CaseStudy2

June 24, 2018

1 Case Study #2: Santa Claus Route Optimization Problem

2 Executive Summary:

In this MVP, I'm trying to find the optimal route for Santa such that the weight-miles (weight * miles) is as low as possible in his night of sleigh flights. I chose to use the optimization method I mentioned on our group phone call: the Simulated Annealing (SA) algorithm (https://en.wikipedia.org/wiki/Simulated_annealing). It's great for problems like this with a massive search space (Traveling Salesperson and combinatorial optimation-type problems). It won't find the global, best solution, but it'll find a pretty good solution (and in a fraction of the time too). Optimizing this route is difficult. There are 100,000 destinations around the world, and if you assume that the sleigh can hold all gifts without the need to return periodically to the North Pole, there are (100,000 - 1)! possible routes to consider. In this problem, Santa has to periodically return to the North Pole, which means there are even more routes to consider. At a high level, when implimenting the SA algorithm, you take an initial route, evaluate it, then randomly perturb the route, and see if it performs any better. In this exercise, I assumed that for every trip out of the North Pole, Santa is carrying as close to 1000 lbs on the sleigh as possible, and for each perturbation in each iteration of the algorithm, I randomly swapped a small amount of gifts around. This way, his night's worth of routes change a bit with each iteration. Because this is just intended as a POC and given the limited time, I only ran the algorithm shown below for 100 iterations. This POC shows that the route improves by 0.55% even in a small number of iterations. As this web app demonstrates, simulated annealing can require 100,000s of iterations before dramatic improvements are made: http://toddwschneider.com/posts/traveling-salesman-with-simulated-annealing-r-and-shiny/ Given additional time to work on the next version of this MVP, I would have tried:

```
* Instead of perturbing the list of gifts and locations between each trip of 1000 pounds, I cou
* If I had access to a computer cluster, it would be effortless to speed up the computation her
```

2.0.1 Library Imports and defining functions

```
In [1]: import time
        import pandas as pd
        import numpy as np
        from multiprocessing import Pool, cpu_count
        from geopy.distance import distance # Function to calculate the distance between two c
        from matplotlib import pyplot as plt
```

```
import matplotlib.gridspec as gs
%matplotlib inline

pd.set_option('display.max_rows', 500)
pd.options.mode.chained_assignment = None

# I have 4 registered cores to work with on my laptop. I'll use that to speed the calc
core_count = cpu_count()
print core_count

# Set the random seed
np.random.seed(512)
```

```
4
```

```
In [2]: # Function to swap a small number of rows (this is the small perturbation in the overa
        def shuffle_a_few_rows(df, n_swaps):
            """
            df: the dataframe you're working with
            n_swaps: the number swaps you want to do
            """
            for i in range(0, n_swaps):
                loc1 = np.random.randint(0, len(df))
                loc2 = np.random.randint(0, len(df))
                b, c = df.iloc[loc1].copy(), df.iloc[loc2].copy()
                df.iloc[loc1],df.iloc[loc2] = c,b
            return df



        # Function that to evaluate the weight-miles (weight * miles) associated with each nor
        # *This can be parallelized easily*
        def evaluate_individual_trip_route(trip_number):
            """
            trip_number: the integer trip number from the route dataframe
            """
            sub_df = new_df.loc[new_df['trip_number'] == trip_number]
            sub_df['shift_Latitude'] = sub_df['Latitude'].shift(-1)
            sub_df['shift_Longitude'] = sub_df['Longitude'].shift(-1)
            #sub_df.loc[sub_df.index[-1], 'shift_Latitude'] = 90
            #sub_df.loc[sub_df.index[-1], 'shift_Longitude'] = 0
            weight_miles = 0
            current_sleigh_weight = sub_df['Weight'].sum() + 10
            weight_miles += (distance(North_Pole, (sub_df['Latitude'].values[0], sub_df['Longi
            for row in sub_df.itertuples():
                if np.isnan(row[7]):
                    weight_miles += (distance((row[2], row[3]), North_Pole).miles) * (current_
```

```
            else:
                current_sleigh_weight -= row[4]
                weight_miles += (distance((row[2], row[3]), (row[7], row[8])).miles) * (cu
        return weight_miles
```

2.0.2 Create an initial random route, and evaluate it

```
In [3]: # Create a list to store the results (total weight-miles the whole night) from each it
        final_results = []

        # Setting the north pole latitude and longitude
        North_Pole = (90, 0)

        ### Read in the gifts dataset
        gifts = pd.read_csv("~/Desktop/gifts.csv")

        # Do an initial shuffling of the dataset
        new_df = gifts.sample(frac = 1, random_state = 512).reset_index()[['GiftId', 'Latitude

        # Calculate cumulative sum for weights
        new_df['cumul_weight'] = new_df['Weight'].cumsum()

        # Break up the gifts into trips (trips can't exceed 1000 lbs)
        new_df['trip_number'] = new_df['cumul_weight'] / 1000
        new_df['trip_number'] = new_df['trip_number'].astype(int)

        # Evaluate the overall route, using parallel processing
        pool = Pool(processes = core_count)
        results = pool.map(evaluate_individual_trip_route, range(0,1410))
        pool.close()

        # Get the toal weight-miles for that initial, random route. It's terrible, not surpris
        old_result = sum(results)
        final_results.append(old_result)

        # Set up the old_df variable for the SA algorithm
        old_df = new_df.copy()
```

2.0.3 Setup the SA hyperparameters (temperature, number of iterations to try, etc.)

```
In [4]: temperature = 1.0 # Initial temperature
        count = 0 # Start the count at 0
        alpha = 0.9995 # The rate at which the temperate will drop over each iteration
        iteration_num = 100 # Total number of iterations to stop at
```

2.0.4 Run the SA algorithm. A while loop begins here...

In [5]:
```
while count < iteration_num:
    temperature *= alpha
    count += 1

    ###### Swap 100 (0.1% of all routes) individual routes randomly (https://stackover
    new_df = shuffle_a_few_rows(old_df, 10)

    ###### Evaluate the new_dataset

    # Calculate cumulative sum for weights
    new_df['cumul_weight'] = new_df['Weight'].cumsum()

    # Break up the gifts into trips (trips can't exceed 1000 lbs)
    new_df['trip_number'] = new_df['cumul_weight'] / 1000
    new_df['trip_number'] = new_df['trip_number'].astype(int)

    pool = Pool(processes = core_count)
    results = pool.map(evaluate_individual_trip_route, range(0,1410))
    pool.close()

    # Evaluate the new route list
    new_result = sum(results)


    #### Compare new result with old result, decide whether to accept the challenger r

    if new_result < old_result:
        old_df = new_df.copy()
        old_result = new_result
        p = None
        final_results.append(old_result)

        print "Iteration: {0} \t Temperature: {1} \t P: {2} \t Score: {3}".format(coun

    else:
        uniform_rand_num = np.random.uniform(0,1)
        p = 1e8 * np.exp(-((np.log(new_result - old_result))/temperature))

        if p > uniform_rand_num:
            old_df = new_df.copy()
            old_result = new_result
            final_results.append(old_result)
            print "Iteration: {0} \t Temperature: {1} \t P: {2} \t Score: {3}".format(c
        else:
            print "Iteration: {0} \t Temperature: {1} \t P: {2} \t Score: {3}".format(c
            final_results.append(old_result)
```

```
                    continue

Iteration: 1          Temperature: 0.9995          P: None          Score: 2.89249626279e+11
Iteration: 2          Temperature: 0.99900025          P: None          Score: 2.89239171925e+1
Iteration: 3          Temperature: 0.998500749875          P: None          Score: 2.889027924
Iteration: 4          Temperature: 0.9980014995          P: 0.517253958905          Score: 2.88
Iteration: 5          Temperature: 0.99750249875          P: 0.384969918726          Score: 2.8
Iteration: 6          Temperature: 0.997003747501          P: 0.192422144307          Score: 2
Iteration: 7          Temperature: 0.996505245627          P: 2.63093349851          Score: 2.8
Iteration: 8          Temperature: 0.996006993004          P: None          Score: 2.890725230
Iteration: 9          Temperature: 0.995508989508          P: 0.24802243159          Score: 2.8
Iteration: 10          Temperature: 0.995011235013          P: 0.215666371708          Score:
Iteration: 11          Temperature: 0.994513729396          P: 0.134463953551          Score:
Iteration: 12          Temperature: 0.994016472531          P: 0.148656708144          Score:
Iteration: 13          Temperature: 0.993519464295          P: None          Score: 2.88819412
Iteration: 14          Temperature: 0.993022704563          P: 0.128300611373          Score:
Iteration: 15          Temperature: 0.99252619321          P: 0.102557436692          Score: 2
Iteration: 16          Temperature: 0.992029930114          P: 0.0877554325677          Score:
Iteration: 17          Temperature: 0.991533915149          P: None          Score: 2.89512904
Iteration: 18          Temperature: 0.991038148191          P: 0.622032776672          Score:
Iteration: 19          Temperature: 0.990542629117          P: None          Score: 2.89141548
Iteration: 20          Temperature: 0.990047357802          P: 1.25346848712          Score: 2
Iteration: 21          Temperature: 0.989552334123          P: 4.76983419232          Score: 2
Iteration: 22          Temperature: 0.989057557956          P: 0.174898399152          Score:
Iteration: 23          Temperature: 0.988563029177          P: None          Score: 2.89179873
Iteration: 24          Temperature: 0.988068747663          P: 0.216753122695          Score:
Iteration: 25          Temperature: 0.987574713289          P: 0.189213045067          Score:
Iteration: 26          Temperature: 0.987080925932          P: 0.180407978065          Score:
Iteration: 27          Temperature: 0.986587385469          P: 1.28586520044          Score: 2
Iteration: 28          Temperature: 0.986094091777          P: 0.341627962218          Score:
Iteration: 29          Temperature: 0.985601044731          P: 0.375987666516          Score:
Iteration: 30          Temperature: 0.985108244208          P: None          Score: 2.88630166
Iteration: 31          Temperature: 0.984615690086          P: 0.0954032625088          Score:
Iteration: 32          Temperature: 0.984123382241          P: 0.0525683032876          Score:
Iteration: 33          Temperature: 0.98363132055          P: 0.0628071676595          Score:
Iteration: 34          Temperature: 0.98313950489          P: 0.0850294079813          Score:
Iteration: 35          Temperature: 0.982647935137          P: 0.103948877329          Score:
Iteration: 36          Temperature: 0.98215661117          P: 0.112592270821          Score: 2
Iteration: 37          Temperature: 0.981665532864          P: 0.140502377735          Score:
Iteration: 38          Temperature: 0.981174700098          P: 0.619562566309          Score:
Iteration: 39          Temperature: 0.980684112748          P: 0.121706922638          Score:
Iteration: 40          Temperature: 0.980193770691          P: 0.524456864111          Score:
Iteration: 41          Temperature: 0.979703673806          P: None          Score: 2.88976590
Iteration: 42          Temperature: 0.979213821969          P: None          Score: 2.88533611
Iteration: 43          Temperature: 0.978724215058          P: 0.980364885017          Score:
Iteration: 44          Temperature: 0.978234852951          P: 7.61482776364          Score: 2
Iteration: 45          Temperature: 0.977745735524          P: 0.686543039713          Score:
Iteration: 46          Temperature: 0.977256862656          P: None          Score: 2.88465673
```

```
Iteration: 47          Temperature: 0.976768234225          P: 0.515636180028          Score:
Iteration: 48          Temperature: 0.976279850108          P: None          Score: 2.88450590
Iteration: 49          Temperature: 0.975791710183          P: 0.128453139363          Score:
Iteration: 50          Temperature: 0.975303814328          P: None          Score: 2.88807333
Iteration: 51          Temperature: 0.974816162421          P: 0.140618396511          Score:
Iteration: 52          Temperature: 0.974328754339          P: 0.114720182613          Score:
Iteration: 53          Temperature: 0.973841589962          P: 0.309780042022          Score:
Iteration: 54          Temperature: 0.973354669167          P: 6.53437233328          Score: 2
Iteration: 55          Temperature: 0.972867991833          P: 0.619509370388          Score:
Iteration: 56          Temperature: 0.972381557837          P: 0.230758867983          Score:
Iteration: 57          Temperature: 0.971895367058          P: 0.810366761929          Score:
Iteration: 58          Temperature: 0.971409419374          P: 0.0939262086143          Score:
Iteration: 59          Temperature: 0.970923714665          P: 0.0971091677924          Score:
Iteration: 60          Temperature: 0.970438252807          P: None          Score: 2.89446324
Iteration: 61          Temperature: 0.969953033681          P: None          Score: 2.89052628
Iteration: 62          Temperature: 0.969468057164          P: 1.03096478602          Score: 2
Iteration: 63          Temperature: 0.968983323135          P: 0.189131232554          Score:
Iteration: 64          Temperature: 0.968498831474          P: 0.43403943251          Score: 2
Iteration: 65          Temperature: 0.968014582058          P: None          Score: 2.87977284
Iteration: 66          Temperature: 0.967530574767          P: 0.0748004233869          Score:
Iteration: 67          Temperature: 0.96704680948          P: 0.0357864966124          Score:
Iteration: 68          Temperature: 0.966563286075          P: 0.11533752878          Score: 2
Iteration: 69          Temperature: 0.966080004432          P: 0.107562946482          Score:
Iteration: 70          Temperature: 0.96559696443          P: 0.057865039715          Score: 2
Iteration: 71          Temperature: 0.965114165948          P: 0.0441425712607          Score:
Iteration: 72          Temperature: 0.964631608865          P: 0.0570606339528          Score:
Iteration: 73          Temperature: 0.96414929306          P: 0.0432372909521          Score:
Iteration: 74          Temperature: 0.963667218414          P: 0.047242195215          Score:
Iteration: 75          Temperature: 0.963185384804          P: 0.0446192663043          Score:
Iteration: 76          Temperature: 0.962703792112          P: 0.046660821786          Score:
Iteration: 77          Temperature: 0.962222440216          P: 0.0562486770757          Score:
Iteration: 78          Temperature: 0.961741328996          P: 0.0317037080542          Score:
Iteration: 79          Temperature: 0.961260458331          P: 0.0473427510934          Score:
Iteration: 80          Temperature: 0.960779828102          P: 0.0228080230809          Score:
Iteration: 81          Temperature: 0.960299438188          P: 0.0296348937234          Score:
Iteration: 82          Temperature: 0.959819288469          P: 0.0259625604013          Score:
Iteration: 83          Temperature: 0.959339378825          P: 0.0213239390589          Score:
Iteration: 84          Temperature: 0.958859709135          P: 0.025943401562          Score:
Iteration: 85          Temperature: 0.958380279281          P: 0.0259542517775          Score:
Iteration: 86          Temperature: 0.957901089141          P: 0.0290801421499          Score:
Iteration: 87          Temperature: 0.957422138597          P: 0.0401776785774          Score:
Iteration: 88          Temperature: 0.956943427527          P: 0.0295293316544          Score:
Iteration: 89          Temperature: 0.956464955814          P: 0.0253106231089          Score:
Iteration: 90          Temperature: 0.955986723336          P: 0.0247420176356          Score:
Iteration: 91          Temperature: 0.955508729974          P: 0.0243557679352          Score:
Iteration: 92          Temperature: 0.955030975609          P: 0.0324639036763          Score:
Iteration: 93          Temperature: 0.954553460121          P: 0.0251154936626          Score:
Iteration: 94          Temperature: 0.954076183391          P: 0.0216268468474          Score:
```

```
Iteration: 95          Temperature: 0.953599145299          P: 0.0190749211822          Score:
Iteration: 96          Temperature: 0.953122345727          P: 0.0185969516035          Score:
Iteration: 97          Temperature: 0.952645784554          P: 0.0238962960655          Score:
Iteration: 98          Temperature: 0.952169461662          P: 0.0216572392355          Score:
Iteration: 99          Temperature: 0.951693376931          P: 0.0219966463459          Score:
Iteration: 100          Temperature: 0.951217530242          P: 0.0284560235701          Score
```

In [6]:
```
fig=plt.figure(figsize=(10, 6), dpi= 80, facecolor='w', edgecolor='k')

g1 = gs.GridSpec(1,1)

ax1 = plt.subplot(g1[0,0])
plt.plot((final_results))
plt.xlabel('Iteration Number', {'fontsize': 12})
plt.ylabel('Total Score For The Night  (100s of billions of weight-miles)', {'fontsize
plt.title(' ')
```

Out[6]: Text(0.5,1,' ')