The Computational Complexity of a Traveling Baseball Fan

Booth, A., Rouse, R., Totten, J.

Northwestern University, Predict 460

Dr. Carrie Dugan

June 2018

1 Abstract

In this paper, Gurobi Optimizer is utilized in Python to illustrate the computational complexity in the traveling salesman problem (TSP). We divide the TSP into two models of (1) 30 nodes and (2) 253 nodes. To define the nodes, we use GPS coordinates of the 30 active MLB ballparks and the 253 locations around the globe which have ever held a MLB game. While the program uses brute force calculations on both problems, an optimal solution is attainable for the 30-node model and we are left with an approximation for the 253-node model. The two models are proposed to demonstrate the rate at which complexity increases as the size of TSP increases. This experiment provides further evidence that TSP optimization is an NP-Hard problem and solutions cannot be proposed or verified in polynomial time.

Keywords: Traveling Salesman Problem, Time Complexity, P vs NP, Geodesics, Optimization

2 Introduction

Optimization is one of many applications in Operations Research (OR) and is used in a wide range of disciplines such as mathematics, computer science, business management, and biophysics. As the name suggests, the goal of this application is to determine an optimal solution to a given problem. In Decision Analytics, optimization serves as a stepping stone between predictive and prescriptive analytics, where predictive analytics is used to predict a range of outcomes and their likelihood and prescriptive analytics uses those predictions to recommend a decision.

The factorial complexities of TSP offer a beneficial and challenging optimization experiment. As factorial computations eventually grow faster than exponential computations (Figure 2.3), an increase in nodes quickly creates an insolvable task for most computing machines. The nature of this challenge is still debated among mathematicians in the conversations of whether or not N = NP. While modern computing enables unprecedented solutions to NP problems like TSP, a generally accepted proof to solve TSP in polynomial time has yet to be explained. Our experiment explores the challenges of TSP optimization applications and suggests places for further investigation. Our hypothesis states that while modern computational advancement improves TSP solvability, absolute optimization is still impossible in polynomial time.

2.1 TSP

In its most general sense, TSP seeks a route that takes a theoretical salesman through each of a required set of cities in the shortest possible distance. It lends itself to graphical representation, where the nodes or vertices of the graph represent the cities to be visited[1]. The problem can be stratified into symmetric and asymmetric subclasses. For example, if two cities, A and B, are connected and the distance traveled between them is equal despite the starting and end points, the problem is said to possess symmetry. However, if the direction of travel between cities A and B yields different distances, the problem is considered asymmetric. The symmetric problem is an undirected graph where the vertices (cities) are connected by a set of edges; the vertices of asymmetric TSP are connected by directed arcs. The problem is constrained by a path which must pass through every node only once and must return to the original starting point. This constraint implies any solution to a TSP cannot contain subtours. As illustrated in Figure 2.1, the concept of subtours is illustrated with a seven-city solution. Nodes 1, 2, and 3 belong to one graph, while nodes 4, 5, 6, and 7 belong to a second graph; there is no path connecting the first graph, and as a result this solution would be considered invalid.

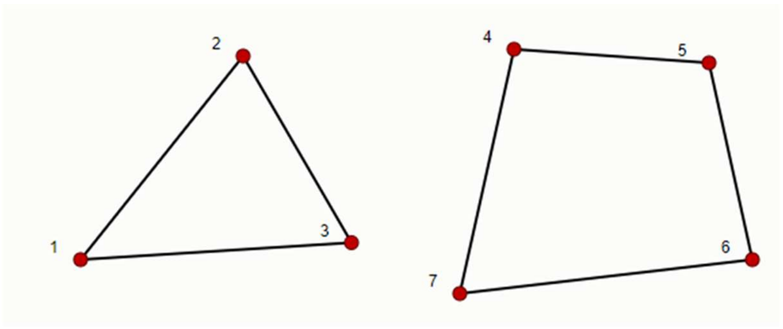


Figure 2.1 This graph illustrates an invalid TSP solution.

[1] Any meaningful cost metric can be substituted for distance.

2.2 P vs NP

In computational mathematics, P vs NP is an unsolved proof which describes the limits of solving and verifying practical and theoretical problems. Simply put, this concept distinguishes between the structure of deterministic and nondeterministic problems. Solving a problem in polynomial time means the problem can be reduced to a simple polynomial function of input and size. In this concept, P represents polynomial; a mathematical expression which contains a sum of many terms. P is a classification of problems which are deterministic, meaning they are easily solvable and their solutions are easily verifiable. Examples of P problems are sorting, searching, and basic math operations (e.g., subtraction, addition, division, multiplication, etc.). NP represents nondeterministic polynomial. This classification characterizes problems which are not easily solvable, but given a solution can be verified with deterministic computations in polynomial time. Within NP lies two subsets, NP-Complete and NP-Hard. NP-Hard represents problems whose solutions cannot be easily verified. A problem is considered NP-Complete only if the problem is both NP-Hard (cannot be solved in polynomial time) and NP (can be verified in polynomial time (Hemaspaandra, L. A., 2012).

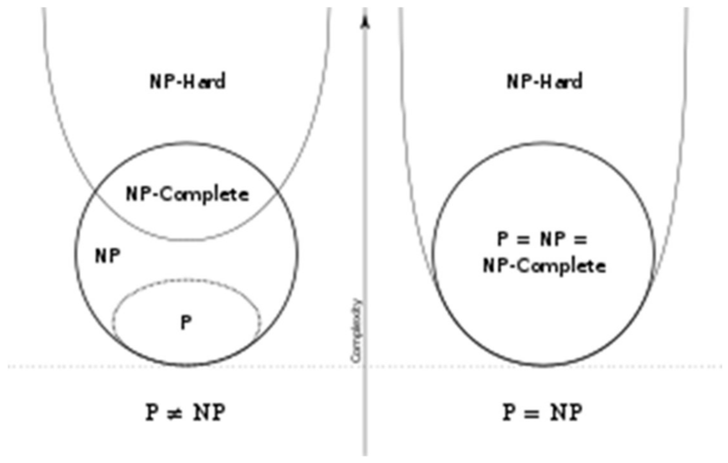


Figure 2.2 This graph illustrates the implications of classifications if the P vs. NP proof is solved.

Two versions of TSP exemplify the classification bounds as they are known today. If a proposed TSP solution stated that a salesman could travel to *n* cities in *x* miles, one could compute the solution in polynomial time and determine if this solution is feasible. In this example the TSP is expressed as a *decision* problem and is considered NP-Complete. The objective of solving this problem is not optimization; it is verifying a proposed distance solution.

When optimization is introduced to the TSP, the problem becomes NP-Hard. In this variation of the problem, the objective is to find the optimal path between *n* cities. Given the factorial nature of the problem, there is no known solution to reduce this problem to a polynomial. Even if an optimal solution is found in a problem where *n* is small, the calculation is still computed using brute force and is therefore NP-Hard.

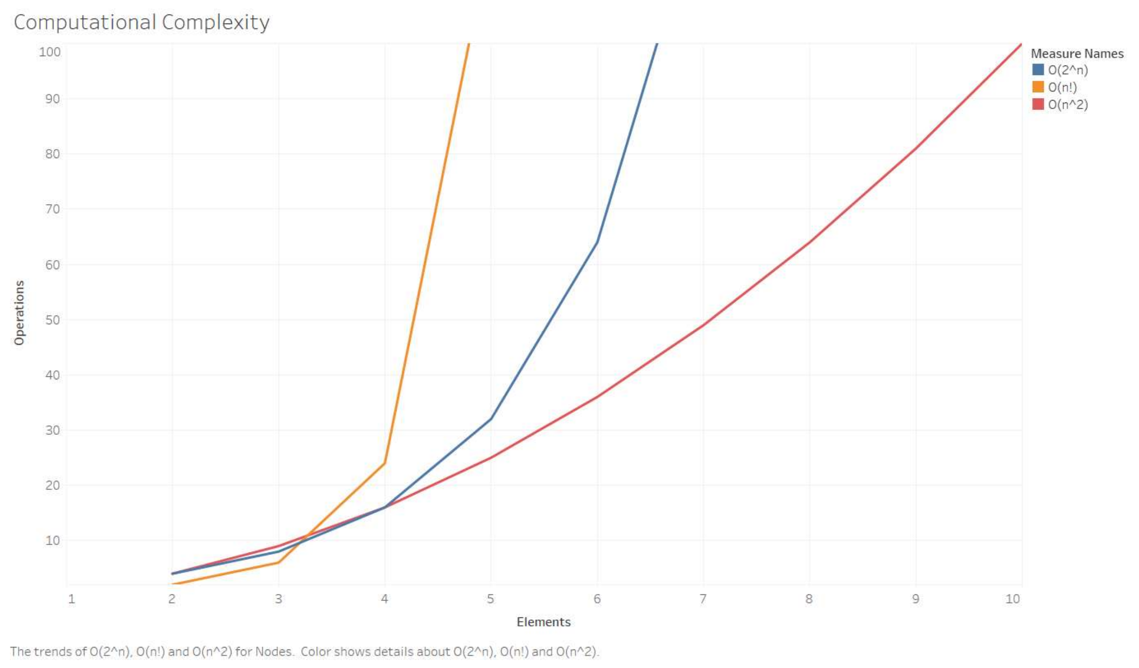


Figure 2.3 Computational Complexity. This graph illustrates the increased complexity as polynomial, exponential, and factorial equations are computed.

That NP solution validation can occur with P efficiency has stoked the debate as to whether these complexity classes will continue to exist separately. While no one has proven P = NP, no one has proven that P ≠ NP, either. If P = NP, then an algorithm to solve any NP-Complete problem would solve any NP problem in polynomial time (Hemaspaandra, L. A., 2012). As illustrated in Figure 2.2, should P = NP, there would no longer be a distinction in computational complexity between P, NP, and NP-Complete problems. This would have serious implications on many fields. The fundamentals of modern cryptography would change, as prime factorization of large numbers would be solvable in polynomial time. In molecular dynamics, improvements in protein folding simulations (simulated annealing) would help doctors find cures to rare diseases (Kholmirzo, K., 2017). However, if P ≠ NP, it means there are problems that will forever be unsolvable in polynomial time.

2.3 TSP Applications

Where theory fails to solve TSP optimization, advancements in applied mathematics have taken shape with the use of approximation methods called “Improvement Heuristics.” A common initial solution is the “Nearest Neighbor” solution. As this algorithm does not reconsider previous decisions, it continues to add the next least-cost edge. Common edge exchanges are “two-optimal” and “three-optimal” where two and three edges, respectively, are swapped in successive routes. The success of this approximation largely depends on the starting point because depending on the starting node’s relation to the rest of the nodes, the solution is susceptible to becoming trapped by local optima. The paths of the initial solution or subsequently “improved” solutions may prevent convergence toward the global optimum (Curtin, Voicu, Rice, & Stefanidis, 2013).

Heuristics are commonly used in Geographic Information Systems (GIS), which conveniently offer users the functionality to plan “efficient” routes. Tabu search is among the heuristics used in GIS applications. As the name implies, Tabu search maintains a list of forbidden moves (or edge/arc swaps) that generally ensure the rules of the TSP are not violated. However, Curtin et al (2013) claims Tabu algorithms can prohibit attractive moves even when the dangers of cycling are omitted. To combat this prohibition, “aspiration” criteria should be applied which allow the algorithm search to override the Tabu list. Given competition among GIS providers, the exact nature of their algorithms is rarely disclosed, however Curtin et al (2013) suggests that many of them are based on Tabu search and fail to determine an optimal solution with more than 10 nodes.

Among the most recognized advancements in applied TSP optimization is the Concorde algorithm. Founded in the early 2000’s this algorithm uses a branch-and-cut-and-price technique, which initially deploys variable and constraint relaxation to reduce the computation and memory requirements. This method assumes in large problems a subset of variables will be non-basic to the optimal solution. The problem is branched into two sub problems and iteratively leverages the optimally relevant variables. The largest TSP optimization problem solved by the Concorde algorithm contains as many as 85,900 nodes (Curtin, 2013).

The importance of solving TSP optimization extends well beyond a cost-efficient salesman. As Cattaruzza et al (2015) claims, in 2009 60% of global oil consumption and 25% of global energy consumption is caused by transportation alone. These economic and environmental impacts have encouraged many transportation-dependent organizations to improve TSP applications like Vehicle Routing Programs (VRPs). Additional complexities in VRP include asymmetry, multiple vehicles, delivery time limits, and access time windows. Asymmetry is almost unavoidable in VRPs and can be imposed by economic factors like the distance traveled by an empty truck and environmental factors like atmospheric trade winds or currents in oceanic shipping lanes.

Route optimization enables industrial cost savings and meaningful environmental improvements. In the early 2000’s, Waste Management, Inc. (WM) partnered with an outside technology provider as it sought mathematical optimization of its disposal route designs; WM’s route designs had previously relied on the tribal knowledge of planners, not mathematical optimization. As WM sought to minimize vehicles and travel time; balance vehicle workload; and maximize “visual attractiveness” of the route, they recognized savings of $18 million in the first year of optimization deployment (Sahoo, S., 2003); they projected savings as high as $44 million in the second year of deployment**.** These improvements to TSP applications are extremely beneficial to industry and the environment, but it is important to remember these benefits only provide marginal advancement to solving TSP optimization in polynomial time. The bounds to solving a TSP optimization are still rooted in our current understanding of problem solving itself.

3 Methodology

3.1 Objective Function, Decision Variables, Constraints

To demonstrate that TSP optimization is an NP-Hard problem, we examined one of ours, and America’s, favorite past-times, baseball. We imagined an arbitrary baseball fan and modeled them in two scenarios. The first scenario allowed the fan to visit every active ballpark in the country. In a total of 30 active ballparks, the objective was to travel the minimal distance and visit each park only once. The second scenario allowed the fan to visit every location to ever host a professional baseball game, regardless of whether the stadium still existed. Over several countries and 253 locations, again the objective was to travel the minimal distance and visit each location only once.

Because we examined these scenarios as a symmetric TSP problem, the proper formulation approach required a binary integer program. To define the objective function, we let A be the set of arcs connecting a set of nodes N.

- Let $A(i)$ be the arcs incident ("connected") to node i
- Let c_e be the distance of arc e
- Let $x_e = \begin{cases} 1 & \text{if arc } e \text{ is in the tour;} \\ 0 & \text{otherwise.} \end{cases}$

$$\begin{aligned} \min \quad & \sum_e c_e x_e \\ \text{subject to} \quad & \sum_{e \in A(i)} x_e = 2 \quad \forall \quad i \in N. \\ & x_e \in \{0,1\} \quad \forall \quad e \in A(i). \end{aligned}$$

This translates to an objective function of minimizing total distance, with the constraint that each node has exactly two arcs, or paths, one coming in and one going out.

To eliminate subtours, however, we needed to add one final constraint, where S is a subset of nodes.

$$\sum_{e:i,j \in S} x_e \leq |S| - 1$$

3.2 Measuring Distance

In both scenarios discussed, the optimization of traveling to each baseball location was modeled as a symmetric TSP. As the objective function states to minimize the distance traveled, we took particular interest in using the right calculation for distance. In a general sense, distance is defined as a measure of the space between two points in that space. The most familiar methods in calculating Euclidean distance are either "straight-line" distance, or the more practical measurement, Manhattan Distance. Figure 3.1 shows the distinction between these two definitions of distance. "Straight-Line" distance ignores geographic obstacles, while Manhattan's Distance is constrained to obstacles like roads and buildings. Both measurement alternatives work well in two-dimensional space, but during our research we discovered these methods do not function well with calculations spanning across hemispheres, various elevations, or even between cities. As the earth is not flat and is rather an oblate ellipsoid, it is better described as a slightly flattened sphere due to Earth's rotation (Karney, 2011). Because of this, using any of the two-dimensional distance calculations would suggest a path through the Earth.

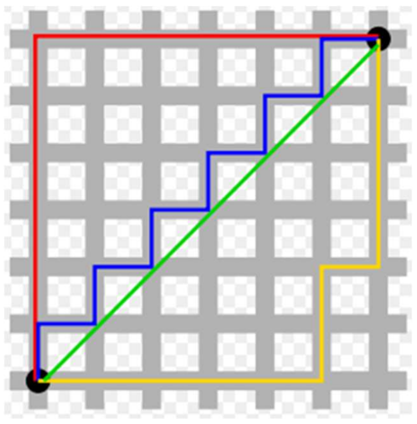


Figure 3.1 This graph illustrates the difference between "straight-line" and Manhattan's distance.

To account for spherical measurements we used Orthodromic distance, or great-circle distance. Orthodromic distance is the shortest distance between two points on the surface of a sphere as measured along the surface of a sphere. In spaces with curvature, including spheroids and ellipsoids, straight lines are replaced by geodesics in distance calculations (Karney, 2011). Geodesics are circles on a sphere whose centers coincide with the center of the sphere, and are consequently called great circles. The arc length on the great circle between the two points represents the smallest distance.

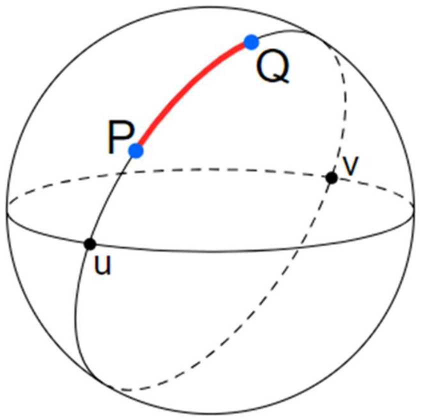


Figure 3.2 This illustration shows the concept of calculating two points along the surface of a sphere.

Calculating orthodromic distance works best for large distances calculations on Earth. To show this calculation, let (x1, y1) and (x2, y2) be the geographical latitude and longitude in radians of two points 1 and 2; and let $\triangle$X and $\triangle$Y be their absolute differences. The central angle between them is given by the spherical law of cosines. The distance (arc length) is defined as the radius of that sphere multiplied by the central angle.

$$\Delta\sigma = \arccos\left(\sin\phi_1 \cdot \sin\phi_2 + \cos\phi_1 \cdot \cos\phi_2 \cdot \cos(\Delta\lambda)\right). \rightarrow \quad d = r\,\Delta\sigma.$$

Orthodromic measurement presents fine spherical calculation; however, as the Earth resembles an oblate ellipsoid, pure orthodromic calculations produce an error when calculating distance. Calculating distance between two or more locations on Earth is a much-researched discipline within triangulation and GIS. The rotation of the Earth complicates calculations because even with the expectation of traveling in a straight line, one would move with the rotation of the Earth (Karney, 2013).

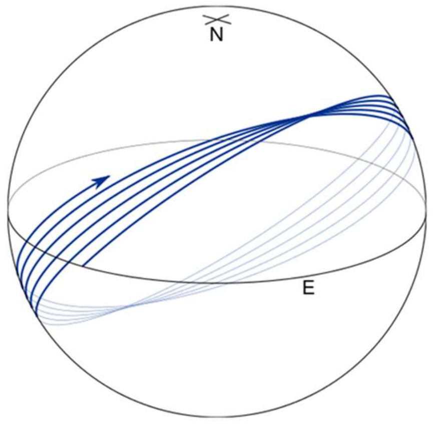


Figure 3.3 This illustration shows the straight-line error produced by Earth's natrual rotation.

While the math needed to calculate geodesic distance delves into the world of differential geometry, a popular Python package, Geopy, accounts for these considerations and accurately calculates global distances. From their documentation: “Geopy can calculate geodesic distance between two points using the geodesic distance or the great-circle distance, with a default of the geodesic distance available as the function.” Within Geopy we used its geodesic distance, calculated by an algorithm developed in 2013 by Karney. This measurement implies we assumed the distance calculation between each ballpark would not be constrained for variations like roads and would more resemble the expression “as the crow flies.”

Due to its speed and functionality, we used Gurobi Python as the language to run our model. While the model converged quickly for the 30-node (ballpark) instance, the 253-node instance caused the model to run until computing power was exhausted and our computer crashed. Using the Gurobi algorithm, our program would require days to find an optimal solution. Instead, we defined a time and error limit so the model would converge within a specific percentage of the optimal solution. The range of time limits and the resulting optimization gap is shown in Figure 3.4.

Time (minutes)	Time (seconds)	Optimization Gap (%)
0.5	30	89.4
1	60	87.4
3.5	310	26.6
8	488	13.1
10	600	11.6
16	950	10.9
30	1800	3.1
60	3600	1.2

Figure 3.4 This table shows the Optimization Gap achieved at different time limits set in Gurobi.

4 Computational Experiment and Results

Our first iteration of the model considered the 30 active ballparks. The optimal path is shown in Figure 4.1 and can start from any position in the order. The optimal geodesic distance traveled was 8,962.77 miles and it seems to follow a path that follows nation’s border. The algorithm did a respectable job configuring the optimal path to include ballparks in states inside this path like Missouri, Minnesota, and Colorado. Similarly, the algorithm accurately mapped clusters in regions like the Northeast where ballparks in New York, Boston, Philadelphia, and D.C., were in close proximity.

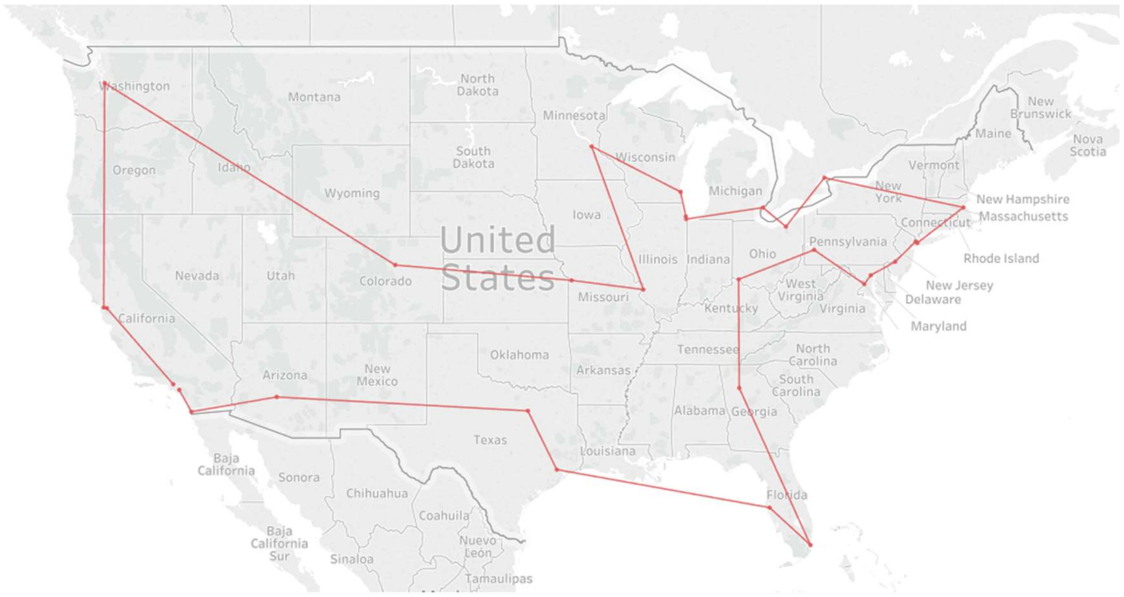


Figure 4.1 Shows the optimal path needed to travel to all 30 active ballparks

Next, we ran the algorithm against all 253 historical ballpark locations around the globe. The optimal path is shown in Figure 4.2 and can also start from any position in the order. The optimal geodesic distance traveled is 29,298.9 miles. The history of the sport is apparent here, as 32% of the locations were found in the Northeast United States. This concentration of nodes likely demanded the most work for the program and warrants the best place to focus future work. The algorithm seemed to have found the optimal solution regarding international travel, the longest legs of the path. These routes include flying from Hawaii, to Australia, to Japan, while returning to Seattle. Further, the optimal route accommodates Puerto Rico among a cluster of Florida ballparks and includes Mexico between ballparks in Texas and Arizona.

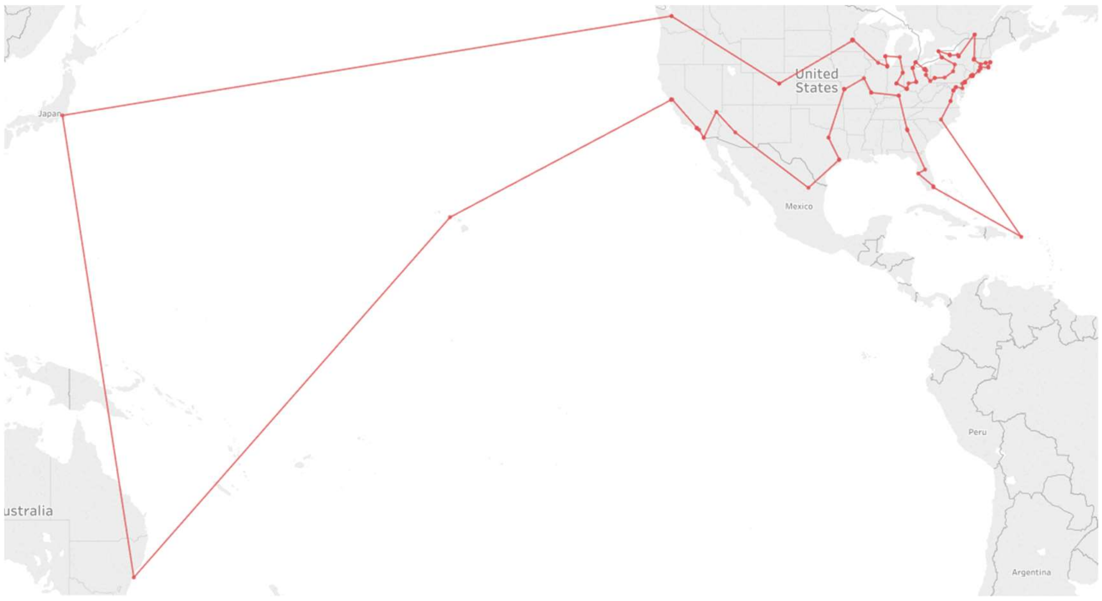


Figure 4.2 Shows the optimal 253-node path approximated by our program.

5 Future Work

When analyzing the results, we noticed a couple areas of improvement. When looking at the 253-node path (Figure 5.1), several obvious clusters and a couple questionable legs appear. In comparison to the 30-node solution path (Figure 4.1), locations in Florida were adjacent to Texas; however, in the 253-node solution, Florida locations are adjacent to Georgia and Kentucky. This very well could be the optimal solution, but with an optimization gap of 1.2%, we cannot say for certain a more optimal path does not exist. The algorithm dedicated a lot of effort testing each ballpark in clusters with every other node in the problem, but in terms of the optimal path these ballparks within the clusters should always connect to each other in some order. We suggest a strategy that finds an optimal solution between these clusters. Determining the optimal path between the clusters could give us a direction the within-clusters path should follow. Once finding the within-cluster optimal solution, we could insert this path into the global solution. This strategy would help explore further research in our program and TSP, as machine learning techniques like k-clusters could be deployed to define nodes which should never be compared to other nodes.

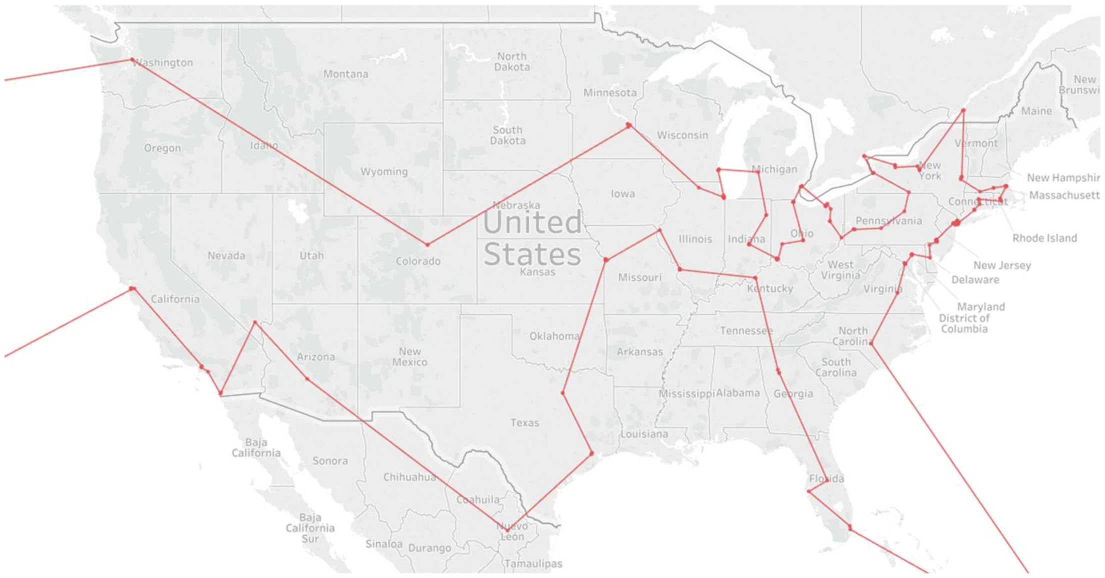


Figure 5.1 Shows the North American-focused 253-node solution

6 Discussion and Conclusions

The purpose of this project was to explore TSP and the computational complexity surrounding it. The exercise analyzed two sizes of a TSP, (1) a 30-ballpark problem and (2) a 253-ballpark problem. As mentioned, the program converged on a solution in less than a second for the 30-ballpark problem. However, the larger problem failed to converge in as long as 12 hours. As a result, candidate solutions for the larger problem were obtained through solver runs of varying times; candidate solutions saw their errors decrease in proportion with the solver time that had been allotted.

As discussed, TSP optimization is an NP-Hard problem; regardless of the size of the problem. As it stands, solutions cannot be presented in polynomial time nor can solutions be verified in polynomial time. The factorial nature of the TSP creates computational complexity which limits modern computing like our program (Figure 6.1). Our problem was relatively simplistic in its setup. It was considered symmetric; it did not investigate transitions between various modes of travel; it did not reconcile the feasibility of applying Manhattan distance measures in urban areas. Despite the problem was stripped to the simplest TSP interpretation, the larger problem could not converge to an optimal solution.

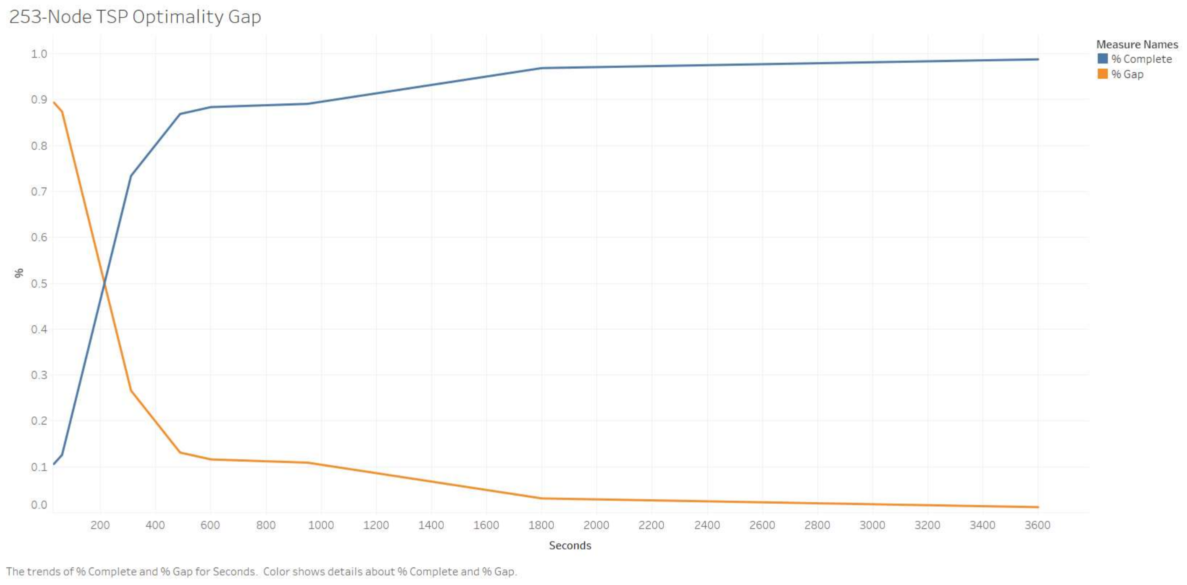


Figure 6.1 illustrates the optimality gap achieved for the 253-node solution with respect to the defined time limit constraint.

Many practical TSP optimization problems contain hundreds or thousands of nodes. Furthermore, most problems cannot ignore asymmetry in their formulation. Although algorithms exist to find exact solutions for extremely large symmetric TSPs (like Concorde), such a silver bullet is not found for asymmetric TSP. The use of heuristics to obtain candidate solutions for mid to large-scale asymmetric TSP means the solutions will always contain some deviation from optimal. However, heuristics provide an attainable, reasonable approximation in timescales of minutes where exact solutions require days. The ability of heuristics to quickly obtain reasonable approximation allows systems to integrate with real-time data streams such as GPS locators and continuously "course-correct" recent solutions. This feature seems to mitigate the presence of errors in sub-optimal solutions. While theoretical proofs in computational complexity remain at large, organizations can improve the practical methods used to bring the benefits of TSP solvability to the public.

7 Works Cited

Cattaruzza, D., Absi, N., Feillet, D., & González-Feliu, J. (2015, January 10). *Vehicle routing problems for city logistics*. Retrieved May 29, 2018, from https://www.springer.com/us. doi:https://doi-org.turing.library.northwestern.edu/10.1007/s13676-014-0074-0

Curtin, K. M., Voicu, G., Rice, M. T., & Stefanidis, A. (2014, March 14). *A Comparative Analysis of Traveling Salesman Solutions from Geographic Information Systems*. Retrieved May 28, 2018, from https://onlinelibrary-wiley-com.turing.library.northwestern.edu/. doi:10.1111/tgis.12045

Gatrell, A. C. *Distance*. In International Encyclopedia of Geography: People, the Earth, Environment and Technology (eds D. Richardson, N. Castree, M. F. Goodchild, A. Kobayashi, W. Liu and R. A. Marston). March 06, 2017. doi:10.1002/9781118786352.wbieg0391, Wiley Online Library

Geopy's Documentation. Retrieved May 28, 2018, from https://geopy.readthedocs.io/en/stable/

Great-circle distance. Retrieved May 28, 2018, from https://en.wikipedia.org/wiki/Great-circle_distance

Hemaspaandra, L. A. *SIGACT News Complexity Theory Column 74*. Retrieved May 29, 2018, from https://www.acm.org/. doi:10.1145/2261417.2261433

Johnson, K. *Seamheads Ballpark Database.* Retrieved May 28, 2018, from https://www.seamheads.com/ballparks/index.php

Karney, C.F.F. *Algorithms for Geodesics*, J. Geodesy 87(1), 43–55. Jan. 2013. https://doi.org/10.1007/s00190-012-0578-z

Karney, C.F.F. *Geodesics on an Ellipsoid of Revolution*, Feb. 2011, arXiv:1102.1215

Kholmurodov, K., Dushanov, E., Khusenov, M., Rahmonov, K., Zelenyak, T., Doroshkevich, A., & Majumder, S. (2017). Molecular dynamics studies on the interaction and encapsulation processes of the nucleotide and peptide chains inside of a carbon nanotube matrix with inclusion of gold nanoparticles. *Journal of Physics: Conference Series, 848*, 012012. doi:10.1088/1742-6596/848/1/012012

Kotov, V. & Kovalev, M. Mathematical Methods of Operations Research 43(2): 169-181. 1996. https://doi-org.turing.library.northwestern.edu/10.1007/BF01680370

Laporte, G. (2010). *A concise guide to the traveling salesman problem*. The Journal of the Operational Research Society, 61(1), 35-40. doi:http://dx.doi.org.turing.library.northwestern.edu/10.1057/jors.2009.76

Lihoreau, M., Chittka, L. and Raine, N. E. *Trade-off between travel distance and prioritization of high-reward sites in traplining bumblebees*. Functional Ecology, 25: 1284-1292. June 28, 2011. doi:10.1111/j.1365-2435.2011.01881.x. Wiley Online Library

Ministry of Defence. *Admiralty Manual of Navigation*, Volume 1, The Stationery Office, 1987, p. 10, ISBN 9780117728806

P versus NP problem. Retrieved May 28, 2018, from https://en.wikipedia.org/wiki/P_versus_NP_problem

Sahoo, S., Kim, S., Kim, B., Kraas, B., and Popov, A., Jr. (2005, February 1). *Routing Optimization for Waste Management*. Retrieved May 29, 2018, from https://www.informs.org/. doi:10.1287/inte.1040.0109

The Traveling Salesman Problem with integer programming and Gurobi. Retrieved May 28, 2018, from http://examples.gurobi.com/traveling-salesman-problem

Time Complexity. Retrieved May 28, 2018, from https://en.wikipedia.org/wiki/Time_complexity

8 Appendix

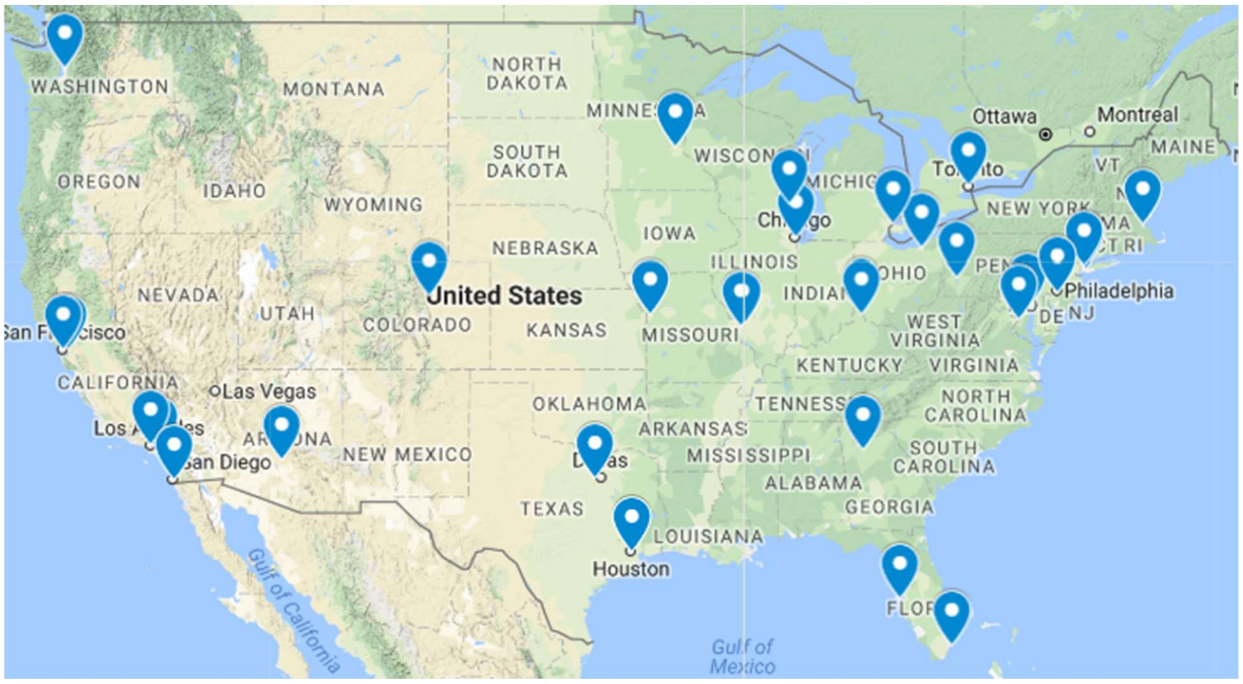


Figure 7.1 30 Active Ballpark Locations

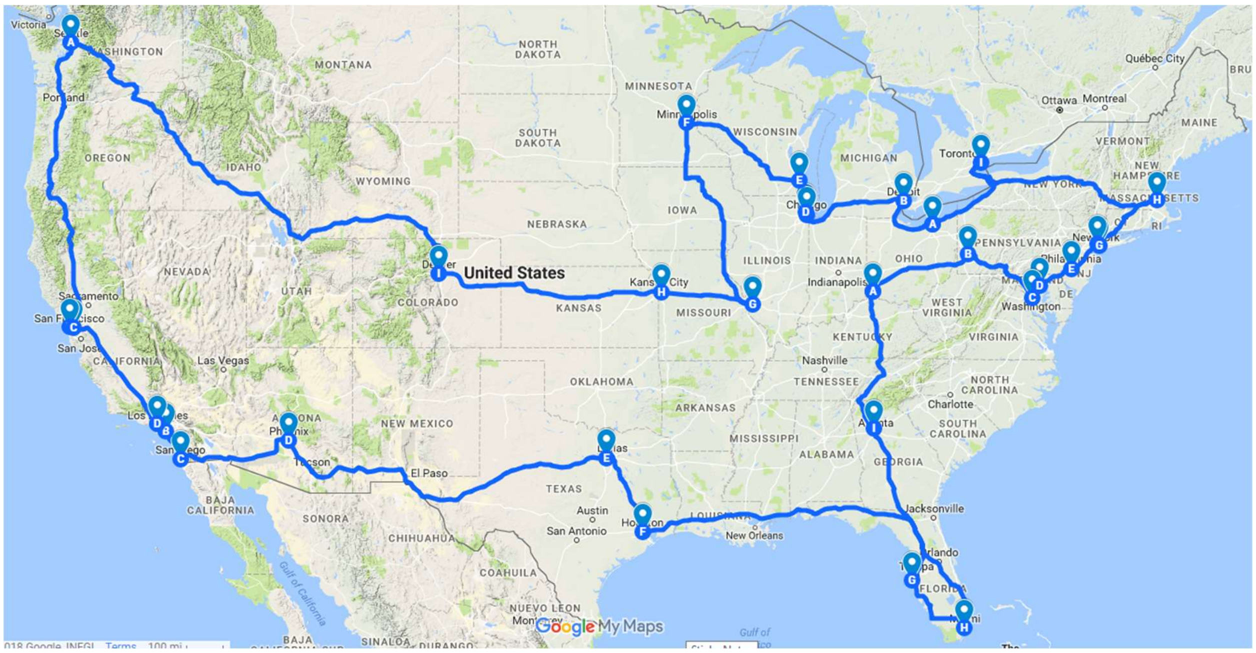


Figure 7.2 Calculated Optimal Path in Google Maps

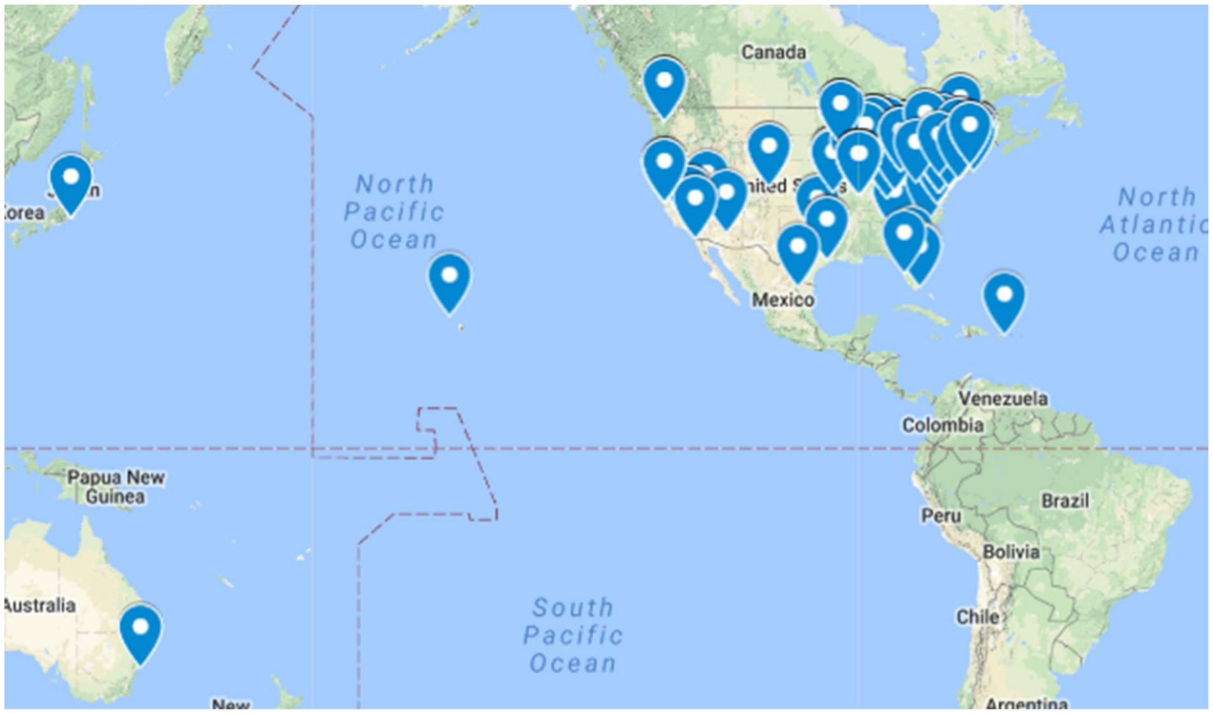


Figure 7.3 All Historic Baseball Locations

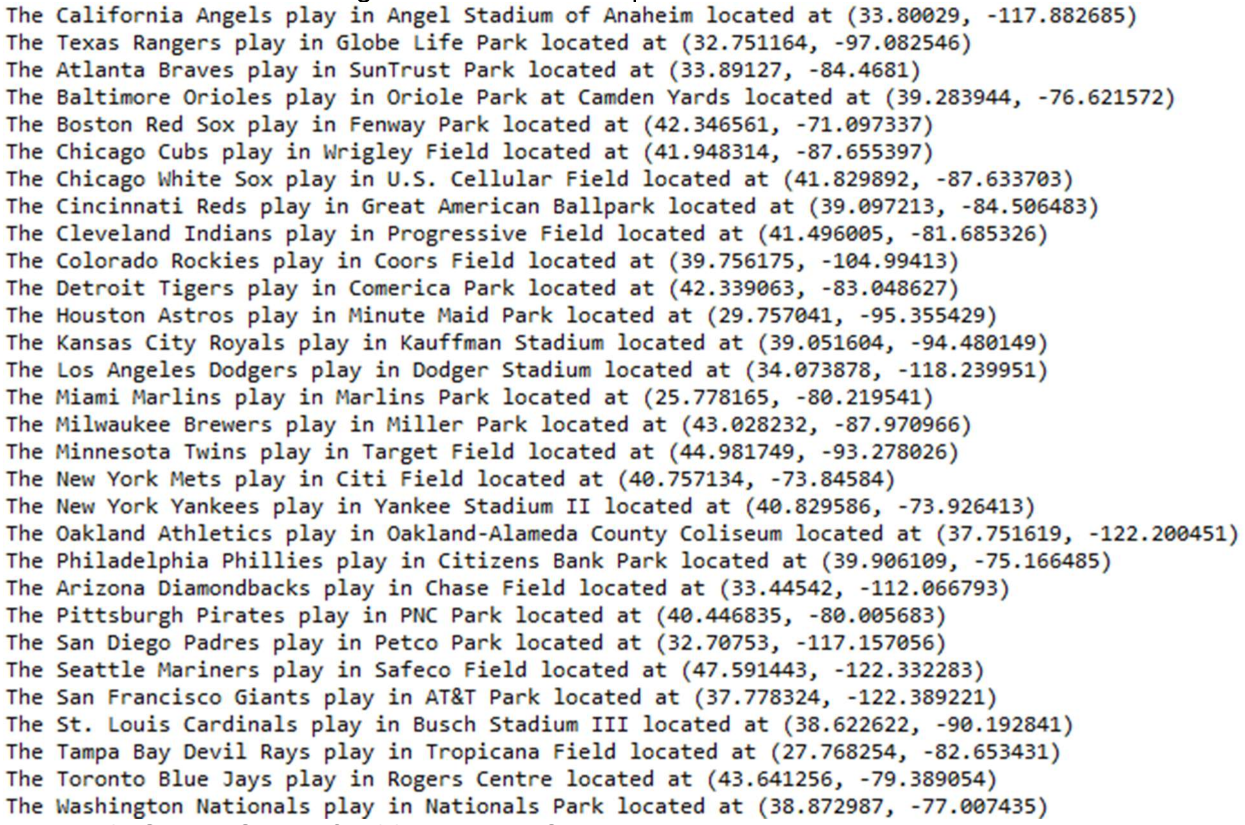
Python Code:

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 15 13:36:32 2018

@authors: Alexander Booth, Richard Rouse, Jordan Totten
"""


#Imports
import pandas as pd
from collections import Counter
import geopy.distance
import itertools
import gurobipy as g

#Load Data
#https://www.seamheads.com/ballparks/index.php
parks =
pd.read_csv(r"C:\Users\Alexander\Documents\baseball\TSP\Seamheads_Ballparks_Database_2017\Park
s.csv")
teams =
pd.read_csv(r"C:\Users\Alexander\Documents\baseball\TSP\Seamheads_Ballparks_Database_2017\Team
s.csv")
park_teams =
pd.read_csv(r"C:\Users\Alexander\Documents\baseball\TSP\Seamheads_Ballparks_Database_2017\Home
Main Data With Parks Breakout.csv")

#Get info from csvs
teamID = []
teamName = []
seasonsPlayed = []

for i in range(len(parks)):
    parkID = parks.iloc[i]["PARKID"]
    parksForTeam = park_teams[park_teams.Park_ID == parkID].TeamID
    teamID.append(Counter(parksForTeam).most_common(1)[0][0]) #Get team that played at home in
the stadium the most
    teamName.append(teams[teams.TeamID == teamID[i]].City.values[0] + " " + teams[teams.TeamID
== teamID[i]].Nickname.values[0])
    seasonsPlayed.append(sum(Counter(parksForTeam).values()))

#Combine into one dataframe
df = parks.copy(deep=True)
df['TeamID'] = teamID
df['TeamName'] = teamName
df['SeasonsPlayed'] = seasonsPlayed

df.START = pd.to_datetime(parks.START, format='%Y%m%d', errors='ignore').dt.date
df.END = pd.to_datetime(parks.END, format='%Y%m%d', errors='ignore').dt.date
df['Active'] = df.END.isnull() & df.START.notnull()

#Sanity Check
#Apparently the California Angels has played more seasons than LAA at Angel Stadium
#Same for the Devil Rays vs Rays at Tropicana
df_active = df[df.Active == True]
for i in range(len(df_active)):
    teamName = df_active.iloc[i].TeamName
    parkName = df_active.iloc[i].NAME
    lat = df_active.iloc[i].Latitude
    long = df_active.iloc[i].Longitude
    print("The " + teamName + " play in " + parkName + " located at (" + str(lat) + ", " +
str(long) + ")")

#Test distance
#http://www.meridianoutpost.com/resources/etools/calculators/calculator-latitude-longitude-
distance.php

coords_1 = (38.872987, -77.007435)
coords_2 = (43.641256, -79.389054)

print(geopy.distance.geodesic(coords_1, coords_2).mi) #351.61 miles

#####################################################################################

# Copyright 2018, Gurobi Optimization, LLC
# Modified by Alexander Booth, 2018
# Thanks Gurobi!

# Solve a traveling salesman problem on a randomly generated set of
# points using lazy constraints.   The base MIP model only includes
# 'degree-2' constraints, requiring each node to have exactly
# two incident edges.  Solutions to this model may contain subtours -
# tours that don't visit every city.  The lazy constraint callback
# adds new constraints to cut them off.

# Callback - use lazy constraints to eliminate sub-tours

def subtourelim(model, where):
    if where == g.GRB.Callback.MIPSOL:
        # make a list of edges selected in the solution
        vals = model.cbGetSolution(model._vars)
        selected = g.tuplelist((i,j) for i,j in model._vars.keys() if vals[i,j] > 0.5)
        # find the shortest cycle in the selected edge list
        tour = subtour(selected)
        if len(tour) < n:
            # add subtour elimination constraint for every pair of cities in tour
            model.cbLazy(g.quicksum(model._vars[i,j]
                                    for i,j in itertools.combinations(tour, 2))
                         <= len(tour)-1)

# Given a tuplelist of edges, find the shortest subtour

def subtour(edges):
    unvisited = list(range(n))
    cycle = range(n+1) # initial length has 1 more city
    while unvisited: # true if list is non-empty
        thiscycle = []
        neighbors = unvisited
        while neighbors:
            current = neighbors[0]
            thiscycle.append(current)
            unvisited.remove(current)
            neighbors = [j for i,j in edges.select(current,'*') if j in unvisited]
        if len(cycle) > len(thiscycle):
            cycle = thiscycle
    return cycle
```

```
####################################################################################

#All Current

n = len(df_active)
points = [(df_active.iloc[i].Latitude, df_active.iloc[i].Longitude) for i in range(n)]

dist = {(i,j) :
    geopy.distance.geodesic((points[i][0],points[i][1]), (points[j][0],points[j][1])).mi
    for i in range(n) for j in range(i)}

m = g.Model()

# Create variables
vars = m.addVars(dist.keys(), obj=dist, vtype=g.GRB.BINARY, name='e')
for i,j in vars.keys():
    vars[j,i] = vars[i,j] # edge in opposite direction

# Add degree-2 constraint
m.addConstrs(vars.sum(i,'*') == 2 for i in range(n))
m.update()

# Optimize model
m._vars = vars
m.Params.lazyConstraints = 1
m.optimize(subtourelim)

vals = m.getAttr('x', vars)
selected = g.tuplelist((i,j) for i,j in vals.keys() if vals[i,j] > 0.5)

tour = subtour(selected)
assert len(tour) == n

print('')
print('Optimal tour: %s' % str(tour))
print('Optimal cost: %g' % m.objVal)
print('')

for cityIndx in tour:
    print(df_active.iloc[cityIndx].NAME + ", " + df_active.iloc[cityIndx].CITY)
##################################################################################
#All Historical

n = len(df)
points = [(df.iloc[i].Latitude, df.iloc[i].Longitude) for i in range(n)]

dist = {(i,j) :
    geopy.distance.geodesic((points[i][0],points[i][1]), (points[j][0],points[j][1])).mi
    for i in range(n) for j in range(i)}

m = g.Model()
m.setParam('TimeLimit', 10*60) #set time limit

# Create variables
vars = m.addVars(dist.keys(), obj=dist, vtype=g.GRB.BINARY, name='e')
for i,j in vars.keys():
    vars[j,i] = vars[i,j] # edge in opposite direction

# Add degree-2 constraint
m.addConstrs(vars.sum(i,'*') == 2 for i in range(n))
m.update()

# Optimize model
m._vars = vars
m.Params.lazyConstraints = 1
m.optimize(subtourelim)

vals = m.getAttr('x', vars)
selected = g.tuplelist((i,j) for i,j in vals.keys() if vals[i,j] > 0.5)

tour = subtour(selected)
assert len(tour) == n

print('')
print('Optimal tour: %s' % str(tour))
print('Optimal cost: %g' % m.objVal)
print('')

for cityIndx in tour:
    print(df.iloc[cityIndx].NAME + ", " + df.iloc[cityIndx].CITY)
```

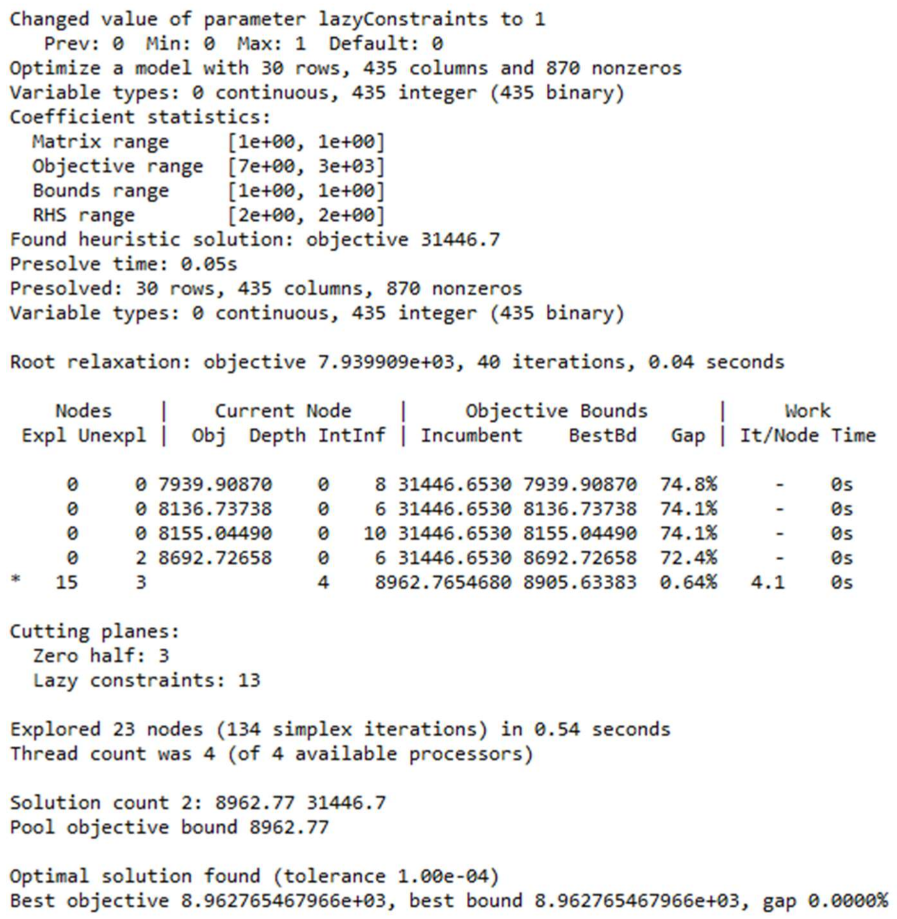
Tables and Figures:

Table A1: Latitude and Longitude of Current Ballparks



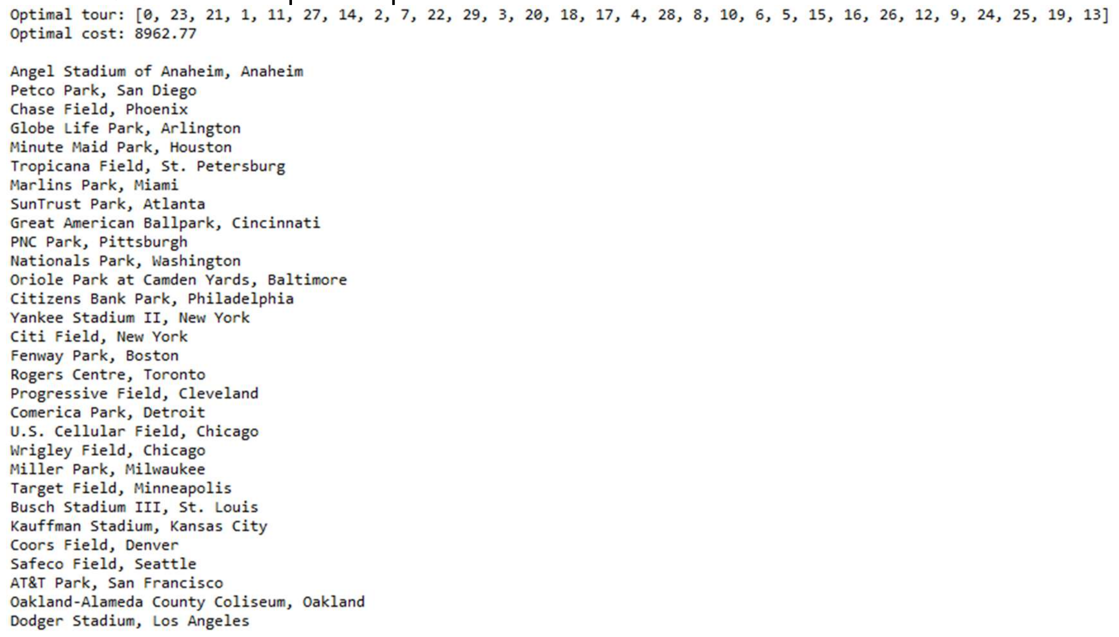
```
The California Angels play in Angel Stadium of Anaheim located at (33.80029, -117.882685)
The Texas Rangers play in Globe Life Park located at (32.751164, -97.082546)
The Atlanta Braves play in SunTrust Park located at (33.89127, -84.4681)
The Baltimore Orioles play in Oriole Park at Camden Yards located at (39.283944, -76.621572)
The Boston Red Sox play in Fenway Park located at (42.346561, -71.097337)
The Chicago Cubs play in Wrigley Field located at (41.948314, -87.655397)
The Chicago White Sox play in U.S. Cellular Field located at (41.829892, -87.633703)
The Cincinnati Reds play in Great American Ballpark located at (39.097213, -84.506483)
The Cleveland Indians play in Progressive Field located at (41.496005, -81.685326)
The Colorado Rockies play in Coors Field located at (39.756175, -104.99413)
The Detroit Tigers play in Comerica Park located at (42.339063, -83.048627)
The Houston Astros play in Minute Maid Park located at (29.757041, -95.355429)
The Kansas City Royals play in Kauffman Stadium located at (39.051604, -94.480149)
The Los Angeles Dodgers play in Dodger Stadium located at (34.073878, -118.239951)
The Miami Marlins play in Marlins Park located at (25.778165, -80.219541)
The Milwaukee Brewers play in Miller Park located at (43.028232, -87.970966)
The Minnesota Twins play in Target Field located at (44.981749, -93.278026)
The New York Mets play in Citi Field located at (40.757134, -73.84584)
The New York Yankees play in Yankee Stadium II located at (40.829586, -73.926413)
The Oakland Athletics play in Oakland-Alameda County Coliseum located at (37.751619, -122.200451)
The Philadelphia Phillies play in Citizens Bank Park located at (39.906109, -75.166485)
The Arizona Diamondbacks play in Chase Field located at (33.44542, -112.066793)
The Pittsburgh Pirates play in PNC Park located at (40.446835, -80.005683)
The San Diego Padres play in Petco Park located at (32.70753, -117.157056)
The Seattle Mariners play in Safeco Field located at (47.591443, -122.332283)
The San Francisco Giants play in AT&T Park located at (37.778324, -122.389221)
The St. Louis Cardinals play in Busch Stadium III located at (38.622622, -90.192841)
The Tampa Bay Devil Rays play in Tropicana Field located at (27.768254, -82.653431)
The Toronto Blue Jays play in Rogers Centre located at (43.641256, -79.389054)
The Washington Nationals play in Nationals Park located at (38.872987, -77.007435)
```

Table A2: Gurobi Output for 30 Ballparks Solution



```
Changed value of parameter lazyConstraints to 1
   Prev: 0  Min: 0  Max: 1  Default: 0
Optimize a model with 30 rows, 435 columns and 870 nonzeros
Variable types: 0 continuous, 435 integer (435 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [7e+00, 3e+03]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e+00, 2e+00]
Found heuristic solution: objective 31446.7
Presolve time: 0.05s
Presolved: 30 rows, 435 columns, 870 nonzeros
Variable types: 0 continuous, 435 integer (435 binary)

Root relaxation: objective 7.939909e+03, 40 iterations, 0.04 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 7939.90870    0    8 31446.6530 7939.90870  74.8%     -    0s
     0     0 8136.73738    0    6 31446.6530 8136.73738  74.1%     -    0s
     0     0 8155.04490    0   10 31446.6530 8155.04490  74.1%     -    0s
     0     2 8692.72658    0    6 31446.6530 8692.72658  72.4%     -    0s
*   15     3               4    8962.7654680 8905.63383  0.64%   4.1    0s

Cutting planes:
  Zero half: 3
  Lazy constraints: 13

Explored 23 nodes (134 simplex iterations) in 0.54 seconds
Thread count was 4 (of 4 available processors)

Solution count 2: 8962.77 31446.7
Pool objective bound 8962.77

Optimal solution found (tolerance 1.00e-04)
Best objective 8.962765467966e+03, best bound 8.962765467966e+03, gap 0.0000%
```

Table A3 Current Ballparks Optimal Tour:



```
Optimal tour: [0, 23, 21, 1, 11, 27, 14, 2, 7, 22, 29, 3, 20, 18, 17, 4, 28, 8, 10, 6, 5, 15, 16, 26, 12, 9, 24, 25, 19, 13]
Optimal cost: 8962.77

Angel Stadium of Anaheim, Anaheim
Petco Park, San Diego
Chase Field, Phoenix
Globe Life Park, Arlington
Minute Maid Park, Houston
Tropicana Field, St. Petersburg
Marlins Park, Miami
SunTrust Park, Atlanta
Great American Ballpark, Cincinnati
PNC Park, Pittsburgh
Nationals Park, Washington
Oriole Park at Camden Yards, Baltimore
Citizens Bank Park, Philadelphia
Yankee Stadium II, New York
Citi Field, New York
Fenway Park, Boston
Rogers Centre, Toronto
Progressive Field, Cleveland
Comerica Park, Detroit
U.S. Cellular Field, Chicago
Wrigley Field, Chicago
Miller Park, Milwaukee
Target Field, Minneapolis
Busch Stadium III, St. Louis
Kauffman Stadium, Kansas City
Coors Field, Denver
Safeco Field, Seattle
AT&T Park, San Francisco
Oakland-Alameda County Coliseum, Oakland
Dodger Stadium, Los Angeles
```

Optimal 30 Ballpark Path

1	Angel Stadium of Anaheim, Anaheim	11	National Park, Washington	21	Wrigley Field, Chicago
2	Petco Park , San Diego	12	Oriole Park at Camden Yards, Baltimore	22	Miller Park, Milwuakee
3	Chase Field, Phoenix	13	Citizens Bank Park, Philadelphia	23	Target Field, Minneapolis
4	Globe Life Park, Arlington	14	Yankee Stadium II, New York	24	Bush Stadium III, St. Louis
5	Minute Maid Park, Houston	15	Citi Field, New York	25	Kauffman Stadium, Kansas City
6	Tropicana Field, St. Petersburg	16	Fenway Park, Boston	26	Coors Field, Denver
7	Marlins Park, Miami	17	Rogers Centre, Toronto	27	Safeco Field, Seattle
8	SunTrust Park, Atlanta	18	Progressive Field, Cleveland	28	AT&T Park, San Francisco
9	Great American Ballpark, Cincinnati	19	Comerica Park, Detroit	29	Oakland-Alameda County Coliseum, Oakland
10	PNC Park, Pittsburgh	20	U.S. Cellular Field, Chicago	30	Dodger Stadium, Los Angeles

Optimal 253 Ballpark Path:

1. Riverside Park, Albany
2. Hampden Park Race Track, Springfield
3. Agricultural County Fair Grounds I, Worcester
4. Agricultural County Fair Grounds II, Worcester
5. Worcester Driving Park Grounds, Worcester
6. Braves Field, Boston
7. Fenway Park, Boston
8. Congress Street Grounds, Boston
9. Huntington Avenue Baseball Grounds, Boston
10. South End Grounds III, Boston
11. South End Grounds II, Boston
12. South End Grounds I, Boston
13. Dartmouth Grounds, Boston
14. Messer Street Grounds, Providence
15. Adelaide Avenue Grounds, Providence
16. Rocky Point Park, Warwick
17. Hartford Trotting Park, Hartford
18. Hartford Ball Club Grounds, Hartford
19. Mansfield Club Grounds, Middletown
20. Hamilton Park, New Haven
21. Howard Avenue Grounds, New Haven
22. Shea Stadium, New York
23. Citi Field, New York
24. Yankee Stadium I, New York
25. Yankee Stadium II, New York
26. Hilltop Park, New York
27. Polo Grounds V, New York
28. Polo Grounds IV, New York
29. Polo Grounds III, New York
30. Metropolitan Park, New York
31. Polo Grounds I (Southeast Diamond), New York
32. Polo Grounds II (Southwest Diamond), New York
33. Monitor Grounds, Weehawken
34. West New York Field Club Grounds, West New York
35. Oakland Park, Jersey City
36. Union Grounds, Brooklyn

37. Long Island Grounds, Maspeth
38. Grauer's Ridgewood Park, Queens
39. Wallace's Ridgewood Park, Queens
40. Eastern Park, Brooklyn
41. Capitoline Grounds, Brooklyn
42. Ebbets Field, Brooklyn
43. Washington Park I, Brooklyn
44. Washington Park II, Brooklyn
45. Washington Park IV, Brooklyn
46. Washington Park III, Brooklyn
47. St. George Cricket Grounds, New York
48. Roosevelt Stadium, Jersey City
49. Wiedenmeyer's Park, Newark
50. Harrison Field, Harrison
51. Waverly Fairgrounds, Waverly
52. Oakdale Park, Philadelphia
53. Forepaugh Park, Philadelphia
54. Huntingdon Grounds I, Philadelphia
55. Huntingdon Grounds II, Philadelphia
56. Baker Bowl, Philadelphia
57. Shibe Park, Philadelphia
58. Recreation Park, Philadelphia
59. Centennial Park, Philadelphia
60. Jefferson Street Grounds, Philadelphia
61. Columbia Park, Philadelphia
62. University of Penn. Athletic Field, Philadelphia
63. Keystone Park, Philadelphia
64. Veterans Stadium, Philadelphia
65. Citizens Bank Park, Philadelphia
66. Gloucester Point Grounds, Gloucester City
67. Fireworks Park, Gloucester City
68. Union Street Park, Wilmington
69. Fairview Park Fair Grounds, Dover
70. Memorial Stadium, Baltimore
71. Terrapin Park, Baltimore
72. Oriole Park IV, Baltimore
73. Oriole Park II, Baltimore
74. Oriole Park I, Baltimore
75. Oriole Park III, Baltimore
76. Madison Avenue Grounds, Baltimore
77. Newington Park, Baltimore
78. Belair Lot, Baltimore
79. Monumental Park, Baltimore
80. Oriole Park at Camden Yards, Baltimore
81. Robert F. Kennedy Stadium, Washington
82. American League Park I, Washington
83. Boundary Field, Washington
84. Griffith Stadium, Washington

85. American League Park II, Washington
86. Athletic Park, Washington
87. Olympic Grounds, Washington
88. National Grounds, Washington
89. Swampoodle Grounds, Washington
90. Capitol Grounds, Washington
91. Nationals Park, Washington
92. Allens Pasture, Richmond
93. Richmond Fair Grounds, Richmond
94. Fort Bragg Field, Fort Bragg
95. Estadio Hiram Bithorn, San Juan
96. Marlins Park, Miami
97. Pro Player Stadium, Miami
98. Tropicana Field, St. Petersburg
99. Champion Stadium, Lake Buena Vista
100. Turner Field, Atlanta
101. Atlanta-Fulton County Stadium, Atlanta
102. SunTrust Park, Atlanta
103. Louisville Baseball Park, Louisville
104. Eclipse Park III, Louisville
105. Eclipse Park II, Louisville
106. Eclipse Park I, Louisville
107. Busch Stadium II, St. Louis
108. Busch Stadium III, St. Louis
109. Red Stocking Base-Ball Park, St. Louis
110. Handlan's Park, St. Louis
111. Palace Park of America, St. Louis
112. Sportsman's Park III, St. Louis
113. Grand Avenue Park, St. Louis
114. Sportsman's Park I, St. Louis
115. Sportsman's Park II, St. Louis
116. Robison Field, St. Louis
117. Perry Park, Keokuk
118. Kauffman Stadium, Kansas City
119. Exposition Park, Kansas City
120. Municipal Stadium, Kansas City
121. Association Park, Kansas City
122. Athletic Park, Kansas City
123. Gordon and Koppel Field, Kansas City
124. Arlington Stadium, Arlington
125. Globe Life Park, Arlington
126. Minute Maid Park, Houston
127. Colt Stadium, Houston
128. Astrodome, Houston
129. Estadio Monterrey, Monterrey
130. Chase Field, Phoenix
131. Cashman Field, Las Vegas
132. Jack Murphy Stadium, San Diego

133. Petco Park, San Diego
134. Angel Stadium of Anaheim, Anaheim
135. Wrigley Field, Los Angeles
136. Los Angeles Memorial Coliseum, Los Angeles
137. Dodger Stadium, Los Angeles
138. Oakland-Alameda County Coliseum, Oakland
139. AT&T Park, San Francisco
140. Seals Stadium, San Francisco
141. Candlestick Park, San Francisco
142. Aloha Stadium, Honolulu
143. Sydney Cricket Ground, Sydney
144. Tokyo Dome, Tokyo
145. Kingdome, Seattle
146. Safeco Field, Seattle
147. Sicks' Stadium, Seattle
148. Mile High Stadium, Denver
149. Coors Field, Denver
150. Metropolitan Stadium, Bloomington
151. Target Field, Minneapolis
152. Athletic Park, Minneapolis
153. Hubert H. Humphrey Metrodome, Minneapolis
154. Ft. Street Grounds, St. Paul
155. Virginia State Agricultural Society Fair Grounds, Rockford
156. West Side Grounds, Chicago
157. West Side Park, Chicago
158. Comiskey Park I, Chicago
159. U.S. Cellular Field, Chicago
160. South Side Park III, Chicago
161. South Side Park I, Chicago
162. South Side Park II, Chicago
163. 23rd Street Park, Chicago
164. White Stocking Park, Chicago
165. White Stocking Grounds, Chicago
166. Wrigley Field, Chicago
167. Miller Park, Milwaukee
168. County Stadium, Milwaukee
169. Milwaukee Base-Ball Grounds, Milwaukee
170. Lloyd Street Grounds, Milwaukee
171. Wright Street Grounds, Milwaukee
172. Borchert Field, Milwaukee
173. Ramona Park, Grand Rapids
174. Jailhouse Flats, Fort Wayne
175. Grand Duchess, Fort Wayne
176. Bruce Grounds, Indianapolis
177. Seventh Street Park III, Indianapolis
178. Seventh Street Park I, Indianapolis
179. Seventh Street Park II, Indianapolis
180. Federal League Park, Indianapolis

181. South Street Park, Indianapolis
182. Indianapolis Park, Indianapolis
183. Avenue Grounds, Cincinnati
184. Bank Street Grounds, Cincinnati
185. Crosley Field, Cincinnati
186. League Park I, Cincinnati
187. League Park II, Cincinnati
188. Palace of the Fans, Cincinnati
189. Lincoln Park Grounds, Cincinnati
190. Ludlow Baseball Park, Ludlow
191. Star Baseball Park, Covington
192. Riverfront Stadium, Cincinnati
193. Great American Ballpark, Cincinnati
194. Pendleton Park, Cincinnati
195. Fairview Park, Dayton
196. Recreation Park II, Columbus
197. Recreation Park I, Columbus
198. Neil Park II, Columbus
199. Neil Park I, Columbus
200. Tri-State Fair Grounds, Toledo
201. League Park, Toledo
202. Armory Park, Toledo
203. Speranza Park, Toledo
204. Burns Park, Detroit
205. Bennett Park, Detroit
206. Tiger Stadium, Detroit
207. Recreation Park, Detroit
208. Comerica Park, Detroit
209. Cleveland Stadium, Cleveland
210. Progressive Field, Cleveland
211. Beyerle's Park, Newburgh Township
212. Brotherhood Park, Cleveland
213. National Association Grounds, Cleveland
214. National League Park I, Cleveland
215. National League Park II, Cleveland
216. League Park IV, Cleveland
217. League Park III, Cleveland
218. Cedar Avenue Driving Park, Cleveland
219. Euclid Beach Park, Collinwood
220. Geauga Lake Grounds, Geauga Lake
221. Mahaffey Park, Canton
222. Pastime Park, Canton
223. Island Grounds, Wheeling
224. Union Park, Pittsburgh
225. Recreation Park, Pittsburgh
226. Three Rivers Stadium, Pittsburgh
227. Exposition Park II, Pittsburgh
228. Exposition Park III, Pittsburgh

229. Exposition Park I, Pittsburgh
230. PNC Park, Pittsburgh
231. Forbes Field, Pittsburgh
232. Columbia Park, Altoona
233. Bowman Field, Williamsport
234. Maple Avenue Driving Park, Elmira
235. International Fair Association Grounds, Buffalo
236. Olympic Park II, Buffalo
237. Olympic Park I, Buffalo
238. Riverside Grounds, Buffalo
239. Exhibition Stadium, Toronto
240. Rogers Centre, Toronto
241. Ontario Beach Grounds, Rochester
242. Windsor Beach Grounds, Irondequoit
243. Culver Field II, Rochester
244. Culver Field I, Rochester
245. Three Rivers Park, Three Rivers
246. Iron Pier, Syracuse
247. Star Park II, Syracuse
248. Star Park I, Syracuse
249. Parc Jarry, Montreal
250. Stade Olympique, Montreal
251. Haymakers' Grounds, Troy
252. Putnam Grounds, Troy
253. Troy Ball Club Grounds, Watervliet