Assignment 4: The Traveling Salesman Problem

In our last assignment, we added crossover to the mix, completing the last major piece of our evolutionary algorithm framework! Let's use this basic pipeline to explore a real-world applied problem to help gain an understanding of the complexity of using respresnetations beyond basic bit-strings.

In this assignment, we'll explore the Traveling Salesman Problem (TSP). This problem specifies that a solution should visit each of a list of locations exactly one time, and do so by traveling the shortest distance possible.

In [28]:
```
# imports
import numpy as np
import copy
import matplotlib.pyplot as plt
plt.style.use('seaborn')

import scikits.bootstrap as bootstrap
import warnings
warnings.filterwarnings('ignore') # Danger, Will Robinson! (not a scalable hack, and may s

import scipy.stats # for finding statistical significance

from IPython.display import clear_output # for real-time displays
import time # in case you want to analyze runtime of your code
import geopy.distance
import random
import math
```

Most of my family's road trips are based around getting outside a seeing national parks, so let's optimize a TSP to visit the list of US National Parks. This list of all parks, and their coordinates, is included in the zip folder for this assignment. The below block of code will load this data, ignoring the parks outside the continental US (so we can pretend to drive, instead of flying to Alaska or Hawaii). Ignoring these, we're left with a list of 50 parks (what counts as a "National Park" is a bit fuzzy, but this is the list I scraped from Wikipedia). For the most part we'll actually be ignoring the `park_names` and working with `park_lat_long`, which is an 2-D array that provides the coordiantes of each park (indexed in alphabetical order to match the names list -- not that this matters much for our purposes in this assignment).

In [29]:
```
file = open("parks_list.csv", encoding='utf8')
park_names = []
park_lat_long = []
for line in file: # get name, latitute, and longitude of park from file
    park_name = line.split(",")[0].strip()
    park_lat = float(line.split(",")[1].strip())
    park_long = float(line.split(",")[2].strip())

    if park_lat > -125 and park_lat < -65 and park_long > 25 and park_lat < 50: # just loc
        park_names.append(park_name)
        park_lat_long.append([park_lat,park_long])

file.close()
park_lat_long = np.array(park_lat_long) # convert to numpy array for easier indexing/slici

print("Number of National Parks in Continental US:",len(park_names))
print(park_names)

print(park_lat_long)
```

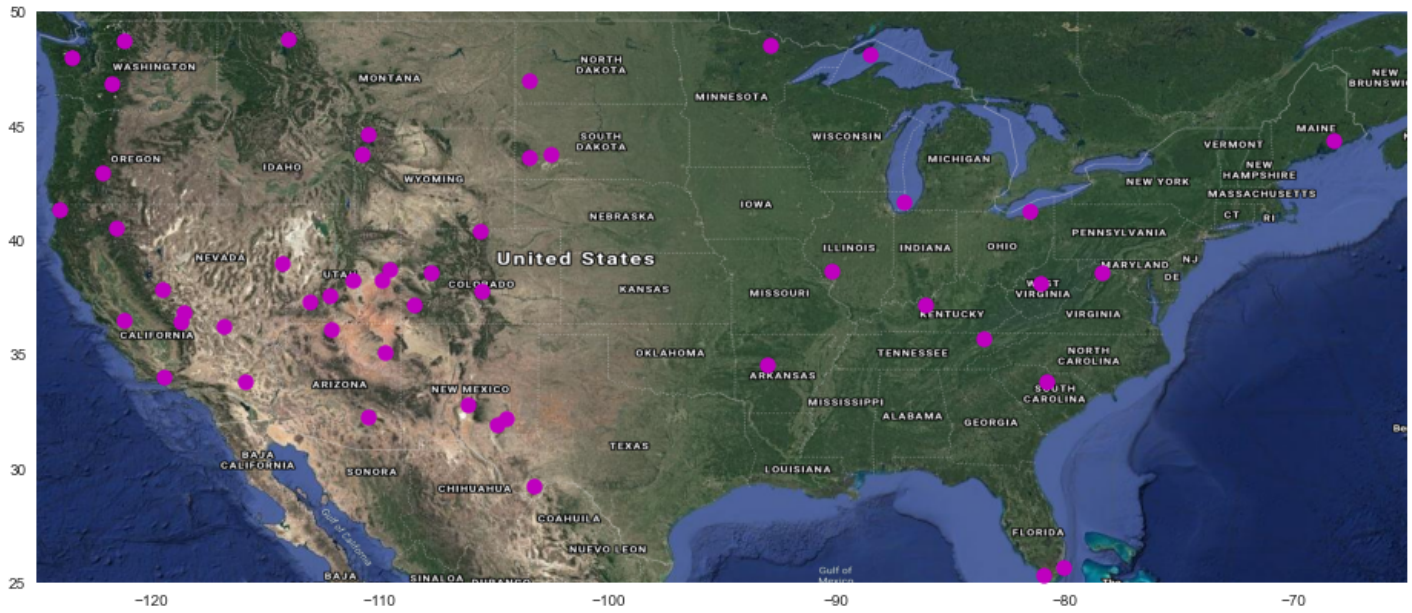
```
Number of National Parks in Continental US: 50
['Acadia', 'Arches', 'Badlands', 'Big Bend', 'Biscayne', 'Black Canyon of the Gunnison',
'Bryce Canyon', 'Canyonlands', 'Capitol Reef', 'Carlsbad Caverns', 'Channel Islands', 'Con
garee', 'Crater Lake', 'Cuyahoga Valley', 'Death Valley', 'Everglades', 'Gateway Arch', 'G
lacier', 'Grand Canyon', 'Grand Teton', 'Great Basin', 'Great Sand Dunes', 'Great Smoky Mo
untains', 'Guadalupe Mountains', 'Hot Springs', 'Indiana Dunes', 'Isle Royale', 'Joshua Tr
ee', 'Kings Canyon', 'Lassen Volcanic', 'Mammoth Cave', 'Mesa Verde', 'Mount Rainier', 'Ne
w River Gorge', 'North Cascades', 'Olympic', 'Petrified Forest', 'Pinnacles', 'Redwood',
'Rocky Mountain', 'Saguaro', 'Sequoia', 'Shenandoah', 'Theodore Roosevelt', 'Voyageurs',
'White Sands', 'Wind Cave', 'Yellowstone', 'Yosemite', 'Zion']
[[ -68.21     44.35  ]
 [-109.57     38.68  ]
 [-102.5      43.75  ]
 [-103.25     29.25  ]
 [ -80.08     25.65  ]
 [-107.72     38.57  ]
 [-112.18     37.57  ]
 [-109.93     38.2   ]
 [-111.17     38.2   ]
 [-104.44     32.17  ]
 [-119.42     34.01  ]
 [ -80.78     33.78  ]
 [-122.1      42.94  ]
 [ -81.55     41.24  ]
 [-116.82     36.24  ]
 [ -80.93     25.32  ]
 [ -90.19     38.63  ]
 [-114.       48.8   ]
 [-112.14     36.06  ]
 [-110.8      43.73  ]
 [-114.3      38.98  ]
 [-105.51     37.73  ]
 [ -83.53     35.68  ]
 [-104.87     31.92  ]
 [ -93.05     34.51  ]
 [ -87.0524   41.6533]
 [ -88.55     48.1   ]
 [-115.9      33.79  ]
 [-118.55     36.8   ]
 [-121.51     40.49  ]
 [ -86.1      37.18  ]
 [-108.49     37.18  ]
 [-121.75     46.85  ]
 [ -81.08     38.07  ]
 [-121.2      48.7   ]
 [-123.5      47.97  ]
 [-109.78     35.07  ]
 [-121.16     36.48  ]
 [-124.       41.3   ]
 [-105.58     40.4   ]
 [-110.5      32.25  ]
 [-118.68     36.43  ]
 [ -78.35     38.53  ]
 [-103.45     46.97  ]
 [ -92.88     48.5   ]
 [-106.17     32.78  ]
 [-103.48     43.57  ]
 [-110.5      44.6   ]
 [-119.5      37.83  ]
 [-113.05     37.3   ]]
```

Given that we know the coordinates of each park, a fun dataviz we can do is to superimpose the parks on a map of the United States.

In [30]:

```
fig, ax = plt.subplots(figsize=(16, 8)) # generate figure and axes
```

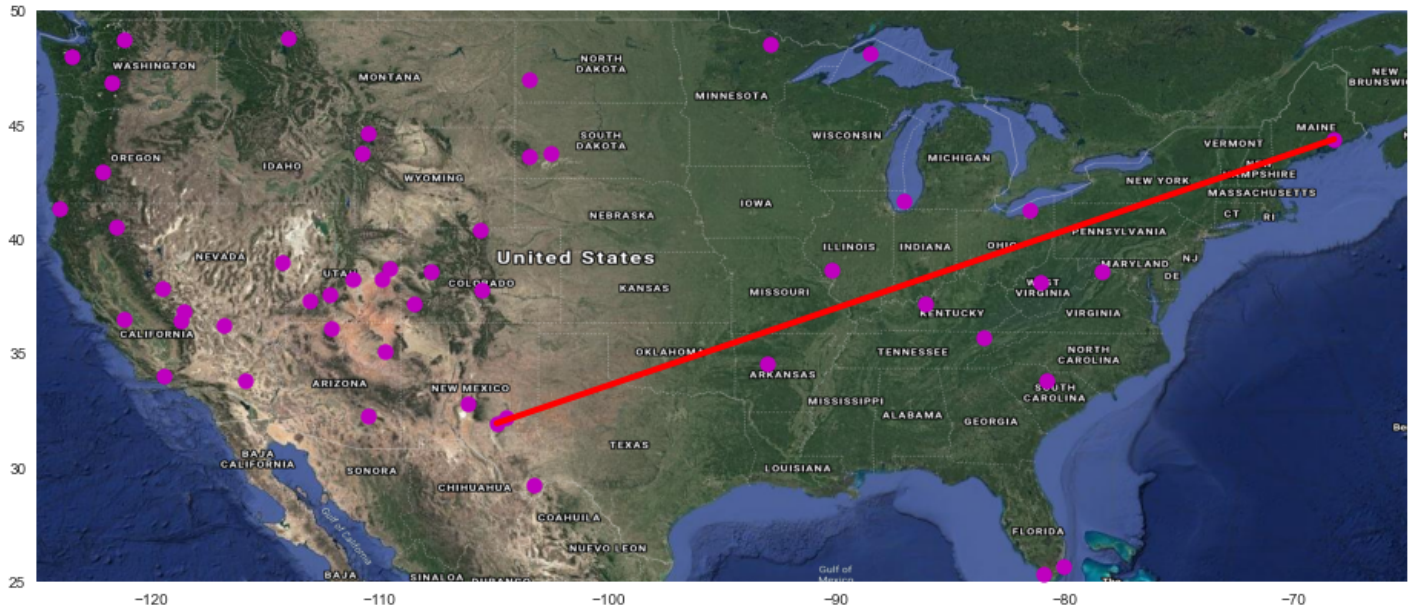
```
ax.imshow(plt.imread("us_map.png"), extent=[-125, -65, 25, 50]) # import backbround
ax.scatter(park_lat_long[:,0],park_lat_long[:,1],color='m',s=100) # plot each park at its
ax.set_xlim([-125,-65]);
ax.set_ylim([25,50]);
ax.grid(False);
```



Since we'll be opitmizing routes between these parks, it would also be good to be able to draw routes between two parks. For the sake of simplicity (i.e. so you don't need to interface with the Google Maps API), we'll just consider straight-line distance between parks as a proxy for driving distance. An example of how to draw a line showing a path between parks might look something like this.

In [31]:

```
fig, ax = plt.subplots(figsize=(16, 8)) # generate figure and axes
ax.imshow(plt.imread("us_map.png"), extent=[-125, -65, 25, 50]) # import backbround
ax.scatter(park_lat_long[:,0],park_lat_long[:,1],color='m',s=100) # plot each park at its
ax.set_xlim([-125,-65]);
ax.set_ylim([25,50]);
ax.grid(False);

park1_index = 0 # pick to arbitrary parks for this demo example
park2_index = 23
ax.plot([park_lat_long[park1_index,0],park_lat_long[park2_index,0]],[park_lat_long[park1_i
```



Q1: TSP Ftiness Function

Define a fitness function that takes a genome of park indexes and calculates the total round trip distance from the starting point (first park) back around a full cycle to end up at the park you started in.

Recall that we are using the straight-line distance between parks (and you may find it useful to make a helper function which calculated the distance between any two indidivual parks in calculating the total distance traveled).

To minimize the modifications necessary to your existing evolutionary algorithm code, which maximized fitness, return the negative of the distance traveled (i.e. fitness will always be less than zero, with longer trips more negative), so we can continue to maximize fitness values when finding the shortest part for the TSP.

Hint: Even though the fitness function definition comes first, it may be easier to complete after defining the genome representation below

In [32]:

```
def distance_between_parks(park1_index,park2_index):
    """ Optional helper function to calculate straight-line distance between two parks

        parameters:
        park1_index: (int) location of first park in genome
        park2_index: (int) location of second park in genome

        returns:
        distance: (float) straight-line distance between parks
    """


    coords_1 = (park_lat_long[park1_index,1], park_lat_long[park1_index,0])
    coords_2 = (park_lat_long[park2_index,1], park_lat_long[park2_index,0])

#     return geopy.distance.distance(coords_1, coords_2).km

    # earth radius
    R = 6373.0


    lat1 = math.radians(coords_1[0])
    lon1 = math.radians(coords_1[1])

    lat2 = math.radians(coords_2[0])
    lon2 = math.radians(coords_2[1])

    # distance
    dlon = lon2 - lon1
    dlat = lat2 - lat1

    # Haversine formula
    a = math.sin(dlat / 2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon / 2)**2

    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

    distance = R * c

#     distance = 1. * (math.sqrt((coords_2[0] - coords_1[0])**2 + (coords_2[1] - coords_1

    return distance
```

```
def traveling_salesman_fitness_function(genome):
    """ Fitness function for TSP.  Round trip (i.e. full cycle) distance traveled by the '

        parameters:
        genome: (numpy array) individual's genome containing ordered list of parks to visi

        returns:
        distance: (float) straight-line total distance for visiting all parks
    """
    fitness = 0
    for i in range(len(genome)):
        if i + 1 == len(genome):
            fitness+=distance_between_parks(genome[i], genome[0])
        else:
            fitness += distance_between_parks(genome[i], genome[i+1])

    return -1.*fitness


genome = list(range(0,50))
traveling_salesman_fitness_function(genome)
```

Out[32]:
```
-85943.84682919852
```

Q2: Adapting the Evolutionary Algorithm for the TSP

Copy over your evolutionary algorithm code (both the `Individual` class and the `evolutionary_algorithm` function) from your last assignemnt (or the posted solutions) to provide a framework to get started with. Now please adopt this code to sovle the TSP. Let's begin with the simple case of just mutation and not crossover. Let's also focus on the simpler case of a hard constraint on visiting each loaction just once.

We mentioned a number of potential representations in class last week, please pick one that enforces each location be visited only once as part of the encoding.

Let's also begin with a mutation opperator we discussed in class, randomly selecting two locations and swapping their place in the visitation order.

Additionally, for visualizations later on, please save the timeseries of solutions over time (in addition to the fitness values over time you were saving last assignment).

Hint: If your genome is a set of indexes, don't forget that they need to be ints (not floats).

Hint: If you're fitness values are all negative, keep this in mind when initializing your best-solution-so-far value.

In [33]:
```
class Individual:

    def __init__(self, fitness_function, genome_length):
        self.fitness_function = fitness_function
        self.genome = np.array(list(range(0,genome_length)))
        np.random.shuffle(self.genome)
        self.fitness = 0

    def eval_fitness(self):
        self.fitness = self.fitness_function(self.genome)
```

In [34]:
```
def evolutionary_algorithm(fitness_function=None, total_generations=100, num_parents=10, r
    """ Evolutinary Algorithm (copied from the basic hillclimber in our last assignment)
```

```
    parameters:
    fitness_funciton: (callable function) that return the fitness of a genome
                      given the genome as an input parameter (e.g. as defined in Lanc
    total_generations: (int) number of total iterations for stopping condition
    num_parents: (int) the number of parents we downselect to at each generation (mu)
    num_childre: (int) the number of children (note: parents not included in this cour
    genome_length: (int) length of the genome to be evoloved
    num_elements_to_mutate: (int) number of alleles to modify during mutation (0 = no
    crossover: (bool) whether to perform crossover when generating children
    tournament_size: (int) number of individuals competing in each tournament
    num_tournament_winners: (int) number of individuals selected as future parents fro

    returns:
    fitness_over_time: (numpy array) track record of the top fitness value at each ger
    solutions_over_time: (numpy array) track record of the top genome value at each ge
"""

# initialize record keeping
fitness_over_time = []
solutions_over_time = []

# the initialization proceedure
parents = []
children = []
for i in range(num_parents):
    parents.append(Individual(fitness_function, genome_length))

# get population fitness
for i in range(num_parents):
    parents[i].eval_fitness()

for i in range(total_generations): # repeat

    # the modification procedure
    children = []

    # inheretance
    if not crossover:
        children = copy.deepcopy(parents)

    # mutation
    if num_elements_to_mutate > 0:
        # loop through each child
        for j in range(len(children)):

            # create storage array for elements to mutate
            elements_mutated = []
            elements_mutated = random.sample(range(0,genome_length), num_elements_to_m

            # loop through the array of indices to be mutated
            for k in range(0, len(elements_mutated), 2):
                # swap the two values in the list
                one = children[j].genome[elements_mutated[k]]
                two = children[j].genome[elements_mutated[k+1]]
                children[j].genome[elements_mutated[k+1]] = one
                children[j].genome[elements_mutated[k]] = two

    # the assessement procedure
    for j in range(num_children):
        children[j].eval_fitness()

    # selection procedure
    parents += copy.deepcopy(children)
    parents = sorted(parents, key=lambda x: x.fitness, reverse=True)
    new_parents = []
```

```
        # loop for the number of parents that we want
        while len(new_parents) < num_parents:
            tournament = np.random.choice(parents, size=tournament_size)
            tournament = sorted(tournament, key=lambda individual: individual.fitness, rev
            new_parents.extend(tournament[:num_tournament_winners])

        # once we have enough new_parents to fill our parents array for the next generatio
        # and select the exact number of parents we want for the next generation
        parents = sorted(new_parents, key=lambda x: x.fitness, reverse=True)[:num_parents]
#         for z in range(10):
#             print(parents[z].fitness)

        # record keeping
        fitness_over_time.append(parents[0].fitness)
        solutions_over_time.append(parents[0].genome)
        random.shuffle(parents)

    return fitness_over_time, solutions_over_time
```

As usual, let's store our results for later plotting

In [35]:
```
experiment_results = {}
solutions = {}
```

Q2: Collect and Analyze Results

Similar to last week, let's run multiple trials to systematically test our algorithm. To keep compute times down, let's start with a smaller subset of the problem, using just the first half of the parks in our dataset.

In [36]:
```
park_names = park_names[:25]
park_lat_long = park_lat_long[:25]
```

In this smaller problem, let's run for `100` generations, `50` parents + `50` childeren in a mu+alpha evolutionary strategies, a tournament selection of tournaments of size `10` with `2` winners selected at each tournament. Let's just run this for `20` independent trials.

For reference (as there were runtime length questions about the last assignment), in my implementation of this, each run taks a little over a second, so all 20 runs finish in under 30 seconds.

In [69]:
```
num_runs = 20
total_generations = 100
num_elements_to_mutate = 1
genome_length = len(park_names)
num_parents = 50
num_children = 50
tournament_size = 10
num_tournament_winners = 2
fitness_function = traveling_salesman_fitness_function
crossover = False

experiment_results['first'] = []
solutions['first'] = []

for i in range(num_runs):
    first_fit, first_sol = evolutionary_algorithm(fitness_function, total_generations, nur
    experiment_results['first'].append(first_fit)
    solutions['first'].append(first_sol)
```

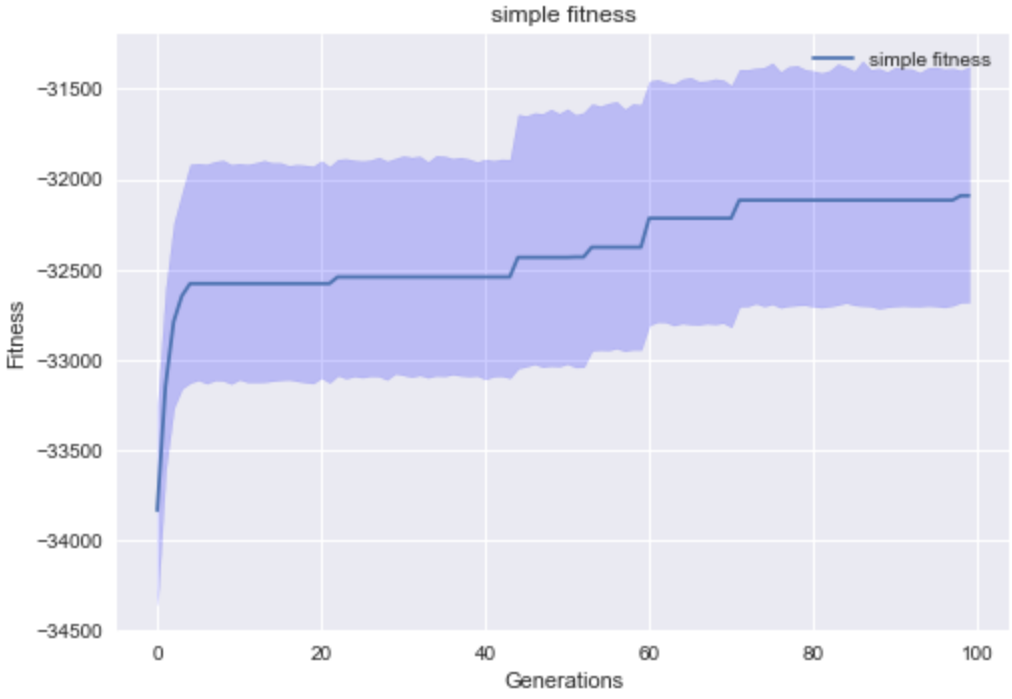
Again pulling from the previous assignment, please plot the mean and boostrapped confidence interval of your experiements. Again, you may find it convenient to make this into a function (and if you do, FYI, later use cases may involve plotting just the mean without the CI, since this boostrapping proceedure can be computationally expensive).

In [70]:
```
def plot_mean_and_bootstrapped_ci_over_time(input_data = None, name = "change me", x_label
    """

    parameters:
    input_data: (numpy array of shape (max_k, num_repitions)) solution metric to plot
    name: (string) name for legend
    x_label: (string) x axis label
    y_label: (string) y axis label

    returns:
    None
    """

    print(input_data.shape)
    generations = input_data.shape[0]

    CIs = []
    mean_values = []
    for i in range(generations):
        mean_values.append(np.mean(input_data[i]))
        CIs.append(bootstrap.ci(input_data[i], statfunction=np.mean))
    mean_values=np.array(mean_values)

    print(CIs)
    high = []
    low = []
    for i in range(len(CIs)):
        low.append(CIs[i][0])
        high.append(CIs[i][1])

    low = np.array(low)
    high = np.array(high)
    fig, ax = plt.subplots()
    y = range(0, generations)
    ax.plot(y, mean_values, label=name)
    ax.fill_between(y, high, low, color='b', alpha=.2)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.legend()
    if (name) and len(name)>0:
        ax.set_title(name)
```

In [71]:
```
# plotting
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
# plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(solutions['firs
```

```
(100, 20)
[array([-34354.83994763, -33255.99234863]), array([-33609.26858447, -32607.47512238]), arr
ay([-33264.74578399, -32243.01054014]), array([-33158.12019688, -32075.82787328]), array
([-33126.75847373, -31916.74762858]), array([-33111.7739821 , -31913.04763225]), array([-3
3130.42995039, -31916.99242988]), array([-33112.82188529, -31902.54203507]), array([-3311
4.13187778, -31895.19544483]), array([-33132.1387968 , -31918.77292106]), array([-33108.30
794489, -31913.11607311]), array([-33123.2809961, -31917.7656889]), array([-33123.5517342
, -31908.86123521]), array([-33122.24834932, -31896.1811383 ]), array([-33116.4314395 , -3
1908.78007094]), array([-33112.14470456, -31908.59555272]), array([-33108.48768506, -3192
```

4.46291528]), array([-33116.96804165, -31919.20063953]), array([-33124.18646958, -31922.03335452]), array([-33129.37111921, -31928.15815142]), array([-33099.24948636, -31898.11334957]), array([-33126.19008856, -31930.03580852]), array([-33085.90828898, -31890.2380451]), array([-33102.46738933, -31887.38297146]), array([-33090.44121996, -31894.228945]), array([-33096.70848945, -31899.11815317]), array([-33089.40263342, -31893.67034042]), array([-33090.7100591 , -31878.93261617]), array([-33109.16207504, -31900.37178853]), array([-33075.63999832, -31885.13240447]), array([-33084.1461843, -31870.3654011]), array([-33093.91065763, -31878.90570666]), array([-33084.10942332, -31872.72423329]), array([-33097.71439464, -31904.5220545]), array([-33086.65202941, -31869.03455404]), array([-33089.23798866, -31872.20466709]), array([-33080.09783729, -31885.10987693]), array([-33086.69811945, -31879.04569325]), array([-33091.01294495, -31888.39118408]), array([-33087.28716746, -31904.15041877]), array([-33106.76284067, -31888.92119294]), array([-33091.25038387, -31896.97400227]), array([-33089.54541738, -31888.70925626]), array([-33100.17825919, -31891.10283602]), array([-33046.4925348 , -31642.04236082]), array([-33036.34096063, -31649.07760303]), array([-33023.16522728, -31632.80014583]), array([-33039.20216697, -31637.21882885]), array([-33033.10814799, -31613.03950316]), array([-33036.29678779, -31636.76973543]), array([-33021.5400431 , -31611.18164016]), array([-33040.6323931 , -31643.74162769]), array([-33038.47146081, -31631.05089901]), array([-32948.52066723, -31582.61932811]), array([-32944.73901083, -31595.40411653]), array([-32946.8214466 , -31580.27733692]), array([-32935.61945594, -31570.07915717]), array([-32950.94316926, -31613.44951942]), array([-32942.49196591, -31581.98494254]), array([-32942.96106846, -31588.23977261]), array([-32807.2808884 , -31457.44092935]), array([-32789.7915866 , -31448.40862601]), array([-32791.14275887, -31464.77578239]), array([-32809.02517372, -31472.17483574]), array([-32796.92929934, -31446.20443625]), array([-32803.17130102, -31436.68440135]), array([-32805.31498613, -31458.17677689]), array([-32797.61927635, -31455.82263166]), array([-32801.94276774, -31444.86927359]), array([-32794.75886717, -31450.20695588]), array([-32817.95596687, -31482.17928376]), array([-32704.08445289, -31392.45748817]), array([-32702.9800107 , -31393.33045186]), array([-32687.03738268, -31385.39913598]), array([-32703.07257057, -31381.54604639]), array([-32690.93962258, -31356.3695314]), array([-32710.26991479, -31406.49525374]), array([-32701.67875596, -31376.26297407]), array([-32697.50709213, -31370.56047839]), array([-32695.71690404, -31393.06807616]), array([-32704.40525416, -31400.12646558]), array([-32709.22297094, -31411.73302324]), array([-32703.72605969, -31398.57101945]), array([-32695.25488089, -31361.45500929]), array([-32683.54083148, -31379.24583204]), array([-32696.65535039, -31402.34029657]), array([-32700.75651352, -31345.70215275]), array([-32703.59636876, -31393.71782116]), array([-32715.7831875 , -31386.55621769]), array([-32707.97484875, -31405.56703518]), array([-32703.60613324, -31380.6102827]), array([-32700.70888945, -31387.71170741]), array([-32703.44897298, -31390.01613249]), array([-32703.86452232, -31409.80639252]), array([-32699.41668418, -31382.44404224]), array([-32703.38382549, -31379.225612]), array([-32708.25040785, -31390.27415312]), array([-32699.01847876, -31386.07347347]), array([-32681.87436362, -31396.85332205]), array([-32681.23127504, -31378.21596879])]



One of the most fun parts of working in machine learning (for me) is to see the solutions take shape over time. Here's I've written a function that takes in your solutions_over_time from a single trial run above, and visualizes

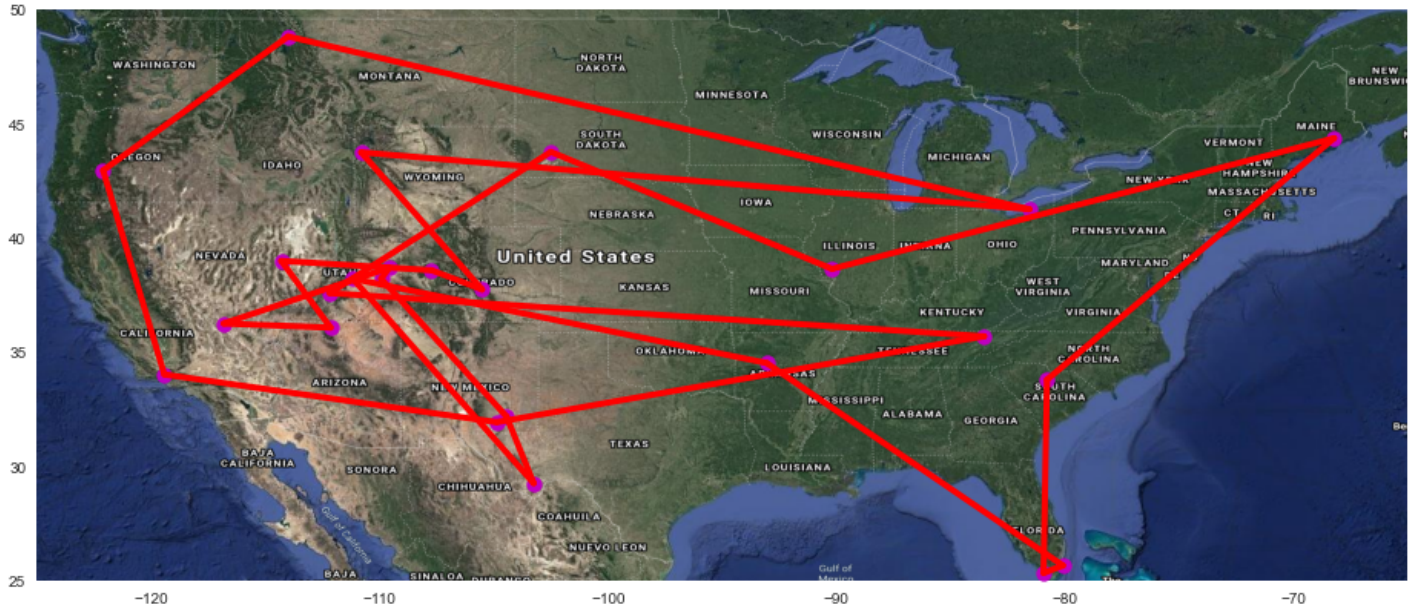
its optimization over time.

In [72]:
```
def show_solution_evolution(solutions_over_time, final_solution_only= False):
    """ Show animation of evolutionary optimization for TSP.

    parameters:
    solutions_over_time: (numpy array) track record of the top genome value at each genera
    final_solution_only: (bool) flag to skip animation

    returns:
    None
    """

    solutions_over_time = solutions_over_time.astype(int) # in case you forgot to cast ari

    last_fitness = 0
    for i in range(total_generations):

        if final_solution_only: i = total_generations-1 # skip to end if not showing full

        genome = solutions_over_time[i]
        fitness = traveling_salesman_fitness_function(genome)

        if fitness != last_fitness: # only show new solution, if different from the last g
            last_fitness = fitness
            print("Generation:",i,"\nFitness:",fitness)
            clear_output(wait=True) # erase prior figure to enable animation
            fig, ax = plt.subplots(figsize=(16, 8)) # generate figure and axes
            ax.grid(False)

            ax.imshow(plt.imread("us_map.png"), extent=[-125, -65, 25, 50]) # plot map bac
            ax.scatter(park_lat_long[:,0],park_lat_long[:,1],color='m',s=100) # plot park
            ax.set_xlim([-125,-65]);
            ax.set_ylim([25,50]);

            for park_index in range(len(genome)): # for each park in the solution
                ax.plot([park_lat_long[genome[park_index],0],park_lat_long[genome[(park_ir
            plt.show()

    last_fitness = fitness
    print("Generation:",i,"\nFitness:",fitness)
```

Q3: Inspecting Results

Write a quick script to find the best performing single run that you've found so far, and show its evolution over time using the `show_solution_evolution` funciton given above.

In [75]:
```
# plot best solution found thus far as interactive plot
best = 0
for i in range(len(experiment_results['first'])):
    if max(experiment_results['first'][best]) < max(experiment_results['first'][i]):
        best = i
show_solution_evolution(np.array(solutions['first'][best]))
```




```
Generation: 99
Fitness: -28555.27489176427
```

Q3b: Applying Intuition

One of the nice parts about viewing the solutions coming together over time is that you can see the stepping stones taken to get to good solutions, or where your algorithm may have gotten stuck. Do the results that you see make sense? Do you notice anything in particular about the intermediate solutions or search strategy that might inform further algorithmic ideas for the TSP?

The results I see here make sense to me. I think the algorithm with such high tournament selection ends up creating a very narrow population that doesn't explore much with such small mutations. The stepping stones seem to be either subnetworks in a smaller area of the map or finding edges around the outer points on the map. This could suggest that looking for subnetworks or looking at the points that have higher distances from other points is a decent strategy for determing where to travel next.

Q4: Alternative Mutation Opperators

What's a different mutation opperator that you might wanto to implement?

One that comes to mind for me would be the idea of randomly moving a single entry (gene/park) to another random location in the genome (vs. swapping two).

What are the potential implication that you could see stemming from this change (or whatever other alternative mutation opperator you want to implement below)?

What if we just take a random entry and put it at the end? The potential implication is that this will find good orderings of entries toward the end of the array. The one that you proposed, would also be interesting since it will probably find good orderings where the mutation occurs. Since we aren't swapping, this feels like a smaller change, but I'm not sure that it is. Since we are moving the single entry, this effects mostly the two entries on either side where the single entry was removed as well as itself and both entries on either side of where it was placed. In my case, only the single entry that is at the end of the list. This may be a smaller change that swapping since this will effect the two entries swapped as well ast the four entries (two for each) on either side of them. This could help with climbing if the landscape is simple enough, but it could also be detrimental if my analysis above is flawed or if the landscape is more rugged. Moving the entry to the end of the array does introduce some positional bias for the mutation.

Q4b: Implementation

Implement this mutation opperator in the cell below

In [76]:

```
def evolutionary_algorithm(fitness_function=None, total_generations=100, num_parents=10, r
    """ Evolutinary Algorithm (copied from the basic hillclimber in our last assignment)

        parameters:
        fitness_funciton: (callable function) that return the fitness of a genome
                          given the genome as an input parameter (e.g. as defined in Lanc
        total_generations: (int) number of total iterations for stopping condition
        num_parents: (int) the number of parents we downselect to at each generation (mu)
        num_childre: (int) the number of children (note: parents not included in this cour
        genome_length: (int) length of the genome to be evoloved
        num_elements_to_mutate: (int) number of alleles to modify during mutation (0 = no
        crossover: (bool) whether to perform crossover when generating children
        tournament_size: (int) number of individuals competing in each tournament
        num_tournament_winners: (int) number of individuals selected as future parents fro

        returns:
        fitness_over_time: (numpy array) track record of the top fitness value at each ger
        solutions_over_time: (numpy array) track record of the top genome value at each ge
    """

        # initialize record keeping
    fitness_over_time = []
    solutions_over_time = []

    # the initialization proceedure
    parents = []
    children = []
    for i in range(num_parents):
        parents.append(Individual(fitness_function, genome_length))

    # get population fitness
    for i in range(num_parents):
        parents[i].eval_fitness()

    for i in range(total_generations): # repeat

        # the modification procedure
        children = []

        # inheretance
        if not crossover:
            children = copy.deepcopy(parents)

        # crossover
        if crossover:
            while len(children)<(num_children):
                random.shuffle(parents)
                for j in range(0, num_parents, 2):

                    # select both parents
                    first_parent = parents[j]
                    second_parent = parents[j+1]

                    # randomly select two points for crossover
                    crossover_points = random.sample(range(0,genome_length), 2)

                    # Do we want to disallow crossover from higher to lower numbers?
                    # TODO: Remove below line and edit crossover to work when higher numbe
                    crossover_points = sorted(crossover_points)
```

```
                    # switch genes between these two points between the two individuals
                    child1 = Individual(fitness_function, genome_length)
                    child1.genome = parents[j].genome[:crossover_points[0]] + parents[j+1]
                    child2 = Individual(fitness_function, genome_length)
                    child2.genome = parents[j+1].genome[:crossover_points[0]] + parents[j]

                    # Or do we just keep the children no matter what
                    children.append(child1)
                    children.append(child2)


            if len(children) > num_children:
                children = children[:num_children]

        # mutation
        if num_elements_to_mutate > 0:
            # loop through each child
            for j in range(len(children)):

                # create storage array for elements to mutate
                elements_mutated = []
                elements_mutated = random.sample(range(0,genome_length), num_elements_to_m
                elements_mutated2 = random.sample(range(0,genome_length), num_elements_to_

                # loop through the array of indices to be mutated
                # loop through the array of indices to be mutated
                for k in range(0, len(elements_mutated)):
                    # place the element at the end of the array
                    children[j].genome = np.append(children[j].genome, children[j].genome[
                    for l in range(elements_mutated[k], genome_length):
                        children[j].genome[l] = children[j].genome[l+1]
                    children[j].genome = children[j].genome[:genome_length]
                    # place the element in a random location
#                     children[j].genome[elements_mutated2[k]] = children[j].genome[elemen

        # the assessement procedure
        for j in range(num_children):
            children[j].eval_fitness()

        # selection procedure
        parents += copy.deepcopy(children)
        random.shuffle(parents)
        new_parents = []

        # loop for the number of parents that we want
        while len(new_parents) < num_parents:
            random.shuffle(parents)
            for j in range(0, len(parents), tournament_size):
                winners = []
                for k in range(0, tournament_size):
                    if len(parents) <= j+k:
                        winners.append(parents[random.randint(0,len(parents)-1)])
                    else:
                        winners.append(parents[j+k])
                winners = sorted(winners, key=lambda x: x.fitness, reverse=True)
                winners = winners[:num_tournament_winners]
                new_parents += winners

        parents = sorted(new_parents, key=lambda x: x.fitness, reverse=True)[:num_parents]

        # record keeping
        fitness_over_time.append(parents[0].fitness)
        solutions_over_time.append(parents[0].genome)

    return fitness_over_time, solutions_over_time
```

Q5: Run and Plot Experiment

Yeah, do that.

In [84]:

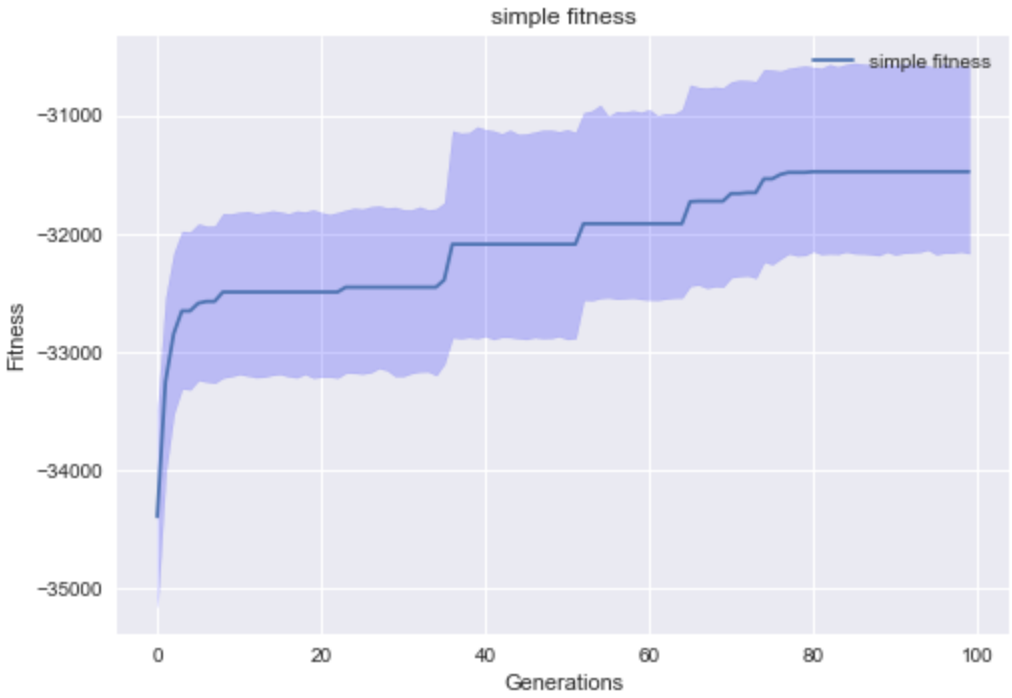
```
num_runs = 20
total_generations = 100
num_elements_to_mutate = 1
genome_length = len(park_names)
num_parents = 50
num_children = 50
tournament_size = 10
num_tournament_winners = 2
crossover = False

experiment_results['first'] = []
solutions['first'] = []

for i in range(num_runs):
    first_fit, first_sol = evolutionary_algorithm(fitness_function, total_generations, nur
    experiment_results['first'].append(first_fit)
    solutions['first'].append(first_sol)
```

In [85]:

```
# plotting
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
```

```
(100, 20)
[array([-35160.65987872, -33491.21576746]), array([-34029.30644782, -32545.62049875]), arr
ay([-33525.90204583, -32166.73390471]), array([-33314.60368209, -31981.7694085 ]), array
([-33322.42577023, -31987.81405058]), array([-33243.88783398, -31919.54996673]), array([-3
3256.83974721, -31938.5967433 ]), array([-33266.4489974 , -31937.35802777]), array([-3322
2.37025749, -31832.43295037]), array([-33210.53872013, -31834.22972512]), array([-33192.53
361713, -31820.82558211]), array([-33204.90621476, -31813.82171313]), array([-33219.050382
35, -31831.88192819]), array([-33215.02522998, -31822.81914401]), array([-33199.64423988,
-31808.60055589]), array([-33192.6197146, -31817.6269099]), array([-33211.08463898, -3183
4.21894619]), array([-33220.42038575, -31810.01479222]), array([-33191.17264756, -31818.12
178571]), array([-33222.44751976, -31800.56250942]), array([-33215.29767097, -31824.088852
26]), array([-33214.62857153, -31839.87671087]), array([-33222.22339   , -31822.7483341
1]), array([-33180.17668533, -31803.54260853]), array([-33182.33067036, -31786.55533384]),
array([-33187.7192169 , -31794.49557009]), array([-33178.45485828, -31774.45011081]), arra
y([-33141.78040553, -31766.95542827]), array([-33158.95678704, -31784.07097405]), array([-
33209.97807597, -31777.41028563]), array([-33209.03707931, -31799.13829494]), array([-3318
3.68134318, -31800.64347789]), array([-33171.41860017, -31775.12736198]), array([-33168.80
990538, -31800.93438599]), array([-33201.71690454, -31792.21972779]), array([-33103.977023
23, -31740.26208745]), array([-32880.19100904, -31131.5658026 ]), array([-32890.07953405,
-31151.55474863]), array([-32878.80112172, -31148.01313611]), array([-32888.04005336, -310
97.88184768]), array([-32870.48994217, -31120.11021921]), array([-32896.257254  , -31132.3
5047524]), array([-32877.05515592, -31156.43560641]), array([-32879.0751809 , -31125.09231
182]), array([-32889.50705465, -31161.12723019]), array([-32895.65676448, -31157.4505648
2]), array([-32879.44127971, -31142.41843055]), array([-32888.96655191, -31125.83109277]),
array([-32886.4786789 , -31125.25003164]), array([-32871.87612343, -31138.27170865]), arra
y([-32895.96655298, -31120.98577717]), array([-32889.10372199, -31145.89064692]), array([-
32563.66135966, -30978.04730048]), array([-32566.61432367, -30962.0321019 ]), array([-3254
9.20065977, -30913.97969427]), array([-32542.55074171, -31009.98425888]), array([-32555.03
373834, -30967.58364824]), array([-32552.09799933, -30971.76522933]), array([-32546.019609
07, -30960.49114084]), array([-32556.88178439, -30973.23887435]), array([-32562.20913727,
-30951.74508237]), array([-32563.52412117, -31004.3964252 ]), array([-32553.86508678, -309
86.30350602]), array([-32546.6137733 , -30988.78712469]), array([-32543.47490374, -30950.8
1254483]), array([-32444.78151935, -30743.32265459]), array([-32435.10661394, -30763.71238
63 ]), array([-32462.16581524, -30769.15233121]), array([-32449.63596774, -30760.2137911
9]), array([-32452.09418757, -30766.99642432]), array([-32372.85781097, -30716.75794483]),
array([-32363.75036386, -30702.07017124]), array([-32357.99440276, -30705.03949777]), arra
y([-32377.85129736, -30713.29033036]), array([-32240.97598674, -30611.38547045]), array([-
```

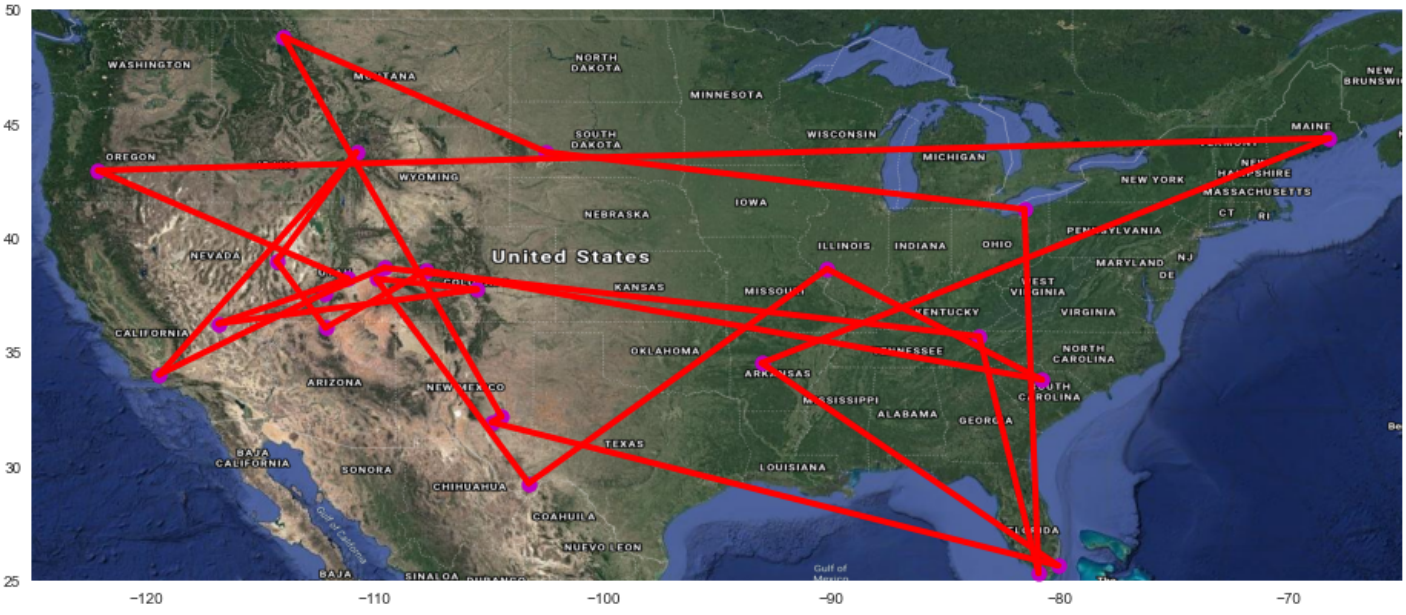
```
32263.22459205, -30616.1553363 ]), array([-32215.52676481, -30625.992067  ]), array([-3217
0.74030428, -30602.62842668]), array([-32187.55858993, -30591.5564495 ]), array([-32184.57
991218, -30581.24744276]), array([-32150.2111215 , -30592.80004628]), array([-32177.523191
31, -30601.1311503 ]), array([-32170.6926565 , -30570.41515285]), array([-32174.06324085,
-30589.27315523]), array([-32155.435903  , -30568.60147824]), array([-32169.45320647, -305
57.4974763 ]), array([-32171.96356469, -30565.48085968]), array([-32177.97746416, -30571.0
1589614]), array([-32186.07078351, -30591.29758011]), array([-32156.37967946, -30564.94602
365]), array([-32180.66862924, -30576.14103297]), array([-32166.02865727, -30568.130957
]), array([-32162.52882792, -30567.59663431]), array([-32157.21612164, -30571.32157121]),
array([-32138.95895973, -30579.46617505]), array([-32178.90727051, -30601.01843359]), arra
y([-32164.20621544, -30558.60144368]), array([-32165.1386923 , -30593.01543328]), array([-
32155.19017228, -30576.72142171]), array([-32169.19946568, -30586.65708155])]
```



It may also be helpful (or just fun) to visualize the evolution over time as well.

In [88]:

```
# interactive viz
# plot best solution found thus far as interactive plot
best = 0
for i in range(len(experiment_results['first'])):
    if max(experiment_results['first'][best]) > max(experiment_results['first'][i]):
        best = i
show_solution_evolution(np.array(solutions['first'][best]))
```



```
Generation: 99
Fitness: -33905.984560815836
```

Q5b: Analysis

Did this approach work better? Worse? Indistinguishable from the first? Was this what you expected? What does the difference in mutation opperator performance suggest to you about the methods or benefits of crossover?

Looks worse. This is understandable given that the first suffered from not exploring enough either and this, I think, is probably an even smaller mutation. It doesn't look like this mutation was able to find a stepping stone very often or at all in some cases and is thus left to the best of the initial population. I think this suggests that using crossover will help since it will add a more exploratory approach to the solution-space. I think the method for crossover should be a more significant change than the mutation operator.

Q6: Crossover

Let's also implement a crossover for the TSP. There are many approaches to crossover that may be appropriate, and you should feel free to choose any (e.g. the simplest) you want. I would encourage you to think about crossover approaches that maintain the visit-each-location-once hard constraint as part of the crossover mechanism (but as dicussed in class, you're welcome to also explore approaches that visit each location multiple times, then prune down to a single complete cycle before sending the resulting genome off for evaluation).

If you're short on ideas, this paper has a few examples that might be fun to implement.

For simplicity's sake, feel free to just add crossover in addition to your favorite of the already implemented mutation opperators (rather than having to explore every permutation of mutation and/or crossover approaches). I'll just tell you that crossover on its own (given the hyperparameters we have here) doesn't work particularly well -- as you might expect and we discussed in class.

In [89]:

```
def evolutionary_algorithm(fitness_function=None, total_generations=100, num_parents=10, r
    """ Evolutinary Algorithm (copied from the basic hillclimber in our last assignment)

        parameters:
        fitness_funciton: (callable function) that return the fitness of a genome
                          given the genome as an input parameter (e.g. as defined in Lanc
        total_generations: (int) number of total iterations for stopping condition
        num_parents: (int) the number of parents we downselect to at each generation (mu)
        num_childre: (int) the number of children (note: parents not included in this cour
        genome_length: (int) length of the genome to be evoloved
        num_elements_to_mutate: (int) number of alleles to modify during mutation (0 = no
        crossover: (bool) whether to perform crossover when generating children
        tournament_size: (int) number of individuals competing in each tournament
        num_tournament_winners: (int) number of individuals selected as future parents fro

        returns:
        fitness_over_time: (numpy array) track record of the top fitness value at each ger
    """

    # initialize record keeping
    fitness_over_time = []
    solutions_over_time = []

    # the initialization proceedure
    parents = []
    children = []
    for i in range(num_parents):
        parents.append(Individual(fitness_function, genome_length))
```

```
    # get population fitness
    for i in range(num_parents):
        parents[i].eval_fitness()

    for i in range(total_generations): # repeat

        # the modification procedure
        children = []

        # inheretance
        if not crossover:
            children = copy.deepcopy(parents)

        # crossover
        if crossover:
            while len(children)<(num_children):
                random.shuffle(parents)
                for j in range(0, num_parents, 2):

                    # select both parents
                    first_parent = parents[j]
                    second_parent = parents[j+1]

                    if len(np.unique(first_parent.genome)) + len(np.unique(second_parent.g
#                       print(sorted(first_parent.genome))
#                       print(sorted(second_parent.genome))
#                       print(len(np.unique(first_parent.genome)) + len(np.unique(second
                        return

                    # randomly select two points for crossover
                    crossover_points = random.sample(range(0,genome_length), 2)

                    # Do we want to disallow crossover from higher to lower numbers?
                    # TODO: Remove below line and edit crossover to work when higher numbe
                    crossover_points = sorted(crossover_points)

                    # switch the tail end of the genomes at this point between the two ind
                    child1 = Individual(fitness_function, genome_length)
                    child2 = Individual(fitness_function, genome_length)
                    child1.genome = np.array([None for x in range(0,genome_length)])
                    child2.genome = np.array([None for x in range(0,genome_length)])
                    child1.genome = np.concatenate([parents[j].genome[:crossover_points[0]
                    child2.genome = np.concatenate([parents[j+1].genome[:crossover_points

                    # for each of these two children look at the genomes to find duplicate
#                     print(child1.genome)
#                     print(np.unique(child1.genome, return_index=True, return_counts=True
                    # get the indices of the duplicate cities
                    duplicate_indices1 = []
                    for k in range(crossover_points[0], crossover_points[1]):
#                       print(child1.genome[k], child1.genome[:crossover_points[0]], chi
#                       print(type(child1.genome[k]), type(child1.genome[:crossover_poin
                        if (child1.genome[k] in child1.genome[:crossover_points[0]]) or (c
                            duplicate_index = (np.where(child1.genome == child1.genome[k])
#                             print(duplicate_index, type(duplicate_index))
                            duplicate_index = [d for d in duplicate_index if d not in rang
                            duplicate_indices1.append(duplicate_index)

                    duplicate_indices2 = []
                    for k in range(crossover_points[0], crossover_points[1]):
                        if (child2.genome[k] in child2.genome[:crossover_points[0]]) or (c
                            duplicate_index = (np.where(child2.genome == child2.genome[k])
                            duplicate_index = [d for d in duplicate_index if d not in rang
                            duplicate_indices2.append(duplicate_index)
```

```
#                     print('child1.genome: {}'.format(child1.genome))
#                     print('crossover_points: {}'.format(crossover_points))
#                     print('parents[j].genome: {}'.format(parents[j].genome))
#                     print('parents[j+1].genome: {}'.format(parents[j+1].genome))
#                     print('duplicate_indices1: {}'.format(duplicate_indices1))

                    # loop through the indices that are duplicates and replace the value w
                    for k in duplicate_indices1:
#                         print(k)
                        for l in range(crossover_points[0], crossover_points[1]):
#                             print('parents[j].genome[l]: {}'.format(parents[j].genome[l]
                            if parents[j].genome[l] not in child1.genome:
                                child1.genome[k] = parents[j].genome[l]
                                l = crossover_points[1]
                            else:
                                continue
#                     print('child1.genome: {}'.format(child1.genome))
#                     print()
#                     print()

#                     print('child2.genome: {}'.format(child2.genome))
#                     print('crossover_points: {}'.format(crossover_points))
#                     print('parents[j].genome: {}'.format(parents[j].genome))
#                     print('parents[j+1].genome: {}'.format(parents[j+1].genome))
#                     print('duplicate_indices2: {}'.format(duplicate_indices2))
                    # loop through the indices that are duplicates and replace the value w
                    for k in duplicate_indices2:
#                         print(k)
                        for l in range(crossover_points[0], crossover_points[1]):
#                             print('parents[j+1].genome[l]: {}'.format(parents[j+1].genor
                            if parents[j+1].genome[l] not in child2.genome:
                                child2.genome[k] = parents[j+1].genome[l]
                                l=crossover_points[1]
                            else:
                                continue
#                     print('child2.genome: {}'.format(child2.genome))
#                     print(sorted(child1.genome))
#                     print(sorted(child2.genome))
#                     print(len(np.unique(child1.genome))+ len(np.unique(child2.genome)))

                    if len(np.unique(child1.genome)) + len(np.unique(child2.genome)) < 50:
                        return

                    # Or do we just keep the children no matter what
                    children.append(child1)
                    children.append(child2)


            if len(children) > num_children:
                children = children[:num_children]

        # mutation
        if num_elements_to_mutate > 0:
            # loop through each child
            for j in range(len(children)):

                # create storage array for elements to mutate
                elements_mutated = []
                elements_mutated = random.sample(range(0,genome_length), num_elements_to_m

                # loop through the array of indices to be mutated
                for k in range(0, len(elements_mutated)):
                    # place the element at the end of the array
                    children[j].genome = np.append(children[j].genome, children[j].genome
                    for l in range(elements_mutated[k], genome_length):
                        children[j].genome[l] = children[j].genome[l+1]
```

```
                    children[j].genome = children[j].genome[:genome_length]

        # the assessement procedure
        for j in range(num_children):
            children[j].eval_fitness()

        # selection procedure
        parents += children.copy()
        random.shuffle(parents)
        new_parents = []

        # loop for the number of parents that we want
        while len(new_parents) < num_parents:
            random.shuffle(parents)
            for j in range(0, len(parents), tournament_size):
                winners = []
                for k in range(0, tournament_size):
                    if len(parents) <= j+k:
                        winners.append(parents[random.randint(0,len(parents)-1)])
                    else:
                        winners.append(parents[j+k])
                winners = sorted(winners, key=lambda x: x.fitness, reverse=True)
                winners = winners[:num_tournament_winners]
                new_parents += winners

        parents = sorted(new_parents, key=lambda x: x.fitness, reverse=True)[:num_parents]

        # record keeping
        fitness_over_time.append(parents[0].fitness)
        solutions_over_time.append(parents[0].genome)

    return fitness_over_time, solutions_over_time # for simplicity, return just the fitnes
```

Q7: Run and Plot

Just like always.

In [90]:

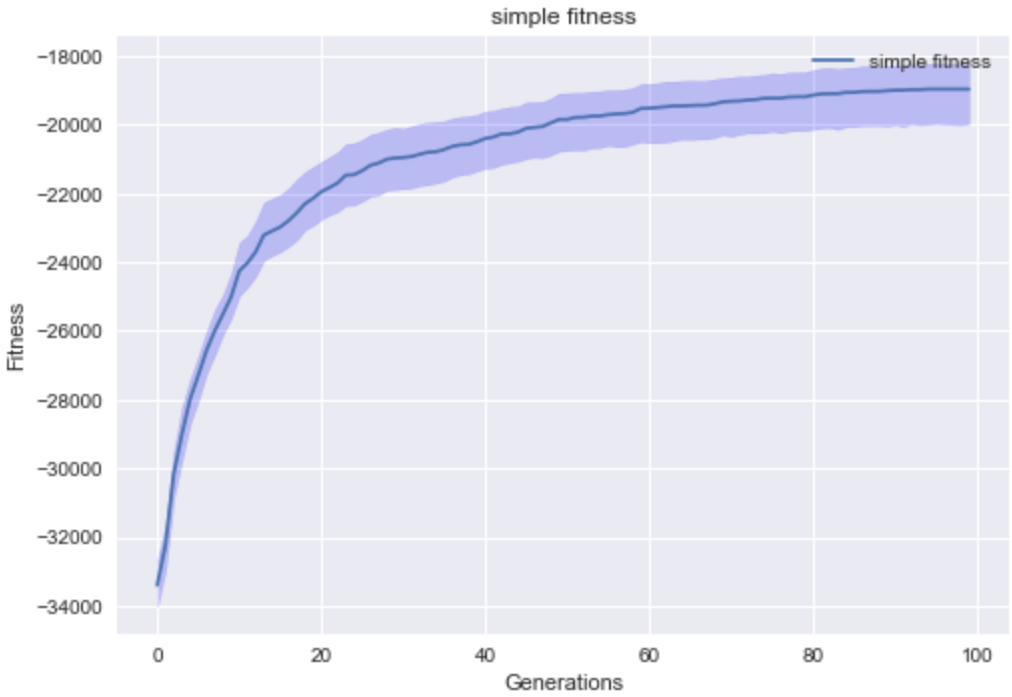
```
num_runs = 20
total_generations = 100
num_elements_to_mutate = 1
genome_length = len(park_names)
num_parents = 50
num_children = 50
tournament_size = 10
num_tournament_winners = 2
crossover = True

experiment_results['first'] = []
solutions['first'] = []

for i in range(num_runs):
    first_fit, first_sol = evolutionary_algorithm(fitness_function, total_generations, nur
    experiment_results['first'].append(first_fit)
    solutions['first'].append(first_sol)
```

In [91]:

```
# plotting
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
```

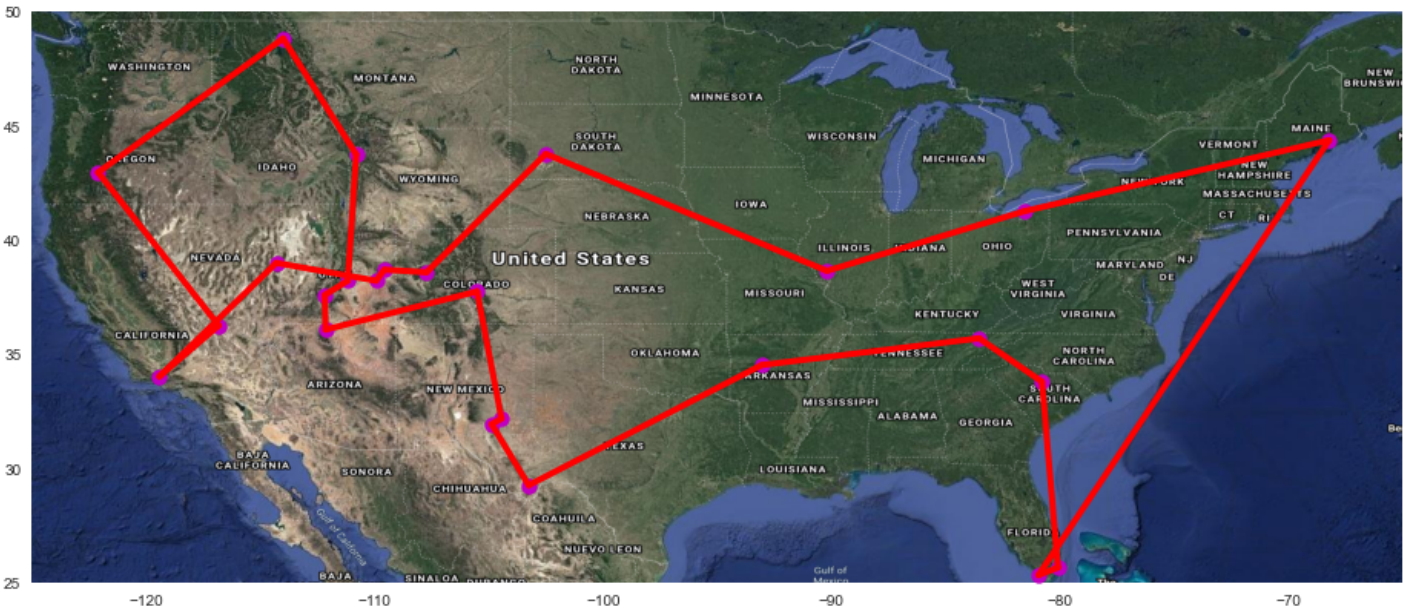
```
(100, 20)
[array([-34014.5833099 , -32639.75234833]), array([-32973.53330959, -31657.69577432]), arr
ay([-30866.8088849, -29491.6125256]), array([-29772.22212017, -28216.66967738]), array([-2
8714.2116269 , -27398.38303284]), array([-28027.72858752, -26722.73617868]), array([-2725
```

0.62938234, -26011.26207722]), array([-26705.93928282, -25387.16072552]), array([-26116.23
183577, -24944.17086628]), array([-25645.10639154, -24329.92638988]), array([-24979.200258
42, -23423.13945136]), array([-24733.92121222, -23227.92900812]), array([-24412.57124438,
-22806.21039394]), array([-23947.19803586, -22265.58969275]), array([-23814.86407333, -221
46.16686851]), array([-23704.28661052, -22044.12497954]), array([-23549.12658586, -21851.6
4306385]), array([-23359.61442116, -21592.27957943]), array([-23072.21146615, -21378.90696
749]), array([-22936.5304665, -21226.8073933]), array([-22760.37216957, -21077.41702514]),
array([-22630.12003106, -20940.20032678]), array([-22540.12428886, -20801.45115027]), arra
y([-22356.32981171, -20560.23283222]), array([-22345.52226334, -20540.10218628]), array([-
22233.80711331, -20435.7480899]), array([-22089.05651709, -20290.12064193]), array([-2205
3.06185221, -20230.10506851]), array([-21911.33298706, -20145.06741124]), array([-21892.18
504855, -20088.57801421]), array([-21870.98258256, -20119.29671686]), array([-21853.494840
89, -20047.87887318]), array([-21779.10745469, -19981.02846567]), array([-21753.74825847,
-19939.34469549]), array([-21703.29744422, -19925.22787346]), array([-21647.79796574, -198
98.95899196]), array([-21528.29768639, -19818.54195841]), array([-21463.53004879, -19776.0
6043097]), array([-21438.04460976, -19751.94677734]), array([-21350.07274652, -19714.17068
387]), array([-21281.64682511, -19616.56890738]), array([-21252.36366292, -19592.9230780
3]), array([-21159.86785896, -19529.46215004]), array([-21115.43550826, -19488.35564343]),
array([-21071.68786761, -19468.36992942]), array([-20995.84753417, -19351.24694298]), arra
y([-20948.75100199, -19346.83414409]), array([-20964.51505457, -19335.86196922]), array([-
20894.04532411, -19239.98142772]), array([-20779.74771072, -19093.6248023]), array([-2075
7.66602263, -19085.42096572]), array([-20741.70872798, -19075.81320104]), array([-20750.56
623004, -19068.5996763]), array([-20684.62459793, -19037.2148081]), array([-20677.807927
4 , -19030.68834937]), array([-20610.04940982, -18996.67619085]), array([-20649.22533686,
-18997.55197681]), array([-20646.16958103, -18978.70537272]), array([-20570.6295218 , -189
32.59101259]), array([-20503.36850106, -18823.15317617]), array([-20532.15867037, -18810.4
7514547]), array([-20532.69979476, -18794.5143662]), array([-20518.43694683, -18745.92607
661]), array([-20473.35720533, -18747.10801646]), array([-20432.38628984, -18733.7215566
8]), array([-20437.57333587, -18717.63592699]), array([-20425.89777916, -18730.11281151]),
array([-20422.15205331, -18720.94195113]), array([-20380.3815623 , -18678.90893012]), arra
y([-20305.90165348, -18647.4477948]), array([-20351.42306299, -18630.69303552]), array([-
20291.54449645, -18596.46653085]), array([-20248.97783225, -18602.19332888]), array([-2025
4.07032512, -18570.30367847]), array([-20247.3788758 , -18552.66647847]), array([-20200.69
598438, -18500.91639924]), array([-20245.98884613, -18528.67022643]), array([-20196.724786
79, -18484.87604045]), array([-20178.43900537, -18477.91064025]), array([-20173.61857637,
-18475.92332652]), array([-20145.7318431, -18410.3985338]), array([-20103.96459871, -1836
2.22710597]), array([-20096.6763573 , -18348.63709976]), array([-20144.47753788, -18379.43
569741]), array([-20070.48862705, -18349.07575338]), array([-20066.56587541, -18342.552776
63]), array([-20034.55863881, -18302.3339348]), array([-20036.19494893, -18298.3914380
1]), array([-20029.04067149, -18319.94524235]), array([-20051.93910368, -18294.90248713]),
array([-20010.46118299, -18276.91287508]), array([-20062.04363365, -18275.05215918]), arra
y([-19982.61158484, -18245.69185208]), array([-20019.27650556, -18262.2810417]), array([-
20005.05528231, -18231.98020264]), array([-19980.27585146, -18213.98825166]), array([-1999
9.24992524, -18240.66670568]), array([-20010.35145849, -18210.97181867]), array([-20021.09
906136, -18248.97816597]), array([-19984.05889674, -18232.3451811])]



In [92]:

```
# interactive viz
# plot best solution found thus far as interactive plot
best = 0
for i in range(len(experiment_results['first'])):
    if max(experiment_results['first'][best]) < max(experiment_results['first'][i]):
        best = i
show_solution_evolution(np.array(solutions['first'][best]))
```




```
Generation: 99
Fitness: -16157.220560414704
```

Q7b: Analysis

Did crossover help? Any thoughts as to why or why not? Is this what you expected?

Crossover did help. There were many more stepping stones and intermediate solutions. Again, I think this is because crossover added diversity to the search and a more exploratory approach to finding solutions. This is better than I expected.

Q8: Scaling Up

Now that you've tested some approaches on a subset of the data, let's try the best method on the full park list. First let's re-import the complete parks list.

In [93]:
```
file = open("parks_list.csv", encoding='utf8')
park_names = []
park_lat_long = []
for line in file: # get name, latitute, and longitude of park from file
    park_name = line.split(",")[0].strip()
    park_lat = float(line.split(",")[1].strip())
    park_long = float(line.split(",")[2].strip())

    if park_lat > -125 and park_lat < -65 and park_long > 25 and park_lat < 50: # just loc
        park_names.append(park_name)
        park_lat_long.append([park_lat,park_long])

file.close()
park_lat_long = np.array(park_lat_long) # convert to numpy array for easier indexing/slici

print("Number of National Parks in Continental US:",len(park_names))
print(park_names)
```

```
Number of National Parks in Continental US: 50
['Acadia', 'Arches', 'Badlands', 'Big Bend', 'Biscayne', 'Black Canyon of the Gunnison',
'Bryce Canyon', 'Canyonlands', 'Capitol Reef', 'Carlsbad Caverns', 'Channel Islands', 'Con
garee', 'Crater Lake', 'Cuyahoga Valley', 'Death Valley', 'Everglades', 'Gateway Arch', 'G
lacier', 'Grand Canyon', 'Grand Teton', 'Great Basin', 'Great Sand Dunes', 'Great Smoky Mo
untains', 'Guadalupe Mountains', 'Hot Springs', 'Indiana Dunes', 'Isle Royale', 'Joshua Tr
ee', 'Kings Canyon', 'Lassen Volcanic', 'Mammoth Cave', 'Mesa Verde', 'Mount Rainier', 'Ne
w River Gorge', 'North Cascades', 'Olympic', 'Petrified Forest', 'Pinnacles', 'Redwood',
'Rocky Mountain', 'Saguaro', 'Sequoia', 'Shenandoah', 'Theodore Roosevelt', 'Voyageurs',
'White Sands', 'Wind Cave', 'Yellowstone', 'Yosemite', 'Zion']
```

This larger dataset will need to run for a bit longer, let's say `1000` generations. We can keep all other parameters the same, though let's drop the number of independent trials down to `10`, again to keep computation reasonable. Each run takes a little under 20 seconds on my laptop, so this should still only take a few minutes to run.

Feel free to just run your best/favorite approach (e.g. combination of mutation and/or crossover opperators) from the smaller dataset here. No need to compare multiple implementations right now.

In [94]:
```
num_runs = 10
total_generations = 1000
num_elements_to_mutate = 1
genome_length = len(park_names)
num_parents = 50
num_children = 50
tournament_size = 10
num_tournament_winners = 2
crossover = True

experiment_results['first'] = []
solutions['first'] = []

for i in range(num_runs):
    first_fit, first_sol = evolutionary_algorithm(fitness_function, total_generations, nur
    experiment_results['first'].append(first_fit)
    solutions['first'].append(first_sol)
```

Please also plot the fitness over time. Though feel free to omit the boostrapped confidence intervals, as they take quite some time to run over 1000 generations.

In [95]:
```
# plotting
plot_mean_and_bootstrapped_ci_over_time(input_data=np.transpose(np.array(experiment_result
```

```
(1000, 10)
[array([-74144.17837491, -70884.99700745]), array([-71154.14525224, -67664.51095164]), arr
ay([-69221.57547325, -65051.37641832]), array([-66705.14903081, -62764.24123185]), array
([-65063.31942894, -60857.22703516]), array([-64345.15585027, -60309.69389724]), array([-6
2717.50764249, -58206.03489482]), array([-61235.99501083, -57467.48008982]), array([-5884
8.54635587, -56045.06523999]), array([-57775.76590196, -54667.98532137]), array([-56950.24
994216, -54189.67466262]), array([-56373.54723895, -53761.39930073]), array([-54827.189762
07, -52638.70246071]), array([-54297.82492869, -51871.19271068]), array([-53428.51931971,
-51597.4646429 ]), array([-53173.0955273, -51028.0965075]), array([-52225.4130306 , -4986
2.97993572]), array([-51672.18843593, -49567.52741903]), array([-50821.01115359, -48866.40
219312]), array([-50676.95252965, -48676.58049239]), array([-50276.35159733, -48510.884197
01]), array([-49653.10120493, -47694.19334218]), array([-48774.9071971 , -47023.3775313
4]), array([-48595.44878304, -46713.09955749]), array([-48208.26019585, -46307.53654253]),
array([-47627.3775089 , -46062.80854769]), array([-47420.22653212, -45707.75838548]), arra
y([-47068.42456045, -45112.70458005]), array([-46741.90410157, -45030.24929431]), array([-
45974.13591352, -44577.38332151]), array([-45558.60897341, -44010.22549054]), array([-4550
8.54369975, -43923.15419006]), array([-45370.77632883, -43804.13809377]), array([-44804.97
688527, -42954.57126833]), array([-44729.84994099, -42890.68218295]), array([-44488.308895
13, -42702.7740837 ]), array([-44389.59058554, -42494.29603662]), array([-44310.35348659,
-42160.68729875]), array([-44234.88698124, -42128.38683802]), array([-44185.84243711, -420
35.28262384]), array([-44083.86399549, -41717.30675386]), array([-44035.7094915 , -41559.3
5381642]), array([-43906.73653034, -41148.66124379]), array([-43821.15652446, -41021.85162
72 ]), array([-43826.76137703, -40968.9185046 ]), array([-43795.70348005, -40910.0540376
7]), array([-43761.9810834, -40870.374361 ]), array([-43538.38889343, -40675.57865389]), a
rray([-43589.00598952, -40665.7439608 ]), array([-43480.21465651, -40589.61905825]), array
([-43434.51191699, -40561.51455767]), array([-43234.12633936, -40410.06564297]), array([-4
3215.94129156, -40384.62986665]), array([-43145.02217917, -40244.44199559]), array([-4317
5.11787131, -40184.38571806]), array([-43119.20074269, -40141.44627523]), array([-43075.36
308517, -39980.53330878]), array([-42947.79653964, -39727.30939798]), array([-42875.118853
22, -39696.74196394]), array([-42905.1848125 , -39687.36484913]), array([-42877.83231897,
-39436.46180213]), array([-42752.66608263, -39469.46387337]), array([-42757.90317653, -393
39.298017  ]), array([-42742.94397591, -39329.11568192]), array([-42645.53897915, -39164.8
1654229]), array([-42644.25932414, -39119.63544558]), array([-42616.6795257, -39104.134335
8]), array([-42580.45650763, -38917.32635941]), array([-42500.52012091, -38832.21645191]),
array([-42587.1723829 , -38894.43155435]), array([-42588.51254703, -38896.73629336]), arra
y([-42561.78150933, -38880.11879827]), array([-42473.23638434, -38725.51532638]), array([-
42461.22542962, -38613.40157218]), array([-42445.34558416, -38625.78622808]), array([-4237
5.93529239, -38509.36592122]), array([-42396.63188523, -38531.24350164]), array([-42414.30
261998, -38396.94488691]), array([-42409.43052973, -38354.02594939]), array([-42323.362604
97, -38298.13957296]), array([-42346.84616035, -38257.75409108]), array([-42138.75535154,
-37866.76304876]), array([-42203.7461768 , -37839.69487576]), array([-42169.17711023, -377
77.2457621 ]), array([-42201.68592254, -37752.99599646]), array([-42159.53041548, -37686.6
2789196]), array([-42139.65952366, -37668.21475383]), array([-42061.13910958, -37603.78019
183]), array([-42106.9169234 , -37601.52346222]), array([-42112.06654187, -37603.3308768
1]), array([-42122.30727777, -37498.56777371]), array([-42077.79074355, -37468.11544687]),
array([-42075.69838306, -37468.66604505]), array([-42068.43683208, -37371.52439234]), arra
y([-42135.25927476, -37455.15912631]), array([-42028.36759182, -37412.61960613]), array([-
42054.782046  , -37374.35547882]), array([-42058.01586332, -37395.55606772]), array([-4208
2.94098644, -37377.73648142]), array([-42110.6506021 , -37384.93902742]), array([-42021.48
649577, -37302.45818181]), array([-42059.68058533, -37415.88620567]), array([-42058.245455
61, -37352.64401408]), array([-41998.04268198, -37225.6617736 ]), array([-42020.90947664,
-37224.23884487]), array([-42116.21906383, -37283.79820498]), array([-41995.49460333, -371
88.42869573]), array([-42016.26966727, -37123.65922652]), array([-42035.73821253, -37186.0
0878915]), array([-41977.63757004, -37169.51472989]), array([-42073.22262556, -37206.04620
438]), array([-41940.60802272, -37151.43653529]), array([-41973.00598974, -37139.3857852
]), array([-41999.75601906, -37159.08807367]), array([-42023.61543674, -37158.48424472]),
array([-42031.63563128, -37169.74790528]), array([-41984.07367162, -37194.20018001]), arra
y([-41947.86947119, -37125.26056188]), array([-41990.04863699, -37130.53201468]), array([-
41979.92631131, -37059.56701145]), array([-42031.63563128, -37162.74716885]), array([-4202
6.88444202, -37141.09224148]), array([-41998.02454148, -37129.79242714]), array([-41968.09
760816, -37122.16801769]), array([-42014.35154074, -37146.63822035]), array([-42030.200501
56, -37237.98189571]), array([-42023.83695468, -37160.09660233]), array([-41900.23495684,
```

-37076.41211912]), array([-41971.88385536, -37152.60540683]), array([-41963.31864325, -370
81.40045917]), array([-42042.35483186, -37181.12624712]), array([-41968.09760816, -37178.6
52101]), array([-41986.18924897, -37117.28045583]), array([-41941.80251478, -37103.66670
039]), array([-41912.64602379, -37007.84339319]), array([-41918.52308006, -36941.6848775
5]), array([-41953.77811355, -37050.23056076]), array([-41895.87727415, -36949.48251123]),
array([-41971.17775622, -37006.11548344]), array([-42016.64032083, -36980.7579706]), arra
y([-41894.91844718, -36893.28698168]), array([-41896.53080479, -36919.36975982]), array([-
41910.54389265, -36978.99730195]), array([-41872.57864905, -36903.85361802]), array([-4194
9.49148873, -36999.37374995]), array([-41898.23450672, -36946.72860381]), array([-41952.14
054685, -36928.34060415]), array([-41943.53399236, -36956.1281106]), array([-41888.480837
2 , -36997.27161881]), array([-41963.31864325, -37010.93789269]), array([-41967.09737881,
-36878.19776035]), array([-41901.01373848, -36893.40866822]), array([-41936.58864951, -370
36.79206656]), array([-42012.34332049, -36943.85674595]), array([-41953.68609109, -36981.2
9162817]), array([-41945.75781003, -36976.70131954]), array([-42018.37179841, -37018.35881
807]), array([-41963.74927986, -36887.23849666]), array([-41969.70996577, -36916.3219468
8]), array([-42006.64677503, -36930.91305309]), array([-41861.88505013, -36899.10107678]),
array([-41887.11977058, -36842.48197285]), array([-41927.72454228, -36876.26893856]), arra
y([-41862.55615479, -36867.71719228]), array([-41901.85682658, -36881.85927431]), array([-
41952.65809603, -36898.86669748]), array([-41972.36868382, -36915.65034325]), array([-4190
7.89483453, -36870.07643121]), array([-41954.45043279, -36842.67893707]), array([-41901.36
448649, -36771.31988849]), array([-41945.10129311, -36848.81569697]), array([-41868.952795
13, -36883.78325118]), array([-42000.70436537, -36911.49905577]), array([-41851.89682215,
-36917.14101434]), array([-41969.19950995, -36924.10056492]), array([-42030.83997987, -369
36.49854391]), array([-41877.45035203, -36853.1030374]), array([-41963.80841677, -36929.6
6724581]), array([-41886.99016612, -36819.11365226]), array([-41885.01763944, -36833.81072
66]), array([-41940.0707946 , -36927.56511467]), array([-41939.72631162, -36711.1262685
4]), array([-41910.55355257, -36752.99873332]), array([-41885.45971819, -36700.61517028]),
array([-41883.35758705, -36768.25562686]), array([-41795.53333134, -36583.42172739]), arra
y([-41823.2025082 , -36628.81338192]), array([-41847.69206089, -36641.90738984]), array([-
41885.01763944, -36704.17121308]), array([-41835.65149972, -36633.97050378]), array([-4188
1.65461431, -36559.40101532]), array([-41811.08788392, -36561.79585504]), array([-41850.18
141743, -36612.84437943]), array([-41867.3351197 , -36542.59837309]), array([-41807.828170
39, -36511.82871983]), array([-41810.59554382, -36409.2387792]), array([-41877.82100559,
-36400.82396458]), array([-41796.47251445, -36382.0260463]), array([-41801.62213293, -364
22.02917423]), array([-41895.67747106, -36455.34228935]), array([-41772.13595076, -36514.9
6449484]), array([-41799.63912175, -36429.28800247]), array([-41800.42548574, -36491.28198
843]), array([-41805.40086849, -36406.51303243]), array([-41832.33872439, -36451.1334080
8]), array([-41845.95781529, -36451.13340808]), array([-41768.68675 , -36478.57300047]),
array([-41826.78739249, -36414.65297006]), array([-41764.29655438, -36406.9352602]), arra
y([-41802.17871983, -36378.78676012]), array([-41802.17871983, -36389.81724529]), array([-
41821.76737848, -36449.57659183]), array([-41817.24757839, -36439.04049917]), array([-4185
4.57315694, -36389.90141155]), array([-41857.44178956, -36445.89033389]), array([-41739.66
869178, -36398.72183344]), array([-41862.55615479, -36463.61534393]), array([-41746.614034
63, -36395.01192063]), array([-41854.94381051, -36468.32816368]), array([-41774.60286995,
-36440.97254493]), array([-41818.87450696, -36394.29160717]), array([-41799.93571221, -364
04.76441733]), array([-41723.38332351, -36359.37276279]), array([-41843.08918267, -36473.0
1890054]), array([-41815.51610082, -36443.57660517]), array([-41834.35467991, -36465.10601
502]), array([-41798.44504113, -36488.11281456]), array([-41774.95638599, -36446.5161603
6]), array([-41792.14451322, -36454.62289715]), array([-41857.23187499, -36351.81339331]),
array([-41791.00735818, -36359.37276279]), array([-41801.93767073, -36410.17454639]), arra
y([-41734.65738321, -36369.03426342]), array([-41780.22769099, -36472.03807649]), array([-
41745.12336355, -36411.06494524]), array([-41799.93571221, -36359.72627883]), array([-4182
4.3204733 , -36414.42591795]), array([-41769.26731814, -36410.52806243]), array([-41777.05
851713, -36355.64499265]), array([-41773.83636847, -36463.84717344]), array([-41859.334006
13, -36412.32507638]), array([-41802.15952989, -36402.24452334]), array([-41821.08118711,
-36411.78433743]), array([-41834.0391421 , -36432.68900584]), array([-41810.67217349, -364
68.46698773]), array([-41835.15915962, -36394.8902341]), array([-41797.6096695 , -36381.3
9082036]), array([-41798.32592117, -36374.09602325]), array([-41789.39500056, -36346.74040
451]), array([-41810.73436788, -36382.0260463]), array([-41846.4311669 , -36459.7324849
7]), array([-41822.21834216, -36463.61534393]), array([-41835.71574652, -36434.43762094]),
array([-41835.15915962, -36449.87713307]), array([-41735.58354945, -36419.71436048]), arra
y([-41803.76932093, -36424.89780685]), array([-41842.59684257, -36398.84351997]), array([-
41771.36358376, -36434.43762094]), array([-41828.27806357, -36493.42460049]), array([-4186
5.25012608, -36444.87040045]), array([-41828.27806357, -36442.5057063]), array([-41837.26
129076, -36484.52001234]), array([-41824.49932801, -36437.3797427]), array([-41784.617886
61, -36406.51303243]), array([-41744.51190349, -36338.10129696]), array([-41909.99696567,

-36417.82692132]), array([-41830.38019471, -36483.49078944]), array([-41824.12867444, -363
37.60895687]), array([-41821.45184068, -36352.07133733]), array([-41796.47251445, -36398.7
2183344]), array([-41746.61403463, -36351.44273975]), array([-41796.47251445, -36359.37276
279]), array([-41857.81244313, -36460.67578874]), array([-41792.63613771, -36400.8239645
8]), array([-41832.23335882, -36385.28416382]), array([-41882.9155083, -36383.4929515]), a
rray([-41795.99731188, -36470.92468733]), array([-41841.04002632, -36410.4063759]), array
([-41801.68432731, -36458.92460707]), array([-41826.91699695, -36432.8278299]), array([-4
1764.41567434, -36393.56513634]), array([-41807.37339516, -36499.07318742]), array([-4178
1.03300232, -36432.68900584]), array([-41860.69507275, -36487.75929853]), array([-41835.15
915962, -36410.4063759]), array([-41821.45184068, -36406.51303243]), array([-41807.865735
26, -36474.64582911]), array([-41798.20423464, -36410.4063759]), array([-41739.73293858,
-36399.07534948]), array([-41843.08918267, -36486.50045695]), array([-41862.55615479, -364
24.89780685]), array([-41774.95638599, -36411.83530285]), array([-41747.10380816, -36336.2
3099533]), array([-41759.63670943, -36399.07534948]), array([-41817.73735192, -36422.02917
423]), array([-41824.49932801, -36450.02001893]), array([-41844.69897371, -36420.1162753
7]), array([-41862.55615479, -36378.17068107]), array([-41743.15083687, -36455.76206157]),
array([-41760.87841348, -36391.28415049]), array([-41853.37907633, -36358.42945902]), arra
y([-41768.13016309, -36440.48020483]), array([-41857.23187499, -36398.72183344]), array([-
41829.47009177, -36399.07534948]), array([-41873.16116064, -36522.75569383]), array([-4183
5.15915962, -36455.26565968]), array([-41754.17340411, -36440.97254493]), array([-41798.44
504113, -36521.29632522]), array([-41826.91699695, -36450.28052952]), array([-41849.794192
04, -36472.03807649]), array([-41794.24664436, -36328.92828029]), array([-41828.21381677,
-36492.31121133]), array([-41852.66282466, -36448.58031326]), array([-41795.97812194, -363
43.79036481]), array([-41881.16484078, -36408.91570481]), array([-41826.78739249, -36413.5
3295254]), array([-41794.74103688, -36444.87040045]), array([-41832.11167229, -36425.92702
975]), array([-41693.29235705, -36394.15278311]), array([-41886.99016612, -36495.6721840
5]), array([-41776.70500109, -36330.53807134]), array([-41731.1745225 , -36410.66431992]),
array([-41733.39834017, -36368.63086698]), array([-41769.26731814, -36335.86034176]), arra
y([-41835.64893315, -36450.07098435]), array([-41811.91131097, -36399.30240158]), array([-
41767.165187 , -36388.32657421]), array([-41773.34402837, -36396.1177732]), array([-4180
3.91019741, -36419.83604702]), array([-41886.74935963, -36419.71436048]), array([-41864.11
297104, -36464.16271124]), array([-41767.165187 , -36437.07920146]), array([-41801.622132
93, -36423.51984531]), array([-41741.46441616, -36421.37511695]), array([-41776.68786357,
-36401.52513114]), array([-41798.3233546 , -36429.51983198]), array([-41827.8431632 , -364
88.11281456]), array([-41826.42260444, -36449.87713307]), array([-41801.62213293, -36442.0
8593408]), array([-41819.34970954, -36394.6451232]), array([-41799.52000178, -36389.81724
529]), array([-41844.32903576, -36452.47365673]), array([-41827.36796063, -36398.7218334
4]), array([-41791.00735818, -36464.96719096]), array([-41759.49768392, -36326.94912525]),
array([-41806.01232855, -36422.44416905]), array([-41802.35338116, -36442.93384264]), arra
y([-41833.54936858, -36511.48163413]), array([-41862.38149346, -36469.48498167]), array([-
41807.37339516, -36476.72881335]), array([-41730.80458455, -36374.23484731]), array([-4181
9.84133402, -36372.82926374]), array([-41830.50188124, -36452.0263735]), array([-41794.37
038331, -36432.68900584]), array([-41819.83948306, -36493.80188241]), array([-41801.667189
79, -36474.64582911]), array([-41821.45184068, -36426.92330832]), array([-41744.88255705,
-36416.40636256]), array([-41837.26129076, -36435.20889983]), array([-41893.92680354, -363
98.72183344]), array([-41746.61403463, -36295.09071236]), array([-41796.47251445, -36440.4
0357516]), array([-41806.01232855, -36424.89780685]), array([-41796.88822488, -36410.17454
639]), array([-41811.08788392, -36442.22881993]), array([-41745.49401711, -36424.8978068
5]), array([-41794.24077885, -36454.62289715]), array([-41804.90647597, -36449.87713307]),
array([-41756.15384872, -36418.13583076]), array([-41856.12602242, -36482.77139724]), arra
y([-41811.9284485, -36395.0885503]), array([-41846.06051334, -36456.05597445]), array([-41
746.1983242 , -36413.53295254]), array([-41896.16981115, -36425.45640313]), array([-41876.
20864798, -36413.27500852]), array([-41795.01314584, -36432.68900584]), array([-41780.2276
9099, -36467.8242252]), array([-41794.61143241, -36503.34169651]), array([-41814.9214926
4, -36468.32816368]), array([-41736.01844982, -36360.98255384]), array([-41826.78739249, -
36393.03276559]), array([-41794.86015684, -36436.95751493]), array([-41795.99731188, -3654
6.21345706]), array([-41869.26258951, -36477.3640393]), array([-41794.56218217, -36486.01
068342]), array([-41768.89666457, -36420.06787652]), array([-41862.0108399 , -36514.964494
84]), array([-41781.27405142, -36440.97254493]), array([-41775.56784605, -36376.8254624
1]), array([-41940.19015717, -36476.72881335]), array([-41766.02803195, -36425.31757907]),
array([-41799.52000178, -36359.37276279]), array([-41777.18020366, -36431.56898832]), arra
y([-41824.81486581, -36469.57284031]), array([-41773.34402837, -36457.31481603]), array([-
41857.81244313, -36401.12450582]), array([-41794.24077885, -36444.53571575]), array([-4184
9.79419204, -36470.42828544]), array([-41800.42805231, -36420.06787652]), array([-41834.03
91421 , -36356.85286881]), array([-41860.94636374, -36382.0260463]), array([-41828.889523
63, -36389.18201935]), array([-41803.91019741, -36433.18134594]), array([-41792.14451322,

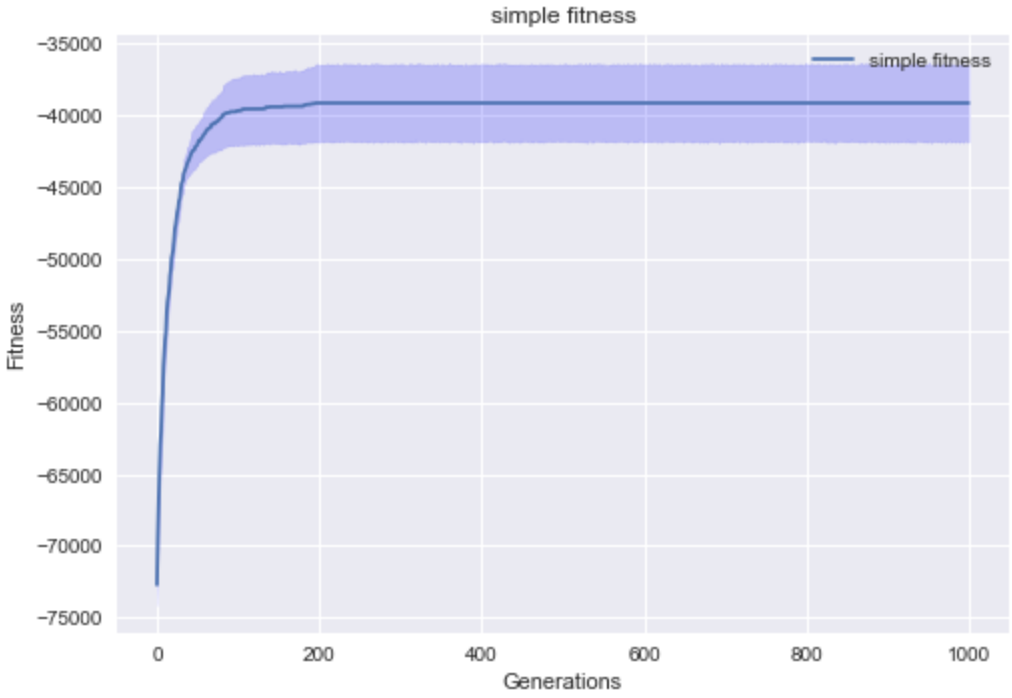
-36382.68010358]), array([-41856.55616812, -36493.80188241]), array([-41723.38332351, -363
57.3452089]), array([-41833.24882733, -36485.65917676]), array([-41800.4273367, -36395.08
85503]), array([-41740.92496678, -36467.20814616]), array([-41749.27275267, -36444.5357157
5]), array([-41793.44421706, -36366.30482426]), array([-41798.3233546 , -36527.82868263]),
array([-41740.34439864, -36417.96574538]), array([-41849.79419204, -36457.66833207]), arra
y([-41888.60252373, -36457.43650256]), array([-41779.7353509 , -36434.43762094]), array([-
41784.49620008, -36372.97600574]), array([-41856.67528808, -36494.38245056]), array([-4183
7.9395642 , -36413.27500852]), array([-41824.81486581, -36351.93507984]), array([-41849.79
419204, -36454.30735935]), array([-41806.01232855, -36312.49043718]), array([-41821.451840
68, -36434.71450731]), array([-41726.51724412, -36401.9439821]), array([-41847.19766838,
-36394.45332435]), array([-41801.68432731, -36357.08469831]), array([-41692.93884101, -363
53.68369494]), array([-41764.11769967, -36399.37589072]), array([-41796.47251445, -36399.0
7534948]), array([-41741.41935929, -36343.79036481]), array([-41855.12974385, -36466.71580
606]), array([-41835.47264501, -36434.71450731]), array([-41743.70742377, -36370.1981677
3]), array([-41865.25012608, -36498.41913014]), array([-41882.42111579, -36408.91570481]),
array([-41837.81787766, -36449.38479298]), array([-41781.95916857, -36415.14531015]), arra
y([-41800.05739874, -36448.27140382]), array([-41810.59554382, -36402.24452334]), array([-
41879.92313673, -36510.61935608]), array([-41820.0359009 , -36396.43376896]), array([-4178
5.19073828, -36461.86672883]), array([-41809.47552631, -36457.31481603]), array([-41792.63
890574, -36426.64642195]), array([-41831.93701096, -36422.02917423]), array([-41883.331218
73, -36403.99313844]), array([-41829.47009177, -36437.3797427]), array([-41709.36251918,
-36371.63832366]), array([-41860.69507275, -36467.20814616]), array([-41817.43937725, -364
52.7457657]), array([-41779.92714975, -36398.72183344]), array([-41801.68432731, -36460.0
3302621]), array([-41807.37339516, -36495.17984395]), array([-41790.53215561, -36471.71756
867]), array([-41744.51190349, -36383.89634794]), array([-41831.93701096, -36475.6270225
7]), array([-41802.03784335, -36446.51616036]), array([-41743.02123241, -36425.92702975]),
array([-41776.70500109, -36455.56620092]), array([-41885.01763944, -36399.14406121]), arra
y([-41729.00814455, -36371.87015316]), array([-41844.69897371, -36416.40636256]), array([-
41804.90647597, -36408.68387531]), array([-41873.16116064, -36464.73536145]), array([-4178
5.19845476, -36410.52806243]), array([-41786.8030959 , -36474.99934515]), array([-41772.13
595076, -36398.72183344]), array([-41862.0108399 , -36419.57553643]), array([-41815.635220
78, -36465.22770155]), array([-41763.92590081, -36452.39224966]), array([-41874.29831569,
-36492.31121133]), array([-41741.41935929, -36346.89082694]), array([-41857.23187499, -364
03.99313844]), array([-41827.15804605, -36434.43762094]), array([-41789.76750509, -36408.9
1570481]), array([-41834.0391421 , -36410.52806243]), array([-41881.16484078, -36490.98144
719]), array([-41830.02667867, -36414.30423142]), array([-41795.99731188, -36433.7182287
4]), array([-41773.83636847, -36355.73285129]), array([-41819.34970954, -36504.8086017]),
array([-41780.22769099, -36399.47874592]), array([-41710.97487679, -36389.18201935]), arra
y([-41806.01232855, -36437.07920146]), array([-41835.28084615, -36468.46698773]), array([-
41819.34970954, -36521.29632522]), array([-41757.76620634, -36346.38097991]), array([-4180
3.91019741, -36486.36419946]), array([-41830.02667867, -36433.71822874]), array([-41768.89
666457, -36456.94416246]), array([-41829.47009177, -36480.81009953]), array([-41766.028031
95, -36417.66520414]), array([-41770.99879572, -36461.86672883]), array([-41753.81066848,
-36449.87713307]), array([-41793.12867926, -36454.62289715]), array([-41832.11167229, -363
89.81724529]), array([-41713.26294127, -36398.84351997]), array([-41803.29668493, -36461.6
9378268]), array([-41821.76737848, -36442.22881993]), array([-41765.55282938, -36449.38479
298]), array([-41752.44192654, -36402.61517691]), array([-41827.15804605, -36362.4732249
2]), array([-41722.24616846, -36323.23921244]), array([-41777.18020366, -36452.7457657]),
array([-41780.71746452, -36417.96574538]), array([-41828.27806357, -36423.77778933]), arra
y([-41857.81244313, -36404.76441733]), array([-41802.38863441, -36401.67187313]), array([-
41821.76737848, -36478.70925796]), array([-41807.37339516, -36456.05597445]), array([-4180
2.03784335, -36476.72881335]), array([-41790.53215561, -36418.01872018]), array([-41774.04
628305, -36402.73686344]), array([-41804.90647597, -36454.07552984]), array([-41782.329822
13, -36432.61237617]), array([-41828.7678371 , -36449.87713307]), array([-41772.43392543,
-36378.78676012]), array([-41827.15804605, -36406.63471896]), array([-41798.81569469, -364
20.06787652]), array([-41840.98705153, -36422.02917423]), array([-41792.63613771, -36353.1
9135485]), array([-41817.73735192, -36328.43979634]), array([-41794.24664436, -36460.67578
874]), array([-41849.79419204, -36465.50941146]), array([-41846.31133133, -36456.3715122
5]), array([-41768.56506347, -36414.23797524]), array([-41833.24882733, -36432.68900584]),
array([-41832.60401239, -36321.77230725]), array([-41882.79176935, -36419.57553643]), arra
y([-41842.59684257, -36426.5622557]), array([-41892.4553224 , -36488.41335581]), array([-
41772.43392543, -36364.87589731]), array([-41726.78432688, -36313.34588231]), array([-4176
6.02803195, -36469.48498167]), array([-41799.52000178, -36447.1513863]), array([-41886.87
104616, -36399.47874592]), array([-41803.91019741, -36376.06854993]), array([-41877.642150
88, -36476.96064286]), array([-41766.39868552, -36390.93063445]), array([-41798.57464559,
-36478.21948443]), array([-41747.75118967, -36387.29735131]), array([-41791.00735818, -363

57.88209171]), array([-41772.31480547, -36402.24452334]), array([-41773.34402837, -36382.1
1113382]), array([-41780.71746452, -36394.45332435]), array([-41824.3204733 , -36387.69134
826]), array([-41824.3204733 , -36402.24452334]), array([-41740.92496678, -36414.2379752
4]), array([-41824.3204733 , -36487.62047447]), array([-41800.42548574, -36493.03060353]),
array([-41715.36507242, -36350.09089272]), array([-41855.35952094, -36482.55871463]), arra
y([-41817.24757839, -36495.41167346]), array([-41858.2876457 , -36468.32816368]), array([-
41787.66537395, -36510.49766955]), array([-41806.01232855, -36456.37151225]), array([-4179
2.14451322, -36431.56898832]), array([-41820.97663811, -36509.00699847]), array([-41828.27
806357, -36395.0885503]), array([-41794.37038331, -36452.62407916]), array([-41807.495081
7 , -36402.61517691]), array([-41749.39443921, -36364.34352656]), array([-41859.54392071,
-36502.70647056]), array([-41802.15952989, -36411.83530285]), array([-41837.81787766, -364
26.64642195]), array([-41862.22075447, -36410.17454639]), array([-41735.77534831, -36404.5
4050575]), array([-41776.6428067 , -36356.13347661]), array([-41768.68675   , -36412.27667
753]), array([-41826.42260444, -36403.27374624]), array([-41743.15083687, -36403.2737462
4]), array([-41774.60286995, -36455.56620092]), array([-41801.25147936, -36457.86218334]),
array([-41762.68419676, -36383.63583735]), array([-41833.546802 , -36399.37589072]), arra
y([-41826.99730706, -36419.57553643]), array([-41815.63522078, -36434.71450731]), array([-
41799.58219617, -36424.89780685]), array([-41890.89850615, -36405.48380953]), array([-4182
5.30463934, -36351.5815638]), array([-41825.45762834, -36358.11392122]), array([-41810.67
217349, -36460.41527815]), array([-41784.70611466, -36446.51616036]), array([-41745.001677
01, -36455.56620092]), array([-41791.00735818, -36402.24452334]), array([-41837.26129076,
-36440.33731898]), array([-41830.99165477, -36388.32657421]), array([-41835.15915962, -363
99.07534948]), array([-41743.70742377, -36340.55107863]), array([-41857.23187499, -36480.3
2161557]), array([-41792.14451322, -36460.03302621]), array([-41857.81244313, -36475.99767
614]), array([-41770.17742108, -36434.43762094]), array([-41695.04097215, -36307.8506657
3]), array([-41817.73735192, -36497.63348176]), array([-41796.47251445, -36480.32161557]),
array([-41748.85704225, -36358.11392122]), array([-41762.38622209, -36409.46307212]), arra
y([-41879.74428202, -36435.29675846]), array([-41747.28103606, -36388.95810777]), array([-
41852.66282466, -36454.07552984]), array([-41784.61788661, -36375.84463836]), array([-4183
3.54936858, -36424.89780685]), array([-41802.38863441, -36468.82050377]), array([-41776.68
786357, -36404.54050575]), array([-41829.78562957, -36368.91257689]), array([-41784.617886
61, -36374.23484731]), array([-41799.93571221, -36454.62289715]), array([-41832.60401239,
-36510.27375798]), array([-41826.11168562, -36386.66212536]), array([-41801.66718979, -363
56.85286881]), array([-41781.483966 , -36408.59601667]), array([-41766.39868552, -36394.1
5278311]), array([-41801.62213293, -36429.51983198]), array([-41798.32592117, -36370.37948
208]), array([-41754.54405768, -36405.74175355]), array([-41883.28616187, -36487.6204744
7]), array([-41828.88952363, -36406.51303243]), array([-41775.56784605, -36363.26610626]),
array([-41774.0110298 , -36455.56620092]), array([-41852.4710258 , -36492.07938182]), arra
y([-41857.23187499, -36463.35739991]), array([-41850.28396556, -36470.07677878]), array([-
41769.26731814, -36399.07534948]), array([-41748.71616577, -36412.27667753]), array([-4179
9.93571221, -36465.10601502]), array([-41748.48638868, -36359.86253632]), array([-41764.29
655438, -36394.94566445]), array([-41797.83358107, -36382.0260463]), array([-41769.880830
61, -36353.19135485]), array([-41807.37339516, -36435.20889983]), array([-41852.90387376,
-36421.72863299]), array([-41818.87450696, -36431.56898832]), array([-41784.61788661, -363
63.43619164]), array([-41825.26582949, -36437.61157221]), array([-41840.25160991, -36422.9
3650914]), array([-41837.26129076, -36552.56495038]), array([-41784.49620008, -36444.23517
45]), array([-41748.36264973, -36397.69261054]), array([-41807.62468616, -36371.6307870
7]), array([-41796.47251445, -36502.17364955]), array([-41765.20388929, -36398.84351997]),
array([-41800.05739874, -36420.06787652]), array([-41794.56218217, -36412.27667753]), arra
y([-41777.05851713, -36429.58854371]), array([-41802.15952989, -36361.77543518]), array([-
41819.34970954, -36374.18388189]), array([-41821.94346516, -36499.07318742]), array([-4188
5.01763944, -36418.01872018]), array([-41759.99002401, -36421.06620751]), array([-41823.06
419829, -36403.99313844]), array([-41807.86573526, -36393.88067414]), array([-41794.786093
74, -36394.15278311]), array([-41778.61533338, -36391.58469173]), array([-41798.44504113,
-36426.64642195]), array([-41775.56784605, -36353.68369494]), array([-41827.36796063, -364
60.04462459]), array([-41844.82066025, -36444.87040045]), array([-41854.98886737, -36483.5
5704562]), array([-41852.66282466, -36491.67598538]), array([-41824.81486581, -36412.27667
753]), array([-41852.88673623, -36462.99060249]), array([-41756.55041225, -36379.7830386
9]), array([-41817.73735192, -36442.22881993]), array([-41765.55282938, -36457.31481603]),
array([-41811.91131097, -36434.43762094]), array([-41793.12867926, -36440.48020483]), arra
y([-41857.44178956, -36490.56259623]), array([-41882.79176935, -36487.62047447]), array([-
41822.58899572, -36455.26565968]), array([-41750.58851982, -36366.79716436]), array([-4184
7.69206089, -36410.03572233]), array([-41849.79419204, -36393.56513634]), array([-41864.69
353918, -36457.36779082]), array([-41842.7185291 , -36502.70647056]), array([-41828.889523
63, -36348.76112857]), array([-41801.68432731, -36399.07534948]), array([-41919.04700623,
-36402.24452334]), array([-41757.4126903 , -36446.28433085]), array([-41835.71574652, -364

70.92468733]), array([-41776.68786357, -36368.27735094]), array([-41762.68419676, -36342.5
3152324]), array([-41712.52774227, -36353.19135485]), array([-41842.35579347, -36400.86656
18]), array([-41810.79129345, -36361.47489394]), array([-41796.71356355, -36404.7644173
3]), array([-41771.49318823, -36467.20814616]), array([-41833.42768204, -36494.05982643]),
array([-41784.49620008, -36408.91570481]), array([-41797.6096695 , -36430.72770813]), arra
y([-41798.57464559, -36372.92760689]), array([-41922.75892842, -36468.20647714]), array([-
41904.84734719, -36487.75929853]), array([-41798.44504113, -36491.82272737]), array([-4171
7.46720356, -36315.99538076]), array([-41830.99165477, -36452.88865155]), array([-41822.21
834216, -36419.57553643]), array([-41830.00954115, -36402.24452334]), array([-41832.252548
76, -36507.67723432]), array([-41777.18020366, -36432.68900584]), array([-41776.68786357,
-36343.65154075]), array([-41828.7678371 , -36403.99313844]), array([-41777.05851713, -364
14.65774746]), array([-41820.48686458, -36474.99934515]), array([-41766.58461886, -36431.5
6898832]), array([-41801.62213293, -36432.68900584]), array([-41831.93701096, -36513.21587
974]), array([-41775.56784605, -36395.01192063]), array([-41824.3204733 , -36468.8205037
7]), array([-41846.31133133, -36342.53152324]), array([-41846.07970328, -36465.10601502]),
array([-41776.70500109, -36382.0260463]), array([-41818.97905597, -36422.93650914]), arra
y([-41743.39188597, -36487.62047447]), array([-41774.13271633, -36374.18388189]), array([-
41773.83636847, -36394.45332435]), array([-41810.67217349, -36371.9918397]), array([-4189
8.14233783, -36451.36780416]), array([-41723.73683955, -36399.37589072]), array([-41771.90
889866, -36444.2351745]), array([-41805.38167855, -36432.3884646]), array([-41767.165187
  , -36336.49556771]), array([-41800.4273367 , -36460.53696469]), array([-41847.19766838,
-36390.17076133]), array([-41817.43937725, -36429.82037322]), array([-41764.29655438, -364
46.83169816]), array([-41887.53548101, -36440.61902889]), array([-41857.44178956, -36454.4
943808]), array([-41794.86015684, -36389.81724529]), array([-41796.34290999, -36415.72864
632]), array([-41802.15952989, -36449.57659183]), array([-41791.00735818, -36372.9760057
4]), array([-41888.60252373, -36412.32507638]), array([-41819.84133402, -36411.06494524]),
array([-41810.67217349, -36408.91570481]), array([-41822.21834216, -36479.82927548]), arra
y([-41796.47251445, -36434.43762094]), array([-41802.03784335, -36535.86916325]), array([-
41780.71746452, -36410.03572233]), array([-41837.9395642 , -36460.67985054]), array([-4184
7.92112237, -36406.63471896]), array([-41773.27978157, -36475.76584663]), array([-41779.73
53509 , -36456.45937089]), array([-41843.08918267, -36417.96574538]), array([-41819.349709
54, -36336.49556771]), array([-41837.81787766, -36387.29735131]), array([-41743.39188597,
-36387.69134826]), array([-41774.95638599, -36455.56620092]), array([-41767.65496052, -364
29.82037322]), array([-41777.47422756, -36440.17966359]), array([-41801.44327822, -36418.3
1926142]), array([-41796.10186089, -36326.60018516]), array([-41849.79419204, -36465.45953
106]), array([-41838.02779224, -36467.20814616]), array([-41821.45184068, -36469.6579278
2]), array([-41794.24664436, -36433.18134594]), array([-41803.29668493, -36419.57553643]),
array([-41826.60145915, -36404.76441733]), array([-41769.26731814, -36374.58836335]), arra
y([-41848.18183442, -36448.12851797]), array([-41754.71871901, -36359.86253632]), array([-
41741.41935929, -36356.85286881]), array([-41842.59684257, -36511.48163413]), array([-4182
1.45184068, -36387.22072164]), array([-41762.99973456, -36404.22496795]), array([-41819.34
970954, -36427.41770084]), array([-41793.8759908 , -36344.14388085]), array([-41762.980544
62, -36378.78676012]), array([-41762.68419676, -36437.61157221]), array([-41788.90522704,
-36382.0260463]), array([-41875.9106733 , -36485.71014218]), array([-41769.26731814, -363
99.37589072]), array([-41865.60364212, -36464.16271124]), array([-41795.99731188, -36342.5
3152324]), array([-41849.79419204, -36415.14531015]), array([-41852.66282466, -36465.10601
502]), array([-41826.42260444, -36403.99313844]), array([-41844.82066025, -36448.7637439
2]), array([-41761.24906705, -36440.97254493]), array([-41819.71964749, -36395.0885503]),
array([-41817.24757839, -36405.48380953]), array([-41885.92568997, -36460.18344865]), arra
y([-41838.52013234, -36399.37589072]), array([-41746.92752002, -36379.42952265]), array([-
41773.83636847, -36514.32926889]), array([-41819.34970954, -36470.92468733]), array([-4179
1.77385966, -36411.78433743]), array([-41812.28196454, -36449.38479298]), array([-41906.16
335695, -36522.75569383]), array([-41775.56784605, -36414.42591795]), array([-41878.136543
39, -36465.65338233]), array([-41784.49620008, -36381.88722225]), array([-41796.34290999,
-36446.83169816]), array([-41721.63470841, -36389.32285278]), array([-41810.1798334 , -364
49.87713307]), array([-41856.61836251, -36487.62047447]), array([-41784.70611466, -36452.6
2407916]), array([-41791.37801174, -36418.45551891]), array([-41741.83506972, -36316.93868
454]), array([-41786.72001776, -36452.2534256]), array([-41784.49620008, -36374.1838818
9]), array([-41756.15384872, -36410.52806243]), array([-41777.18020366, -36486.26862744]),
array([-41794.24664436, -36396.43376896]), array([-41818.42354329, -36437.04168118]), arra
y([-41823.06419829, -36339.09962795]), array([-41750.58851982, -36425.75694437]), array([-
41769.07551929, -36407.16708971]), array([-41768.89666457, -36396.97321834]), array([-4177
6.70500109, -36408.91570481]), array([-41810.1798334 , -36418.01872018]), array([-41777.05
851713, -36422.02917423]), array([-41768.56506347, -36424.89780685]), array([-41801.808066
26, -36351.5815638]), array([-41824.49932801, -36406.51303243]), array([-41757.76620634,
-36368.27735094]), array([-41809.47552631, -36479.82927548]), array([-41828.88952363, -364

80.688413]), array([-41833.66848854, -36432.68900584]), array([-41771.90889866, -36421.37511695]), array([-41773.27978157, -36470.78180148]), array([-41675.87054935, -36343.79036481]), array([-41795.01314584, -36444.74871392]), array([-41859.33400613, -36442.22881993]), array([-41801.68432731, -36414.30423142]), array([-41846.4311669 , -36498.28030608]), array([-41915.9995189 , -36495.55049752]), array([-41797.48006503, -36387.29735131]), array([-41722.24616846, -36379.66135215]), array([-41745.49401711, -36487.38864496]), array([-41857.23187499, -36400.8665618]), array([-41857.23187499, -36401.35633533]), array([-41877.64215088, -36398.72183344]), array([-41791.37801174, -36409.87400515]), array([-41819.34970954, -36396.43376896]), array([-41934.01131579, -36533.27263959]), array([-41819.34970954, -36430.93376237]), array([-41764.11769967, -36406.86177106]), array([-41802.15952989, -36421.06620751]), array([-41772.13595076, -36446.83169816]), array([-41771.36358376, -36367.74452993]), array([-41756.27553526, -36495.90401356]), array([-41783.93961318, -36507.67723432]), array([-41830.00954115, -36399.07534948]), array([-41801.80806626, -36386.66212536]), array([-41801.62213293, -36395.17749395]), array([-41800.00977531, -36450.02001893]), array([-41781.03300232, -36356.55232757]), array([-41851.52566961, -36414.30423142]), array([-41821.45184068, -36435.85166236]), array([-41815.63522078, -36342.53152324]), array([-41807.13234607, -36451.36780416]), array([-41840.6243159 , -36452.53991291]), array([-41803.91019741, -36463.92636968]), array([-41825.45762834, -36470.91805897]), array([-41686.90103452, -36353.19135485]), array([-41791.77385966, -36348.36142452]), array([-41822.39719687, -36398.72183344]), array([-41771.73423733, -36440.33731898]), array([-41827.33270738, -36537.30886891]), array([-41769.26731814, -36486.01068342]), array([-41801.66718979, -36468.05605471]), array([-41826.66570596, -36500.10241032]), array([-41734.3418454 , -36332.99170914]), array([-41883.10730716, -36446.9195568]), array([-41786.72001776, -36437.30625356]), array([-41854.7649558 , -36302.88191135]), array([-41788.90522704, -36372.74417623]), array([-41769.44617285, -36378.78676012]), array([-41865.60364212, -36482.4370281]), array([-41770.78888114, -36398.04612658]), array([-41886.57050492, -36331.03041144]), array([-41720.51469089, -36398.72183344]), array([-41832.83311691, -36440.61902889]), array([-41799.93571221, -36485.15523829]), array([-41752.32024001, -36279.80885562]), array([-41794.74103688, -36452.47365673]), array([-41781.83748204, -36444.87040045]), array([-41805.40086849, -36344.14388085]), array([-41872.80027597, -36458.92460707]), array([-41792.14451322, -36355.64499265]), array([-41761.24906705, -36432.68900584]), array([-41810.1798334 , -36527.95036917]), array([-41804.49076555, -36502.48255899]), array([-41798.44504113, -36456.05597445]), array([-41803.91019741, -36450.02001893]), array([-41849.79419204, -36502.97104294]), array([-41819.83948306, -36420.06787652]), array([-41890.70465488, -36455.26565968]), array([-41742.89954587, -36361.77543518]), array([-41811.9284485 , -36495.67218405]), array([-41820.48686458, -36437.3797427]), array([-41780.10600446, -36406.634718956]), array([-41851.52566961, -36515.60725737]), array([-41842.7185291 , -36465.10601502]), array([-41812.40365107, -36460.53696469]), array([-41804.90647597, -36459.81757249]), array([-41911.13412071, -36486.50045695]), array([-41793.8759908 , -36403.27374624]), array([-41777.05851713, -36447.47446069]), array([-41748.36264973, -36410.17454639]), array([-41878.48729141, -36455.26565968]), array([-41854.57315694, -36440.97254493]), array([-41777.18020366, -36425.80991917]), array([-41799.93571221, -36460.41527815]), array([-41769.26731814, -36406.51303243]), array([-41867.82745979, -36460.18344865]), array([-41848.18183442, -36472.6653845]), array([-41794.37038331, -36393.88067414]), array([-41802.15952989, -36394.8902341]), array([-41817.73735192, -36432.68900584]), array([-41879.0772806, -36452.7457657]), array([-41777.18020366, -36429.82037322]), array([-41766.02803195, -36302.950623068]), array([-41856.67528808, -36488.11281456]), array([-41736.26974082, -36441.96424755]), array([-41837.9395642 , -36433.48188718]), array([-41824.81486581, -36439.36018731]), array([-41796.47251445, -36483.72261894]), array([-41792.7580257 , -36439.04049917]), array([-41790.53215561, -36460.41527815]), array([-41865.23298856, -36411.78433743]), array([-41837.26129076, -36505.21199814]), array([-41842.84813357, -36437.30625356]), array([-41686.4853241 , -36408.61516357]), array([-41865.72532865, -36461.86672883]), array([-41825.26582949, -36413.62852456]), array([-41825.23057624, -36486.59125157]), array([-41875.08905606, -36356.55232757]), array([-41817.73735192, -36402.73686344]), array([-41803.91019741, -36410.22752119]), array([-41885.5550364 , -36465.10601502]), array([-41824.3204733, -36361.1213779]), array([-41858.22815356, -36405.48380953]), array([-41865.72532865, -36366.304824 26]), array([-41740.34439864, -36448.76374392]), array([-41726.41367331, -36422.93650914]), array([-41750.58851982, -36463.13348834]), array([-41800.4273367 , -36433.24760212]), array([-41824.49932801, -36440.97254493]), array([-41780.71746452, -36374.23484731]), array([-41802.80434483, -36406.86654847]), array([-41824.49932801, -36487.75929853]), array([-41746.61403463, -36439.36018731]), array([-41875.54001974, -36476.72881335]), array([-41796.88822488, -36411.92316149]), array([-41830.00954115, -36476.72881335]), array([-41855.48120747, -36357.08469831]), array([-41821.45184068, -36427.50555947]), array([-41879.55248317, -36450.07098435]), array([-41756.27553526, -36260.39485829]), array([-41789.2758806 , -36417.66520414]), array([-41768.68675 , -36457.66833207]), array([-41931.80896899, -36495.41167346]), array([-41798.44504113, -36468.32816368]), array([-41743.39188597, -36390.93063

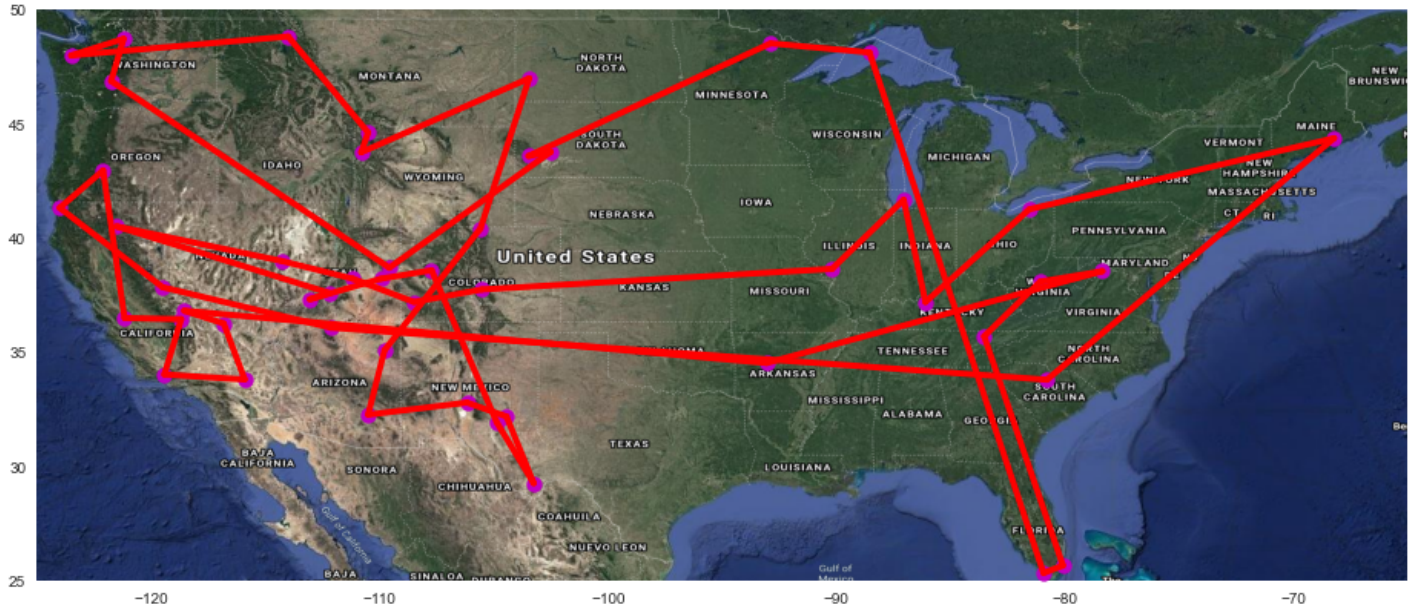
445]), array([-41843.95838219, -36534.20840678]), array([-41799.52000178, -36405.4838095
3]), array([-41777.18020366, -36440.48020483]), array([-41783.58609714, -36333.77735752]),
array([-41845.96058332, -36512.9153385]), array([-41804.1512465 , -36451.62574818]), arra
y([-41788.90522704, -36401.44004363]), array([-41832.60401239, -36368.27735094]), array([-
41812.69767497, -36474.76751564]), array([-41876.26347373, -36515.96077341]), array([-4180
1.68432731, -36440.97254493]), array([-41760.87841348, -36353.68369494]), array([-41796.47
251445, -36455.34228935]), array([-41865.72532865, -36371.63078707]), array([-41881.179411
74, -36471.95391023]), array([-41854.57315694, -36437.3797427]), array([-41804.49076555,
-36424.89780685]), array([-41826.78739249, -36450.02001893]), array([-41711.51227375, -363
25.20051015]), array([-41739.31446013, -36465.22770155]), array([-41877.64215088, -36376.0
6854993]), array([-41781.03300232, -36376.42206597]), array([-41857.81244313, -36497.42079
915]), array([-41792.63613771, -36382.37956234]), array([-41802.15952989, -36474.9993451
5]), array([-41815.06236911, -36465.45953106]), array([-41853.37907633, -36455.56620092]),
array([-41857.81244313, -36415.14531015]), array([-41773.83636847, -36455.26565968]), arra
y([-41789.09702589, -36389.18201935]), array([-41831.93701096, -36480.32161557]), array([-
41830.02667867, -36444.87040045]), array([-41886.37870606, -36493.80188241]), array([-4173
0.32938197, -36397.11405177]), array([-41770.78888114, -36348.03722432]), array([-41846.07
970328, -36459.73248497]), array([-41844.82066025, -36383.85974892]), array([-41873.041325
07, -36387.69134826]), array([-41777.05851713, -36487.88098506]), array([-41729.07239136,
-36417.02997819]), array([-41744.15838745, -36430.23536804]), array([-41803.91019741, -365
02.33581699]), array([-41878.31077912, -36484.52001234]), array([-41812.8193615 , -36411.7
8433743])]



Let's also visualize the optimization over time!

In [96]:

```
# interactive viz
# plot best solution found thus far as interactive plot
best = 0
for i in range(len(experiment_results['first'])):
    if max(experiment_results['first'][best]) < max(experiment_results['first'][i]):
        best = i
show_solution_evolution(np.array(solutions['first'][best]))
```




```
Generation: 999
Fitness: -32786.249615448
```

Q9: Analysis at Scale

Do you notice anything different viewing these results at scale? Does it work as well? What relationships or patterns do you see between the smaller and larger TSP problems that might help to inform how well of a proxy one is for the other?

It looks like better solutions stopped being found after about 200 generations. It also looked like it had a much more difficult time finding the optimal subnetworks within different portions of the country and even somewhat connecting the subnetworks. It looks like finding the best subnetworks and connections along points that don't have many nearby neighbors are both good strategies with both the smaller and larger TSP problems. This could help to inform crossover by trying to keep these subnetworks intact while better routes are found within them.

Q10: Future Work

If any of these results were less than perfect solutions (including this last one at scale), what might still be standing in the way? You might think about our discussions on search landscapes, local optima, and selection pressure, or consider additonal modificaitons to the variation pperators, selction criteria, or viability constraints, among many other ideas of potential future directions.

I think that selection pressure here is quite high leading to lower exploration of the space after not so many generations. I think that with this being the case, local optima are favored far greater than they might should be. The epistasis between points is obvious and is something that could use some attention in the mutation and crossover operations for better performance. This might allow for subnetworks to form and the connections between them to be optimized as well.

Congratulations, you made it to the end!

Nice work -- and hopefully you're starting to get the hang of these!

Please save this file as a .ipynb, and also download it as a .pdf, uploading **both** to blackboard to complete this assignment.

For your submission, please make sure that you have renamed this file (and that the resulting pdf follows suit) to replce `[netid]` with your UVM netid. This will greatly simplify our grading pipeline, and make sure that you receive credit for your work.

Academic Integrity Attribution

During this assignment I collaborated with:

Alican for comparison of results.

In [ ]: