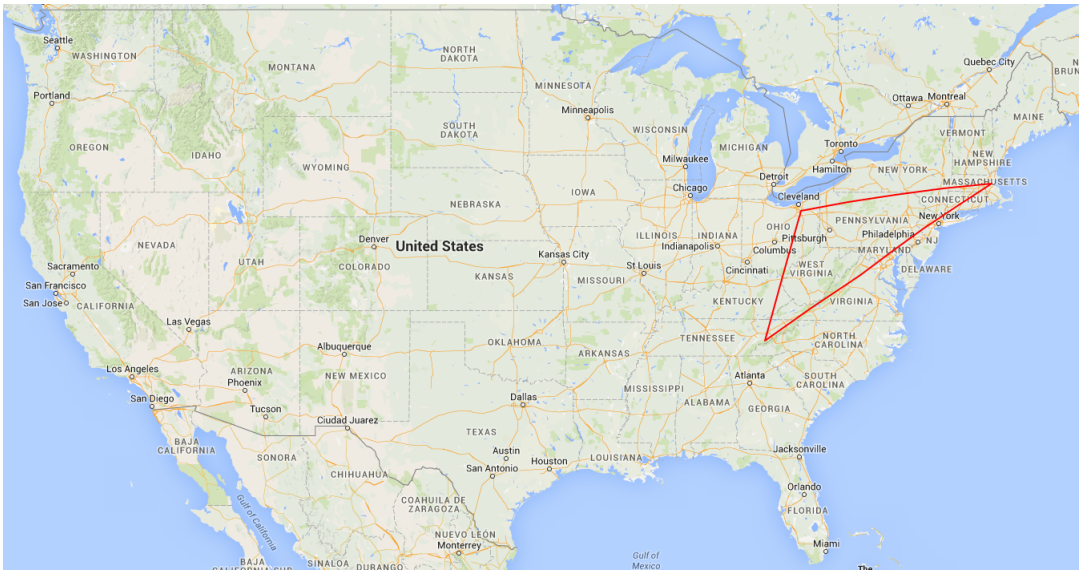

FIGURE 6. The optimal solution for maximum distance of $d = 3,000$ km.

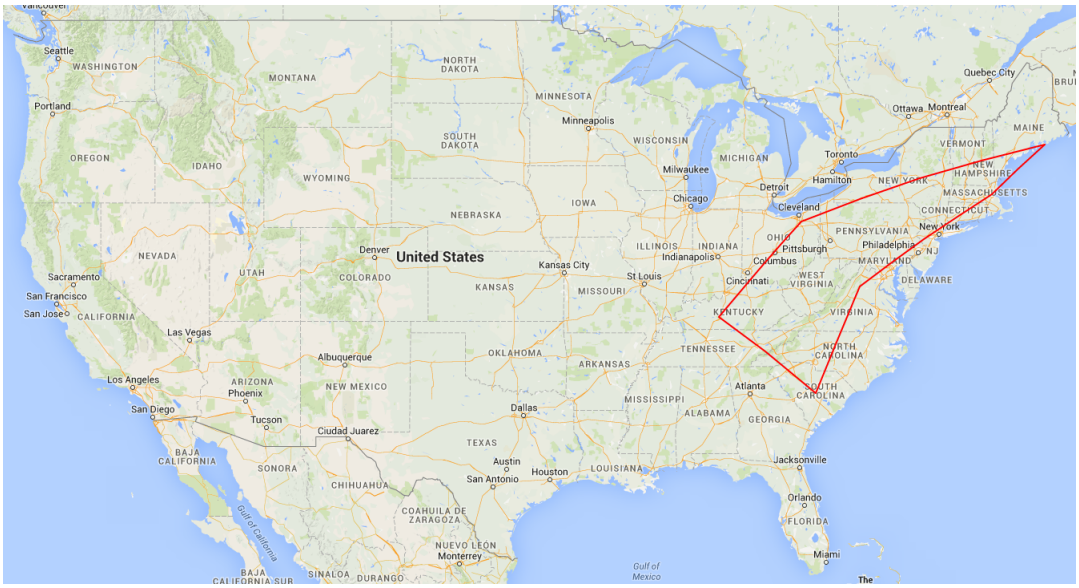

FIGURE 7. The optimal solution for maximum distance of $d = 4,000$ km.

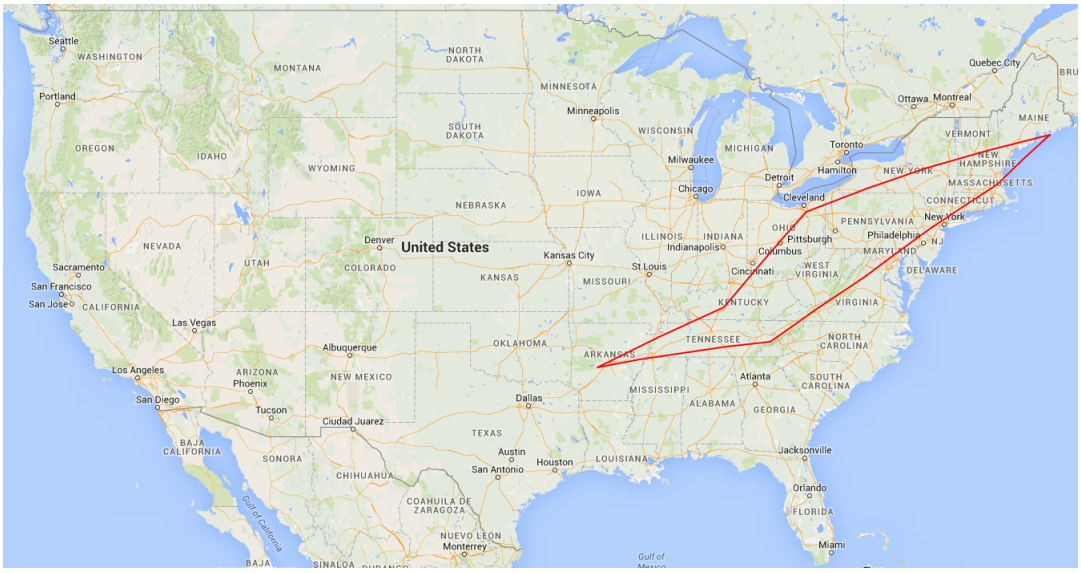

FIGURE 8. The optimal solution for maximum distance of $d = 5,000$ km.

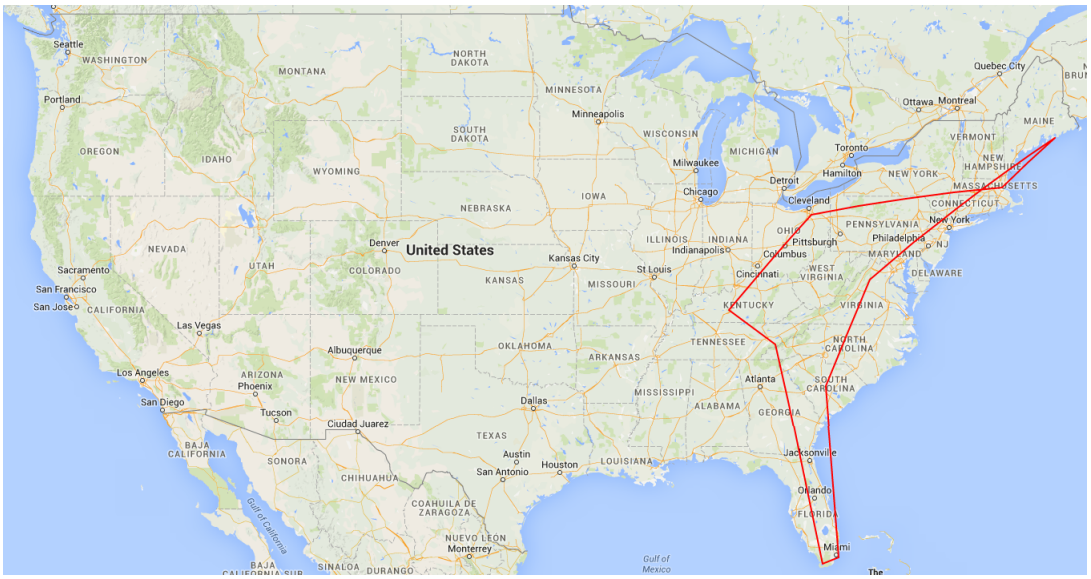

FIGURE 9. The optimal solution for maximum distance of $d = 6,000$ km.

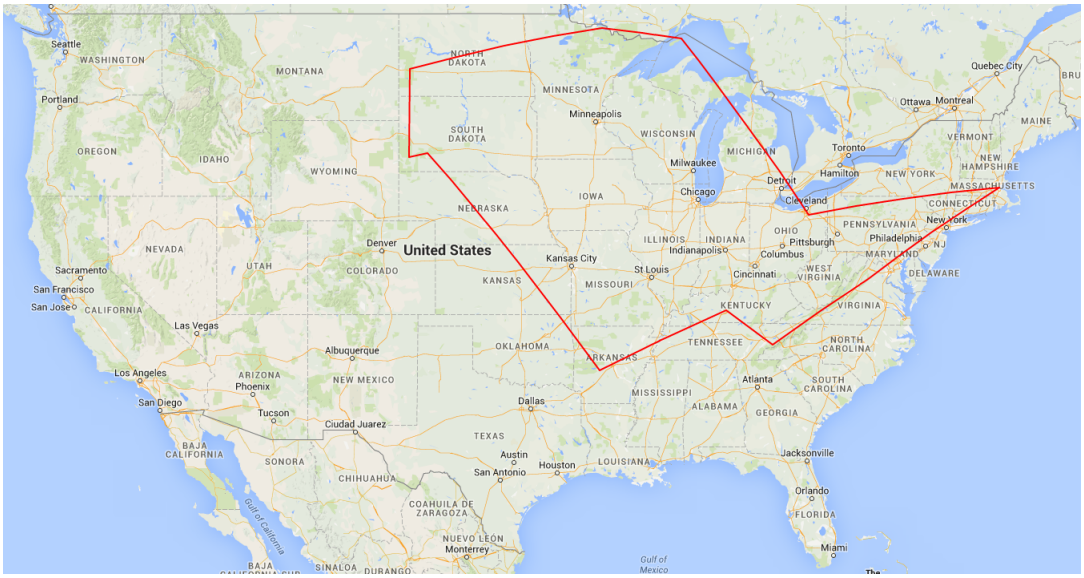

FIGURE 10. The optimal solution for maximum distance of $d = 7,000$ km.

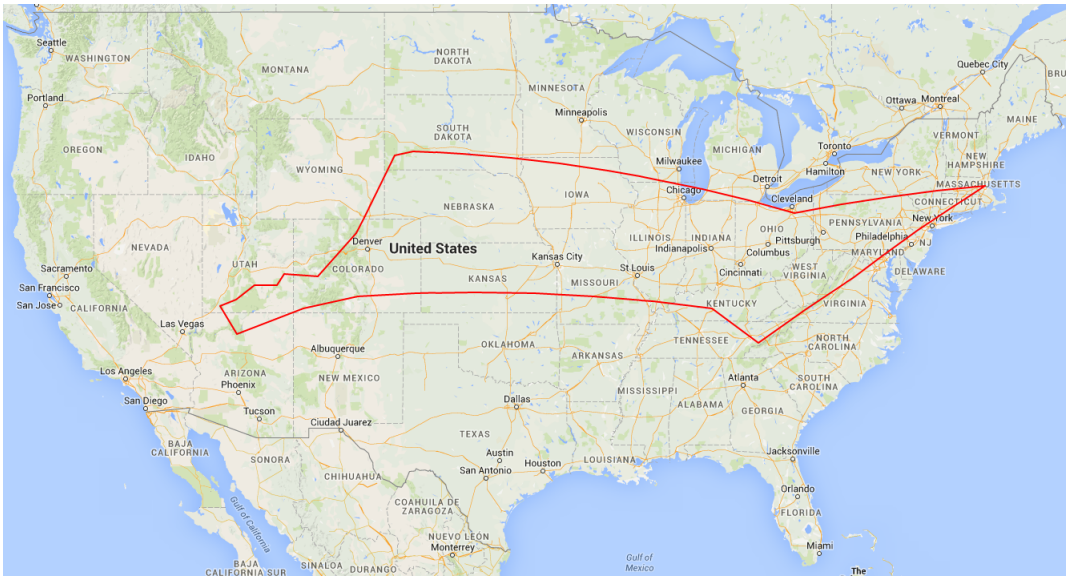

FIGURE 11. The optimal solution for maximum distance of $d = 8,000$ km.

FIGURE 12. The optimal solution for maximum distance of $d = 9,000$ km.

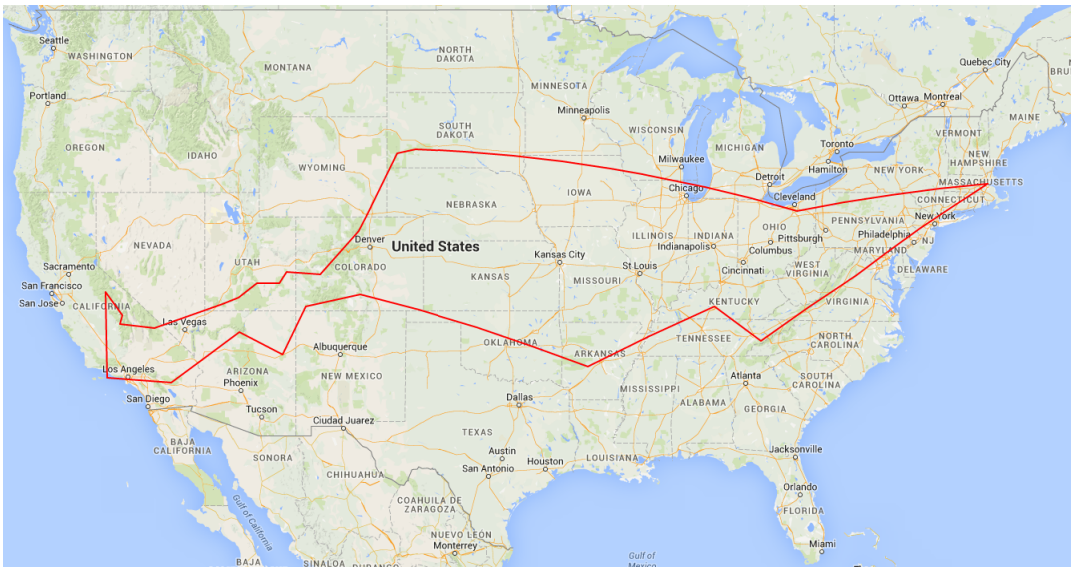

FIGURE 13. The optimal solution for maximum distance of $d = 10,000$ km.

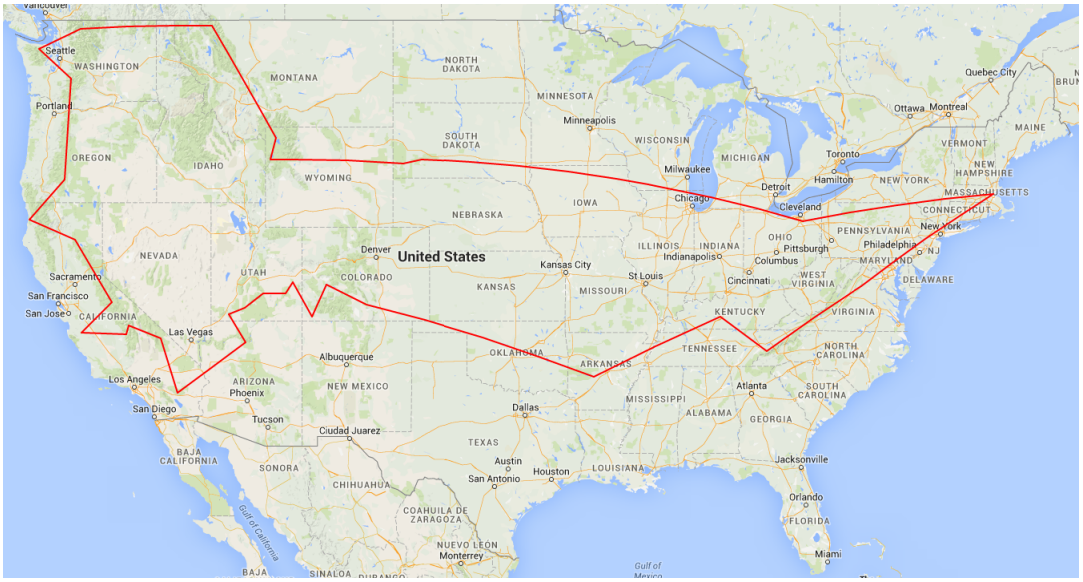

FIGURE 14. The optimal solution for maximum distance of $d = 12,000$ km.

FIGURE 15. The optimal solution for maximum distance of $d = 13,000$ km.

FIGURE 16. The optimal solution for maximum distance of $d = 14,000$ km.

FIGURE 17. The optimal solution for maximum distance of $d = 16,000$ km.

Appendix B. Julia/JuMP Code

B.1. **Julia Code for Linear Model: PCTSP.jl.** The pseudocode for the GPSToDist function was obtained from [1], based on the *haversine* formula. This was then written in Julia code.

```
using JuMP
using Gurobi
using Base.Test

#used to replace the norm function to get the distances between each park
function GPSToDist(lat1, long1, lat2, long2)
  R = 6371000 # metres
  a = sin((lat2-lat1)/2) * sin((lat2-lat1)/2) +  cos(lat1) * cos(lat2) *   sin(
  c = 2 * atan2(sqrt(a), sqrt(1-a))
  return R * c
end

# extractTour
# Given a n-by-n matrix representing the solution to an undirected TSP,
# extract the tour as a vector
# Input:
#  n        Number of cities
#  sol      n-by-n 0-1 symmetric matrix representing solution
# Output:
#  tour     n+1 length vector of tour, starting and ending at 1
function extractTour(n, sol)
    tour = [1]  # Start at city 1 always
    cur_city = 1
    while true
        # Look for first arc out of current city
        for j = 1:n
            if sol[cur_city,j] >= 1-1e-6
                # Found next city
                push!(tour, j)
                # Don't ever use this arc again
                sol[cur_city, j] = 0.0
                sol[j, cur_city] = 0.0
                # Move to next city
                cur_city = j
                break
            end
        end
```

```
        # If we have come back to 1, stop
        if cur_city == 1
            break
        end
    end  # end while
    return tour
end

# findSubtour
# Given a n-by-n matrix representing solution to the relaxed
# undirected TSP problem, find a set of nodes belonging to a subtour
# Input:
#  n        Number of cities
#  sol      n-by-n 0-1 symmetric matrix representing solution
# Outputs:
#  subtour  n length vector of booleans, true iff in a particular subtour
#  subtour_length   Number of cities in subtour (if n, no subtour found)
function findSubtour(n, sol)
    # Initialize to no subtour
    subtour = fill(false,n)
    # Always start looking at city 1
    cur_city = 1
    subtour[cur_city] = true
    subtour_length = 1
    while true
        # Find next node that we haven't yet visited
        found_city = false
        for j = 1:n
            if !subtour[j]
                if sol[cur_city, j] >= 1 - 1e-6
                    # Arc to unvisited city, follow it
                    cur_city = j
                    subtour[j] = true
                    found_city = true
                    subtour_length += 1
                    break  # Move on to next city
                end
            end
        end
        if !found_city
            # We are done
            break
```

```
        end
    end
    return subtour, subtour_length
end

# solveTSP
# Given a matrix of city locations, solve the TSP
# Inputs:
#   n       Number of cities
#   cities  n-by-2 matrix of (x,y) city locations
# Output:
#   path    Vector with order to cities are visited in
function solveTSP(n, cities, maxdis)

    # Calculate pairwise distance matrix
    dist = zeros(n, n)
    prizes = zeros(1,n)
    for i = 1:n
      prizes[i] = cities[i,3]
    end
    for i = 1:n
        for j = i:n
            d = GPSToDist(cities[i,1], cities[i,2], cities[j,1], cities[j,2])
            dist[i,j] = d
            dist[j,i] = d
        end
    end

     # Create a model that will use Gurobi to solve
    # We need to tell Gurobi we are using lazy constraints
    m = Model(solver=GurobiSolver())

    # x[i,j] is 1 iff we travel between i and j, 0 otherwise
    # Although we define all n^2 variables, we will only use
    # the upper triangle
    @defVar(m, x[1:n,1:n], Bin)
    @defVar(m, v[1:n], Bin)
    @defVar(m, n1>=0)
    @addConstraint(m, n1 == sum{v[i],i=1:n})


    # maximize the values of the prizes
```

```
@setObjective(m, Max, sum{prizes[i]*v[i], i=1:n})
@addConstraint(m,sum{dist[i,j]*x[i,j], i=1:n,j=i:n}<=maxdis)
# Make x_ij and x_ji be the same thing (undirectional)
# Don't allow self-arcs
for i = 1:n
    @addConstraint(m, x[i,i] == 0)
    @addConstraint(m, v[i] ==sum{x[i,j],j=1:n}/2)
    for j = (i+1):n
        @addConstraint(m, x[i,j] == x[j,i])
    end
end

# We must enter and leave every city we visit once and only once
for i = 1:n
    @addConstraint(m, sum{x[i,j], j=1:n} == 2*v[i])
end

function subtour(cb)
    # Optional: display tour starting at city 1
    println("----\nInside subtour callback")
    println("Current tour starting at city 1:")
    print(extractTour(n, getValue(x)))

    # Find any set of cities in a subtour
    subtour, subtour_length = findSubtour(n, getValue(x))

    if subtour_length == getValue(n1)
        # This "subtour" is actually all cities that are visited, so we are
        println("Solution visits all cities")
        println("----")
        return
    end

    # Subtour found - add lazy constraint
    # We will build it up piece-by-piece
    arcs_from_subtour = AffExpr()

    for i = 1:n
        if !subtour[i]
            # If this city isn't in subtour, skip it
            continue
        end
```

```
            # Want to include all arcs from this city, which is in
            # the subtour, to all cities not in the subtour
            for j = 1:n
                if i == j
                    # Self-arc
                    continue
                elseif subtour[j]
                    # Both ends in same subtour
                    continue
                else
                    # j isn't in subtour
                    arcs_from_subtour += x[i,j]
                end
            end
        end

        # Add the new subtour elimination constraint we built
        println("Adding subtour elimination cut")
        println("----")
        addLazyConstraint(cb, arcs_from_subtour >= 2)
    end  # End function subtour

    # Solve the problem with our cut generator
    setLazyCallback(m, subtour)
    solve(m)

    # Return best tour
    return extractTour(n, getValue(x))
end  # end solveTSP

cities = readcsv("C:\\Users\\username\\Google_Drive\\15.053_Project\\data.csv")
n = size(cities,1)
maxdis = 22000000
tour = solveTSP(n, cities, maxdis)

println("Solution: ")
println(tour)

print("[")
for i=tour
    print("(")
    print(cities[i,:1])
```

```
    print(",")
    print(cities[i,:2])
    print(")")
    print(",")
end
println("]")
println(length(tour))
```

The code here was modified from a TSP implementation from the jump examples found at https://github.com/JuliaOpt/JuMP.jl/blob/master/examples/tsp.jl.

Appendix C. US National Parks Data

Spreadsheet data follow on the next two pages.

MIT	Massachusetts	0.739297752	1.240812615	42.358641	71.093326

US National Parks

Name	State	Latitude (rad)	Longitude (rad)	Latitude (N, °)	Longitude (W, °)	Recreation Visitors (2013)	Area (ac)	Date est.
Acadia	Maine	0.774	1.190	44.35	68.21	2,254,922	47,389	2/26/1919
American Samoa	American Samoa	0.249	2.979	14.25	170.68	17,919	9,000	10/31/1988
Arches	Utah	0.675	1.912	38.68	109.57	1,082,866	76,518	4/12/1929
Badlands	South Dakota	0.764	1.789	43.75	102.50	892,372	242,755	11/10/1978
Big Bend	Texas	0.511	1.802	29.25	103.25	316,953	801,163	6/12/1944
Biscayne	Florida	0.448	1.398	25.65	80.08	486,848	172,924	6/28/1980
Black Canyon of the Gunnison	Colorado	0.673	1.880	38.57	107.72	175,852	32,950	10/21/1999
Bryce Canyon	Utah	0.656	1.958	37.57	112.18	1,311,875	35,835	2/25/1928
Canyonlands	Utah	0.667	1.919	38.20	109.93	462,242	337,597	9/12/1964
Capitol Reef	Utah	0.667	1.940	38.20	111.17	663,670	241,904	12/18/1971
Carlsbad Caverns	New Mexico	0.561	1.823	32.17	104.44	388,566	46,766	5/14/1930
Channel Islands	California	0.594	2.084	34.01	119.42	212,029	249,561	3/5/1980
Congaree	South Carolina	0.590	1.410	33.78	80.78	120,340	26,545	11/10/2003
Crater Lake	Oregon	0.749	2.131	42.94	122.10	523,027	183,224	5/22/1902
Cuyahoga Valley	Ohio	0.720	1.423	41.24	81.55	2,103,010	32,860	10/11/2000
Death Valley	California, Nevada	0.633	2.039	36.24	116.82	951,972	33,724	10/31/1994
Denali	Alaska	1.105	2.627	63.33	150.50	530,922	47,409	2/26/1917
Dry Tortugas	Florida	0.430	1.446	24.63	82.87	58,401	64,701	10/26/1992
Everglades	Florida	0.442	1.412	25.32	80.93	1,047,116	15,085	5/30/1934
Gates of the Arctic	Alaska	1.183	2.676	67.78	153.30	11,012	75,238	12/2/1980
Glacier	Montana	0.852	1.990	48.80	114.00	2,190,374	10,135	5/11/1910
Glacier Bay	Alaska	1.021	2.391	58.50	137.00	500,590	32,248	12/2/1980
Grand Canyon	Arizona	0.629	1.957	36.06	112.14	4,564,840	12,174	2/26/1919
Grand Teton	Wyoming	0.763	1.934	43.73	110.80	2,688,794	309,994	2/26/1929
Great Basin	Nevada	0.680	1.995	38.98	114.30	92,893	77,180	10/27/1986
Great Sand Dunes	Colorado	0.659	1.841	37.73	105.51	242,841	42,983	9/13/2004
Great Smoky Mountains	North Carolina, Tennessee	0.623	1.458	35.68	83.53	9,354,695	521,490	6/15/1934
Guadalupe Mountains	Texas	0.557	1.830	31.92	104.87	145,670	86,415	10/15/1966
Haleakala	Hawaii	0.362	2.726	20.72	156.17	785,300	29,093	8/1/1916
Hawaii Volcanoes	Hawaii	0.338	2.709	19.38	155.20	1,583,209	323,431	8/1/1916
Hot Springs	Arkansas	0.602	1.624	34.51	93.05	1,325,719	5,550	3/4/1921
Isle Royale	Michigan	0.840	1.545	48.10	88.55	16,274	571,790	4/3/1940
Joshua Tree	California	0.590	2.023	33.79	115.90	1,383,340	789,745	10/31/1994
Katmai	Alaska	1.021	2.705	58.50	155.00	28,966	36,745	12/2/1980
Kenai Fjords	Alaska	1.046	2.612	59.92	149.65	283,502	669,982	12/2/1980
Kings Canyon	California	0.642	2.069	36.80	118.55	567,544	461,901	3/4/1940
Kobuk Valley	Alaska	1.179	2.780	67.55	159.28	16,875	17,507	12/2/1980
Lake Clark	Alaska	1.064	2.678	60.97	153.42	13,000	26,197	12/2/1980

US National Parks

Name	State	Latitude (rad)	Longitude (rad)	Latitude (N, °)	Longitude (W, °)	Recreation Visitors (2013)	Area (ac)	Date est.
Lassen Volcanic	California	0.707	2.121	40.49	121.51	427,409	106,372	8/9/1916
Mammoth Cave	Kentucky	0.649	1.503	37.18	86.10	494,541	52,830	7/1/1941
Mesa Verde	Colorado	0.649	1.894	37.18	108.49	460,237	52,121	6/29/1906
Mount Rainier	Washington	0.818	2.125	46.85	121.75	1,148,552	235,625	
North Cascades	Washington	0.850	2.115	48.70	121.20	21,623	504,780	10/2/1968
Olympic	Washington	0.837	2.155	47.97	123.50	3,085,340	922,650	6/29/1938
Petrified Forest	Arizona	0.612	1.916	35.07	109.78	644,648	93,532	12/9/1962
Pinnacles	California	0.637	2.115	36.48	121.16	237,677	26,605	1/10/2013
Redwood	California	0.721	2.164	41.30	124.00	393,364	112,512	10/2/1968
Rocky Mountain	Colorado	0.705	1.843	40.40	105.58	2,991,141	265,828	1/26/1915
Saguaro	Arizona	0.563	1.929	32.25	110.50	678,261	91,439	10/14/1994
Sequoia	California	0.636	2.071	36.43	118.68	909,274	404,051	
Shenandoah	Virginia	0.672	1.367	38.53	78.35	1,136,505	199,045	5/22/1926
Theodore Roosevelt	North Dakota	0.820	1.806	46.97	103.45	545,090	70,446	11/10/1978
Virgin Islands	United States Virgin Islands	0.320	1.130	18.33	64.73	438,601	14,688	8/2/1956
Voyageurs	Minnesota	0.846	1.621	48.50	92.88	233,390	218,200	1/8/1971
Wind Cave	South Dakota	0.760	1.806	43.57	103.48	516,142	28,295	1/9/1903
Wrangell St. Elias	Alaska	1.065	2.478	61.00	142.00	69,984	83,231	12/2/1980
Yellowstone	Wyoming, Montana, Idaho	0.778	1.929	44.60	110.50	3,188,030	22,197	3/1/1872
Yosemite	California	0.660	2.086	37.83	119.50	3,691,191	761,266	10/1/1890
Zion	Utah	0.651	1.973	37.30	113.05	2,807,387	146,597	11/19/1919

Department of Mathematics, Sloan School of Management, Massachusetts Insitute of Technology, Cambridge, MA 02139, USA
E-mail address: choij@mit.edu

Department of Mathematics, Massachusetts Insitute of Technology, Cambridge, MA 02139, USA
E-mail address: vgnegron@mit.edu

Department of Electrical Engineering and Computer Science, Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
E-mail address: wheatman@mit.edu